**Math 156 Final Project**

The Quest for Truth: Detecting Artificially Generated Faces using CNNs

By Ryan Kawamura, Alyssa Liu, and Sarah Sotoudeh

## 

## **1 Abstract**

With the rapid advancement of generative artificial intelligence, distinguishing authentic images from those that are artificially generated is becoming an increasingly pressing problem to secure a notion of visual truth. We attempt to address this challenge by building a Convolutional Neural Network (CNN) to detect real faces from fake faces. During training, we experiment with pooling type, batch size, specific optimizer, learning rate, and dropout to pick our most accurate model. Our best model has a peak testing accuracy of 67%. Although this is not incredibly accurate, our model does serve as a meaningful step towards detecting false images.

**2 Introduction**

The dawn of artificial intelligence has sparked excitement, curiosity, and uncertainty amongst society. In the early 2010s, people were amazed by Apple’s voice assistant Siri’s ability to report the weather and complete a simple web search—tasks which we now consider quite elementary today. Only a decade later, generative AI models have been developed which hold more knowledge than a single human could ever imagine in their entire life. Alongside this development, many advancements have been made specifically in the field of image generation [1]. With only a few words, anyone can now effortlessly summon highly realistic and intricately detailed visuals, transforming the fields of design, media, and marketing. Curious what a dog riding a unicycle while drinking lemonade looks like? Leverage any of the publicly available generative AI models, and within a few seconds, you will find out. Unfortunately, generative AI is not all fun and games; there are major societal consequences that are inescapable.

When any image can be generated at any time by anyone, the line between what is real and what is not begins to dissipate. In essence, it becomes increasingly more difficult for any individual to discern the truth. This is particularly concerning when considering images and videos of human faces. False human portrayals raise major alarms regarding issues of identity manipulation, privacy violations, and the creation of false narratives which threaten to undermine any trust society has in visual media [2].

One prime example at the forefront of many concerns is deep fakes, which are highly convincing digital manipulations of image or video data that makes it appear as though someone is saying or doing something that they never actually did. In 2019 [3], a video of politician Nancy Pelosi intoxicated and stumbling over her words at an official White House event went viral [4]. A year later, a sexually explicit video of Kristen Bell circulated the internet [5]. In 2023, a video of Tom Hank was spread all across the web, portraying the actor in a series of advertisements for an obscure dental plan [6]. That same year, a video of Elon Musk encouraged people to enter their personal information into specific websites in exchange for a supposed cryptocurrency giveaway. The one thing these videos all had in common? They were not real. Everyday people had leveraged deep fake technology and successfully manipulated public perception while risking the reputations of real people.

Naturally, the question arises: how can we detect whether any image or video is real or fake? In this study, we examine this issue through leveraging facial image data. We analyze a dataset which contains two groups of images: real faces and artificially generated faces. We then build and train CNNs to construct a model that can distinguish whether or not a face is real. By doing so, we examine the abilities (and limitations) of existing machine learning infrastructure to detect artificially generated content. We hope that our learnings bring awareness to the importance of detecting fake human portrayals and make a productive step towards this successful detection.

**3 Background**

**Early Image Processing Techniques.** In the early stages of fake face detection, many papers and research efforts were aimed towards developing and refining methods that could effectively identify manipulated images using basic image processing techniques as this was a growing problem. Some examples of techniques and models implemented were Error Level Analysis [7], Frequency Analysis [8], Lighting Consistency Analysis [9], and Texture Analysis [10]. These techniques laid the groundwork for deepfake detection models and were instrumental in establishing the foundation of methodologies and knowledge for models used today. They contributed to the deep understanding of various features and data that could help classify fakes, such as texture, motion and compression.

**Generative Adversarial Networks (GANs).**  Before presenting more information on the background of deepfake detection methods and techniques, it is important to understand the threat that manipulated images cause and how they are created. Deepfake videos and images are created by taking images and videos and combining them using Generative Adversarial Networks to create a new image/video. GANs consist of two key components: a generator neural network and a discriminator neural network. Yada *et al.* describes how these two neural networks work together such that the generator takes the images and videos and combines them to create a new, realistic image and the discriminator evaluates the new image to be able to distinguish between manipulated and real images [11]. The creation of these realistic fake images can result in very harmful situations such as defamation, catfishing, and misinformation.

**SVMs for Deepfake Detection.** Recent developments in Machine Learning lead to Support Vector Machines (SVMs) becoming a staple in the world of machine learning for its robust classification abilities. In the case of deepfake detection, SVMs play a critical role in distinguishing between real and fake images [12]. Through the use of classical frequency analysis, SVMs are able to extract features from the image input data to analyze and effectively classify the image. Using SVMs made vast contributions and unlocked new potentials of deepfake detection by creating a model to accurately detect fake images with less training data. This stems from SVMs’ ability to construct an optimal hyperplane that can differentiate between real and fake images.

**Recurrent Neural Networks & Long Short Term Memory.**  Another method Güera *et al.* utilizes to detect manipulated images and videos is through the use of recurrent neural networks and long-short term memory. Their strategy involved using a CNN to extract important features from each frame of a suspicious video and pass those features to an LSTM for analysis where they would then calculate the likelihood that the video was manipulated or not [13]. Güera *et al.* utilized a “2048-wide LSTM unit with 0.5 chance of dropout” with “a softmax layer to compute the probabilities of the frame sequence being either pristine or deepfake” [13]. This work presented a new way to accurately detect manipulated images and videos in an efficient and effective manner.

