Applications of Existing Convolutional Neural Networks to Deepfake Detection

Saanvi Sheoran

Received January 10, 2025
Accepted May 05, 2025
Electronic access June 30, 2025

As technology becomes increasingly advanced, AI-generated media known as Deepfakes have raised concerns in areas like cybersecurity, misinformation, and privacy. Since Convolutional Neural Networks (CNNs) have proved effective in other fields like image classification tasks, the objective of this research is to apply these CNNs to the growing problem of detecting Deepfake images. Even if they are not initially designed for this task, this study will determine which of these architectures perform best for detecting Deepfake images. A dataset of real and Deepfake images was constructed and several CNN architectures including pre-existing and pre-trained models like VGG 16, VGG 19, DenseNet121, and ResNet50, were tested. A previously developed architecture for age classification was also modified to fit this task and all were compared to a custom architecture designed. The custom model achieved the highest validation accuracy of 97%, though it lacks generalizability and had a validation accuracy of 81.1% when tested on another dataset. Pre-trained transfer learning models underperformed even with fine-tuning, with ResNet50 yielding a maximum accuracy of 94%. The highest performing modified age-classification model had a validation accuracy of 92.36%. Increasing the number of convolutions in the modified age-classification model roughly improved accuracy. Overall, this study found that though pre-trained image classifiers can be applied to Deepfake detection, they were not designed for this task. Evidence of overfitting and differences between saliency maps for different models show the importance of developing large, generalizable datasets specifically for Deepfake detection. Custom CNNs and adapted models can be developed and run quickly with high performance on one dataset, but the lack of representative training data and State Of The Art (SOTA) methods leave models at risk of overfitting and limited applications of their results.
Keywords: Machine Learning, Computer Science, Deepfakes, Artificial Intelligence, Convolutional Neural Networks, Image Classification, AI-Generated Media, Model Performance Evaluation

Introduction

Motivation

Deepfakes, or digitally manipulated media, are becoming more common and sophisticated, creating challenges in areas like cybersecurity, misinformation, and privacy. Deepfakes have significant implications for cybersecurity, as they can be used to impersonate individuals in phishing attacks or bypass biometric authentication systems. In the realm of misinformation, Deepfakes can spread false narratives or discredit legitimate sources, causing widespread confusion and distrust. From a privacy standpoint, the ability to create hyper-realistic synthetic media poses a risk to personal reputations and can be used for blackmail, harassment, or identity theft. Although AI detection exists in current research, it is often inaccurate when faced with new, sophisticated deep fakes, limiting their usefulness ((N. Jacobson, Deepfakes and Their Impact on Society, CPI OpenFox, 2024)). By enhancing detection systems, this research aims to fill this gap, mitigating the threats of AI generated media and protecting individuals, institutions, and the public.

This research focuses on answering the question: Which CNN architectures are best to detect deep fake media effectively? To achieve this, this study aims to train, test, and compare the performance of various CNN architectures with different features to determine which architectures are better suited to this task. The CNN architectures tested in this study are most, if not all, key performers in image recognition, but have yet to be applied to Deepfake detection. The scope of this research is limited to binary classification of images as either real or fake, since CNN architectures are trained on labeled data. Deepfake detection for images is focused on within this study, so conclusions or data drawn from this study may not be applicable to other forms of media such as videos, time lapses, etc.

Background

The industry standard tools for detecting Deepfake media consist of various deep learning techniques, most notably CNNs and Generative Adversarial Networks (GANs). CNNs are a type of artificial neural network primarily used for image recognition tasks due to their ability to detect minute details through kernel or filter optimization. One of the most used CNNs for identify-

ing AI-generated images is XceptionNet, a CNN that excels in detecting pixel anomalies often indicative of Deepfakes.

Patel et. al. conducted a case study on Deepfake Generation and Detection in 2023 [1], and they report three approaches to Image/ Video Detection: Physical/Physiological based approaches, Signal Level Feature based approaches, and Data Driven models. Data Driven models relate the most to this proposed study, and Patel et. al. describes common CNNs that have performed well for problems like this. Rather than focusing on specific features of images or faces (for face forgery problems), data driven models like CNNs and Recurrent Neural Networks (RNNs) are trained over a general dataset of AI and real images/videos. One of the models they discuss alongside other industry standard models like Xception and DenseNet is MesoNet, which uses Inception as the foundation of its architecture, allowing it to detect compressed images and Deepfakes on social media platforms, where not all images are high resolution.

Additionally, Ashok V and Dr. Preetha Theresa Joy applied XceptionNet to Deepfake detection in 2023 [2], where they used XceptionNet to categorize images and video frames into real or fake. They compared the results to other SOTA models like VGG-16, ResNet-152, and Inception V3 and found that XceptionNet performed better than the other models. However, using CNNs for this problem risks overfitting the data, essentially training the model to memorize rather than predict. The model becomes used to seeing the same data but is not actually learning anything new during the epochs. Approaches including CNNs often struggle to generalize their results to other datasets because of this, meaning that their model performs well only on their training datasets, leading to underperformance on new data and an overall not robust conclusion. Ashok V and Dr. Joy combated this by taking their data from FaceForensics++ Database, a forensics dataset containing 1000 videos that have been altered using four different face forging techniques: Face2Face, Deepfakes, FaceSwap, and NeuralTextures. This helps minimize confounding variables due to only one technique being used to encode the real videos into Deepfakes. They train their model using 5000 real images and 5000 Deepfakes and then test their data on 794 video frames taken from FaceForensics++.

This study uses a large dataset to combat overfitting and increase the generalizability of the custom architecture CNN. It uses 60,000 synthetically generated images and 60,000 real (Collected from Krizhevsky & Hinton's CIFAR-10 dataset) images to build a dataset for the models; a large dataset like this helps expose the model to many different images and helps create a more complex model through a thorough training process. Furthermore, the personal CNN model was tested on data from Manjil Karkis Deepfake and Real Images dataset to identify how it performs on unseen data and test its generalizability.

Furthermore, A GAN is another widely used tool that creates two neural networks: a generator network and a discriminator network. The two are pitted against each other such that the generator outputs progressively convincing fake images while the discriminator attempts to differentiate the AI-generated images from the real ones. This technique allows GANs to develop high accuracies and better identify subtle inconsistencies and patterns within data [3]. GANs and other deep learning methods like CNNs are staples within the field of AI-generated media detection as they provide a robust system for detecting and recognizing differences between real and fake images.

Though GANs are commonly used for Deepfake generation, Galamo Monkam, Weifeng Xu, and Jie Yan introduced G-Job GAN in 2023 [4], a GAN-based machine learning model that outperformed other industry standard GAN models. Their proposed architecture consists of two CNNs, a discriminator and a generator that work together to progressively create and detect Deepfake images. They pulled images from CelebA, a large-scale dataset containing over 200,000 celebrity images with 40 different facial attributes per image, using 162,770 images for training; 19,867 images for validation; and 19,962 images for testing. They measured their results using F1 score and additionally noted that their accuracy positively improved as their dataset size increased.

Though GANs bring unprecedented accuracy and immense potential to the field of Deepfake detection, they are notoriously hard to train, and the generators can easily become biased if not trained with a large and diverse dataset. Finding datasets like this is a key limitation of GANs since current real world Deepfake detection datasets often lack variance between different demographics (ethnicity, gender, image quality, age, etc) and manipulation techniques (lip syncing, expression transfer, face swapping, etc). Deepfake technology evolves quickly, which means that datasets can quickly become outdated as new technology surfaces. Privacy concerns with peoples real facial data is also a concern when trying to compile real images for these datasets, which is also why some public datasets are unable to capture the full range of images.

This was combated by utilizing transfer learning for the industry standard models incorporated into the study. These models will be imported using pre-trained weights that have already been determined due to exposure to different datasets, which will help increase the generalizability of the results and reduce the likelihood of overfitting. Additionally, the personal CNN model is tested on Manjil Karkis Deepfake and Real Images Dataset [5] to test generalizability of the solution and expose the model to different kinds of images. Use of images from different sources helps expose the model to a diverse set of images and helps determine the actual strength of the model.

Methodology Overview

This problem falls under supervised learning and is a classification task. In supervised learning, the model is trained on labeled data, where we know whether each image is real or fake.

Since the output is a prediction of whether an image is real or fake, this is a binary classification problem. The labels for the data are 0 for fake images and 1 for real images. A mix of real images and deep fake images was gathered to build the dataset. The data used is vision dataspecifically, images that were either generated by artificial intelligence (fake) or captured in the real world (real). After gathering the images, they were converted into numerical values that can be used by the CNN model. The models output is a label, either 0 or 1, depending on whether it predicts the image is fake or real.

After data preprocessing, several accepted image-recognition models with differing architectural priorities and goals such as DenseNet121, ResNet50, VGG-16, and VGG-19 were tested. For comparison, an age classifier was modified for this problem and a personalized CNN architecture was developed. Post testing and fine-tuning, these models were evaluated based on validation accuracy, validation loss, validation precision, validation recall, and validation f1 score. The generalizability of the personal CNN architecture was further tested on the Manjil Karki Deepfake and Real Images dataset from Kaggle[6]. By experimenting with different CNN architectures, the goal is to identify which one produces the most accurate predictions in order to understand which types of architectures in general are best applied to this situation. After all the models have been evaluated, saliency maps were visualized for true real predictions, true fake predictions, false real predictions, and false fake predictions for the ResNet50 model, 4 Convolution [512, 512, 2] model from the Age Classifier, Personal CNN on CIFAKE Dataset, and Personal CNN on the Manjil Karki Dataset. This research is significant because finding the best CNN architecture can improve deep fake detection systems and help address the growing issue of media manipulation.

Results

The transfer learning models had the lowest validation accuracy rates overall, as shown in the table below. ResNet50 overall had the highest test validation accuracy of 94 percent but was lower than the experimental model. The modified Age classifier model with the highest accuracy was the 4 Convolution [512, 512, 2] model with an accuracy of 92.36 percent.

These models were evaluated by comparing their validation losses, accuracies, precision, recall, and f1 score. Loss is an evaluation metric that works by comparing the true labels to the predicted outputs, using a specific loss function such as cross-entropy loss or mean squared error. This is helpful to visualize and compare how inaccurate different models are compared to each other. This study uses cross-entropy loss due to its common use in classification problems, and it works by comparing the true probability distribution of the data to the predicted probability distribution. Cross-entropy loss also helps to identify areas where possible over or underfitting occurred, which is essential in evaluating the overall success of the image classification models. Equation 1 shows the mathematical formula for calculating Cross-entropy loss.

$$Loss = -\frac{1}{n}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c} log(p_{i,c}) \tag{1}$$

Validation test accuracy, on the other hand, measures the percentage of correct labels over the percentage of total predictions. To make it a percentage, simply multiply by 100. Equation 2 shows the formula for accuracy.

$$Accuracy = \frac{CorrectPredictions}{TotalNumberofPredictions} \tag{2}$$

Including precision, recall, and F1 score helps identify the predicting habits of the model by assessing the likelihood of correctly identifying Deepfake images compared to the real images. This in addition to the accuracy and loss results creates a thorough group of evaluation metrics for the study. Below, Equation 3, Equation 4, and Equation 5 show the mathematical formulas used to calculate Precision[7], Recall[8], and F1 Score[9], respectively.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{3}$$

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{4}$$

$$F1Score = \frac{2 * Precision * Recall}{precision + recall} \tag{5}$$

Below is a table comparing the successes of the Transfer Learning Models, personal experimental model, and the raw architectures of VGG-16 and VGG-19 (Note the following metrics are all after fine-tuning).

For the Modified Age Classifier, 12 different models were generated. For each number of convolutional layers, the number of nodes vary. This pattern was repeated for convolutional layers of 2,3,4 and for node amounts of [2], [512,2], [512,512,2], and [1024,256,2].

