AI4E Assignment 3 Report

Jarod Tang, Xinyu Shen, Chethaka Jayasekaramudeli

May 2024

Contents

1 Overview **1**

2 Getting Data and Pre-processing **2**
2.1 Explaination 2
2.2 Data Extraction 2

3 Classification Methods **3**
3.1 InceptionV3 3
3.2 ResNet50 4

4 Results **4**

5 GradCAM visualisation **6**

6 Potential Future Extensions and Improvements **9**

1 Overview

Deepfake technology, which leverages deep learning to create hyper-realistic fake videos and images, has emerged as a significant challenge in digital media. These manipulations often involve altering facial features or synthesizing entirely new faces, making it difficult to distinguish between genuine and fabricated content. The rapid advancements in deepfake generation have raised concerns regarding misinformation, privacy violations, and potential threats to individuals and institutions.

This assignment aims to address the challenges posed by deepfakes through the development and evaluation of advanced machine learning models. By leveraging variants of traditional convolutional neural networks (CNNs) such as ResNet and InceptionNet. This study seeks to enhance the accuracy and reliability of deepfake detection.

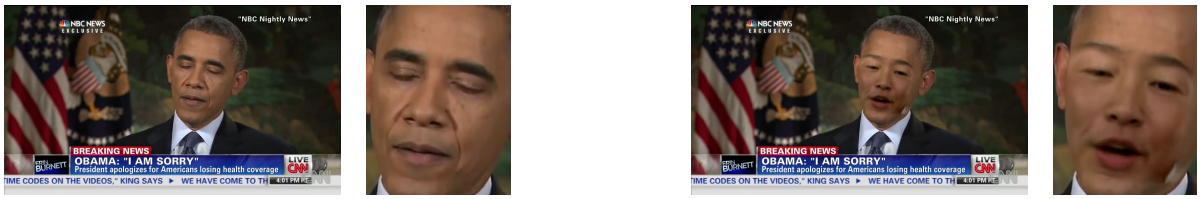
2 Getting Data and Pre-processing

2.1 Explaination

The video data of both the original video and the DeepFake videos are fetched from Face Forensics++ [4]. Since this task only focuses on deepfake detection, it is only necessary to download the original video and the deepfake videos.

Every single frame from each of the videos are then extracted into folders that are named after the video IDs so it is easy to know what images relate to which video.

For any given image, the face region is the most important for the deepfake detection. MTCNN [1] can be used to crop the images so that only the face is remaining. A GPU based MTCNN model was used from the 'Facenet-Pytorch' module to speed up the preprocessing steps. See Figure 1.



(a) Full image (b) Cropped image (c) Full image (d) Cropped image

Original Deepfake

Figure 1: MTCNN demo

2.2 Data Extraction

Each video file can contain several hundred frames, and most of the frames extracted from the same video are relatively similar. Using hundreds of pictures can not only be computationally intensive, but it may also cause the model to be over-fitted to a specific face. Therefore, it is ideal to have a wider variety of faces to train the model. The steps to properly extract the data is listed below.

1. 250 of both the original and deepfake videos are downloaded.
2. Every single frame of each video is converted into an image.
3. For every 100 images extracted, only 1 of these images is retained.
4. MTCNN is used to extract and crop the faces of remaining images.
5. Since only the cropped images are to be used in later steps, all vidoes and original images can be deleted to save disk space.
6. Split the deepfake and original images into 3 files. 70% images in training set, 20% images in validation set, 10% images in testing set.

This process leads to a dataset of around 2300 images in total with a 50:50 ratio between real and manipulated images. The above steps are a conceptual overview, the specific list of commands to execute are outlined in the `README.md` file. If all steps are performed correctly, a file titled `organized_data` should be found in the main directory. Its file configuration should resemble figure 2. Where the files named `manipulated` contains the cropped Deepfake image (by MTCNN), and the `original` file contains the cropped original images.