**4 Dataset Description**

The dataset that was used to train and validate this model comes from Yonsei University. The downloaded zip file contains 2,041 face images, separated into two folders, called training\_fake and training\_real. Training\_fake contains 960 images of artificially-generated photoshopped face images, and training\_real contains 1081 photos of real faces. For each fake face, the features are gathered from a variety of real faces. The eyes, nose, mouth, or whole face of these fake faces come from separate faces.

Each image in the fake face folder is labeled with a difficulty, either easy, mid, or hard, a unique image number, and then four binary digits that indicate which part of the faces were replaced with features from other faces. The first binary digit corresponds with the left eye, the second with the right eye, the third with the nose, and the fourth with the mouth. A value of one indicates the specific face part was manipulated and replaced while a value of zero indicates the part has been left as the real image.

The fake face folder contains 240 images labeled easy, 480 labeled mid, and 240 labeled hard, but because these groups were created subjectively, we do not use them as explicit categories when building and training the model.

The image sizes and image resolutions are standardized across all 2,041 images in the dataset. Each image contains 600 pixels and a resolution of 100 DPI. As for RGB color channels, there does not appear to be a significant difference in color distributions between the two image classes in the dataset.



## 

## **5 Methods**

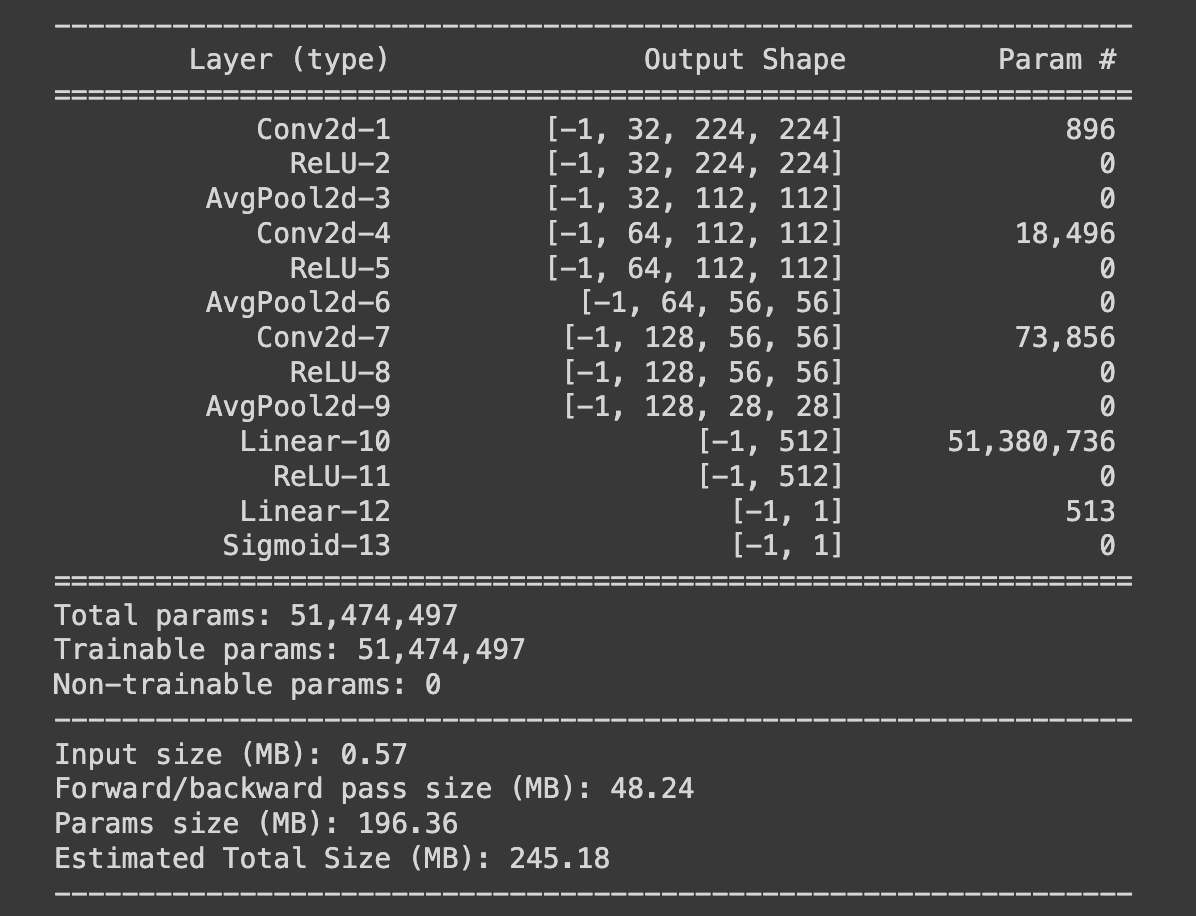
**5.1 Convolutional Neural Networks**

Since deep learning models have the potential to capture complex patterns and details in images, we proceed with a neural network approach, particularly a CNN. This model will generally be more robust to variations in input images, including differences in scale, lighting, or augmented features.

The selected model contains three convolutional layers, each with a kernel of size three by three. The input layers are also padded with a single layer of zeroes, allowing the kernels to be applied to all regions of the images and preventing size reductions. The first convolutional layer takes RGB channel layers to produce 32 feature maps, the second layer outputs 64 feature maps, and the third layer outputs 128. The feature maps double between each layer, allowing the network to detect increasingly complex details. The convolutional layers are effective for image processing, and more computationally efficient than fully connected layers.

