Nena Beecham
DATS 6303: Deep Learning
Individual Project Report
12/11/2024

Table of Contents

1 Introduction

As generative AI continues to occupy our daily lives, deepfake detection has become of utmost important. In our group project, we sought to leverage the Celeb-DF dataset and deep learning models to deploy a deepfake video detection application on streamlit. Using a transfer-learning approach, we aimed to use two network architectures: CNN + Dense and CNN + GRU. My primary contribution to the group project was the development of four CNN + Dense models. In this report, I go over how I managed the video data, how I built the models, and what my results were.

There are two main folders in my GitHub architecture: src and data. The src folder contains main and components subfolders. The purpose of this structure was to keep the functions and classes I developed separate from the main code to prevent accidental edits or deletions. Within the main folder, there are four subfolders: VGG16, EfficientNetB1, ResNet50, and InceptionV3. These four subfolders contain the train and test files for the models they are named after as well as training images. Other scripts within the main folder include a test_script.py file for generating a prediction for a single video. Since my CNN + Dense architecture only classifies images, the test_script.py file extracts a specified number of frames from a video, makes a prediction for each frame, and returns the average score of the frame predictions as the final prediction for the video.

The data folder contains the fake and real videos, the frames extracted from those videos, and the metadata files for the images and videos. It also contains videos downloaded from YouTube, as well as the frames for those videos. The video and image files were too large to push to GitHub, however, Table 1 outlines the structure they have in my EC2 Instance.

The remainder of the report is structured as follows: In the Section 2, I provide an overview of my individual work. In Section 3, I describe in detail the individual portions of work I completed. In Section 4, I share my modeling results. In the final section, describe what I learned through the project.

Table 1 EC2 Instance Data Structure

Folder Name	Subfolders	Description
Celeb-real	N/A	The real videos from the Celeb-DF dataset.
Celeb-synthesis	N/A	The fake videos from the Celeb-DF dataset.
YT_Fake_Videos	N/A	The deepfake videos downloaded from YouTube.
YT_Real_Videos	N/A	The real videos downloaded from YouTube.
YT_Frames_test	Class_1, Class_2	The frames extracted from the fake and real YouTube videos. These frames are used for testing only.
YT_Frames_test_ela	Class_1, Class_2	The frames extracted from the fake and real YouTube videos after error level analysis was applied. These frames are used for testing only.
Frames_train	Class_1, Class_2	The frames extracted from the fake and real Celeb-DF videos. These frames are used for training only.
Frames_valid	Class_1, Class_2	The frames extracted from the fake and real Celeb-DF videos. These frames are used for validation only.
Frames_test	Class_1, Class_2	The frames extracted from the fake and real Celeb-DF videos. These

		frames are used for testing only.
Frames_train_ela	Class_1, Class_2	The frames extracted from the fake and real Celeb-DF videos after error level analysis was applied. These frames are used for training only.
Frames_valid_ela	Class_1, Class_2	The frames extracted from the fake and real Celeb-DF videos after error level analysis was applied. These frames are used for validation only.
Frames_test_ela	Class_1, Class_2	The frames extracted from the fake and real Celeb-DF videos after error level analysis was applied. These frames are used for testing only.

2 Description of Individual Work

In terms of individual work, there were several areas I contributed. When we were contemplating doing deepfake image classification for our project, I developed a proposal for that project. Although we did not move forward with that project, it is similar to our current project and helped provide some guidance. The second area I contributed was model training and testing. I developed a total of four finetuned models for deepfake video detection, including ResNet50, EfficientNetB1, VGG16, and InceptionV3. The third area I contributed was plot and image creation. We did not have many plots to work with, so my training plots for my model were useful additions to our group report and presentation. Additionally, I was able to tailor and run the ELA script found by Vishal. This allowed us to present what ELA does on real and fake images.

Finally, I contributed to the development of both the group report and group presentation.