The model with the highest accuracy for the age classifier was the 4 convolution [512, 512, 2], while the model with the lowest accuracy for the age classifier was 2 convolution [2]. The following confusion matrices show highest and lowest performing models. A label of 0 means fake, and a label of 1 means real.

Figure 1: Validation Metrics for Image Classification Models

Model Type	Validation Loss	Validation Accuracy	Validation Precision	Validation Recall	Validation F1 Score
VGG-16 Architecture	0.6931	0.5025	0.5025	0.5025	0.6689
VGG-19 Architecture	0.6931	0.5025	0.5025	0.5025	0.6689
VGG Transfer Learning	0.576	0.8097	0.8097	0.8097	0.8221, 0.7955
DenseNet121 Transfer Learning	0.4374	0.8378	0.8378	0.8378	0.8474, 0.8268
ResNet50 Transfer Learning	0.17463	0.9355	0.9355	0.9355	0.9335, 0.9374
Personal CNN	0.1314	0.9696	0.9696	0.9696	0.9697, .9695
Personal CNN on Manjil Karkis Dataset	0.9635	0.8112	0.8215	0.8112	0.8103

Figure 2: Validation Metrics for Modified Age Classifier

Model Name	**Validation Accuracy**	**Validation Loss**	**Validation Precision**	**Validation Recall**	**Validation F1 Score**
2 convolution [2]	0.86175	0.3390811	0.86175001	0.86175001	0.863423 0.86003536
2 convolution [512, 2]	0.8851666	0.2985414	0.88516665	0.88516665	0.89142764 0.87813926
2 convolution [512, 512, 2]	0.9014167	0.2953568	0.90141666	0.90141666	0.9040628 0.8986202
2 convolution [1024, 512, 256, 2]	0.8991666	0.3799755	0.89916664	0.89916664	0.90309143 0.89491045
3 convolution [2]	0.9109167	0.2644065	0.91091669	0.91091669	0.9118205 0.90999407
3 convolution [512, 2]	0.9051667	0.2593969	0.90516669	0.90516669	0.91045004 0.89922065
3 convolution [512, 512, 2]	0.9159167	0.2499795	0.91591668	0.91591668	0.9135167 0.91818696
3 convolution [1024, 512, 256, 2]	0.91775	0.4092084	0.91775	0.91775	0.91954017 0.91587824
4 convolution [2]	0.9123333	0.2316059	0.91233331	0.91233331	0.9096375 0.9148729
4 convolution [512, 2]	0.9188333	0.2288809	0.91883332	0.91883332	0.9226492 0.91462123
4 convolution [512, 512, 2]	0.9236667	0.226338	0.92366666	0.92366666	0.92419726 0.9231285
4 convolution [1024, 512, 256, 2]	0.9219167	0.2523551	0.92191666	0.92191666	0.92404956 0.9196604

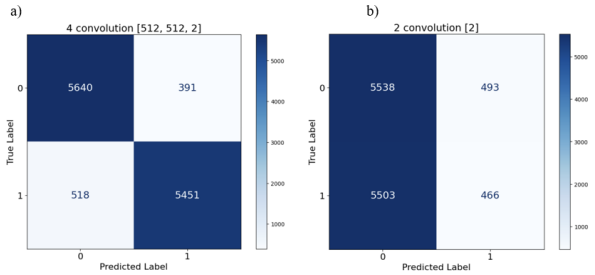

Figure 3: Confusion Matrices for Modified Age Classifier Model. (a) 4-convolution model [512, 512, 2] with the highest validation accuracy (92.36%). (b) 2-convolution model [2] with the lowest validation accuracy (86.17%)

Methodology

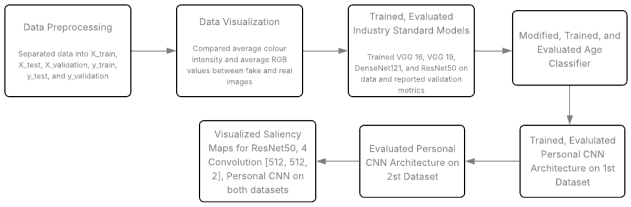

Figure 4: Overview of Methodology Visualization

Data Preprocessing

This project uses CIFAKE: Real and AI-Generated Synthetic Images[5] taken from Kaggle, which contains 60,000 synthetically generated images (generated the equivalent of CIFAR-10 with Stable Diffusion version 1.4) and 60,000 real (Collected from Krizhevsky & Hinton's CIFAR-10 dataset) images. At the beginning of the project, the images are converted to numerical values of 0 and 1, where 0 represents an AI generated image and 1 represents a real image. Data was split using Sci-kit Learns train test split using an 80:20 split. The data is preprocessed by looping through the image paths and separating them into four lists: real_test, real_train, fake_test, fake_train. The labels are converted into 0 and 1 and set equal to the y. The X became all four of the lists added together and the y is the labelled 0 and 1 numbers. The test set is later split in half to create a separate validation set.

To preprocesses specifically for CNN testing, the values are converted back from binary into an image of 32x32 such that the sizes of X_train, X_test, X_validation, y_train, y_test, y_validation, respectively, are as follows: (96000, 32, 32, 3), (24000, 32, 32, 3), (12000, 32, 32, 3), (96000, 2), (24000, 2), (12000,2). The goal of this is to better organize our data into different lists to be used later when experimenting with scikit-learn

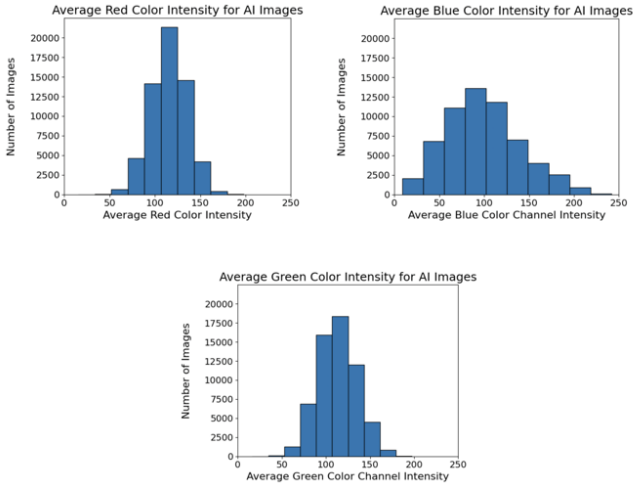

Figure 5: Color Channel Intensities for AI Images

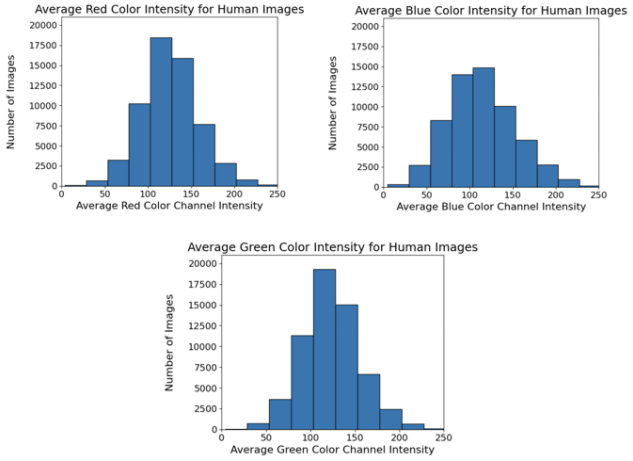

Figure 6: Color Channel Intensities for Human Images

models and visualizing the data. To better understand the data, the different RGB color distributions, as well as the average brightness for pixels, are visualized for the fake images and the real images separately.

To visualize the RGB color intensities, the red value in each of the images is isolated and used in a row major order to loop through each individual pixel in the image. For each pixel, the red, blue, and green values were summed individually, and the dimensions of the image was used to return the average color value of that specific image color. Then, for each color per fake and real images, a total of 6 lists was created and a histogram was visualized for each. Shown in Figure 5 are the average red, blue, and green color intensity values for the AI-Generated images, and in Figure 6 for the Human images (total dataset, not relevant to train or test). We observe that though the distributions of red and green color intensity are approximately normal for both human and ai images, the distribution of blue

color intensity appears skewed right, which may indicate that most images do not include a lot of blue color in them. However, there are no significant differences in the distributions between human and AI images.

To visualize the brightness values, the images are converted into black and white values and then using the same row major approach to loop through each pixel in the image. To find the average brightness per image, each grayscale image was added to a total sum for the image and divided by the dimensions to find the average value. Then by looping through the list of real and fake images, this code was repeated for each image in the study and saved in two lists: one for average brightnesses of all real images, and one for all fake images.

Both are plotted on a histogram to make comparison easier and both are shown within Figure 7 below. Both graphs show an approximately normal distribution, with most images having a color intensity not far on either end of the brightness spectrum. This indicates that there is not a significant skew for color intensity between AI images or human images. However, we observe that the average color intensity for human images has a slightly greater spread than that of AI images, which may be due to colors and shades found in real life that AI images do not replicate.

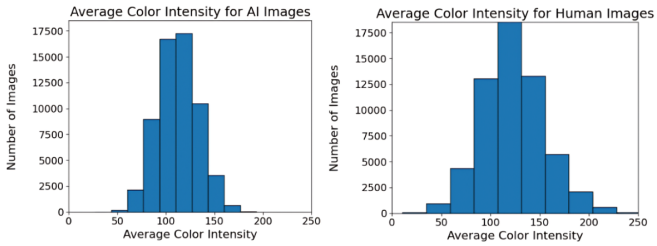

Figure 7: Average Color Intensity for AI Images and Human Images

To solve the research problem, this work began by implementing a few scikit- learn methods (KNN Model, Random Forest Model, Linear Regression Model, SVC Model) using both the brightness training data described in the Dataset Section, and the full image data. After experimenting with the data to better understand how in general it responds to model classification, the study progressed to experimentation with the neural network MLP classifier and then began testing various CNN architectures on the data.

Introduction to Models Used

The VGG-16 and 19 are two pre-existing CNN architectures commonly used for image recognition within this field. The architecture is known to be simple and consists of 16 and 19 layers respectively, with small 3x3 filters. They are known for their simplicity and are the benchmark in image classification, often also being implemented in image style transfer problems [10]. These models are manually implemented by building their layers in Tensorflow and Keras without pre-trained weights.

The idea of transfer learning is also applied by importing the VGG model from Keras Applications that includes predetermined weights based on past training. Transfer learning uses models that have already been trained on large datasets and adapts them to new tasks, meaning that this model has already been configured to produce its best accuracy for image recognition tasks, which is what it is commonly used for in this field [11]. For the VGG-16, VGG-19, and transfer learning VGG models, the models are later fine-tuned by unfreezing the layers for all the models from layer 14 onward, which includes the last two convolutional blocks (each block includes one max pooling layer and three convolutional layers) and connected dense layers. This is so that the earlier layers that are useful for more general observations were kept frozen, while the later layers that are involved in leading abstraction and pattern identification are retrained to better distinguish deepfakes from real images.

DenseNet121 is another architecture looked at, with a unique structure where each layer is connected to every other layer, instead of stacking layers one after another. DenseNet121 ensures each layer receives input from all previous layers which makes the flow of information more efficient [12]. This model is imported from Tensorflow Keras applications to obtain its predetermined weights from prior training on image recognition tasks. This model is commonly used for text generation, image generation, and image feature extractions. For DenseNet121s finetuning, it was unfrozen from layer 299 out of 429, keeping the final Dense Block and classifier layers, which are responsible for the decision making and minor pattern identification, two crucial aspects of Deepfake detection. Retraining the model on the later dense blocks from the architecture helped reduce the risk of overfitting from unnecessarily rerunning earlier layers and helps improve performance by letting the model specialize its predictions.