Following each convolutional layer is a pooling layer. The pooling layers have a specified window size of 2x2 and a stride of 2, essentially halving the number of feature mappings, which increases efficiency and abstraction of the features. This also helps prevent overfitting. The pooling layers help the network determine if certain features are present in an input image, but may miss the actual relevant information of the feature because it focuses primarily on obtaining the correct location of the specified feature[14].

After the convolutional and pooling layers, the model has two fully connected layers. The first fully connected layer reduces the dimension of the input from 128x28x28 to 512. The second reduces this dimension to 1, preparing the input to be transformed to binary output. The fully connected layers allow the network to take in featuresfrom the input and transform them to the desired output.For the convolutional and first fully connected layer, ReLu activation was selected, as it is a more computationally efficient activation function and helps avoid the common vanishing gradient problem. The last fully connected layer uses sigmoid activation, which is suitable for binary classification. This allows the network to output values between 0 and 1, the probability the image belongs to one class. We selected BCEWithLogitsLoss() as our loss function.



**5.2 Optimizer**

In CNNs, optimizers play a critical role in the learning process by updating the parameters on each training iteration. At each step, the parameters are adjusted in the direction of the gradient. We evaluated various optimizers in our CNN architecture to find which would optimize training and validation loss, increase training and validation accuracy, and prevent overfitting.

We first implemented a Stochastic Gradient Descent (SGD) optimizer with momentum. This configuration resulted in an overall increase in both the training and validation accuracy matched with a decrease in the training loss. The training and validation accuracy reached 0.6366 and 0.6088, respectively. The training loss reduced to 0.5351 from 0.7058 and the validation loss stayed consistent at around 0.68. The implementation of momentum helped in accelerating the parameter updates in the direction of steepest descent, enhancing convergence speed. Additionally, the stochasticity of the optimizer helped avoid local minimas and find the global minimum.

After observing the success of the SGD optimizer, we tested the accuracy of the CNN with the Averaged Stochastic Gradient Descent (ASGD) optimizer to see if performance would improve. Although this resulted in decreasing loss for both training and validation data, the training and validation accuracy remained the same, indicating the ASGD was less effective than SGD with Momentum optimizer.

Tran *et al.* [17] utilizes an Adam optimizer in her CNN model for a high performance deepfake detection model. This was the next optimizer that we tried for our CNN model [17], but Adam did not yield any significant changes in accuracy or loss. Thus, this was not a suitable optimizer for our model.

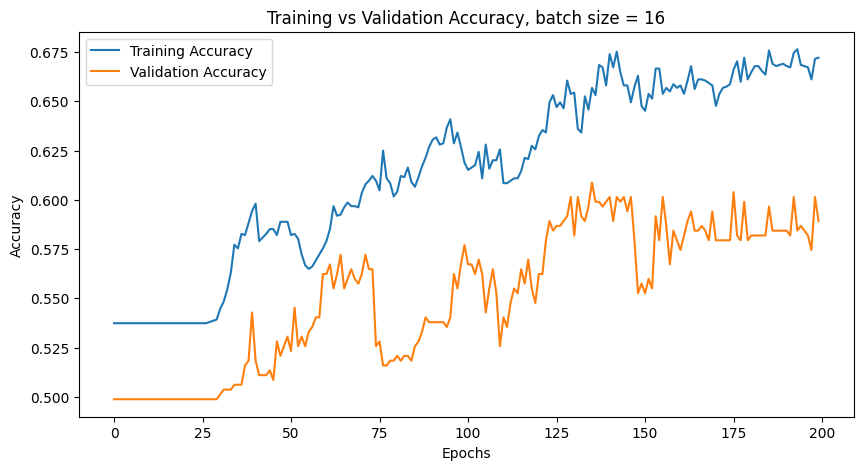
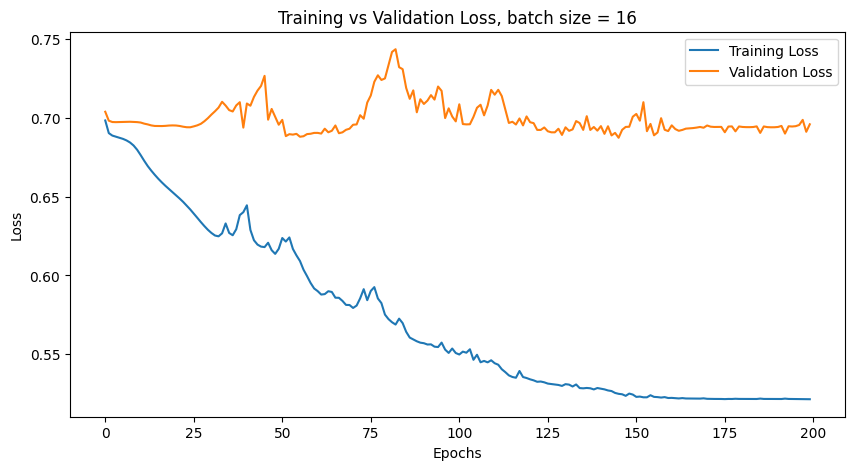
The final optimizer we tested was RMSprop (Root Mean Square Propagation). This optimizer is designed to adapt the learning rate for each parameter. Unlike previous optimizers, RMSprop adjusts the learning rate dynamically, using a moving average of squared gradients. However, this optimizer also proved to not be as effective as SGD because there was no change in the accuracy or the loss over 200 epochs.

