Deep Fake Video Classification using Deep Learning Techniques

George Washington University

Final Term Project

Mentor: Prof. Amir Jafari

Authors

Vishal Bakshi
Department of Data Science
George Washington University
Vishal.bakshi@gwu.edu

Nena Beecham
Department of Data Science
George Washington University
nbeecham@gwu.edu

Bharat Khandelwal
Department of Data Science
George Washington University
bharat.khandelwal@gwu.edu

Khush Shah
Department of Data Science
George Washington University
khushjayant.shah@gwu.edu

Table of Contents:

Introduction

The exponential growth of multimedia content has revolutionized how we communicate and share information. Approximately 3.2 billion images and 720,000 hours of videos are shared daily (source). However, this surge in content has also given rise to challenges such as misinformation and deepfake manipulation.

Deepfakes—synthetically generated or altered videos and images—pose a significant threat to digital trust. These manipulations have been used for:

- Political Propaganda: Fake videos of politicians making inflammatory statements.
- Financial Fraud: Identity theft through morphed images.
- Misinformation Campaigns: Spreading false narratives with convincing fake visuals.

The societal impact of deepfakes is profound, ranging from eroding public trust to causing financial losses and reputational damage. Detecting these manipulations is crucial to mitigating their harmful effects.

Our project aims to address this issue by developing robust deep-learning models for forgery detection. By leveraging state-of-the-art datasets like CelebDF V2 and implementing advanced architectures such as GRU, and VGG. We strive to create a reliable system for identifying manipulated multimedia content.

The report is structured as follows: In the Dataset section, we introduce our dataset and the information that is contained within it. In the Experimental Setup section, we discuss our strategy for batching and for developing class weights, as well as data augmentations. In the Deep Learning Network Architecture section, we discuss our approach to transfer learning as well as the classification heads developed for the CNN + Dense and CNN + GRU models. In the Results section, we share our performance metrics for each network architecture. Finally, in the last section we share what we learned and ideas for future work.

Dataset Description

Source: Celeb DF V2 Kaggle

Celeb-DF V2 is specifically designed to address the growing concern of deep fakes.

- It contains both real and fake (deepfake) videos of celebrities.
- This unique feature makes it invaluable for research and development in deep fake detection algorithms.

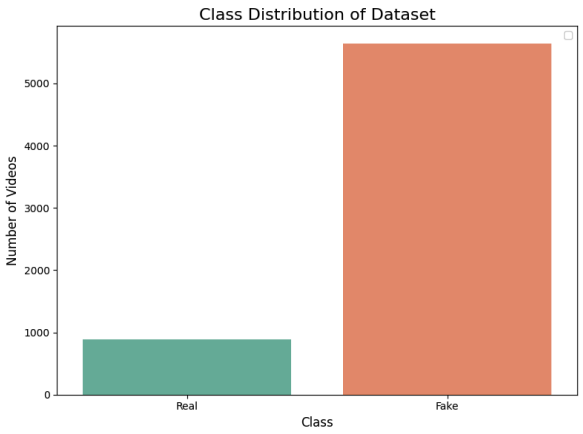


Fig 1. Distribution of Dataset

- There is clear imbalance in the dataset with fake videos reaching at its peak.

YouTube Dataset

In order to appropriately assess how well the models performed on outside data, video samples were taken from YouTube as well. The real and fake videos sampled from YouTube were YouTube "Shorts," so they were all under a minute in length. A total of 8 real videos and 9 deepfake videos were used. The videos belonging to each class had diverse lighting, zooms, and actions being carried out. Additionally, the videos were balanced in terms of sex. Half of the videos had female subjects, while the other half had male subjects.

Experimental Setup

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 the CNN + Dense architecture, 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.

Class Weights

The Celeb-DF dataset contains an extreme class imbalance. There are more samples of deep fake videos compared to samples of real videos. If left unaddressed, this class imbalance 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 with 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 1, 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.

```
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
```

Table 1 Function to Compute Class Weights for Generators

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.

