Classification of Real and Fake Human Faces Using Deep Learning

Martas the Atom GAUTAM and Jefrin Jo WINSLINE CHANDRA

Group Project Report

COMP 4026

Computer Vision and Pattern Recognition

Faculty of Computer Science

Hong Kong Baptist University, Kowloon, Hong Kong

Abstract

The project aims to develop a method for distinguishing between real human faces and AI-generated synthetic faces. The objective is to address the increasing prevalence of digitally manipulated images and educate the public about the capabilities of machine learning in creating realistic face images. The proposed methods involve the use of different algorithms to differentiate between real and synthetic faces and evaluate its effectiveness using evaluation metrics such as AUROC and F1 score. The project utilizes a dataset from Kaggle for training, validation, and testing, and the results are presented through tables and graphs.

1. Introduction

In today's digital age, advancements in machine learning have enabled the creation of highly realistic synthetic faces through AI algorithms. However, this also raises concerns about the authenticity and trustworthiness of digital images, as they can be easily manipulated and used deceitfully. To address this issue, the project "Which Face Is Real?" aims to develop a method for accurately distinguishing between real human faces and AI-generated synthetic faces.

By developing an accurate method for differentiating between real and synthetic faces, the project aims to raise awareness about the potential misuse of digitally manipulated images and the need to scrutinize and verify the authenticity of images online.

Fake and Real Human Face

With the development of Generative Adversarial Network (GAN) computers can generate vivid face images that can easily deceive human beings. These generated fake faces will inevitably bring serious social risks, e.g., fake news, fake evidence, and pose threats to security.

Thus, powerful techniques to detect these fake faces are highly desirable. However, in contrast to the intensive studies in GANs, our understanding of generated faces is fairly superficial and how to detect fake faces is still an under-explored problem. Moreover, fake faces in practical scenarios are from different unknown sources, i.e. different GANs, and may undergo unknown image distortions such as down sampling, blur, noise, and JPEG compression, which makes this task even more challenging. In this study, we aim to analyse different architectures performances and find the best one amongst them.

1.1. Problem statement

The fake image can be made by using image editor, face effect, or any program to change the facial features.

Effects on the face can change facial features and it’s difficult to know true identity for someone. In this application, we hope to train a model using the provided dataset to recognize such fake faces.

Computerized applications can use deep learning techniques to increase accuracy and efficiency in diagnosis. These include image processing techniques, feature extraction and evaluation.

1.2. Objectives of the Study

Main objective for this study is implementation of a software model to detect and classify face to real or fake for expert-generated high-quality photo shopped face images.

Other objectives are:

- Detect fake face images rapidly.
- Get high accuracy and validation in the testing and training images.
- Implementation of network architectures to find the best model with the best result.
- To assess the performance of the proposed methods.

2. Literature Review:

"Deep fake detection and classification using error-level analysis and deep learning"[1], published in Scientific Reports, explores the use of error-level analysis and deep learning for detecting and classifying deep fake images. The authors emphasize the growing concern of deep fake technology and its potential threats in various domains, including security and fake news dissemination. The study proposes a novel approach that combines error-level analysis and deep learning to detect deep fake

images. Error-level analysis involves analysing the inconsistencies in the compression levels of different regions within an image. The authors use a deep learning model trained on a large dataset consisting of both real and manipulated images.

"Classification of Real and Fake Human Faces Using Deep Learning"[2], focuses on the use of deep learning techniques to detect and classify real and fake human faces. The authors highlight the importance of artificial intelligence, deep learning, and neural networks in solving real-world problems. They explain that deep learning is a subset of machine learning that can learn from unstructured or unlabelled data. We use this paper as a reference for our report preparation. The study presented in this source aims to develop a suitable method for detecting real and fake faces using deep learning techniques. The authors use a dataset of 9,000 images and train the models for 150 epochs. The ResNet50 model is found to be the best model, achieving 100% training accuracy, 99.18% validation accuracy, and 99% testing accuracy. The study concludes that deep learning techniques can effectively detect and classify real and fake human faces.

3. Methodology

Our proposed methodology includes working on the provided dataset, identifying the tools and libraries to be used, pre-processing the images in the dataset, data augmentation and construction of the model architecture, compiling the model, training and validating the model.

3.1. Dataset

The dataset in this project consists of 2041 human faces of real-fake images. The dataset of real-fake human faces was provided by the instructor.

Real-Fake	Training Samples	Validation Samples	Testing Samples	Total
Real	720	61	300	1081
Fake-Easy	158	26	56	240
Fake-Mid	324	54	102	480
Fake-Hard	157	30	53	240
Total	1359	171	511	2041

Table 1: Dataset division for training, validation and testing

3.2. Libraries and tools used

Python language was used alongside its various libraries. Both Pytorch and TensorFlow with Keras were used to train the models. Several additional libraries such as matplotlib, cv2, NumPy, sklearn, pandas, plotly, PIL, seaborn, torchvision etc. were used for better facilitation of the code during pre-processing, training, and evaluation.

Several tools were used, the most important of which was Google Colab to write and run the python codes, a research tool for training machine learning, easy to use and does not require any preparation for use. The most important benefit being free-to-use access to big amount of fast processor, RAM and GPU that can store and read notebooks directly from Google Drive.

3.3. Image format

The dataset contained all the images in .jpg format with resolution of 600 by 600 pixels, resolution density of 96 dpi and 24-bit depth.

3.4. Pre-processing

Although all the images in the dataset were 600 by 600 pixels, the images were still first resized to 128 by 128 pixels. Although our preferred resize resolution was 256 by 256 but the colab system crashed because of lack of memory. Hence, we went with 128 by 128 as we determined it was high enough resolution with optimal resource consumption for efficient training. All images were then converted to tensors then normalized to ImageNet standards.

The original images were stored in google drive then imported into Google Colab environment. While pre-processing, all the pre-processed data was stored as variables in the environment, and no changes were made to original images.