In conclusion, our evaluation showed that SGD with momentum was the most ideal optimizer to use for our CNN model, which is what we proceed with.

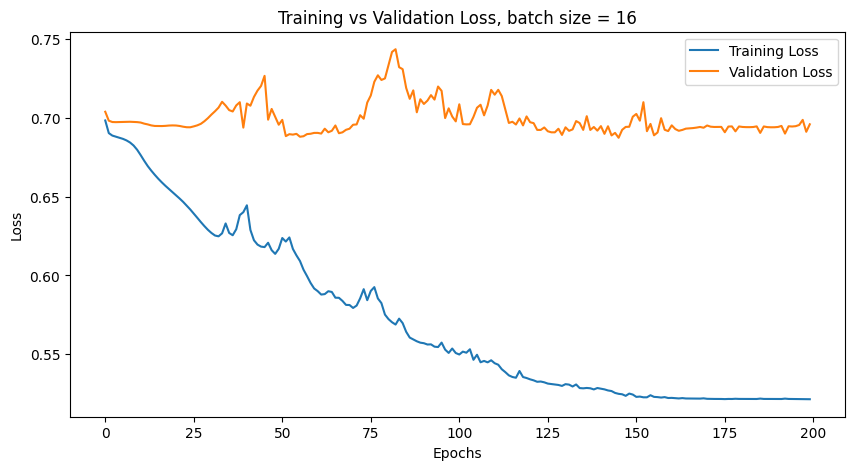
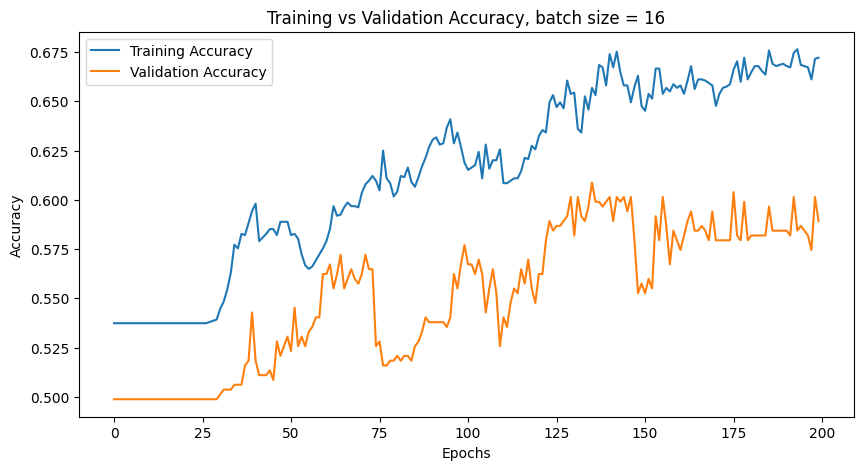
**5.3 Pooling Type**

The first item we experimented with was the pooling layer within our network by implementing both max pooling and average pooling. Max pooling involves sliding a window across the input feature map and taking the maximum from within each window. This captures most prominent features in each window to be captured and discards the more significant ones. Average pooling calculates the average of each window. This generally leads to a smoothed representation of the input because all four values in the window are considered equally.

We initially proceeded with max pooling because it is more commonly used in deep neural network architectures, and is typically better at capturing intricate patterns in the data. It can also lead to creating a more pronounced feature representation that captures the most important features in the images. However, even after tuning hyperparameters and experimenting with batch size, the model overfit the data.



Thus, we decided to try average pooling instead, as it could help reduce the models’ sensitivity to noise in the data, possibly reducing overfitting. After continuing with the same hyperparameter and batch size tuning process, we found a more optimal model that did not exhibit as extreme overfitting behavior. Thus, we proceed with average pooling in our network.

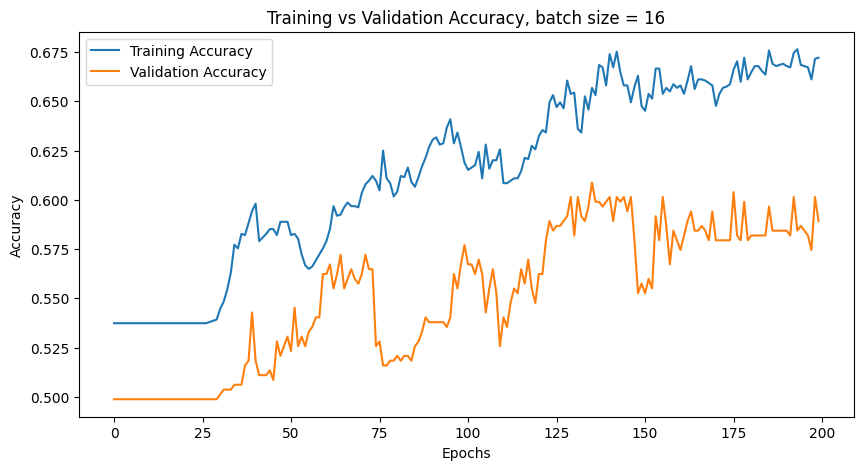
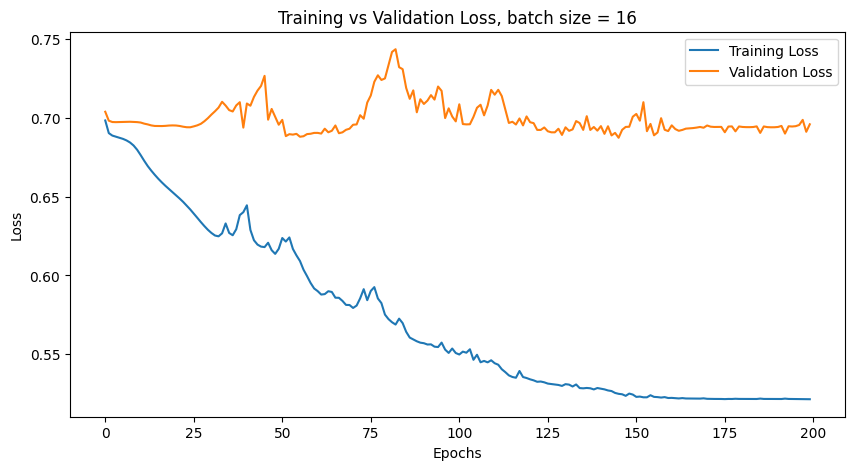


