Proceedings of the Seminar

Deep Learning

University of Colorado, Colorado Springs

August 6, 2021

Editors: Jugal K. Kalita, Oluwadare Oluwatosin and Adham Atyabi

Funded by

National Science Foundation

Preface

It is with great pleasure that we present to you the papers describing the research performed by the NSF-funded Research Experience for Undergraduates (REU) students, who spent 10 weeks during the summer of 2021 at the University of Colorado, Colorado Springs. Within a very short period of time, the students were able to choose cutting-edge projects involving machine learning in the areas of natural language processing, bioinformatics and computational medicine; write proposals; design interesting algorithms and approaches; develop code; and write scholarly papers describing their findings. We hope that the students will continue working on these projects and submit papers to conferences and journals within the next few months. We also hope that it is the beginning of a fruitful career in research and innovation for all our participants.

We thank the National Science Foundation for funding our REU site. We also thank the University of Colorado, Colorado Springs, for providing an intellectually stimulating environment for research. In particular, we thank Dr. Terrance Boult, who was a helpful and stimulating mentor for the REU students. We also thank Alessandra Langfels for working out all the financial and administrative details. We thank Dr. Donald Rabern, the Dean of the College of Engineering and Applied Science, and Dr. Thottam Kalkur, the Chair of the Electrical and Computer Engineering Department for their support. We also thank our graduate and undergraduate students, in particular, Victor Akpokiro, Ali Al Shami, Basemah Alshemali, Ahmed Bensaoud, Thomas Conley, Keith Paulson, Joseph Worsham and Zanyar Zohourianshahzadi for helping the students with ideas as well as systems and programming issues. Our gratitude to Ginger Boult for being the "REU Mom" and having the welfare of the REU interns at her heart all through the summer. Special thanks to Dr. Walter Scheirer of Notre Dame, Dr. David Inoyue of Purdue (UCCS REU 2010), David Foley (UCCS REU 2016) of Penn State, Sridhama Prakhya (UCCS REU 2017, heading to UMass in Fall 2021 for PhD) and Diptodip Deb (UCCS REU 2017, heading to Johns Hopkins in Fall 2021 for PhD) for taking part in panel discussions on how to apply to graduate school.

Sincerely,

Jugal Kalita
jkalita@uccs.edu
Professor

Oluwadare Oluwatosin
ooluwada@uccs.edu
Assistant Professor

Adham Atyabi
aatyabi@uccs.edu
Assistant Professor

August 6, 2021

Table of Contents

NSF REU Proposal Presentation Meeting

Department of Computer Science
University of Colorado, Colorado Springs
Engineering Building, Room 101
June 10, 2021: Friday

1:45-1:50 PM: Welcome Remarks by Dr. Thottam Kalkur, Chair, Electrical and Computer Engineering, College of Engineering and Applied Science.

1:55-2:55 PM
Session Chair: Dr. Adham Atyabi, Department of Computer Science, University of Colorado, Colorado Springs.

Devin Haggitt, University of Colorado, Colorado Springs, CO: Navigating Tello Drones through Cluttered Environments using a Differential Evolution Algorithm

Daniel Theng, California State University, Fresno, CA: EEG Data Restructuring and Combination for Image Representation for Deep Learning Motor Imagery Models to Subject-transfer

Theodore Zaremba, University of Illinois, Chicago, IL: Classification of EEG Motor Imagery Tasks for Subject Transfer using Image-based 2D Convolutional Neural Networks

3:05-4:05 PM
Session Chair: Zanyar Zohoruianshahzadi, Department of Computer Science, University of Colorado, Colorado Springs, CO

Wesley Robbins, Montana State University, Bozeman, MT: Richer Text Generation from Multimodal Vision & Language Models using Special Tokens

Abigail Swenor, University of Colorado, Colorado Springs, CO: Application of Synonym Substitution Defense Against Data Poisoning and Backdoor Attacks

Séraphin Bassas, McGill University, Montreal, QC, Canada: Leveraging Priming and Centroid Based Learning to Convert Neural Classifiers into Class Incremental Learning Networks

4:15-4:55 PM
Session Chair: Ahmed Bensaoud, University of Colorado, Colorado Springs, CO

Trevor Martin, Oberlin College, Oberlin, OH: EnsembleSplice: Ensemble Learning for Splice Site Prediction

Parker Hicks, Concordia University, Irvine, CA: HiCARN: Resolution Enhancement of Hi-C Data Using a Cascading Residual Network

Our Session Chairs

Dr. *Adham Atyabi* is an assistant professor in the Department of Computer Science, University of Colorado, Colorado Springs. His expertise are in cognitive and computational neuroscience, brain-computer interface, image and signal processing, and deep learning. At UCCS for 3 years. He obtained his PhD from Flinders University, Australia; and held post-doc positions at Yale and University of Washington. Dr. Atyabi has published 60 papers in various conferences and journals.

Zanyar Zohoruianshahzadi is a doctoral student at the University of Colorado, Colorado Springs. Zanyar has recently passed his PhD dissertation proposal examination. The title of his dissertation is Improving Neural Attention for Image Captioning. Zanyar has published two papers recently, one at the IEEE International Conference on Humanized Computing and Communication with Artificial Intelligence (HCCAI, 2020), and the other in the International Journal of Semantic Computing (2021).

Ahmed Bensaoud is a doctoral student at the University of Colorado, Colorado Springs. His research focuses on Deep Learning for Malware Detection. He has published a paper in the *International Journal of Computer Security*.

NSF REU Midsummer Presentation Meeting

Department of Computer Science
University of Colorado, Colorado Springs
Engineering Building, Room 101
July 9, 2021: Friday

1:45-1:50 PM: Welcome Remarks by Dr. Terrance Boult, El Pomar Chair of Security and Innovation, University of Colorado, Colorado Springs

1:55-2:55 PM
Session Chair: Brandon Collins, Department of Computer Science, University of Colorado, Colorado Springs.

Wesley Robbins, Montana State University, Bozeman, MT: Generalizing OCR Tokens to Special Token for Vision-Language Models

Abigail Swenor, University of Colorado, Colorado Springs, CO: Application of Synonym Substitution Defense Against TextAttack Classification Models

Séraphin Bassas, McGill University, Montreal, QC, Canada: Leveraging Priming and Centroid Based Concept Learning to Convert Neural Classifiers into Open Set Classifiers

3:05-3:45 PM
Session Chair: Dr. Oluwatosin Oluwadare, Department of Computer Science, University of Colorado, Colorado Springs, CO

Trevor Martin, Oberlin College, Oberlin, OH: EnsembleSplice: Ensemble Learning for Splice Site Prediction

Parker Hicks, Concordia University, Irvine, CA: HiCARN: Resolution Enhancement of Hi-C Data Using a Cascading Residual Network

3:55-4:55 PM
Session Chair: Justin Leo, Department of Computer Science, University of Colorado, Colorado Springs, CO

Devin Haggitt, University of Colorado, Colorado Springs, CO: Generating Depth Maps from Image Pairs collected from Tello Drones

Daniel Theng, California State University, Fresno, CA: EEG Data Restructuring and Combination for Image Representation for Deep Learning Motor Imagery Models to Subject-transfer

Theodore Zaremba, University of Illinois, Chicago, IL: Classification of EEG Motor Imagery Tasks for Subject Transfer using Image-based 2D Convolutional Neural Networks

Our Session Chairs

Brandon Collins is a Ph.D. student in the Computer Science Department at the University of Colorado, Colorado Springs. His research interests lie in algorithmic game theory, and in particular in developing a game theoretic model for the spread of conventions in society called a coordination game. He has published a paper in the American Control Conference (ACC) 2020 on the impact of a malicious and a self-interested adversary on a coordination game as a function of the strength of the adversary.

Dr. Oluwatosin Oluwadare is an Assistant Professor of Computer Science and Innovation at the University of Colorado, Colorado Springs. Dr. Oluwadare's research focus areas are Bioinformatics & Computational Biology, Machine Learning, Deep Learning, and Big Data Analytics. He led the development of a software app called EyeCYou (http://eyecyouapp.com/) that uses AI to provide the facial description of a person to the visually impaired.

Justin Leo is a PhD student in the Department of Computer Science at the University of Colorado, Colorado Springs (UCCS). He participated in the REU program on Machine Learning at UCCS in 2019. He published a conference paper based on his REU work in 2020, followed by a recent paper on incremental class learning in IEEE Transactions on Neural Networks and Learning Systems.

NSF REU Seminar on Deep Learning

Department of Computer Science
University of Colorado, Colorado Springs
Engineering 101
August 6, 2021: Friday

10*:00-10:10 AM:* Welcome Remarks by Dr. Thomas Christensen, Professor of Physics, and Provost and Executive Vice Chancellor for Academic Affairs (2017-2021), University of Colorado, Colorado Springs

10:10-*11:25 AM* Session Chair*:* Dr. Thomas Halford, Chief Scientist, Caliola Engineering, Colorado Springs

- 10:10-10:35 *Abigail Swenor,* University of Colorado, Colorado Springs, CO: Using Random Perturbations to Mitigate Adversarial Attacks on NLP Models
- 10:35-11:00 *Wesley Robbins*, Montana State University, Bozeman, MT: Multimodal Vision-Language Models Generating Non-Generic Text
- 11:00-11:25 *Séraphin Bassas,* McGill University, Montreal, QC, Canada: Exploring Class Ordering Heuristics for Incremental Learning

11:25-11:35 Break

11:35-12:25 PM Session Chair: Dr. Philip Brown, Assistant Professor of Computer Science, University of Colorado, Colorado Springs, CO

- 11:35-12:00 *Parker Hicks*, Concordia University, Irvine, CA: HiCARN: Super Enhancement of Hi-C Contact Maps
- 12:00-12:25 *Trevor Martin*, Oberlin College, Oberlin, OH: EnsembleSplice: Ensemble Learning for Splice Site Prediction

12:25-1:25 PM: Lunch

1:25-2:40 PM Session Chair: Daniel Otter, B.S. in Computer Science and former REU student, University of Colorado, Colorado Springs, CO

- 1:25-1:50 *Theodore Zaremba*, University of Illinois, Chicago, IL: Classification of EEG Motor Imagery Tasks for Subject Transfer using Image-based 2D Convolutional Neural Networks
- *1*:50-2:15 D*aniel Theng*, California State University, Fresno, CA: EEG Data Restructuring and Combination for Image Representation for Deep Learning Motor Imagery Models to Subject-transfer
- *2*:15-2:40 *Devin Haggitt,* University of Colorado, Colorado Springs, CO: Generating Depth Maps from Image Pairs collected from Tello Drones

2:40 PM: Closing Remarks by Drs. Oluwatosin Oluwadare, Adham Atyabi and Jugal Kalita

Our Session Chairs and Guests

Dr. Adham Atyabi is an assistant professor in the Department of Computer Science, University of Colorado, Colorado Springs. His expertise are in cognitive and computational neuroscience, brain-computer interface, image and signal processing, and deep learning. At UCCS for 3 years. He obtained his PhD from Flinders University, Australia; and held post-doc positions at Yale and University of Washington. Dr. Atyabi has published 60 papers in various conferences and journals.

Dr. Philip Brown has been an assistant professor of computer science at UCCS since 2018; before this, he received his PhD from UC Santa Barbara. Philip's research uses game theory to optimize the interconnections between technology and society. Philip is a native of Colorado Springs and plans to compete in this year's Pike's Peak Marathon..

Dr. Thomas M. Christensen was the Provost and Executive Vice Chancellor for Academic Affairs at the University of Colorado at Colorado Springs from 2017-2021. He has served the campus as a faculty member, department chair, associate dean and dean. Dr. Christensen has received both the College and campus Outstanding Teaching Awards and the Chancellor's Award to recognize his service and teaching. Currently, he is a professor in the Department of Physics and Energy Sciences.

Dr. Thomas Halford is the Chief Scientist at Caliola Engineering, LLC. He is a researcher, fund-raiser, and "full stack" engineer focused on military and industrial communications. His company is developing products that enable assured and secure communications in austere environments. He was the Chief Scientist at WPL, Inc., before co-founding Caliola Engineering. He holds a PhD in Electrical Engineering from the University of Southern California.

Dr. Jugal Kalita is a Professor in the Department of Computer Science at the University of Colorado, Colorado Springs. His research areas are in Natural Language Processing, and Machine Learning and its application to diverse fields including Computer Vision and Security.

Dr. Oluwatosin Oluwadare is an Assistant Professor of Computer Science and Innovation at the University of Colorado, Colorado Springs. Dr. Oluwadare's research focus areas are Bioinformatics & Computational Biology, Machine Learning, Deep Learning, and Big Data Analytics. He led the development of a software app called EyeCYou (http://eyecyouapp.com/) that uses AI to provide the facial description of a person to the visually impaired.

Daniel Otter is a software engineer with four years experience in the astronautical industry, where he has also worked as a machine learning engineer. He is also an undergraduate Computer Science, Mechanical Engineering, and Mathematics student at UCCS, where he has worked as a research assistant in the Language, Information, and Computing (LINC) Lab. His research interests include natural language understanding topics such as text summarization and machine translation. His first paper was published in IEEE Transactions on Neural Networks and Learning Systems last year and has been cited more than 225 times.

Using Random Perturbations to Mitigate Adversarial Attacks on NLP Models

Abigail Swenor
University of Colorado Colorado Springs
1420 Austin Bluffs Pkwy
Colorado Springs, CO 80918
aswenor@uccs.edu

Jugal Kalita
University of Colorado Colorado Springs
1420 Austin Bluffs Pkwy
Colorado Springs, CO 80918
jkalita@uccs.edu

Abstract

Deep learning models have excelled in solving many difficult problems in Natural Language Processing (NLP), but it has been demonstrated that such models are susceptible to extensive vulnerabilities. There are attacks that can be concealed, and therefore are difficult to protect against. This problem is exacerbated by the use of public datasets that typically are not manually inspected before use. Some work has moved towards defending against these attacks during the training and testing phases of NLP models. In this paper, we offer a solution to this vulnerability by using random perturbations such as spell checking, synonym substitutions, and drop words to defend NLP models against adversarial attacks. Our Random Perturbations Defense and Increased Randomness Defense methods are successful in returning attacked models to similar accuracy of models before attacks. The original accuracy of the model used in this work is 80% for sentiment classification. After undergoing attacks, the accuracy drops to an accuracy between 0% and 44%. After applying our defense, the accuracy of the model is returned to the original accuracy within statistical significance.

1 Introduction

Deep learning models have excelled in solving difficult problems in many machine learning tasks, including Natural Language Processing (NLP) (Zhang, Zhao, and LeCun 2015; Kim 2014; Devlin et al. 2019). However, research has discovered that inputs can be modified to cause deep learning models to produce incorrect results and predictions (Szegedy et al. 2014). Models in computer vision are vulnerable to these attacks (Goodfellow, Shlens, and Szegedy 2015), and studies have found that models in the NLP domain are also vulnerable (Kuleshov et al. 2018; Gao et al. 2018; Garg and Ramakrishnan 2020). One use of these adversarial attacks is to test and verify the robustness of NLP models. While adversarial training (Goodfellow, Shlens, and Szegedy 2015) was first developed in computer vision, efforts are being made to provide centralized resources for NLP adversarial training and data augmentation (Morris et al. 2020).

With the potential for adversarial attacks, there comes the need for prevention and protection. There are three main categories of defense methods: identification, reconstruction, and prevention (Goldblum et al. 2020). Identification methods rely on detecting either poisoned data or the poisoned model (Chen et al. 2019). While reconstruction methods actively work to repair the model after training (Zhu et al. 2020), prevention methods rely on input preprocessing, majority voting, and other techniques to mitigate adversarial attacks (Goldblum et al. 2020; Alshemali and Kalita 2020). Although most NLP adversarial attacks are easily detectable, some new forms of adversarial attacks have become more difficult to detect (Wallace et al. 2021; Chen et al. 2020). The use of these concealed and hard-to-detect attacks has revealed new vulnerabilities in NLP models. Considering the increasing difficulty in detecting attacks, a more prudent approach would be to work on neutralizing the effect of potential attacks rather than solely relying on detection. Here we offer a novel and highly effective defense solution that preprocesses inputs by random perturbations to mitigate potential hard-to-detect attacks.

2 Related Work

The work in this paper relates to the attack on NLP models using the TextAttack library (Morris et al. 2020), the current state-of-the-art defense methods for NLP models, and using randomness against adversarial attacks.

2.1 TextAttack Library

The TextAttack library and the associated GitHub repository (Morris et al. 2020) represent current efforts to centralize attack and data augmentation methods for the NLP community. The TextAttack library allows researchers to better understand the state-of-the-art attack models and to create new kinds of attacks to test the robustness of NLP models. There is a standard command-line uage of the library and also an API. Both support attack creation through the use of four components: a goal function, a search method, a transformation, and constraints. An attack method uses these components to perturb the input to fulfill the given goal function while complying with the constraints, and the search method finds transformations that produce adversarial examples.

The library contains a total of 16 attack models based on literature. These attacks are presented as recipes that are readily available for use. There are 14 classification attack recipes and 2 sequence-to-sequence attack recipes available on the library. The work reported in this paper pertains to the ready-to-use classification attack recipes from the TextAttack library.

2.2 Input Preprocessing Defenses

There are many methods to defend NLP models against adversarial attacks, including input preprocessing. Input preprocessing defenses require inserting a step between the input and the given model that aims to mitigate any potential attacks. Alshemali and Kalita (2020) use an input preprocessing defense that employs synonym set averages and majority voting to mitigate synonym substitution attacks. Their method is deployed before the input is run through a model. This distinction is what makes the defense an input preprocessing method. Another defense against synonym substitution attacks, Random Substitution Encoding (RSE) encodes randomly selected synonyms to train a robust deep neural network (Wang and Wang 2020). The RSE defense occurs between the input and the embedding layer.

2.3 Randomness in Defense

Randomness has been deployed in computer vision defense methods against adversarial attacks. Levine and Feizi (2020) use random ablations to defend against adversarial attacks on computer vision classification models. Their defense is based on a random-smoothing technique that creates certifiably robust classification. Levine and Feizi defend against sparse adversarial attacks that perturb a determined number of features in the input images. They found their random ablation defense method to produce certifiably robust results on the MNIST, CIFAR-10, and ImageNet datasets.

3 Input Perturbation Approach & Adversarial Defense

The use and availability of successful adversarial attack methods reveal the need for defense methods that do not rely on detection and leverage intuitions gathered from popular attack methods to protect NLP models. In particular, we present a simple but highly effective defense against deep learning models that perform sentiment analysis.

The approach taken is based on certain assumptions about the sentiment analysis task. Given a short piece of text, we believe that a human does not need to necessarily analyze every sentence carefully to get a grasp on the sentiment. Our hypothesis is that humans can ascertain the expressed sentiment in a text by paying attention to a few key sentences, while skimming over the others. This thought experiment led us to make intermediate classifications on all sentences of a review in the IMDB dataset, and then combining the results for a collective final decision.

This process was refined further by considering how attackers actually perturb data. Usually, they select a small number of characters or tokens within the original data to perturb. To mitigate those perturbations, we choose to perform our own random perturbations. Because the attacking perturbations could occur anywhere within the original data, and we do not necessarily know where they are, it is prudent to randomly select tokens for us to perturb. This randomization has the potential to negate the effect the attacking perturbations have on the overall sentiment analysis.

We wish to highlight the importance of randomness in our approach and in possible future approaches for defenses against adversarial attacks. The impact of randomness can be found in work using Random Forests (Breiman 2001). Random Forests have been useful in many domains to make predictions, including disease prediction (Lebedev et al. 2014; Corradi et al. 2018; Paul et al. 2017; Khalilia, Chakraborty, and Popescu 2011) and stock market price prediction (Khaidem, Saha, and Dey 2016; Ballings et al. 2015; Nti, Adekoya, and Weyori 2019). The use of randomness has made these methods of prediction robust and useful. We have chosen to harness the capability of randomness in defense of adversarial attacks in NLP. We believe that the impact randomness has on our defense method is positive and its use in defense against adversarial attacks of neural networks should be explored further.

3.1 Algorithm

Our algorithm is based on a random process: the randomization of perturbations of the sentences of a review R followed by majority voting to decide the final prediction for sentiment analysis. We consider each review R to be represented as a set $R = \{r_1, r_2, ..., r_N\}$ of sentences r_i. Once R is broken down into its sentences (Line 1), we create l replicates of sentence r_i: $\{\hat{r}_{i1}, ..., \hat{r}_{ij}, ..., \hat{r}_{il}\}$. Each replicate $\hat{r}_{ij}$ has k number of perturbations made to it. Each perturbation is determined randomly (Lines 4-7).

In Line 5, a random token t where $t \in \hat{r}_{ij}$ is selected, and in Line 6, a random perturbation is performed on t. This random perturbation could be a spellcheck with correction, a synonym substitution, or a drop word. A spellcheck is performed using SpellChecker which is based in Pure Python Spell Checking. The synonym substitution is also performed in a random manner. A synonym set for token t is found using the WordNet function synsets (Fellbaum 1998). Once a synonym set is found, it is processed to remove any duplicate synonyms or copies of token t. Once the synonym set is processed, a random synonym from the set is chosen to replace token t in $\hat{r}_{ij}$. A drop word is when the randomly selected token t is removed from the replicate altogether and replaced with a space.

Once l replicates have been created for the given sentence r_i and perturbations made to tokens, they are put together to create replicate review set $\hat{R}$ (Line 8). Then, in Line 9, each $\hat{r}_{ij} \in \hat{R}$ is classified as $f(\hat{r}_{ij})$ using classifier $f()$. After each replicate has been classified, we perform majority voting with function $V()$. We call the final prediction that this majority voting results in as $\hat{f}(R)$. This function can be visualized as follows (Line 12):

$$\hat{f}(R) = V(\{f(\hat{r}_{ij}) \mid \hat{r}_{ij} \in \hat{R}\}).$$

The goal is to maximize the probability that $\hat{f}(R) = f(R)$ where $f(R)$ is the classification of the original review R. This maximization is done through tuning of the parameters l and k. The certainty T for $\hat{f}(R)$ is also determined for each calculation of $\hat{f}(R)$. The certainty represents how sure the algorithm is of the final prediction it has made. In general, the certainty T is determined as follows (Lines 13-17):

$$T = count(f(\hat{r}_{ij}) == \hat{f}(R)) \ / \ N * l.$$

The full visual representation of this algorithm can be seen in Algorithm 1 and in Figure 1.

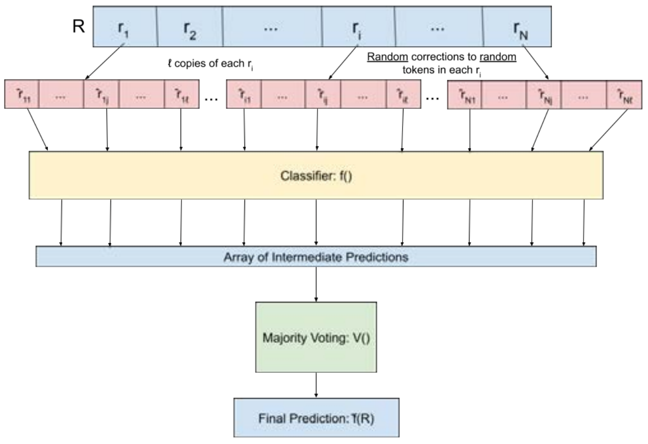


Figure 1: Visual representation of Algorithm 1.

Algorithm 1: Random Perturbation Defense

```
Result: $\hat{f}(R)$, the classification of $R$ after defense
Input      : Review $R = \{r_1, r_2, ..., r_N\}$ where $r_i$
             is a sentence
Parameters: $l$ = number of copies made of each $r$
             $k$ = number of corrections made per $r_i$
             $C = \{c_1, c_2, ..., c_k\}$, set of corrections
             $\hat{R} = \emptyset$
1  for $r_i \ \epsilon \ R$ do
2      for j = 1 to l do
3          $\hat{r}_{ij} = r_i$
4          for k do
5              Select random token $t$ where $t \ \epsilon \ \hat{r}_{ij}$
6              Perform random correction $c \ \epsilon \ C$ to $t$
7          end
8          Append $\hat{r}_{ij}$ to $\hat{R}$
9          Classify: $f(r_{ij})$
10     end
11 end
12 $\hat{f}(R) = V(\{f(\hat{r}_{ij}) \mid \hat{r}_{ij} \epsilon \hat{R}\})$, $V()$ is a voting
     function
13 if $f(\hat{R}) == negative$ then
14     $T = count(f(\hat{r}_{ij}) == negative) \ / \ N * l$
15 else
16     $T = count(f(\hat{r}_{ij} == positive) \ / \ N * l$
17 end
```

3.2 Increasing Randomness

Our first algorithm represented in Algorithm 1 and in Figure 1 shows randomness in two key points in the decision making process for making the perturbations. This is the main source of randomness for our first algorithm. In our next algorithm, we introduce more randomness into our ideas from our original algorithm to create a modified algorithm. This more random algorithm is visually represented in Figure 2 and presented in Algorithm 2. This new defense method adds a third random process before making random corrections to a sentence. Randomly chosen r_i from R are randomly corrected to create replicate $\hat{r}_j$ which is placed in $\hat{R}$ (Lines 2-6). The original sentence r_i is placed back into R and a new sentence is randomly selected; this is random selection with replacement. This process of random selection is repeated until there is a total of k replicates $\hat{r}_j$ in $\hat{R}$.

In Line 2, we randomly select a sentence r_i from R. This is one of the largest differences between Algorithm 1 for our Random Perturbations Defense and Algorithm 2 for our Increased Randomness Defense. That extra random element allows for more randomization in the corrections we make to create replicates $\hat{r}_j$. In Lines 3 and 4, the process is practically identical to Lines 5 and 6 in Algorithm 1. The only difference is that only one random correction is being made to get the final replicate $\hat{r}_j$ for our Increased Randomness Defense, while our Random Perturbations Defense makes k random corrections to get the final replicate $\hat{r}_{ij}$.

3.3 Overcoming the Attacks

We define an attack as making random perturbations to an input, specifically for this work, a review R. We assume a uniform distribution for randomness. We interpret these random changes to occur throughout each review R with probability $\frac{1}{W}$ or $\frac{1}{N*m}$, where W is the number of words in R, N is the number of sentences in R, and m is the average length of each sentence in R. We refer to this probability as P_{attack} where a is the total number of perturbations made by the attack:

$$P_{attack} = \frac{a}{W} = \frac{a}{N * m}.$$

If each random perturbation performed by the attack has a probability of $\frac{1}{N*m}$, then our defense method needs to overcome that probability to overcome the attack.

Our two defense methods, Random Perturbations Defense and Increased Randomness Defense, both offer ways to overcome the attack, i.e., undo the attack change, with a probability greater than $\frac{a}{N*m}$. Our Random Perturbations Defense picks a random token t from each sentence $r_i \in R$ and repeats k times to get a final replicate $\hat{r}_{ij}$. This gives an initial probability for P_{RPD} to be:

$$P_{RPD} = \frac{N * l * m!}{k!(m-k)!}.$$

We find this probability from choosing k tokens from r_i with length m which breaks down to a binomial coefficient $\binom{m}{k} = \frac{m!}{k!(n-k)!}$. This is then repeated l times for each sentence in R which equates to that initial probability being multiplied by l and N. After doing some rearranging of the

```
Algorithm 2: Increased Randomness Defense
  Result: $\hat{f}(R)$, the classification of $R$ after defense
  Input      : Review $R = \{r_1, r_2, ..., r_N\}$ where $r_i$
               is a sentence
  Parameters: $k$ = number of replicates $\hat{r}_j$ made for $\hat{R}$
               $C = \{c_1, c_2, ..., c_k\}$, set of corrections
               $\hat{R} = \emptyset$, $P = []$
for j = 1 to k do
    Randomly select $r_i \in R$
    Select random token $t$ where $t \in r_i$
    Perform random correction $c \epsilon C$ to $t$ to get $\hat{r}_j$
    Append $\hat{r}_j$ to $\hat{R}$
end
for j = 1 to k do
    Classify: $f(\hat{r}_j)$
    Append results to predictions array $P$
end
$\hat{f}(R) = V(P)$, $V()$ is a voting function
if $f(\hat{R}) == negative$ then
    $T = count(f(\hat{r}_{ij}) == negative) \ / \ N * l$
else
    $T = count(f(\hat{r}_{ij} == positive) \ / \ N * l$
end
```

probabilities, we can see that for certain values of l and k where $k < m$:

$$P_{RPD} = \frac{N^2 m^2 l(m-1)(m-2)...(m-k+1)}{k!} > a.$$

We know that $W = N * m$, that $a = W$ only for one of our attack methods, and that k should be selected so that $k << W$. This means that generally, $W^2 > a$, $W^2 > k!$, and $l(m-1)(m-2)...(m-k+1) > 0$, which gives us the necessary conditions to assert that $P_{RPD} > Pattack$. Therefore, our Random Perturbations Defense will overcome the P_{attack} and should overcome the given attack method.

Our Increased Randomness Defense first chooses a random sentence r_i which is selected with probability $\frac{1}{N}$. Next, we choose a random word within that sentence which is selected with probability $\frac{1}{m}$. This gives us a probability for changes as follows:

$$P_{IRD} = \frac{1}{N} * \frac{1}{m} = \frac{1}{N * m}.$$

We can see that $P_{IRD} * a = Pattack$. We need to overcome the attack probability and we do this in two ways: we either find the attack perturbation by chance and reverse it, or we counter balance the attack perturbation with enough replicates $\hat{r}_j$. With each replicate $\hat{r}_j$ created, we increase our probability P_{IRD} so that our final probability for our Increased Randomness Defense is as follows:

$$P_{IRD} = \frac{k}{N * m}.$$

As long as our selected parameter value for k is greater than the number of perturbation changes made by the attack method a, then $P_{IRD} > P_{attack}$ and our Increased Randomness Defense method will overcome the given attack method.

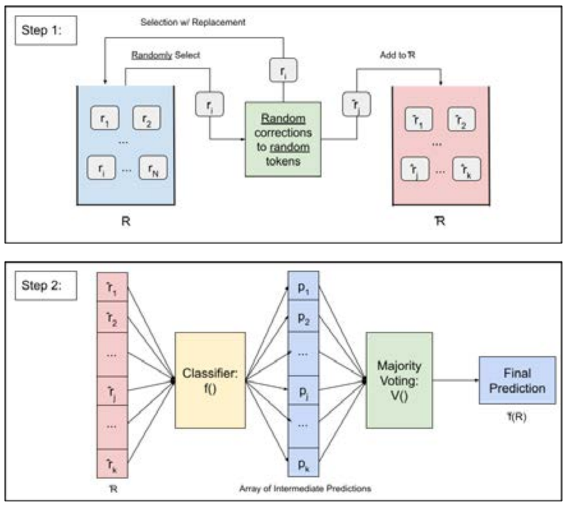

Step 1:
Selection w/ Replacement
Randomly Select
Add to R̂
r_1 r_2 ... r_i ... r_N
r_i
Random corrections to random tokens
$\hat{r}_j$
$\hat{r}_1$ $\hat{r}_2$ ... $\hat{r}_j$ ... $\hat{r}_k$
R
R̂
Step 2:
$\hat{r}_1$ $\hat{r}_2$ ... $\hat{r}_j$ ... $\hat{r}_k$
Classifier: f()
p_1 p_2 ... p_j ... p_k
Majority Voting: V()
Final Prediction
f̂(R)
R̂
Array of Intermediate Predictions

Figure 2: Visual representation of Algorithm 2 that includes more randomness.

4 Experiments & Results

4.1 Dataset

We used the IMDB dataset (Maas et al. 2011) for our experiments. Each attack was used to perturb 100 reviews from the dataset. The 100 reviews were selected randomly from the dataset with a mix of positive and negative sentiments. Note that the Kuleshov attack data (Kuleshov et al. 2018) only had 77 reviews.

4.2 Models

The models used in this research are from the TextAttack (Morris et al. 2020) and HuggingFace (Wolf et al. 2020) libraries. These libraries offer many different models to use for both attacked data generation and general NLP tasks. For this research, we used the bert-base-uncased-imdb model that resides in both the TextAttack and HuggingFace libraries. This model was fine-tuned and trained with a cross-entropy loss function. This model was used with the API functions of the TextAttack library to create the attacked reviews from each of the attacks we used. We chose this model because BERT models are useful in many NLP tasks and this model specifically was fine-tuned for sequence classification and was trained on the dataset we wanted to use for these experiments.

The HuggingFace library was also used in the sentiment-analysis classification of the attacked data and the defense method. We used the HuggingFace transformer pipeline for sentiment-analysis to test our defense method. This pipeline returns either “negative” or “positive” to classify the sentiment of the input text and a score for that prediction (Wolf et al. 2020). This pipeline was used to classify each replicate $\hat{r}_{ij}$ in our algorithm and is represented as the function $f()$.

4.3 Experiments

The attacks from the TextAttack library were used to generate attack data. Attack data was created from 7 different models from the library: BERT-based Adversarial Examples (BAE) (Garg and Ramakrishnan 2020), DeepWordBug (Gao et al. 2018), FasterGeneticAlgorithm (Jia et al. 2019), Kuleshov (Kuleshov et al. 2018), Probability Weighted Word Saliency (PWWS) (Ren et al. 2019), TextBugger (Li et al. 2019), and TextFooler (Jin et al. 2020) (Morris et al. 2020). Each of these attacks were used to create 100 perturbed sentences from the IMDB dataset (Maas et al. 2011). These attacks were chosen from the 14 classification model attacks because they represent different kinds of attack methods, including misspelling, synonym substitution, and antonym substitution.