Additionally, this study uses the ResNet architecture, which is unique in the fact that it prioritizes model efficiency by using residual blocks to skip unnecessary layers to reduce computation time and make it easier for the model to train itself. This clever approach allows ResNet to have extremely deep networks while avoiding performance degradation that often comes with an increase in convoluted layers. Due to its often deep, complicated nature, this model is commonly used for deep computer learning tasks like image segmentation and object detection [13]. Within the field it is most applied to image recognition tasks, and in this study is used transfer learning to import it from the Tensorflow Keras application library. For the ResNet50 transfer learning, the first 7 blocks are preserved, which primarily serve as general feature analysis, and unfroze the layers starting at Block 8, which includes the final convolutional layers, global average pooling, and fully connected output layer. The model is retrained

with these upper layers so that it could adapt and detect subtle patterns within deepfake images that were not determined during original training.

Finally, to act as a comparison point, this study looks at a personalized CNN with strategically composed convolutional and pooling layers to see if it could outperform the standard models used for image classification. This methodology helps determine whether it is necessary to develop specific and targeted models or if other models perform well too.

This model takes images of size 32x32 with three RGB color channels with 128 filters. The first block contains an alternating pattern of a convolutional layer, a Leaky ReLU activation function, and then runs a Batch Normalization. A Leaky ReLU activation function is a variant of a ReLU that helps the network learn complex patterns by reducing the likelihood of the network becoming inactive while training. Batch Normalization helps to normalize the outputs of the previous layers to help stabilize the model [14]. This three-layer combination is repeated twice more. The first block ends with a pooling layer that reduces the dimensions by half, so it becomes 16x16 and a dropout layer that drops 20% of the neurons to prevent overtraining.

The second block has 256 filters of size (3,3) but follows the same repetition pattern of the first block. The third block follows the same pattern but with 512 filters and only two repetitions of the three-layer combination consisting of a convolutional layer, LeakyReLU layer, and batch normalization layer.

Finally, it finishes by flattening the output into a 1D vector in preparation for the following dense layers that ultimately output the probabilities for each class, used for binary classification. The Adam optimizer, an adaptive learning rate method commonly used for deep networks [15], is utilized here. It was trained with a batch size of 64 over 10 epochs. See the figure below for a visual representation of the architecture.

While performing transfer learning, the model is unfrozen starting at layer 23 of 40 so that the model layers of the final block can be retrained (one convolutional layer, one LeakyReLU layer, Batch Normalization, Convolutional, LeakyReLU, Batch Normalization, Max pooling, and a dropout layer) and the upper most dense layer block. Essentially, the low to mid-level features are kept the same while allowing the higher up layers to be retrained based on the dataset, which creates better performance since the model is able to identify new patterns from the data. Figure 8 below visualizes the architecture and labels the different layers of the model.

The section below gives an in depth explanation and walk through of M. Fatih Aydogdu and M. Fatih Demircis Age Classifier CNN [16] which uses an extremely unique technique to determine a persons age from an image. Their methodology is then applied to this research problem and replicated.

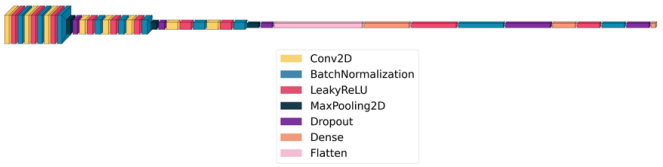

Figure 8: Personal CNN Architecture

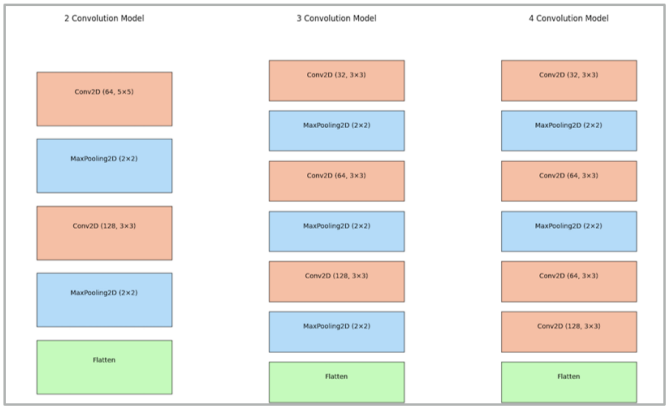

Figure 9: Base Convolutional Models

Age Classification CNN Architecture

This methodology was taken from research conducted by M. Fatih Aydogdu and M. Fatih Demirci, where they used a unique approach to classify age from images using a convolutional neural network. In their paper, they created multiple CNN stages that consist of increasing layers and varying node amounts. Then they combined each layer with each other layer to create an exponential number of models to find the most accurate one. This approach was successful because they ensured that the ending format of the images was common amongst each CNN stage.

The age classification model is included due to its unique architecture and methodology. The systematic combination of multiple CNN models is an interesting way to increase model performance since it allows for experimentation with different amounts of layers and nodes at once. Using a multi-stage approach, helps build a more robust model with varying levels of complexity which ultimately allows for flexible and progressive analysis of fine image details, which is crucial for technical problems like age classification and Deepfake detection.

A 2-layer, 3 layer, and four-layer approach is used to replicate their approach. Next, matrix math is performed to find a combination of layers and pooling that would result in the final image size for all three CNN stages and create the architecture to be the same. However, the approach needed to be adjusted to fit the image sizes for the CIFAKE dataset that differ from theirs.

After building and saving each individual CNN stage, the

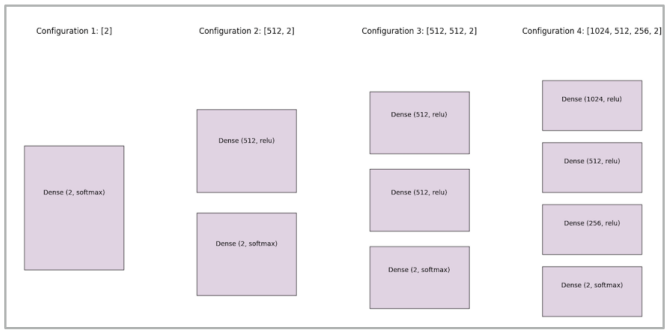

Figure 10: Dense Layer Configurations

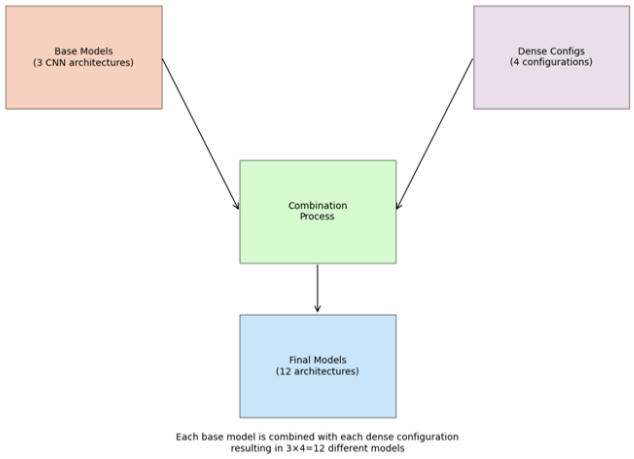

Figure 11: Model Composition Process

magnitude of nodes to be used for each structure was determined; Four different groups were selected. These four nodes will be used on each CNN stage, creating 12 different models. After creating a for loop and looping through the number of nodes, each model was trained and saved in a list of all 12 of them to make training easier later. A for loop was used to cycle through the list of models and their names. It trained them, saved their validation curves as a file, and saved their accuracies in a table as shown for clarity. Changing the number of node groups or CNN groups would have led to different amounts of models, but to not deviate from the original study, 4 node groups and 4 CNN groups are used. This created 12 models of varying convolutional layers and amounts of nodes.

Evaluation Metrics and Modes of Comparison

For each of these models, the preprocessed dataset of real and fake images is trained using each model for ten epochs. The 12 models generated in the Age Classifier Replica were individually trained and the validation accuracies, loss, precision, recall, and f1 score were saved in a table shown in the results section and confusion matrices were generated for the highest and lowest

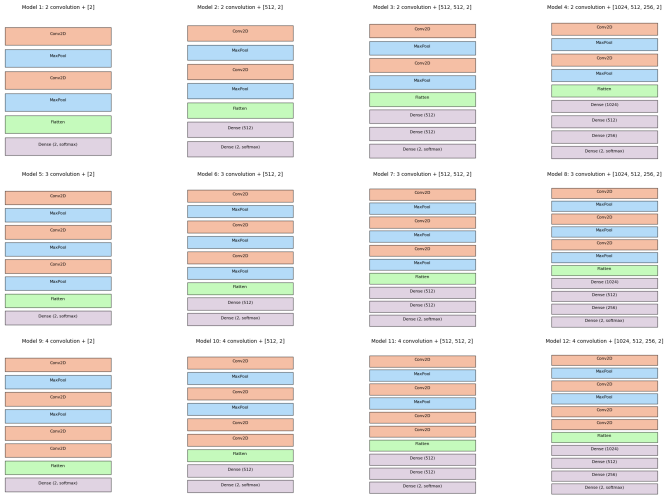

Figure 12: All 12 Final Models for Modified Age Classifier

performing models. The confusion matrices assisted in evaluating model precision, and the accuracies helped generalize the overall performance of all the models.

Analyzing the accuracy of the models helped provide a general sense of model performance, but can be misleading with imbalanced datasets, which is why other metrics are included as well. Validation loss was useful because it quantified the error between the predicted and actual outputs, which helped identify potential areas of over or underfitting by the models.

Precision helps quantify the likelihood of the model correctly identifying a Deepfake image as fake, while Recall similarly indicates how many fake images the models correctly identified. F1 Score is the harmonic mean of precision and recall, which balances false positives and false negatives into a robust score that gives a broader idea of the exactness of the model. Though these three scores are similar, using all of them provides a nuanced understanding of how well the model predicted fake or real images, and provides a more complete and reliable evaluation of model performance in addition to the validation accuracy and loss.

Error Analysis

This section visualizes, analyzes, and compares the saliency maps for ResNet50 (Highest performing SOTA Image Classifier), 4 Convolution [512, 512, 2] (Highest Performing Model from Modified Age Classifier), Personal CNN on CIFAKE Dataset (Original Dataset), and Personal CNN on Manjil Karkis Deepfake and Real Images Dataset (Test Dataset). These models are used for visualizing the saliency maps because they performed the best in each category: The ResNet50 model had the highest performance across all metrics for the pretrained image classifier, the 4 Convolution [512, 512, 2] model had the highest performance out of 12 of the models from the Modified

Age classifier, and the Personal CNN had the highest performance across all models on the CIFAKE Dataset. To make the saliency maps, images are randomly selected based on whether they had a high confidence score or not. Confidence score refers to the level of confidence that the model had when making its predictions for a certain image.

From the saliency map visualizations, a clear pattern emerges within the focuses of the four different models. The personal CNN, which exhibits a higher accuracy on the CIFAKE dataset compared to the ResNet50 and 4 Convolution [512, 512, 2] models, focuses on the central part of the image, typically around key features of the image such as human faces and animal faces. The size of the area of focus is also smaller and contains brighter colored pixels, indicating that the center of the images was extremely crucial to the models predictions. This is good because it means the model is efficient but can also lead to overfitting on limited patterns of images specifically in the CIFAKE dataset. In comparison, the ResNet50 and 4 Convolution [512, 512, 2] models, while also sometimes focusing on the center of the image, typically have a wider area that the model considers in its predictions with less bright pixels. This indicates a less precise and specific area influencing the model predictions, which could have contributed to their lower performance.