**5.4 Batch Size**

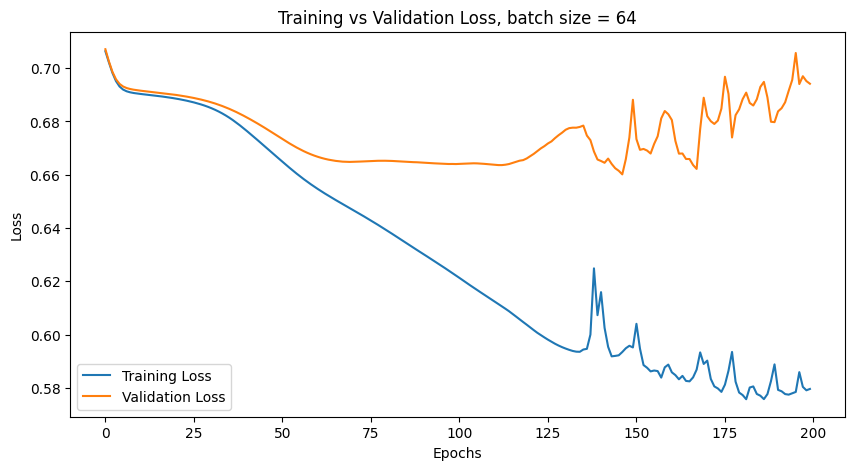
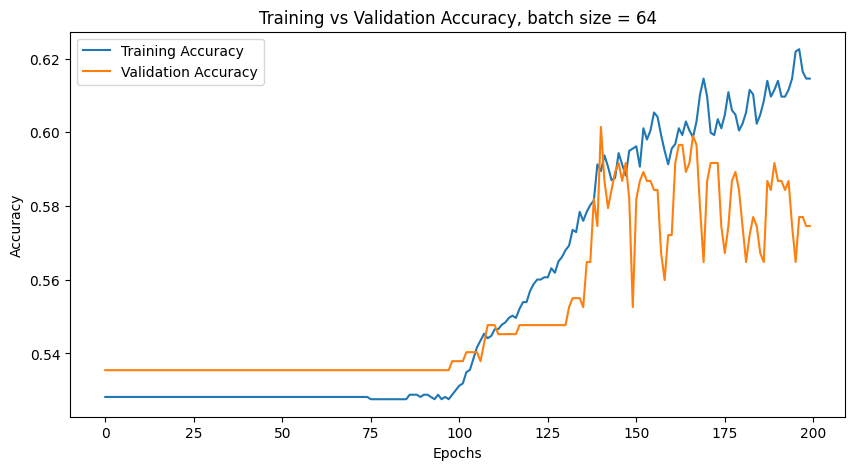
Aside from experimenting with hyperparameters directly related to the model, batch size was an additional important hyperparameter that we identified. The size of the batch is the number of images being used to train at a time for each iteration. While a higher batch size may lead to increased computational efficiency during training because of available parallel processing capabilities, this can lead to poor generalization and large use of memory. A lower batch size tends to use up less memory. However, the relationship between batch size and computational efficiency is not a completely linear relationship [15]. When it comes to optimizing functions, there is a trade-off that comes with choosing batch size. A large batch size increases the possibility of convergence to the global optima of the function, but can be slow. Smaller batch sizes may speed up the convergence to relatively good optima since the model parameters are frequently updated, but it is not guaranteed that the function will converge to the global optima. The lower the batch size gets, the more the network may bounce around when trying to converge. This can create noisy gradient estimates and prevent accuracy from increasing [16].

The bottom line is that batch size must be carefully considered and experimented with to find a balance between computational efficiency, memory usage, and noisy gradients and generalization ability.

