FACEDAR: A CNN CAPABLE OF DETECTING PHOTOSHOPPED FACES

FACEDAR: A CNN CAPABLE OF DETECTING PHOTOSHOPPED FACES

A project report submitted in partial
fulfillment of the requirements for the degree of
Master of Science

By

Matthew Sullivan
University of Dallas, 2009
Bachelor of Science. Biology

April 2020
University of Colorado Denver

ABSTRACT

Our consumption of news is increasingly digital and social-media oriented. In the 2016 election year alone, over 37 million false news posts were spread on social media[1]. Many of these posts are accompanied by images which have been modified to fit the headlines they accompany. As consumption from these sources rise, so does the risk of exposure to false or misleading articles or “fake news”. In this study, I have designed a process in which Error Level Analysis is applied to a collection of modified and non-modified images of faces in order to detect the presence of image modification. These identified differences are then used to train a custom convolutional neural network to look for patterns indicating image manipulation. As image manipulation detection is a subjective field, this process is not intended to be an absolute indicator that an image has or has not been modified. But it is my intent to show that this method can be successfully used as another tool in a growing toolkit for people to combat misinformation.

This Project Report is approved for recommendation to the Graduate Committee.

Project Advisor:

____

Ashis Biswas, PhD

MS Project Committee:

____

Name2

____

Name3

TABLE OF CONTENTS

LIST OF FIGURES

1. Introduction

1.1 Problem

In 1840, Mr. Hippolyte Bayard staged a drowning photo depicting his apparent suicide on a photograph of his own design[2]. This is one of the earliest known "fake" photos, from an era when photography itself was a developing science. From William Mumler's spirit photos, to the Cottingley Fairies, to the Surgeon's Photo "proving" the existence of the Loch Ness monster, hoaxers have been manipulating images to elicit responses for almost 2 centuries[3,4,5]. While early manipulation was mostly a matter of photo composition and props, as time went on, techniques were developed to edit the photographs themselves, though it was much more of a painstaking task than it is today.

With the advent of digital photography, it has become much easier for the general populace to create such manipulations by modifying the images directly. Since it was first produced in 1988, Adobe Photoshop has become famous (and infamous) for its ability to create or modify images that are indistinguishable from an original to a casual observer. In some industries, use of Photoshop for post-processing images is recognized as a common practice, or even the "standard", which has led to counterintuitive instances where legitimate criticisms are questioned[6]. However, even in those industries saturated by image manipulation, such as advertising, overuse of image manipulation has been criticized as being inherently untruthful and creating a false representation of reality.

With the widespread adoption of the internet and social media, spreading disinformation has become almost trivial. In late 2016, "Fake News" took off in google searches at an exponential rate[7]. While the queries on the topic have slowed, they periodically return, and it seems we barely go a week without a new story or claim about widespread data manipulation and false narratives affecting our society on a large scale.

This is widely acknowledged as a growing problem, with 64% of Americans saying that false news has created “a great deal of confusion” and 24% saying “some confusion”[9]. This same poll shows that 39% of Americans are “very confident” and an additional 45% are “confident” they can differentiate fake news from real news. Unfortunately, in this case people’s perceptions do not accurately reflect reality. Another poll shows that 75% of Americans were unable to recognize fake news stories they were presented with[8].

Meanwhile, 40% of Americans polled reported social media as a “major” source of their news[8]. During the last few months of 2016, over 37 million false news posts related to the Presidential election were shared on social media[1]. With so much of our “news” presented as “Infographics” and memes shared on social media pages, we as a society have never been more vulnerable to image manipulation which accompanies many of these false stories in order to increase their credibility.

1.2 Project Statement

I propose that a convolutional neural network (CNN) trained using a dataset of manipulated and non-manipulated JPEG images could be used to classify an image as “real” (unmodified) or “fake” (modified). The CNN by itself is not a one-size fits all solution, but is intended to be an effective first line tool for the general populace to detect manipulated images.

1.3 Approach

The approach of this study is to train a custom designed CNN to distinguish between modified and unmodified images. For this project, the Real and Fake Face Detection Dataset created by the Computational Intelligence and Photography Lab, Yonsei University[10] was used. This dataset consists of 980 photoshopped images and 1081 unedited images. All images are 600x600, color, and stored in JPEG format.

The lossy JPEG compression format allows the use of a technique called Error Level Analysis (ELA), covered by Dr. N. Krawetz[11]. This technique saves a JPEG image at a lower quality and then compares the saved file to the original file. The resulting image shows the amount of error created by the save. The pixels should have a fairly uniform rate of descent, while any pixels introduced artificially will have a different rate of error due to the lossy compression.

These ELA images are fed into a custom CNN and train the model based on basic computer vision concepts. By comparing the pixels present in the images, the model will be able to detect the presence of altered data using a binary classification model.

1.4 Organization of this Project Report

Chapter 2 covers two key concepts needed to understand this paper, namely convolutional neural networks and error level analysis, as well as a literature review of papers and articles relevant to computer vision, image analysis, and image manipulation. Chapter 3 discusses the methods and architecture of the model, along with the software, hardware, libraries, and datasets used in this project, for the purposes of enabling recreation of the findings in this paper. Chapter 4 contains implementation details, results, and analysis of the findings. Chapter 5 will include the conclusions drawn from the results, the limitations of the study, and future work.

.

2. Background

2.1 Key Concepts

To fully understand this project report, the reader must be familiar with two key concepts. One is the convolutional neural network, the other is the principle of Error Level Analysis and how it relates to image manipulation.