Each attack method used for our experiments has a slightly different approach to perturbing the input data. Each perturbation method is unique and follows a specific distinct pattern. The BAE attack determines the most important token in the input and replaces that token with the most similar replacement using a Universal Sentence Encoder. This helps the perturbed data remain semantically similar to the original input (Garg and Ramakrishnan 2020). The DeepWordBug attack identifies the most important tokens in the input and performs character-level perturbations on the highest-ranked tokens while minimizing edit distance to create a change in the original classification (Gao et al. 2018). The FasterGeneticAlgorithm perturbs every token in a given input while maintaining the original sentiment. It chooses each perturbation carefully to create the most effective adversarial example (Jia et al. 2019). The Kuleshov attack is a synonym substitution attack that replaces 10% - 30% of the tokens in the input with synonyms that do not change the meaning of the input (Kuleshov et al. 2018).

The PWWS attack determines the word saliency score of each token and performs synonym substitutions based on the word saliency score and the maximum effectiveness of each substitution (Ren et al. 2019). The TextBugger attack determines the important sentences from the input first. It then determines the important words in those sentences and generates 5 possible "bugs" through different perturbation methods: insert, swap, delete, sub-c (visual similarity substitution), sub-w (semantic similarity substitution). The attack will implement whichever of these 5 generated bugs is the most effective in changing the original prediction (Li et al. 2019). Finally, the TextFooler attack determines the most important tokens in the input using synonym extraction, part-of-speech checking, and semantic similarity checking. If there are multiple canididates to substitute with, the most semantically similar substitution will be chosen and will replace the original token in the input (Jin et al. 2020).

After each attack had corresponding attack data, the TextAttack functions gave the results for the success of the attack. The accuracy of the sentiment-analysis task under attack, without the defense method, is reported in the first column in Table 3. Each attack caused a large decrease in the accuracy of the model. The model began with an average accuracy of 80% for the IMDB dataset. Once the attack data was created and the accuracy under attack was reported, the attack data was ran through our Random Perturbations and our Increased Randomness defense methods.

4.4 Results

We began by testing on the HuggingFace sentiment analysis pipeline with the original IMDB dataset. This gave an original accuracy of 80%. This percentage represents the goal for our defense method accuracy as we aim to return the model to its original accuracy, or higher. The accuracy under each attack is listed in Table 3 in the first column. These percentages show how effective each attack is at causing misclassification for the sentiment analysis task. The attacks range in effectiveness with PWWS (Ren et al. 2019) and Kuleshov (Kuleshov et al. 2018) with the most successful attacks at 0% accuracy under attack and FasterGeneticAlgorithm (Jia et al. 2019) with the least successful attack at 44% accuracy under attack, which is still almost a 40% drop in accuracy.

Average Synonym Embeddings Defense We had initially adapted a defense method from Alshemali and Kalita (2020) to test against the TextAttack library attack methods. This defense method was originally created to mitigate synonym substitution attacks using a form of input preprocessing. The defense method substitutes the important words in the input with the average of their synonym set. Those changed inputs are then classified and those predictions are used for majority voting to determine the final prediction for the task. When using this defense method on the created perturbed data from a couple of the TextAttack attack methods, the accuracy of the model did not return to the original accuracy of 80% and were much lower than expected with no clear way to improve them. These tests were performed multiple times with the same results for every test for the given attack method. The results from these initial experiments can be found in Table 1. Since the defense was credible but not as strong as we wanted it to be, we decided to explore additional defenses, based on randomization.

Attack	w/o Defense	w/ Defense
BAE	33%	70%
PWWS	0%	65%

Table 1: Accuracy for each of the attack methods under attack and under attack with the adapted defense from Alshemali and Kalita (2020) deployed. The accuracy prior to attack is 80%.

Random Perturbations Defense For the Random Perturbations Defense to be successful, it is necessary to obtain values of the two parameters, l and k. Each attack was tested against our Random Perturbations Defense 5 times. The accuracy was averaged for all 5 tests and the standard deviation was calculated for the given mean. The mean accuracy with standard deviation is presented for each attack in the second column of Table 3. The results presented are for $l = 7$ and $k = 5$. These parameters were chosen after testing found greater values of l and k resulted in a longer run time and too many changes made to the original input; with lower values for l and k, the model had lower accuracy and not enough perturbations to outweigh any potential adversarial attacks. The values behind this logic can be seen in Table 2.

Attack	l	k	Accuracy w/ Defense
BAE	5	2	55%
BAE	10	5	50%
BAE	7	5	79%

Table 2: This table explains values of l and k

The defense method was able to return the model to original accuracy within statistical significance while under attack for most of the attacks with the exception of the Kuleshov method (Kuleshov et al. 2018). The accuracy for the other attacks all were returned to the original accuracy ranging from 76.00% to 83.20% accuracy with the Random Perturbations defense deployed. This shows that our defense method is successful at mitigating most potential adversarial attacks on sentiment classification models. Our defense method was able to increase the accuracy of model while under attack for the FasterGeneticAlgorithm, PWWS, and TextFooler. These three attack methods with our defense achieved accuracy that was higher than the original accuracy with statistical significance.

Attack	w/o Defense	w/ Defense
BAE	33%	80.80%±1.47
DeepWordBug	34%	76.60%±1.85
FasterGeneticAlgo	44%	82.20%±1.72
Kuleshov*	0%	60.00%±2.24
PWWS	0%	81.80%±1.17
TextBugger	6%	79.20%±2.32
TextFooler	1%	83.20%±2.48

Table 3: Accuracy for each of the attack methods under attack and under attack with the defense method from Algorithm 1 deployed with $l = 7$ and $k = 5$. The accuracy prior to attack is 80%.

Increased Randomness Defense The Increased Randomness Defense was also tested on by all seven of the attacks. Each attack was tested against this defense 5 times. The results for these experiments can be seen in Table 5. There were tests done to determine what the proper value for k should be. These tests were performed on the BAE (Garg and Ramakrishnan 2020) attack and the results can be found in Table 4. These tests revealed that 40-45 replicates $\hat{r}_j$ was ideal for each $\hat{R}$ with $k = 41$ being the final value used for the tests on each attack. This defense method was more efficient to use.

The runtime and the resources used for this method were lower than the original random perturbations defense method with the runtime for the Random Perturbations Defense being nearly 4 times longer than this increased random method. A comparison of the two defense methods on the seven attacks tested can be seen in Figure 3. This defense was successful in returning the model to the original

Attack	k	Accuracy w/ Defense
BAE	10	67%
BAE	20	76%
BAE	25	72%
BAE	30	76%
BAE	35	74%
BAE	40	82%
BAE	45	74%
BAE	41	77%

Table 4: This table shows the results for the tests for different values of k for the increased randomness experiments.

accuracy, within statistical significance, for most of the attacks with the exception of the Kuleshov attack (Kuleshov et al. 2018). A t-test was performed to determine the statistical significance of the difference in the defense method accuracy to the original accuracy.

Attack	w/o Defense	w/ Defense
BAE	33%	78.40%±3.14
DeepWordBug	34%	76.80%±2.64
FasterGeneticAlgo	44%	82.80%±2.48
Kuleshov*	0%	66.23%±4.65
PWWS	0%	79.20%±1.72
TextBugger	6%	77.00%±2.97
TextFooler	1%	80.20%±2.48

Table 5: Accuracy for increased randomness defense from Algorithm 2 against each attack method with $k = 41$. The accuracy prior to attack is 80%.

5 Conclusion

The work in this paper detail a useful defense method against adversarial attacks generated from the TextAttack library. These attack methods use multiple different perturbation approaches to change the predictions made by NLP models. Our defense methods utilized randomization to mitigate these adversarial attacks. Our Random Perturbations Defense was successful in mitigating attacks from the following methods: BAE, DeepWordBug, FasterGeneticAlgorithm, PWWS, TextBugger, and Textfooler. This defense method returned the attacked models to their original accuracy within statistical significance. Our second method, Increased Randomness Defense, used more randomization to create an equally successful defense method that was 4 times more efficient than our Random Perturbations Defense. Overall, our defense methods are successful and effective in mitigating a wide range of NLP adversarial attacks, presenting evidence for the effectiveness of randomness in NLP defense methods.

Acknowledgement

The work reported in this paper is supported by the National Science Foundation under Grant No. 2050919. Any

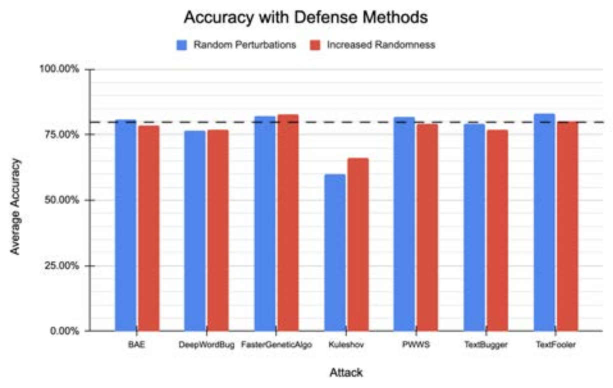


Figure 3: Comparing the average accuracy of the Random Perturbations Defense and the Increased Randomness Defense methods on the seven attacks.

opinions, findings and conclusions or recommendations expressed in this work are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

Alshemali, B., and Kalita, J. 2020. Generalization to mitigate synonym substitution attacks. In *Proceedings of Deep Learning Inside Out (DeeLIO): The First Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, 20–28.

Ballings, M.; Van den Poel, D.; Hespeels, N.; and Gryp, R. 2015. Evaluating multiple classifiers for stock price direction prediction. *Expert systems with Applications* 42(20):7046–7056.

Breiman, L. 2001. Random forests. *Machine learning* 45(1):5–32.

Chen, H.; Fu, C.; Zhao, J.; and Koushanfar, F. 2019. Deepinspect: A black-box trojan detection and mitigation framework for deep neural networks. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, 4658–4664. International Joint Conferences on Artificial Intelligence Organization.

Chen, X.; Salem, A.; Backes, M.; Ma, S.; and Zhang, Y. 2020. Badnl: Backdoor attacks against NLP models. *arXiv preprint arXiv:2006.01043*.

Corradi, J. P.; Thompson, S.; Mather, J. F.; Waszynski, C. M.; and Dicks, R. S. 2018. Prediction of incident delirium using a random forest classifier. *Journal of medical systems* 42(12):1–10.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT (1)*.

Fellbaum, C. 1998. *WordNet: An Electronic Lexical Database*. Bradford Books.

Gao, J.; Lanchantin, J.; Soffa, M. L.; and Qi, Y. 2018. Black-box generation of adversarial text sequences to evade deep learning classifiers. In *2018 IEEE Security and Privacy Workshops (SPW)*, 50–56. IEEE.

Garg, S., and Ramakrishnan, G. 2020. Bae: Bert-based adversarial examples for text classification. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 6174–6181.

Goldblum, M.; Tsipras, D.; Xie, C.; Chen, X.; Schwarzschild, A.; Song, D.; Madry, A.; Li, B.; and Goldstein, T. 2020. Data security for machine learning: Data poisoning, backdoor attacks, and defenses. *arXiv preprint arXiv:2012.10544*.

Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2015. Explaining and harnessing adversarial examples. *stat* 1050:20.

Jia, R.; Raghunathan, A.; Göksel, K.; and Liang, P. 2019. Certified robustness to adversarial word substitutions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 4129–4142.

Jin, D.; Jin, Z.; Zhou, J. T.; and Szolovits, P. 2020. Is bert really robust? a strong baseline for natural language attack on text classification and entailment.

Khaidem, L.; Saha, S.; and Dey, S. R. 2016. Predicting the direction of stock market prices using random forest. *arXiv preprint arXiv:1605.00003*.

Khalilia, M.; Chakraborty, S.; and Popescu, M. 2011. Predicting disease risks from highly imbalanced data using random forest. *BMC medical informatics and decision making* 11(1):1–13.

Kim, Y. 2014. Convolutional neural networks for sentence classification. In *EMNLP*.

Kuleshov, V.; Thakoor, S.; Lau, T.; and Ermon, S. 2018. Adversarial examples for natural language classification problems.

Lebedev, A.; Westman, E.; Van Westen, G.; Kramberger, M.; Lundervold, A.; Aarsland, D.; Soininen, H.; Kłoszewska, I.; Mecocci, P.; Tsolaki, M.; et al. 2014. Random forest ensembles for detection and prediction of alzheimer's disease with a good between-cohort robustness. *NeuroImage: Clinical* 6:115–125.

Levine, A., and Feizi, S. 2020. Robustness certificates for sparse adversarial attacks by randomized ablation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 4585–4593.

Li, J.; Ji, S.; Du, T.; Li, B.; and Wang, T. 2019. Textbugger: Generating adversarial text against real-world applications. In *26th Annual Network and Distributed System Security Symposium*.

Maas, A. L.; Daly, R. E.; Pham, P. T.; Huang, D.; Ng, A. Y.; and Potts, C. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 142–150. Portland, Oregon, USA: Association for Computational Linguistics.

Morris, J.; Lifland, E.; Yoo, J. Y.; Grigsby, J.; Jin, D.; and Qi, Y. 2020. Textattack: A framework for adversarial attacks, data augmentation, and adversarial training in NLP. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 119–126.

Nti, K. O.; Adekoya, A.; and Weyori, B. 2019. Random forest based feature selection of macroeconomic variables for stock market prediction. *American Journal of Applied Sciences* 16(7):200–212.

Paul, D.; Su, R.; Romain, M.; Sébastien, V.; Pierre, V.; and Isabelle, G. 2017. Feature selection for outcome prediction in oesophageal cancer using genetic algorithm and random forest classifier. *Computerized Medical Imaging and Graphics* 60:42–49.

Ren, S.; Deng, Y.; He, K.; and Che, W. 2019. Generating natural language adversarial examples through probability weighted word saliency. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, 1085–1097.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2014. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014*.

Wallace, E.; Zhao, T.; Feng, S.; and Singh, S. 2021. Concealed data poisoning attacks on nlp models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 139–150.

Wang, Z., and Wang, H. 2020. Defense of word-level adversarial attacks via random substitution encoding. In Li, G.; Shen, H. T.; Yuan, Y.; Wang, X.; Liu, H.; and Zhao, X., eds., *Knowledge Science, Engineering and Management*, 312–324. Cham: Springer International Publishing.

Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; Davison, J.; Shleifer, S.; von Platen, P.; Ma, C.; Jernite, Y.; Plu, J.; Xu, C.; Scao, T. L.; Gugger, S.; Drame, M.; Lhoest, Q.; and Rush, A. M. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 38–45. Online: Association for Computational Linguistics.

Zhang, X.; Zhao, J.; and LeCun, Y. 2015. Character-level convolutional networks for text classification. *Advances in neural information processing systems* 28:649–657.

Zhu, L.; Ning, R.; Wang, C.; Xin, C.; and Wu, H. 2020. Gangsweep: Sweep out neural backdoors by gan. In *Proceedings of the 28th ACM International Conference on Multimedia*, MM '20, 3173–3181. New York, NY, USA: Association for Computing Machinery.

Towards Multimodal Vision-Language Models Generating Non-Generic Text

Wes Robbins
Montana State University
100 Culbertson Hall
Bozeman, MT 59717
wesley.robbins@students.montana.edu

Zanyar Zohourianshahzadi & Jugal Kalita
University of Colorado, Colorado Springs
1420 Austin Bluffs Pkwy
Colorado Springs, CO 80918
{zzohouri,jkalita}@uccs.edu

Abstract

Vision-language models can assess visual context in an image and generate descriptive text. While generated text may be accurate and syntactically correct, it is often overly general. Recent work has used optical character recognition to supplement visual information with text extracted from an image. In many cases, using text in the image improves the specificity and usefulness of generated text. In this work, we contend that vision-language models can benefit from additional sets of tokens that can be extracted from an image, but are ignored by current models. We modify previous multimodal frameworks to accept relevant information from any number of auxiliary classifiers. In particular, we focus on person names as an additional set of tokens and create a novel image-caption dataset to facilitate this task. The dataset, Politicians and Athletes in Captions (PAC), consists of captioned images of well-known people in context. By fine-tuning pretrained models with this dataset, we demonstrate a model that can naturally integrate facial recognition tokens into generated text by training on only a few samples. For the PAC dataset, we provide an in-depth discussion about collection, analysis, and fairness. Finally, we present baseline benchmark scores on PAC.

1 Introduction

Vision-language models combine deep learning techniques from computer vision and natural language processing to assimilate visual and textual understanding. Models in this domain can demonstrate visual and linguistic knowledge by performing tasks such as vision question answering (VQA) and image captioning. There are many applications of these tasks, including aiding the visually impaired by providing scene information and screen reading (Morris et al. 2018).

To perform a vision-language task a model needs to understand visual context and natural language, and operate in a shared embedding space between the two. To transcribe visual information to text, an encoder-decoder architecture is trained to learn the necessary shared embeddings. Approaches in the literature have improved performance by pre-training models for both visual context and language understanding (Chen et al. 2020; Lu et al. 2019a; Su et al. 2019; Li et al. 2020; Tan and Bansal 2019). These frameworks have yielded accurate and semantically appropriate VQAs or captions. However, the text generated from these are general and overlook clues that allow for richer text generation with improved contextualization.

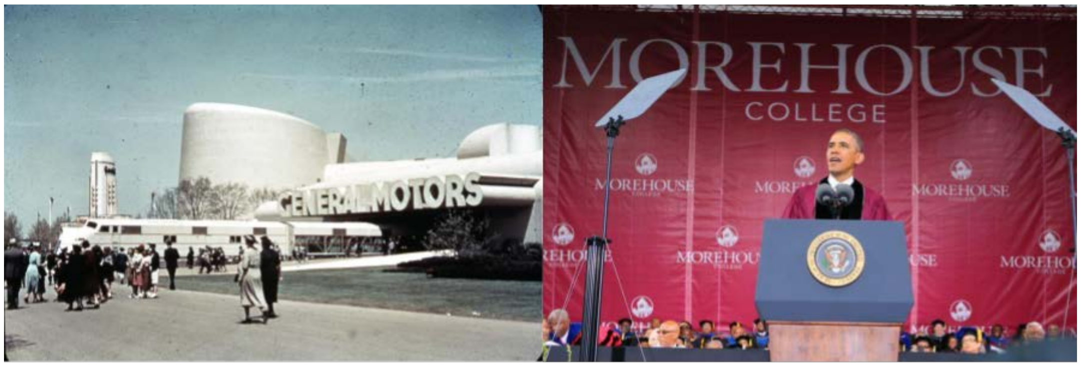
Recent work has used optical character recognition (OCR) in order to incorporate text that appears in images (Zhu et al. 2021; Gao et al. 2020b; Mafla et al. 2021; Hu et al. 2020; Kant et al. 2020; Wang et al. 2021; Han, Huang, and Han 2020; Liu et al. 2020). In many cases, this significantly enhances the usefulness of the generated text (Hu et al. 2020). Such frameworks include OCR as an additional input modality. This results in three modalities for VQA (image, question, and OCR) and two modalities for image captioning (image and OCR). Attention based mechanisms have been successful in integrating these multiple modalities for vision-language tasks.

While using OCR allows enhancement of some generated text, specific information that exists in human-level description may also come from additional sources. Without proper nouns or other specific vocabulary, the generated text is at risk the of being awkwardly general, demonstrating a lack of shared knowledge that is expected throughout society. Figure 1 show two images where using specific terms is critical to human like descriptions.

A parallel field of study is entity-aware image captioning. The focus of this task is to extract relevant information from an image-article pair to generate a caption. While models in this domain can generate captions with non-generic text, they rely on an article for specific vocabulary rather than strictly on the image content.

In this work, we propose a method for entity-aware text generation that can be based solely on image content. We propose generalizing the OCR input modality to accept helpful tokens from any number of auxiliary classifiers. This framework allows a model to leverage easily available sophisticated libraries for tasks like face recognition and OCR extraction. We refer to all tokens from upstream sources, including OCR tokens, as special tokens.

We focus on person names as an example special token. To facilitate this task we create a novel image-caption dataset, Politicians and Athletes in Captions (PAC), which includes person names in captions in addition to relevant scene-text found on signs or labels. PAC has 1,572 images and three captions per image. A full discussion on the dataset is provided in Section 4.



An image from *TextCaps*.

Human caption: A bunch of people are standing outside the General Motors building.

An image from *PAC*.

Human caption: "Barack Obama speaks at Morehouse College."

Figure 1: In both human captions, non-generic terms are used. Incorporation of external vocabulary tokens is decisive to whether a vision-language model can generate similar text.

By training on PAC in addition to other image-caption datasets, we create a model that can naturally integrate person names into captions. The same model still performs well on more general image captioning tasks. We show this can be learn by training on a limited number of samples. Evaluation of the methods is available in Section 5.

In summary, this paper makes the following contributions:

1. Proposes special tokens as a framework to incorporate tokens from external sources into generated text.
2. Releases PAC image-captioning dataset and baseline results.
3. Demonstrates a model that integrates OCR and facial recognition into image captioning.

2 Related Work

The encoder-decoder architecture divides the image captioning task into two parts. The encoder acts as feature extractor and the decoder handles word generation. Before the advent of attention mechanism, deep learning models for image captioning used CNN encoders for feature extraction from the whole input image as a whole (Kiros, Salakhutdinov, and Zemel 2014; Karpathy, Joulin, and Fei-Fei 2014; Vinyals et al. 2015).

Attention mechanism was first introduced in encoder-decoder architecture for neural machine translation (Bahdanau, Cho, and Bengio 2015). Enabling the model to perform better translation due to its ability to focus on the relevant parts of the input for generating the output at each time step. The seminal image captioning model, Show, Attend and Tell (Xu et al. 2015), applied attention mechanism on input visual features and previously generated word (during inference) at each time step for textual caption word generation. Soft and hard visual attention were the first types of attention mechanisms used for image captioning.

The majority of current state-of-the-art methods for image captioning and visual question answering benefit from the bottom-up and top-down attention mechanism (Anderson et al. 2018). Bottom-up and top-down attention was introduced in the context of encoder-decoder architecture (Sutskever, Vinyals, and Le 2014) for image captioning and visual question answering .

Bottom-up attention acts as a hard attention mechanism and leverages an object detector, Faster R-CNN (Ren et al. 2015) for detecting the most important regions in the image (Anderson et al. 2018). Top-down attention acts as a soft attention mechanism as it performs a soft modulation over the set of input visual features from object detection regions. Similar to how bottom-up attention was adopted for feature extraction in OCR (Hu et al. 2020) we adopt this mechanism for feature extraction in facial recognition. Rather than Faster R-CNN, we use RFBNet (Liu, Huang, and Wang 2018) facial region detection. For facial feature extraction we use ArcFace (Deng et al. 2019) pre-trained on MegaFace dataset (Kemelmacher-Shlizerman et al. 2016).

Several techniques have been proposed to handle OCR tokens in vision-language tasks. The M4C algorithm uses an indiscriminate attention layer followed by a dynamic pointer (Hu et al. 2020). The SS-Baseline model uses individual attention blocks for each input modality followed by a single fusion encoding layer (Zhu et al. 2021). Several approaches have been proposed to better handle spatial information about OCR tokens (Gao et al. 2020b; 2020a; Wang et al. 2021; Kant et al. 2020; Han, Huang, and Han 2020). The MMR method was proposed to utilize spacial information of objects and scene-text via a graph structure (Mafla et al. 2021). TextOCR was introduced as an end-to-end method for identifying OCR tokens (Singh et al. 2021). TAP was introduced as a method to integrate OCR tokens into pre-training. We adopt M4C as a base model for our work. Modifications are enumerated in Section 3.

Entity aware image captioning focuses on creating captions from image-article pairs (Biten et al. 2019a; Tran, Mathews, and Xie 2020; Lu et al. 2018). Our work is distinct in that it focuses on generating captions strictly from visual information rather than articles. More similar to our work, Zhao et al. uses an upstream vision classifier as input to a captioning model. They introduce a multi-gated decoder for handling input from external classifiers (Zhao et al. 2019). Comparatively we use general OCR and facial recognition classifiers rather than a web entity recognizer as an upstream classifier. Our approach is different from Zhao et al. in that we use bottom-up and top-down attention rather than a stand alone CNN for object detection, use a common embedding space rather than a gated decoder for handling multi-modal inputs, and use rich representations (see Section 3.2) rather than only textual information for handling tokens from upstream classifiers.

MS-COCO (Lin et al. 2014) is a large dataset for common objects in context used for image captioning. Similar to MS-COCO, Flickr30k (Young et al. 2014) is another common dataset used for image captioning. For MS-COCO, Karpathy's split (Karpathy and Fei-Fei 2015) is used as a common benchmark for image captioning. Google's conceptual captions (Sharma et al. 2018) is a vast dataset used for pre-training multitasking vision-language models and fine-tuning them on other vision-language down stream tasks (Lu et al. 2019b; 2020).

To facilitate use of optical character recognition in the Vision-Language domain, several datasets have been released, including ST-VQA (Biten et al. 2019b) for scene text visual question answering and TextCaps (Sidorov et al. 2020) for image captioning with reading comprehension. Along with the introduction of TextCaps dataset, the M4C model (Hu et al. 2020) originally used for visual question answering was adopted for image captioning. We modify the M4C model via adding another modality of information that includes bottom-up facial recognition features.

3 Special Tokens

We propose special tokens as a placeholder for extracted textual information that is identified as present in an image by an upstream source and then subsequently used by a vision-language model. In this approach there are two modalities that hold information about an image. The first modality is generic visual features which are responsible for informing the model of general context. The second modality, special tokens, is responsible for informing the model of specific terms that are relevant to the image. The first modality is represented by visual feature vectors where as the special token modality consists of visual feature vectors and textual feature vectors. Special tokens are additionally made available for direct copy into generated text which allows for zero-shot inclusion of words not seen prior. This structure has been successful on OCR vision-language datasets. The key hypothesis of this paper is that a model can further learn to differentiate types of tokens with in the special token modality and subsequently use each token type appropriately. Passing all tokens from upstream sources through the same modality allows the model to accept any tokens from any number of sources at inference time. A multi-modal transformer is used such that all special tokens can attend to each other during training and inference. This is the substrate for which a model can not only learn to differentiate tokens, but also learn relationships between tokens and what effect the presence of each token should have on generated text.

In this approach, OCR tokens are placed under the umbrella of special tokens and received along side tokens from other upstream sources. In this work, we demonstrate adding facial recognition tokens received from a face recognition module. We focus our experimentation on learning to integrate facial recognition tokens by training on the PAC dataset. However, any set of words that can be identified by some module can conceivably be a set of special tokens.

3.1 Motivation

The goal of special tokens is to integrate vocabulary tokens from external sources into generated text. An opposing approach would be to use an enhanced object detector for all detection and correspondingly extend the model vocabulary to include all desired vocabulary tokens. The special tokens approach can be motivated based on several following observations.

1) Different architectures perform better on different tasks. Several tasks, such as OCR detection and facial recognition, benefit from specialized approaches that differ from traditional object detection. In OCR, detection finds characters individually rather than classifying words. In facial recognition, a regression model is trained to output face embeddings which are subsequently compared to embeddings of known individuals. Even on standard classification tasks, significant research is put into fine-tuning architectures to get state of the art results on dataset benchmarks. This work can be leveraged by utilizing the fine-grained classifiers as upstream sources.

2) The space complexity of all possible vocabulary tokens is intractably large. By appending special tokens to the vocabulary at inference time the captioning model's vocabulary is prevented from inflating.

3) Using non-generic terms does not always increase the complexity of the caption. For example in Figure 1, the names 'General Motors' and 'Barack Obama' are substitutions for what could have been generic terms such as building or person. If a captioning model can generate a caption such as 'A person giving a speech', it does not need a significant increase in contextual understanding to generate the caption 'Barack Obama giving a speech.' Rather, the model just needs to know to *use* 'Barack Obama'. The special token approach takes advantage of this and allows the model to learn these correlations in few iterations and on few samples. At the same time, the special token approach does not limit the model from learning more nuanced correlations between tokens and context assuming rich enough data is provided.

4) Not all applications will expect the same special tokens. A captioning model that supplements screen reading for the visually impaired deployed in one area of the world may desire different sets or subsets of tokens than the same model deployed in a different part of the world. In this sense special tokens are highly practical. The same model can be transferred between applications simply by adding or detracting the external sources used for generating tokens.

3.2 Rich Representations

Utilizing the same representation for all special tokens allows the model to accept tokens from any external source.

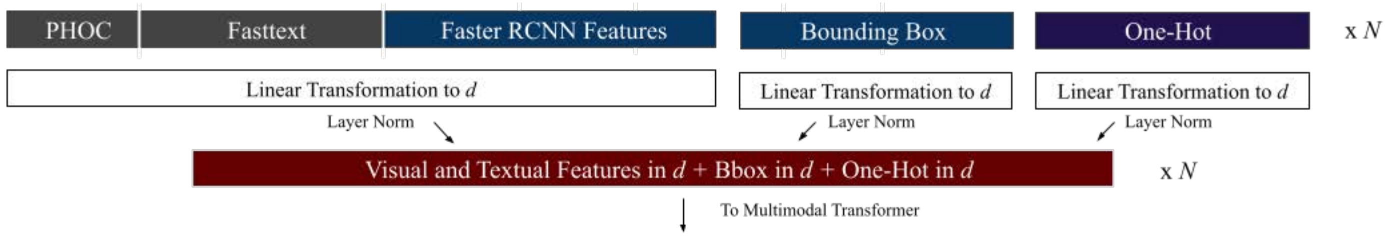

PHOC
Fasttext
Faster RCNN Features
Bounding Box
One-Hot
x N
Linear Transformation to d
Linear Transformation to d
Linear Transformation to d
Layer Norm
Layer Norm
Layer Norm
Visual and Textual Features in d + Bbox in d + One-Hot in d
x N
To Multimodal Transformer

Figure 2: The representation of a special token where N is the number of tokens and d is the tuned dimensionality. We adopt the representation from Hu et al. and add the projected one-hot encoding classifier feature.

The representation encodes several types of information about the token allowing the model to learn how different components of a token effect its use. We adopt the the visual and textual feature representation from Hu et al. and add a token source feature (Hu et al. 2020). A formal description of the special representation is described below and a visual representations is provided in Figure 2.

Special tokens are represented by a feature vector x_i^{spec}, where $i = 1...N$. x_i^{spec} incorporates visual features, textual features, and a classifier feature. The visual features include a bounding box x_i^b and a feature vector from an object detector x_i^{fr}. Following previous work we use a pretrained Faster-RCNN with a ResNet backbone to generate x_i^{fr} from the RoI created by the tokens bounding box. The textual features are a fastText encoding x_i^{ft} and a pyramidal hierarchy of characters (PHOC) encoding x_i^p. The classifier feature x_i^c is a one-hot encoding of sources used for generating special tokens. x_i^{fr}, x_i^{ft}, and x_i^p are concatenated together and projected into a tuned encoding dimensionality d by a learned linear transformation W_1. Additionally, x_i^b and x_i^c are projected into d by separate learned linear transformations W_2 and W_3. Layer normalization LN is applied to the three d dimensional vectors. x_i^{spec} is a result of element wise addition of these three vectors after layer normalization as shown below:

$$x_i^{spec} = LN(W_1(x_i^{fr}+x_i^{ft}+x_i^p))+LN(W_2x_i^b)+LN(W_3x_i^c) \tag{1}$$

3.3 Adopting M4C

We utilize the multimodal mesh copy module (M4C) introduced it Hu et al. in order to copy special tokens into generated text (Hu et al. 2020). This method has been directly adopted by many subsequent OCR Vision-Language papers for copying scene text into generated text. We generalize the input from OCR tokens to special tokens, but make no change to the copying mechanism. Here we formalize the the differences between our captioning model and the M4C Captioning model.

The input modalities into the M4C captioning model are object features $\{x_1^{obj}, ..., x_M^{obj}\}$ for M objects and OCR tokens $\{x_1^{ocr}, ..., x_N^{ocr}\}$ for N OCR tokens. We generalize OCR tokens to special tokens such that the inputs are $\{x_1^{obj}, ..., x_M^{obj}\}$ and $\{x_1^{spec}, ..., x_N^{spec}\}$ for N special tokens.

M4C captioner predicts fixed vocab scores $\{y_{1,t}^{voc}, ..., y_{K,t}^{voc}\}$ where K is a fixed vocab size and t is the decoding step and ocr vocab scores $\{y_{1,t}^{ocr}, ..., y_{N,t}^{ocr}\}$ where N is the number of ocr tokens. The selected word at each time step $w_t = argmax(y_t^{all})$ where $y_t^{all} = [y_t^{voc}; y_t^{ocr}]$. We substitute $y_t^{spec} = \{y_{1,t}^{spec}, ..., y_{N,t}^{spec}\}$, where N is the number of special tokens, for y_t^{ocr} such that $y_t^{all} = [y_t^{voc}; y_t^{spec}]$.

4 PAC Dataset

With this paper we release the Politicians and Athletes in Captions (PAC) dataset. PAC is image-caption dataset consisting of images well-known individuals in context. PAC has 1,572 images and three captions per image. Samples from PAC can be seen in Figure 4.