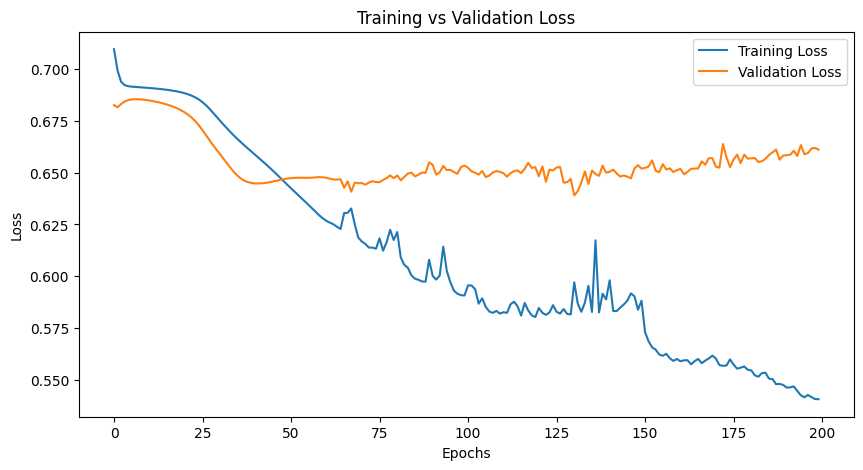
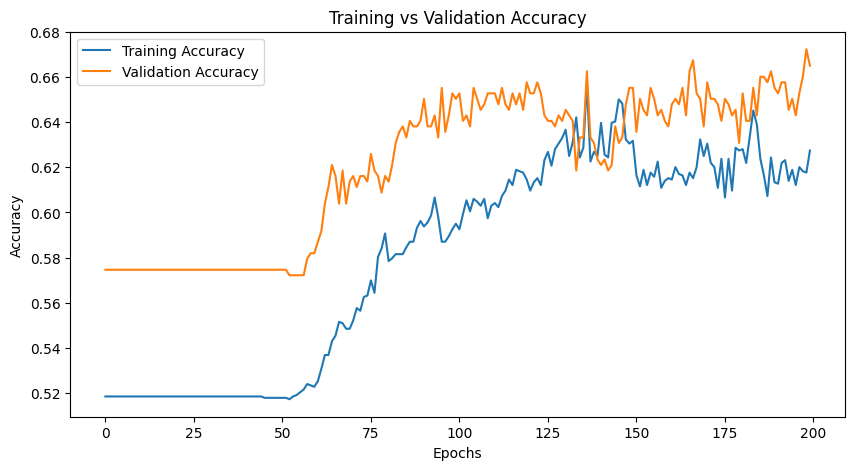
Similar to the process of selecting a learning rate, the model was first run with a relatively small batch size of 16. The resulting loss and accuracy graphs are shown below, with both displaying training and validation error over time. While both training and validation accuracy appear to be improving over time, there is a large disparity between the two curves, with the training curve significantly higher than the validation one. This indicates that the model is likely overfitting because it has learned features that are very specific to the training data and it cannot generalize as well to the unseen testing data. This idea is further supported by the loss graph, where the validation loss appears mostly stagnant, while the training loss continually decreases. While the model’s ability to represent the training data improves, this improvement doesn’t translate to the validation data performance.



To try and avoid overfitting, we use batch size 64 next. The accuracy graph displays similar curves for both training and validation errors, and they both increase over time. However, both of the curves contain many oscillations. The loss graph shows similar overfitting behavior to when the model was run with batch size 16. Validation loss appears to increase after initially decreasing, while the training loss decreases.



Since high variance and noise was demonstrated with the batch size of 64, we proceed with a more moderate batch size of 32 in an attempt to obtain a good balance between computational efficiency and memory usage while also avoiding overfitting. The resulting accuracy graph is much improved compared to the one created with batch size 64. Though both of the curves still oscillate, the oscillations are much less drastic. The training and validation curves also still match each other very closely. The issues previously observed in the loss graph still persist. Validation loss decreases a bit initially, but then plateaus, while training loss continues to decrease.



This ovefitting behavior is observed even with a batch size of 10. This leads to the question of whether this problem of overfitting can be solved by batch size and hyperparameter tuning alone, as this behavior persisted even with a variety of different loss functions and learning rate schedulers.

**5.5 Learning Rate Changes & Schedulers**

A critical method of optimizing machine learning models is experimenting with learning rate. The loss function evaluates the CNN’s performance by comparing predicted labels to actual labels. Based on this loss function, the network will calculate gradients to adjust the weight that each node will contribute to future layers and the ultimate prediction output. The learning rate then scales these gradients which determines how much to adjust the weights of the network’s nodes. A well-tuned learning rate will balance timely convergence and accuracy [18].

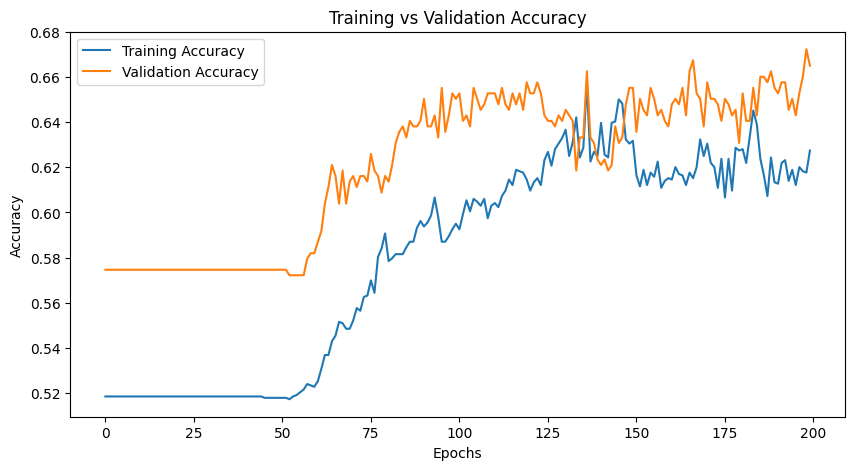
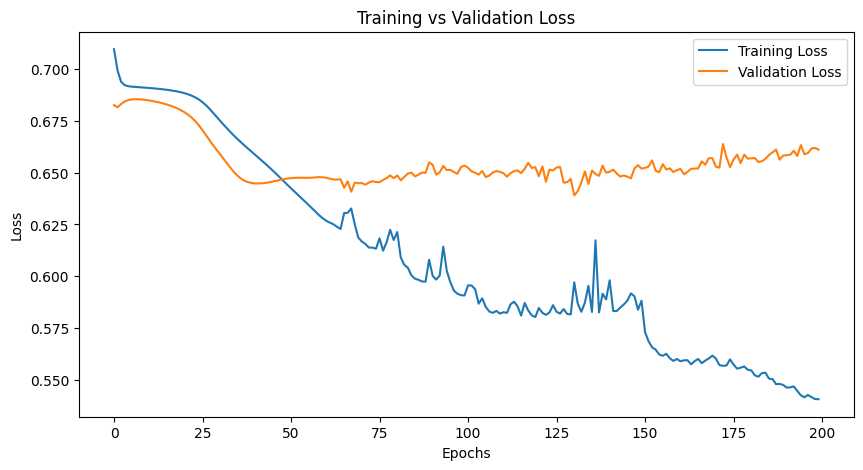
Another consideration was using more advanced methods to vary learning rate. We experimented with learning rate schedulers to optimize convergence through dynamically changing learning rate [19]. The first we used was exponential decay. This scheduler smoothly reduces the learning rate over time, which can be beneficial for training CNNs since it allows for more granular updates in the later stages of training and helps fine-tune the complex feature detectors. The method tested was ReduceLROnPlataeu [20]. This scheduler reduces learning rate when a model’s performance on a validation set stops improving to prevent overfitting and ensure the model does not overshoot the global minimum.

