Project Information

In this project, we will base our implementation mainly off of the paper *An Improved Dense CNN Architecture for Deepfake Image Detection* by Yogesh Patel et al. (2023) [1]. Since the source code to the implementation in this paper wasn't provided, we will replicate the architecture from scratch and modify that instead.

Problem Statement

With the recent rise and progression of AI technology, society finds itself face to face with a growing problem: Deepfake (DF) content. DFs allow for the creation of highly realistic but fake images, videos, or even audio. One of the most common form of DF is generated using Generative Adversarial Networks (GANs). The GAN model consists of two networks. A generator and a discriminator, with the goal of generating a DF real enough to spoof the discriminator. Hence why DF images are getting harder to differentiate from real images. These highly realistic images can be used to spread misinformation, posing as a threat to personal reputations or even public opinion.

Our project addresses this issue by focusing on improving the Dense CNN (D-CNN) architecture proposed by Yogesh Patel et al. in the paper *An Improved Dense CNN Architecture for Deepfake Image Detection* for detecting DF images generated using GANs. The D-CNN model makes use of dense connections that allows it to capture subtle and complex features which may indicate manipulated content. The approach outlined in the paper is able to achieve an accuracy of 97.2%, but since the network is comprised mainly of traditional convolution layers, it creates a large number of parameters which causes longer training time. We aim to enhance the speed and training efficiency of the model while maintaining accuracy, making it suitable in real world scenarios where computational resources might be limited.

To do this, we decided to integrate an efficient CNN implementation developed by Google known as MobileNet into the proposed D-CNN architecture. By combining the lightweight and efficient structure of MobileNet with a dense network such as D-CNN, we hope to not only address the growing challenge of DF detection, but also contribute to the development of real time solutions that may combat DF and help people regain trust in digital media.

Proposed Solution and Implementation Details

Description of algorithm/method

We decided to reimplement the D-CNN architecture with MobileNet due to two reasons. Firstly, MobileNets reduces parameters. MobileNets are comprised of a depth-wise separable convolution layer that separates tradition convolution into two operations: a depthwise convolution that applies a single filter to each input channel, and a pointwise convolution that uses a 1x1 convolution filter to combine the output of depthwise convolutions. This separation drastically reduces the number of operations required, depending on the the size of the original CNN architecture. Since a D-CNN is just a CNN with a larger number of layers, this makes MobileNet a very suitable solution.

Standard: $(kernel\ size\text{^}2\ *\ input\ channels\ )\ *\ output\ channels$ (1)

Depthwise: $(kernel\ size^2)\ *\ input\ channels$

(2)

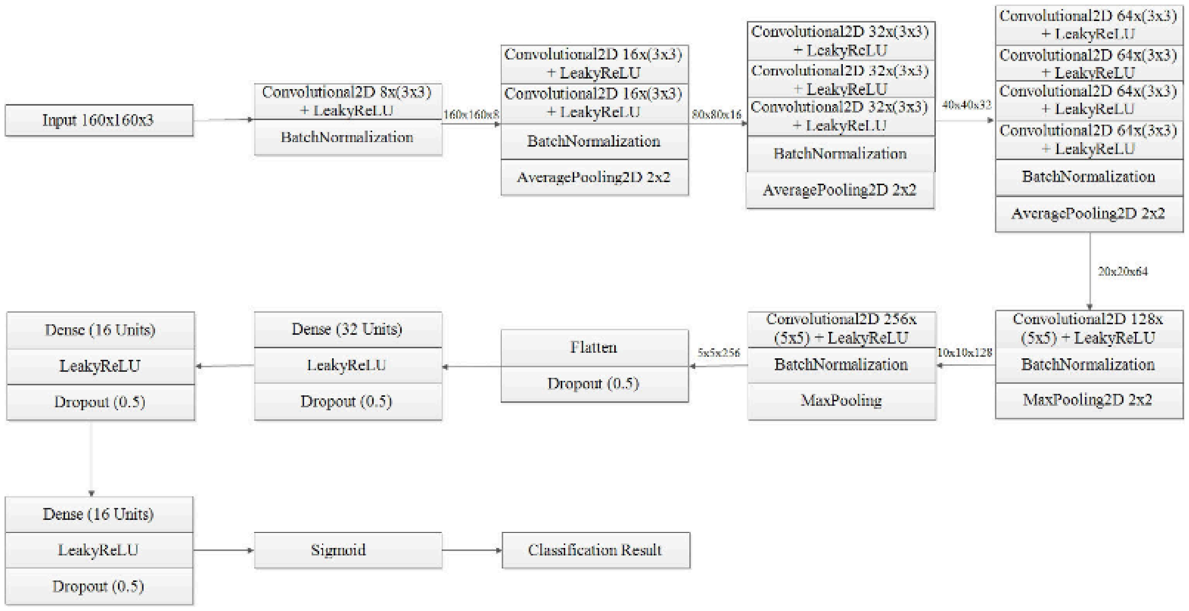
Pointwise: $input\ channels\ *\ output\ channels$ (3)