ResNet50 in particular, shows saliency maps in Figure 13 that do not exhibit a circular area of focus; For all visualizations displayed, the area of focus by the model is scattered around the image compared to the other models that are clearly circular in the center of the image. The 4 Convolution [512, 512, 2] model also does not display a circular locus of focus but instead seems to consider the whole image. This is a clear difference from the personal CNN, which exhibits a very small area of focus in the form of a circle usually in the center of the image shown.

Also, it can be observed that the model had more difficulty with images where the center object or person was a similar color to the background. In Figure 13 with the personal CNN on the new dataset, the two images that were misclassified showed people in the dark so that there was a less distinct difference between the people and the background. In comparison, the two correct images have a clear difference between the people and the background since they are taken in better lighting with starker lines.

The visualizations for the ResNet50 Model, 4 Convolution [512, 512, 2] model, Personal CNN on CIFAKE dataset, and Personal CNN on Manjil Karki Dataset are shown below. There are four subplots per visualization. Each subplot contains the actual image on the left side by side with the saliency map overlay to the right. All images used for saliency map analysis have high confidence levels (close to or equal to 1.000), which indicates that the model was completely sure about their prediction. This is a helpful metric to consider while analyzing the process of the different models and the role that plays in its overall performance.

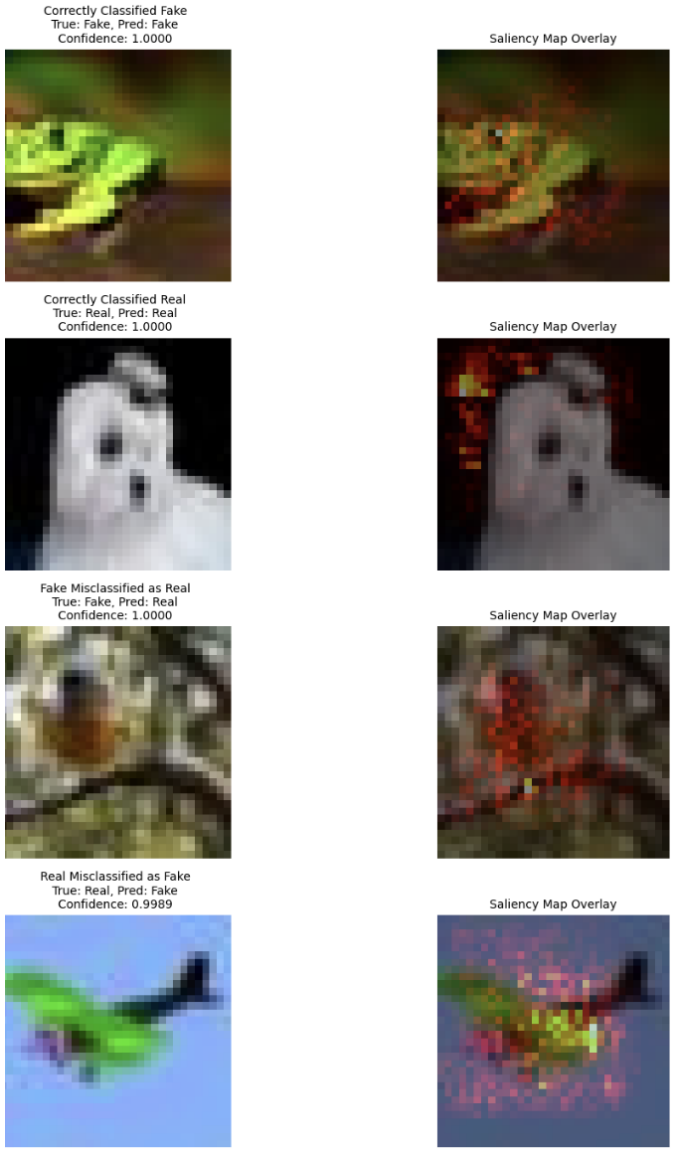

Figure 13: ResNet50 Multiple Saliency Map Overlays and Image

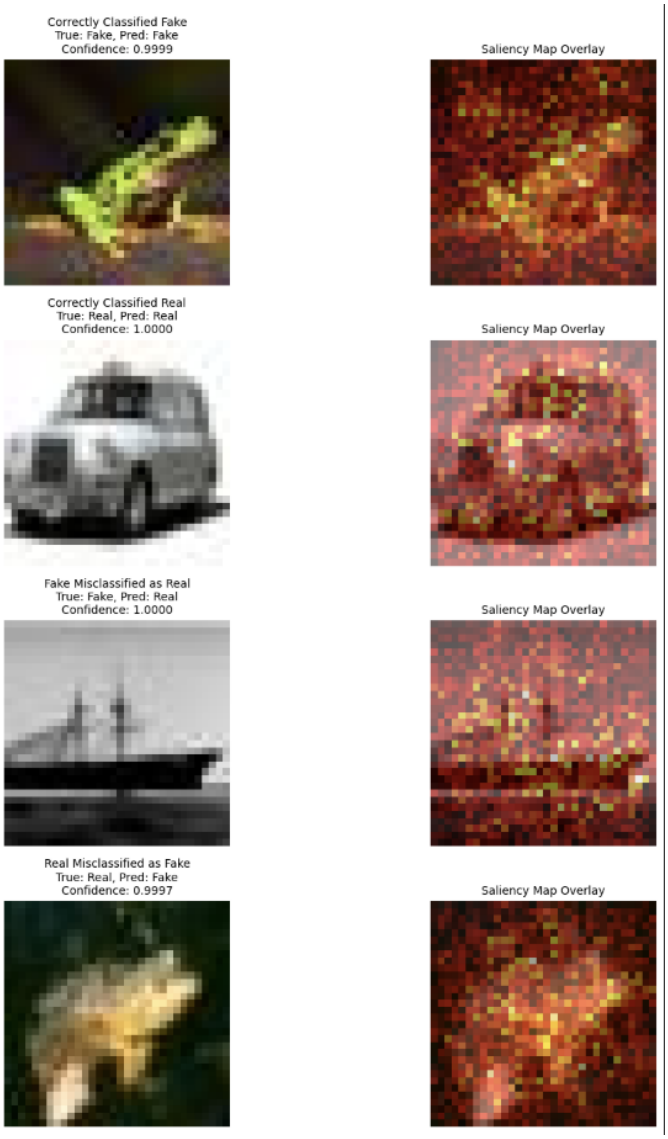

Figure 14: 4 Convolution [512, 512, 2] For Multiple Saliency Map Overlays and Images

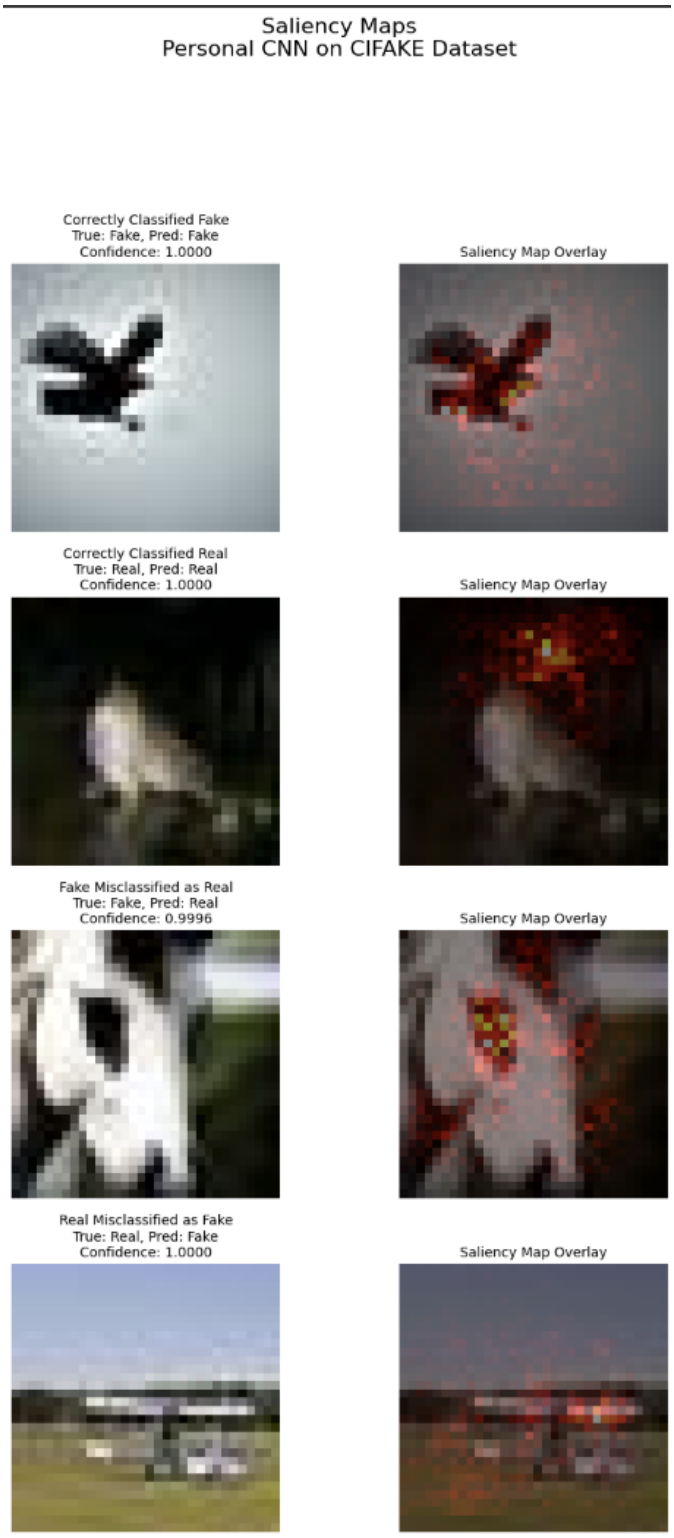

Figure 15: Personal CNN Architecture on CIFAKE Dataset For Multiple Saliency Map Overlays and Images

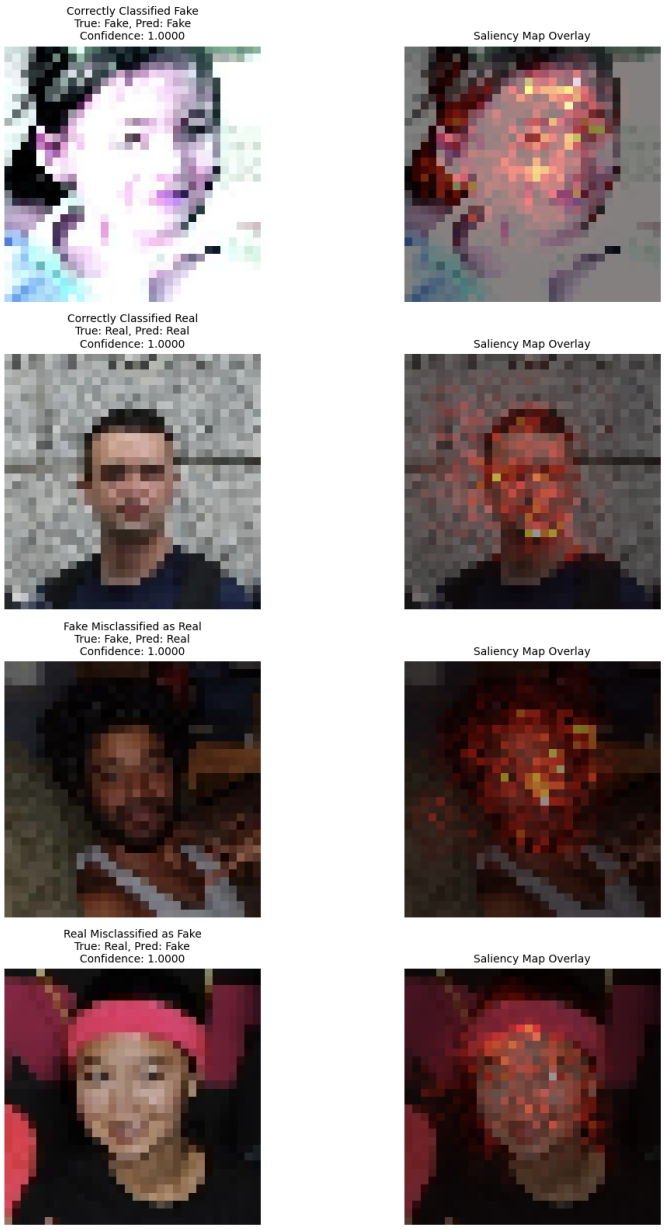

Figure 16: Personal CNN on Manjil Karki Dataset For Multiple Saliency Map Overlays and Images

Overfitting Analysis

Detailed analysis of the training and validation metrics for the pre-trained image classifiers as well as the custom CNNs show that alongside the potential for recognizing underlying patterns in images lies the risk of memorizing instead of analyzing the data.