2.1.1 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a class of deep neural networks modeled on the visual cortices of cats and monkeys[12]. These biological processes were translated into computer science by K. Fukushima in his paper “Neocognitron”[13]. A CNN consists of multiple layers, including an input layer, an output layer, and at least one (but usually several) “hidden” layers, which the user does not have direct control over. The information fed to a CNN can be multi-dimensional (for instance, a black and white image is commonly input as 2d, and has corresponding 2d filters and pooling algorithms, while a color picture is a stack of 3 2d layers for the 3 color combinations of pixels).

These hidden layers come in several types, the primary of which is known as a convolutional layer. This layer applies a number of filters across an array of numbers from the initial layer. These filters are convolved across the information sequentially and the dot product of data and the filter is stored as the activation map of that filter. These activation maps collectively form the output of this layer. These activation maps are then fed through an activation function, which is selected by the user in order to refine the results.

As the activation filters increase the depth of the graph dramatically, a solution to shrink the initial dataset is almost always a necessity. Originally, Fukushima proposed spatial averaging to reduce sample size[13]. In 1990, J. Weng et al. implemented a pooling layer as an alternative to spatial averaging[14]. A pooling layer is a layer which also steps across the data, and combines the data in a section (or pool) and returns a singular value for the pool such as an average, the minimum, or the maximum value. This allows the input image to be compressed, while retaining the orientation of the feature in relation to the other features.

Convolutional and pooling layers make up most of the hidden layers of a CNN. Once convolution and pooling is done (several times, usually) to detect features and associate them with other features in the data, the results are flattened into a 1d array and fed to a fully linked layer, which is where the "reasoning" is done on the information that has been extracted and condensed. The results of these layers are finally sent to the output layer, where the user can extract the prediction.

Another key concept to the CNN is the idea of backpropagation, or the process by which the network "learns" from training data. The connections between each neuron have weights, which effect the answer in the output layer as the data flows through the network. The network's prediction is compared to the ground truth of the sample, and this comparison is used to adjust the weights to each node connected to the output layer. This process is repeated for each layer in reverse order all the way back through the neural network. In this fashion, adjustments are propagated backwards through the neural network.

While neural networks were proposed in the 80's, they have only recently become practical tools, as more powerful processors, the availability of GPUs as general purpose parallel processors, and large data sets to train the networks have become much more available in the last 10 years.

2.1.2 Error Level Analysis

Error level analysis (ELA) is a method of detecting changes in JPEG images based on the changes in the pixels due to the compression method JPEG uses.
JPEG uses a lossy data compression model which compresses groups of 8x8 pixels using discrete cosine transform[15]. This allows JPEG images to be adjusted to scale quality vs size. In practical application, it means that we can measure the change in these groups of pixels to see if they are changing at the same rate. The effects are cumulative, so saving a JPEG at 90% quality (which implies 10% data loss) twice is the equivalent of saving once at 81%[11]. So if new pixels are introduced, either by editing or by copy/pasting from another image, they will have a different compression rate than the rest of the image.

ELA uses this attribute of JPEG images by saving a new copy of the picture we are examining at a compressed size, then comparing the differences between the original and the newly saved image. This comparison can be visualized as an image showing the degree of differences between the pixels.

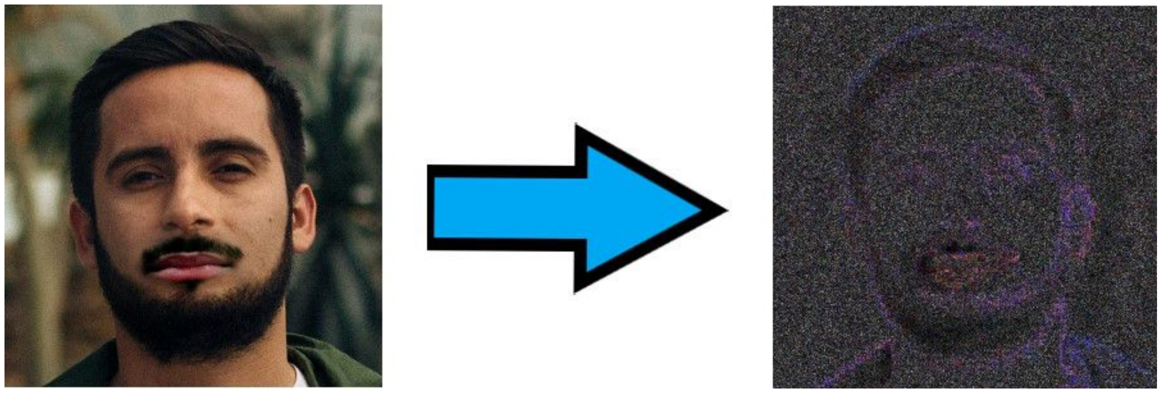


Figure 1: A photoshopped image where the mouth has been modified

ELA has several drawbacks, one of which is the fact that it is heavily based on the JPEG lossy compression algorithm and will not work on lossless images without conversion. Another issue is the fact that numerous re-saves could obscure modified elements, as their pixels' compression ratio will eventually approach the same level as the rest of the image. However, a tool that can detect even amateur editing would be incredibly helpful in the fight against misinformation.

2.2 Related Work

2.2.1 Convolutional Neural Networks

Convolutional neural networks and their impact on image analysis is an incredibly large field of study. While the concept was first proposed in "Neocognitron" in the 80's[13], they were rarely used until the 2000's, when advances in hardware made CNNs widely applicable to the larger computer using community[14].