Total: $Input\ channels\ *\ (kernel\ size^2\ +\ output\ channels)$ (4)

Reduction: $(\frac{1}{output\ channels})\ +\ (\frac{1}{kernel\ size^2})$ (5)

Based on the parameter equations (5) we see that the reduction is roughly proportionate to the kernel size (1/ output channels is negligible). Therefore the average reduction assuming a 3x3 convolution kernel would be around 9.

Second, MobileNet has been observed to produce comparable accuracy with its CNN counterpart, but with significantly lower complexity. Since the original model is able to produce a high accuracy of 97.2%, we thought that a small trade off of an already impressive accuracy in favor of faster training time might give the new model more leeway for fine tuning and other optimization that may take longer to adjust due to the longer training duration of the original model.



Input 160x160x3
Convolutional2D 8x(3x3) + LeakyReLU
BatchNormalization
160x160x8
Convolutional2D 16x(3x3) + LeakyReLU
Convolutional2D 16x(3x3) + LeakyReLU
BatchNormalization
AveragePooling2D 2x2
80x80x16
Convolutional2D 32x(3x3) + LeakyReLU
Convolutional2D 32x(3x3) + LeakyReLU
Convolutional2D 32x(3x3) + LeakyReLU
BatchNormalization
AveragePooling2D 2x2
40x40x32
Convolutional2D 64x(3x3) + LeakyReLU
Convolutional2D 64x(3x3) + LeakyReLU
Convolutional2D 64x(3x3) + LeakyReLU
Convolutional2D 64x(3x3) + LeakyReLU
BatchNormalization
AveragePooling2D 2x2
20x20x64
Convolutional2D 128x (5x5) + LeakyReLU
BatchNormalization
MaxPooling2D 2x2
10x10x128
Convolutional2D 256x (5x5) + LeakyReLU
BatchNormalization
MaxPooling
5x5x256
Flatten
Dropout (0.5)
Dense (32 Units)
LeakyReLU
Dropout (0.5)
Dense (16 Units)
LeakyReLU
Dropout (0.5)
Dense (16 Units)
LeakyReLU
Dropout (0.5)
Sigmoid
Classification Result

Figure 1. Flow diagram of D-CNN implementation, Yogesh Patel et al.