Comparison of the training accuracy with the validation accuracy after fine tuning for the ResNet50 model show that though that training accuracy continued to increase over the 10 epochs, its validation accuracy fluctuates between 94 and 93 percent accuracy and doesnt get stronger. This is a strong indicator of potential overfitting, where the model is memorizing the training data but is not increasing its performance on unseen validation data. This could occur since the image classifiers were imported with pre-existing weights from prior training on datasets used for image classification, not Deepfake detection. Findings where the models validation accuracy fluctuates between two values while the training accuracy continues to increase are also consistent with the ResNet50 model results for before fine tuning, and the Personal CNN post fine tuning on the original dataset (CIFAKE dataset). The values for each evaluation metric for both training and validation data before and after fine tuning are shown in Figure 17.

Conclusion

For the CIFAKE dataset, the pretrained image classifiers were outperformed by the personal CNN architecture. The best performing model out of them was the RestNet50 Model, with a validation accuracy of 94 percent. This was surprising considering the model had an accuracy of about 97 percent. The age classifier model had accuracies that approximately increased with the number of nodes and number of convolutional layers. The highest accuracy for the age classifier replication was for the 4 convolution [512, 512, 2] model with a validation accuracy of 92.36 percent. However, testing the personal CNN on Manjil Karkis Deepfake and Real Images dataset resulted in a validation accuracy of about 81 percent, a significant drop from its performance on the CIFAKE dataset.

While transfer learning and fine tuning are expected to improve the performance of the models, the results showed that on certain datasets, they can be outperformed by raw CNN architectures without pre-existing weights. This can also be since the weights for the pre-trained image classifiers are determined for image classification, an entirely separate problem than Deepfake detection.

These results contribute to the idea that although pre-trained image classifiers can be applied for Deepfake detection, the lack of specific datasets for this task and lack of SOTA methods for benchmarking creates models prone to overfitting and limited generalizability. Creating the personal CNN was easier and

faster than the industry standard models and even performed better on the CIFAKE dataset but underperformed on Manjil Karkis Deepfake and Real Images Dataset. This is due to potential overfitting with the training dataset, shown by the increase in training validation over 10 epochs while the validation accuracy did not change and often decreased. This calls into question the diversity and strength of current datasets used for Deepfake detection.

However, one limitation of this study is the use of one dataset to train the transfer learning models and the modified age classifier. Although the dataset contains many images, they have all been taken from the same AI generated platform, which calls into question if there is a potential bias skewing the data. All the fake images were generated the equivalent of CIFAR-10 with Stable Diffusion version 1.4, and all the real images were collected from Krizhevsky & Hinton's CIFAR-10 dataset. This may have caused the data to not be representative of all images. Furthermore, the personal CNN was tested on Manjil Karkis Deepfake and Real images to determine its generalizability and found a significant drop in its performance from a validation accuracy of about 97 percent to a validation accuracy of 81 percent, indicating that the models performance is not as strong as previously thought. Additionally, this study was limited to only a few different architectures for testing, and did not explore further beyond the transfer learning models (VGG 16, VGG 19, DenseNet121, ResNet50) and the Age Classifier Model. To stay accurate to the original age classifier model, the Age Classifier methodology did not stray too far from their original design, but did find that the accuracy kept increasing.

Although there are existing datasets standardly used for Deepfake detection tasks, this study highlights the importance of developing evolved datasets that account for the latest changes to generative AI. These newer datasets, alongside clearer SOTA models specifically for this task, will allow for clear benchmarking standards to improve both training and evaluation. Future research should focus on developing more specialized CNN architectures and even experiment with the modified age classifier to see if it will continue to follow the trend of increasing validation accuracy with increasing number of nodes and CNN groups. Further research should test the impacts of changing nodes, CNN groups, and CNN layers.

As deep fake technology continues to evolve and become more undetectable, it becomes clearer that specific models designed to detect fake images are becoming more and more necessary. Though this study works to search for applications of existing models to this new problem, it also encourages the development of more specified models and more substantial datasets to mitigate this problem. Detecting deepfakes is not just a technical challenge but a necessary step in safeguarding digital content and safety.

Acknowledgments

References

1 Y. Patel *et al.*, *IEEE Access*, 2023, **11**, 143296–143323.

2 *Deepfake Detection Using XceptionNet*, https://ieeexplore.ieee.org/document/10363477, 2024, IEEE.

3 *What is a GAN? - Generative Adversarial Networks Explained*, https://aws.amazon.com/what-is/gan/, 2024, Amazon Web Services, Inc.

4 G. Monkam, W. Xu and J. Yan, 2023 26th ACIS International Winter Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD-Winter), 2023, pp. 229–232.

5 *CIFAKE: Real and AI-Generated Synthetic Images*, https://www.kaggle.com/datasets/birdy654/cifake-real-and-ai-generated-synthetic-images, 2024.

6 *Deepfake and Real Images*, https://www.kaggle.com/datasets/manjilkarki/deepfake-and-real-images, 2024.

7 *precision_score*, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html, scikit-learn.

8 *recall_score*, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html, scikit-learn.

9 *f1_score*, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html, scikit-learn.

10 G. Boesch, *Very Deep Convolutional Networks (VGG) Essential Guide*, https://viso.ai/deep-learning/vgg-very-deep-convolutional-networks/, 2021, viso.ai.

11 *What Is Transfer Learning? A Guide for Deep Learning*, https://builtin.com/data-science/transfer-learning, 2024, Built In.

12 S. Das, *Implementing DenseNet-121 in PyTorch: A Step-by-Step Guide*, https://medium.com/deepkapha-notes/implementing-densenet-121-in-pytorch-a-step-by-step-guide-c0c2625c2a60, 2023, deepkapha notes.

13 G. Boesch, *Deep Residual Networks (ResNet, ResNet-50) - 2024 Guide*, https://viso.ai/deep-learning/resnet-residual-neural-network/, 2023, viso.ai.

14 L. Gupta, *Batch Normalization and ReLU for solving Vanishing Gradients*, https://medium.com/analytics-vidhya/how-batch-normalization-and-relu-solve-vanishing-gradients-3f1a8ace1c88, 2021, Analytics Vidhya.

15 *Adam Optimizer - an overview — ScienceDirect Topics*, https://www.sciencedirect.com/topics/computer-science/adam-optimizer, 2024, ScienceDirect.

16 M. F. Aydogdu and M. F. Demirci, Proceedings of the International Conference on Compute and Data Analysis, Lakeland, FL, USA, 2017, pp. 233–239.

Appendix

Figure 17: Table of all Training and Validation Evaluation Metrics Per Epoch Before and After Fine Tuning

Epoch Number:	Validation Loss	Validation Accuracy	Validation Precision	Validation Recall	Validation F1 Score
1	0.8218	0.7968	0.8173	0.7968	0.7944
2	0.6918	0.814	0.814	0.814	0.814
3	0.7282	0.8088	0.8144	0.8088	0.8084
4	0.7506	0.8108	0.8155	0.8108	0.8105
5	0.7647	0.8044	0.8074	0.8044	0.8043
6	0.7409	0.8164	0.8165	0.8164	0.8163
7	0.7355	0.8092	0.8098	0.8092	0.8092
8	0.7369	0.8232	0.824	0.8232	0.8232
9	0.718	0.8112	0.8113	0.8112	0.8112
10	0.9635	0.8112	0.8215	0.8112	0.8103