We create PAC with the goal of studying the use of non-generic vocabulary in image captioning. The non-generic terms emphasized in PAC are person names and OCR tokens. The PAC dataset offers the following technical challenges:

1. Correctly identifying people in a variety of settings.
2. Reasoning about the effect of the *presence* of the individual. If a known person is in a scene, the description of the scene often based on the known person.
3. Naturally integration of a name into a generated caption.

Recent work has shown the ability of pretrained vision-language models to adapt to new domains with limited samples (Tsimpoukelli et al. 2021). PAC can be used in the sense to test a pre-trained models ability to learn problems 2 and 3.

4.1 Collection

Images were collected from the Creative Commons image database which are made available under the CC licence. In the collection of the dataset, 62 public figures were searched for. We selected images from the first 100 returned, filtering out duplicate images and images without visible faces.

Annotators were instructed to provide a caption of the image including the name of the individual in which was searched for when collecting the image. Other famous individuals who happened to appear in the image may also be mentioned in the captions. Additionally, annotators were instructed to use scene-text if it improved the quality of the caption. These annotation instructions differ those for caption collection of previous datasets. For example, in the collection of MS-COCO captions, annotators were instructed

1. Manny Pacquiao in a AIBA world championship
2. Manny Pacquiao poses for a photo at an event.
3. Boxer Manny Pacquiao is posing for a photo.

1. Megan Rapinoe moves the ball down the field.
2. Megan Rapinoe trains hard for the next World Championship.
3.Megan Rapinoe plays soccer and wears the shirt number 15.

1. Paul Kagame on mic at the World Economic Forum.
2. Paul Kagame looks on intensely as he listens to a question in a press conference at the World Economic Forum.
3. Paul Kagame speaking at the World Economic Forum .

1. Kamala Harris is sworn in at the 2020 inauguration.
2. Kamala Harris wearing a purple coat with one hand raised.
3. Kamala Harris being sworn in and standing in front of the Capitol Building.

Figure 3: Samples from Politicians and Athletes in Captions

to *not* use proper nouns (Chen et al. 2015) and annotators for TextCaps were instructed to always use text in the scene (Sidorov et al. 2020). 658 images were captioned by college students and 914 were outsourced to Ground Truth. Captions were scanned for grammar and spelling errors.

4.2 Analysis

PAC includes images 1,572 images with 3 captions each. All images include at least one famous politician or athlete. Overlap exists in several images. 62 different individual are in the dataset for an average of 25.2 images per person. 23 of the individuals are athletes while 39 are athletes.

Each corresponding caption includes the name of at least one person name in the image. 66.1% of the data has scene text that is recognized by Google Cloud OCR. 35.9% of use scene text exactly as recognized by Google Cloud OCR in one of the captions. Comparatively, 96.9% of TextCaps images have OCR and 81.3% of captions use at least one OCR token. In 96.3% of the images a face RoI is detected by the RFB net, the face detector we use throughout this work. (Sidorov et al. 2020)

PAC captions have a lower average word count than other image-caption datasets at 8.35 words per caption. Comparatively, TextCaps has 12.4 words per caption, Conceptual Captions has 9.7, and MS-COCO has 10.5 (Sidorov et al. 2020). A number of images in PAC are close-up photos of individuals. Often, a precise caption for such an image contains few words (e.g 'Joe Biden wearing a suit') which explains the lower average. The names of individuals range between one and five words. Names account for 25.7% of the words in PAC captions.

4.3 Fairness

The fact that machine learning model learn bias from datasets has been well documented in the literature (Belkin et al. 2019). When the dataset includes information about people, discriminatory biases may be encoded in the data (Zliobaite 2015). A model can learn to correlate any characteristic in the data to an outcome which can result discriminatory model behavior. In PAC, the human characteristics in the dataset are visual appearance and name. In order to mitigate bias that may be encoded in the data, we considered the following criteria during collection: equitable distribution of races, equitable distribution of sexes, equal distribution of scenes and contexts across sexes and races.

While we have taken steps to limit encoded bias in the dataset, it does not mean it has been does not mean it does not exist. This dataset is only available for research use.

4.4 Limitations

We identify two primary limitations of the PAC dataset. The dataset with 1,610 images is small relative to other image-datasets. TextCaps has 28,408 images and MS-COCO has 330,000 images. Due to this PAC cannot represent the breadth of scenes that is found in other datasets. It is recommended to use PAC in conjunction with other dataset in order to mitigate this constraint.

The second primary limitation is narrow scene representation. The dataset is of famous athletes and politicians and therefore is biased towards scenes in which athletes and politicians are commonly photographed in. The captions also reflect this representation. For example, the word 'suit' is found in 1.82% of PAC captions while only 0.14% of TextCaps captions and 0.55% in MS-COCO. The word 'microphone' is found in 1.25% of PAC captions, 0.11% in TextCaps, and 0.05% in MS-COCO. Training on PAC combined with other datasets can mitigate this limitation while still allowing the model to learn to integrate person names as demonstrated in Section 5.

5 Experiments

We compare results on PAC with and without special tokens. We test several configurations of combining TextCaps and PAC for training and highlight the best results in Table 2.

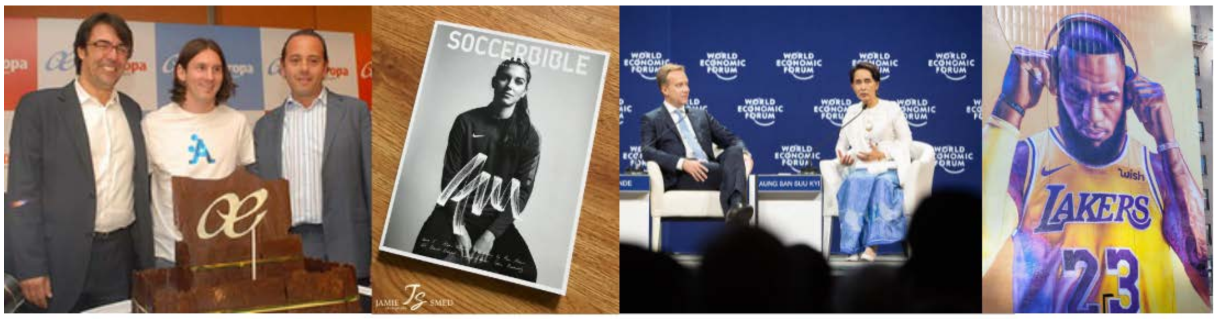
5.1 Implementation Details

We build our implementation on top of the MMF library (Singh et al. 2020). For detecting regions in the image with faces we use RFB net (Liu, Huang, and Wang 2018). For facial recognition we use ArcFace (Deng et al. 2019).

Table 1: Training on PAC with and without special tokens.

			PAC Test Set Metrics				
#	Model	Training	B-4	M	R	C	S
1	M4C	TextCaps→PAC	2.1	6.4	14.3	24.6	4.3
2	M4C+ST	TextCaps→PAC	14.5	22.4	42.0	156.8	30.3

ST: Special Tokens; B-4: BLEU-4; M: METEOR; R: ROUGUE; C: CIDEr, S: SPICE



M4C+ST: lionel messi poses with two other people.
M4C: a man and a woman are posing for a photo with the word " ropa " on it

M4C+ST: alex morgan standing on a book cover.
M4C: jamie photography on a book cover

M4C+ST: aung san suu kyi is at the world economic forum.
M4C: a photo of a man in front of a screen that says world world world world economic forum.

M4C+ST: lebron james wearing a purple jersey with the number 23
M4C: a close up a soccer player wearing a purple shirt with the word "lakers" on it

Figure 4: Captions generated for PAC test set images. Green words indicate tokens from the face recognition module and blue words indicate tokens from the OCR module. Corresponding metrics found in Table 1.

Using ArcFace we extract facial embedding for all individuals in the dataset. At inference, a face token if is extracted if the l_2 distance between the new embeddings and the pre-calculated embedding of a known individual is less than a threshold T. First names and last names become separate tokens with copied visual features. Face RoIs were found to be present 96.3% of images by RFB net. Approximately 80% of the predictions made by ArcFace on the the face RoIs were correct. To reduce noise in the training set, we manually update the face tokens to be present in all images and further assure by the face token is the correct name by referencing the name that was searched for to get the respective image. All reported quantitative and qualitative test set results use unmodified tokens generated from the aforementioned facial recognition models.

We use Google Cloud OCR for extracting OCR tokens and set a limit at $N = 50$ OCR tokens. Following previous work, we use a pretrained faster RCNN with a ResNet-101 backbone to propose RoIs and extract features for each region. A limit is let at $M = 100$ object features. PAC is broken up into the same 80-20 train-test split for all experiments.

5.2 Benchmarks

In Table 1 we compare results on PAC with vanilla M4C and M4C with special tokens (M4C+ST). Both models are trained on TextCaps for 12,000 iterations and subsequently on PAC for 1,300 iterations. M4C+ST sees 200-700% increases across metrics on the PAC test set. Without access to name tokens the vanilla M4C model has a small chance of using the correct name only if the name happens to be in model vocabulary. If the name is not in model vocabulary there is no chance. Corresponding qualitative examples are provided in Figure 4. In the samples, M4C+ST has appropriately used the face token in the caption. The right two images are samples where M4C+ST switched between model vocabulary, face tokens, and OCR tokens. This demonstrates the model has learned to differentiate separate special token types.

In Table 2, we report scores after training on several different combinations of PAC and TextCaps. The best captioning model is the one that performs well on PAC while still performing while on previous datasets. For this reason, scores are reported for PAC and TextCaps for all training combinations. The model trained on a ratio of TextCaps 8:1 PAC (Table 2 line 4) scores the highest in this regard.

6 Conclusion

Text generated by vision-language models often lacks specific terms that would be present in human level descriptions or answers. The special token approach can be used to introduce non-generic information to a vision-language model and consequently improve generated text. The special token approach accepts information from any number of upstream sources. The Politicians and Athletes in Captions dataset consists of image-caption pairs with well-known individuals. By using the special token approach and the PAC dataset, we train a model to integrate person names into

Table 2: PAC Baselines using Special Tokens and M4C Architecture. In the training column, a → between datasets indicates one dataset was trained on before the other where as a comma in between datasets indicates they were trained on simultaneously.

				Metrics				
#	Tokens	Training	Test	B-4	M	R	C	S
1 a.	Special	TextCaps	PAC	0.7	5.3	11.8	10.6	3.2
b.			TextCaps	22.9	22.1	46.0	89.7	15.3
2 a.	Special	PAC	PAC	14.9	22.3	41.9	156.7	30.5
b.			TextCaps	2.0	7.5	22.0	8.5	2.6
3 a.	Special	TextCaps→PAC	PAC	14.5	22.4	42.0	156.8	30.3
b.			TextCaps	20.7	20.1	43.0	80.4	13.4
4 a.	Special	PAC,TextCaps	PAC	13.6	22.3	42.1	157.2	104.5
b.			TextCaps	22.1	20.9	45.3	84.5	24.0
5 a.	Special	TextCaps→PAC,TextCaps	PAC	6.0	18.1	34.4	23.3	104.5
b.			TextCaps	23.2	22.0	46.2	91.0	15.1

B-4: BLEU-4; M: METEOR; R: ROUGUE; C: CIDEr, S: SPICE

text. This paper works towards vision-language models than generate human-like non-generic text, but comes far from solving the problem. Possible improvements to the proposed method include inclusion of more external sources, integrating open-domain knowledge with special tokens, or other architecture improvements.

Acknowledgement

The work reported in this paper is supported by the National Science Foundation under Grant No. 2050919. Any opinions, findings and conclusions or recommendations expressed in this work are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

Anderson, P.; He, X.; Buehler, C.; Teney, D.; Johnson, M.; Gould, S.; and Zhang, L. 2018. Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering. Technical report.

Bahdanau, D.; Cho, K.; and Bengio, Y. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015*.

Belkin, M.; Hsu, D.; Ma, S.; and Mandal, S. 2019. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences* 116(32):15849–15854.

Biten, A. F.; Gomez, L.; Rusinol, M.; and Karatzas, D. 2019a. Good news, everyone! context driven entity-aware captioning for news images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 12466–12475.

Biten, A. F.; Tito, R.; Mafla, A.; Gomez, L.; Rusinol, M.; Valveny, E.; Jawahar, C.; and Karatzas, D. 2019b. Scene text visual question answering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Chen, X.; Fang, H.; Lin, T.-Y.; Vedantam, R.; Gupta, S.; Dollar, P.; and Zitnick, C. L. 2015. Microsoft coco captions: Data collection and evaluation server. *arXiv e-prints* arXiv–1504.

Chen, Y.-C.; Li, L.; Yu, L.; El Kholy, A.; Ahmed, F.; Gan, Z.; Cheng, Y.; and Liu, J. 2020. Uniter: Universal image-text representation learning. In *European conference on computer vision*, 104–120. Springer.

Deng, J.; Guo, J.; Xue, N.; and Zafeiriou, S. 2019. Arcface: Additive angular margin loss for deep face recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Gao, C.; Zhu, Q.; Wang, P.; Li, H.; Liu, Y.; Hengel, A. v. d.; and Wu, Q. 2020a. Structured Multimodal Attentions for TextVQA.

Gao, D.; Li, K.; Wang, R.; Shan, S.; and Chen, X. 2020b. Multi-modal graph neural network for joint reasoning on vision and scene text. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 12746–12756.

Han, W.; Huang, H.; and Han, T. 2020. Finding the evidence: Localization-aware answer prediction for text visual question answering. In *Proceedings of the 28th International Conference on Computational Linguistics*, 3118–3131.

Hu, R.; Singh, A.; Darrell, T.; and Rohrbach, M. 2020. Iterative answer prediction with pointer-augmented multimodal transformers for textvqa. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 9992–10002.

Kant, Y.; Batra, D.; Anderson, P.; Schwing, A.; Parikh, D.; Lu, J.; and Agrawal, H. 2020. Spatially aware multimodal transformers for textvqa. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IX 16*, 715–732. Springer.

Karpathy, A., and Fei-Fei, L. 2015. Deep visual-semantic alignments for generating image descriptions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3128–3137. USA: IEEE Computer Society.

Karpathy, A.; Joulin, A.; and Fei-Fei, L. F. 2014. Deep fragment embeddings for bidirectional image sentence mapping. In Ghahramani, Z.; Welling, M.; Cortes, C.; Lawrence, N. D.; and Weinberger, K. Q., eds., *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc. 1889–1897.

Kemelmacher-Shlizerman, I.; Seitz, S. M.; Miller, D.; and Brossard, E. 2016. The megaface benchmark: 1 million faces for recognition at scale. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Kiros, R.; Salakhutdinov, R.; and Zemel, R. 2014. Multimodal neural language models. In Xing, E. P., and Jebara, T., eds., *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, 595–603. Bejing, China: PMLR.

Li, X.; Yin, X.; Li, C.; Zhang, P.; Hu, X.; Zhang, L.; Wang, L.; Hu, H.; Dong, L.; Wei, F.; et al. 2020. Oscar: Object-semantics aligned pre-training for vision-language tasks. In *European Conference on Computer Vision*, 121–137. Springer.

Lin, T.-Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollar, P.; and Zitnick, C. L. 2014. Microsoft coco: Common objects in context. In Fleet, D.; Pajdla, T.; Schiele, B.; and Tuytelaars, T., eds., *Computer Vision – ECCV 2014*, 740–755. Cham: Springer International Publishing.

Liu, F.; Xu, G.; Wu, Q.; Du, Q.; Jia, W.; and Tan, M. 2020. Cascade reasoning network for text-based visual question answering. In *Proceedings of the 28th ACM International Conference on Multimedia*, 4060–4069.

Liu, S.; Huang, D.; and Wang, a. 2018. Receptive field block net for accurate and fast object detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Lu, D.; Whitehead, S.; Huang, L.; Ji, H.; and Chang, S.-F. 2018. Entity-aware image caption generation. *arXiv preprint arXiv:1804.07889*.

Lu, J.; Batra, D.; Parikh, D.; and Lee, S. 2019a. Vilbert: pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 13–23.

Lu, J.; Batra, D.; Parikh, D.; and Lee, S. 2019b. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc. 13–23.

Lu, J.; Goswami, V.; Rohrbach, M.; Parikh, D.; and Lee, S. 2020. 12-in-1: Multi-task vision and language representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Mafla, A.; Dey, S.; Biten, A. F.; Gomez, L.; and Karatzas, D. 2021. Multi-modal reasoning graph for scene-text based fine-grained image classification and retrieval. In *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 4022–4032. IEEE.

Morris, M. R.; Johnson, J.; Bennett, C. L.; and Cutrell, E. 2018. Rich Representations of Visual Content for Screen Reader Users. *CHI conference on human factors in computing systems.*

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In Cortes, C.; Lawrence, N. D.; Lee, D. D.; Sugiyama, M.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc. 91–99.

Sharma, P.; Ding, N.; Goodman, S.; and Soricut, R. 2018. Conceptual captions: A cleaned, hypernymed, image alt-text dataset for automatic image captioning. In *Proceedings of ACL*.

Sidorov, O.; Hu, R.; Rohrbach, M.; and Singh, A. 2020. TextCaps: A Dataset for Image Captioning with Reading Comprehension. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 12347 LNCS, 742–758. Springer Science and Business Media Deutschland GmbH.

Singh, A.; Goswami, V.; Natarajan, V.; Jiang, Y.; Chen, X.; Shah, M.; Rohrbach, M.; Batra, D.; and Parikh, D. 2020. Mmf: A multimodal framework for vision and language research. https://github.com/facebookresearch/mmf.

Singh, A.; Pang, G.; Toh, M.; Huang, J.; Galuba, W.; and Hassner, T. 2021. Textocr: Towards large-scale end-to-end reasoning for arbitrary-shaped scene text. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 8802–8812.

Su, W.; Zhu, X.; Cao, Y.; Li, B.; Lu, L.; Wei, F.; and Dai, J. 2019. Vl-bert: Pre-training of generic visual-linguistic representations. In *International Conference on Learning Representations*.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In Ghahramani, Z.; Welling, M.; Cortes, C.; Lawrence, N. D.; and Weinberger, K. Q., eds., *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc. 3104–3112.

Tan, H., and Bansal, M. 2019. Lxmert: Learning cross-modality encoder representations from transformers. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 5100–5111.

Tran, A.; Mathews, A.; and Xie, L. 2020. Transform and tell: Entity-aware news image captioning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 13035–13045.

Tsimpoukelli, M.; Menick, J.; Cabi, S.; Eslami, S.; Vinyals, O.; and Hill, F. 2021. Multimodal few-shot learning with frozen language models. *arXiv preprint arXiv:2106.13884*.

Vinyals, O.; Toshev, A.; Bengio, S.; and Erhan, D. 2015. Show and tell: A neural image caption generator. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3156–3164.

Wang, J.; Tang, J.; Yang, M.; Bai, X.; and Luo, J. 2021. Improving ocr-based image captioning by incorporating geometrical relationship. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 1306–1315.

Xu, K.; Ba, J.; Kiros, R.; Cho, K.; Courville, A.; Salakhudinov, R.; Zemel, R.; and Bengio, Y. 2015. Show, attend and tell: Neural image caption generation with visual attention. In Bach, F., and Blei, D., eds., *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, 2048–2057. Lille, France: PMLR.

Young, P.; Lai, A.; Hodosh, M.; and Hockenmaier, J. 2014. From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions. *TACL* 2:67–78.

Zhao, S.; Sharma, P.; Levinboim, T.; and Soricut, R. 2019. Informative image captioning with external sources of information. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 6485–6494.

Zhu, Q.; Gao, C.; Wang, P.; and Wu, Q. 2021. Simple is not easy: A simple strong baseline for textvqa and textcaps. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 3608–3615.

Zliobaite, I. 2015. A survey on measuring indirect discrimination in machine learning. *arXiv e-prints* arXiv–1511.

Exploring Class Ordering Heuristics for Incremental Learning

Séraphin Bassas
Department of Computer Science
McGill University
845 Rue Sherbrooke O
Montréal, QC H3A0G4
seraphin.bassas@mail.mcgill.ca

Jugal Kalita
Department of Computer Science
University of Colorado at Colorado Springs
1420 Austin Bluffs Pkwy
Colorado Springs, CO 80918
jkalita@uccs.edu

Abstract

Incremental Learning research is principally focused on overcoming the challenge of "catastrophic forgetting" to increase model accuracy. However, in this endeavor of increasing model accuracy, one factor that is almost consistently overlooked is the way training data is ordered with respect to their classes. Although the entirety of a dataset is always available to the authors before training, random or seeded orderings are typically used, but this is problematic when trying to reproduce studies and peak accuracy results. We bring to the table a method to sort classes deterministically such that the order in which they are presented to the IL algorithm will, on average, consistently yield greater mean accuracy than random orderings do. Thus, through this novel ordering heuristic, we provide the computer vision community with a tool to perform more easily reproduced and benchmarked results, as well as a trick to boost model accuracy by as much as 5%.

1. Introduction

When *AlexNet* (Krizhevsky, Sutskever, and Hinton 2012) emerged in 2012, the Computer Vision, NLP, and Neural Machine Translation communities, to name a few, shifted gears. As mentioned in (Can and Ezen-Can 2020), while before scientists used to spend their time crafting features to extract information from images, documents, and other data types, the bursting of deep neural networks into the limelight has shifted the paradigm towards tuning the networks' many parameters. As a direct consequence, substantial progress has been made in engineering more efficient models and parameters since then.

Nevertheless, one of the "parameters" that has been underwhelmingly addressed in the literature is the tuning of the order in which data is being fed to the networks to maximize accuracy. In the literature, the relationship between the ordering of data, classes, or tasks and network accuracy is poorly qualified and/or quantified. Despite inadequate understanding of this relationship, the notion that the order of the data may have an effect on classification is not new. In fact, (MacGregor 1988; Giraud-Carrier 2000; Wenzel and Hotz 2010) dabbled with these concepts in the late 80s, early 2000s, and early 2010's respectively. While the problems these authors tried to solve were very different, (Wenzel and Hotz 2010) provides a formal definition of the notion of *ordering sensitivity* which varies across classification algorithms. What's more, the author establishes a measure to estimate this sensitivity which is computed as the variance of a network's accuracy when trained across randomly picked data orderings.

While descendants of *AlexNet* now excel at the ILSVRC competition, more challenging problems such as Incremental Learning (IL) are being addressed. IL experiments often use the same Imagenet dataset (Deng et al. 2009) that *AlexNet* won ILSVRC-2012 on. In IL, a network is trained on an initial set of classes, and at each incremental step thereafter, the model keeps getting retrained on data from different classes without previous data available. As such, one can see how data or class orderings are particularly relevant to the IL problem.

With that in mind, we turn the attention towards Wenzel and Hotz (2010), who brought unto the computer vision community the intuition that similar classes should be introduced in proximity to each other or according to some measure of relatedness. This idea is inspired from the way humans learn in school for instance, where the incremental learning of new concepts is timed precisely through carefully designed curricula. However, the opposite intuition can also be defended wherein classifiers will more easily classify classes that are further apart in feature space. For instance, linear classifiers such as support vector machines (Drucker et al. 1997) have more ease drawing an accurate hyperplane between two classes that are further apart in feature space.

Regardless of which approach is the "correct" one, the proposed research aims to investigate how both these ordering strategies can be fitted to various heuristics for class ordering applied to the incremental image classification setting.

To summarize, we seek to address the gap in the literature regarding the effects of class orderings on model accuracy in the IL setting. Our contributions are the following:

1. Creating diverse class ordering heuristics inspired from various learning philosophies,
2. Finding a heuristic that consistently yields higher model accuracies independent of model, dataset, or hyperparameters,
3. Starting a discussion about the way data ordering is used in comparison studies and state-of-the-art papers in IL and

how it should be revisited.

2. Related Works

2.1 Class Incremental Learning

Some notable contributions to incremental learning include early work by Li and Hoiem (2017) with their Learning without Forgetting (LwF) regularization based method. Their strategy is to combine the distillation loss from distillation networks and regularization to update model weights for conserved old task accuracy while tuning for new task accuracy. The Incremental Classifier and Representation Learning (iCaRL) model (Rebuffi et al. 2017), builds on LwF to update its feature representation, but the authors also augment the training dataset with exemplars, or example images from previous tasks that represent their class seemingly well during training. Further, they adopt an exemplar management strategy to cope with the increasing memory requirements of exemplar storage and a method termed *nearest-mean-of-exemplars* to overcome LwF's tendency to favor new class data at the classifier level. Liu et al. (2020) supplement the feature representation and exemplar storage updating strategy by framing the optimization of model weights and exemplar selection as a bi-level optimization problem. Finally, (Hou et al. 2019) address the imbalances in dataset size per class when using exemplars that decrease old class classification accuracy. They do so by rebalancing the output probabilities of the softmax layer via cosine normalization to once again reduce bias towards new classes during classification.

2.2 Task Ordering

Although not necessarily representative of the literature on this subject, a recent survey on continual learning (Delange et al. 2021) suggests that task ordering has no effect on accuracy for two datasets, one balanced (Tiny Imagenet) [1] and another unbalanced (iNaturalist) (Van Horn et al. 2018). In other words, they change around the order in which each increment is learned rather than changing the order of the classes within each increment, or let alone the ordering of the data within training batches. And, they claim to have found no effect based on changing the tasks from an order of easiest to hardest and hardest to easiest. However, they did point out the drop in accuracy surrounding training on classes with disproportionately small training sets regardless of order, thus suggesting a relationship between accuracy and class types. Further, Nguyen et al. (2019) find that for task ordering, there is variability of results based on different datasets and models. However, they show that when they factor out possible confounding variables, the heterogeneity of the sequence of tasks in incremental learning is negatively correlated with error rate, thus suggesting that increasingly different tasks will yield higher network accuracies.

2.3 Data Ordering

Can and Ezen-Can (2020) show that the ordering of data within training batches has a significant impact on network accuracy regardless of learning rate, batch size, or model. The authors provide statistical analyses to demonstrate the significance in accuracy disparities across different data orderings on Imagenet, but do not provide any insight into generating favorable orderings.

[1] https://www.kaggle.com/c/tiny-imagenet

2.4 Class Ordering

An early approach before the aforementioned "deep learning boom" to order classes for incremental classification was based on finding the distance between classes. They used Baye's error as a measure of distance and then ordered the classes on the basis of which ordering minimized that distance. The most relevant work to our proposed approach is that of Masana, Twardowski, and Van de Weijer (2020) in which they investigate the effect of class orderings on the accuracies of state-of-the-art IL models. They obtain a fitness score for various class orderings which is computed by taking the trace of the matrix multiplication of a weight matrix by the confusion matrix obtained after model training and testing. In their approach, each weight matrix represents a different heuristic for class ordering, and the authors use a simulated annealing algorithm to generate class orderings that will maximize a fitness function for a given heuristic. That being said, the authors did not find a heuristic that consistently outperformed random class orderings across models. Occasionally, mention of class orderings can be spotted in IL papers (Rebuffi et al. 2017; Castro et al. 2018). Rebuffi et al. (2017) the authors use a shuffled version of the data that is seeded so that it's always the same, and Castro et al. (2018) emphasize the fact that for fair comparison of IL model accuracies in experiments, the same ordering of data needs to be maintained on different models.

3. Approach

Our overarching goal is to find a heuristic for class ordering that consistently yields greater or equivalent average classification accuracy relative to random class orderings. We tested three classes of heuristics, each with two versions engineered towards the opposing learning strategies mentioned above:

1. Introducing most visually similar classes will yield best results
2. Introducing most visually distinct classes will yield best results.

In total, we have six different heuristics, each with their own minor variations depending on distance measurements and image type (greyscale vs. RGB).

3.1 Distance Learning

The first heuristic that we call *distance learning* (DL), is a greedy heuristic. DL computes the exemplar for each class, or the pixel wise average of all the images belonging to the same class. Subsequently, the distances separating each exemplar from one another are computed and stored in an adjacency matrix. The particular distance measure depends on

Algorithm 1: Min_Distance_Learning

```
   Inputs: dataset $\mathcal{D}$ with set of classes $\mathcal{C} = \{C_1, C_2, ..., C_n\}$
           $m$ = number of classes per increment
1  $exemplars$←Compute_Exemplars($\mathcal{C}$)
2  for $i, j$ = 0 to $n$ do
3      $distance\_adjacency\_matrix[i][j]$← euclidean_distance($exemplars[i]$,$exemplars[j]$)
4  end
5  $first\_class\_index$←one of the two indices for the min value entry in $distance\_adjacency\_matrix$
6  $class\_order[0]$←$first\_class\_index$
7  while $class\_order.size$ != $n$ do
8      append to $class\_order$ $class_\theta$ s.t. $class_\theta$ not in $class\_order$ & $class_\theta$ nearest to $class\_order[-1]$
9  end
10 while $class\_order$ not empty do
11     $increments[i]$←remove next $m$ classes from $class\_order$
12 end
13 return $increments$
```

Algorithm 2: Compute_Exemplars

```
   Inputs: $\mathcal{C} = \{C_1, C_2, ..., C_n\}$
1  for each $C_i$ in $\mathcal{C}$ do
2      $exemplars[i]$←pixel-wise mean of each image in $C_i$
3  end
4  return $exemplars$
```

the number of channels in the input images: if the images are greyscale, the pixel-wise euclidean distance is used and we leave it at that. However, if the image has RGB channels, two scenarios exist. We first tested the *flat* method, wherein we take the mean of the three channels at each pixel for each image in the given class, and then perform regular euclidean distance between class exemplars. This is more or less what is depicted by Algorithm 2. On the other hand, in the second measure tested, termed *layered*, exemplars have three channels, where each value is the channel-wise mean at that pixel for each image in the given class. Moreover, instead of the traditional euclidean distance, we take the sum of the euclidean distances computed with respect to each channel as the pixel-wise distance between exemplars.

Finally, the ordering of the classes is then determined based on the preferred learning strategy. For the first strategy, similarity, the first class is randomly picked among the two classes that have the minimum distance in the adjacency matrix, as depicted in Algorithm 1. Then, until all classes have been picked, we search in the adjacency matrix for the class closest to the last one that was picked. For the second strategy, conceptual distinction, we proceed identically but interchanging minimum with maximum distance.

In the Results section pertaining to DL, all datasets reference the following six class ordering strategies: random, semantic, flat-max, layer-max, flat-min, and layer-min. In the random learning strategy, we use the *rand.sample()* Python method to randomly generate an ordering of integers from 0 to the number of classes in the dataset. We define the semantic ordering as the order given by the class labels, i.e. images labeled 3 and 4 or 56 and 57 in the CIFAR100 dataset are introduced to the model consecutively. Initially, since we conducted our first experiments on the MNIST digit dataset and the labels correspond to the digit being introduced, the label ordering represents a sort of semantic ordering. However this did not translate to the other datasets, and other possible semantic orderings will be discussed in the Future Works section. The *flat* and *layer* orderings represent whether the distance between classes is computed as the average of RGB pixel-wise values (flat) or the sum of channel-wise and then pixel-wise values (layer). And then the min and max keywords represent whether the ordering of classes is based on the minimum or maximum distance separating class exemplars, which represents the two intuitive learning strategies mentioned earlier.

3.2 Cluster Learning

The second and third heuristics we present are less conventional–although one could argue convention isn't so much the problem in such a sparse field–as we introduce heuristics that create tasks with variable amounts of classes per increment. Similarly to the first heuristic, the second and third heuristics also compute the exemplars for each class, which are then passed through a feature extraction routine to obtain their vectorized feature representations. Subsequently, the feature vectors of the exemplars are clustered, in our case using mini-batch kmeans (Sculley 2010), and the centroids of each cluster are evaluated. Afterwards, the cluster-wise distances are computed and stored in an adjacency matrix.

The second heuristic, termed *distance clustering*, is similar to DL but with clusters. We define the initial set of classes for training as those found in one of the two clusters with the greatest distance separating them, found as the maximum entry in the adjacency matrix. The classes in the cluster that was not picked as the initial set then become the set of classes comprising the second increment. Finally, we keep referring to the adjacency matrix to find the next closest cluster to the last one picked in the ordering, and define the following increment as the set of classes composing that cluster.

In the third heuristic, termed *scattered* (or scattered distance clustering), we also compute the maximum distances separating the centroids of each cluster, but we don't define the clusters themselves as the classes in a single cluster. Instead, the first increment is composed of one randomly picked class from each cluster. Therefore, the first increment contains as many classes as there are clusters defined in the algorithm. The second increment contains randomly picked classes from all non empty clusters left. We repeat the process until all clusters are empty. It is worth noting that the number of classes contained in consecutive increments either stays constant or decreases using this heuristic.