Since the author didn't provide the original source code, we will do our best to implement it from scratch using Pytorch with the architecture provided. In the original paper, the model (Figure 1) consists of a series of 2D convolution layers, followed by Batch Normalization and 2D Average Pooling. Finally, it gets flattened, and dropout is applied to the flattened layer as well as to each subsequent fully connected layer, before finally outputting one value that is fed into a Sigmoid function, producing a classification result. The model starts by taking in an image of dimension 160 x 160 x 3. Inspired by the MobileNet implementation outlined by Andrew G. Howard et al. (2017), we start by leaving the first convolution layer as is. According to their paper [2], this singular standard convolution layer is meant to capture lower level features from the input image and use this new representation for the rest of the separable convolution. Therefore, our first block consists of a 2D convolution layer of size (3x3) and outputs 8 channels, followed by LeakyReLU and then BatchNorm. This produces an output shape of 160 x 160 x 8. From here on, we replace all instances of 2D convolution layer with a custom MobileBlock, which is a depthwise convolution, followed by a batchnorm, followed by a pointwise convolution, followed by another batchnorm, before LeakyReLU is applied. Therefore, the next block which was originally a 2D convolution of 16 x (3 x 3), followed by LeakyReLU, followed by another 2D convolution of 16 x (3 x 3), followed by another LeakyReLU, and then BatchNorm and AveragePooling2D (2x2), is now comprised of a MobileBlock with an input of 8 channels and output of 16 channels, followed by another MobileBlock with an input of 16 channels and output of 16 channels, and then AveragePooling2D (2x2) (note that LeakyReLU and BatchNorm has been applied inside the MobileBlock). This produces an output of dimension 80 x 80 x 16. The next block consists of MobileBlock (32), MobileBlock (32), MobileBlock (32), and then another average pooling. This outputs a dimension of 40 x 40 x 32. The following block uses four MobileBlock (64) and average pooling, outputting a dimension of 20 x 20 x 64. At this point, the model should have extracted deep image features. Therefore, the block after consists of one MobileBlock that uses a pointwise layer with kernel size 5 instead of the previous 3, and a MaxPooling2D (2x2). This produces an output of 5 x 5 x 256. This output is flattened, and dropout (0.2) is applied before it goes into a series of fully connected layers. Note that in the original paper, a dropout of 0.5 was used for all instances where dropout was applied. But during the actual training, we found that a dropout rate of p = 0.5 causes vanishing gradients, and the model isn't converging. Therefore we lowered all dropout to 0.2 instead in order for the model to get past 50% accuracy. The first fully connected (FC) layer outputs 32 features. LeakyReLU and dropout is applied. The next FC layer outputs 16 features, and LeakyReLU and dropout is applied once again. This is followed by a FC layer of the same structure, and then finally, it outputs one feature that goes into a Sigmoid function, outputting the final classification result. The dimensions are displayed below:

Layer	Layer Type	Output Dimension
1	Input (Conv 2D)	(160 x 160 x 8)
2	MobileBlock (Depthwise + BatchNorm + LeakyReLU + Pointwise + BatchNorm + LeakyReLU)	(160 x 160 x 16)
3	MobileBlock (Depthwise + BatchNorm + LeakyReLU + Pointwise + BatchNorm + LeakyReLU)	(160 x 160 x 16)
4	Average Pooling 2D	(80 x 80 x 16)
5	MobileBlock (Depthwise + BatchNorm + LeakyReLU + Pointwise + BatchNorm + LeakyReLU)	(80 x 80 x 32)
6	MobileBlock (Depthwise + BatchNorm + LeakyReLU + Pointwise + BatchNorm + LeakyReLU))	(80 x 80 x 32)
7	MobileBlock (Depthwise + BatchNorm + LeakyReLU + Pointwise + BatchNorm + LeakyReLU)	(80 x 80 x 32)
8	Average Pooling 2D	(40 x 40 x 32)
9	MobileBlock (Depthwise + BatchNorm + LeakyReLU + Pointwise + BatchNorm + LeakyReLU)	(40 x 40 x 64)
10	MobileBlock (Depthwise + BatchNorm + LeakyReLU + Pointwise + BatchNorm + LeakyReLU)	(40 x 40 x 64)
11	MobileBlock (Depthwise + BatchNorm + LeakyReLU + Pointwise + BatchNorm + LeakyReLU)	(40 x 40 x 64)
12	MobileBlock (Depthwise + BatchNorm + LeakyReLU + Pointwise + BatchNorm + LeakyReLU)	(40 x 40 x 64)
13	Average Pooling 2D	(20 x 20 x 64)
14	MobileBlock (Depthwise + BatchNorm + LeakyReLU + Pointwise + BatchNorm + LeakyReLU, kernel size=5)	(20 x 20 x 128)
15	Max Pooling 2D	(10 x 10 x 128)
16	MobileBlock (Depthwise + BatchNorm + LeakyReLU + Pointwise + BatchNorm + LeakyReLU, kernel size=5)	(10 x 10 x 256)
17	Max Pooling 2D	(5 x 5 x 256)
18	Flatten	(6400)
19	Dropout	(6400)
20	Fully Connected (FC1)	(32)
21	Leaky ReLU Activation	(32)
22	Dropout	(32)

23	Fully Connected (FC2)	(16)
24	Leaky ReLU Activation	(16)
25	Dropout	(16)
26	Fully Connected (FC3)	(1)
27	Sigmoid Activation	(1)

Table 1. Output dimensions for MobileNet D-CNN Implementation

Instructions for Using Program

Start by pip installing the requirements.txt file under the data folder. The dataset can be found in the provided Google Drives Zip file under data/data.txt. Download and unzip the dataset in the src folder. The folder structure of the dataset is 3 subfolders corresponding to training set, validation set, and testing set. Within each set, images are under a deepfake subfolder and a real subfolder, acting as label separators. There are 6000 images in training set, consisting of 3000 real images and 3000 DF images. There are 1000 images in validation set, consisting of 500 real images and 500 DF images. Finally, there are 3000 images in test set, with 1500 of them being real images and 1500 of them being DF images. Nothing has to be done to the datasets, except placing them in the same directory as the Python notebook. To train the model from scratch, running the notebook from start to finish should be sufficient. But a pretrained model will also be provided to skip this step.