		BEFORE FINE TUNING										AFTER FINE TUNING									
	Epoch	Training Accuracy	Training F1	Training Loss	Training Precision	Training Recall	Validation Accuracy	Validation F1	Validation Loss	Validation Precision	Validation Recall	Training Accuracy	Training F1	Training Loss	Training Precision	Training Recall	Validation Accuracy	Validation F1	Validation Loss	Validation Precision	Validation Recall
	1	0.5009	0.4986	0.9268	0.5009	0.5009	0.5024	0.3344	0.6931	0.5024	0.5024	0.5004	0.4938	0.6935	0.5004	0.5004	0.5024	0.3344	0.6931	0.5024	0.5024
	2	0.5268	0.5043	0.6784	0.6784	0.5268	0.8786	0.8783	0.2914	0.8786	0.8786	0.5017	0.4944	0.6934	0.5017	0.5017	0.5024	0.3344	0.6931	0.5024	0.5024
	3	0.8675	0.8675	0.3629	0.3629	0.8675	0.8727	0.8721	0.2873	0.8727	0.8727	0.5012	0.4944	0.6934	0.5012	0.5012	0.5024	0.3344	0.6931	0.5024	0.5024
VGG 16 & 19 Architecture	4	0.8836	0.8835	0.2868	0.2868	0.8836	0.9062	0.9062	0.2312	0.9062	0.9062	0.5015	0.4925	0.6933	0.5015	0.5015	0.5024	0.3344	0.6931	0.5024	0.5024
	5	0.8989	0.8988	0.2556	0.2556	0.8989	0.8994	0.8993	0.2361	0.8994	0.8994	0.502	0.4927	0.6933	0.502	0.502	0.5024	0.3344	0.6931	0.5024	0.5024
	6	0.7819	0.7818	0.5022	0.5022	0.7819	0.5024	0.3344	0.6934	0.5024	0.5024	0.5012	0.4915	0.6933	0.5012	0.5012	0.5024	0.3344	0.6931	0.5024	0.5024
	7	0.5001	0.4953	0.6942	0.6942	0.5001	0.5024	0.3344	0.6932	0.5024	0.5024	0.5025	0.4932	0.6933	0.5025	0.5025	0.5024	0.3344	0.6931	0.5024	0.5024
	8	0.5006	0.4945	0.6938	0.6938	0.5006	0.5024	0.3344	0.6932	0.5024	0.5024	0.5017	0.4875	0.6932	0.5017	0.5017	0.5024	0.3344	0.6931	0.5024	0.5024
	9	0.5005	0.4942	0.6937	0.6937	0.5005	0.5024	0.3344	0.6932	0.5024	0.5024	0.5024	0.4858	0.6932	0.5024	0.5024	0.5024	0.3344	0.6931	0.5024	0.5024
	10	0.5007	0.4942	0.6935	0.6935	0.5007	0.5024	0.3344	0.6932	0.5024	0.5024	0.5017	0.4848	0.6932	0.5017	0.5017	0.5024	0.3344	0.6931	0.5024	0.5024
	1	0.7655	0.7654	8.7	0.7655	0.7655	0.7566	0.7478	9.2903	0.7566	0.7566	0.8147	0.8146	3.2673	0.8147	0.8147	0.8218	0.8218	0.611	0.8218	0.8218
	2	0.7772	0.7771	8.9734	0.7772	0.7772	0.7478	0.7384	10.4275	0.7478	0.7478	0.8008	0.8008	0.6637	0.8008	0.8008	0.8172	0.8165	0.5386	0.8172	0.8172
	3	0.7776	0.7775	8.59	0.7776	0.7776	0.7383	0.7264	12.5176	0.7383	0.7383	0.7982	0.7982	0.6314	0.7982	0.7982	0.8162	0.8155	0.5371	0.8162	0.8162
	4	0.7786	0.7786	8.7495	0.7786	0.7786	0.7305	0.7152	11.5866	0.7305	0.7305	0.7985	0.7985	0.6286	0.7985	0.7985	0.8164	0.8157	0.536	0.8164	0.8164
VGG Transfer Learning	5	0.7781	0.778	8.7931	0.7781	0.7781	0.7653	0.7586	8.1137	0.7653	0.7653	0.7986	0.7986	0.6285	0.7986	0.7986	0.8167	0.8161	0.5358	0.8167	0.8167
	6	0.777	0.7769	8.7185	0.777	0.777	0.7558	0.7476	8.2847	0.7558	0.7558	0.7987	0.7987	0.6285	0.7987	0.7987	0.8167	0.8161	0.5356	0.8167	0.8167
	7	0.7759	0.7759	8.772	0.7759	0.7759	0.7503	0.7407	10.3571	0.7503	0.7503	0.7987	0.7987	0.6285	0.7987	0.7987	0.8164	0.8157	0.5355	0.8164	0.8164
	8	0.7769	0.7768	8.7321	0.7769	0.7769	0.7435	0.7327	9.0423	0.7435	0.7435	0.7989	0.7988	0.6284	0.7989	0.7989	0.8164	0.8157	0.5354	0.8164	0.8164
	9	0.7768	0.7767	8.444	0.7768	0.7768	0.7324	0.7178	10.8274	0.7324	0.7324	0.7988	0.7988	0.6284	0.7988	0.7988	0.8163	0.8156	0.5353	0.8163	0.8163
	10	0.7774	0.7774	8.7683	0.7774	0.7774	0.7509	0.7405	10.9039	0.7509	0.7509	0.7988	0.7987	0.6284	0.7988	0.7988	0.8163	0.8156	0.5352	0.8163	0.8163
	1	0.7821	0.7821	0.7568	0.7821	0.7821	0.861	0.8609	0.359	0.861	0.861	0.8446	0.8446	0.4078	0.8446	0.8446	0.8602	0.8599	0.3517	0.8602	0.8602
	2	0.8353	0.8353	0.4256	0.8353	0.8353	0.844	0.843	0.4019	0.844	0.844	0.8437	0.8437	0.4091	0.8437	0.8437	0.8678	0.8678	0.3301	0.8678	0.8678
	3	0.8385	0.8385	0.417	0.8385	0.8385	0.8557	0.8556	0.3726	0.8557	0.8557	0.8451	0.8451	0.4026	0.8451	0.8451	0.8525	0.8518	0.3864	0.8525	0.8525
	4	0.8369	0.8369	0.442	0.8369	0.8369	0.8298	0.8278	0.4332	0.8298	0.8298	0.844	0.844	0.4105	0.844	0.844	0.8555	0.8555	0.3881	0.8555	0.8555
DenseNet121 Transfer Learning	5	0.8367	0.8367	0.4211	0.8367	0.8367	0.8568	0.8563	0.3696	0.8568	0.8568	0.8429	0.8429	0.4109	0.8429	0.8429	0.8324	0.8302	0.4268	0.8324	0.8324
	6	0.8389	0.8389	0.4214	0.8389	0.8389	0.8231	0.82	0.4774	0.8231	0.8231	0.8467	0.8467	0.4006	0.8467	0.8467	0.8663	0.8662	0.3424	0.8663	0.8663
	7	0.8408	0.8408	0.4178	0.8408	0.8408	0.832	0.83	0.4356	0.832	0.832	0.8445	0.8445	0.4024	0.8445	0.8445	0.8462	0.8448	0.4452	0.8462	0.8462
	8	0.8424	0.8428	0.4102	0.8424	0.8424	0.8559	0.8559	0.3649	0.8559	0.8559	0.8481	0.8481	0.3932	0.8481	0.8481	0.8329	0.8309	0.4568	0.8329	0.8329
	9	0.8386	0.8386	0.4178	0.8386	0.8386	0.8618	0.8618	0.3445	0.8618	0.8618	0.8424	0.8424	0.4074	0.8424	0.8424	0.8285	0.8261	0.4601	0.8285	0.8285
	10	0.8403	0.8403	0.4212	0.8403	0.8403	0.8503	0.8494	0.3847	0.8503	0.8503	0.8462	0.8462	0.3984	0.8462	0.8462	0.8406	0.8399	0.4263	0.8406	0.8406
	1	0.8145	0.8144	0.4234	0.8145	0.8145	0.8861	0.8861	0.2812	0.8861	0.8861	0.942	0.942	0.1503	0.942	0.942	0.9409	0.9409	0.1623	0.9409	0.9409
	2	0.8949	0.8949	0.2662	0.8949	0.8949	0.8834	0.8834	0.3099	0.8834	0.8834	0.9445	0.9445	0.14	0.9445	0.9445	0.9386	0.9385	0.1776	0.9386	0.9386
	3	0.9096	0.9096	0.2356	0.9096	0.9096	0.8632	0.8632	0.3076	0.8632	0.8632	0.9477	0.9477	0.1364	0.9477	0.9477	0.9373	0.9372	0.1667	0.9373	0.9373
	4	0.9163	0.9163	0.2181	0.9163	0.9163	0.8959	0.8959	0.2617	0.8959	0.8959	0.9491	0.9491	0.1305	0.9491	0.9491	0.9442	0.9442	0.1489	0.9442	0.9442
ResNet50 Transfer Learning	5	0.9216	0.9216	0.2051	0.9216	0.9216	0.912	0.912	0.2252	0.912	0.912	0.9502	0.9502	0.1271	0.9502	0.9502	0.9427	0.9427	0.1589	0.9427	0.9427
	6	0.9238	0.9238	0.1987	0.9238	0.9238	0.9283	0.9283	0.1927	0.9283	0.9283	0.9511	0.9511	0.1276	0.9511	0.9511	0.9379	0.9379	0.1603	0.9379	0.9379
	7	0.9283	0.9282	0.1843	0.9283	0.9283	0.9164	0.9163	0.2081	0.9164	0.9164	0.9518	0.9518	0.1251	0.9518	0.9518	0.9406	0.9405	0.1718	0.9406	0.9406
	8	0.9314	0.9314	0.1783	0.9314	0.9314	0.9287	0.9286	0.1805	0.9287	0.9287	0.9531	0.9531	0.1212	0.9531	0.9531	0.9392	0.9391	0.1684	0.9392	0.9392
	9	0.933	0.933	0.1738	0.933	0.933	0.9036	0.9032	0.2628	0.9036	0.9036	0.9539	0.9539	0.1209	0.9539	0.9539	0.9387	0.9386	0.1613	0.9387	0.9387
	10	0.9344	0.9344	0.1702	0.9344	0.9344	0.9151	0.9148	0.2083	0.9151	0.9151	0.9529	0.9529	0.1197	0.9529	0.9529	0.9398	0.9397	0.164	0.9398	0.9398
	1	0.8538	0.8537	0.4022	0.8538	0.8538	0.9227	0.9226	0.2027	0.9227	0.9227	0.9753	0.9753	0.0686	0.9753	0.9753	0.9684	0.9684	0.0981	0.9684	0.9684
	2	0.9312	0.9312	0.1091	0.9312	0.9312	0.909	0.9086	0.2515	0.909	0.909	0.9802	0.9802	0.0556	0.9802	0.9802	0.9695	0.9695	0.0881	0.9695	0.9695
	3	0.9412	0.9412	0.1558	0.9412	0.9412	0.9317	0.9315	0.1853	0.9317	0.9317	0.9829	0.9829	0.0479	0.9829	0.9829	0.9595	0.9595	0.096	0.9595	0.9595
	4	0.9437	0.9437	0.145	0.9437	0.9437	0.9538	0.9538	0.1254	0.9538	0.9538	0.9853	0.9853	0.0398	0.9853	0.9853	0.9687	0.9687	0.1046	0.9687	0.9687
Personal CNN on CIFAKE Dataset	5	0.9506	0.9506	0.1305	0.9506	0.9506	0.9523	0.9523	0.1306	0.9523	0.9523	0.9883	0.9883	0.0337	0.9883	0.9883	0.9686	0.9686	0.1033	0.9686	0.9686
	6	0.9543	0.9543	0.1203	0.9543	0.9543	0.9268	0.9267	0.2075	0.9268	0.9268	0.9898	0.9898	0.029	0.9898	0.9898	0.9693	0.9692	0.1126	0.9693	0.9693
	7	0.9669	0.9669	0.0849	0.9669	0.9669	0.9647	0.9647	0.0972	0.9647	0.9647	0.9908	0.9908	0.0261	0.9908	0.9908	0.9698	0.9698	0.1187	0.9698	0.9698
	8	0.974	0.974	0.0691	0.974	0.974	0.9672	0.9672	0.0923	0.9672	0.9672	0.9917	0.9917	0.0245	0.9917	0.9917	0.9682	0.9682	0.1384	0.9682	0.9682
	9	0.978	0.978	0.059	0.978	0.978	0.9684	0.9684	0.0899	0.9684	0.9684	0.9927	0.9928	0.0195	0.9927	0.9927	0.9678	0.9678	0.137	0.9678	0.9678
	10	0.9793	0.9793	0.0543	0.9793	0.9793	0.9702	0.9702	0.0914	0.9702	0.9702	0.9935	0.9935	0.0174	0.9935	0.9935	0.9702	0.9702	0.1479	0.9702	0.9702

Figure 18: Evaluation Metrics Per Epoch for Fine Tuning Personal CNN on Manjil Karki Dataset