Deep Learning Network and Algorithms

CNN + Dense Architecture

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 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] 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.

```
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)
```

Table 1 Dense Classification Head

As detailed in Table 1, 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 2 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.

```
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)
```

Table 2 Attention-Based Classification Head

Frozen and Unfrozen Layers

After the dense classification head was applied to the VGG16 and EfficientNetB1 models, the process of freezing and unfreezing their pre trained 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 3. 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.

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

Table 3 Freezing of Base Model Layers

CNN + GRU Architecture

After preprocessing and getting the cropped faces from the frames we pass it through the pretrained model InceptionV3 which actually is very good at feature extraction, we extract the 2048 features and keeping the max sequence length of extracting frames as 16 we then pass it through the GRU model.

The GRU model architecture is designed to process sequential data, specifically video frames represented as feature vectors. The architecture comprises the following layers

1. Input Layers:

- **frame_features:** This layer accepts a sequence of feature vectors extracted from each video frame. The vectors are of shape (None, 16, 2048), where:
    - None: Represents the batch size, which can vary.
    - 16: The maximum sequence length, i.e., the number of frames considered in each input sequence.
    - 2048: The dimensionality of each feature vector.
- **mask:** This layer takes a mask tensor of shape (None, 16), used to handle variable-length sequences by masking out unused positions.

2. Recurrent Layers:

- **GRU Layers:** The core of the model consists of multiple Gated Recurrent Unit (GRU) layers. GRUs are effective in capturing long-range dependencies within sequential data. The layers are stacked with batch normalization layers to improve training stability and generalization:
    - **GRU layer 1:** Takes the output of the frame_features and mask layers as input.
    - Batch normalization layer 1: Normalizes the output of the first GRU layer.
    - **GRU layer 2:** Takes the output of the first batch normalization layer as input.
    - **Batch normalization layer 2:** Normalizes the output of the second GRU layer.
    - **GRU layer 3:** Takes the output of the second batch normalization layer as input.

3. Dense Layers:

- **Dense Layer:** A fully connected layer that transforms the output of the final GRU layer into a higher-dimensional representation.
- **Dropout Layer:** A dropout layer is applied to prevent overfitting by randomly dropping out units during training.
- **Output Layer:** A final dense layer with a single output neuron, producing the model's prediction.

4. Model Summary:

- Total Parameters: 418,657
- Trainable Parameters: 418,465
- Non-trainable Parameters: 192

This model architecture is well-suited for tasks such as video classification, action recognition, and other applications involving sequential video data.

Results

CNN + Dense Models (Finetuned VGG16)

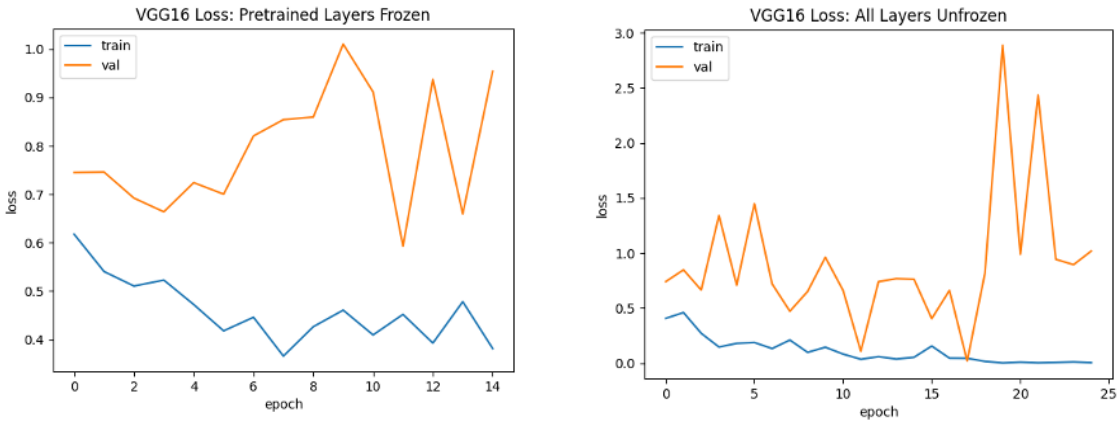


Figure 2 VGG16 Loss Plots

Figure 2 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 3 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.