Algorithm 3: Min_Distance_Clustering

```
Inputs: dataset D with set of classes $\mathcal{C}$ = $\{C_1, C_2, ..., C_n\}$
        $k$=number of clusters
        boolean $scattered$
1  $exemplars$←Compute_Exemplars($\mathcal{C}$)
2  $feature\_vectors$←retrieve feature vector for each
     exemplar from model's penultimate layer output
3  $predictions$←output predictions of clustering
     algorithm on $feature\_vectors$ with $k$ clusters
4  $exemplars\_per\_cluster$←array containing at each
     index, all the exemplars $\in i^{th}$ cluster
5  $classes\_per\_cluster$←array containing, at each
     index, all the classes $\in i^{th}$ cluster
6  $centroids$←array containing all the centroids
     computed for each cluster
7  for $i, j$ = $0$ to $k$ do
8      $distance\_adjacency\_matrix[i][j] \leftarrow$
         euclidean_distance($centroids[i]$, $centroids[j]$)
9  end
10 $first\_cluster\_index \leftarrow$ one of the two indices for
     the min distance entry in
     $distance\_adjacency\_matrix$
11 $cluster\_order[0] \leftarrow first\_cluster\_index$
12 while $cluster\_order.size$ != $k$ do
13     append $cluster_\theta$ to $cluster\_order$, where
         $cluster_\theta$ not in $cluster\_order$ & $cluster_\theta$
         nearest to $cluster\_order[-1]$
14 end
15 if $scattered$ then
16     while $classes\_per\_cluster$ not empty do
17         $increments[i] \leftarrow$ remove one class from
             each non empty cluster in
             $classes\_per\_cluster$
18     end
19 else
20     $increments[i] \leftarrow i^{th}$ cluster in $cluster\_order$
```

4. Implementation

4.1 Datasets

Testing was performed on common baseline datasets for easy comparison to previous work and as well as future comparison in this field. Further, we chose datasets of varying scales to put the scalability of our technique to the test. We chose two greyscale image datasets, the MNIST digit (LeCun) and MNIST fashion datasets (Xiao, Rasul, and Vollgraf 2017), and two RGB image datasets, CIFAR10 and CIFAR100 (Krizhevsky, Hinton, and others 2009). All datasets have 10 classes, except for CIFAR100 which has 100.

4.2 Models

We experimented with three main models: one for digit MNIST, another for fashion MNIST, and finally a VGG16 model (Simonyan and Zisserman 2014) for CIFAR10&CIFAR100. The first model for MNIST is a simple neural network taken from the Keras "Simple MNIST convnet"[2] and we used the Adam (Kingma and Ba 2014) optimizer with categorical cross-entropy loss. For the fashion MNIST model, we use a model with 3 convolutional layers, each with 256 units, Relu activation, 3x3 kernels, and "HeUniform" kernel initializers (He et al. 2015). These layers are followed by a max-pooling layer, a flattening layer, a dense layer with 75 units, and a softmax layer. This model has the same optimizer and loss function as the one above.

The VGG16 model is loaded from the Keras package with Imagenet weights and followed up by a global max-pooling layer, a dropout layer, and a dense layer with 2048 units, and a softmax classifier. For all results presented in this paper, we train the models for 50 epochs at each increment without freezing any weights (for VGG16). We also used an EarlyStopping callback with a patience of 15 and minimum delta of 0.001.

4.3 Data

Data was loaded from the keras API and ordered by label for easier downstream manipulation prior to and during training and testing. To generate class orderings, we treat all classes from the dataset prior to the network ever seeing the data, such that once the classes are sorted in accordance with a heuristic, no further manipulation of that data is needed before training the network. At each increment, we make data from previous classes available for training. We justify this decision by mentioning that we can use this data as an initial baseline for comparison of how class orderings affect identical incremental learning mechanisms while allowing us to test the robustness of different combinations of parameters. Moreover, we hope to make the incremental learning mechanism itself a variable down the road for further experimentation.

5. Results

In figures 1 and 2, we present experimental data from both CIFAR10 and CIFAR100. For both datasets, each data point represents values averaged out over 4-6 replicate experiments. In the case of CIFAR10, each increment, including the first one, saw the addition of one extra class at each time point. For CIFAR100, the model was trained on 10 initial classes, and each increment saw the addition of 5 new classes. Different combinations of numbers of classes introduced at the first and all subsequent increments were experimented with, and yielded similar results.

In Figure 1, the blue bars show the average accuracy over all increments for all the distance heuristics presented in this paper. We see that for CIFAR10 (Figure 2a), the accuracy of the random heuristic is inferior only to the semantic and flat-max heuristics by 2.43% and 3.25% respectively. Similarly, for CIFAR100 (Figure 2b) the random heuristic has a lower accuracy than the semantic, flat-max, and layer-max heuristics by 0.05%, 5.24%, and 3.44% respectively. Thus we conclude that flat-max is the only heuristic so far that consistently gives the network higher accuracy than the random heuristic. Further, from data on the two MNIST datasets and

[2]https://keras.io/examples/vision/mnist_convnet/

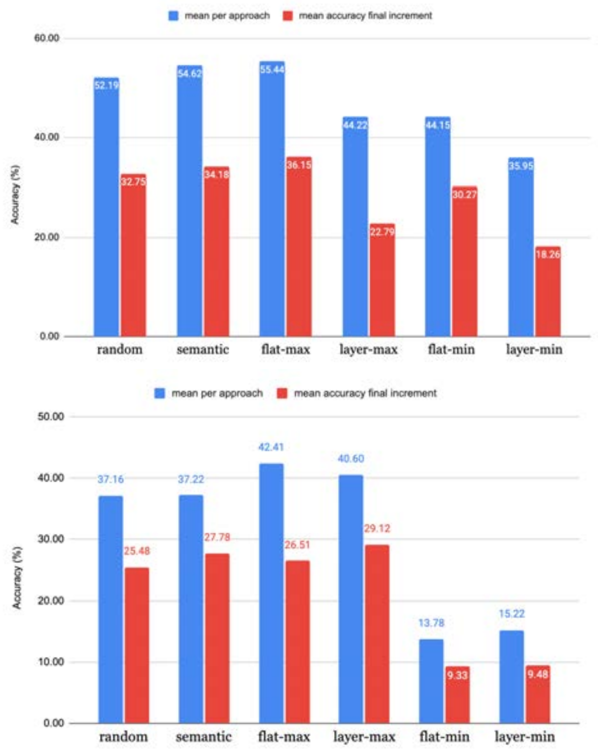


Figure 1: Comparison of mean accuracies over all increments and mean accuracies at the final increment obtained using different ordering strategies for CIFAR10 (top) and CIFAR100 (bottom).

the two CIFAR datasets, we observe that the more complex the problem becomes (number of channels per image, number of images per class, number of classes, etc.) the greater the margin between the random heuristic and the flat-max heuristic grows.

In Figure 2, the plots show the accuracy of the six heuristics at each incremental step of the IL experiment. Firstly, one of the more drastic observations to make is by how much the minimum distance heuristics under-perform in comparison to the other heuristics, and, moreover, the consistency of this drop in accuracy from the very first increments. Secondly, the inconsistency of the accuracies in figure 2b, even for flat-max, shows that there is still room for much improvement when it comes to designing better heuristics, as those initial increments are the ones that provide the largest increases to the per approach mean accuracy.

6. Discussion

In this paper, we qualify and quantify the relationship between the ordering of classes and the accuracy of IL algorithms while finding a heuristic that lets us obtain accuracy boosting class orderings that generalize across datasets, models, and hyperparameters. In the results section above, we train the aforementioned models using an IL mechanism described as *joint training* in (Li and Hoiem 2017). Thus, it is important to raise the issue that the effects of these ordering heuristics may not translate equivalently to actual IL algorithms where data from previous classes is scarce or completely absent. Nevertheless, as *joint training* is often used as an “upper bound” on the performance that IL algorithms can potentially reach, it is quite promising to show that our heuristic works on this baseline.

On a different note, the accuracy trends of the heuristics tested on CIFAR100 reveal volatile behaviors, even for our most efficient heuristic. This alone serves as motivation to seek improved heuristics that will yield even greater boosts in accuracy by smoothing the curve, and also begs the question about the scalability of ordering heuristics as only the min-distance heuristics had relatively constant accuracies throughout. Finally, our findings put into question the way data orderings have traditionally been taken for granted as negligible factors in the experimental IL setup. We want to bring attention to this problem by showing that there does indeed exist ordering heuristics that grant an edge over others. Thefefore, we would like to invite the community to revisit previous works where ordering was overlooked in comparison studies and even to boost peak accuracy.

7. Future Work

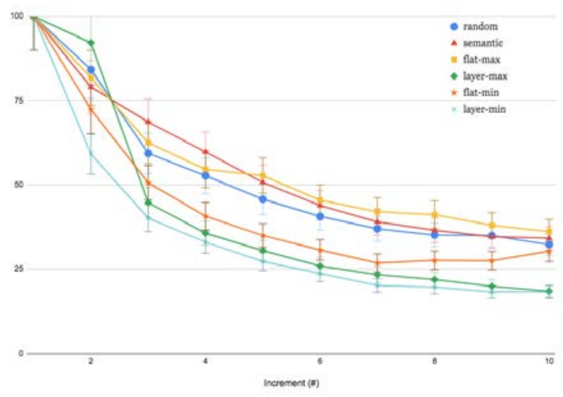
First and foremost, our future works includes a statistical analysis to investigate the significance of the accuracy boosts and decreases provided by the ordering heuristics. Likewise, as mentioned in an earlier section, our study still lacks rigor in comparing the effect of data orderings for various incremental learning strategies as carried out in (Masana, Twardowski, and Van de Weijer 2020). We seek to test the effect of all the heuristics presented in this paper on the following three algorithms: Lwf by (Li and Hoiem 2017), iCaRL by (Rebuffi et al. 2017), and LUCIR by (Hou et al. 2019). Additionally, we would like to test the scalability of the flat-max heuristics and others on heavier duty data sets like tiny Imagenet and Imagenet1000. Incidentally, these experiments would also give us the opportunity to investigate if any heuristics based on the semantics of image datasets such as the inherited hierarchical relationships from the WordNet dataset, have any meaningful impact on IL algorithm accuracy. Finally, following our finding that minimum distance heuristics actually significantly reduce model accuracy, we would like to investigate how class ordering may be used as an alternate strategy to carry out adversarial attacks.

8. Acknowledgements

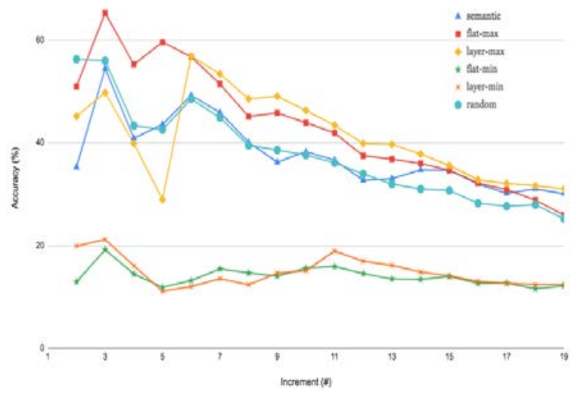
The work reported in this paper is supported by the National Science Foundation under Grant No. 2050919. Any opinions, findings and conclusions or recommendations expressed in this work are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

Can, E. F., and Ezen-Can, A. 2020. The effect of data ordering in image classification. *arXiv preprint*



(a) CIFAR10



(b) CIFAR100

Figure 2: Comparing mean accuracies obtained at each increment for the different class ordering strategies.

arXiv:2001.05857.

Castro, F. M.; Marín-Jiménez, M. J.; Guil, N.; Schmid, C.; and Alahari, K. 2018. End-to-end incremental learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 233–248.

Delange, M.; Aljundi, R.; Masana, M.; Parisot, S.; Jia, X.; Leonardis, A.; Slabaugh, G.; and Tuytelaars, T. 2021. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 248–255. IEEE.

Drucker, H.; Burges, C. J.; Kaufman, L.; Smola, A.; Vapnik, V.; et al. 1997. Support vector regression machines. *Advances in Neural Information Processing Systems* 9:155–161.

Giraud-Carrier, C. 2000. A note on the utility of incremental learning. *Ai Communications* 13(4):215–223.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, 1026–1034.

Hou, S.; Pan, X.; Loy, C. C.; Wang, Z.; and Lin, D. 2019. Learning a unified classifier incrementally via rebalancing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 831–839.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krizhevsky, A.; Hinton, G.; et al. 2009. Learning multiple layers of features from tiny images.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems* 25:1097–1105.

LeCun, Y. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

Li, Z., and Hoiem, D. 2017. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40(12):2935–2947.

Liu, Y.; Su, Y.; Liu, A.-A.; Schiele, B.; and Sun, Q. 2020. Mnemonics training: Multi-class incremental learning without forgetting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 12245–12254.

MacGregor, J. N. 1988. The effects of order on learning classifications by example: heuristics for finding the optimal order. *Artificial Intelligence* 34(3):361–370.

Masana, M.; Twardowski, B.; and Van de Weijer, J. 2020. On class orderings for incremental learning. *arXiv preprint arXiv:2007.02145*.

Nguyen, C. V.; Achille, A.; Lam, M.; Hassner, T.; Mahadevan, V.; and Soatto, S. 2019. Toward understanding catastrophic forgetting in continual learning. *arXiv preprint arXiv:1908.01091*.

Rebuffi, S.-A.; Kolesnikov, A.; Sperl, G.; and Lampert, C. H. 2017. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Sculley, D. 2010. Web-scale k-means clustering. In *Proceedings of the 19th International Conference on World Wide Web*, 1177–1178.

Simonyan, K., and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Van Horn, G.; Mac Aodha, O.; Song, Y.; Cui, Y.; Sun, C.; Shepard, A.; Adam, H.; Perona, P.; and Belongie, S. 2018. The inaturalist species classification and detection dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 8769–8778.

Wenzel, S., and Hotz, L. 2010. The role of sequences for incremental learning. In *ICAART (1)*, 434–439.

Xiao, H.; Rasul, K.; and Vollgraf, R. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

HiCARN: Super Enhancement of Hi-C Contact Maps

Parker Hicks
Concordia University Irvine
1530 Concordia
Irvine, CA 92612
parker.hicks@eagles.cui.edu

Oluwatosin Oluwadare
University of Colorado Colorado Springs
1420 Austin Bluffs Pkwy
Colorado Springs, CO 80918
ooluwada@uccs.edu

Abstract

High throughput chromosome conformation capture (Hi-C) contact matrices are used to predict three-dimensional (3D) chromatin structures in eukaryotic cells. High resolution Hi-C data are less available than low resolution Hi-C data due to sequencing costs, but provides greater insight into the intricate details of 3D chromatin structures such as enhancer-promoter interactions and sub-domains. To provide a cost effective solution to high resolution Hi-C data collection, deep learning models are used to predict high resolution Hi-C matrices from existing low resolution matrices across multiple cell types. We developed a Hi-C Cascading Residual Network (HiCARN) that outperforms state-of-the-art Hi-C resolution enhancement models in predictive accuracy.

Introduction

Chromosome 3D conformation structures are important to consider when exploring genomic processes within eukaryotic cell nuclei. Hi-C is a biochemical technique that supports an all versus all mapping of the interaction of the fragments in a chromosome and a genome. This interaction between the pair read assays are further converted to an *nxn* interaction frequency (IF) contact matrix, where *n* is the number fragments in a chromosome or genome at a given Hi-C data resolution (Lieberman-Aiden et al. 2009). Today, these data are used as the input to many algorithms for advanced understanding of the genome organization (Oluwadare, Highsmith, and Cheng 2019).

However, a major problem in understanding the genome organization is the lack of high resolution (HR) data necessary for understanding inherent topologies in the human genome such as enhancer-promoter interactions or sub-domains (Zhang et al. 2018), which are only discoverable at high resolutions such as $\leq$ 10kb. Thus, the critical need in the chromatin genomics field is the development of a cost effective method to increase the availability of HR Hi-C data for advanced study and an in-depth elucidation of the genome organization.

Deep learning models are used to fill this demand by predicting the high resolution data from low resolutions (LR) with great accuracy. Current models include HiCPlus (Zhang et al. 2018), HiCNN (Liu and Wang 2019a), hicGAN (Liu, Lv, and Jiang 2019), Boost-HiC (Carron et al. 2019), HiCSR (Dimmick, Lee, and Frey 2020), SRHiC (Li and Dai 2020), HiCNN2 (Liu and Wang 2019b), VEHiCLE (Highsmith and Cheng 2021), and DeepHic (Hong et al. 2020). Currently, each of the Hi-C enhancement models have their various strengths, but performance can still be improved. Hence, the ultimate objective of this work was to develop an efficient Hi-C enhancement algorithm that would potentially outperform existing methods for enhancements, using a cascading residual network (CARN).

A CARN was developed for image super resolution (Ahn, Kang, and Sohn 2018) and was shown to have a computational cost of 10 times less than the super-resolution convolutional neural network model (Li et al. 2019). The CARN also outperforms previous super resolution models in peak-to-noise-ratio (PSNR) and structural similarity index measure (SSIM) (Ahn, Kang, and Sohn 2018). Here we present the application of a Hi-C CARN (HiCARN) to enhance the resolution of LR Hi-C data and outperform state-of-the-art Hi-C super resolution models.

Related Work

Previous models are categorized into three groups based on their respective network architectures: convolutional neural networks (CNNs), autoencoders, and generative adversarial networks (GANs).

CNN-Based

The first model used for Hi-C resolution enhancement was HiCPlus (Zhang et al. 2018) which used a CNN to identify patterns of IFs from neighboring reference regions to generate HR Hi-C data from LR inputs. HiCNN improved the accuracy of Hi-C resolution enhancement with a network composed of 54 layers that consistently outperformed HiCPlus (Liu and Wang 2019a). This was shortly followed by HiCNN2 where three models were generated using a combination of one, two, or three CNNs in HiCNN2-1, HiCNN2-2, and HiCNN2-3 respectively (Liu and Wang 2019b).

Autoencoder-Based

HiCSR is another notable model that uses a denoising autoencoder consisting of five convolutional layers preceding five deconvolutional layers (Dimmick, Lee, and Frey 2020).

HiCSR outperformed VEHiCLE, another autoencoder based model in overall GenomeDISCO, HiCRep, and QuASAR-Rep scores in four tested chromosomes (Highsmith and Cheng 2021).

GAN-Based

Alternative network architectures besides CNNs have also been utilized for Hi-C resolution enhancement. A GAN was used in the hicGAN model where a generator and discriminator were implemented to produce super resolution Hi-C data and discriminate against real HR data and the super resolution data (Liu, Lv, and Jiang 2019).

The current overall best performing model, DeepHiC, is also a GAN. DeepHiC outperformed HiCPlus and HiCNN in SSIM score and Pearson Correlation (Hong et al. 2020), while also outperforming VEHiCLE and HiCSR overall in the previously cited structural similarity scores (Highsmith and Cheng 2021).

Methods

Architecture

The following two architectures were compared to choose the optimal architecture for the performance of HiCARN: a CARN generator (HiCARN-1)(Figure 1) and a GAN with a CARN generator and a discriminator (HiCARN-2)(Figure 2).

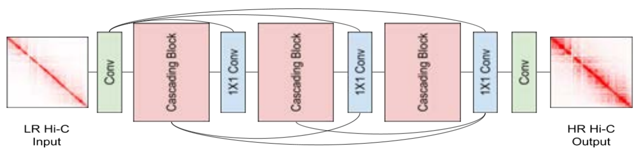


Figure 1: Cascading residual network architecture with local and global cascading layers modified for application of Hi-C data.

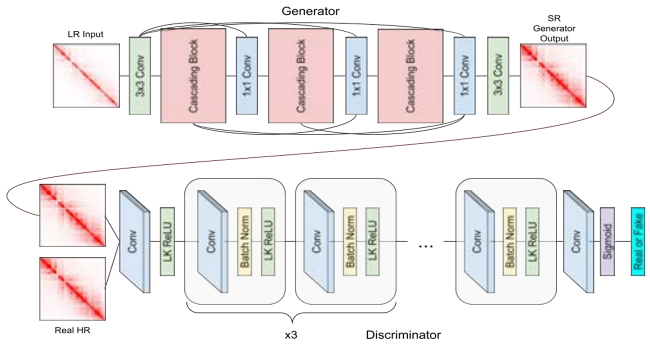


Figure 2: HiCARN-2 GAN architecture with a cascading block generator and discriminator. LR images are passed through the generator where predicted SR images are created. The predicted SR images then are passed through the discriminator along with the real HR images where the discriminator attempts to classify them as real or fake.

HiCARN-1 and HiCARN-2's generator retain a similar architecture to CNNs, except each cascading block contains two residual network (ResNet) blocks with a 1x1 convolutional layer (Conv) between both ResNet blocks. Each ResNet block contains two 3x3 Convs and two ReLU activation functions with local cascading connections (Figure 3). Intermediate outputs from each block cascade into the concatenation function of the next block and parameters are shared between cascading blocks.

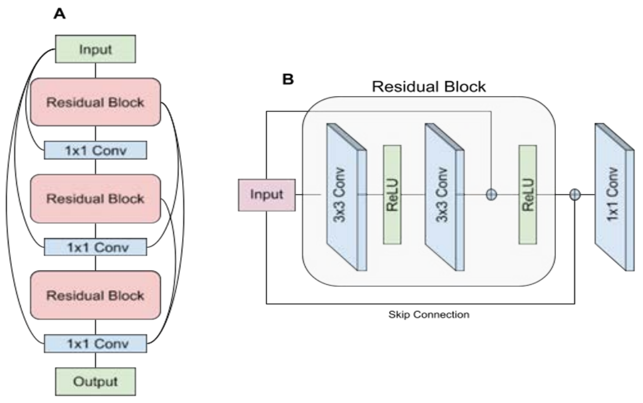


Figure 3: Overview of the cascading block architecture. (A) Cascading block architecture with local and skip connections. Features extracted from previous layers are propagated through the end of the cascading block via concatenation. The features are then condensed into a single channel as the output. (B) Each residual block follows standard ResNet architecture, however an additional skip connection from the input to the block output is added before the convolution.

HiCARN's overall generator network shares the same connection properties as a single cascading block (Figure 1) which function to maintain and reintroduce features from multiple layers. This not only contributes to the performance of HiCARN, but the efficiency as well since the multi-level connections act as forward and backward propagation shortcuts (Ahn, Kang, and Sohn 2018), thus allowing for quick training and accurate predictions. Similarly to ResNet, the residual and cascading blocks of HiCARN use many skip connections as well as ReLU activation functions to solve the vanishing gradient problem.

The discriminator of HiCARN-2 consists of a series of seven 3x3 Convs, leaky ReLU's, and batch normalizations proceeded by a 3x3 Conv and leaky ReLU and followed by a 3x3 Conv, sigmoid activation, and average pooling (Figure 3). There are no global or local cascading connections between blocks and layers.

Loss Functions

HiCARN-1 utilizes mean squared error (MSE), perceptual loss from the pretrained VGG16 CNN (VGG), and total variation (TV) loss, while HiCARN-2 adds an additional adversarial (AD) loss from the discriminator as the generator loss function; and the binary entropy (BCE) loss function for the discriminator.

Generator loss for HiCARN-1 and HiCARN-2 are respectively defined by the following equations[1, 2] where α, β, and γ are scalar weights ranging from 0-1:

$$L_G = l_{MSE} + \alpha(l_{VGG}) + \beta(l_{TV}) \tag{1}$$

$$L_G = l_{MSE} + \alpha(l_{VGG}) + \beta(l_{TV}) + \gamma(l_{AD}) \tag{2}$$

In L_G, both l_{MSE} and l_{VGG} compute MSE loss. MSE measures the cross entropy of the distributions of the generator SR output and the real HR image by computing the average squared difference between the two images[3], where y is a real HR 40x40 submatrix and $\hat{y}$ is the predicted 40x40 submatrix.

$$MSE(\hat{y}, y) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \tag{3}$$

AD loss of the HiCARN discriminator represents the probability of discriminator classification error of the generated fake SR images and real HR images[4].

$$l_{AD} = 1 - \frac{(\sum_i \hat{y}_i)}{N} \tag{4}$$

TV loss functions to remove noise within the generated SR image. Total generator TV loss is defined by the following function[5] where ψ is a weight scalar, F is the number of filters in a tensor of dimensions [F, C, H, W], and h_{TV}[6] and w_{TV}[7] are the TV losses of H and W respectively:

$$l_{TV} = \frac{2\psi * (h_{TV} + w_{TV})}{F} \tag{5}$$

Height and width TV loss are calculated by the sum of the squared difference of the generator output matrix y divided by the respective dimensions of h_{TV} and w_{TV}. For h_{TV}, $\hat{y}_{(2:i)j}$ is the generator output with the first row removed and $\hat{y}_{(1:i-1)j}$ is the same output with the last row removed. A similar computation is repeated for w_{TV} where the first and last columns are removed.

$$h_{TV} = \frac{\sum(\hat{y}_{(2:i)j} - \hat{y}_{(1:i-1)j})^2}{C * (H-1) * W} \tag{6}$$

$$w_{TV} = \frac{\sum(\hat{y}_{i(2:j)} - \hat{y}_{i(1:j-1)})^2}{C * H * (W-1)} \tag{7}$$

The discriminator of HiCARN-2 utilizes the BCE loss function[8] to penalize the discriminator for misclassifying fake SR images from real HR images. Total discriminator loss is computed as the sum of the BCE losses for the classification of fake SR images and real HR images[9].

$$H_p(q) = -\frac{1}{N}\sum_{i-1}^{N} y_i * log(p(y_i) + (1-y_i) * log(1-p(y_i)) \tag{8}$$

$$L_D = H_p(q)_{real} + H_p(q)_{fake} \tag{9}$$

Data

Hi-C data was collected from the Restructured Gene Expression Omnibus (ReGEO) database. The training dataset used was obtained from the GEO GSE63525 human GM12878 cell line, the most common training data across all Hi-C resolution enhancement models. From this cell line, chromosomes 1, 3, 5, 7, 8, 9, 11, 13, 15, 17, 18, 19, 21, and 22 were used for training and chromosomes 2, 6, 10, and 12 were used for validation. The datasets used to test the HiCARN model were GM12878 chromosomes 4, 14, 16, and 20; the human K562 cell line; and the CH12-LX mouse embryonic stem cell (mESC) line. The ESC data was used to test the model's accuracy across species. For each chromosome, the whole matrix was divided into 40x40 submatrices using a window of 40 and stride of 40 with no overlap of submatrices.

Baseline Model Implementations

All baseline models were trained on our generated datasets for 1/16, 1/32, 1/64, and 1/100 downsampled inputs. Python source code for DeepHiC, from which we used their code for data preprocessing and network architecture, was obtained from https://github.com/omegahh/DeepHiC. Source code for HiCSR, HiCNN, and HiC-Plus were obtained from https://github.com/PSI-Lab/HiCSR, http://dna.cs.miami.edu/HiCNN2/, and https://github.com/wangjuan001/hicplus respectively.

HiCSR, HiCNN, and HiCPlus all take 40x40 inputs and decrease the dimensions to output 28x28 matrices. For these models to produce 40x40 outputs, during training and testing LR inputs were padded with zeros to dimension 52x52.

For HiCNN, we attempted to train HiCNN2-1 per the SGD optimizer parameters as cited (Liu and Wang 2019b), however the model continuously produced poor results even with varied parameters. HiCNN2-3 produced the best results when compared to HiCNN2-2. Hence, we do not report results for HiCNN2-1 and HiCNN2-2.

Evaluation and Validation

The model was trained and tested using the data previously cited. A leave-P-out cross validation method was used for training and validation. Chromosomes 2, 6, 10, and 12 from the GM12878 human cell line represent P and were tested at the end of each training epoch to calculate the best SSIM score over time.

During testing, Pearson Correlation Coefficient (PCC), Spearman Correlation Coefficient (SPC), Mean Squared Error (MSE), SSIM, and PSNR scores were calculated for each 40x40 submatrix predicted by HiCARN, DeepHiC, HiCSR, HiCNN, and HiCPlus. Novel Hi-C analysis metrics such as GenomeDISCO (Ursu et al. 2018) and HiCRep (Yang et al. 2017) were used to calculate the reproducibility of the generated 40x40 submatrices. The models were tested across four randomly selected chromosomes to compare the efficiency and accuracy of HiCARN to state-of-the-art models. To ensure generalizability of HiCARN, the human K562 and mESC cell lines were not seen by the model until training was complete.

Image Evaluation Metrics

The following equations define PCC, SCC, SSIM, and PSNR where y denotes the real HR target and $\hat{y}$ represents the enhanced LR input. For MSE see equation[3].

PCC calculates the correlation coefficient *r* of two matrices along the matrix diagonal.

$$PCC_r(\hat{y}, y) = \frac{\sum(\hat{y}_i - \bar{\hat{y}})(y_i - \bar{y})}{\sqrt{\sum(\hat{y}_i - \bar{\hat{y}})^2 \sum(y_i - \bar{y})^2}} \tag{10}$$

SCC is also calculated along the matrix diagonal and computes the strength and direction of the monotonic relationship between the two matrices ρ, whereas PCC is the linear relationship strength. Here, d represents the difference between two observation rankings and *n* is the number of observations.

$$SCC_\rho(\hat{y}, y) = 1 - \frac{6\sum d_i^2}{n(n^2-1)} \tag{11}$$

SSIM computes the similarity of two given images. We used DeepHiC's implementation of SSIM scoring (Hong et al. 2020). The function compares contrast, structure, and luminance across the two images via a moving convolution window, extracting the values μ_y and $\mu_{\hat{y}}$. The values σ_y and $\sigma_{\hat{y}}$ are computed by moving a convolution window across y^2 and $\hat{y}^2$ and subtracting their respective μ values. The constants C_1 and C_2 were set to 0.01^2 and 0.03^2 respectively.

$$SSIM(\hat{y}, y) = \frac{(2\mu_{\hat{y}}\mu_y + C_1)(2\sigma_{\hat{y}y} + C_2)}{(\mu_{\hat{y}}^2 + \mu_y^2 + C_1)(\sigma_{\hat{y}}^2 + \sigma_y^2 + C_2)} \tag{12}$$

PSNR measures the ratio of the maximum signal power to the power of corrupting noise in the image.

$$PSNR(\hat{y}, y) = 10 * log_{10}\left(\frac{N}{MSE_{(\hat{y},y)}}\right) \tag{13}$$

Hi-C Reproducibility Metrics

GenomeDISCO and HiCRep provide a more biologically significant analysis measure compared to standard image evaluation metrics. GenomeDISCO uses a random walk of *t* steps to denoise Hi-C contact matrices from which a difference vector is computed. The concordance score is calculated by subtracting the difference vector from 1 in the range [-1, 1], where larger values indicate increased similarity. We used the optimal step value *t*=3 as cited (Ursu et al. 2018).

Similarly, HiCRep denoises the contact matrices prior to analysis. A Pearson correlation coefficient is calculated for each stratum. Coefficients are then combined together via a weighted average producing a stratum adjusted correlation coefficient. Scores are in the range [-1, 1]. We used the R implementation of this method.

Results

HiCARN-1&2 were trained on 40x40 submatrices in 200 epochs using the Adam optimizer with a batch size of 64 and an initial learning rate of $1.0\text{x}10^{-3}$. A variable learning rate was used and is defined by the following equation where lr_n and E_n are the current learning rate and epoch respectively:

$$lr_n = lr_{n=1} * (0.1^{\lfloor E_n/30 \rfloor}) \tag{14}$$

The variable learning rate allows for increased learning in early epochs after-which it stabilizes. HiCARN learns and converges very quickly for SSIM scores during training (Figure 4).

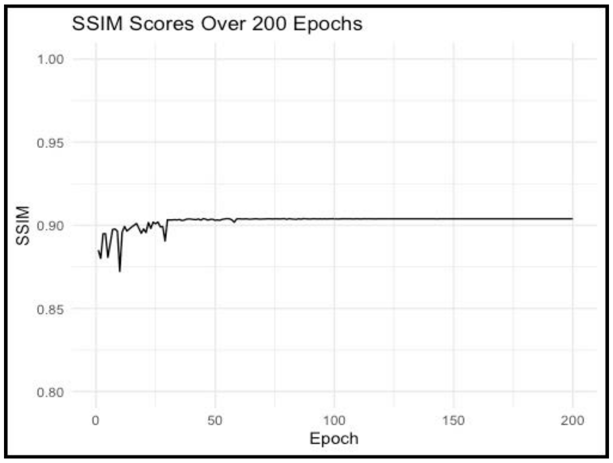


Figure 4: SSIM scores captured throughout 200 epochs of training for HiCARN-2.

Differences among varied cascading blocks quantities

We trained HiCARN with varying numbers of cascading blocks. Using 3 cascading blocks proved sufficient for outperforming state-of-the-art models, however when more blocks are added, almost all evaluation metric scores increase, albeit slightly. We tested the performance of 3, 4, 5, 6 and 10 cascading block networks. HiCARN with 10 cascading blocks outperformed all other architectures at the cost of training and predicting speed. A network of 5 cascading blocks was chosen to balance the added accuracy of higher block counts and reduced computational cost of lower block counts.

HiCARN frequently outperforms baseline models in image and Hi-C evaluation metrics

A visual comparison of predicted contact maps is provided for chromosomes 4 and 14 from GM12878 (Figure 5) and chromosomes 11 and 19 from K562 (Figure 7). HiCARN-1 and HiCARN-2 produce nearly identical contact maps which is confirmed by the evaluation metrics (Figure 6). Within the GM12878 cell line, each of the models reconstructs a contact map fairly comparable to the HR target.