3 Portions of Work Completed

3.1 Experimental Setup

3.1.1 Batching and Image Generators

Processing video and image data requires significant computational resources. Coupled with the computational power required to run large deep learning models, memory allocations can be easily exhausted. One work around to this is batching and the use of image generators. Using keras ImageDataGenerator objects, a workaround was developed for the issue of memory allocation. The ImageDataGenerator allows for efficient memory allocation by streaming batches of images during training, instead of holding the entire image dataset in memory. Using image generators allowed for faster training times as well as experimentation with different batch sizes. For my models, batch sizes of 64 were used. Furthermore, the flow_from_directory method was used. This method takes a folder of images with subfolders dividing the classes. Batches are then generated with samples from each subfolder.

3.1.2 Class Weights

The Celeb-DF dataset contains an extreme class imbalance. There are more samples of deepfake videos compared to samples of real videos. If left unaddressed, this class imbalance could have could have hindered the models' performance, particularly for detecting real images. Two approaches were used to address this problem. The first was the implementation of class weights.

Given that there are more samples of fake videos compared to real videos, the real video samples could be provided heavier class weights. Through providing heavier weights to real videos, the loss function would be adjusted real video errors are more heavily penalized than fake video errors. As a result, the models will concentrate more on real video samples training. Since image generators were used to batch the images, a separate function was developed to calculate the class weights. As detailed in Table 2, the function takes as input a generator object and a desired number of steps. Next, it takes into account the number of total samples and number of classes to calculate the class weights.

Table 2 Function to Compute Class Weights for Generators

```
def compute_class_weights_from_generator(generator, steps):
    label_counts = Counter()

    for _ in tqdm.tqdm(range(steps)):
        _, y_batch = next(generator)

        if len(y_batch.shape) > 1:
            y_batch = np.argmax(y_batch, axis=1)

        label_counts.update(y_batch)

    total_samples = sum(label_counts.values())
    num_classes = len(label_counts)
    class_weights = {
        cls: total_samples / (num_classes * count) for cls, count in label_counts.items()
    }

    return class_weights
```

3.1.3 Data Augmentation

The second approach to the class imbalance issue was data augmentation. Additional samples of the real videos class were created by adding transformations to the video frames. The first transformation applied was a random crop. The benefit of this transformation is that it urges the model to concentrate on different aspects of the image. Next, a resizing layer was added to resize the image to the desired image size after the random crop reduced the image height and width. Next, the images were randomly flipped. Finally, random contrast and brightness were added to the images. These two layers made the model more versatile in that it could handle different lighting conditions.

3.2 Deep Learning Network and Algorithms

3.2.1 CNN Classification Head

For the CNN + Dense architecture, there were a total of four pretrained models that were experimented with: ResNet50, InceptionV3, EfficientNetB1, and VGG16. Initial training showed that VGG16 and EfficientNetB1 performed best out of the four models, so these two were chosen for additional training. Each model contains its own characteristics that made it a suitable choice for experimentation with deepfake detection. VGG16 was among the first models tested as its architecture is easy to understand and when compared[1] with the other three models, its time per inference step on a GPU was the lowest. Furthermore, a study conducted by Boongasame et el. (2024) [1]

[1] https://keras.io/api/applications/

on VGG16 for deepfake detection with the Celeb-DF dataset found that a VGG16-LSTM architecture had the lowest training time and highest accuracy when compared to VGG19-LSTM and ResNet101-LSTM architectures.

EfficientNetB1 contains the least number of parameters out of all four models and is known for being able to capture intricate patterns and features within images. Typically, models with lower numbers of parameters are able to run without exhausting available memory and computational power. The benefit of EfficientNetB1 is that it is able to preserve computational resources while also not sacrificing accuracy. Additionally, research [2] has shown that EfficientNet-B4, which is also in the EfficientNet family of models, performs well when applied for deepfake detection on the FaceForensics++ dataset and Celeb-DF dataset.