Epoch	Model	Training Accuracy	Training F1	Training Loss	Training Precision	Training Recall	Validation Accuracy	Validation F1	Validation Loss	Validation Precision	Validation Recall
1	2 convolution [2]	0.788989604	0.78474957 0.79306567	0.5307711	0.7889896	0.7889896	0.7847	0.7479515 0.81204534	0.46884793	0.78466666	0.78466666
2	2 convolution [2]	0.836218774	0.8331547 0.8391723	0.3739428	0.83621877	0.83621877	0.852	0.85810155 0.84535	0.35651657	0.852	0.852
3	2 convolution [2]	0.847874999	0.84482974 0.8508029	0.3544433	0.847875	0.847875	0.8431	0.8356463 0.84987634	0.35761303	0.84308332	0.84308332
4	2 convolution [2]	0.855406225	0.8528417 0.85788286	0.3362458	0.85540622	0.85540622	0.8677	0.8647359 0.8704731	0.310826	0.86766666	0.86766666
5	2 convolution [2]	0.859802067	0.8575299 0.8620028	0.3276378	0.85980207	0.85980207	0.8668	0.861041 0.87200826	0.31602463	0.86675	0.86675
6	2 convolution [2]	0.866104186	0.8638174 0.8683153	0.3167645	0.86610419	0.86610419	0.8572	0.8485597 0.8648478	0.3294017	0.85716665	0.85716665
7	2 convolution [2]	0.868697941	0.8665841 0.8707458	0.3119476	0.86869794	0.86869794	0.8562	0.8573317 0.85498226	0.33797687	0.85616666	0.85616666
8	2 convolution [2]	0.87408334	0.8719789 0.8761196	0.3027943	0.87408334	0.87408334	0.8629	0.8594617 0.86620575	0.32725355	0.86291665	0.86291665
9	2 convolution [2]	0.873916686	0.87192607 0.8758462	0.2993332	0.87391669	0.87391669	0.8733	0.86952776 0.876923	0.30836919	0.87333333	0.87333333
10	2 convolution [2]	0.876114607	0.8741973 0.8779742	0.2962695	0.87611461	0.87611461	0.8618	0.863423 0.86003536	0.33908108	0.86175001	0.86175001
1	2 convolution [512, 2]	0.877499998	0.8767941 0.8781978	0.294995	0.8775	0.8775	0.8743	0.87466747 0.8739972	0.29964435	0.87433332	0.87433332
2	2 convolution [512, 2]	0.890812516	0.8901396 0.89147717	0.2657129	0.89081252	0.89081252	0.8879	0.8847965 0.8908722	0.27310556	0.88791668	0.88791668
3	2 convolution [512, 2]	0.898343742	0.89782536 0.89885676	0.2496742	0.89834374	0.89834374	0.8897	0.8896114 0.8897217	0.26315969	0.88966668	0.88966668
4	2 convolution [512, 2]	0.901864588	0.90132076 0.90240234	0.2425793	0.90186459	0.90186459	0.8974	0.89723676 0.8975959	0.25416011	0.89741665	0.89741665
5	2 convolution [512, 2]	0.905291677	0.90473187 0.90584475	0.2333247	0.90529168	0.90529168	0.8987	0.8995539 0.89776355	0.25744644	0.89866668	0.89866668
6	2 convolution [512, 2]	0.908541679	0.90805125 0.9090268	0.2256881	0.90854168	0.90854168	0.8975	0.9010298 0.8937088	0.26932913	0.89749998	0.89749998
7	2 convolution [512, 2]	0.911052108	0.91054606 0.9115523	0.2213363	0.91105211	0.91105211	0.8961	0.8967629 0.8953946	0.25873128	0.89608335	0.89608335
8	2 convolution [512, 2]	0.913552105	0.91306394 0.91403466	0.2155704	0.91355211	0.91355211	0.8982	0.8972308 0.899249	0.25451669	0.89824998	0.89824998
9	2 convolution [512, 2]	0.916531265	0.9159578 0.91709685	0.2081763	0.91653126	0.91653126	0.8996	0.89789 0.90122133	0.25563014	0.89958334	0.89958334
10	2 convolution [512, 2]	0.91980207	0.91935086 0.92024815	0.2008466	0.91980207	0.91980207	0.8852	0.89142764 0.87813926	0.29854137	0.88516665	0.88516665

1	2 convolution [512, 512, 2]	0.90360415	0.90339077 0.90381646	0.2605672	0.90360415	0.90360415	0.8923	0.8912073 0.8932727	0.2925795	0.89225	0.89225
2	2 convolution [512, 512, 2]	0.915802062	0.9156254 0.9159779	0.208997	0.91580206	0.91580206	0.8938	0.89258003 0.8950576	0.270392	0.89383334	0.89383334
3	2 convolution [512, 512, 2]	0.920822918	0.9205805 0.92106384	0.1985895	0.92082292	0.92082292	0.8933	0.8909338 0.8956294	0.27322865	0.89333332	0.89333332
4	2 convolution [512, 512, 2]	0.923489571	0.92315245 0.9238236	0.189956	0.92348957	0.92348957	0.8957	0.89757854 0.89368206	0.27458662	0.89566666	0.89566666
5	2 convolution [512, 512, 2]	0.928218722	0.92794174 0.92849356	0.1795121	0.92821872	0.92821872	0.8967	0.899814 0.8934926	0.26836169	0.89674997	0.89674997
6	2 convolution [512, 512, 2]	0.931697905	0.93130076 0.9320904	0.1705223	0.93169791	0.93169791	0.9093	0.90990317 0.9085872	0.23966083	0.90925002	0.90925002
7	2 convolution [512, 512, 2]	0.935520828	0.9351831 0.93585485	0.1620712	0.93552083	0.93552083	0.9008	0.9017341 0.89991593	0.25651225	0.90083331	0.90083331
8	2 convolution [512, 512, 2]	0.938885391	0.93852097 0.9392454	0.1535343	0.93888539	0.93888539	0.9021	0.90024614 0.90385395	0.2851826	0.90208334	0.90208334
9	2 convolution [512, 512, 2]	0.940989554	0.94064516 0.94132996	0.1487781	0.94098955	0.94098955	0.9052	0.90446603 0.905857	0.30126214	0.90516669	0.90516669
10	2 convolution [512, 512, 2]	0.944729149	0.9444281 0.9450269	0.1392965	0.94472915	0.94472915	0.9014	0.9040628 0.8986202	0.29535681	0.90141666	0.90141666
1	2 convolution [1024, 512, 256, 2]	0.914677084	0.91436934 0.91498256	0.263503	0.91467708	0.91467708	0.8963	0.89877224 0.8935988	0.26859054	0.89625001	0.89625001
2	2 convolution [1024, 512, 256, 2]	0.929614604	0.9293961 0.9298316	0.1778875	0.9296146	0.9296146	0.9028	0.9034339 0.90205616	0.25424725	0.90275002	0.90275002
3	2 convolution [1024, 512, 256, 2]	0.936333358	0.93612164 0.93654346	0.1616159	0.93633336	0.93633336	0.9007	0.8977174 0.9034504	0.33738667	0.90066665	0.90066665
4	2 convolution [1024, 512, 256, 2]	0.9395625	0.9391772 0.9399428	0.1522067	0.9395625	0.9395625	0.899	0.89616174 0.9016872	0.27564654	0.89899999	0.89899999
5	2 convolution [1024, 512, 256, 2]	0.943562508	0.9432087 0.94391185	0.1433329	0.94356251	0.94356251	0.9069	0.9075407 0.9062841	0.2685158	0.90691668	0.90691668

6	2 convolution [1024, 512, 256, 2]	0.947583318	0.9473286 0.9478354	0.1342261	0.94758332	0.94758332	0.9025	0.9031135 0.9018785	0.30235219	0.90249997	0.90249997
7	2 convolution [1024, 512, 256, 2]	0.951156259	0.9508752 0.95143396	0.1255684	0.95115626	0.95115626	0.9024	0.9008718 0.9039139	0.30426848	0.90241665	0.90241665
8	2 convolution [1024, 512, 256, 2]	0.954406261	0.9541113 0.9546974	0.1166738	0.95440626	0.95440626	0.9055	0.9052631 0.9057356	0.36407983	0.90549999	0.90549999
9	2 convolution [1024, 512, 256, 2]	0.956531227	0.9561627 0.9568935	0.1105272	0.95653123	0.95653123	0.9034	0.90396875 0.9028581	0.31792954	0.90341669	0.90341669
10	2 convolution [1024, 512, 256, 2]	0.960906267	0.9606747 0.96113497	0.1010118	0.96090627	0.96090627	0.8992	0.90309143 0.89491045	0.37997547	0.89916664	0.89916664
1	3 convolution [2]	0.824947894	0.82221633 0.8275968	0.4699504	0.82494789	0.82494789	0.876	0.8743879 0.8775711	0.2986407	0.87599999	0.87599999
2	3 convolution [2]	0.873374999	0.87207705 0.8746468	0.303154	0.873375	0.873375	0.8787	0.88704413 0.86894685	0.29633519	0.87866664	0.87866664
3	3 convolution [2]	0.885406256	0.8843351 0.8864576	0.278165	0.88540626	0.88540626	0.8917	0.89231527 0.8911786	0.26814547	0.89174998	0.89174998
4	3 convolution [2]	0.894093752	0.8930633 0.8951044	0.2597511	0.89409375	0.89409375	0.881	0.8732693 0.88784164	0.27643648	0.88099998	0.88099998
5	3 convolution [2]	0.900291681	0.89940506 0.90116256	0.2467767	0.90029168	0.90029168	0.8885	0.88310325 0.8934204	0.26756758	0.88849998	0.88849998
6	3 convolution [2]	0.90417707	0.9033281 0.9050112	0.2377152	0.90417707	0.90417707	0.8973	0.89728194 0.8973846	0.25060147	0.89733332	0.89733332
7	3 convolution [2]	0.908562481	0.9077281 0.90938175	0.2287647	0.90856248	0.90856248	0.8805	0.87226075 0.8877406	0.28034496	0.88050002	0.88050002
8	3 convolution [2]	0.91038543	0.90951157 0.91124237	0.2233876	0.91038543	0.91038543	0.9079	0.9062208 0.9095522	0.24345693	0.90791667	0.90791667
9	3 convolution [2]	0.91222918	0.9113705 0.9130712	0.2199629	0.91222918	0.91222918	0.9032	0.90572464 0.9006418	0.25256103	0.90324998	0.90324998
10	3 convolution [2]	0.913635433	0.91276383 0.91448957	0.2137071	0.91363543	0.91363543	0.9109	0.9118205 0.90999407	0.26440653	0.91091669	0.91091669
1	3 convolution [512, 2]	0.915104151	0.91490203 0.9153052	0.218658	0.91510415	0.91510415	0.9068	0.909503 0.9040013	0.2358119	0.90683335	0.90683335
2	3 convolution [512, 2]	0.922979176	0.9228908 0.9230673	0.1932107	0.92297918	0.92297918	0.9083	0.9088649 0.9077954	0.23260929	0.90833336	0.90833336
3	3 convolution [512, 2]	0.926802099	0.9267158 0.9268881	0.1844657	0.9268021	0.9268021	0.9138	0.9154399 0.9121644	0.22672021	0.91383332	0.91383332

4	3 convolution [512, 2]	0.930541694	0.93049526 0.9305879	0.1765732	0.93054169	0.93054169	0.9103	0.90995806 0.9107054	0.24702165	0.91033334	0.91033334
5	3 convolution [512, 2]	0.932187498	0.9322002 0.93217474	0.1714786	0.9321875	0.9321875	0.905	0.9010588 0.9086392	0.24204436	0.90499997	0.90499997
6	3 convolution [512, 2]	0.93589586	0.93586504 0.93592644	0.1635885	0.93589586	0.93589586	0.8969	0.9021128 0.8911379	0.26776287	0.89691669	0.89691669
7	3 convolution [512, 2]	0.934906244	0.9347859 0.93502593	0.1667564	0.93490624	0.93490624	0.9139	0.91192764 0.9158177	0.2161483	0.91391665	0.91391665
8	3 convolution [512, 2]	0.939499974	0.9393863 0.93961316	0.1549075	0.93949997	0.93949997	0.9126	0.91073096 0.9143602	0.26077133	0.91258335	0.91258335
9	3 convolution [512, 2]	0.941124976	0.94101304 0.9412363	0.1498104	0.94112498	0.94112498	0.9168	0.9161842 0.9173081	0.21813864	0.91675001	0.91675001
10	3 convolution [512, 2]	0.943385422	0.94323933 0.9435307	0.1457279	0.94338542	0.94338542	0.9052	0.91045004 0.89922065	0.25939688	0.90516669	0.90516669
1	3 convolution [512, 512, 2]	0.940687478	0.9405872 0.9407874	0.2158384	0.94068748	0.94068748	0.9126	0.910594 0.91448593	0.23283741	0.91258335	0.91258335
2	3 convolution [512, 512, 2]	0.945395827	0.9453023 0.94548887	0.1410887	0.94539583	0.94539583	0.9183	0.9178598 0.91863644	0.24214751	0.91825002	0.91825002
3	3 convolution [512, 512, 2]	0.945947945	0.945813 0.9460821	0.1388552	0.94594795	0.94594795	0.9128	0.9116529 0.91382	0.22175165	0.91275001	0.91275001
4	3 convolution [512, 512, 2]	0.94771874	0.94756293 0.9478734	0.1333048	0.94771874	0.94771874	0.913	0.9153284 0.9105398	0.22404693	0.91299999	0.91299999
5	3 convolution [512, 512, 2]	0.950614572	0.9504063 0.950821	0.1272133	0.95061457	0.95061457	0.9187	0.9196046 0.9177065	0.24162309	0.91866666	0.91866666
6	3 convolution [512, 512, 2]	0.951166689	0.9508739 0.95145583	0.1243553	0.95116669	0.95116669	0.9153	0.91555846 0.9151069	0.22500971	0.91533333	0.91533333
7	3 convolution [512, 512, 2]	0.953999996	0.95381814 0.95418036	0.1192809	0.954	0.954	0.9178	0.9175783 0.917921	0.27619636	0.91775	0.91775
8	3 convolution [512, 512, 2]	0.941593766	0.94090354 0.94226784	0.1528568	0.94159377	0.94159377	0.9118	0.9102629 0.9133496	0.23878926	0.91183335	0.91183335
9	3 convolution [512, 512, 2]	0.957312524	0.9570764 0.95754594	0.1107506	0.95731252	0.95731252	0.9154	0.91467 0.9161502	0.27671066	0.91541666	0.91541666
10	3 convolution [512, 512, 2]	0.957781255	0.95750993 0.9580491	0.1115583	0.95778126	0.95778126	0.9159	0.9135167 0.91818696	0.2499795	0.91591668	0.91591668
1	3 convolution [1024, 512, 256, 2]	0.955020845	0.9549353 0.95510584	0.160582	0.95502084	0.95502084	0.9178	0.91717064 0.91848534	0.30675942	0.91783333	0.91783333