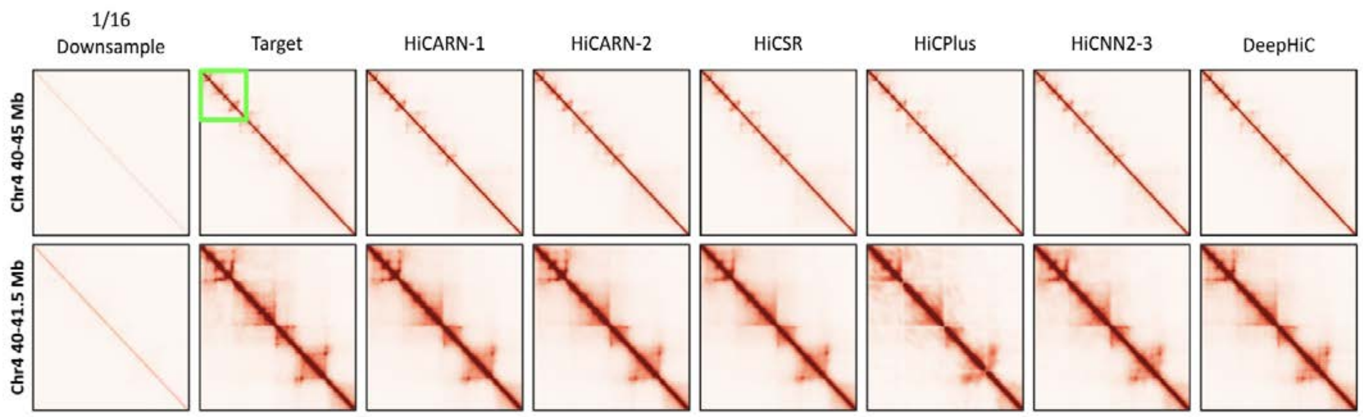


Figure 5: Heat map diagram of GM12878 chromosome 4 predictions from HiCARN and baseline models. The top row displays the 40-45Mb region and the bottom is a zoomed in view from 40-41.5Mb. The green box in the target heat map indicates zoomed in area.

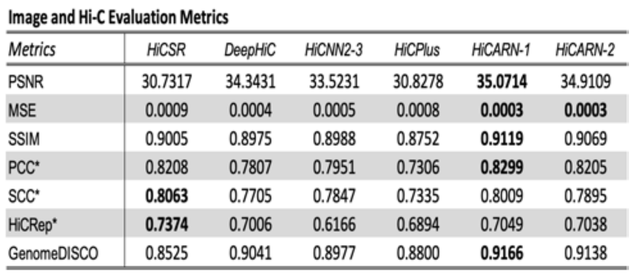


Image and Hi-C Evaluation Metrics

Metrics	*HiCSR*	*DeepHiC*	*HiCNN2-3*	*HiCPlus*	*HiCARN-1*	*HiCARN-2*
PSNR	30.7317	34.3431	33.5231	30.8278	**35.0714**	34.9109
MSE	0.0009	0.0004	0.0005	0.0008	**0.0003**	**0.0003**
SSIM	0.9005	0.8975	0.8988	0.8752	**0.9119**	0.9069
PCC*	0.8208	0.7807	0.7951	0.7306	**0.8299**	0.8205
SCC*	**0.8063**	0.7705	0.7847	0.7335	0.8009	0.7895
HiCRep*	**0.7374**	0.7006	0.6166	0.6894	0.7049	0.7038
GenomeDISCO	0.8525	0.9041	0.8977	0.8800	**0.9166**	0.9138

Figure 6: Image and Hi-C evaluation results of HiCARN and baseline models for the GM12878 cell line. Bolded values identify the top score. SSIM, PSNR, MSE, and GenomeDISCO scores were calculated from predicted 40x40 submatrices during testing. All scores are displayed as averages across chromosomes 4, 14, 16, and 20. PCC, SCC, and HiCRep scores were calculated for the entire matrix, indicated by *.

Overall, HiCARN-1 produces the top results for PSNR, SSIM, MSE, PCC, and GenomeDISCO. HiCARN-2 scores are quite close with very minimal difference. HiCSR achieved top scores in SCC and HiCRep.

HiCARN performance across unseen cell lines

HiCARN and baseline models were tested on chromosomes 3, 11, 19, and 21 from the K562 human cell line. Overall, HiCARN-1 outperforms all other models with HiCARN-2 following close behind. DeepHiC and HiCNN also produced comparable results in reproducibility scores and image reconstruction.

HiCARN also conditionally outperforms all baseline models in all evaluation metrics when predicting across species. If 4 cascading blocks are used for prediction, then performance is high among baseline models. A network of 10 cascading blocks performs similarly. However, if our standard model of 5 cascading blocks is used, then accuracy drops significantly for Hi-C reproducibility metrics.

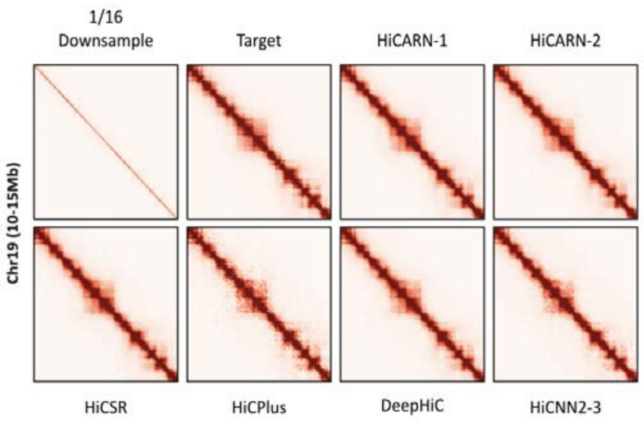


Figure 7: Heat map diagram of K562 chromosome 19 predictions from HiCARN and baseline models from the 10-15Mb region.

3D chromatin reconstruction

We reconstructed 3D chromatin models using the 3DMax structure prediction tool (Oluwadare, Zhang, and Cheng 2018). Models were generated for the 1/16 downsampled matrix, the real HR target, and HiCARN's prediction for chromosome 4 (40-45Mb) from the GM12878 cell line (Figure 8). HiCARN is able to produce 3D conformations similar to that of the real HR contact maps with a spearman correlation of 0.8310 between the two.

Conclusion

In this work, we presented a novel framework for HR Hi-C contact map predictions. Variations in the number of cascading blocks and the overall framework (CNN vs GAN) do not significantly hinder or improve the performance of HiCARN. However, if the user requires a quick training process, HiCARN-1 with 3-5 cascading blocks should be used to reduce computational load during training and predicting.

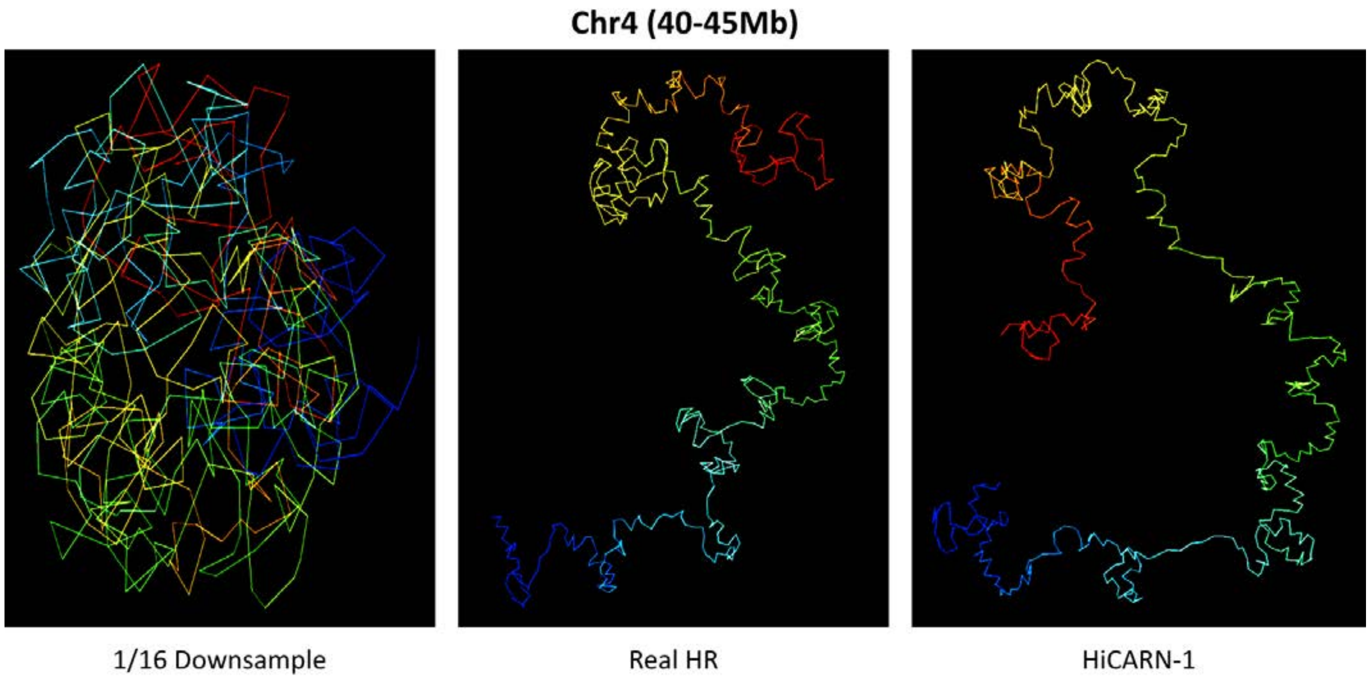


Figure 8: 3D reconstruction of the 1/16 downsampled (left), real high resolution (center) and HiCARN-1's generated (right) contact matrices. The region presented is from chromosome 4 (40-45Mb) from the GM12878 cell line. Each colored section corresponds to a 10kb reference region from the human hg19 reference genome.

We also demonstrated HiCARN's superior performance over all baseline models in both image and Hi-C evaluation metrics. Baseline models performed well, however our network outperformed these models across all tested cell lines.

HiCARN contributes further to the development of high fidelity predictions of HR Hi-C contact maps from state-of-the-art resolution enhancement models.

Availability and Implementation

HiCARN can be accessed and utilized as an open-sourced software at: https://github.com/OluwadareLab/HiCARN.

Acknowledgements

The work reported in this paper is supported by the National Science Foundation under Grant No. 2050919. Any opinions, findings and conclusions or recommendations expressed in this work are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

Ahn, N.; Kang, B.; and Sohn, K.-A. 2018. Fast, accurate, and lightweight super-resolution with cascading residual network. 252–268.

Carron, L.; Morlot, J.; Matthys, V.; Lesne, A.; and Mozziconacci, J. 2019. Boost-hic: Computational enhancement of long-range contacts in chromosomal contact maps. *Bioinformatics* 35(16):2724–2729.

Dimmick, M. C.; Lee, L. J.; and Frey, B. J. 2020. Hicsr: a hi-c super-resolution framework for producing highly realistic contact maps. *bioRxiv*.

Highsmith, M., and Cheng, J. 2021. Vehicle: a variationally encoded hi-c loss enhancement algorithm for improving and generating hi-c data. *Scientific Reports* 11(1):1–13.

Hong, H.; Jiang, S.; Li, H.; Du, G.; Sun, Y.; Tao, H.; Quan, C.; Zhao, C.; Li, R.; Li, W.; et al. 2020. Deephic: A generative adversarial network for enhancing hi-c data resolution. *PLoS computational biology* 16(2):e1007287.

Li, Z., and Dai, Z. 2020. Srhic: a deep learning model to enhance the resolution of hi-c data. *Frontiers in genetics* 11:353.

Li, X.; Wu, Y.; Zhang, W.; Wang, R.; and Hou, F. 2019. Deep learning methods in real-time image super-resolution: a survey. *Journal of Real-Time Image Processing* 1–25.

Lieberman-Aiden, E.; Van Berkum, N. L.; Williams, L.; Imakaev, M.; Ragoczy, T.; Telling, A.; Amit, I.; Lajoie, B. R.; Sabo, P. J.; Dorschner, M. O.; et al. 2009. Comprehensive mapping of long-range interactions reveals folding principles of the human genome. *science* 326(5950):289–293.

Liu, T., and Wang, Z. 2019a. Hicnn: a very deep convolutional neural network to better enhance the resolution of hi-c data. *Bioinformatics* 35(21):4222–4228.

Liu, T., and Wang, Z. 2019b. Hicnn2: Enhancing the resolution of hi-c data using an ensemble of convolutional neural networks. *Genes* 10(11):862.

Liu, Q.; Lv, H.; and Jiang, R. 2019. hicgan infers super resolution hi-c data with generative adversarial networks. *Bioinformatics* 35(14):i99–i107.

Oluwadare, O.; Highsmith, M.; and Cheng, J. 2019. An overview of methods for reconstructing 3-d chromosome and genome structures from hi-c data. *Biological procedures online* 21(1):1–20.

Oluwadare, O.; Zhang, Y.; and Cheng, J. 2018. A maximum likelihood algorithm for reconstructing 3d structures of human chromosomes from chromosomal contact data. *BMC genomics* 19(1):1–17.

Ursu, O.; Boley, N.; Taranova, M.; Wang, Y. R.; Yardimci, G. G.; Stafford Noble, W.; and Kundaje, A. 2018. Genomedisco: a concordance score for chromosome conformation capture experiments using random walks on contact map graphs. *Bioinformatics* 34(16):2701–2707.

Yang, T.; Zhang, F.; Yardımcı, G. G.; Song, F.; Hardison, R. C.; Noble, W. S.; Yue, F.; and Li, Q. 2017. Hicrep: assessing the reproducibility of hi-c data using a stratum-adjusted correlation coefficient. *Genome research* 27(11):1939–1949.

Zhang, Y.; An, L.; Xu, J.; Zhang, B.; Zheng, W. J.; Hu, M.; Tang, J.; and Yue, F. 2018. Enhancing hi-c data resolution with deep convolutional neural network hicplus. *Nature communications* 9(1):1–9.

Appendix

Equations

$$a1 = Conv(ReLU(Conv(x))) \tag{15}$$

$$b1 = ReLU(a1 + x) \tag{16}$$

$$c1 = b1 + x \tag{17}$$

EnsembleSplice: Ensemble Deep Learning for Splice Site Prediction

Trevor Martin[1], **Oluwatosin Oluwadare**[2]
[1]Department of Computer Science, Oberlin College
[2]Department of Computer Science, University of Colorado, Colorado Springs
tmartin2@oberlin.edu, ooluwada@uccs.edu

Abstract

Identifying splice site (SS) regions is an important step in genomic DNA sequencing pipelines for biomedical and pharmaceutical research. Within this research purview, efficient and accurate SS detection is highly desirable, and a variety of computational models have been developed towards this end. In particular, neural network (NN) architectures have recently been shown to outperform classical machine learning (ML) approaches for the task of SS prediction. Despite these advances, there is still considerable potential for improvement, especially in terms of model accuracy and inter-species generalizability. Bearing these issues in mind, EnsembleSplice is a deep learning (DL) model that incorporates the hitherto unseen method of ensemble learning for splice site prediction. When evaluated on genomic DNA datasets for the species *Homo sapiens* and *Arabidopsis thaliana*, EnsembleSplice outperformed existing state-of-the-art SS detection models, attaining average accuracies of 96.02% for donor SS and 94.59% for acceptor SS.

Introduction

Organismal genomes are studied primarily through genome annotation, which involves classifying genomic elements based on their function or location (Abril and Castellano Hereza 2019). This annotation is typically performed at the nucleotide-level to determine the locations of key genetic elements in DNA sequences, at the protein-level to evaluate proteomic function, or at the process-level to study the mechanisms underlying gene interaction (de Sá et al. 2018).

Genes responsible for protein coding are composed of alternating nucleotide regions called introns, which are the non-protein coding regions, and exons, which are the protein coding regions. During DNA transcription in eukaryotic cells, introns are cut out by spliceosomes and exons are combined together; this general process is called RNA splicing, and is critical for the creation of mature mRNA from pre-mRNA and for protein synthesis (Pohl et al. 2013). The dinucleotides AG and GT are often present in the $3'$ intron boundary, or donor splice site (DoSS) region, and the $5'$ intron boundary, or acceptor splice site (AcSS) region, respectively, and are biological markers involved in RNA splicing (Pertea, Lin, and Salzberg 2001) (see Figure 1). Nucleotide-level annotation was designed to accurately detect the location of these splice sites, which can be used to identify genes in eukaryotic genomes; a variety of other computational approaches have also been developed for this purpose.

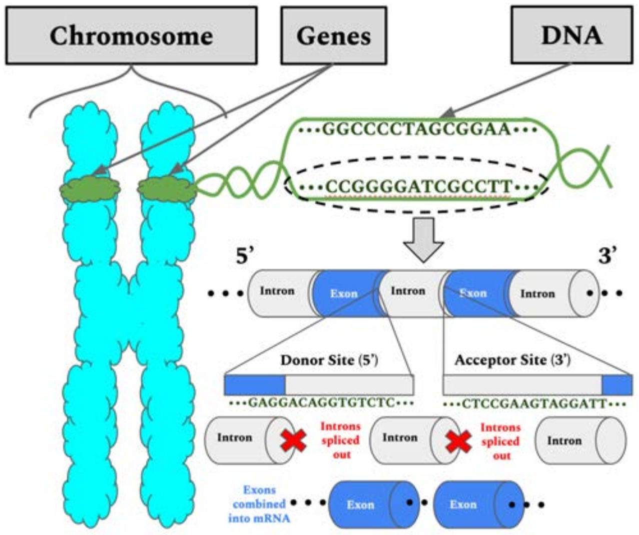


Figure 1: Illustration detailing the process of splicing.

EnsembleSplice is one such computational method, and is a deep learning pipeline that employs ensemble learning for splice site prediction. Ensemble learning methods have been shown to enhance classification results, and have, in recent years, been successfully applied within the field of bioinformatics (Sagi and Rokach 2018; Cao et al. 2020).

We contribute the following to research on splice site prediction:

- We develop EnsembleSplice, a DL architecture that learns from an ensemble of convolutional neural network (CNN) and dense neural network (DNN) architectures to achieve state-of-the-art performance at predicting splice sites.
- We evaluate the performance of EnsembleSplice across three datasets and two organisms.
- We create a usage tutorial, detail all architectural design choices, and, for reproducibility, make the code available at https://github.com/tmartin2/EnsembleSplice.

Methodology

Datasets

Each dataset used in this paper consists of both confirmed true (positive) DoSS/AcSS and confirmed false (negative) AcSS/DoSS. Evaluation of classification performance is separated by splice site type, which means that one model is trained to distinguish between false/true DoSS regions and another is trained to distinguish between false/true AcSS regions. It is important to note that EnsembleSplice is tested on both imbalanced datasets (HS^{3}D) and balanced ones (*Homo sapiens* and *Arabidopsis thaliana*). See Table 1.

HS^{3}D. The Homo Sapiens Splice Sites Dataset (HS^{3}D) consists of human genomic DNA introns and exons extracted from the Primate Division of GenBank Rel.123 (Pollastro and Rampone 2002). There are $2,796$ confirmed positive DoSS regions, $2,880$ confirmed positive AcSS regions, $271,937$ confirmed negative DoSS regions, and $329,374$ confirmed negative AcSS regions. This paper randomly selects 10000 false DoSS regions and 10000 false AcSS regions from the $271,937$ and $329,374$ available in the dataset, respectively; the Python code `random.seed(123454)` is used to shuffle the entire HS^{3}D dataset before the false DoSS and false AcSS subsets are selected. The nucleotide consensus AG for AcSS regions occurs at positions 69 and 70, and the nucleotide consensus GT for DoSS regions occurs at positions 71 and 72. The HS^{3}D dataset can be accessed at http://www.sci.unisannio.it/docenti/rampone/.

***Homo sapiens* and *Arabidopsis thaliana*.** The *Homo sapiens* and *Arabidopsis thaliana* datasets consist of splice site regions selected from annotated genomic DNA sequences for *Homo sapiens* and *Arabidopsis thaliana* in Ensembl (Zerbino et al. 2018). The peripheral nucleotide sequences padding each AcSS or DoSS were determined via Bedtools (Albaradei et al. 2020; Quinlan and Hall 2010). Each splice site region in the datasets is 602 nucleotides long; each DoSS region has consensus GT at positions 301 and 302, and each AcSS has consensus AG also at positions 301 and 302. In the *Homo sapiens* dataset, there are $250,400$ confirmed positive and negative DoSS regions, and $248,150$ confirmed positive and negative AcSS regions. The *Arabidopsis thaliana* dataset includes $110,314$ confirmed positive and negative DoSS regions, and $112,336$ confirmed positive and negative AcSS regions. The confirmed negative AcSS and DoSS regions were selected from chromosomes 21, 2, $2L$, 1, and I. Again, this paper randomly selects 8000 true/false DoSS regions and 8000 true/false AcSS regions from both datasets. As with the HS^{3}D dataset, the Python code `random.seed(123454)` is used for shuffling the *Homo sapiens* and *Arabidopsis thaliana* datasets before the DoSS and AcSS subsets are selected. The *Homo sapiens* and *Arabidopsis thaliana* datasets can be accessed at https://github.com/SomayahAlbaradei/Splice_Deep.

EnsembleSplice Pipeline

We now propose EnsembleSplice, a DL architecture that consists of an ensemble of three CNN and three DNN sub-models, for the task of splice site detection. See Figure 2 for the full architecture.

The sub-models in EnsembleSplice generate predictions for whether genomic DNA input sequences are positive DoSS or negative DoSS, or if the AcSS model is being used, for whether the sequences are positive AcSS or negative AcSS. These binary predictions are then aggregated (stacked) into a new dataset, where each sub-model's predictions become a column vector, and this dataset is then fed into a Logistic Regression classifier, which produces the final predictions for the inputted sequences.

Consider a family

$$\mathcal{D} = \{S_0, S_1, \ldots, S_n\}$$

of nucleotide splice site regions. We have the ordered set

$$S_i = \{x_1, x_2, \ldots, x_{|S_i|}\}$$

where S_i is the i-th nucleotide splice site region, and

$$x_j \in X = \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}, 0 \leqslant j \leqslant |S_i|$$

For all $0 \leqslant i \leqslant n$, S_i is encoded as a $|S_i| \times |X|$ binary matrix through one-hot encoding. These encoded sequences are fed to the three CNNs and three DNNs.

Each CNN sub-model in EnsembleSplice is composed of three convolutional layers and a dropout layer. The convolutional layers automatically extract local and global features from the AcSS or DoSS input sequences. In particular, these layers form complex representations of the sequences, and are the components of the CNN that allow it to accurately discriminate between the true/false acceptor/donor sites. The first layer has 72 convolutional filters and a kernel size of 5, the second layer has 144 convolutional filters and a kernel size of 7, and the third layer has 168 convolutional filters and a kernel size of 7. Each convolutional layer employs the $ReLU$ activation function as its final component; this removes noisy or otherwise irrelevant features, thus improving feature selection (Hahnloser et al. 2000; Krizhevsky and Hinton 2010). Additionally, each convoluational layer has a stride size of 1. Next, a dropout layer

Table 1: Dataset Metrics

Splice Site	Dataset	Sequence Count	Pos:Neg Ratio
Acceptor (AcSS)	HS^{3}D	2,880 (true), 10000 (false)	1:3.472
	Homo Sapiens	8000 (true), 8000 (false)	1:1
	Arabidopsis thaliana	8000 (true), 8000 (false)	1:1
Donor (DoSS)	HS^{3}D	2,796 (true), 10000 (false)	1:3.577
	Homo Sapiens	8000 (true), 8000 (false)	1:1
	Arabidopsis thaliana	8000 (true), 8000 (false)	1:1

prunes a percentage (20%) of each network's total convolutional nodes, which limits the co-dependencies each node in the network has on other nodes in the network, subsequently reducing model overfitting (Srivastava et al. 2014). Lastly, the output is fed through a $Softmax$ activation function, which produces, for each given input sequence, a probability of that sequence being a true/false acceptor/donor site. The ADAM optimizer with an inverse time decay learning rate schedule is used during model compilation (Kingma and Ba 2014). This architecture is consistent across the CNN sub-models, and is used for both AcSS prediction and DoSS prediction. The CNN architecture parameters were selected using hyperparameter tuning. For the hyperparameter tuning, see Table 2.

The DNN sub-models in EnsembleSplice consist of two fully-connected dense layers, followed by a dropout layer, another fully-connected dense layer, and another dropout layer. The first two fully-connected dense layers have 704 and 224 nodes, respectively, and both use a kernel regularizer with an L2 regularization penalty of 0.025. The third fully-connected densee layer has 512 nodes, but uses no regularization penalties. The first dropout layer prunes 10% of the DNN's nodes and the second dropout layer prunes 15%. Each fully-connected layer incorporates the $ReLU$ activation function. Identical to the CNN sub-models, the output layer is a $Softmax$ activation function and model compilation for the DNN sub-models is completed via the ADAM optimizer with an inverse time decay learning rate schedule. All DNN sub-models use this architecture for both AcSS prediction and DoSS prediction, and the parameters for this architecture were also selected using hyperparameter tuning.

EnsembleSplice is implemented via the TensorFlow/Keras framework (Abadi et al. 2016; Chollet and others 2018). For all experiments conducted, 30 was the maximal number of epochs used for training. The early model stopping callback, which ceases training if the model's validation loss does not improve for a selected number of epochs, was used during training and validation, which occurred in Google Colaboratory https://colab.research.google.com/ and made use of Graphical Processing Unit (GPU) hardware. Cross validation was used for initial CNN and DNN sub-model architecture testing.

One-hot Encoding

Genomic DNA splice site regions are composed of four nucleotides - A (Adenine), C (Cytosine), G (Guanine), or T (Thymine). Given the input constraints of DL architectures, these nucleotides are encoded numerically, as opposed to categorically. Each nucleotide corresponds to a row in a 4×4 identity matrix. The encoding scheme utilized in this paper is that A corresponds to `[1, 0, 0, 0]`, C corresponds to `[0, 1, 0, 0]`, G corresponds to `[0, 0, 1, 0]`, and T corresponds to `[0, 0, 0, 1]`. Since each splice site region consists of some N nucleotides, the final numerical representation for each splice site region is a $N \times 4$ matrix, where each row is a one-hot encoded nucleotide that occurs at the same index as it did in the splice site region's original representation.

Table 2: Sub-model Hyperparameter Tuning

CNN Parameters	**Search Space**
Conv. Layer 1 Filters	$\{8, 16, \ldots, \mathbf{72}, \ldots, 400\}$
Conv. Layer 1 Kernel Size	$\{1, 3, \mathbf{5}, 7, 9\}$
Conv. Layer 2 Filters	$\{8, 16, \ldots, \mathbf{144}, \ldots, 400\}$
Conv. Layer 2 Kernel Size	$\{1, 3, 5, \mathbf{7}, 9\}$
Conv. Layer 3 Filters	$\{8, 16, \ldots, \mathbf{168}, \ldots, 400\}$
Conv. Layer 3 Kernel Size	$\{1, 3, 5, \mathbf{7}, 9\}$
Dropout Layer	$\{\frac{1}{20}, \frac{2}{20}, \frac{3}{20}, \mathbf{\frac{4}{20}}, \ldots, \frac{10}{20}\}$
DNN Parameters	**Search Space**
Dense Units 1	$\{32, 64, \ldots, \mathbf{704}\}$
Dense Kernel Reg. 1	$\{\frac{1}{1000}, \frac{1}{400}, \frac{1}{200}, \frac{1}{100}, \mathbf{\frac{1}{40}}, \frac{1}{20}\}$
Dense Units 2	$\{32, 64, \ldots, \mathbf{224}, \ldots, 704\}$
Dense Kernel Reg. 2	$\{\frac{1}{1000}, \frac{1}{400}, \frac{1}{200}, \frac{1}{100}, \mathbf{\frac{1}{40}}, \frac{1}{20}\}$
Dropout Layer 1	$\{\frac{1}{20}, \mathbf{\frac{2}{20}}, \ldots, \frac{10}{20}\}$
Dense Units 3	$\{32, 64, \ldots, \mathbf{512}, \ldots, 704\}$
Dropout Layer 2	$\{\frac{1}{20}, \frac{2}{20}, \mathbf{\frac{3}{20}}, \ldots, \frac{10}{20}\}$

Cross Validation, Training, and Testing

The HS^{3}D, *Homo sapiens*, *Arabidopsis thaliana* dataset subsets were each split into a training (80% of the data) and a testing (20% of the data) subset. Cross-validation has been demonstrated to be an effective tool for model selection, and as such, 10-fold cross validation was used for evaluating alternative EnsembleSplice sub-model architectures (Shao 1993). For each dataset, the training subset of that dataset was partitioned into 10 approximately equal sized subsets. Every subset was used at some point to evaluate the performance of EnsembleSplice; for each of the 10 runs, training occurred on 9 subsets, and testing occurred on the last subset. The cross-validation performance for a particular dataset was the average performance over the 10 folds. Once the sub-model architectures for EnsembleSplice were chosen, EnsembleSplice was trained on the full training subset of each dataset, and then tested **once** on the testing subset of the respective dataset. No additional training or validation occurred following the final test run; the results reported for EnsembleSplice in this paper are the performances from this final test run.

Experiments and Results

Evaluation Metrics

To measure the performance of EnsembleSplice, and to compare EnsembleSplice with with other splice site detection models, the counts of correctly identified true AcSS or DoSS (true positive, "TP"), correctly identified false AcSS or DoSS (true negative, "TN"), incorrectly identified true AcSS or DoSS (false positive, "FP"), and incorrectly identified false AcSS or DoSS (false negative, "FN") are used. See Table 3.

Table 3: Confusion Matrix for Binary Classification Tasks

Actual Class	Predicted Class	
	Class Positive	*Class Negative*
Class Positive	True Positive (TP)	False Negative (FN)
Class Negative	False Positive (FP)	True Negative (TN)

From these metrics, additional metrics common to classification tasks can be used for evaluation: Accuracy (Acc) - the fraction of AcSS or DoSS correctly identified, Precision (Pre) - the fraction of positive classifications for AcSS or DoSS that were positive, Sensitivity (Sn) - the fraction of positive AcSS or DoSS with a positive classification (true positive rate), Specificity (Sp) - the fraction of negative AcSS or DoSS with a negative classification (true negative rate), Matthew's correlation coefficient (Mcc) - the correlation between true/false AcSS and DoSS and the classifications for them generated by the mode, and F_1 score - the harmonic means of the fraction of positive classifications for AcSS or DoSS that were positive and the fraction of positive AcSS or DoSS that were correctly identified. Lastly, the error rate measures how often the classifier misclassified the data. The equations for these metrics are in Table 4.

Table 4: Evaluation Metrics

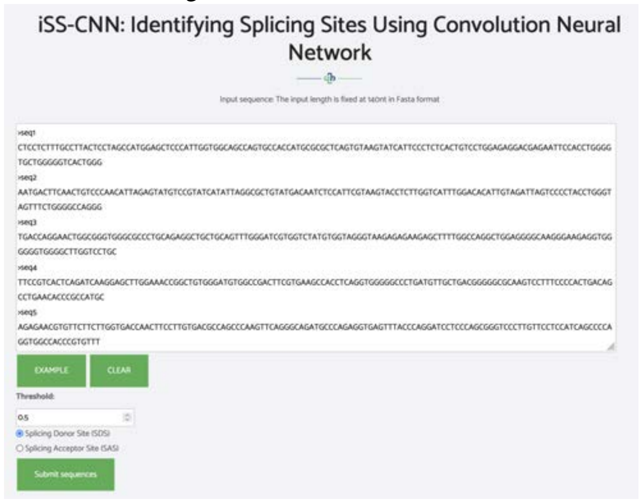
Metric	Equation
Acc	$\frac{TP+TN}{TP+FN+TN+FP}$
Sp	$\frac{TN}{TN+FP}$
Sn	$\frac{TP}{TP+FN}$
F_1	$\frac{2\times TP}{2\times TP+FP+FN}$
Pre	$\frac{TP}{TP+FP}$
Mcc	$\frac{TP\times TN-FP\times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$
Error rate	$1-$ Accuracy

Model Benchmarking

The state-of-the-art models iss-CNN and SpliceRover were used in a comparision with EnsembleSplice (Zuallaert et al. 2018; Tayara, Tahir, and Chong 2019).

iss-CNN was trained on a subset of HS^3D data, and consists of a convolutional layer with 16 filters, a kernal size of 7, and stride size of 3, a dropout layer that prunes 30% of the nodes, and a fully-connected dense layer that uses the *Sigmoid* activation function. The testing was conducted on iss-CNN's public web server, which can be found at http://nsclbio.jbnu.ac.kr/tools/iSS-CNN/, and the classification threshold used for predicting AcSS or DoSS was 0.5. The HS^3D testing subset, which was also used to evaluate EnsembleSplice, was used for benchmarking iss-CNN. See Figure 3

Figure 2: iss-CNN Webserver



iSS-CNN: Identifying Splicing Sites Using Convolution Neural Network

Input sequence: The input length is fixed at 140nt in Fasta format

```
>seq1
CTCCTCTTTGCCTTACTCCTAGCCATGGAGCTCCCATTGGTGGCAGCCAGTGCCACCATGCGCGCTCAGTGTAAGTATCATTCCCTCTCACTGTCCTGGAGAGGACGAGAATTCCACCTGGGG
TGCTGGGGGTCACTGGG
>seq2
AATGACTTCAACTGTCCCAACATTAGAGTATGTCCGTATCATATTAGGCGCTGTATGACAATCTCCATTCGTAAGTACCTCTTGGTCATTTGGACACATTGTAGATTAGTCCCCTACCTGGGT
AGTTTCTGGGGCCAGGG
>seq3
TGACCAGGAACTGGCGGGTGGGCGCCCTGCAGAGGCTGCTGCAGTTTGGGATCGTGGTCTATGTGGTAGGGTAAGAGAGAAGAGCTTTTGGCCAGGCTGGAGGGGCAAGGGAAGAGGTGG
GGGGTGGGGCTTGGTCCTGC
>seq4
TTCCGTCACTCAGATCAAGGAGCTTGGAAACCGGCTGTGGGATGTGGCCGACTTCGTGAAGCCACCTCAGGTGGGGGCCCTGATGTTGCTGACGGGGGCGCAAGTCCTTTCCCCACTGACAG
CCTGAACACCCGCCATGC
>seq5
AGAGAACGTGTTCTTCTTGGTGACCAACTTCCTTGTGACGCCAGCCCAAGTTCAGGGCAGATGCCCAGAGGTGAGTTTACCCAGGATCCTCCCAGCGGGTCCCTTGTTCCTCCATCAGCCCCA
GGTGGCCACCCGTGTTT
```

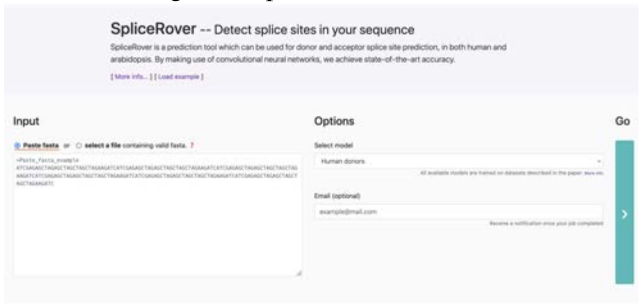
Threshold: 0.5

[✓ Splicing Donor Site (SDS)]
Splicing Acceptor Site (SAS)

Figure 3: The iss-CNN public webserver.