Table 3 Dense Classification Head

```
def add_custom_layers(self):
    """
    Add custom top layers to the model.
    """
    x = self.base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(256, activation='relu')(x)
    x = Dense(128, activation='relu')(x)
    x = Dense(64, activation='relu')(x)
    predictions = Dense(1, activation='sigmoid')(x)
    self.model = Model(inputs=self.base_model.input, outputs=predictions)
```

As detailed in Table 3, the final classification head consisted of a GlobalAveragePooling2D layer, and four dense layers, including a final layer with sigmoid activation function to make the prediction. In addition to this classification head, an attention-based head was also experimented with. Using an attention mechanism allows the model to concentrate on certain parts of the image. This is important in deepfake detection, where peculiarities may be more visible in parts of the image like the eyes and mouth, as opposed to other areas.

A CNN attention-based architecture has also been carried out in research. Majid et el. (2022) [3] used a transfer-learning approach combined with an attention mechanism to detect fires in real-world fire breakout images. Using an EfficientNet model, they were able to detect fire in images with a recall score of 97.61%. Chen et al. (2016) [4] presented a visual attention based convolutional neural network (CNN) to solve the image classification problems. Simulating "human vision mechanism[s] of multi-glance and visual attention", the attention-based CNN provided stronger results than a VGG58 model.

Table 4 represents the code used to develop the attention-based classification head. The first Dense layer called "attention" is where the attention mechanism starts. Weights are generated for each feature map that is outputed from the GlobalAveragePooling layer. These weights allow the model to highlight key features. In the Multiply layer, the generated attention weights are applied to the feature maps. The following layers compress the new feature maps into a single vector and a prediction is made for the image. Unfortunately, when compared to the classification head in Table 1, the attention-based head did not produce as great metrics. Consequently, further experiments were only carried out on the first classification head.

Table 4 Attention-Based Classification Head

```
def add_custom_layers(self):
    """
```

```
    Add custom top layers to the model.
    """
    x = self.base_model.output
    gap = GlobalAveragePooling2D()(x)
    attention = Dense(self.base_model.output_shape[-1], activation='sigmoid')(gap)
    attention_output = Multiply()([x, attention])
    gap_output = GlobalAveragePooling2D()(attention_output)
    predictions = Dense(1, activation='sigmoid')(gap_output)

    self.model = Model(inputs=self.base_model.input, outputs=predictions)
```

3.2.2 Freezing and Unfreezing Base Model Layers

After the dense classification head was applied to the VGG16 and EfficientNetB1 models, the process of freezing and unfreezing their pretrained layers began. The first training process involved freezing all of the base model layers for feature extraction. In this step, key features and patterns useful in differentiating between real and fake images were developed. The second training process involved unfreezing some or all of the pretrained layers, allowing weight updates for these layers. This phase can be considered the finetuning phase where the model adapts to the Celeb-DF dataset.

A different number of layers were unfrozen for both of our pretrained models, as detailed in Table 5. For VGG16, all of the base model layers were unfrozen. Prior to this, 5 and 10 layers were unfrozen and experimented with. However, the finetuned model performed best when all layers were unfrozen. This suggests that the pretrained features that were learned from the base model and ImageNet dataset were not transferable to the Celeb-DF dataset, and that more weight updates were required. For EfficientNetB1, 30 and 80 of the pretrained layers were unfrozen. Similar to VGG16, unfreezing more of the base layers led to better performance.

Table 5 Freezing of Base Model Layers

Model Name	Epochs Trained Frozen	Pretrained Layers Unfrozen	Epochs Finetuned
VGG16	15	All	25
EfficientNetB1	15	80	25