3.5. Error Level Analysis (ELA)

Error Level Analysis (ELA) operates by analysing the compression levels between original and compressed images to reveal inconsistencies that may indicate manipulation or editing. The results are visualized in an output that highlights areas with potential manipulation, but various factors may influence ELA's accuracy, such as different compression algorithms and the quality of the original image. ELA can help identify various image manipulations such as cloning, splicing, retouching etc. ELA itself is a manipulation detection technique which paired with ML models enhances the detection rate even further.[1]

In our model, cv2.convertScaleAbs(image, alpha=80, beta=-30) is used after ELA compression for a clearer view of the image. The function will perform the following steps:

1. Multiply each pixel value of the input image by the scaling factor (alpha value), which in this case is 80. This scaling factor adjusts the contrast of the image.
2. Add the beta value of -30 to the scaled pixel values. This shifting offsets the intensity level of the image.

Figure 1: ELA of a Real Image

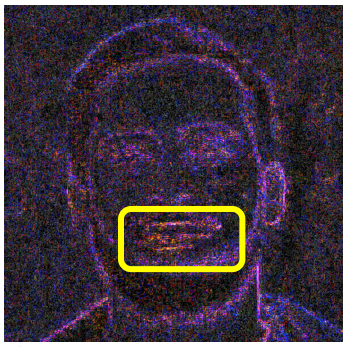


Figure 2: ELA of a Fake Image

3.6. Data Augmentation

Lack of labelled training data is always an important issue for deep learning computer vision applications. In this project too, since we decided to use only the provided dataset, we decided to have data augmentation to improve the performance of the model without using new dataset. Data augmentation also helps the model be more robust and prevent overfitting. We used tensorflow's inbuilt functionality tf.image.[augmentation_type](original_image) to get 6 different augmentations, (flip_left_right, flip_up_down, rot90, random_brightness, random_contrast, random_saturation).

After augmentation, the tensors were normalized, then stored as numpy arrays with correct labels together with original image. Hence, we had 7 images (6 augmented and 1 original) at the end. Data augmentation was done for training and validation set only and not the test set.

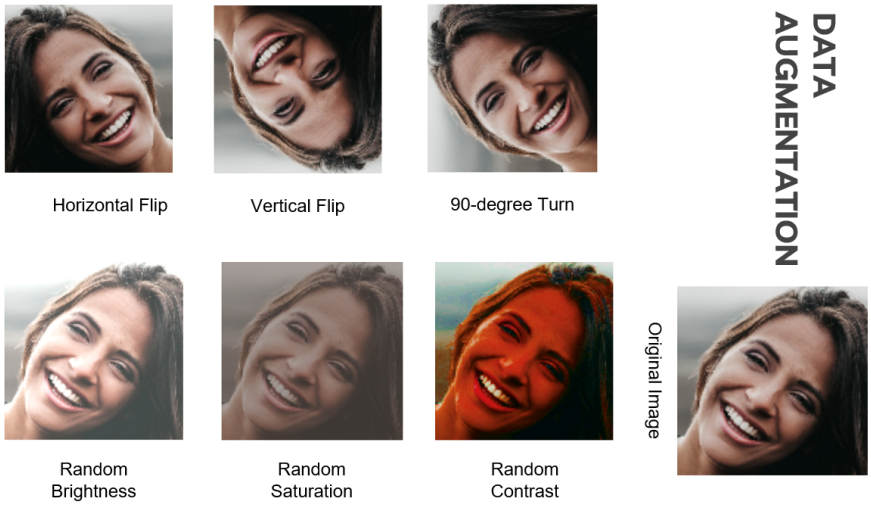


Figure 3: Six Different Data augmentation used in project along with original Image.

Train_images.shape	(9513, 128, 128, 3)
Train_labels.shape	(9513)
Validate_images.shape	(1197, 128, 128, 3)
Validate_labels.shape	(1197)
Test_images.shape	(511, 128, 128, 3)
Test_labels.shape	(511)

Table 2: Size of NumPy arrays of test, validation and train image & labels after pre-processing & data augmentation

3.7. Network Architecture

We have trained our real-fake image dataset using two models we created after getting inspired from a similar research paper and feedback from the professor during the presentation. In addition to the two models we created, we also used three pre-trained models for deep learning: ResNet-50, VGG16 and MobileNet.

3.8. Training and Validating the Models

3.8.1. Proposed Model 1

We took inspiration from general models available on internet and created a similar model from scratch with 5 convolutional layers followed by a fully connected hidden layer (as shown in Figure 7). We also used Dropout layers in between to regularise the network to prevent from overfitting, and all layers have ReLU activation function except the output layer. Output layer uses a sigmoid activation as it outputs the probability of image being real; Binary_CrossEntropy as loss function, to optimize the network 'Adam' optimizer and learning rate of 0.0001 were used. The batch size was 32, and the epoch used was 30. After every epoch, the updates will be made to the weights of the network automatically. At the end of 30 epoch, the training accuracy was 0.6744 and validation accuracy was 0.6057.

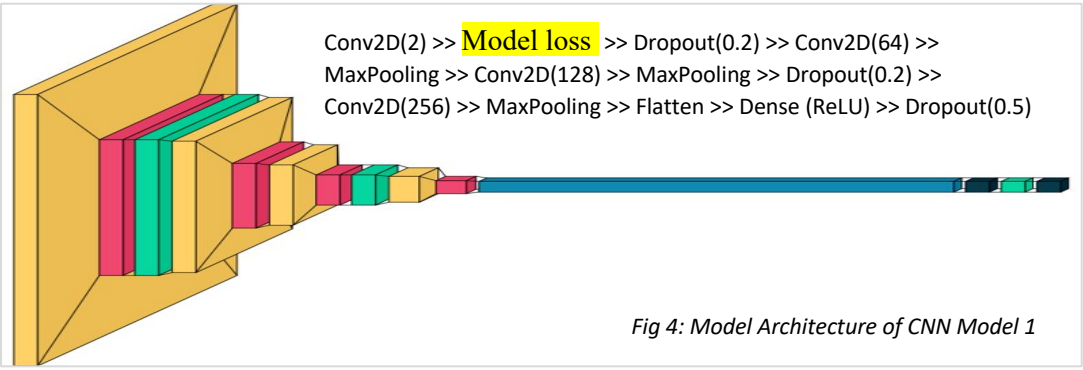


Fig 4: Model Architecture of CNN Model 1

The figure below shows the model loss and model accuracy curves for both training and validation data across the 30 epoch with ELA is as follows.