SpliceRover was trained on human genomic DNA data and *Arabidopsis thaliana* genomic DNA data, and is another CNN. Its architecture consists of a convolutional layer with filters equal in number to the AcSS or DoSS length, a max-pooling layer, and a series of convolutional and max-pooling layers. A fully-connected dense layer follows the convolutional layers, and the output is lastly fed through a *Softmax* activation function. To benchmark SpliceRover, their publically available web server was used; a cut of 0.5 was again utilized, as this is what EnsembleSplice uses. The web server can be found at the following link: http://bioit2.irc.ugent.be/rover/splicerover. See Figure 5

Figure 4: SpliceRoverWebserver



SpliceRover -- Detect splice sites in your sequence

SpliceRover is a prediction tool which can be used for donor and acceptor splice site prediction, in both human and arabidopsis. By making use of convolutional neural networks, we achieve state-of-the-art accuracy.

[✓ Paste fasta] or select a file containing valid fasta.

Select model: Human donors

All available models are trained on datasets described in the paper.

Email (optional): example@mail.com

Receive a notification once your job completed

Figure 5: The SpliceRover public webserver.

The benchmarked results, along with EnsembleSplice's results, can be found in Table 5.

Table 5: Model Performances

Dataset	Splice Site	Model	Sp	Sn	Pre	Err	Acc	Mcc	F1
HS^{3}D	Acceptor	issCNN	89.20	91.84	83.05	9.83	90.16	79.53	87.22
		EnsembleSplice	**97.75**	**92.36**	**92.20**	**3.45**	**96.55**	**90.07**	**92.29**
	Donor	issCNN	94.50	94.99	90.61	5.32	94.68	88.61	92.75
		EnsembleSplice	**98.25**	**96.96**	**93.93**	**2.03**	**97.97**	**94.14**	**95.42**
Arabidopsis thaliana	Acceptor	SpliceRover	88.31	89.25	88.42	11.22	88.78	77.57	88.83
		EnsembleSplice	**93.88**	**93.44**	**93.85**	**6.34**	**93.66**	**87.31**	**93.64**
	Donor	SpliceRover	86.88	87.13	86.91	13.00	87.00	74.00	87.02
		EnsembleSplice	**94.06**	**94.81**	**94.11**	**5.56**	**94.44**	**88.88**	**94.46**
Homo Sapiens	Acceptor	SpliceRover	88.25	93.44	88.83	9.16	90.84	81.80	91.08
		EnsembleSplice	**93.19**	**93.94**	**93.24**	**6.44**	**93.56**	**87.13**	**93.59**
	Donor	SpliceRover	85.44	91.13	86.22	11.72	88.28	76.69	88.61
		EnsembleSplice	**95.31**	**96.00**	**95.34**	**4.34**	**95.66**	**91.31**	**95.67**

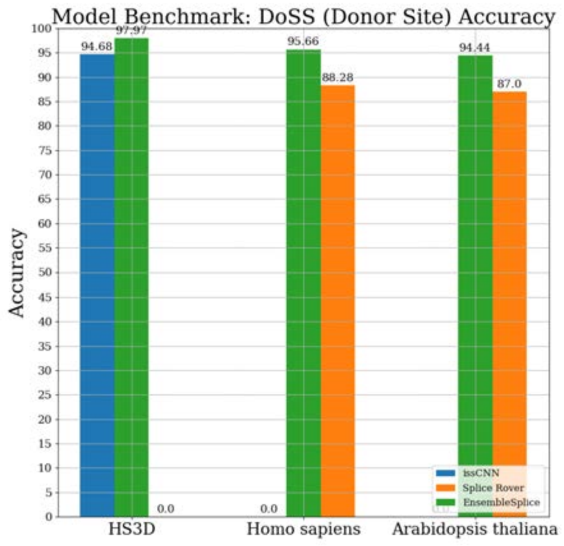


Figure 6: Donor site detection accuracies for each method tested.

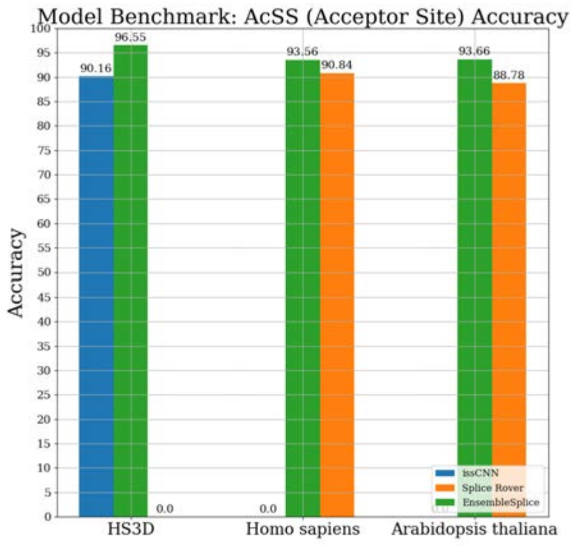


Figure 7: Acceptor site detection accuracies for each method tested.

The accuracies for each DoSS model benchmarked can be found in Figure 6, and the accuracies for each AcSS model benchmarked can be found in Figure 7.

Conclusion and Future Work

From the results, it can be observed that on all metrics employed, EnsembleSplice performed better than either iss-CNN or SpliceRover, two state-of-the-art methods that exist for splice site prediction and that use DL architectures. For future work, we consider evaluating model robustness; this would consist of testing a model trained on genomic DNA from one species on another species genomic DNA. In this case, that would mean testing the EnsembleSplice models trained on *Homo sapiens* data on the *Arabidopsis thaliana* data, and seeing how well the performance generalizes across species.

Acknowledgement

The work reported in this paper is supported by the National Science Foundation under Grant No. 2050919. Any opinions, findings and conclusions or recommendations expressed in this work are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[Abadi et al. 2016] Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, 265–283.

[Abril and Castellano Hereza 2019] Abril, J. F., and Castellano Hereza, S. 2019. Genome annotation. Elsevier.

[Albaradei et al. 2020] Albaradei, S.; Magana-Mora, A.; Thafar, M.; Uludag, M.; Bajic, V. B.; Gojobori, T.; Essack, M.; and Jankovic, B. R. 2020. Splice2deep: An ensemble of deep convolutional neural networks for improved splice site prediction in genomic dna. *Gene: X* 5:100035.

[Cao et al. 2020] Cao, Y.; Geddes, T. A.; Yang, J. Y. H.; and Yang, P. 2020. Ensemble deep learning in bioinformatics. *Nature Machine Intelligence* 2(9):500–508.

[Chollet and others 2018] Chollet, F., et al. 2018. Keras: The python deep learning library. *Astrophysics Source Code Library* ascl–1806.

[de Sá et al. 2018] de Sá, P. H.; Guimarães, L. C.; das Graças, D. A.; de Oliveira Veras, A. A.; Barh, D.; Azevedo, V.; da Silva, A. L. d. C.; and Ramos, R. T. 2018. Next-generation sequencing and data analysis: strategies, tools, pipelines and protocols. In *Omics Technologies and Bio-Engineering*. Elsevier. 191–207.

[Hahnloser et al. 2000] Hahnloser, R. H.; Sarpeshkar, R.; Mahowald, M. A.; Douglas, R. J.; and Seung, H. S. 2000. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature* 405(6789):947–951.

[Kingma and Ba 2014] Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Krizhevsky and Hinton 2010] Krizhevsky, A., and Hinton, G. 2010. Convolutional deep belief networks on cifar-10. *Unpublished manuscript* 40(7):1–9.

[Pertea, Lin, and Salzberg 2001] Pertea, M.; Lin, X.; and Salzberg, S. L. 2001. Genesplicer: a new computational method for splice site prediction. *Nucleic acids research* 29(5):1185–1190.

[Pohl et al. 2013] Pohl, M.; Bortfeldt, R. H.; Grützmann, K.; and Schuster, S. 2013. Alternative splicing of mutually exclusive exons—a review. *Biosystems* 114(1):31–38.

[Pollastro and Rampone 2002] Pollastro, P., and Rampone, S. 2002. Hs3d, a dataset of homo sapiens splice regions, and its extraction procedure from a major public database. *International Journal of Modern Physics C* 13(08):1105–1117.

[Quinlan and Hall 2010] Quinlan, A. R., and Hall, I. M. 2010. Bedtools: a flexible suite of utilities for comparing genomic features. *Bioinformatics* 26(6):841–842.

[Sagi and Rokach 2018] Sagi, O., and Rokach, L. 2018. Ensemble learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8(4):e1249.

[Shao 1993] Shao, J. 1993. Linear model selection by cross-validation. *Journal of the American statistical Association* 88(422):486–494.

[Srivastava et al. 2014] Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* 15(1):1929–1958.

[Tayara, Tahir, and Chong 2019] Tayara, H.; Tahir, M.; and Chong, K. T. 2019. iss-cnn: Identifying splicing sites using convolution neural network. *Chemometrics and Intelligent Laboratory Systems* 188:63–69.

[Zerbino et al. 2018] Zerbino, D. R.; Achuthan, P.; Akanni, W.; Amode, M. R.; Barrell, D.; Bhai, J.; Billis, K.; Cummins, C.; Gall, A.; Girón, C. G.; et al. 2018. Ensembl 2018. *Nucleic acids research* 46(D1):D754–D761.

[Zuallaert et al. 2018] Zuallaert, J.; Godin, F.; Kim, M.; Soete, A.; Saeys, Y.; and De Neve, W. 2018. Splicerover: interpretable convolutional neural networks for improved splice site prediction. *Bioinformatics* 34(24):4180–4188.

Classification of motor imagery: application of subject transfer in image-based 2D convolutional neural networks

Teddy Zaremba
University of Illinois - Chicago
1200 W Harrison St
Chicago, IL 60607
tzarem2@uic.edu

Adham Atyabi
University of Colorado - Colorado Springs
1420 Austin Bluffs Pkwy
Colorado Springs, CO 80918
aatyabi@uccs.edu

Abstract

This paper examines the effectiveness of deep learning models at predicting motor imagery (MI) tasks when adding data across subjects within the same dataset and from different datasets. We generate STFT representations of EEG data and classify those representations with a simple CNN architecture to test this influence. In addition to that, we compare the simple CNN with AlexNet for single trial.

The simple CNN outperforms AlexNet across every subject for single trial and gets close to state-of-the-art performance in BCI Competition IV Dataset I, and BCI Competiton III Dataset IVa, with it achieving 2.22% higher accuracy on subject E than the highest state-of-the-art model compared, while also outperforming some of the other studies for average accuracy.

When testing the influence of pertaining on other subjects, 9 of the 20 subjects tested saw an improvement in accuracy when the additional data was added. The overall trend is, of those 9, the lower performing subjects improved the most.

I. Introduction

Brain-computer interfaces (BCI) are communication technologies that allow people's brains to send commands to the outside world without passing through normal pathways such as nerves or muscles (Wolpaw et al. 2002). Electroencephalogram (EEG) data can store these encrypted signals through brain activity recorded by electrodes placed on a subject's scalp.

One set of tasks for BCI studies is a subject's intention to move their body, separate from the signals of actually moving their body. This set of tasks is called *motor imagery* (MI). The goal of MI classification models is to predict which body part a subject is imagining moving. A perfect MI prediction model could allow robotic prosthetics to receive directions from the brain about how to move, mimicking the way humans interact with their natural limbs (Katyal et al. 2014). Another application is that it would let people play video games with their thoughts instead of a controller or keyboard (Parthasarathy et al. 2020).

Our hypothesis is that a unique pattern in the EEG data would allow a model to correctly classify MI tasks for most people. Like image classification, the problem is that to generalize a MI prediction model across subjects effectively, the model needs to be robust to the variations across samples. This study aims to test the influence of adding all available subjects to the training data. However, unlike image classification, MI datasets typically have less than 20 subjects; the standard benchmark datasets from BCI competition III and IV have at most 9 subjects.

The solution this paper tests is the effect of combining multiple subject's data to pre-train before evaluating the last subject. This paper's approach combines MI datasets collected by different labs with different procedures conducting different experiments. For example, one lab could be testing left foot or right foot, while another lab is testing right hand and left foot. To combine these datasets, the MI prediction model has to predict all these tasks or generalize these tasks (ex. left and right foot become just foot). In addition to that, the model has to classify across recording sessions, recording devices, protocols, and experiment designs.

The model uses representations of the EEG data's component frequencies using the Short-time Fourier transform (STFT). After this, we test for single trial on a simple CNN and AlexNet. The motivation for testing AlexNet is that there are more resources put into image classification models, so finding an accurate image classifier of EEG data for motor imagery would help the BCI community by giving access to a much larger pool of possible networks for motor imagery classification.

The rest of the paper is organized as follows. Related work covers some of the state-of-the-art machine learning methods that have been applied to motor imagery classification. The method covers the prepossessing, STFT feature extraction, simple CNN and AlexNet architecture, and how we evaluated the impact of training on additional subjects. The results show the impact of the additional data along with a comparison with other state-of-the-art methods. The discussion section suggests possible explanations and limitations of these results. Lastly, the future work section provides a brief overview of possible strategies to improve on these results.

II. Related Work

This section provides an overview of some of the state-of-the-art machine learning models designed to classify MI tasks.

A. STFT

(Tabar and Halici 2016) generated STFT representations of EEG data. The STFT produced an image of both the mu band and the beta band for each epoch, and a specially designed convolutional neural network (CNN) model called CNN-SAE is used to classify those images. They reported accuracy of 77.6% with the highest accuracy of 95.3% on subject 4 on BCI Competition IV dataset 2b.

(Chaudhary et al. 2019) generated STFT and CWT representations of EEG data. From that, they fed the images to the image classification model AlexNet. They evaluated their model on BCI Competition III Dataset IVa. For single-trial, they obtained an accuracy of 99.35% with the CWT representations and 98.7% accuracy for STFT representations.

(Lu et al. 2016) implemented a restricted Boltzmann machine (RBM) to classify motor imagery tasks. They stack three RBM models together using a Frequential Deep Belief Network (FDBN) with a final softmax layer to classify the signals. They used BCI competition IV dataset 4a to test their model, and they reported 71% accuracy for subject transfer.

(Schirrmeister et al. 2017) applied three different CNN architectures. They tested CNN with 2, 5, and 31 layers. The shallow CNN provided the best results on the BCI Competition IV dataset IIa with 73.7% accuracy for frequencies ranging from 0-38 Hz and 60.8% accuracy for frequencies ranging from 4-38 Hz. They didn't report the accuracy of their model specifically for subject transfer, but (Amin et al. 2019) reported that they obtained 41.0% accuracy on the BCI Competition IV dataset IIa.

(Zhou et al. 2018) proposed a method that extracted the wave features from the signal data and then used that as input for an LSTM model. The wave extraction methods they used were Hilbert transform (HT) and discrete wavelet transform (DWT). They reported an accuracy of 91.43% on BCI competition II dataset III.

B. CSP

(Kumar, Sharma, and Tsunoda 2019) proposed a method that combines a common spatial pattern (CSP) model and a long short-term memory (LSTM) model to motor imaginary EEG signals. They used data BCI Competition IV Dataset I for single trial classification. The average accuracy across the subjects was 82.16%.

C. Raw Signal

(Luo and Lu 2018) used a Conditional Wasserstein GAN (CWGAN) to generate more EEG signal data to classify emotions. They only used high-quality data, which they evaluated using discriminator loss, maximum mean discrepancy, and two-dimensional mapping. The additional data reported accuracy improvements of 2.9%, 9.15%, and 20.13%, depending on the dataset.

(Amin et al. 2019) built on the work done by (Schirrmeister et al. 2017) by developing a CNN fusion method for EEG data. The networks are first pre-trained on a high gamma dataset (HGD) with 20 subjects. The fusion models they used were multilayer perceptron and an autoencoder. When testing for subject transfer classification on the BCI Competition IV dataset IIa, the proposed cross-encoding model scored 55.3% accuracy with it getting 69.43% accuracy on subject 9.

III. Method

A. Data

Some of the widely used benchmark datasets are from BCI Competition III and IV (Blankertz et al. 2005), (Tangermann et al. 2012). We used dataset IIa and dataset I from BCI Competition IV and dataset IVa from BCI Competition III.

BCI competition IV dataset IIa consists of 9 subjects labeled A01, A02, A03, A04, A05, A06, A07, A08, A09. In the experiment, they are asked to think about moving their right hand, left hand, both feet, and tongue, with the exception of A04, who did not perform the foot task. Motor imagery takes place for three seconds after the cue. The experiment was recorded with a sampling rate of 250Hz on 22 EEG channels. We evaluate this dataset on 3 classes: left, right, and foot.

BCI Competition IV dataset I consists of 7 subjects, A, B, C, D, E, F, G. It this experiment, subjects B, C, D, E, and G are asked to think about their left hand or right hand. Subjects A and F are asked to think about their left hand or feet. Motor imagery takes place for times varying from 1.5s to 8s. The experiment was recorded with a sampling rate of 1000Hz on 59 EEG channels.

BCI Competition III Dataset IVa consists of 5 subjects labeled aa, al, av, aw, ay. In the experiment, all the subjects were asked to think about moving their right hand or right foot. Motor imagery lasted for 3.5s. The experiment was recorded at 1000Hz on 118 EEG channels.

To make the datasets compatible, we did the following. We chose 12 channels, C1, C2, C3, C4, C5, C6, CP3, CP4, Cz, FC3, FC4, Fz, which are common across all the datasets. We chose an epoch length of 3.5s for each of the datasets when training across datasets. Lastly, all of the datasets were downsampled to 250Hz.

B. Preprocessing

The primary purpose of the prepossessing stage is to split the EEG data into time segments that represent a specific MI task.

The first step was to pick the 12 channels common across all the datasets. After this, we downsampled every subject to the lowest sample rate of 250 Hz. After this, we applied common average referencing for each of the channels. We apply a current source density filter to the data. After this, we applied a bandpass filter to only look at the Alpha and Betta waves (0.5-100 Hz). A bandpass filter of 8-30Hz is later applied when we generate the STFT representations. After this, we split the raw EEG into epochs of length 3.5s. The only exception is that we used a window of 4s when testing single trial for BCI Competition IV dataset IIa.

C. Short-Time Fourier Transform (STFT)

Short-time Fourier Transformation (STFT) has been proven to work as a feature representation to classify MI tasks using EEG data (Chaudhary et al. 2019), (Tabar and Halici 2016). STFT can represent the change in component frequencies over time. The formula is shown below.

$$X(t,f) = \int x(\tau)h(\tau - t)e^{-j2\pi f\tau}d\tau, \qquad (1)$$

where $x(t)$ is the STFT of the signal, $h(t)$ is the lowpass filter, and $X(t,f)$ is the correlation between $x(\tau)$ and $h(\tau - t)e^{-j2\pi}$. STFT representations effectively capture the temporal or frequency features of the signal with larger windows increasing the frequency resolution and shorter windows increasing the time resolution (Kwok and Jones 2000). For all subjects, we had a window size of 1s with an overlap size of 0.9s.

For AlexNet, the STFT spectrograms were transformed from shape 540 x 26 x 1 to RGB images with shape 224 x 224 x 3. The simple CNN took the STFT spectrograms at their original size of 540 x 26 x 1.

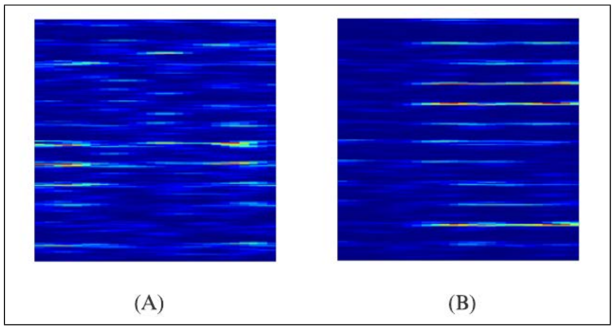


Figure 1: Resized STFT spectogram for AlexNet on subject al: right hand (A), foot (B)

D. CNN Models

After the STFT representations are generated, they are fed into either AlexNet or a simple CNN for classification. AlexNet consists of eight layers: 5 convolutional layers, 3 max-pooling layers, and a dropout layer, generating about 62 million parameters.

In contrast, the simple CNN we tested has 2,800,009 parameters with 2 convolutional layers and 1 max-pooling layer. The layers are shown below. In all experiments, the learning rate was set to 0.0001.

Layer Type	Output Shape	Parameters
Conv2D	(1, 540, 26, 4)	104
MaxPool2D	(1, 270, 13, 4)	0
Con2D	(1, 270, 13, 4)	404
Flatten	(1, 14040)	0
Dense	(1, 200)	2808200
Dropout	(1, 200)	0
Dense	(1, 2)	402

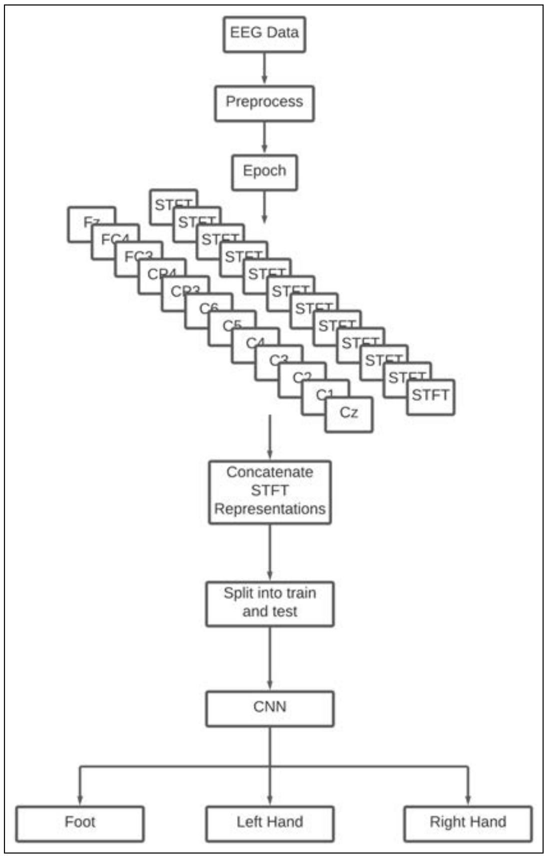


Figure 2: An overview of the method's structure from importing the EEG data to classification

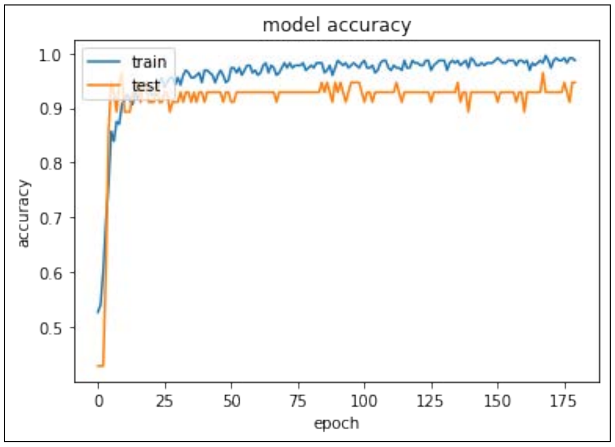


Figure 3: Validation and training accuracy for simple CNN plotted over epochs for subject al

V. Expiriments

Experiment 1

To test the influence of using all available data on motor imagery classification and the effectiveness of

STFT representations with the CNN models, we ran four experiments for each of the subjects, single-trial, cross-subject within-dataset (CSWD), and cross-subject cross-dataset (CSCD).For single-trial, the evaluation and the training data were combined into one dataset except for the subjects in BCI Competition IV dataset I because the tasks in the evaluation set were different from those in the training set. After this the data was shuffled, they are split into 2 groups; one that contains 80% of the subject's data for training and 20% of the subject's data for evaluation. The accuracy reflects the average of 3 iterations of a 5-fold cross validation. For each of the folds, training ran for 180 epochs with a batch size of 20.

Experiment 2

For the cross-subject within-dataset (CSWD) experiment, we ran a leave-one-out validation for zero-trial where none of the last subject's data is used and an experiment where 50% of the last subject's data is trained on after pretraining on the other subjects' data. Pretraining ran for 240 epochs with a batch size of 100. Retraining with the 50% trial ran for 120 epochs with a batch size of 20.

Experiment 3

For the cross-subject cross-dataset (CSCD) experiment, we ran a leave-one-out validation for zero-trial and a 50% trial. Even though each subject's data is symmetric, it's not symmetric across datasets, so the classes were weighted by taking the ratio between the number of total samples over the number of samples in a specific class. Pretraining ran for 400 epochs with a batch size of 100. Retraining with the 50% trial ran for 160 epochs with a batch size of 20. For both CSWD and CSCD, each of the subject's 50% was tested separately, and the average accuracy for an iteration reflects the average of those sets.

VI. Discussion

A. Single-Trial

For single trial, both a simple CNN and AlexNet were evaluated. In every subject, the simple CNN outperformed AlexNet. Two possible contributing factors to this are the number of parameters in AlexNet and the inherent distortion when reshaping the input to 224 x 224 x 3. The simple CNN has 22x fewer parameters, which might be better suited for a smaller amount of data. AlexNet was developed with the intention of classifying ImageNet, which has 22 million images, substantially more images than the number of samples in the BCI competition datasets. BCI Competition IV dataset IIa has the most at only 432 samples for 3-class between training and validation. The other issue with AlexNet could be that the STFT spectrograms have to be distorted before AlexNet can train on them. Since we chose 12 channels, the shape of the spectrogram is 540 x 26, a rectangular matrix that gets converted to a square matrix.

Comparing the results to other studies, the model performs the best on BCI Competition III dataset IVa. Just with single-trail the average accuracy shows an increase of 7.49% from (Su et al. 2020), 4.59% from (She et al. 2018), and 1.00% from (Park and Chung 2018).

B. Adding additional data

There isn't a large amount of EEG data for classifying motor imagery tasks. Thus we tested the influence of adding more data both from within that dataset and across other datasets. The subjects where adding more helped were A05, A06, A07 from BCI competition IV dataset IIa, C, F, and G from BCI Competition IV dataset I and subjects al, av, and aw from BCI competition III dataset IVa. The average change in accuracy from single trial to CSWD - 50% was -6.81%, -2.77%, and 0.794% for BCI Competition IV Dataset IIa, BCI Competition IV Dataset I, and BCI Competition III Dataset IVa, respectively. The average change in accuracy from single trial to CSCD - 50% was -3.69%, -0.31%, and -3.65% for BCI Competition IV Dataset IIa, BCI Competition IV Dataset I, and BCI Competition III Dataset IVa, respectively. A significant contributing factor could be that the single trial takes 30% more of the subjects' data to train before evaluation.

That being said, some subjects improved their accuracy with more training data from the other subjects. Going from single trial to CSWD - 50%, subject C, from BCI Competition IV dataset I, had a 9.42% increase. Going from single-trial to CSCD, subject A05 from BCI Competition IV dataset IIa saw an improvement of 8.45%. These subjects both performed the worst for single trial in their respective datasets. Since these were low-performing subjects, this could imply that these subjects need a more robust model to classify their tasks. If these subjects have lower quality data, meaning the signals do not match the task, then it follows that when the model sees other subjects' low-quality data, the model becomes more robust to those outlier samples where the subject may not be thinking about the task. On the other side, for subjects with high-quality data, the results seem to suggest that the model does not need to capture these outlier data points.

VII. Conclusion

This paper demonstrated that adding additional subjects from the same dataset as well as from other datasets can mainly improve accuracy of low-performing subjects for classifying motor imagery tasks because 9 of the 20 subjects saw an improvement when increasing the amount of data. This paper also outlines some of the potential benefits of opting for a simple CNN as opposed to a network with many more parameters like AlexNet. It also outlined that feature representations meant for subject transfer can still remain competitive with state-of-the-art single-trial methods for BCI Competition III Dataset IVa, with it improving on three of the existing state-of-the-art techniques.

VIII. Future work

This study tested the impact of adding additional data by combining the data from all the other subjects in a specific subject's dataset or all the data from all the datasets used. Given that the poor subject's improved in accuracy and the

Table 1: BCI Competition IV Dataset IIa

	A01	A02	A03	A05	A06	A07	A08	A09	Average
CNN	**79.98%**	**67.28%**	**82.01%**	53.14%	55.78%	74.31%	**69.14%**	**76.10%**	**69.18%**
AlexNet	77.28%	58.2%	77.18%	42.74%	53.09%	63.53%	68.29%	72.61%	63.49%
CNN CSWD - 0%	58.45%	36.19%	46.37%	36.80%	36.57%	26.77%	51.15%	47.22%	42.44%
CNN CSWD - 50%	73.53%	63.27%	75.85%	49.23	47.61%	64.49%	64.43%	72.76%	63.90%
CNN CSCD - 0%	59.03%	35.01%	45.68%	39.74%	36.57%	32.79%	43.40%	42.82%	42.44%
CNN CSCD - 50%	70.76%	53.55%	74.54%	**61.61%**	**63.27%**	**81.25%**	66.59%	67.67%	67.41%

Table 2: BCI Competition IV Dataset I

	A	B	C	D	E	F	G	Average
CNN	**82.00%**	**72.66%**	69.00%	**74.99%**	**98.66%**	77.00%	89.50%	**80.54%**
AlexNet	68.67%	63.67%	67.67%	72.33%	96.16%	63.17%	88.67%	74.33%
CNN CSWD - 0%	56.75%	64.83%	54.25%	50.67%	53.42%	50.67%	54.42%	55.00%
CNN CSWD - 50%	81.16%	68.33%	**78.42%**	68.83%	89.42%	79.58%	78.67%	77.77%
CNN CSCD - 0%	64.99%	59.00%	51.08%	52.75%	87.66%	49.67%	73.33%	62.64%
CNN CSCD - 50%	74.00%	68.83%	61.67%	69.00%	96.83%	**95.67%**	**95.67%**	80.24%

Table 3: BCI Competition III Dataset IVa

	aa	al	av	aw	ay	Average
CNN	**75.47%**	95.11%	72.38%	93.45%	**89.92%**	85.23%
AlexNet	65.83%	93.45%	65.12%	82.38%	86.55%	78.67%
CNN CSWD - 0%	61.25%	88.69%	51.07%	71.73%	64.22%	67.39%
CNN CSWD - 50%	75.42%	**97.50%**	73.75%	**96.61%**	87.02	**86.06%**
CNN CSCD - 0%	53.99%	76.49%	57.44%	54.76%	58.87%	67.39%
CNN CSCD - 50%	66.55%	86.93%	**77.26%**	92.86%	84.49%	81.62%

Table 4: BCI Competition IV Dataset I: Single Trial Comparison

Method	A	B	C	D	E	F	G	Average
(Qian et al. 2020)	66.9%	65.2%	**82.35%**	**94.55%**	94.9%	84.25%	81.15%	81.33%
(Kumar, Mamun, and Sharma 2017)	**88.1%**	59.1%	67.9%	84.3%	90.2%	85.9%	**92.2%**	81.1%
(Amirabadi and Kahaei 2020)	85.44%	**75.27%**	78.16%	80.72%	96.44%	**91.00%**	91.83%	**85.55%**
Proposed	82.00%	72.66%	69.00%	74.99%	**98.66%**	77.00%	89.5%	80.54%

Table 5: BCI Competition III Dataset IVa: Single Trial Comparison

Method	aa	al	av	aw	ay	Average
(Singh, Lal, and Guesgen 2019)	81.25%	**100%**	76.53%	87.05%	91.26%	87.22%
(Su et al. 2020)	76.43%	98.21%	72.35%	75.43%	66.46%	77.78%
(Kevric and Subasi 2017)	96%	92.3%	**88.9%**	**95.4%**	91.4%	**92.8%**
(She et al. 2018)	61.7%	100%	73.88%	88.17%	79.64%	80.68%
(Park and Chung 2018)	**100%**	74.11%	67.85%	90.07%	89.29%	84.26%
(Jin et al. 2020)	82.1%	93.9%	73.6%	93.6%	**93.2%**	87.28%
Proposed	75.47%	95.11%	72.38%	93.45%	89.92%	85.27%

Table 6: Best Performance Achieved

Dataset	Subject	Method	Average Performance
BCI Competition IV Dataset IIa	A01	CNN + Singe Trial	79.98 %
	A02	CNN + Singe Trial	67.28 %
	A03	CNN + Singe Trial	82.01 %
	A05	CNN + CSCD - 50%	61.61 %
	A06	CNN + CSCD - 50%	63.27 %
	A07	CNN + CSCD - 50%	81.25%
	A08	CNN + Singe Trial	69.14 %
	A09	CNN + Singe Trial	76.10 %
BCI Competition IV Dataset I	A	CNN + Singe Trial	82.00 %
	B	CNN + Singe Trial	72.66 %
	C	CNN + CSWD - 50%	78.42 %
	D	CNN + Singe Trial	74.99 %
	E	CNN + Singe Trial	98.66 %
	F	CNN + CSCD - 50%	95.67 %
	G	CNN + CSCD - 50%	95.67 %
BCI Competition III Dataset IVa	aa	CNN + Singe Trial	75.47 %
	al	CNN CSWD - 50%	97.50 %
	av	CNN CSCD - 50%	77.26 %
	aw	CNN CSWD - 50%	96.61 %
	ay	CNN + Singe Trial	89.92 %

good subjects did not see an improvement, this begs the question if there's a way to improve the accuracy for all the subjects. A future study could develop a method for finding the ideal subjects to train on. This could involve a clustering algorithm like KNN or looking at the co-variance of all pairs of subjects. The objective would be to find the subject that is most similar to reduce the amount of extraneous data.