Fukushima et al. proposed to model the brain's recognition by connecting S-cells (input) and C-cells(output) connected by an unmodifiable connection. The C-cells are then connected to the input of the S-cells by a modifiable connection. The S-cells are all groups into a 2d layer, which is able to detect features from the input layer. The weights of the connections between one layer of C-cells and the next layer of S-cells are changed based on if the layer matched the value expected. This is a very early explanation of the convolution process over an image.

Building on the foundations of Fukushima, several refinements have become commonplace. Max-pooling, first described by K. Yamaguchi et al in “A neural network for speaker-independent isolated word recognition”, describes a processing event between two neural networks that selects the maximum value from multiple outputs[16]. This concept was brought into the realm of computer vision when it was incorporated into the Node Reduction Layer of J. Weng et al's Cresceptron design in “Learning Recognition and Segmentation of 3-D Objects from 2-D images”, which similarly takes the max value of a set of inputs and inputs it into the next layer. As discussed in 2.1.1, most CNNs use max-pooling to reduce data size when dimensionality is increasing.

There is an incredible amount of papers concerning CNNs performing image classification. While this brief overview can barely begin to scratch the surface, W. Rawat et al’s “Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review”[17], which compiles over 300 published papers, is highly recommended to those who would like a more in-depth review. In more recent years, CNN’s have been employed more frequently to detect manipulated images, using methods such as edge detection and pattern recognition. Many of these propose new CNN architectures. Unfortunately, the hardware limitations in section 3.2

prevented me from recreating a truly cutting edge CNN. Which is why the related works are foundational, rather than focusing on a novel approach.

2.2.2 Error Level Analysis

While Error Level Analysis is a commonly used tool for image manipulation detection, due to its subjective nature, it has not been thoroughly examined by the scientific community, and the literature covering it is somewhat limited.

The technique is first described by N. Krawitz in "A Picture's Worth..."[11]. In section 3.4.2, he details how the JPEG compression algorithm [15] generates error in 8x8 pixel blocks, and proposes that the differences between modified and "real" pixels could be detected by saving an image at a known quality, then comparing the differences between the two pictures. Z. Weiguo et al[18] used ELA on images they generated themselves mimicking the process of deepfaked face swaps in video. By generating images using the same principles as the deepfake (facial feature mapping, rotation/resize/orientation of the new face to align with the original, adding gaussian blur and superimposing the new facial image onto the old one), they were able to train their CNN to 97% accuracy.

W. Wang et al[19] used a combination of ELA (called "JPEG Compression Noise" in the paper) and principle component analysis to generate masks on modified images where suspected modification took place. Their proposed algorithm was not 100% successful, but it did prove to be an effective method of identifying potential areas of altered images. Also of note was the fact that they used lossless .TIFF format photos and converted them to JPEG files for their experiment, indicating that the technique can be effective even when the original images are not stored in the JPEG format.

3. Architecture

3.1 High Level Design

First, the dataset of edited/non-edited photos had to be converted to ELA images. After this was completed, it was necessary to design a convolutional neural network capable of recognizing the differences between the ELA of the “fake” and “real” images. As discussed in section 2.1.1, CNN structure is fairly straight-foward, consisting of an input layer connected to one(or more) convolutional and max-pooling layers, which are then flattened and connected to a fully connected layer (or layers) before being connected to an output layer. While the basics are simple to understand, the vast majority of the work involves choosing appropriate layer sizes and dimensions for your dataset, and there are very few (if any) hard and fast rules. The baseline design was adapted from a Keras CNN tutorial[20], but there were dozens of proposed architectures, and a half dozen underwent significant tests before landing on the final architecture delivered in this report. Additionally, hardware limitations detailed in section 3.2 played a large role in the design process.

3.2 Implementation

3.2.1 Hardware

The project hardware was computed on a MSI laptop PC with the following specifications:

- Intel Core i5-4200H Processor @ 2.80 Ghz
- NVIDIA Geforce GTX 860M / 2GB GDDR5 (CUDA/cuDNN enabled)
- 8GB DDRIII(L) RAM

3.2.2 Software

The CNN was developed on a Windows 10 Laptop using Anaconda 1.9.7/Juypter Notebook 6.0.3 using the following Python 3.6 libraries:

Tensorflow 2.1.0

Numpy 1.18.1

cudnn 7.6.5

keras-base 2.3.1

keras-gpu 2.3.1

matplotlib 3.1.3

seaborn 0.9.0

pandas 1.0.3

With all dependencies installed.

Image conversion and transformation was implemented using the Pillow 7.0.0 library, also implemented in a Python 3.6 Jupyter notebook

4. Methodology, Results and Analysis

4.1 Methodology

The dataset was obtained via Kaggle[10]. Using the Pillow library (which is a fork of the common Python Image Library (PIL)), the entire dataset was converted using a `convertToELA` function based on the method covered by Krawetz which was modified from an implementation found on github[21]. (Please see appendix 1 for code examples). This function saves a temporary JPEG image for each photo at a specified compression ratio, compares the temporary image and the original image and returns this difference, which is the ELA of the image. Several different datasets at various compression ratios and brightness were prepared, but I was unable to determine a significant difference in learning rate, loss or accuracy between any of the datasets. For the results presented in this report, the training was done at 90% quality with 2x brightness enhancement.