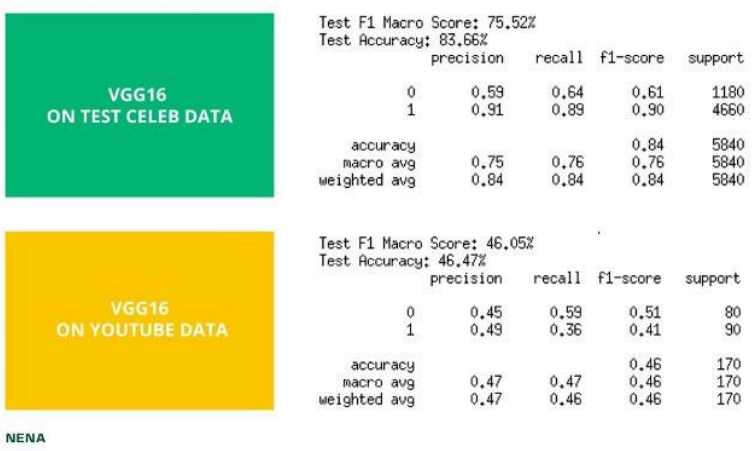


Figure 3 Test Performance of Finetuned VGG16

Finetuned EfficientNetB1

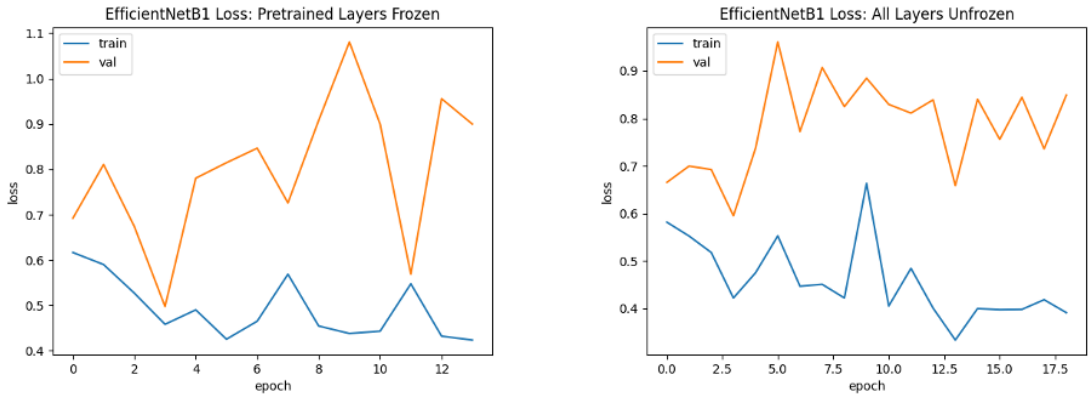


Fig. 4 Training Loss for EfficientNetB1

Figure 4 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 5 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.