4 Results

4.1 Finetuned VGG16

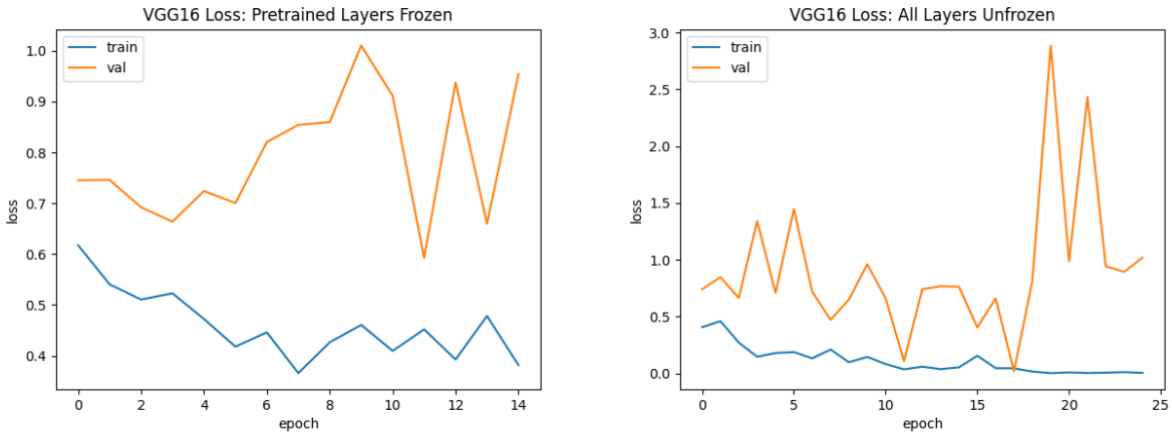

Figure 1 VGG16 Loss Plots

Figure 1 represents the training loss for the two training phases of the VGG16 model. The plot to the left displays the loss during the feature extraction process. In this plot, the training loss appears to continue decreasing while the validation loss appears to increase for about 10 epochs before fluctuating. The increase in the validation loss seems to suggest that the model is overfitting to the training data. The plot to the right depicts the loss during the finetuning process. In this plot, the training loss steadily decreases while the validation loss appears to decrease for 17 epochs before drastically increasing. Again, this suggests the model began to overfit at some point.

Figure 2 represents the test metrics for the finetuned VGG16 model on the tests Celeb-DF and YouTube datasets. On the test Celeb-DF dataset, it achieved overall precision, recall, and f1 scores of about 75%. When taking into account the classes, the model had higher metrics for the fake images. This makes sense when considering the class imbalance. For real images, the model's predictions were correct a little over half of times. On the test YouTube dataset, the model did not perform as well. Here, the model exhibited precision, recall, and f1 scores of about 47%. With the exception of the precision score, the individual metrics for each class were starkly different. For real images, the model exhibited recall and f1 scores of 59% and 51%, opposed to fake images where the scores were 36% and 41%. Given that the YouTube dataset was more balanced, this seems to suggest the model is better at identifying real images as opposed to fake images.