**5.6 Dropout**

The last method we utilized was dropout. CNNs, due to their complexity and size, have a tendency to overfit data. Anticipating this, we experimented with dropout to prevent extreme overfitting [20]. Intuitively, dropout works by randomly deactivating a subset of neurons in the network during each training iteration. This results in preventing the model from heavily relying on a given set of neurons, promoting the development of more robust features. Dropout did not succeed in improving overfitting.

**6 Results**

As discussed in the Methods section, we experimented with pooling type, batch size, optimizer, and learning rate to determine the highest performing model. We found average pooling built a more accurate model than max pooling. We suspect this is the case because average pooling aggregates over a matrix of values which can result in smoother, more generalized feature representations. Max pooling simplifies the representation by only taking the maximum value. In terms of batch size, we found the best size was 32. We experimented with 16 and 64 as well, but the first severely overfit and the second had a lot of noise and variation. The best optimizer for our model was SGD. We also tried Adam, RMSprop, and ASGD optimizers but the accuracy for these did not improve even over the course of 200 epochs. SGD is a simpler optimizer which can lead to more stable and reliable training. We also varied the learning rate and implemented schedulers to change the rate in which we updated the weights. We found that the learning rate value of 0.001 was the most appropriate value because it was small enough to not overshoot the minimum. Unfortunately, the learning rate schedulers did not provide any additional accuracy to our model. The final item we tried was dropout in an attempt to prevent overfitting, but it did not lead to any better accuracy than when it is not implemented.

*Above is the training and testing loss and accuracy of our final model*

## **7 Discussion & Conclusion**

Our findings reveal that by leveraging deep learning architectures and experimenting with different hyperparameters, CNNs can distinguish between images of real and fake faces with a degree of accuracy significantly better than random guessing. Our selected model achieved an accuracy of 67%. While this number is promising, it still has a considerable margin of error, indicating that CNNS should not be solely trusted when it comes to fake face detection.

However, it is crucial to identify the limitations of our study, particularly related to the depth of our model. Due to resource constraints, the model was not built as deeply as may have been ideal. A deeper model would have captured more complex details of input images.

Another limitation arises from the dataset. The fake faces in the dataset were created by swapping certain features between other faces, which is not comparable to the sophisticated processes used in deepfake generation today. The provided data was also rather small at 2,041 images. This discrepancy indicates that our model may not perform as effectively on more advanced deepfakes, but we can still consider the model to be a stepping stone in the process to create a more generalizable CNN for fake face detection.

Future research should focus on expanding model depth and diversifying datasets. Including a diverse array of deepfakes as well as face photos from real-world situations would allow the model to potentially reach human-level accuracy in different real environments. This could open up the possibility of integrating fake face detection softwares into more practical applications.

Overall, our study represents an important step in developing machine learning tools for deepfake detection. The insights show the clear strengths of taking a CNN-based approach, but also highlight the need for more sophisticated model architectures and diverse datasets to better tackle the problem of increasingly advanced deepfake photos.

**Data Availability**

This is publicly available on Kaggle at the following link: <https://www.kaggle.com/datasets/ciplab/real-and-fake-face-detection>.

**Contribution Statement**

We collaborated heavily on this project and found value in meeting at least once a week since the start of the quarter. This kept us on track and ensured a successful completion of our project free of procrastination. The beginning stages of project ideation were shared equally amongst the group members as we all brought ideas to the table and selected one together. We also loaded the data and built the initial model together which was helpful because we had one another to lean on for research and questions. We split up the model training as such: Alyssa experimented with batch size and pooling type, Sarah with learning rate and dropout, and Ryan with the three different optimizers. We wrote the paper according to the methods we personally experimented with and evenly split the remaining sections.

**References**

[1] Ghosh, Ankush, et al. Advanced Communication and Intelligent Systems Second International Conference, ICACIS 2023, Warsaw, Poland, June 16-17, 2023, Revised Selected Papers, Part II. Springer International Publishing AG, 2023.