2	3 convolution [1024, 512, 256, 2]	0.959427059	0.9594089 0.95944524	0.1065118	0.95942706	0.95942706	0.9165	0.9182841 0.9146362	0.23219275	0.91649997	0.91649997
3	3 convolution [1024, 512, 256, 2]	0.95990622	0.9597574 0.960054	0.1055173	0.95990622	0.95990622	0.9168	0.9193899 0.9139312	0.24641497	0.91675001	0.91675001
4	3 convolution [1024, 512, 256, 2]	0.959354162	0.95924795 0.9594597	0.1052728	0.95935416	0.95935416	0.914	0.9126903 0.91527086	0.30685768	0.91399997	0.91399997
5	3 convolution [1024, 512, 256, 2]	0.958906233	0.95874596 0.95906526	0.1103718	0.95890623	0.95890623	0.9047	0.90064263 0.90837735	0.31684148	0.90466666	0.90466666
6	3 convolution [1024, 512, 256, 2]	0.960989595	0.9607479 0.9612282	0.1034229	0.96098959	0.96098959	0.9139	0.9144513 0.9133752	0.26623866	0.91391665	0.91391665
7	3 convolution [1024, 512, 256, 2]	0.962583363	0.9623866 0.9627779	0.098196	0.96258336	0.96258336	0.9186	0.9190354 0.9181262	0.35842156	0.91858333	0.91858333
8	3 convolution [1024, 512, 256, 2]	0.964739561	0.96453524 0.96494156	0.0927844	0.96473956	0.96473956	0.9158	0.91424197 0.91720575	0.63828677	0.91575003	0.91575003
9	3 convolution [1024, 512, 256, 2]	0.951760411	0.95137995 0.9521348	0.1289576	0.95176041	0.95176041	0.9057	0.9087096 0.9024138	0.28610879	0.90566665	0.90566665
10	3 convolution [1024, 512, 256, 2]	0.963645816	0.9634179 0.9638708	0.096591	0.96364582	0.96364582	0.9178	0.91954017 0.91587824	0.40920845	0.91775	0.91775
1	4 convolution [2]	0.825385392	0.82519776 0.82557255	0.4069786	0.82538539	0.82538539	0.8787	0.87836254 0.87896913	0.29182222	0.87866664	0.87866664
2	4 convolution [2]	0.878697932	0.8785397 0.8788556	0.2918336	0.87869793	0.87869793	0.8927	0.89300543 0.8923257	0.27339843	0.89266664	0.89266664
3	4 convolution [2]	0.893468738	0.89320725 0.89372885	0.2612748	0.89346874	0.89346874	0.9028	0.90410054 0.90136075	0.24554966	0.90275002	0.90275002
4	4 convolution [2]	0.902125001	0.9018448 0.90240353	0.2444034	0.902125	0.902125	0.8961	0.89981514 0.8920626	0.25723475	0.89608335	0.89608335
5	4 convolution [2]	0.908604145	0.90820444 0.9090003	0.2281207	0.90860415	0.90860415	0.8957	0.8909613 0.90013564	0.24817251	0.89574999	0.89574999
6	4 convolution [2]	0.912374973	0.91196597 0.9127801	0.2203645	0.91237497	0.91237497	0.905	0.9031106 0.9068171	0.23909487	0.90499997	0.90499997
7	4 convolution [2]	0.915635407	0.9153488 0.91592	0.2122299	0.91563541	0.91563541	0.9058	0.9018074 0.9095421	0.23956862	0.9058333	0.9058333

8	4 convolution [2]	0.920104146	0.919857 0.9203498	0.2013357	0.92010415	0.92010415	0.9156	0.9156605 0.9155058	0.21526207	0.91558331	0.91558331
9	4 convolution [2]	0.922031224	0.92166644 0.92239255	0.1971539	0.92203122	0.92203122	0.923	0.923712 0.92227453	0.20072971	0.92299998	0.92299998
10	4 convolution [2]	0.924822927	0.92449015 0.92515266	0.1889171	0.92482293	0.92482293	0.9123	0.9096375 0.9148729	0.23160587	0.91233331	0.91233331
1	4 convolution [512, 2]	0.921937525	0.92195374 0.9219212	0.2024347	0.92193753	0.92193753	0.9157	0.9171849 0.91409165	0.21685731	0.91566664	0.91566664
2	4 convolution [512, 2]	0.927708328	0.92784804 0.92756796	0.1824732	0.92770833	0.92770833	0.9133	0.9140308 0.9124547	0.2153962	0.91325003	0.91325003
3	4 convolution [512, 2]	0.932031274	0.9321308 0.9319312	0.1727632	0.93203127	0.93203127	0.8901	0.89893484 0.87953234	0.25054652	0.89008331	0.89008331
4	4 convolution [512, 2]	0.932979167	0.9330767 0.93288124	0.1701388	0.93297917	0.93297917	0.9234	0.923525 0.92330796	0.19966917	0.92341667	0.92341667
5	4 convolution [512, 2]	0.934354186	0.93445516 0.93425274	0.1676292	0.93435419	0.93435419	0.9228	0.9233063 0.92235446	0.21751548	0.92283332	0.92283332
6	4 convolution [512, 2]	0.937343776	0.937377 0.9373104	0.158438	0.93734378	0.93734378	0.9218	0.92200345 0.9214948	0.21298091	0.92175001	0.92175001
7	4 convolution [512, 2]	0.938145816	0.9381045 0.93818694	0.1570591	0.93814582	0.93814582	0.9266	0.92703927 0.9261215	0.19250248	0.92658335	0.92658335
8	4 convolution [512, 2]	0.941885412	0.94176704 0.94200313	0.1504942	0.94188541	0.94188541	0.9125	0.91780174 0.9064671	0.23308511	0.91250002	0.91250002
9	4 convolution [512, 2]	0.942218721	0.9421772 0.9422602	0.14907	0.94221872	0.94221872	0.9267	0.9279868 0.9252971	0.20453076	0.92666668	0.92666668
10	4 convolution [512, 2]	0.943750024	0.9436372 0.9438622	0.1424073	0.94375002	0.94375002	0.9188	0.9226492 0.91462123	0.22888085	0.91883332	0.91883332
1	4 convolution [512, 512, 2]	0.946416676	0.946335 0.946498	0.1591994	0.94641668	0.94641668	0.925	0.925187 0.92481196	0.20051064	0.92500001	0.92500001
2	4 convolution [512, 512, 2]	0.94784373	0.94783884 0.9478486	0.1340954	0.94784373	0.94784373	0.9252	0.9245124 0.92580956	0.2024491	0.92516667	0.92516667
3	4 convolution [512, 512, 2]	0.945666671	0.94552934 0.9458032	0.138993	0.94566667	0.94566667	0.9258	0.9266644 0.9249831	0.20809408	0.92583334	0.92583334
4	4 convolution [512, 512, 2]	0.948343754	0.9482245 0.94846237	0.1324655	0.94834375	0.94834375	0.9253	0.92754316 0.9229843	0.20810941	0.92533332	0.92533332
5	4 convolution [512, 512, 2]	0.951145828	0.9510459 0.9512453	0.1265738	0.95114583	0.95114583	0.9184	0.9158428 0.92083764	0.24437803	0.91841668	0.91841668
6	4 convolution [512, 512, 2]	0.948614597	0.94847333 0.94875497	0.1358263	0.9486146	0.9486146	0.9235	0.923994 0.9229994	0.22352879	0.9235	0.9235

7	4 convolution [512, 512, 2]	0.95358336	0.9534641 0.95370185	0.1194251	0.95358336	0.95358336	0.9243	0.92454547 0.92395204	0.20587842	0.92425001	0.92425001
8	4 convolution [512, 512, 2]	0.954187512	0.95410717 0.95426744	0.1198573	0.95418751	0.95418751	0.9261	0.92690563 0.9252422	0.23839986	0.92608333	0.92608333
9	4 convolution [512, 512, 2]	0.954583347	0.95443803 0.95472753	0.1171446	0.95458335	0.95458335	0.9189	0.9187609 0.91907173	0.23677161	0.91891664	0.91891664
10	4 convolution [512, 512, 2]	0.955364585	0.95523065 0.9554976	0.1150082	0.95536458	0.95536458	0.9237	0.92419726 0.9231285	0.226338	0.92366666	0.92366666
1	4 convolution [1024, 512, 256, 2]	0.96047914	0.96037346 0.9605842	0.1350988	0.96047914	0.96047914	0.9178	0.91562545 0.9199285	0.25295427	0.91783333	0.91783333
2	4 convolution [1024, 512, 256, 2]	0.960218728	0.9602125 0.96022487	0.1046333	0.96021873	0.96021873	0.9247	0.9251035 0.92422456	0.2158418	0.92466664	0.92466664
3	4 convolution [1024, 512, 256, 2]	0.956416667	0.9564003 0.95643294	0.1141225	0.95641667	0.95641667	0.9213	0.92248315 0.9201488	0.29025495	0.92133331	0.92133331
4	4 convolution [1024, 512, 256, 2]	0.960489571	0.9603735 0.96060485	0.1049497	0.96048957	0.96048957	0.9137	0.91044253 0.9166666	0.24000394	0.91366667	0.91366667
5	4 convolution [1024, 512, 256, 2]	0.957104147	0.95709336 0.9571148	0.1137457	0.95710415	0.95710415	0.9214	0.9218788 0.92094886	0.21314529	0.92141664	0.92141664
6	4 convolution [1024, 512, 256, 2]	0.958552063	0.95834154 0.9587603	0.1075858	0.95855206	0.95855206	0.9243	0.92524284 0.9234014	0.2382298	0.92433333	0.92433333
7	4 convolution [1024, 512, 256, 2]	0.961687505	0.96152073 0.9618527	0.0995631	0.96168751	0.96168751	0.9199	0.9189097 0.92089874	0.24683224	0.91991669	0.91991669
8	4 convolution [1024, 512, 256, 2]	0.95863539	0.9584166 0.9588518	0.1074529	0.95863539	0.95863539	0.921	0.9190434 0.9228641	0.26743102	0.921	0.921
9	4 convolution [1024, 512, 256, 2]	0.962343752	0.96216434 0.9625213	0.0989532	0.96234375	0.96234375	0.9237	0.92550415 0.92173606	0.26576996	0.92366666	0.92366666
10	4 convolution [1024, 512, 256, 2]	0.962145805	0.9619803 0.9623099	0.0997536	0.96214581	0.96214581	0.9219	0.92404956 0.9196604	0.25235507	0.92191666	0.92191666