To run the pretrained model at the bottom of the notebook in the cell titled *Test the Model!*, simply download and save the ***best_mobile_deepfake_model.pth*** model provided in data/model.txt to the src subfolder, in the same directory as the notebook. change the ***test_image_path*** variable to point to the filepath of the image you want to classify. Running that cell will provide a classification result.

Data

We will use a combination of real and DF image datasets for training and testing. For real images, we will use the CelebA and FFHQ Datasets as used by the paper, while DF images will be sourced from various GAN architectures, such as StyleGAN, StarGAN, StarGAN2, AttGAN, and GDWCT. Staying as close to the original implementation as possible, 1000 images each from the DF datasets are chosen, totaling 5000 DF images. 2500 each from the real dataset are chosen to evenly match the 5000 DF images, giving us 10000 images to work with. In the original paper, 60% of the images are used for training, 10% for validation, and the rest 30% are used for testing. However, the paper also specified the exact amount sampled from each data sources, where out of 3500 images for training, every 10th is reserved for validation. This process is more complicated than necessary for our purposes, so we did the best to randomly sample 60, 10, and 30% of both DF and real data without any overlaps, giving us a trainsize of 6000, validation size of 1000, and test size of 3000. The labels are 0 for DF and 1 for real images. Following the paper, we also applied data augmentation to our training dataset using the Pytorch Transforms method. It includes random horizontal and vertical flipping, random rotation of up to 360 degrees, shear range by 0.2, width and height shift range by 0.2, and a rescale of 0.8 to 1.0 to mimic random zooming of up to 0.2. The images are set to 160 x 160. According to the author, detecting DF in lower resolution images has been a historically difficult task since there are less features to work with. Plus, DF images are commonly downscaled to lower transmission and storage costs [1]. Therefore 160 x 160 x 3 is selected to be more useful in real world scenarios. But a big reason could also be that the minimum size in the given dataset is only 178 x 218 in size.

Results and Discussion

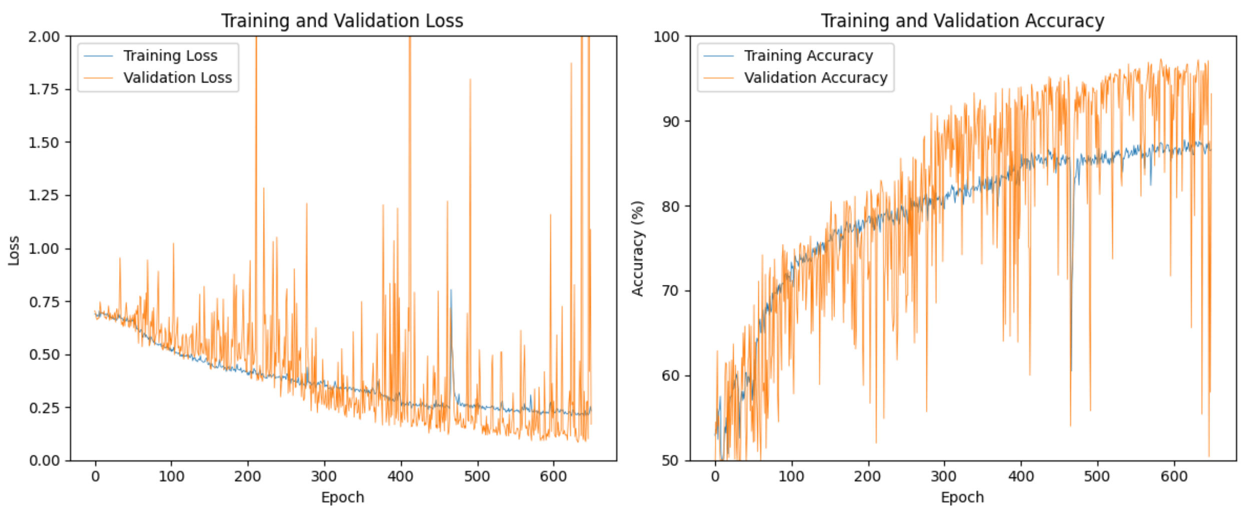


Figure 2. Training, and Validation Loss and Accuracy.