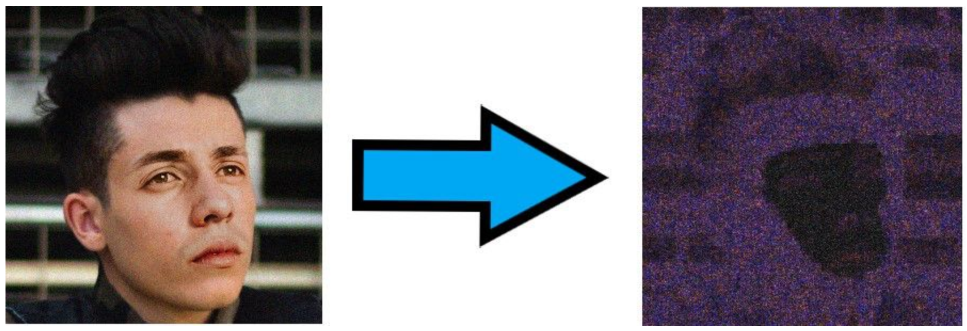


Figure 2: In this image, the eyes, nose, and mouth has been replaced

These images were then saved, and every fifth image pulled to separate training data from validation data. The data was organized so that each converted dataset had a train/validation folder, with both the training and validation folders each containing a “fake” and “real” folder.

The first designs for the CNN were based on a simple CNN in the keras tutorial[20], adjusted to accommodate a much larger input size (600x600). The data was not resized because it was unknown what data or artifacts would be lost, and, since JPEG compression works on 8x8, it is believed that the initial frame could be quite large, allowing relatively fast data size reduction while still getting the large features. Once the initial design was completed and loaded, the model was fitted with the ELA generated images, but it was found that the model quickly (within 20 epochs) overfit to the data.

To overcome overfitting, first the experiment was repeated with some smaller CNN designs, but it was found that any significantly smaller designs would require extensive shrinking of the input data, or several max-pooling layers before many convolutions were done. The resulting architectures did not show any indications that they were learning, as the validation data never wavered much more than a few percentage points from the baseline of 50, which could easily be accounted for by a difference in the random sampling of the two classes from the batches.

Since smaller architectures were not working, it was clear that the dataset would have to be augmented. In the first attempt, the data was augmented on-site (using a custom `dataAugmentation()` function which flipped the image horizontally, and then rotating both the original and the flipped image by 90 degrees 3 times. The resulting set of images was eight times larger than the original. However, using the default `model.fit()` method, the entire dataset must be loaded into the GPU RAM, with the addition of all of the layers of the neural network. Once the dataset was increased, it was impossible to store in the GPU memory (2GB per section 3.2) with an architecture of adequate size.

Due to these limitations, it was decided to use Keras' `model.fit_generator()`. This function uses the `ImageDataGenerator()` function, which preprocesses and loads samples by batch to feed into the neural network. Once batched, the samples are loaded batchwise into the network. Originally, the process was attempted using the generated augmented data, with `ImageDataGenerator()` only normalizing the pixel data (0-1) rather than (0-255), but after over 200 epochs and 12 hours of training, the model had reached a plateau. The model's loss wasn't decreasing, training accuracy was hovering in the mid 60%, and validation accuracy was stuck in the mid 50%, which made it questionable if it was learning any data at all, or if the proportion of “real” to “fake” images were simply fluctuating.

It was then decided to try to use the built in data augmentation features of `ImageDataGenerator()` in an attempt to augment the data without losing vital features of the data. It was found that, with data augmentation as well as scaling taking place, the amount of time increased about 400%, with the additional time presumably spent augmenting the data. Therefore, in the interests of time, the epoch length was shortened from 8000 total samples per epoch to 2000, with the validation also being reduced from 800 to 200 samples.

After much experimentation, the current architecture was designed, which was a balance between size and efficiency. Upon testing it was found that a much larger neural network did not increase accuracy, yet the architecture detailed here seemed to be able to overcome overfitting (see section 4.2). It was too large to be captured in one figure with the CNN diagram generator used to create these figures[22], so they have been split it into 3 parts. Please see the appendix for the `model.summary()` of the CNN design for a text-based blueprint.

Figure 3: The front half of the CNN (Please see appendix for full size image)

Figure 3 shows the first two convolutional layers, as well as the first 2 max-pooling layers. Convolution layer 1 was size 11x11, 16 features, with a stride of 2. These hyperparameters were chosen in order to capture high level features (since the JPEG compression algorithm works on groups of 8x8 pixels it was hypothesized that the first convolutional filters could be slightly larger than that and still see features) while also taking into the account the necessity of reducing the layer size quickly due to hardware limitations. Hence, the first max pooling layer is also rather aggressive, with a 4x4 grid with stride 2.

Convolutional Layer 2 was slightly smaller, still 5x5 with 32 features and a stride of 2. This was thought to be a reasonable compromise between large features and smaller ones. Max-pooling 2 was a traditional 2x2, though it still had stride of 2 to reduce size.