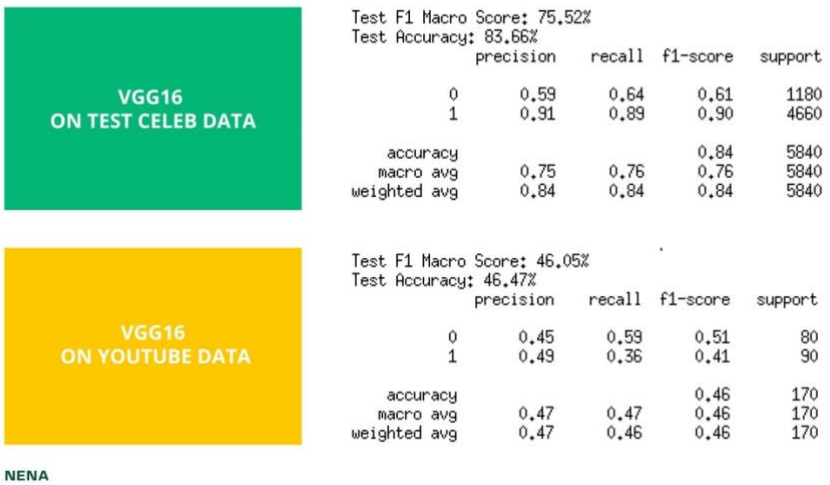

Figure 2 Test Performance of Finetuned VGG16

4.2 Finetuned EfficientNetB1

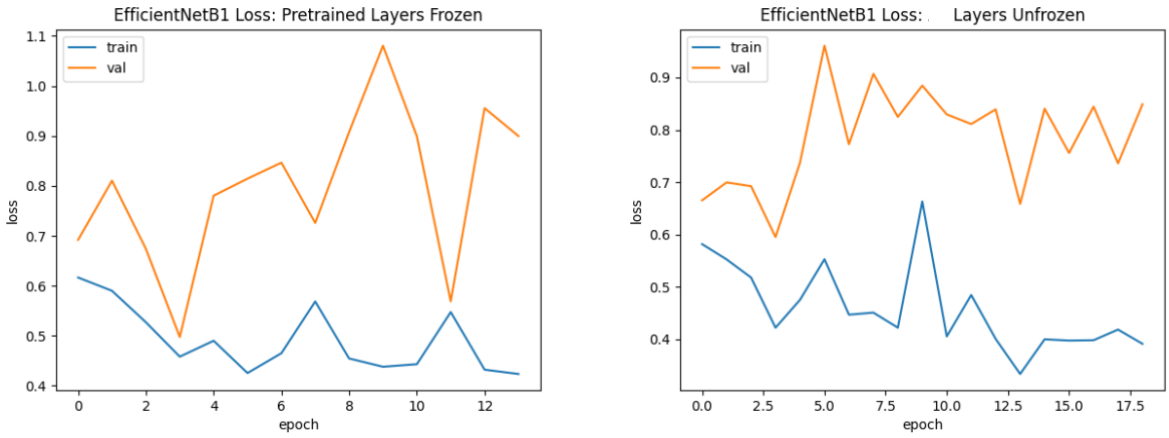

Figure 3 Training Loss for EfficientNetB1

Figure 3 represents the training loss for the two training phases of the EfficientNetB1 model. The plot to the left displays the loss during the feature extraction process. In this plot, the training loss appears to continue decreasing while the validation loss appears to fluctuate throughout training. The overall increase in the validation loss seems to suggest that the model began to overfit to the training data. The plot to the right depicts the loss during the finetuning process. In this plot, the training loss fluctuates a bit but decreases overall. The validation loss appears to increase overall. Similarly, this suggests the model began to overfit at some point.

Figure 4 represents the test metrics for the finetuned EfficientNetB1 model on the tests Celeb-DF and YouTube datasets. On the test Celeb-DF dataset, it achieved overall precision, recall, and f1 scores of around 50%. When taking into account the classes, the model exhibited stronger metrics for the fake images. Again, this is partially due to the class imbalance. For real images, the model's predictions were correct around a quarter of times. On the test YouTube dataset, the model performed slightly better. Here, the model exhibited precision, recall, and f1 scores of around 55%. When taking into account the metrics of each individual class, the model seems to be better at identifying real images.