From figure 2, we see that the model had a slow start at around 50 epochs, before steadily increasing in accuracy and decreasing in loss. 550 epochs was the number of iteration chosen in the original D-CNN implementation [1], but during our training, we found that the model was still improving around the 550th epoch, thus we added another 100 epochs to let the model converge more.

In addition to accuracy, we evaluated the model's performance using precision, recall, and F1 score, which are metrics for binary classification tasks like deepfake detection. These metrics help to assess the model’s reliability beyond simple accuracy.

- Precision represents the ratio of true positive predictions (correctly identified real images) to all positive predictions (both true positives and false positives). It answers the question: *How many of the images predicted as real were actually real?*
- Recall indicates the model's ability to identify all real images, calculated as the ratio of true positives to the sum of true positives and false negatives. It answers the question: *How many of the real images did the model correctly identify?*

- F1 Score provides a harmonic mean of precision and recall, offering a single metric that balances both. It is especially useful when there is an uneven distribution between classes or when both false positives and false negatives need to be minimized.

Our results are as follows:

- Precision: 96.43%
- Recall: 97.27%
- F1 Score: 96.85%
- Test Accuracy: 96.83%

These metrics illustrate that the model performs well in distinguishing between real and deepfake images. The model achieved a high recall, which is desirable in applications where failing to detect real images could have serious implications.

Limitations

There are a couple limitations with our model. One limitation is generalizability. Although the model performs well on the test dataset, its performance on unseen GAN architectures hasn't been fully evaluated. With that in mind, DF technologies are rapidly evolving, and new methods may generate more realistic images that could challenge the model. Another limitation is data diversity. The datasets used for training are limited to specific GAN architectures. DF generated through other methods may produce vastly different results. However, this adaptation of the original model is a good stepping stone to creating DF detectors that may detect more advanced forms of AI. Finally, although we aimed to reduce training time from the original model, the training time for the original model wasn't disclosed, therefore we cannot know for sure that the MobileNet implementation does indeed provide a faster training time. We can only base our assumptions off calculating the number of parameters and computation cost.

Comparison to Baseline Model

When compared to the original implementation, this adaptation allows us to achieve greater parameter efficiency while maintaining a comparable level of accuracy. Below is a summary of the differences:

Model Feature	Original Dense CNN	Improved MobileNet-Inspired CNN
Convolutions	Standard	Depthwise Separable
Parameter Count	Higher	Lower
Test Accuracy	97.2%	96.83%
Precision	98%	96.43%
Recall	97%	97.27%
f1-score	97%	96.85%

Table 2. Comparison Between D-CNN and MobileNet Inspired D-CNN

The use of depthwise separable convolutions significantly reduced the computational cost by lowering the number of parameters, making our model suitable for deployment on devices with limited resources. The actual training time wasn't provided in the original paper, but hypothetically, lower computational cost should directly correlate with lower execution time. All the result metrics from MobileNet D-CNN are very close to the original D-CNN implementation. In fact, it appears that recall performed just as, if not better than the original.

Conclusion

In this project, we developed an improved CNN model inspired by MobileNet to detect deepfake images. Through a combination of depth wise separable convolutions, data augmentation, and comprehensive evaluation metrics, our model achieved competitive performance with an accuracy of 96.83%, precision of 96.43%, recall of 97.27%, and F1 score of 96.85%.

This work addresses the growing need for reliable deepfake detection tools, offering a lightweight and effective model that can be implemented in various practical applications. While our model shows promise, future research focusing on generalizability and the inclusion of diverse deepfake datasets could further improve its robustness.

References

[1] Patel, Y., Tanwar, S., Bhattacharya, P., Gupta, R., Alsuwian, T., Davidson, I. E., & Mazibuko, T. F. (2023). An improved dense CNN architecture for Deepfake Image detection. *IEEE Access*, *11*, 22081–22095. https://doi.org/10.1109/access.2023.3251417

[2] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*. https://doi.org/10.48550/arXiv.1704.04861