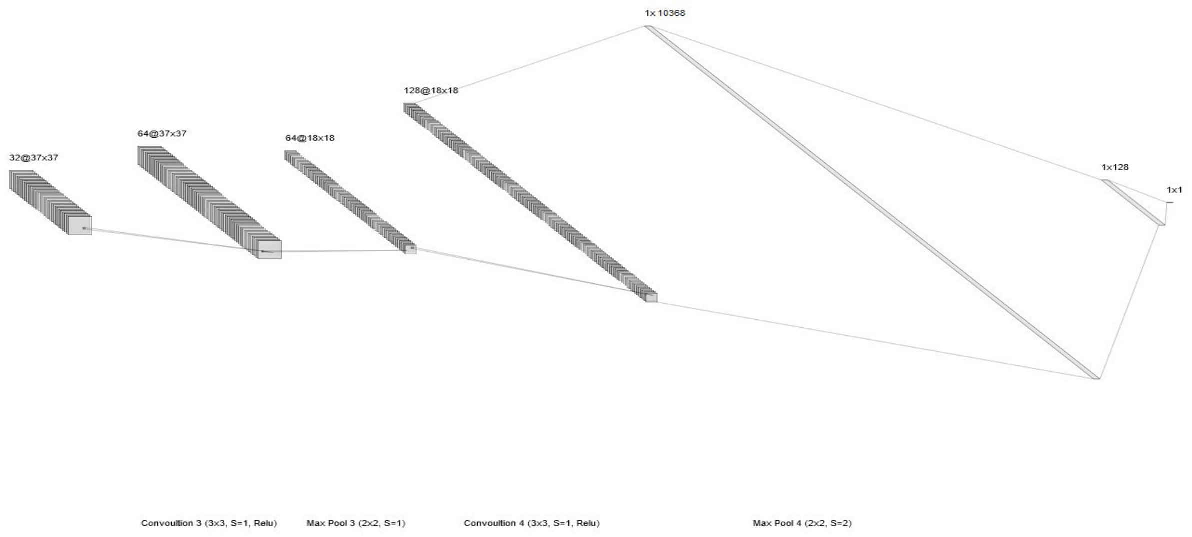


Figure 4: The back half of the CNN (Please see appendix for full size image)

Convolutional layers 3 and 4 are identical, 3x3 grids with 64 and 128 features, respectively. Max pooling layers 3 and 4 are also 2x2 with stride 1. The result of the final max-pooling layer was then flattened to a 1d array of length 10368. This layer was then connected to a fully connected dense layer of 128 nodes with an aggressive dropout rate (0.5) in the interest of reducing overfitting. These nodes all connected to a single output node with a sigmoid activation function.

A note about the activation functions. Due to its efficiency, simplicity, and proven effectiveness in computer vision tasks, ReLU was chosen as the activation function for all convolutional layers, as well as the 128 dense layer. The final node was of course given a sigmoid function, as this is a binary classifier.

Testing took place in batches of 32, with epochs ranging from 20 to 100. Weights were saved between shorter trainings to allow for smaller batch testing when necessary.

All experiments were run on a computer with the hardware and software specifications as detailed in section 3.2.

4.2 Results

The results from the model after the first 100 epochs run are as follows.

Figure 5: Training and Validation Accuracy for first 100 epochs

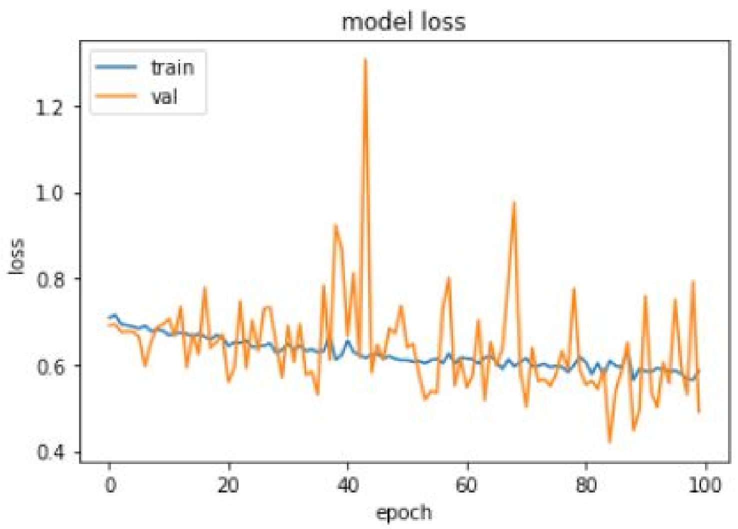


Figure 6: Training and Validation Loss for the first 100 epochs

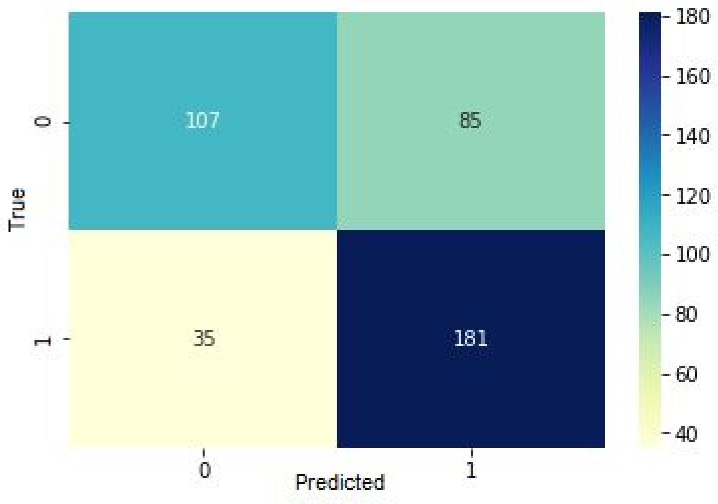


Figure 7: The Confusion Matrix for the CNN

Figure 8: The Classification Report

4.3 Analysis

As demonstrated by Figure 5, the model gains accuracy steadily throughout the first 100 epochs of testing, and ends with a value of 71% accuracy. While the validation fluctuates compared to the training accuracy, it still trends upwards until it is higher than random variances in validation data selection statistically account for.