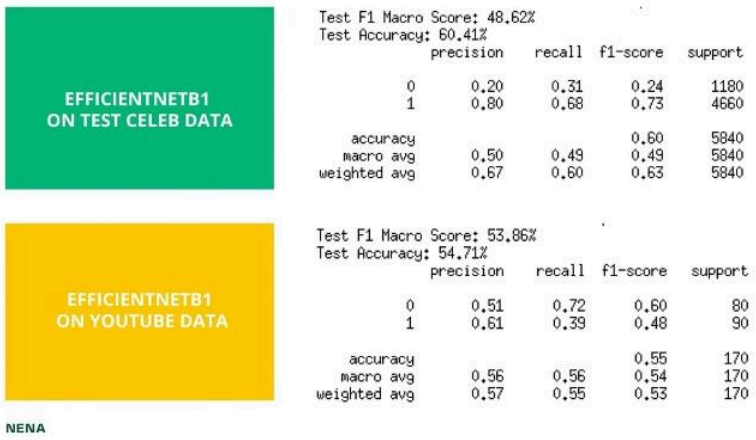


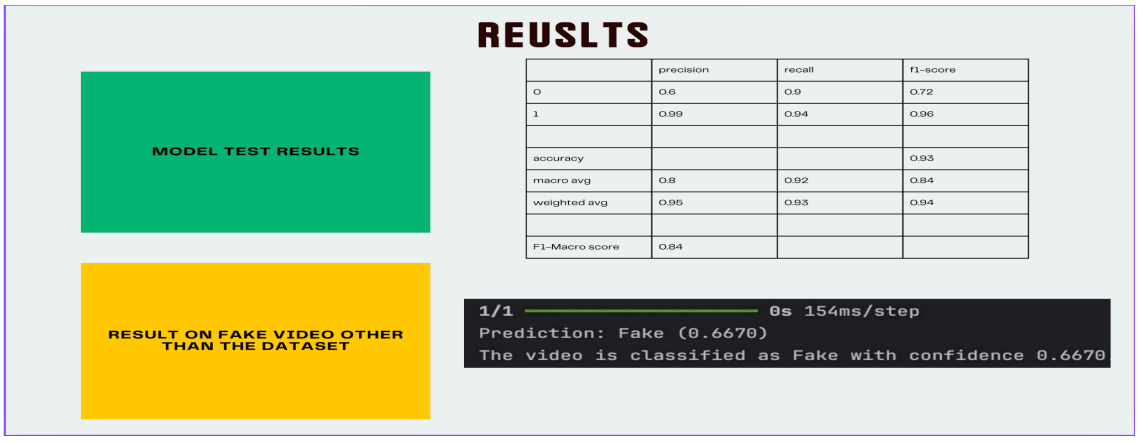
Figure 5 Test Performance of EfficientNetB1

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.

CNN + GRU Model

The model was evaluated on a dataset of real and fake videos. The following performance metrics were obtained:

- Precision: 0.95
- Recall: 0.93
- F1-Score: 0.94
- Accuracy: 0.93



These results indicate that the model is accurate in classifying both real and fake videos. The F1-score, which balances precision and recall, is particularly high, suggesting that the model is both precise and sensitive, but this is not completely true and it has a lot of overfitting because of the class imbalance as focal loss was not correctly used, other than that number of real videos were very less compared to fake videos.

Additional Testing:

The model was further tested on a fake video that was not included in the training dataset. The model correctly classified this video as fake with a confidence score of 0.6670.

Overall:

The model has an overfitting issue because of which it is not able to classify the real video correctly. The model with bidirectional GRU is also developed which performed better compared to this model with a f1 macro score of 0.49.

Conclusion:

In this project, we sought to address the issue of deepfake videos by developing robust deep-learning models. Using the CelebDF V2 dataset and taking a transfer learning approach, we were able to develop two deep learning network architectures. The CNN + Dense architecture leveraged the strength of pretrained models including VGG16 and EfficientNetB1 and added additional dense layers on top for classification heads. This architecture was giving better results in comparison to the CNN + GRU as the GRU architecture has an overfitting issue which can be resolved with proper handling of class imbalance and adding complexity using bidirectional GRU.

Future Scope

Future work includes the following:
multi-modal analysis for deepfakes, performing masking techniques and adding as auxiliary data, training the models on ELA (Error-Level Analysis) images, and implementing encoder-decoder models.

We have performed an initial analysis of the ELA image on the Inception and Vgg model and it performed really well without any advanced preprocessing. We will pursue to train the model with ELA based videos/images to see the progress.

References

1. https://www.nature.com/articles/s41598-023-34629-3
2. 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
3. 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.
4. 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.
5. 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