Another area to investigate is a transformation to make one subject's data similar to another subject's data. Assuming subjects' data lie on their own feature plane, it follows that there should be a mapping from one subject to another. This could have the benefit of increasing the quality of the additional data, which seems to be especially important for the high-performing subjects.

Acknowledgement

The work reported in this paper is supported by the National Science Foundation under Grant No. 2050919. Any opinions, findings and conclusions or recommendations expressed in this work are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

Amin, S. U.; Alsulaiman, M.; Muhammad, G.; Mekhtiche, M. A.; and Hossain, M. S. 2019. Deep learning for eeg motor imagery classification based on multi-layer cnns feature fusion. *Future Generation computer systems* 101:542–554.

Amirabadi, M. A., and Kahaei, M. H. 2020. A new fast approach for an eeg-based motor imagery bci classification. *IETE Journal of Research* 1–10.

Blankertz, B.; Müller, K.-R.; Krusienski, D.; Schalk, G.; Wolpaw, J. R.; Schlögl, A.; Pfurtscheller, G.; Millán, J. d. R.; Schröder, M.; and Birbaumer, N. 2005. Bci competition iii. *Fraunhofer FIRST. IDA, http://ida. first. fraunhofer. de/projects/bci/competition_iii*.

Chaudhary, S.; Taran, S.; Bajaj, V.; and Sengur, A. 2019. Convolutional neural network based approach towards motor imagery tasks eeg signals classification. *IEEE Sensors Journal* 19(12):4494–4500.

Jin, J.; Liu, C.; Daly, I.; Miao, Y.; Li, S.; Wang, X.; and Cichocki, A. 2020. Bispectrum-based channel selection for motor imagery based brain-computer interfacing. *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 28(10):2153–2163.

Katyal, K. D.; Johannes, M. S.; Kellis, S.; Aflalo, T.; Klaes, C.; McGee, T. G.; Para, M. P.; Shi, Y.; Lee, B.; Pejsa, K.; et al. 2014. A collaborative bci approach to autonomous control of a prosthetic limb system. In *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 1479–1482. IEEE.

Kevric, J., and Subasi, A. 2017. Comparison of signal decomposition methods in classification of eeg signals for motor-imagery bci system. *Biomedical Signal Processing and Control* 31:398–406.

Kumar, S.; Mamun, K.; and Sharma, A. 2017. Csp-tsm: Optimizing the performance of riemannian tangent space mapping using common spatial pattern for mi-bci. *Computers in biology and medicine* 91:231–242.

Kumar, S.; Sharma, A.; and Tsunoda, T. 2019. Brain wave classification using long short-term memory network based optical predictor. *Scientific reports* 9(1):1–13.

Kwok, H. K., and Jones, D. L. 2000. Improved instantaneous frequency estimation using an adaptive short-time fourier transform. *IEEE transactions on signal processing* 48(10):2964–2972.

Lu, N.; Li, T.; Ren, X.; and Miao, H. 2016. A deep learning scheme for motor imagery classification based on restricted boltzmann machines. *IEEE transactions on neural systems and rehabilitation engineering* 25(6):566–576.

Luo, Y., and Lu, B.-L. 2018. Eeg data augmentation for emotion recognition using a conditional wasserstein gan. In *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2535–2538. IEEE.

Park, Y., and Chung, W. 2018. Bci classification using locally generated csp features. In *2018 6th International Conference on Brain-Computer Interface (BCI)*, 1–4. IEEE.

Parthasarathy, P.; Mantri, A.; Mittal, A.; and Kumar, P. 2020. Digital brain building a key to improve cognitive functions by an eeg–controlled videogames as interactive learning platform. In *Congress on Intelligent Systems*, 241–252. Springer.

Qian, L.; Feng, Z.; Hu, H.; and Sun, Y. 2020. A novel scheme for classification of motor imagery signal using stockwell transform of csp and cnn model. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 3673–3677. IEEE.

Schirrmeister, R. T.; Springenberg, J. T.; Fiederer, L. D. J.; Glasstetter, M.; Eggensperger, K.; Tangermann, M.; Hutter, F.; Burgard, W.; and Ball, T. 2017. Deep learning with convolutional neural networks for eeg decoding and visualization. *Human brain mapping* 38(11):5391–5420.

She, Q.; Chen, K.; Ma, Y.; Nguyen, T.; and Zhang, Y. 2018. Sparse representation-based extreme learning machine for motor imagery eeg classification. *Computational intelligence and neuroscience* 2018.

Singh, A.; Lal, S.; and Guesgen, H. W. 2019. Small sample motor imagery classification using regularized riemannian features. *IEEE Access* 7:46858–46869.

Su, J.; Yang, Z.; Yan, W.; and Sun, W. 2020. Electroencephalogram classification in motor-imagery brain–computer interface applications based on double-constraint nonnegative matrix factorization. *Physiological Measurement* 41(7):075007.

Tabar, Y. R., and Halici, U. 2016. A novel deep learning approach for classification of eeg motor imagery signals. *Journal of neural engineering* 14(1):016003.

Tangermann, M.; Müller, K.-R.; Aertsen, A.; Birbaumer, N.; Braun, C.; Brunner, C.; Leeb, R.; Mehring, C.; Miller, K. J.; Mueller-Putz, G.; et al. 2012. Review of the bci competition iv. *Frontiers in neuroscience* 6:55.

Wolpaw, J. R.; Birbaumer, N.; McFarland, D. J.; Pfurtscheller, G.; and Vaughan, T. M. 2002. Brain–computer interfaces for communication and control. *Clinical neurophysiology* 113(6):767–791.

Zhou, J.; Meng, M.; Gao, Y.; Ma, Y.; and Zhang, Q. 2018. Classification of motor imagery eeg using wavelet envelope analysis and lstm networks. In *2018 Chinese Control And Decision Conference (CCDC)*, 5600–5605. IEEE.

Increasing EEG training data for motor imagery deep learning models to subject transfer

Daniel Theng
California State University, Fresno
5241 N Maple Ave,
Fresno, CA 93740
dantheng@mail.fresnostate.edu

Adham Atyabi
University of Colorado, Colorado Springs
1420 Austin Bluffs Pkwy.
Colorado Springs, CO 80918
aatyabi@uccs.edu

Abstract

Advanced motor imagery brain-computer interfaces have several, diverse applications from providing communication capability to paralyzed and locked-in patients to controlling movement of wheelchairs for patients will lack of motor control/functionality. Non-invasive electroencephalogram (EEG) is commonly used due to its ease of use and relatively low risk with the drawback of having limited spatial resolution. Machine learning and computational data analytics are widely utilized across BCI studies due to the effectiveness of these methods in identifying and translating brain patterns to understandable commands. The recent success of deep learning methods, specifically computer vision models in terms of their ability to identify relationships between key patterns in an image, attracted its utilization in other fields such as BCI systems. Yet the development of EEG-based deep learning motor imagery BCI has struggled with its own issues, mainly a lack of large high quality training sets and challenges creating a model that can perform well across multiple subjects. Past works have been able to create models with high accuracies, but often with limited data thus rendering the models less practical than expected. This paper proposes a method to increase the amount of available motor imagery data by sampling overlapping windows from the EEG data and utilizing that data to create image representations of the EEG signals which will be used to train a simple Convolutional Neural Network(CNN) in hopes of creating an accurate model across many subjects.

Introduction

Brain-computer interfaces (BCIs) are systems that allow for communication between the human brain and an external device. BCIs are able to understand brain activity patterns and use that information to perform the task associated with that pattern. A well-known example of a BCI system is brain controlled wheelchair where BCI reads brain activities of a subject and provides motor controlling parameters to the wheelchair. Assisting those with motor disabilities is one of the many applications of BCIs. MI signals are commonly used in BCI studies due to the practicality of translating thought patterns to commands in contrast to other variations of EEG-based BCI systems where specially designed interfaces and the presence of continuous external stimuli is necessary.

When developing BCIs, brain activity recording devices are vital for collecting data. EEG records the electrical potentials that are measured across the scalp, and are one of the most common devices used in BCI studies. EEG is a non-invasive method for recording brain activity that requires relatively simple equipment compared to other recording methods such as fNIRS or fMRI (Gu et al., 2021). In addition, EEG have excellent temporal resolution in the range of milliseconds, but low spatial resolution (Gu et al., 2021).

Deep learning models are able to interpret complex and low resolution data and thus have been used to study EEG data with varying degrees of success. One of the main challenges within EEG-based BCI research is the lack of access large data sets. Collecting EEG data is a time-consuming process and thus the available data sets have limited number of subjects enrolled and the amount of data recorded from each subject is also limited. This poses a large challenge for use of DL method in EEG-based BCI studies, since proper training of DL models requires large amounts of samples (Tabar and Halici, 2016). Another issue with current BCIs is inconsistent performance across subjects which makes the task of of training models across subjects (subject transfer) difficult (Padfield et al., 2019). It is necessary that BCIs are able to perform well with multiple subjects, since in practical/industrial applications a BCI systems, a single BCI system is expected to accurately be used by several subjects with varying BCI skills and brain patterns. Yet EEG signals are extremely variable across subjects, recording sessions, recording devices and protocols, and experiment design.

This instability and pattern variability considerably increases the difficulty of implementing multi-subject, multi-session, multi-experiment BCI system. Since EEG patterns for the same stimuli between different subjects are known to be different and this instability is also observed in recording from the the same subject across multiple recording sessions (Abbass et al., 2014).

Because of the amount of noise in EEG data, feature representation methods are incredibly important to increase the accuracy of DL models. Feature extraction methods are ways reduce the dimensionality of data and may even remove redundant data (Dey et al., 2018). A common feature used for BCI MI studies is the Continuous Wavelet Transformation (CWT). CWT is able to extract both temporal and spatial features from the signal data, which is why it is very useful for EEG.

We believe that there is in fact a common pattern for motor imagery within EEG data and, by increasing the amount of data through overlapping and incorporating other subjects within the training data, a deep learning model can elucidate those patterns and accurately classify motor imagery EEG signals.

Related Works

Tabar and Halici (2016) used short time Fourier transformation (STFT) to create images of EEG signals to train their novel deep learning model combining a convolutional neural network (CNN) and a stacked autoencoder (SAE). They achieved an average accuracy of 77.6% with their proposed model, but note that the lack of large amount of data render deep learning methods less preferable for practical applications. They found their network outperformed previous networks using features based on EEG signals and not images.

Lu et al. (2016) used 3 Restricted Boltzmann Machines (RBMs) to classify motor imagery tasks from BCI Competition IV Dataset 2b. Using leave-one-out cross validation, they achieved an 84% average accuracy for session-transfer and 71% average accuracy for subject-transfer. This reflects the difficulty of subject-transfer.

Atyabi, Shic, and Naples (2016) attempts to resolve this issue by using multiple data sets and down sampling all of them to a 250 Hz sampling rate and spliting each epoch into smaller 0.5 second sub-epochs, but did not use an electrode reduction method.

Zhang et al. (2018) implemented ERSP image representation to train a recurrent convolution neural network to assess mental workload.

Raghu et al. (2020) also used STFTs to create images of EEG signals, but used those images with pretrained models such as Alexnet, Googlenet, Resnet, and Inceptionv3

Table 1: BCI Competition III & IV Datasets

Competition	Dataset	Sample Rate (Hz)	Number of Channels	Number of Subjects	Task
BCI Competition III	Dataset 2	240	64	2	character sequence
	Dataset 3a	250	60	3	left hand, right hand, foot, tongue
	Dataset 3b	125	2	3	left hand, right hand
	Dataset 4a	1000	118	5	right hand, foot
	Dataset 4b	1000	118	1	left hand, foot
	Dataset 4c	1000	118	1	left hand, foot
	Dataset 5	512	32.	3	left hand, right hand, word association
BCI Competition IV	**Dataset 1**	1000	64	7	left hand, right hand, foot
	Dataset 2a	250	22	9	left hand, right hand, foot, tongue
	Dataset 2b	250	3	3	left hand, right hand

Table 2: Common Ground Data set

Competition	Dataset	Sample Rate (Hz)	Number of Channels	Number of Subjects	Task
BCI Competition III	Dataset 4a	1000	118	5	right hand, foot
BCI Competition IV	Dataset 1	1000	64	7	left hand, right hand, foot
	Dataset 2a	250	22	9	left hand, right hand, foot, tongue
		250	**22**	**21**	**right hand, left hand, foot**

for seizure detection. They achieved a highest accuracy of 88.30%.

Xu et al. (2018) trained their own CNN on images from Continuous Wavelet Transformations from BCI competition IV Data set 2a. They achieved average best accuracy across subjects of 85.59%.

Wu et al. (2019) trained their own CNN on signal data from BCI competition IV Data set 2a and achieved an average accuracy of 75.8% across all subjects

Methods

Data Sets

One of the most widely used data sets in BCI studies are the BCI competition III and IV data sets (Al-Saegh, Dawwd, and Abdul-Jabbar, 2021). These data sets have different numbers of subjects, number of channels, sampling rates, and classes. **Table 1** provides the details on various BCI competition data sets. The data sets chosen for this paper are shown in **Table 2**. **Figure 1** depicts the data processing from the raw EEG data to training the CNN.

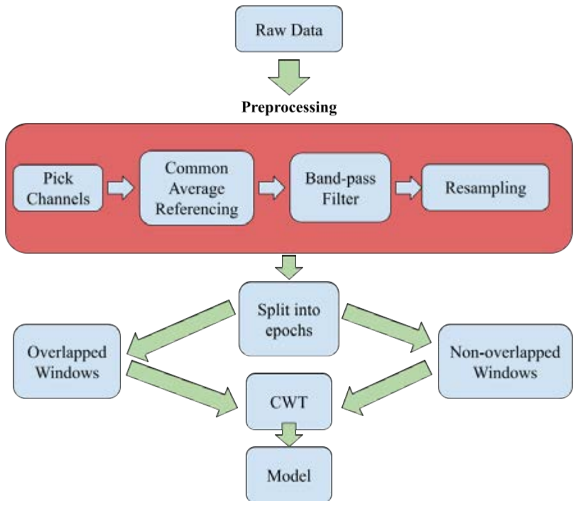


Figure 1: Methods

Preprocessing

In order to increase the size of available data, the BCI competition data sets was normalized relative to each subject. This includes creating a data set with the same number of channels, length of epochs, sampling rate, and classes. **Table 2** shows the data sets used and the common ground that was achieved. The preprocessing pipeline is outlined in **Figure 1** . First the raw data has all channels removed except for the channels in question; C1, C2, C3, C4, C5, C6, CP3, CP4, Cz, FC3, FC4, Fz. Next, the data was denoised using common average referencing and a band-pass filter was used to extract just the alpha and beta waves between 8 and 30 Hz. Alpha and beta waves are associated with motor movement, and by removing delta, gamma, and theta waves unnecessary information could be removed. The denoised data is then resampled to 250 Hz.

Continuous Wavelet Transformation

Those epochs will undergo a CWT and the transformed data will be used to create image arrays. Wavelets are similar to waves in that they oscillate, but differ that wavelets are localized. Wavelets have a spatial property called scaled that is similar to frequency or wavelength of wave(Addison, 2005). Scale refers to how tight or spaced out the wavelet is. The second property of wavelets is location which or time.

At a core level, the wavelet transformation attempts to find how much of the wavelet is within the signal. The wavelet is essentially convolved through the signal, and the wavelet is multiplied by the signal. For a Discrete Wavelet Transformation this occurs for a certain set of scales and times, but the Continuous Wavelet Transformation does this for all possible scales and times. The mathematical expression of a CWT is below.

$$X_w(a,b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} x(t)\overline{\psi}\left(\frac{t-b}{a}\right) dt$$

Where $\overline{\psi}(\frac{t-b}{a})$ is the Daubechies 7 and 10 (db7 and db10) Wavelet expression. After the signals were transformed, each channel array was stacked next to each other. These signal arrays can be converted to scalograms as shown in **Figure 3** and **Figure 4**. The signal arrays were then saved to train the CNN.

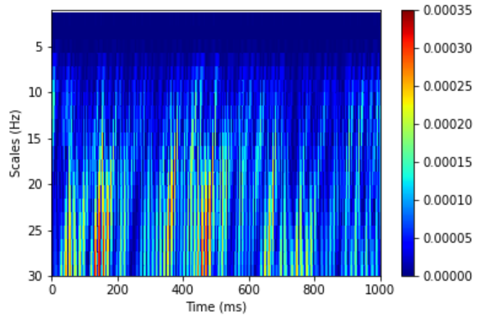


Figure 2: Daubechies 7 Wavelet Scalogram for a single channel.

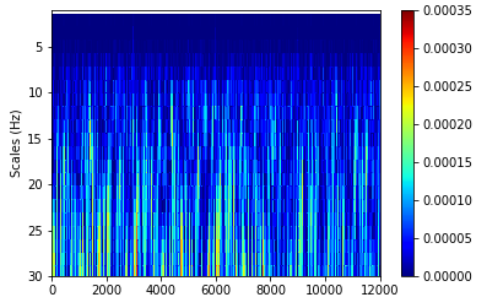


Figure 3: Daubechies 7 Wavelet Scalogram for a 12 channels stacked next to each other.

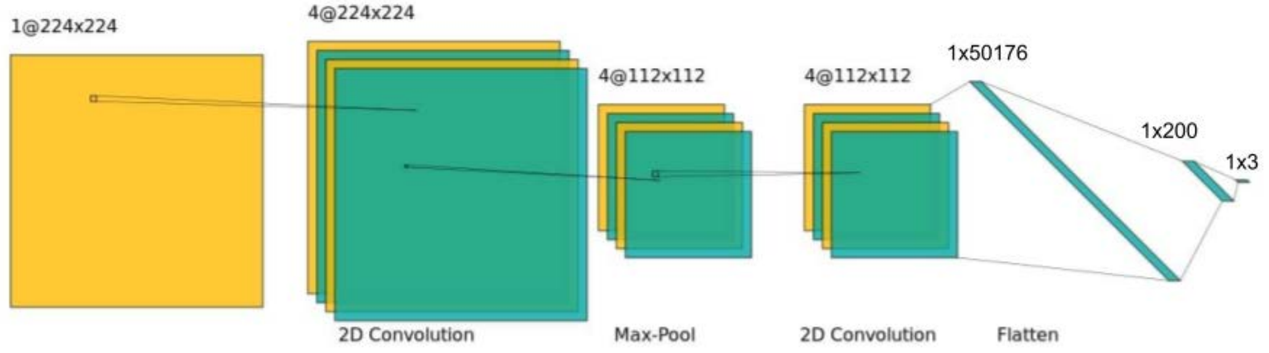


Figure 4: CNN Architecture

Overlapping

To increase the amount of data, one second overlapping windows were sampled from each of the epochs. The sampling starts with a 25% overlap at the beginning of the trial and increases to a 90% overlap near the center of the epoch, and then tapers off to 25% towards the end of the trial. This sampling method ensures that the center of the epoch, where the strongest motor imagery patterns are likely to be, are sampled more, but does not neglect the information near the beginning and the end of trials. Through overlapping, the amount of available data increased by a factor of 39. On caveat when using this method is to ensure that overlapped windows from the same epoch are not separated into training and testing sets. Instead all the overlapped windows of one epoch must be separated into either training and testing sets. Failure to complete this step can artificially increase the evaluation accuracy.

The Convolutional Neural Network

The model consists a 2D Convolutional layer, Max-pooling layer, 2D Convolutional layer, Flattening layer, Dense layer, a Dropout layer, and then a final dense layer. **Figure 5** displays the architecture.

Single Trial

For all the subjects in the datasets, the CNN was first trained and evaluated through a 5-fold cross validation solely on that subjects non-overlapped db7 data. The 5-fold cross validation was repeated 3 times and the average accuracy of all 3 repetitions was reported. Then the CNN was trained and evaluated on solely that subject's db7 overlapped data through the same 5-fold cross validation (3 repetitions). These steps were then repeated for the db10 data.

Cross the Subject

For all the subjects in each dataset, the CNN was first trained on all subject's db7 non-overlapped data within that dataset except the target subject and then evaluated on the target subjects data. The model's weight's were saved to be used as checkpoint. After loading the model's weights, it was calibrated to 50% of the target subject's data and evaluated on the other 50% and this was repeated 5 times. The average accuracy was reported. This process was repeated with db10 non-overlapped data, as well as the overlapped data for both wavelet types.

BCI Competition IV Data set 2a Results

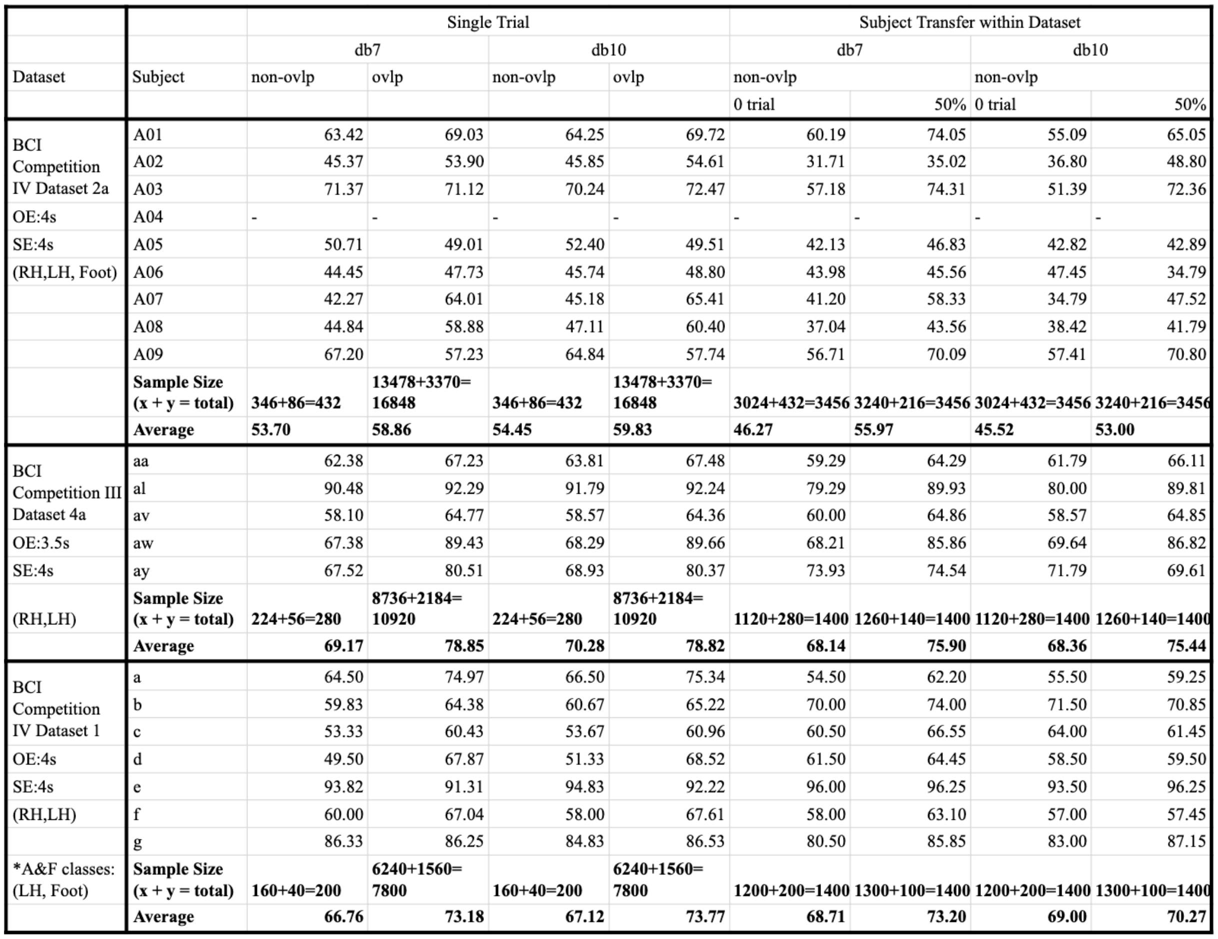
Table 3 shows the accuracies for each subject in every data set and a corresponding method. **Table 4** displays the best and second best accuracy achieved with that subject and the corresponding method to achieve that result.

Discussion

Within single trial, overlapping the data provides a substantial boost to the accuracy ranging from 5% to 9% average increase in accuracy compared to the non-overlapped single trial accuracies. In addition, most of the results found in **Table 3** are well above chance level, which illustrates that there does exist a common pattern within motor imagery EEG and that a neural network can recognize that pattern. Furthermore, **Table 4** clearly demonstrates that the most successful methods were overlapping and 50% subject transfer for nearly all subjects in every data set. Increasing the amount of available training data through overlapping and incorporating other subject's data provides a substantial boost to the accuracy. The implementation of these methods into other motor imagery BCI studies would strengthen future BCI systems.

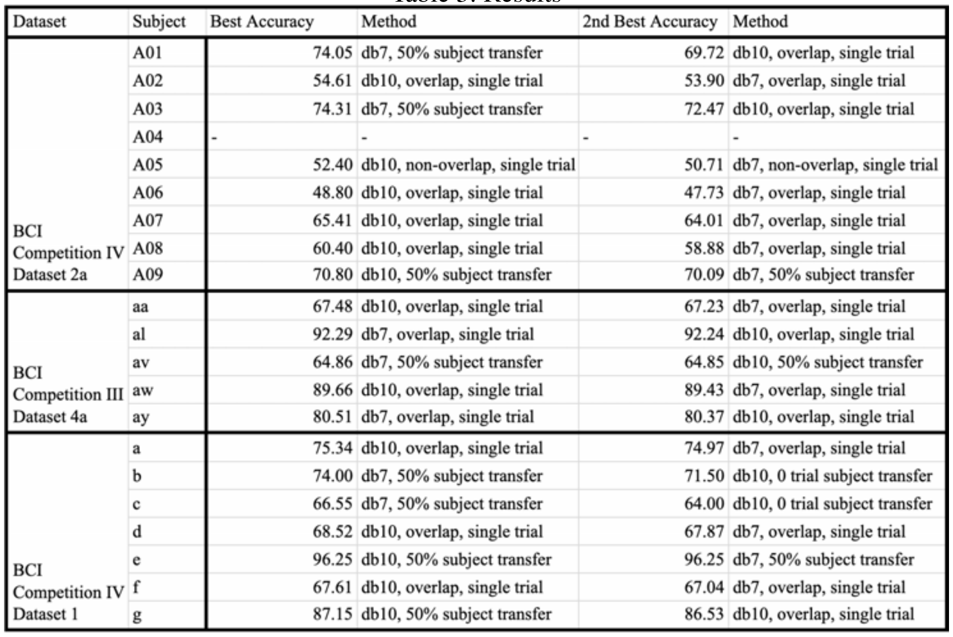
Conclusion

The possible practical applications of BCIs are very attractive. Yet the development of these systems is impeded by a lack of data and struggles with stable performance over multiple subjects. With the increase data size via overlapping and subject transfer, there is an increase of classification accuracy with a simple deep learning model. While the accuracies themselves may not be strong enough to build a successful BCI system, the increase in accuracy with the increase of data indicates that these methods can improve accuracies.



		Single Trial				Subject Transfer within Dataset			
		db7		db10		db7		db10	
Dataset	Subject	non-ovlp	ovlp	non-ovlp	ovlp	non-ovlp		non-ovlp	
						0 trial	50%	0 trial	50%
BCI	A01	63.42	69.03	64.25	69.72	60.19	74.05	55.09	65.05
Competition	A02	45.37	53.90	45.85	54.61	31.71	35.02	36.80	48.80
IV Dataset 2a	A03	71.37	71.12	70.24	72.47	57.18	74.31	51.39	72.36
OE:4s	A04	-	-	-	-	-	-	-	-
SE:4s	A05	50.71	49.01	52.40	49.51	42.13	46.83	42.82	42.89
(RH,LH, Foot)	A06	44.45	47.73	45.74	48.80	43.98	45.56	47.45	34.79
	A07	42.27	64.01	45.18	65.41	41.20	58.33	34.79	47.52
	A08	44.84	58.88	47.11	60.40	37.04	43.56	38.42	41.79
	A09	67.20	57.23	64.84	57.74	56.71	70.09	57.41	70.80
	Sample Size (x + y = total)	**346+86=432**	**13478+3370= 16848**	**346+86=432**	**13478+3370= 16848**	**3024+432=3456**	**3240+216=3456**	**3024+432=3456**	**3240+216=3456**
	Average	**53.70**	**58.86**	**54.45**	**59.83**	**46.27**	**55.97**	**45.52**	**53.00**
BCI	aa	62.38	67.23	63.81	67.48	59.29	64.29	61.79	66.11
Competition III	al	90.48	92.29	91.79	92.24	79.29	89.93	80.00	89.81
Dataset 4a	av	58.10	64.77	58.57	64.36	60.00	64.86	58.57	64.85
OE:3.5s	aw	67.38	89.43	68.29	89.66	68.21	85.86	69.64	86.82
SE:4s	ay	67.52	80.51	68.93	80.37	73.93	74.54	71.79	69.61
(RH,LH)	**Sample Size (x + y = total)**	**224+56=280**	**8736+2184= 10920**	**224+56=280**	**8736+2184= 10920**	**1120+280=1400**	**1260+140=1400**	**1120+280=1400**	**1260+140=1400**
	Average	**69.17**	**78.85**	**70.28**	**78.82**	**68.14**	**75.90**	**68.36**	**75.44**
BCI	a	64.50	74.97	66.50	75.34	54.50	62.20	55.50	59.25
Competition	b	59.83	64.38	60.67	65.22	70.00	74.00	71.50	70.85
IV Dataset 1	c	53.33	60.43	53.67	60.96	60.50	66.55	64.00	61.45
OE:4s	d	49.50	67.87	51.33	68.52	61.50	64.45	58.50	59.50
SE:4s	e	93.82	91.31	94.83	92.22	96.00	96.25	93.50	96.25
(RH,LH)	f	60.00	67.04	58.00	67.61	58.00	63.10	57.00	57.45
	g	86.33	86.25	84.83	86.53	80.50	85.85	83.00	87.15
*A&F classes: (LH, Foot)	**Sample Size (x + y = total)**	**160+40=200**	**6240+1560= 7800**	**160+40=200**	**6240+1560= 7800**	**1200+200=1400**	**1300+100=1400**	**1200+200=1400**	**1300+100=1400**
	Average	**66.76**	**73.18**	**67.12**	**73.77**	**68.71**	**73.20**	**69.00**	**70.27**

Table 3: Results



Dataset	Subject	Best Accuracy	Method	2nd Best Accuracy	Method
	A01	74.05	db7, 50% subject transfer	69.72	db10, overlap, single trial
	A02	54.61	db10, overlap, single trial	53.90	db7, overlap, single trial
	A03	74.31	db7, 50% subject transfer	72.47	db10, overlap, single trial
	A04	-	-	-	-
	A05	52.40	db10, non-overlap, single trial	50.71	db7, non-overlap, single trial
	A06	48.80	db10, overlap, single trial	47.73	db7, overlap, single trial
	A07	65.41	db10, overlap, single trial	64.01	db7, overlap, single trial
	A08	60.40	db10, overlap, single trial	58.88	db7, overlap, single trial
BCI Competition IV Dataset 2a	A09	70.80	db10, 50% subject transfer	70.09	db7, 50% subject transfer
	aa	67.48	db10, overlap, single trial	67.23	db7, overlap, single trial
	al	92.29	db7, overlap, single trial	92.24	db10, overlap, single trial
	av	64.86	db7, 50% subject transfer	64.85	db10, 50% subject transfer
	aw	89.66	db10, overlap, single trial	89.43	db7, overlap, single trial
BCI Competition III Dataset 4a	ay	80.51	db7, overlap, single trial	80.37	db10, overlap, single trial
	a	75.34	db10, overlap, single trial	74.97	db7, overlap, single trial
	b	74.00	db7, 50% subject transfer	71.50	db10, 0 trial subject transfer
	c	66.55	db7, 50% subject transfer	64.00	db10, 0 trial subject transfer
	d	68.52	db10, overlap, single trial	67.87	db7, overlap, single trial
	e	96.25	db10, 50% subject transfer	96.25	db7, 50% subject transfer
	f	67.61	db10, overlap, single trial	67.04	db7, overlap, single trial
BCI Competition IV Dataset 1	g	87.15	db10, 50% subject transfer	86.53	db10, overlap, single trial

Table 4: Best results for each subject

Acknowledgement

The work reported in this paper is supported by the National Science Foundation under Grant No. 2050919. Any opinions, findings and conclusions or recommendations expressed in this work are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

Abbass, H. A.; Tang, J.; Amin, R.; Ellejmi, M.; and Kirby, S. 2014. Augmented cognition using real-time eeg-based adaptive strategies for air traffic control. In *Proceedings of the human factors and ergonomics society annual meeting*, volume 58, 230–234. SAGE Publications Sage CA: Los Angeles, CA.