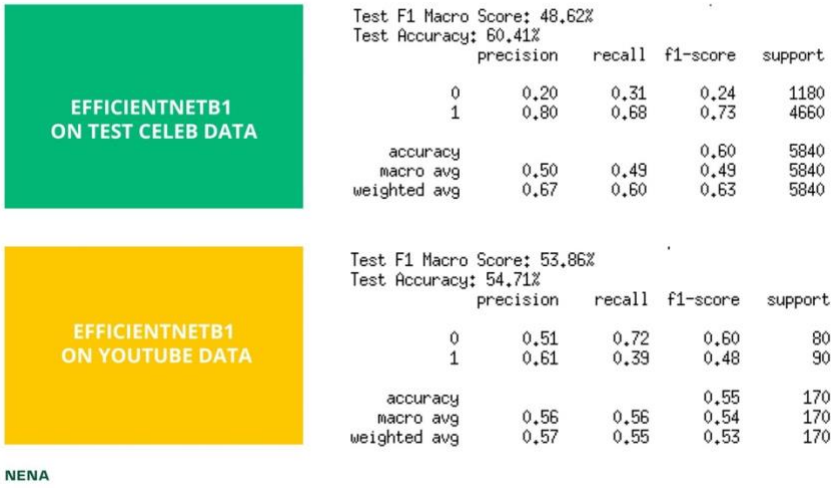

Figure 4 Test Performance of EfficientNetB1

When comparing the performance of the finetuned VGG16 and EfficientNetB1 models, there are a few conclusions to be made. First, the VGG16 model outperformed the EfficientNetB1 model for the Celeb-DF images. The VGG16 model's precision score represented a 50% increase from the EfficientNetB1's score: from 50% to 75%. However, on the YouTube dataset, the EfficientNetB1 outperformed the VGG16 model. The EfficientNetB1 model's precision score represented a 20% increase from the EfficientNetB1's score: from 47% to 56%. For each model, we can safely assume that when applied to other deepfake datasets, its predictions will be correct about half of the time.

5 Learning Curve and Conclusion

5.1 Video Classification

There were several individual learning curves for me within this project. The greatest learning curve was that I have not worked with a video classification problem before. Thankfully, I have worked with image classification tasks. That previous work helped provide an initial foundation for the group project. However, there were still many things to learn. To help get over the initial learning curve, I read tutorials and other video classification projects. These projects gave me a useful method to approach the problem with. In the future, I would like to work more on sequence-based methodologies such as RNNs.

5.2 Target Leaking

The second learning curve was approaching the problem of target leaking. For the very first VGG16 model I finetuned, the model showed exceptionally high accuracy scores at 93%. However, after meeting with my groupmates, we discussed the issue of target leaking. This happened because train, test, and validation sets were being developed after frames were extracted from the videos. A single video could have frames sent to all three sets, inadvertently leaking target information. To solve this issue, we decided to separate the videos into individual train, test, and validation sets, and then extract frames from those given sets. Bharat helped with this by developing the first code to segment the videos into train and test sets. I modified his code to split the data into train, test, and validation sets for the CNN + Dense architecture.

5.3 Model Architecture

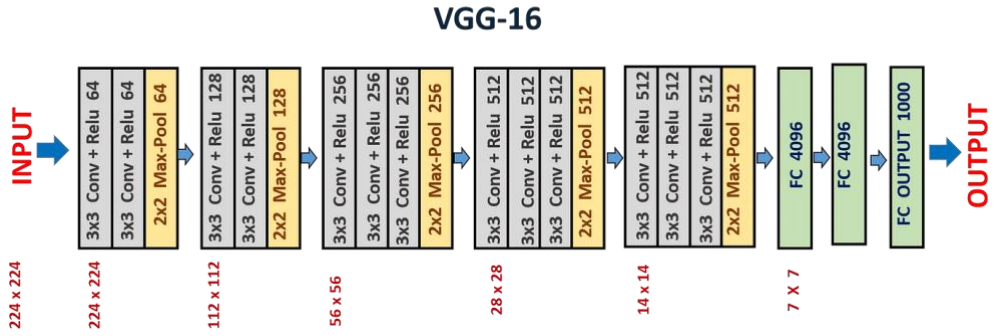

Figure 5 VGG16 Model Architecture

While not necessarily a learning curve, the group project allowed me to interact more intimately with the model architectures of ResNet50, InceptionV3, VGG16, and EfficientNetB1. In previous cases where I had to use a pretrained model, I would pay less attention to the model architecture and more attention to what research has shown is the best model for a given type of project. However, for the case of the group project, we had to be strategic about

freezing and unfreezing pretrained layers. This encouraged me to pay more attention to the model architectures and how I could approach freezing and unfreezing their base layers.