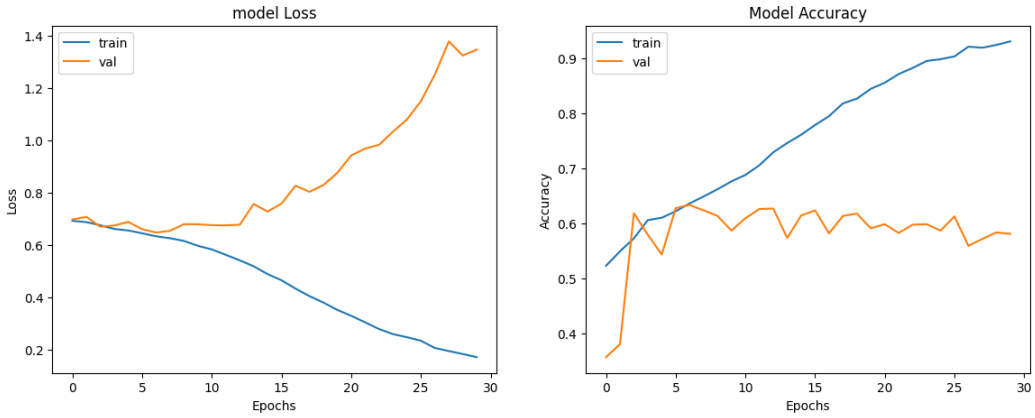


Figure 5: Model Loss and Accuracy for Custom CNN without ELA

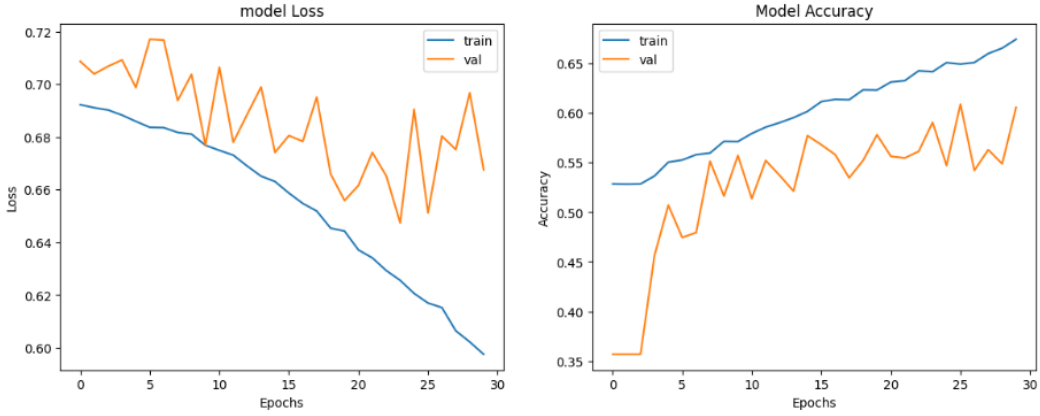


Figure 6: Model Loss and Accuracy for Custom CNN with ELA

3.8.2. VGG16 Model

We used a pre-trained model called VGG16 network with 16 convolutional layers followed by a fully connected hidden layer, also used a dropout layer in between to regularise the model.

For all pre-trained models, we trained the models for only 15 epochs while keeping the batch size, learning rates and optimisers the same.

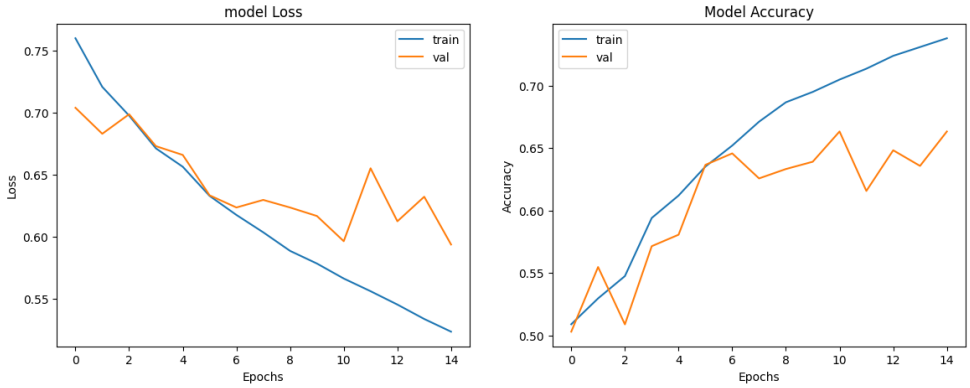


Figure 7: Model Loss and Accuracy for VGG16 without ELA

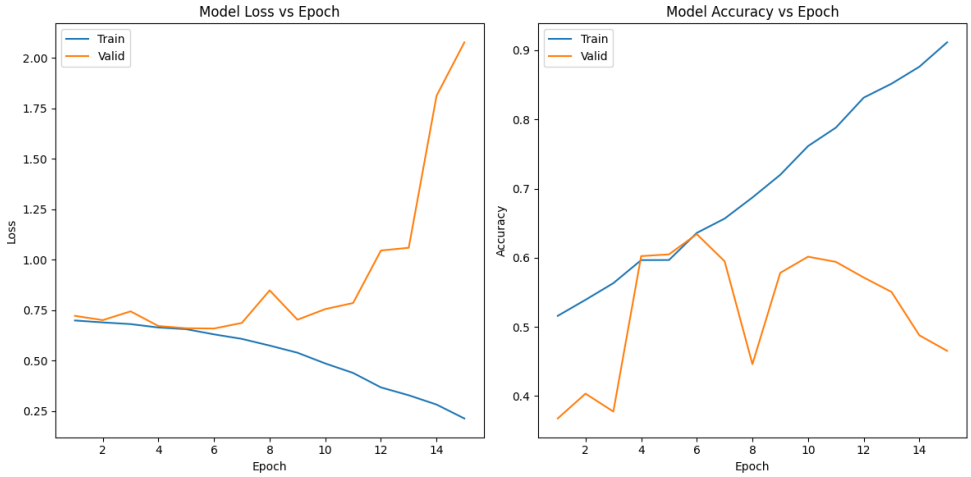


Figure 8: Model Loss and Accuracy for VGG16 with ELA

3.8.3. ResNet50 model

We also tried the pre-trained ResNet50. We can see that ResNet50 was able to attain an appreciable training accuracy of 99.56% in 15 epochs. *Cross Entropy Loss* function was used. The optimization used was *Stochastic Gradient Descent* with a *learning rate* of 0.001 and a *momentum* of 0.9.