Addison, P. S. 2005. Wavelet transforms and the ecg: a review. *Physiological measurement* 26(5):R155.

Al-Saegh, A.; Dawwd, S. A.; and Abdul-Jabbar, J. M. 2021. Deep learning for motor imagery eeg-based classification: A review. *Biomedical Signal Processing and Control* 63:102172.

Atyabi, A.; Shic, F.; and Naples, A. 2016. Mixture of autoregressive modeling orders and its implication on single trial eeg classification. *Expert systems with applications* 65:164–180.

Dey, N.; Borra, S.; Ashour, A. S.; and Shi, F. 2018. *Machine Learning in Bio-Signal Analysis and Diagnostic Imaging*. Academic Press.

Gu, X.; Cao, Z.; Jolfaei, A.; Xu, P.; Wu, D.; Jung, T.-P.; and Lin, C.-T. 2021. Eeg-based brain-computer interfaces (bcis): A survey of recent studies on signal sensing technologies and computational intelligence approaches and their applications. *IEEE/ACM transactions on computational biology and bioinformatics*.

Lu, N.; Li, T.; Ren, X.; and Miao, H. 2016. A deep learning scheme for motor imagery classification based on restricted boltzmann machines. *IEEE transactions on neural systems and rehabilitation engineering* 25(6):566–576.

Padfield, N.; Zabalza, J.; Zhao, H.; Masero, V.; and Ren, J. 2019. Eeg-based brain-computer interfaces using motor-imagery: Techniques and challenges. *Sensors* 19(6):1423.

Raghu, S.; Sriraam, N.; Temel, Y.; Rao, S. V.; and Kubben, P. L. 2020. Eeg based multi-class seizure type classification using convolutional neural network and transfer learning. *Neural Networks* 124:202–212.

Tabar, Y. R., and Halici, U. 2016. A novel deep learning approach for classification of eeg motor imagery signals. *Journal of neural engineering* 14(1):016003.

Wu, H.; Niu, Y.; Li, F.; Li, Y.; Fu, B.; Shi, G.; and Dong, M. 2019. A parallel multiscale filter bank convolutional neural networks for motor imagery eeg classification. *Frontiers in neuroscience* 13:1275.

Xu, B.; Zhang, L.; Song, A.; Wu, C.; Li, W.; Zhang, D.; Xu, G.; Li, H.; and Zeng, H. 2018. Wavelet transform time-frequency image and convolutional network-based motor imagery eeg classification. *IEEE Access* 7:6084–6093.

Zhang, P.; Wang, X.; Zhang, W.; and Chen, J. 2018. Learning spatial–spectral–temporal eeg features with recurrent 3d convolutional neural networks for cross-task mental workload assessment. *IEEE Transactions on neural systems and rehabilitation engineering* 27(1):31–42.

Depth Estimation Images Generated from Monocular UAV Camera Images

Devin Haggitt
University of Colorado, Colorado Springs
1420 Austin Bluffs Pkwy
Colorado Springs, Colorado, USA
dhaggitt@uccs.edu

Adham Atyabi
University of Colorado, Colorado Springs
1420 Austin Bluffs Pkwy
Colorado Springs, Colorado, USA
aatyabi@uccs.edu

Abstract

Autonomous navigation of vehicles is a difficult problem to solve, with numerous approaches developed over the last few decades. Understanding the known and unknown dynamics of a vehicle's environment requires a flexible and adaptive technique to ensure the minimization of costs and constraints. These can include collisions, range, time, and computational resources. In the field of aerial vehicles, these can be particularly limited, which increases the need of further research. In this research, we aim to reduce the amount of collisions and flight time by using a Deep Learning (DL) network to assist the drone in identifying nearby objects in its environment. By training and testing the network using a series of flight data collected using a monocular UAV camera, we hope to reduce the computational time and complexity needed to accomplish obstacle avoidance. Our pretrained model will then generate depth estimation images using binary classification, denoting where objects are close and faraway. This will allow us to identify nearby and faraway objects in the path of the drone.

Introduction

The autonomous navigation of drones is a problem seen in a variety of different areas. These areas include photogrammetry, the study and mapping of landscapes using image data, networking, the field that aims to develop strong wireless signals, traffic monitoring, and in loads carrying (Skorobogatov, Barrado, and Salamí 2020). Some uses in photogrammetry include using drones to navigate and map out a forested area (Brust and Strimbu 2015). In this case, the drones need to collaborate and navigate around a forest environment, through trees and branches that could result in damaging collisions. The drones also need to know where previous drones have been, so that the data is not overwritten for that area. In networking, a fleet of drones can be used to increase the cellular signal in crowded events (Sawalmeh et al. 2017). In this environment, the drones need to navigate around other aerial objects and cell towers in order to get to a destination point. The drones also need to collaborate in strengthening the signal, and moving if there are locations with disruptions or high traffic. Creating a solution for these navigational problems, amongst similar problems, should produce fruitful results for technological innovation.

In this research, we aim to offer a solution to assist the autonomous aerial vehicle navigation problem. This solution will use a DL network that is trained to generate depth estimation images from frames collected from an aerial monocular camera attached to a drone. Using these depth estimation images, we can approximate location of obstacles that are close to the aerial vehicle in an indoor environment. This knowledge will be used in assisting drones that are either too small or financially limited in identifying nearby objects without the addition of payloads.

The DL network was trained to perform a binary classification on image sections, using a depth estimation image generated from an optical flow field as the ground truth depth for each section. From this binary classification, we can determine what objects might present potential collisions with our drone, allowing us to develop solutions to navigate around these objects. Creating a framework using a binary classification will allow a starting point to develop applications with a higher number of classes, to give the drone more options in navigating through its environment.

Related Work

There have been several approaches to the autonomous navigation of aerial vehicles. These approaches have used a variety of techniques, including the usage of game engines and deep learning networks. Some also use additional sensors to track the drone's location and surrounding environment. Below are some of the techniques used in developing a solution to the autonomous navigation problem, as well as some solutions to monocular depth estimation.

EGO-Swarm

EGO-Swarm was developed as a multi-robot autonomous navigation system (Zhou et al. 2020b). Each robot in the system maps out its own local surroundings, and communicates it with the rest of the drones in the network. These surroundings are captured using additional sensors attached to the drone, and the drone's location is also sent and registered in the ground control system. The ground control then constructs the global, obstacle-rich environment to assist the drones in the network in planning their trajectories.

These trajectories are planned using a local path planning solution. Using an Euclidean Signed Distance Field (ESDF) free gradient-based local planner known as EGO-Planner

(Zhou et al. 2020a), the drones in the network are able to plan out local trajectory solutions for their current surroundings. This was an improvement upon previous ESDF solutions, as the EGO-Planner solution minimized the updating range to only feasible solutions. By minimizing the computation time and complexity, this improvement allowed the trajectory planner to be computed using the on-board processor of each drone. By sharing these trajectories with other drones in the EGO-Swarm network, the computation time and complexity was further reduced, allowing the drones in the network to collaborate towards a common goal.

However, there are various issues associated with EGO-Swarm. First, none of their hardware specifications were shared, so replicating their results would be a difficult task to accomplish. Certain makes and models of drones also don't have the capability for added hardware, reducing the accessibility of their results and methods. Second, their EGO-Planner solution spends processing time to filter out other drones in the swarm. Not only does this add instructions to each drones' on-board processor, but it also open up a vulnerability where each drone relies on a previous drone to have travelled further through a common path. Each drone in the swarm also follows the same path calculated by a previous drone, causing an issue if a miscalculation were to happen or a drone were to fail to avoid a collision. Since each drone is not programmed to recalculate the path, mistakes will be replicated as each drone continues down the path of a previous drone. This also causes an issue for obstacles that are not stationary in the environment, like the other drones in the swarm.

ROSUnitySim

In contrast to the above solution, the ROSUnitySim technique aims to assist a fleet of drones in navigating an environment through the use of a game engine. By using Unity, this method seeks to generate solutions to local trajectories by mapping the location and local environment of each drone into a global environment (Hu and Meng 2016). This allows each drone in the swarm to discover their own solutions to traversing an obstacle-rich environment, using information gathered from the game engine. These trajectories are generated using Unity's in-built path finding algorithms, reducing the complexity and requirements of the system. It also maps out the environment by using Unity's rendering methods, further placing an emphasis on the Unity software.

Nonetheless, the ROSUnitySim method also has some disadvantages in the Unity software. One major disadvantage is Unity's lack of multi-threading solutions, requiring the use of a single thread on the CPU to generate calculations. This can be circumnavigated by transferring data over to the GPU to compute, but that increases the risk of memory failure and still requires a substantial amount of time for each rendering and path-finding solution. In both techniques, the time to compute one rendering or trajectory solution takes over 20ms, which impairs the drone from making quick and efficient maneuvers. This latency becomes exponential as more drones are added to the system, making ROSUnitySim a potentially unfit solution to the autonomous navigation problem.

DroNet and FlowDroNet

The DroNet model aims to solve the autonomous aerial navigation problem by training drones on data sets collected in real-world environments. (Loquercio et al. 2018) These data sets are taken from human controlled ground vehicles, such as cars and bicycles, and translated into a convolutional neural network (CNN) to train drones on obstacle detection and avoidance. This model is accurate in predicting trajectories for drones in urban road environments. However, it is limited to only environments where cars and bicycles can drive, making it unfit for other urban and rural environments.

To improve upon the results of DroNet, a team of researchers developed FlowDroNet (Sperling et al. 2020). This model wanted to fix the generalization problem in DroNet by applying optical flow fields. These fields created a general classification of several objects, creating a framework to improve issues found in the DroNet model. The FlowDroNet research team also wanted to increase the amount of data for training the CNN, so they used a simulated environment developed in Unity to try and fill up gaps in real-world data. This was successful and allowed the drones to generalize multiple different environments. The team applied their work in a realistic environment by having an individual drone fly through a parking garage with various amounts of lighting to ensure the drone could overcome different kinds of collisions. Using their results from physical testing, the FlowDroNet team published their findings with IEEE.

Even so, the FlowDroNet model still has its limitations. The drones used in the study have the ability to detect collisions, but they are not able to navigate around those potential collisions. In the real world application, the drone they used would stop right before a pillar, and either await a human navigator or land. The research team did not develop a path finding algorithm for the drone to use in order to navigate around obstacles. Therefore, their results are more of a landing point to solve the autonomous navigation problem, but they are not a suitable solution.

(Eigen, Puhrsch, and Fergus 2014) Method

Monocular depth estimation is important to understanding the geometric relations of objects within a scene or environment. In most cases of depth estimation, a collection of stereo images is used, or motion data, in order to collect information about the depth of objects in the environment. However, there a several uses where we are limited to a monocular image, which requires us to attempt a different approach to solve the problem.

Some applications where monocular depth estimation is important include uses in economics, electronics, and mathematics (Eigen, Puhrsch, and Fergus 2014). If a potential client to a real-estate company wanted details about the length of the driveway, or how far the kitchen island is from the oven, then the company would be interested in determining the depth of objects in the scene using a single image. In our case, we want our drones to be able to determine where obstacles are to avoid collisions using only a single camera attached to the drone.

To approach a solution to this problem, this method created a deep learning network that uses regression as a loss function. Using both the NYUD2 and KITTI data sets, this method trained and validated his depth estimation calculated from an 11-layer deep learning network. This network is formed from two component stacks, once determines the global depth of the image using a coarse-scale, while the other refines the coarse image to determine the depth of individual objects. The accuracy of their prediction is then measured against the ground truth accuracy, and predictions within an error rate of 25% are evaluated as a good result. (Eigen, Puhrsch, and Fergus 2014) introduced the idea of reporting accuracies within 1.25^2 and 1.25^3 as separate metrics.

(Eigen, Puhrsch, and Fergus 2014) accuracy			
	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
0-10 meters	61.1%	88.7%	97.1%

(Cao, Wu, and Shen 2018) Method

Taking inspiration from (Eigen, Puhrsch, and Fergus 2014), this project sought to improve the regression results by discretizing the ground-truth depth into 10 bins and running a deep learning network that uses classification as a loss (Cao, Wu, and Shen 2018). It was their hypothesis that depth estimation as a regression function would make it harder for a deep learning network to determine an accurate depth value. Since humans find it difficult to determine the exact distance, it was formulated that the deep learning network would have a similar issue. Therefore, it was predicted that the network would be better at generating rough estimates.

Using the NYUD2 dataset generated by their predecessor, this research was able to train and test in a similar environment, using the ResNet101 model. They were able to demonstrate improvements to the (Eigen, Puhrsch, and Fergus 2014) method using classification through a series of experiments. One of these experiments included an investigation into the network finding the mean depth of a given image, and determining the best accuracy by returning that mean. By splitting the input data into 3 different depth ranges, this method was able to demonstrate an improvement to the results of their predecessor. The following are this method's results:

(Cao, Wu, and Shen 2018) accuracy			
	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
0-3 meters	69.6%	91.2%	97.2%
3-7 meters	76.0%	94.9%	98.6%
7-10 meters	49.7%	74.9%	93.1%
Overall	69.4%	92.4%	97.5%

From these results, it can be interpreted that this model outperforms the (Eigen, Puhrsch, and Fergus 2014) model. By using classification instead of regression, and breaking up the ground-truth depth values into three different classes, this method was able to obtain a better accuracy (As shown in the 3-7 meter row). Comparing these results to our model will give us insight into where we want our performance to be, and will be used to determine best fit factors in our experimentation.

Our Contribution

By using deep learning methods introduced by (Cao, Wu, and Shen 2018) and (Eigen, Puhrsch, and Fergus 2014), we will be developing a depth estimation method to be used for unmanned aerial vehicles. This approach will establish efficiency in identifying nearby objects, which will allow drones with limited computational resources to maneuver through cluttered environments. It will also provide an accuracy that is close to modeling a depth camera, to allow drones without the ability to carry or incorporate a depth camera to identify potential collisions for trajectory paths.

Depth Estimation

Optical Flow

During our initial research, we created the objective to have the ability for drones to detect and capture the depth of objects. Our research brought us to using optical flow fields. These can be calculated using functions in the OpenCV library (Bradski 2000). Once a optical flow field is generated, we can calculate the depth of each pixel from an image pair using the equation:

$$Depth = v_{drone} * \frac{D_{focal}}{Mag_{flow}} \tag{1}$$

where v_{drone} is the velocity of the drone (constant 30 cm/s), D_{focal} is the distance of the pixel from the focal point, and Mag_{flow} is the magnitude of the flow vector. Since the flow vector is in pixels per frame, we also need to multiply the magnitude by the frame per second amount, set at 10 fps. Once the depth for each pixel is generated, it is stored in an array for depth estimation. From there the pixels are grouped together to form an 8x8 grid, which sections the image into an 8x8 grid of depths.

Deep Learning

Since the optical flow calculation takes 3-4 seconds to generate a depth estimation image, it is unsuitable for use in obstacle avoidance. In order to try and reduce the computation time, we decided to try using a deep learning network to classify depths for each section of a frame. By using the OpenCV library (Bradski 2000), we had a drone fly through a series of different courses under the control of a human operator, then stored each frame collected from the drone (at 10 frames per second) in a directory. From there, we used the OpenCV library to generate an optical flow field for each image pair, then calculated the depths. We then created depths estimation images by grouping depths together in 8x8 sections, with the minimum depth getting reported for each chunk with the exception of the center chunks.

Once the depth estimation images were generated, we cropped each depth estimation image and frame into 8x8 sections, then sorted the frames into 6 different classes. Each of these classes corresponded to a range of depth, with the closest being within 1 second of the drone (at a speed of 30 cm/s) and the furthest being more than 5 seconds away. From this sorting, we were able to create a deep learning network that used the classification as a loss function to generate a depth for each section in an image.

The creation of the deep learning network was then broken up into three phases. First, we had our drone fly through a course with a human navigator, then generated depth estimation images using optical flow fields for 5 different runs in different environments. With each run consisting of 500-800 frames, and a depth estimation image generated between each frame, we moved to processing the images. We cropped each frame and depth estimation image into 64 different chunks, forming an 8x8 grid, then matched each depth estimation image with its respective sectioned image. After cropping, we sorted each sectioned frame into their respective classes using data collected from the corresponding depth estimation image. This allowed us to develop a classification model for 2, 3, and 6 classes.

We hope that by using a classification model, we can strive to be accurate to as close to 80% as possible, to match the results of (Cao, Wu, and Shen 2018) and (Eigen, Puhrsch, and Fergus 2014) from their data sets. For us to accomplish this task, we performed a series of tests with several factors and levels. These factors include the amount of layers, the optimizer and learning rate, the amount of classes and their weights, and the dropout rate and validation split.

Experimentation

To determine and discover the best validation accuracy, we went through a series of experiments, adjusting our model in each step. We want to see changes in the model run time using different amounts of layers, and changes in both loss and accuracy in other factors. Our preliminary factor level for each trial was to use 9 layers, the Adam Optimizer, a 1e-3 learning rate, 2 classes, 0.8 Background : 1.2 Foreground class weights, 0.3 node dropout rate, and a 0.1 validation split. Once we discovered a factor level that showed improvements from these preliminary levels, we adapted our experiment to utilize the superior factor level.

Below are our results for our experimentation on the network factors. To obtain these results, we took an average over the last 10 epochs (or iterations) in each run, with each run consisting of 30 epochs. We predict that using this to compare our results will assist in keeping our model consistent, while offering insight into each factor level. We then analyze each level to determine the best version of our model, and repeat those factor levels in future results with additional changes to other factors. We also maintain the same random seed for splitting the training and validation data sets, to ensure that our results can both be replicated and compared to the same training and validation data.

Layers

To determine the best layer structure, we wanted to examine response variables such as run time, accuracy, and loss. We wanted to have a network that could compute 3 batches of 64 images in an efficient amount of time, while having a high amount of accuracy. Each amount of layers follows the same structure in regards to the model. Our deep learning network follows the model:

Model Architecture		
Layer	Type	Parameters
1	Conv2D	448
2	MaxPooling2D	0
3	Conv2D	4640
4	MaxPooling2D	0
5	Conv2D	18496
6	MaxPooling2D	0
7	Conv2D	73856
8	MaxPooling2D	0
9	Conv2D	295168
10	MaxPooling2D	0
11	Flatten	0
12	Dense	196736
13	Dense	258

Using this structure, we developed experiments for 7, 9, 11, and 13 layers. For 7 layers, we removed layers 5-10 in the above structure. We then adapted the structure for 9 and 11 layers by removing layers 7-10 and 9-10, respectively. Removing these layers also corresponded to a change in the number of trainable parameters. For 13 layers we have 589,602 parameters, 671,266 parameters for 11 layers, 1,375,658 for 9 layers, and finally 2,708,834 for 7 layers. Our results for each factor level is shown below (With the highest average value for each variable shown in bold).

7 Layers			
Run	Runtime	Accuracy	Loss
1	25.9	0.74183	1.02952
2	25.7	0.74277	1.04960
3	25.7	0.74587	1.06790
Average	25.8	0.74349	1.04901

9 Layers			
Run	Runtime	Accuracy	Loss
1	26.7	0.76392	0.74193
2	22.6	0.76176	0.80639
3	25.0	0.76092	0.72216
Average	**24.8**	0.76220	0.75683

11 Layers			
Run	Runtime	Accuracy	Loss
1	26.0	0.77384	0.62207
2	23.0	0.77582	0.59601
3	28.4	0.77645	0.60856
Average	25.8	**0.77537**	0.60888

13 Layers			
Run	Runtime	Accuracy	Loss
1	24.0	0.77260	0.59473
2	27.0	0.76720	0.60057
3	28.4	0.76696	0.60538
Average	26.5	0.76892	**0.60023**

Our findings show that the validation accuracy starts to deteriorate after 11 layers, with only slight decreases in validation loss. We also notice that the run time for each batch starts to increase after 9 layers. From these findings, we conclude that the best amount of layers to have in the model is 11, since that produced our highest average accuracy, while maintaining a low run time for each batch.

Optimizer

We continue our research by examining our optimizer. Most image classification models use Adam since it combines the best properties of RMSprop and AdaGrad, while maintaining itself as a replacement for stochastic gradient descent. We wanted to ensure Adam was the best optimizer for our use, so we tested it against one of its predecessors, RMSprop. Below are our results in analyzing both the accuracy and loss on the validation data set.

Adam		
Run	Accuracy	Loss
1	0.77387	0.59821
2	0.77014	0.58039
3	0.77336	0.62200
Average	**0.77246**	**0.60020**

RMSprop		
Run	Accuracy	Loss
1	0.64722	0.63728
2	0.65008	0.61736
3	0.63992	0.62605
Average	0.64574	0.62690

Our results conclude that Adam outperforms RMSprop in both validation accuracy and loss. Even on RMSprop's best run, the Adam optimizer still out performs on all 3 of their runs. Therefore, our findings support the use of the Adam optimizer, and we will continue to use it for our experiments.

Learning Rate

After determining the best optimizer, we shifted our experiment into finding the best learning rate. The default value for the Adam optimizer in TensorFlow Keras is 0.001 (Chollet 2015), so we wanted to increase and decrease from there to determine the best learning rate for our model. Below are our results for 1e-4, 1e-3, and 1e-2.

0.0001 Learning Rate		
Run	Accuracy	Loss
1	0.76192	0.51198
2	0.76509	0.51178
3	0.76978	0.49805
Average	0.76560	**0.50727**

0.001 Learning Rate		
Run	Accuracy	Loss
1	0.77372	0.61191
2	0.77113	0.61005
3	0.77412	0.61271
Average	**0.77299**	0.61156

0.01 Learning Rate		
Run	Accuracy	Loss
1	0.41220	0.69917
2	0.41220	0.70115
3	0.41220	0.70098
Average	0.41220	0.70043

These findings inspire additional research and experimentation to investigate some causes. Firstly, we want to look into the 0.01 learning rate. It is hypothesized that since the learning rate affects how fast the model adapts to the training data, that a model with a high training rate is more likely to over fit the data. This occurs when the model discovers commonalities in the training data, and adjusts to always get the training data correct. Since it is not trained to adjust to the validation data, it has a low accuracy when classifying the validation data. In this case, this capped the validation accuracy at 41.22%. This could be related to the foreground data making up 41.22% of the validation data, making the assumption that the model was classifying everything as foreground to generate the least amount of loss.

Secondly, when decreasing the learning rate, the loss decreased but so did the accuracy. Further research could be included to have the 1E-4 learning rate runs iterate over more epochs to see additional changes, however, due to our time restriction, we were not able to run over these epochs. Since 1E-3 gave the highest accuracy for 30 epochs, we decided to continue to use this as the learning rate of our model.

Number of Classes

After determining the number of layers and the optimizer and learning rates, we shift our investigation into the number of classes. Since we want our drone to be able to understand multiple aspects of our environment, we want to expand our amount of classes. However, we also want to establish our model, so we want to go with the amount of classes that gives us the best accuracy rate. This will allow us to compete with similar models, such as (Eigen, Puhrsch, and Fergus 2014) and (Cao, Wu, and Shen 2018).

2 Classes		
Run	Accuracy	Loss
1	0.77429	0.61231
2	0.77642	0.60582
3	0.77314	0.61379
Average	**0.77462**	**0.61064**

3 Classes		
Run	Accuracy	Loss
1	0.65431	0.96873
2	0.65207	0.94917
3	0.64714	0.98961
Average	0.65117	0.96917

6 Classes		
Run	Accuracy	Loss
1	0.41534	1.71566
2	0.43045	1.70062
3	0.41690	1.70270
Average	0.42090	1.70633

According to the table above, our best accuracy rate is two classes. This also is close to (Cao, Wu, and Shen 2018)'s results for a 25% error rate, motivating our research. While the average error rate across classes decreases, the loss greatly increases in proportion to the accuracy decrease. Therefore, adding additional classes will require further research and tuning, while the binary classification task allows us to accomplish our goal with the drones without causing too much error.

Class Weights

Since the proportion of nearby objects to far away objects is not even in our dataset (41.56% to 58.44&), we did some investigation into adjusting the class weights. During our preliminary investigation we found using 0.8:1.2 gave great results in determining nearby accuracy, so we decided to evaluate it against 0.9:1.1 and 1:1. We then compare the accuracy and loss of these factor levels to determine the best set of class weights.

0.8 Far : 1.2 Nearby		
Run	Accuracy	Loss
1	0.77586	0.61488
2	0.77654	0.59633
3	0.77781	0.61340
Average	0.77674	0.60820

0.9 Far : 1.1 Nearby		
Run	Accuracy	Loss
1	0.77677	0.59214
2	0.78041	0.58367
3	0.77704	0.58985
Average	0.77807	**0.58855**

1.0 Far : 1.0 Nearby		
Run	Accuracy	Loss
1	0.77724	0.60574
2	0.78137	0.59118
3	0.77723	0.61686
Average	**0.77861**	0.60459

According to these results, we achieve the best overall accuracy using 1:1 weights. Nonetheless, it is not much of an improvement over the 0.9:1.1 average overall accuracy. The 0.9:1.1 also has the best loss out of all 3 levels. We believe that should be a strong determining factor, since we want to be able to classify nearby objects correctly. It is hypothesized that the reduction in loss will help with both the difference in proportion, and classifying the nearby objects accurately. Since the overall accuracy is only a slight decrease from 1:1, we determined that 0.9:1.1 is the best set of weights for our classification model.

Node Dropout Rate

Another way to discourage the model over fitting is to use dropout on the nodes within your layers. Since the model cannot continue to use the same path through the network for the training data, it is forced to find new solutions, which also allows it to have a better validation accuracy. Originally we were using a dropout rate of 0.3, but we wanted to do an investigation into 0.5, and from there into 0.6. Below are our results for each dropout rate.

0.3 Dropout		
Run	Accuracy	Loss
1	0.77696	0.60919
2	0.77873	0.60563
3	0.77723	0.61365
Average	0.77657	0.60949

0.5 Dropout		
Run	Accuracy	Loss
1	0.77873	0.51042
2	0.77952	0.50884
3	0.78164	0.50163
Average	**0.77996**	0.50696

0.6 Dropout		
Run	Accuracy	Loss
1	0.77723	0.49051
2	0.77485	0.49027
3	0.78186	0.48900
Average	0.77798	**0.48993**

Supporting our original hypothesis, we found that increasing the dropout rate lead to a higher accuracy. Similar to the amount of layers, there seems to be a point of diminishing returns. That point here is a 0.5 dropout rate, with increases to the dropout rate decreasing the accuracy. The loss decreasing as we continue to increase the dropout rate seems to require further investigation, however. The dropout rate will be another factor to consider when tuning results in the future, as changes to the dropout might constitute a better validation accuracy or loss as we continue to add classes.

Training and Validation Split

Finally, we wanted to finish our investigation by changing the training and validation split. Since these results have a possibility of changing the comparison in the results during previous factor testing, we wanted to save this experimentation for last. For this factor, we predicted that increasing the amount of training data would reduce the risk of over-fitting, since the model would have more data to train over. We also estimated that there would be another point of diminishing returns, using our knowledge from previous factors. It is also predicted that there is a certain point where we don't have enough validation data to compare our results. The tables below show our findings.

0.95 Training : 0.05 Validation		
Run	Accuracy	Loss
1	0.78314	0.49888
2	0.77230	0.51224
3	0.77711	0.50879
Average	0.77752	0.50664

0.90 Training : 0.10 Validation		
Run	Accuracy	Loss
1	0.77871	0.50251
2	0.77804	0.50074
3	0.77851	0.49619
Average	**0.77842**	**0.49981**

0.80 Training : 0.20 Validation		
Run	Accuracy	Loss
1	0.77147	0.52714
2	0.77389	0.52181
3	0.77327	0.52905
Average	0.77288	0.52600

These findings support our thoughts from the preliminary experimentation, and enforce our decision to use a 0.9 training to 0.1 validation split. Our hypothesis was correct in determining there was a point of diminishing returns, and we already had the best amount of training data to train over. Our thoughts on increasing the training data also are shown, as there are several improvements in the 0.1 split over the 0.2 split. In further experimentation with more classes, we would like to continue to test results on the validation split to determine the best amount of data for different amounts of classes.

State-of-the-Art Comparison

To compare our results with the state of the art, we will want to review the overall results of (Cao, Wu, and Shen 2018) and (Eigen, Puhrsch, and Fergus 2014). We hope to be comparable to these results, as they are both validation accuracy and loss on their trained data sets, and since we are also training on our own data set, we only want to try and achieve the same accuracy. Their results are as follows:

State of the art			
	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
(Cao, Wu, and Shen 2018)	69.4%	92.4%	97.5%
(Eigen, Puhrsch, and Fergus 2014)	61.1%	88.7%	97.1%

Since (Cao, Wu, and Shen 2018) uses classification instead of regression, which is similar to our project, and it outperforms (Eigen, Puhrsch, and Fergus 2014), we choose to compare it to our results. Since we are not using an error rate for our results, it is hard to compare the results. Since we are looking to determine a rough estimate with minimal calculations in an indoor environment, we want to have a good accuracy while maintain our run time. Since we also run the last three frames through our model to determine each class, we can also assume an accuracy close to 95.09%, assuming a classification accuracy of 0.77842.

Using that last metric, we are confident that our results are performing close to the state of the art. We want to continue our work by adding classes while maintaining the same accuracy rate. We also want to modify the majority voting system for border cases where a section might fall into more than one class. Nonetheless, we feel that we have established a secure proof of concept to apply to the drone navigation problem.

Future Work

Some future objectives of our work include visual odometry, swarming, and creating a common framework. Odometry is important in ensuring the drones are reporting an accurate current location, and to make sure the drones are accurate in arriving to their destinations. Next, we want to be able to use swarming with our drones, so that we can accomplish tasks that require a fleet of drones. Finally, we want to create a common framework of our research, so that our results and methods can be replicated in other makes and models of drones. This is important because the majority of drone research is in very specific models of drones. This makes other findings hard to replicate, as each drone has different capabilities in hardware and software, as shown in our findings regarding EGO-Swarm (Zhou et al. 2020b).

Conclusion

In conclusion, this research aims to offer assistance in solving the autonomous vehicle navigation problem using Deep Learning. By using deep learning to skip the optical flow field step, we have reduced the computational time between frame collection and depth estimation image generation by 70%, and we hope to continue to reduce this time. We also want to expand our classification model into more than 2 classes, to gather accurate data about each drone's surroundings. In conjunction, we also want to achieve and maintain a depth accuracy of more than 80%. Once these are finished, we hope to expand our research into visual odometry, swarming, and creating a common framework for others to replicate for their own usage.

Acknowledgement

The work reported in this paper is supported by the National Science Foundation under Grant No. 2050919. Any opinions, findings and conclusions or recommendations expressed in this work are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

Bradski, G. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Brust, M. R., and Strimbu, B. M. 2015. A networked swarm model for uav deployment in the assessment of forest environments. In *2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, 1–6. IEEE.

Cao, Y.; Wu, Z.; and Shen, C. 2018. Estimating depth from monocular images as classification using deep fully convolutional residual networks. *IEEE Transactions on Circuits and Systems for Video Technology* 28(11):3174–3182.

Chollet, F. 2015. Keras.

Eigen, D.; Puhrsch, C.; and Fergus, R. 2014. Depth map prediction from a single image using a multi-scale deep network. *arXiv preprint arXiv:1406.2283*.

Hu, Y., and Meng, W. 2016. Rosunitysim: Development and experimentation of a real-time simulator for multi-unmanned aerial vehicle local planning. *Simulation* 92(10):931–944.

Loquercio, A.; Maqueda, A. I.; Del-Blanco, C. R.; and Scaramuzza, D. 2018. Dronet: Learning to fly by driving. *IEEE Robotics and Automation Letters* 3(2):1088–1095.

Sawalmeh, A.; Othman, N. S.; Shakhatreh, H.; and Khreishah, A. 2017. Providing wireless coverage in massively crowded events using uavs. In *2017 IEEE 13th Malaysia International Conference on Communications (MICC)*, 158–163. IEEE.

Skorobogatov, G.; Barrado, C.; and Salamí, E. 2020. Multiple uav systems: a survey. *Unmanned Systems* 8(02):149–169.

Sperling, M.; Bouteiller, Y.; de Azambuja, R.; and Beltrame, G. 2020. Domain generalization via optical flow: Training a cnn in a low-quality simulation to detect obstacles in the real world. In *2020 17th Conference on Computer and Robot Vision (CRV)*, 117–124.

Zhou, X.; Wang, Z.; Ye, H.; Xu, C.; and Gao, F. 2020a. Ego-planner: An esdf-free gradient-based local planner for quadrotors. *IEEE Robotics and Automation Letters*.

Zhou, X.; Wen, X.; Zhu, J.; Zhou, H.; Xu, C.; and Gao, F. 2020b. Ego-swarm: A fully autonomous and decentralized quadrotor swarm system in cluttered environments. *arXiv preprint arXiv:2011.04183*.

Author Index