Meanwhile, figure 6 shows the training loss is steadily decreasing. This, combined with the fact that the validation loss is decreasing as well, shows that the model is learning and not overfit. The final value of the evaluated model is 0.49.

The fact that the validation and training curves are not as steady can be attributed to the relatively small sample size of the data set. Both could be smoothed with a larger dataset and longer epochs to train (a trade-off of the time available) However, both the accuracy and the loss graph indicate that our CNN is learning how to predict modified and non-modified images.
The results curves, especially the volatility of the validation curves against the training curves, could be in part attributed to the dataset. Ideally, a larger dataset would be collected or created to train a better generation of image manipulation detectors, as this would alleviate many of the issues encountered in this report.

Likewise, the confusion matrix shows a great deal of promise. Figures 7 and 8 show 71% accuracy when tested after only 100 epochs. When the CNN does make an error, the precision and values from the confusion matrix show that it is usually more accurate about assigning a “fake” value (at 75%), and is more likely that borderline cases will be assigned as “real”. While we would ideally have no error at all, if we are going to design such a device as a forensic tool, it may prove to be better to give the benefit of the doubt (subject to further analysis, of course) to images, especially if there is legal precedent. The cost of a false positive could greatly outweigh the cost of a false negative. Of course these issues can be alleviated by returning the true output from the CNN to the user, as the CNN by default puts out percentages of likelihoods, rather than absolute values for predictions.

5. Conclusions

5.1 Summary

Despite the many challenges presented by this project, it has been demonstrated that it is possible to begin to train a neural network to recognize images that have been modified by a team of human experts. I have also identified many issues with relying on this technique exclusively, including the non-effectiveness of ELA on non lossy datasets, the difficulty of differentiating some images, even to a human observer, the issue of an expert resaving images, and the limited dataset easily available for training all have contributed to the slow training of this model. However, the fact that I have demonstrated that even a relatively low powered and simple CNN can examine these complex images and make relatively accurate predictions shows great potential for the use of machine learning to combat disinformation.

5.2 Potential Impact

My model shows that, given a dataset of images modified by human experts[10] it is possible to design a CNN capable of detecting these images to a reasonable degree. Of course, there are limits to our CNN. For example, a manipulator aware of the limits of ELA could in theory completely void the test by re-saving the image multiple times after manipulating it, effectively bringing all JPEG compression down to baseline and showing no difference at all using ELA. While it may be unlikely that an average legitimate image would be resaved that many times, it is not out of the realm of possibility, so there could be no definite determination. Also, a very clever and aware manipulator could find a suitable modification image with a similar degradation value for its JPEGs to the source image, leading to false negatives predictions. While this is a drawback, our tool is intended to be used in conjunction with other analysis tools and likely expert analysis to make more credible judgments about the validity of an image.

Despite the many drawbacks, I believe that it is extremely encouraging that it was possible to train a CNN to detect modified images with limited resources, and it shows that CNNs could be a great strength in the future in the fight against misinformation.

5.3 Future Work

There are several ways that this work could be improved and implemented in the future. This project was limited in three ways: Dataset (size and makeup), hardware, and scope.
With a larger, more generalized dataset (not just of human faces), it may be possible to expand the predictive ability of this CNN (or a similar one) to a general purpose detection system for manipulated images. While the legitimate uses of image manipulation would always prevent this from being catchall in image manipulation detection, it could definitely provide awareness on a large scale to the general population.

With a more powerful hardware system, more time to train, and a larger dataset, it should be possible to train a CNN to have a much larger accuracy than 71%.

Additionally, while I was unable to determine any significant differences when testing the various datasets with differing compression ratios, it is possible that these may have a larger impact than I suspect, and would be a good avenue for future research.

The most successful deployment of the technique detailed in this report would be as a part of a suite of tools attempting to highlight data manipulation. ELA analysis, while it can be very effective, is but one of many techniques employed by image manipulation detection experts currently. To get the most accurate “picture” of whether an image is manipulated or not, a composite approach must be adopted. This toolsuite could definitely be a great weapon in the war currently being waged against misinformation.