A dense layer with ReLU is added over the ResNet50 model with a dropout value of 0.5 before the final classification layer.

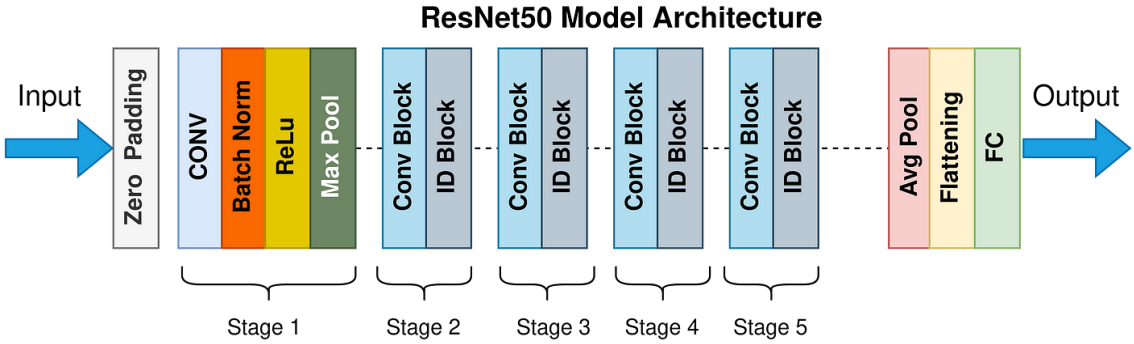


Figure 9: ResNet50 Architecture

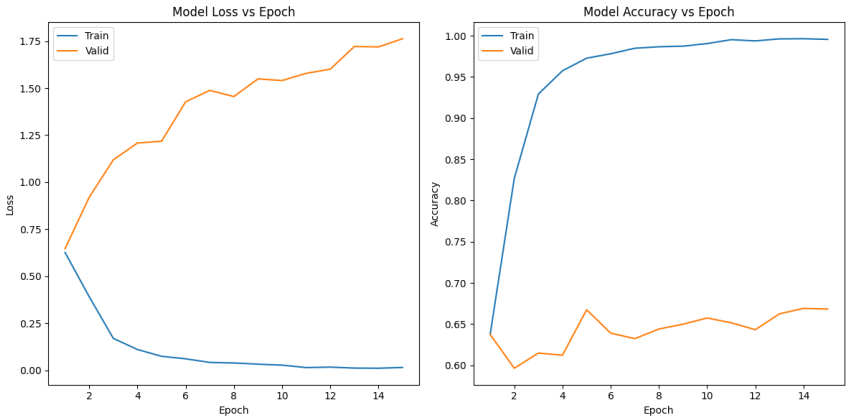


Figure 10: Model Loss and Accuracy for ResNet50 without ELA

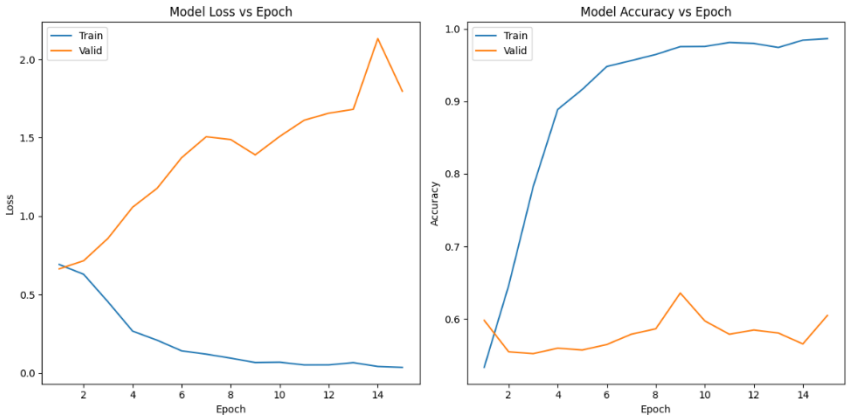


Figure 11: Model Loss and Accuracy for ResNet50 with ELA

3.8.4. MobileNet model

We also tried the pre-trained MobileNet. We can see that MobileNet did well on test data even with a lesser validation accuracy (remember, for all models, validation was augmented but test was **not**). We believe MobileNet was able to ignore the unnecessary parameters and train well. Just like with ResNet50, *Cross Entropy Loss* function was used. The optimization used was *Stochastic Gradient Descent* with a *learning rate* of 0.001 and a *momentum* of 0.9.