5.4 ELA

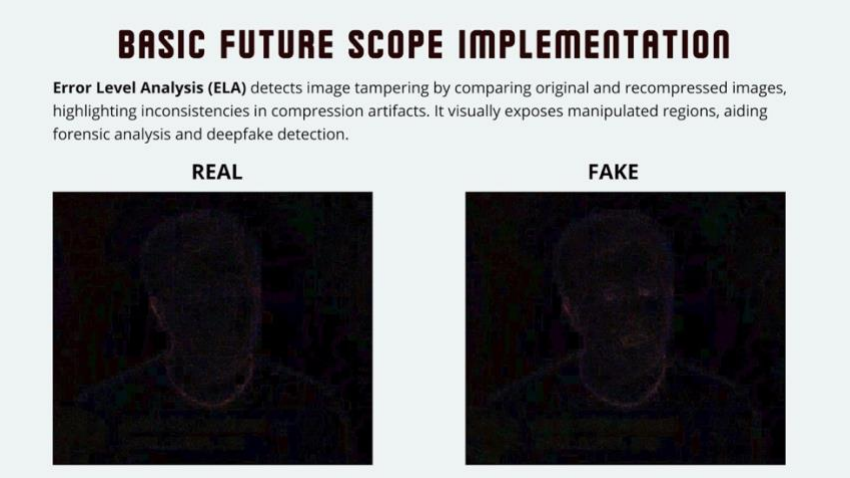

Figure 6 ELA Image Comparison

An additional area in which I was able to learn was error level analysis (ELA). ELA is a process through which images are compressed to highlight inconsistencies in compression artifacts and potentially manipulated regions. In the case of deepfake detection, the video manipulation was mostly visible in the eyes and mouth. The research for this methodology was conducted by Vishal. Furthermore, he provided a script which I was able to modify and run. Through his research contributions, I discovered a new and exciting approach that can be taken towards deepfake detection.

6 Disclosure: Outside Code

Calculate the percentage of the code that you found or copied from the internet. For example, if you used 50 lines of code from the internet and then you modified 10 of lines and added another 15 lines of your own code, the percentage will be $\frac{50-10}{50+15} \times 100.$

Figure 7 Percentage of Outside Code Formula

For most of the initial code to extract frames from images, I followed tutorials found on the internet. This helped me get over the initial learning curve for the project. Following this, for training and testing I tried to write the code on my own first. However, there were issues when I ran into memory constraints and needed to use image generators. I consulted the internet for this issue as well. Overall, for the majority of code I found on the internet, I needed to modify it for our specific project. Using the formula in Figure 7, I estimate that for every 100 lines of code I found on the internet, I modified 40 lines of code. Furthermore, I added 25 lines of my own code. This brings the final calculation for the percentage of outside code to 48%.

References

1. Boongasame, Laor, Boonpluk, Jindaphon, Soponmanee, Sunisa, Muangprathub, Jirapond, Thammarak, Karanrat, Design and Implement Deepfake Video Detection Using VGG-16 and Long Short-Term Memory, *Applied Computational Intelligence and Soft Computing*, 2024, 8729440, 11 pages, 2024. https://doi.org/10.1155/2024/8729440
2. Yasser, Basma & Hani, Jumana & Elgayar, Salma Mohamed & Abdelhameed, Omar & Ahmed, Nourhan & Ebied, Hala & Amr, Habiba & Salah, Mohamed. (2024). Deepfake Detection Using EfficientNet and XceptionNet. 10.1109/ICICIS58388.2023.
3. Saima Majid, Fayadh Alenezi, Sarfaraz Masood, Musheer Ahmad, Emine Selda Gündüz, Kemal Polat, Attention based CNN model for fire detection and localization in real-world images, Expert Systems with Applications, Volume 189, 2022, 116114, ISSN 0957-4174, https://doi.org/10.1016/j.eswa.2021.116114.
4. Y. Chen, D. Zhao, L. Lv and C. Li, "A visual attention based convolutional neural network for image classification," 2016 12th World Congress on Intelligent Control and Automation (WCICA), Guilin, China, 2016, pp. 764-769, doi: 10.1109/WCICA.2016.7578651