References

- [1] H. Allcott et al, “Social Media and Fake News in the 2016 Election”, *Journal of Economic Perspectives*—Volume 31, Number 2—Spring 2017—Pages 211–236
- [2] M. Sapir "The Impossible Photograph: Hippolyte Bayard's Self-Portrait as a Drowned Man." *MFS Modern Fiction Studies*, vol. 40 no. 3, 1994, p. 619-629. *Project MUSE*, doi:10.1353/mfs.1994.0007.
- [3]L. Kaplan, “The Strange Case of William Mumler*”. University of Minnesota Press*. ISBN 978-0-8166-5157-3, 2008
- [4] M. Magnusson, “Fakers, Forgers & Phoneys”, *Mainstream Publishing*, ISBN 1-84596-190-0, 2006
- [5] R. P. Mackal, “The Monsters of Loch Ness”*, Swallow Pr.*, ISBN 978-0-8040-0704-7, 1976
- [6] I. Steadman*,* “Fake' World Press Photo isn't fake, is lesson in need for forensic restraint”*, WIRED,* 5/16/13 https://www.wired.co.uk/article/photo-faking-controversy. last accessed 4/24/2020
- [7] Google Trends, “fake news” https://trends.google.com/trends/explore?date=today%205-y&geo=US&q=fake%20news, last accessed 4/22/2020
- [8] C. Silverman, “Most Americans Who See Fake News Believe It, New Survey Says”, *Buzzfeed News*, 12/6/16, https://www.buzzfeednews.com/article/craigsilverman/fake-news-survey#.mlxj9PKPG4h ttps://www.buzzfeed.com/craigsilverman/fake-news-survey?utm_term=.rmnKLVXVYv, last accessed 4/22/2020
- [9] M. Barthel et al, “Many Americans Believe Fake News Is Sowing Confusion”, 12/15/16, *Pew research Center*, https://www.journalism.org/2016/12/15/many-americans-believe-fake-news-is-sowing-confusion/?utm_source=Pew+Research+Center&utm_campaign=e080af4f66-EMAIL_CAMPAIGN_2016_12_14&utm_medium=email&utm_term=0_3e953b9b70-e080af4f66-400229353, last accesssed 4/22/2020
- [10] Real and Fake Face Detection, Computational Intelligence and Photography Lab, Yonsei University, https://www.kaggle.com/ciplab/real-and-fake-face-detection, last accessed 4/22/2020
- [11] N. Krawetz, “A Picture’s Worth… Digital Image Analysis and Forensics,” *Black Hat Briefings USA,* Las Vegas, Nevada, July/August, 2007
- [12] David H. Hubel et al.. “Brain and visual perception: the story of a 25-year collaboration”. *Oxford University Press,* US. ISBN 978-0-19-517618-6, 2008
- [13] K. Fukukshima “Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position”, *Biol. Cybernetics 36*, 193-[13202, 1980
- [14] J. Weng, et al..”Learning recognition andsegmentation of 3-D objects from 2-D images”. *InProc. IEEE 4th Int’lConf. Computer Vision, pages 121–128*, May 1993.
- [15] G. Wallace et al, “The JPEG Still Picture Compression Standard”, *IEEE Transactions on Consumer Electronics, Vol. 38, No. 1*, February 1992

- [16] K. Yamaguchi[et al. "A Neural Network for Speaker-Independent Isolated Word Recognition" *First International Conference on Spoken Language Processing (ICSLP 90)*. Kobe, Japan, November 1990
- [17] W. Rawat, et al. "Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review." *Neural Computation. 29. 1-98. 10.1162/NECO_a_00990*. 2017
- [18] Z. Weiguo, et al., "Exposing Face-Swap Images based on Deep Learning and ELA Detection" *5th International Electronic Conference on Entropy and Its Applications,* Online, MDPI Sciforum Platform, November 2019
- [19] W. Wang et al. Tampered Region Localization of Digital Color Images Based on JPEG Compression Noise". *6526. 120-133. 10.1007/978-3-642-18405-5_10*, 2010.
- [20] F. Chollet, "Building Poweriful Image Classification Models Using Very Little Data" *Keras Blog* 6/5/16 https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html, retrieved 4/22/2020
- [21] E. Cheslack-Postava, "Quick, simple implementation of Error Level Analysis" *Github,* https://gist.github.com/ewencp/3356622, last accessed 4/24/2020
- [22] LeNail, Alexander.. "NN-SVG: Publication-Ready Neural Network Architecture Schematics." *Journal of Open Source Software. 4. 747. 10.21105/joss.00747*. 2019 http://alexlenail.me/NN-SVG/LeNet.html, last accessed 4/24/2020

Appendix A. Code Examples

ConvertToELA:
```
def convertToELA(origin, destination, enhancement, q, suffix):
	for image in os.listdir(origin):
	imageR = Image.open(origin+image)
	tempName=image[:-4]+'_temp.jpg'
	imageR.save(origin+tempName, 'JPEG', quality=q)
	temp = Image.open(origin+tempName)
	ela=ImageChops.difference(imageR,temp)
	extrema = ela.getextrema()
	max_diff = max([ex[1] for ex in extrema])
	scale = 255.0/max_diff
	ela_im = ImageEnhance.Brightness(ela).enhance(scale*enhancement)
	ela_im.save(origin+image[:-4]+suffix, 'JPEG')
	shutil.move(origin+image[:-4]+suffix, destination+image[:-4]+suffix)
	os.remove(origin+tempName)
```