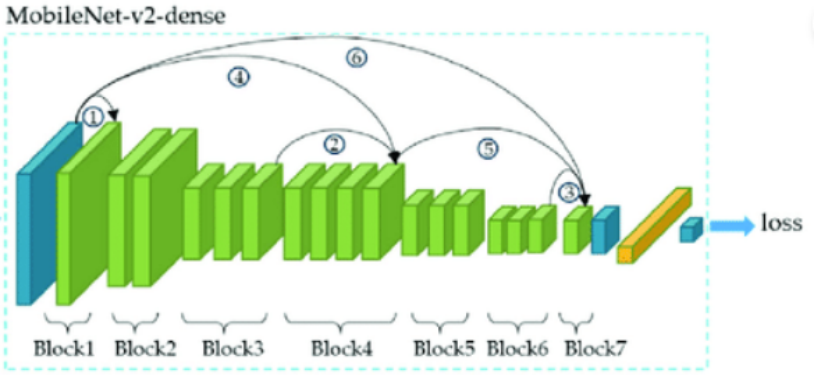


Figure 12: MobileNet Architecture

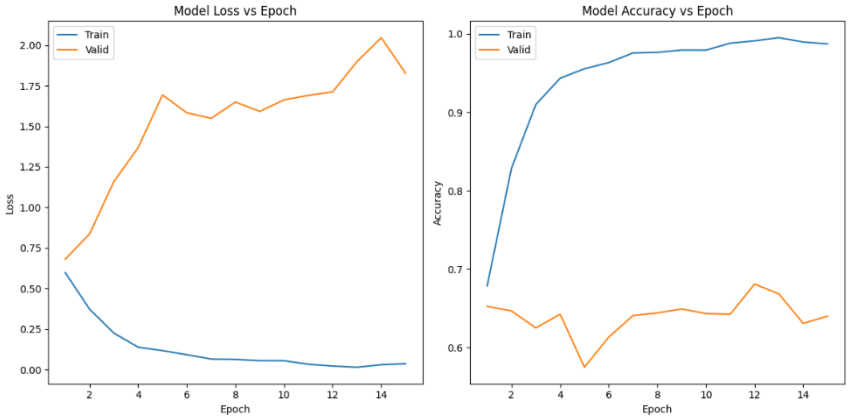


Figure 13: Model Loss and Accuracy for MobileNet without ELA

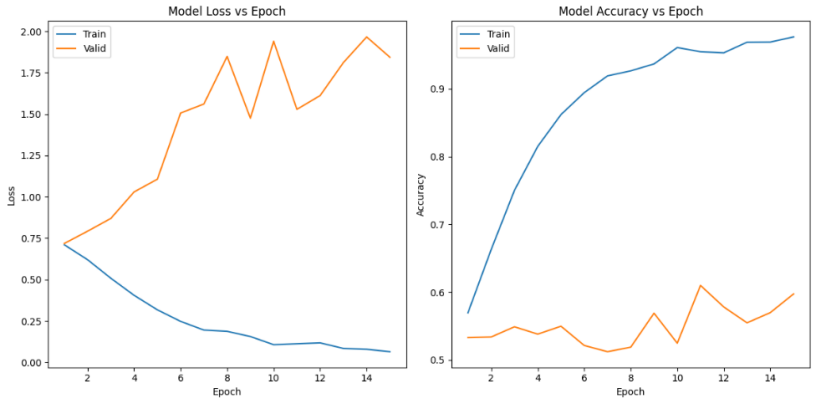


Figure 14: Model Loss and Accuracy for MobileNet with ELA

3.8.5. Fusion model

As we can see from *Figure 2*, ELA gives some visual differences for fake images. But it doesn't work so well when used in deep learning models. So, we have tried concatenating the features extracted from both original and ELA-processed images. We tried three types in this, as follows:

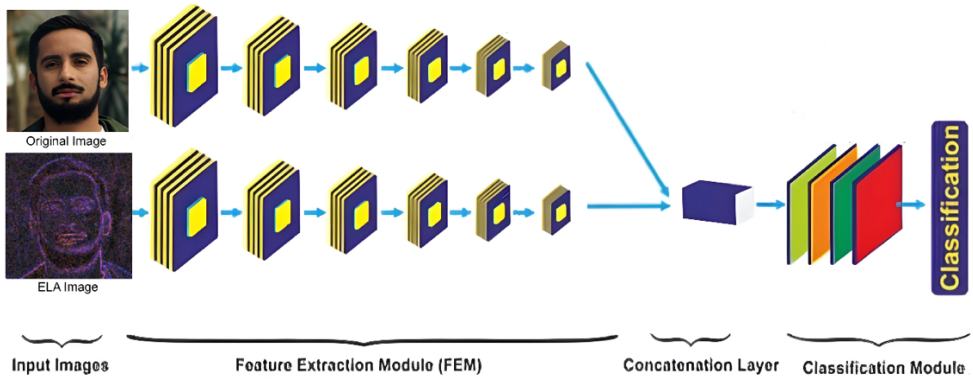


Figure 15: Architecture of Concatenation of Models

3.8.5.1. Type 1:

For Type 1, we used MobileNet for extracting features from the original images and ResNet50 for extracting features from the ELA-processed images and concatenated them. *Cross Entropy Loss* function was used. The optimization used was *Stochastic Gradient Descent* with a *learning rate* of 0.001 and a *momentum* of 0.9.

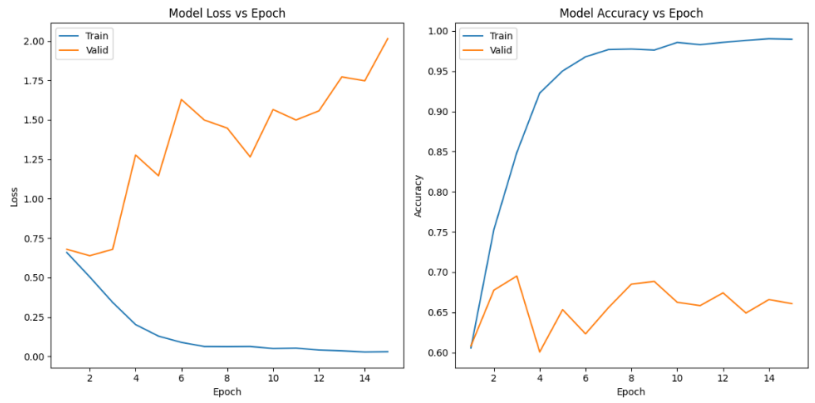


Figure 16: Model Loss and Accuracy for Type 1 of Fusion Model

3.8.5.2. Type 2:

For Type 2, we used our own CNN for extracting features from both the original images and the ELA-processed images and concatenated them. *Cross Entropy Loss* function was used. The optimization used was *Adam* with a *learning rate* of 0.0001. Number of epochs = 30.