[2] Ferrara, Emilio. “Genai against humanity: Nefarious applications of Generative Artificial Intelligence and large language models.” SSRN Electronic Journal, 2023, <https://doi.org/10.2139/ssrn.4614223>.

[3] Hancock, Jeffrey T., and Jeremy N. Bailenson. “The social impact of deepfakes.” Cyberpsychology, Behavior, and Social Networking, vol. 24, no. 3, 2021, pp. 149–152, <https://doi.org/10.1089/cyber.2021.29208.jth>.

[4] “Doctored Nancy Pelosi Video Highlights Threat of ‘Deepfake’ Tech.” CBS News, CBS Interactive, 26 May 2019, [www.cbsnews.com/news/doctored-nancy-pelosi-video-highlights-threat-of-deepfake-tech-2019-05-25/](http://www.cbsnews.com/news/doctored-nancy-pelosi-video-highlights-threat-of-deepfake-tech-2019-05-25/).

[5] Moyer, Tate. “Can Deepfakes Change Election Outcomes?” The Michigan Daily, 2 Nov. 2023, [www.michigandaily.com/opinion/how-generative-ai-will-threaten-democracy/](http://www.michigandaily.com/opinion/how-generative-ai-will-threaten-democracy/).

[6] “Tom Hanks Says Ai Version of Him Used in Dental Plan Ad without His Consent.” The Guardian, Guardian News and Media, 2 Oct. 2023, [www.theguardian.com/film/2023/oct/02/tom-hanks-dental-ad-ai-version-fake](http://www.theguardian.com/film/2023/oct/02/tom-hanks-dental-ad-ai-version-fake).

[7] Rafique, Rimsha, et al. "DeepFake detection using error level analysis and deep learning." 2021 4th International Conference on Computing & Information Sciences (ICCIS). IEEE, 2021.

[8] Frank, Joel, et al. "Leveraging frequency analysis for deep fake image recognition." International conference on machine learning. PMLR, 2020.

[9] Wu, Wenxuan, et al. "Capturing the Lighting Inconsistency for Deepfake Detection." International Conference on Artificial Intelligence and Security. Cham: Springer International Publishing, 2022.

[10] Yang, Jiachen, et al. "Mtd-net: learning to detect deepfakes images by multi-scale texture difference." IEEE Transactions on Information Forensics and Security 16 (2021): 4234-4245.

[11] Yadav, Digvijay, and Sakina Salmani. "Deepfake: A survey on facial forgery technique using generative adversarial network." 2019 International conference on intelligent computing and control systems (ICCS). IEEE, 2019.

[12] Agarwal, Harsh, Ankur Singh, and D. Rajeswari. "Deepfake detection using svm." 2021 Second International Conference on Electronics and Sustainable Communication Systems (ICESC). IEEE, 2021.

[13] Güera, David, and Edward J. Delp. "Deepfake video detection using recurrent neural networks." 2018 15th IEEE international conference on advanced video and signal based surveillance (AVSS). IEEE, 2018.

[14] Alzubaidi, Laith, et al. “Review of Deep Learning: Concepts, CNN Architectures, Challenges, Applications, Future Directions.” *Journal of Big Data*, vol. 8, no. 1, Mar. 2021, https://doi.org/10.1186/s40537-021-00444-8.

[15] Kandel, Ibrahem, and Mauro Castelli. “The Effect of Batch Size on the Generalizability of the Convolutional Neural Networks on a Histopathology Dataset.” *ICT Express*, vol. 6, no. 4, May 2020, https://doi.org/10.1016/j.icte.2020.04.010.

[16] Shen, Kevin. “Effect of Batch Size on Training Dynamics.” *Medium*, 19 June 2018, medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e.

[17] Tran, Van-Nhan, et al. "High Performance deepfake video detection on CNN-based with attention target-specific regions and manual distillation extraction." Applied Sciences 11.16 (2021): 7678.

[18] Rakhecha, Aditya. “Understanding Learning Rate.” Medium, Towards Data Science, 7 July 2019, [towardsdatascience.com/https-medium-com-dashingaditya-rakhecha-understanding-learning-rate-dd5da26bb6de](http://towardsdatascience.com/https-medium-com-dashingaditya-rakhecha-understanding-learning-rate-dd5da26bb6de).

[19] Li, Katherine (Yi). “How to Choose a Learning Rate Scheduler for Neural Networks.” Neptune.Ai, 9 Aug. 2023, [neptune.ai/blog/how-to-choose-a-learning-rate-scheduler#:~:text=A%20Learning%20rate%20schedule%20is,iterations%20as%20the%20training%20progresses](http://neptune.ai/blog/how-to-choose-a-learning-rate-scheduler#:~:text=A%20Learning%20rate%20schedule%20is,iterations%20as%20the%20training%20progresses).

[20] Suki Lau. “Learning Rate Schedules and Adaptive Learning Rate Methods for Deep Learning.” *Medium*, Towards Data Science, 29 July 2017, towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1.

[21] Martins, Carla. “Dropout Layer Explained in the Context of CNN.” Medium, Medium, 20 Sept. 2023, [cdanielaam.medium.com/dropout-layer-explained-in-the-context-of-cnn-7401114c2c#:~:text=The%20Dropout%20Layer%20is%20a,generalize%20well%20to%20unseen%20data](http://cdanielaam.medium.com/dropout-layer-explained-in-the-context-of-cnn-7401114c2c#:~:text=The%20Dropout%20Layer%20is%20a,generalize%20well%20to%20unseen%20data).