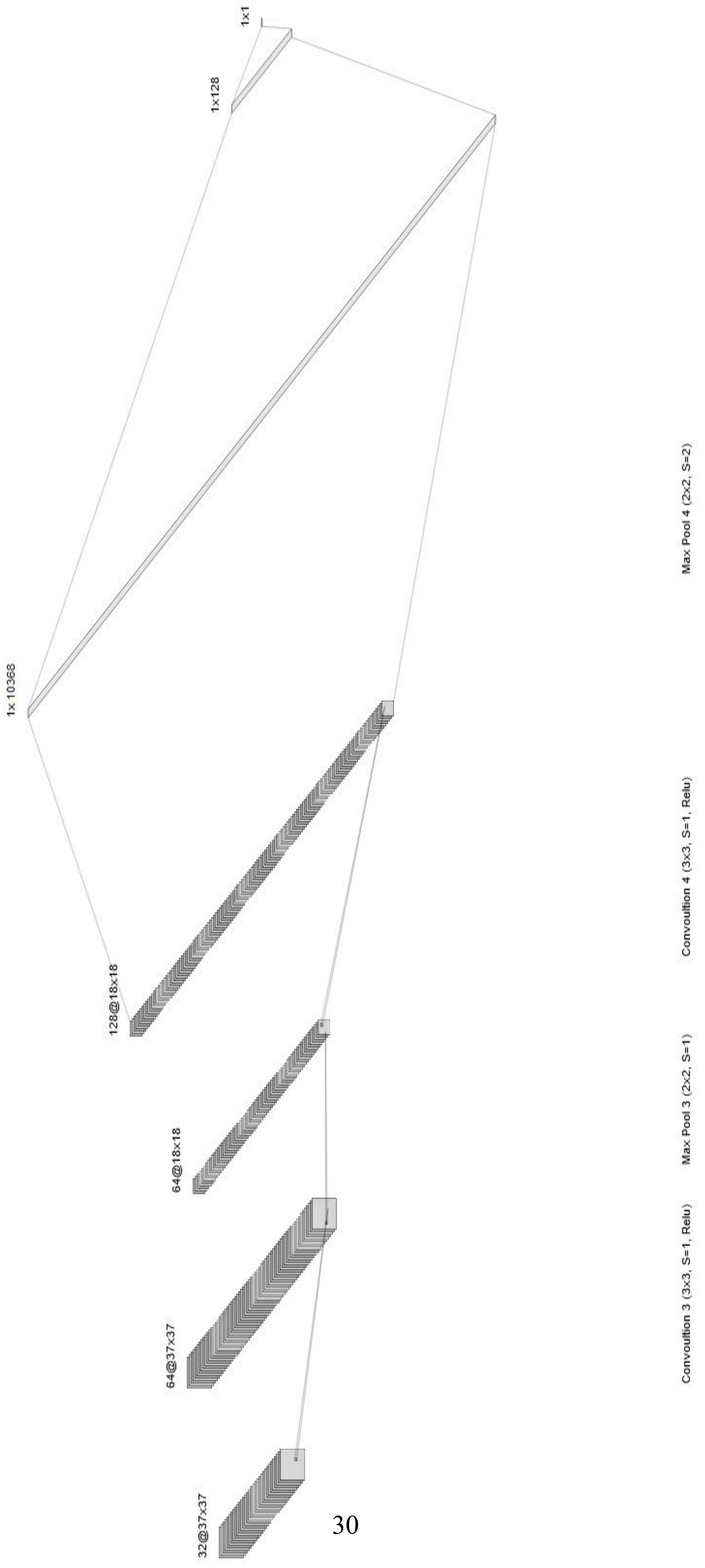
dataAugmentation:
```
def dataAugmentation():
	os.chdir(os.path.abspath(r'/Users/oneey/Desktop/Matt
Stuff/real_and_fake_face/converted80/test/fake'))
	for         file         in         os.listdir(r'/Users/oneey/Desktop/Matt
Stuff/real_and_fake_face/converted80/test/fake'):
	fileName=file[:-4]
	im=Image.open(file)
	im_90=im.rotate(90)
	im_90.save(fileName+"_90.jpg", format='JPEG')
	im_180=im.rotate(180)
	im_180.save(fileName+"_180.jpg", format='JPEG')
	im_270=im.rotate(270)
	im_270.save(fileName+"_270.jpg", format='JPEG')
	imr=im.transpose(Image.FLIP_LEFT_RIGHT)
	imr.save(fileName+"_flip.jpg", format='JPEG')
	imr=Image.open(fileName+"_flip.jpg")
	imr_90=imr.rotate(90)
	imr_90.save(fileName+"_flip90.jpg", format='JPEG')
	imr_180=imr.rotate(180)
	imr_180.save(fileName+"_flip180.jpg", format='JPEG')
	imr_270=imr.rotate(270)
	imr_270.save(fileName+"_flip270.jpg", format='JPEG')
```

finalDataAugmentation
```
train_datagen = ImageDataGenerator(rescale=1. / 255,
	rotation_range=20,
	width_shift_range=0.2,
	height_shift_range=0.2,
	horizontal_flip=True,
	shear_range=0.2,
	zoom_range=0.2)
```

APPENDIX B: OUTPUT/CHARTS

Final Model Architecture

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_15 (Conv2D)           (None, 300, 300, 16)      5824      
_________________________________________________________________
activation_22 (Activation)   (None, 300, 300, 16)      0         
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 150, 150, 16)      0         
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 150, 150, 32)      12832     
_________________________________________________________________
activation_23 (Activation)   (None, 150, 150, 32)      0         
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 75, 75, 32)        0         
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 75, 75, 64)        18496     
_________________________________________________________________
activation_24 (Activation)   (None, 75, 75, 64)        0         
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 37, 37, 64)        0         
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 37, 37, 128)       73856     
_________________________________________________________________
activation_25 (Activation)   (None, 37, 37, 128)       0         
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 18, 18, 128)       0         
_________________________________________________________________
flatten_4 (Flatten)          (None, 41472)             0         
_________________________________________________________________
dense_10 (Dense)             (None, 128)               5308544   
_________________________________________________________________
activation_26 (Activation)   (None, 128)               0         
_________________________________________________________________
dropout_7 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_11 (Dense)             (None, 128)               16512     
_________________________________________________________________
activation_27 (Activation)   (None, 128)               0         
_________________________________________________________________
dropout_8 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 129       
_________________________________________________________________
activation_28 (Activation)   (None, 1)                 0         
=================================================================
Total params: 5,436,193
Trainable params: 5,436,193
Non-trainable params: 0
_________________________________________________________________
```


32@37x37
64@37x37
64@18x18
128@18x18
1x 10368
1x128
1x1
Convoultion 3 (3x3, S=1, Relu)
Max Pool 3 (2x2, S=1)
Convoultion 4 (3x3, S=1, Relu)
Max Pool 4 (2x2, S=2)