The model reached only reached a training accuracy of 68.16% in 30 epochs. We tried an increased learning rate of 0.001, but the model *did not move* towards *the* right solution even after 15 epochs.

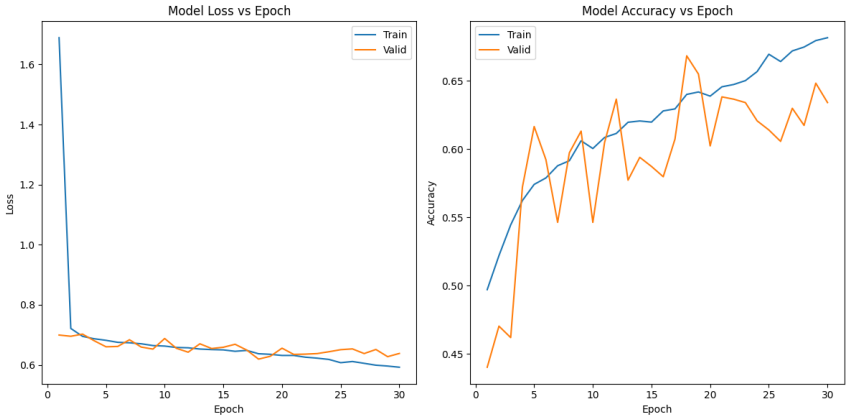


Figure 17: Model Loss and Accuracy for Type 2 of Fusion Model

3.8.5.3. Type 3:

For Type 3, we used MobileNet for extracting features from the original images and ResNet50 for extracting features from the ELA-processed images and concatenated them. *Cross Entropy Loss* function was used. *Cross Entropy Loss* function was used. The optimization used was *Adam* with a *learning rate* of 0.0001. Number of epochs = 15.

The model reached a training accuracy of 99.12% and a testing accuracy of 70.45% in 15 epochs We tried increasing it to 30 epochs, but it didn't change much, and the training and testing accuracy reduced to 99.09% and 68.69% respectively.

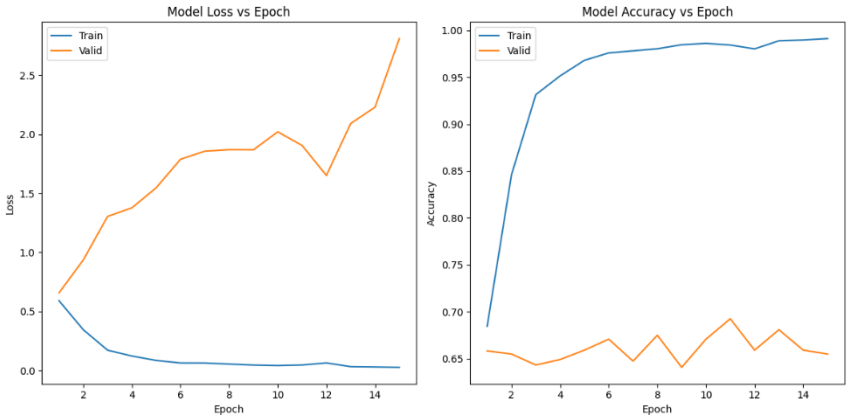


Figure 18: Model Loss and Accuracy for Type 3 of Fusion Model

4. Evaluation of the model

4.1. Testing the model

Testing dataset consisted of 511 images of Real-Fake images in .jpg format, different from the training or validation data. There were mix of 300 real images, 56 easy fake images, 102 mid fake images and 53 hard fake images.

Testing the models was done through loading the test_images and predicting their affinity to be real image using the model.predict(test_images).

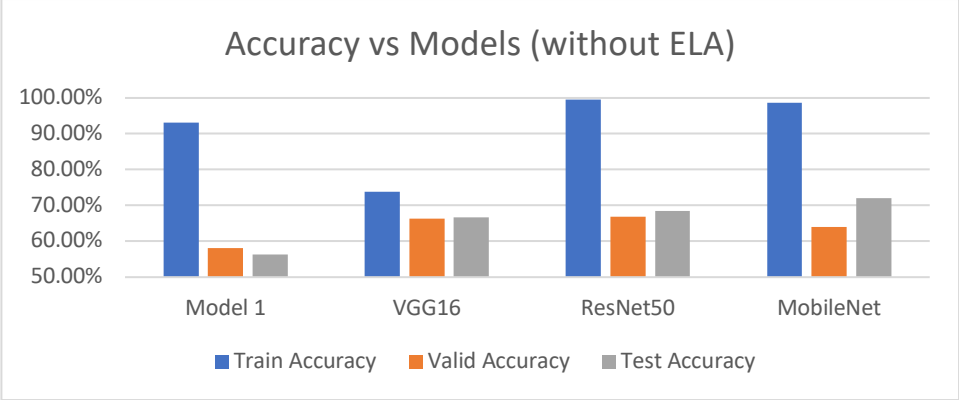
We also used other metrics for evaluation of our models such as precision, recall, F1-Score, and AU-ROC.

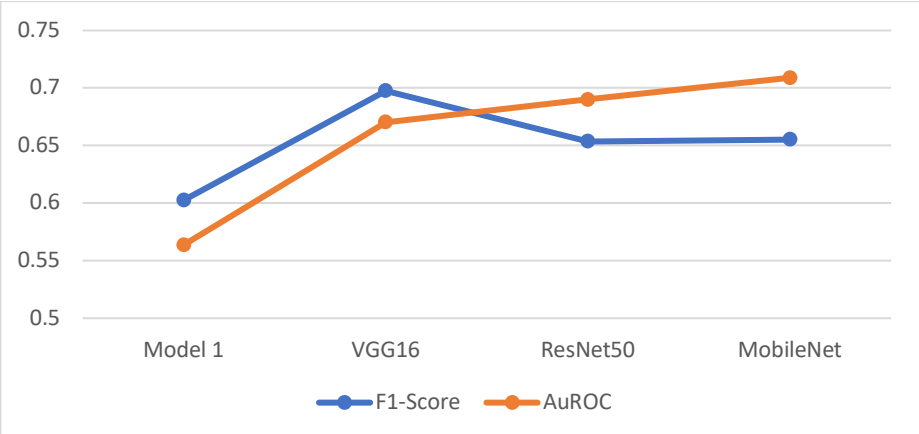
4.2. Result

We trained our custom models on the training dataset using 1359 images for 30 epoch while for pre-trained models we trained them for 15 epoch. The results were as follows:

Without ELA

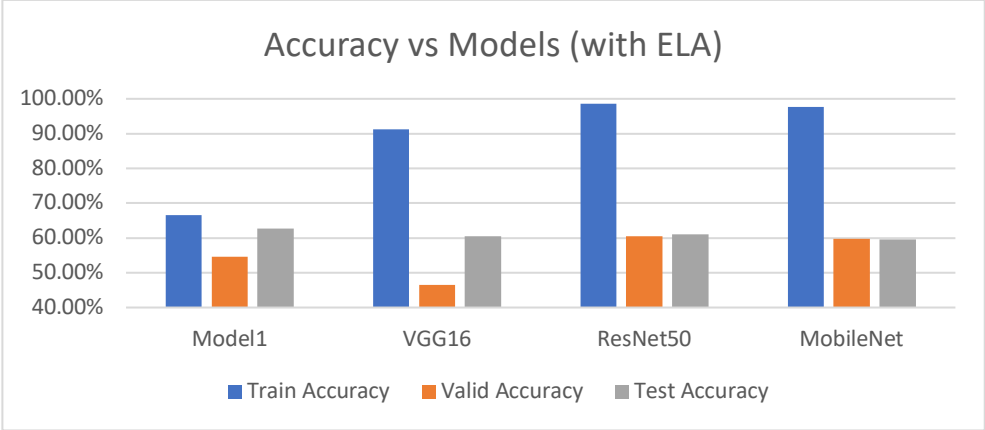
Criterion	Model 1	VGG16	ResNet50	MobileNet
Training Accuracy	93.09%	73.80%	99.56%	98.73%
Validation Accuracy	58.15%	66.33%	66.83%	63.99%
Testing Accuracy	56.36%	66.73%	68.49%	72.02%
Precision	0.64751	0.74809	0.59840	0.66670
Recall	0.56333	0.65333	0.72037	0.64454
F1-Score	0.60250	0.69751	0.65376	0.65542
AU-ROC	0.56365	0.67027	0.69019	0.70894

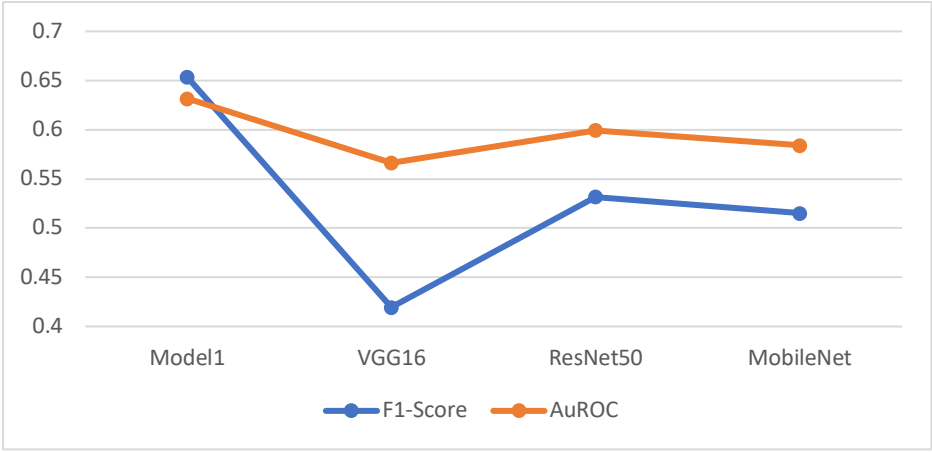




With ELA

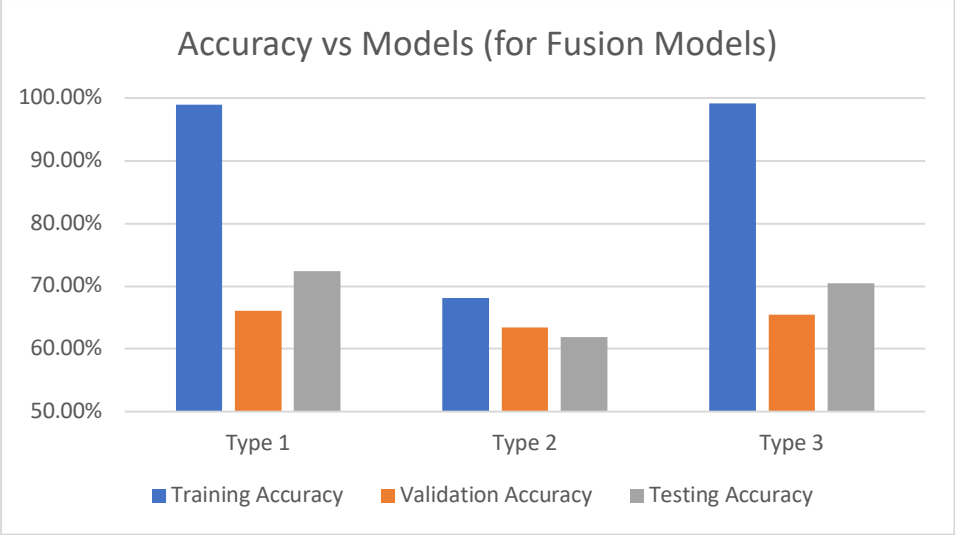
Criterion	Model_1	VGG16	ResNet50	MobileNet
Training Accuracy	66.55%	91.14%	98.65%	97.66%
Validation Accuracy	54.59%	46.53%	60.48%	59.73%
Testing Accuracy	62.62%	60.47%	61.06%	59.49%
Precision	0.71713	0.53280	0.52800	0.50930
Recall	0.60000	0.34580	0.53554	0.52132
F1-Score	0.65336	0.41954	0.53176	0.51522
AU-ROC	0.63175	0.56632	0.59944	0.58400

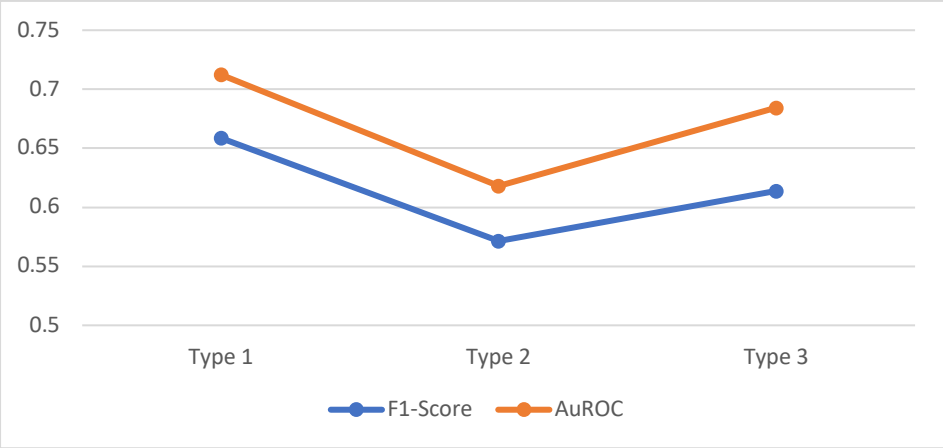




Fusion Models

Criterion	Type 1		Type 2		Type 3	
	ResNet50	MobileNet	Custom CNN	Custom CNN	Custom CNN	MobileNet
Training Accuracy	98.97%		68.16%		99.12%	
Validation Accuracy	66.08%		63.41%		65.50%	
Testing Accuracy	72.41%		61.84%		70.45%	
Precision	0.67330		0.5328		0.66670	
Recall	0.64454		0.61611		0.56872	
F1-Score	0.65859		0.57142		0.61381	
AU-ROC	0.71227		0.61806		0.68436	





Based on the table of results, without ELA, MobileNet was the best model, then ResNet50, followed by VGG16 and our Model 1 at last, which was created and trained from scratch. With ELA, our Model_1 had the best accuracy with MobileNet and ResNet50 closing against each other. Amongst the fusion-based models, the model with ResNet50 and MobileNet had best accuracy, followed by the fusion of MobileNet and Model1 and at the last with both ends Model_1. However, fusion models showcased significant improvements in accuracy as they took advantage of both ELA as well as feature of original images. This study was done to find the best deep learning approach for detecting if a face image is real or fake.

5. Discussion & Conclusion

The advancement of image capability and image generation techniques have now provided the ability to create security-less and convincing fake face images. The challenging nature of data, whether in terms of visual perception or algorithm discovery, is present in recent works. Although we were inspired by previous works from others and wanted to replicate a similar result, we were not able to achieve good replication. We trained the provided limited dataset using various epoch, learning rates, optimizers, pre-processing steps, and such, whichever seemed the best for the moment, and got the MobileNet model as the best model of network architectures without ELA, our Model_1 as the best with ELA, and fusion models have good accuracy rates too.

Looking at the pre-trained models that did well, MobileNet and ResNet50 have different architectures, which can affect their performance on specific tasks. MobileNet is designed to be lightweight and efficient, making it suitable for resource-constrained environments. On the other hand,

ResNet50 has a deeper architecture with more parameters, which can potentially lead to overfitting if the dataset is not large enough. The architecture of MobileNet might be better suited for the characteristics of the real and fake image dataset, hence resulting in higher accuracy.

Looking at the Fusion models, Type 1 and Type 3 did pretty well. But, it still is close to the MobileNet model, not any better. We might need to change the number of nodes in the neural layers after feature extraction of the Fusion Models and consider various combinations for a better accuracy.

References

[1] Rafique, R., Gantassi, R., Amin, R., Frnda, J., Mustapha, A., & Alshehri, A. H. (2023). Deep fake detection and classification using error-level analysis and deep learning. *Scientific Reports*, *13*(1). https://doi.org/10.1038/s41598-023-34629-3

[2] Salman, F. M. (2022). *Classification of real and fake human faces using deep learning*. Retrieved November 20, 2023, from https://philarchive.org/rec/SALCOR-3

[3] *Fake Image Detector | Fake Image Detector online | FotoForensics | Error Level Analysis*. (n.d.). https://www.fakeimagedetector.com/blog/shedding-light-ela-comprehensive-guide-error-level-analysis/

[4] Yadav, D. P., Kishore, K., Gaur, A., Kumar, A., Singh, K. U., Singh, T., & Swarup, C. (2022). A novel Multi-Scale feature Fusion-Based 3SCNet for building crack detection. Sustainability, 14(23), 16179. https://doi.org/10.3390/su142316179

[5] Hands-on Transfer Learning with Keras and the VGG16 Model. (n.d.). https://www.learndatasci.com/tutorials/hands-on-transfer-learning-keras/

[6] MobileNet-V2: Summary and Implementation - HackMD. (n.d.). HackMD. https://hackmd.io/@machine-learning/ryaDuxe5L

[7] OpenAI. (2023). ChatGPT 3.5 (November 6 version) [Large language model]. https://chat.openai.com/chat

[8] Danielsen, N. (2021, December 12). Simple Image Classification with ResNet-50 - Nina Danielsen - Medium. Medium. https://medium.com/@nina95dan/simple-image-classification-with-resnet-50-334366e7311a

Work Distribution

Task	Person Responsible
ELA function	Jefrin
Data Augmentation and Pre-processing	Martas
Model_1	Martas
VGG16	Martas
ResNet50	Jefrin
MobileNet	Jefrin
Fusion Model Type1	Jefrin
Fusion Model Type 2	Martas
Fusion Model Type 3	Both

In addition to the tasks, both of us tried to understand each other's code and ran the models using PyTorch as well as TensorFlow with Keras. Jefrin was responsible for implementation using PyTorch, while Martas was responsible for implementation via Tensorflow with Keras. So we both had understanding of each other's tasks as well as model working.