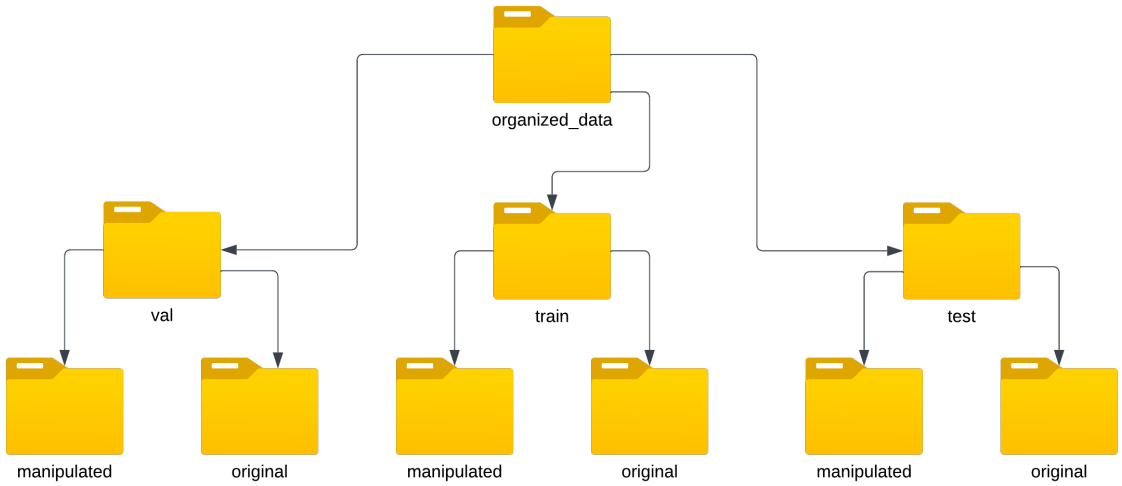


Figure 2: File structure for input data

3 Classification Methods

Due to hardware limitations, the number of images being used needs to be limited to a few thousand for realistic training times. Pre-trained models can be beneficial for this task, as they leverage learned representations from large-scale datasets, providing a strong starting point for fine-tuning on specific tasks with limited data. Moreover, this transfer learning approach is highly effective as the models have already learnt general features that are useful for many image classification tasks through datasets such as ImageNet. The following sections will compare the difference between three different models, InceptionV3 and a trained and untrained version of ResNet50. More specifcally, one version of ResNet50 will not utilised the pre-trained weights. These models expect the images to be loaded in the range of [0, 1] and then normalized using mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225] (Standard weights for images trained on ImageNet). All three models are trained for 20 epochs.

3.1 InceptionV3

Designed to achieve high performance in image recognition tasks, this model utilises a series of inception modules that perform multiple convolutions of different sizes in parallel, along with factorised convolutions and asymmetric convolutions to reduce computational cost [3].

Auxiliary classifiers are inserted inbetween layers during training to improve gradient flow and provide regularisation. The architecture is depicted in Figure 3 below.

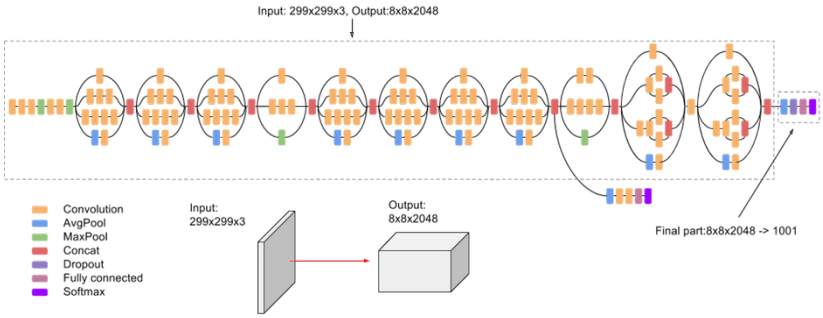


Figure 3: InceptionV3 Architecture [3]

InceptionV3 expects input size of $(3 \times H \times W)$, where H and W should be at least 299. Hence, for the InceptionV3 version of the script, the images are pre-processed according to the aforementioned conditions.

Loss function: `nn.CrossEntropyLoss()`

Optimiser: `optim.SGD(model.parameters(), lr=0.001, momentum=0.9)`

LR scheduler: `optim.lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.1)`

3.2 ResNet50

ResNets or Residual networks are a type of deep convolutional neural network (CNN) architecture that uses the concept of residual learning to bypass one or more layers. Moreover, this helps to bypass the vanishing gradient problem. This enables the training of very deep networks without suffering the consequences of the degradation problem.

Similar to InceptionV3, ResNet expects an input size of $(3 \times H \times W)$, where H and W should be at least 224 pixels.

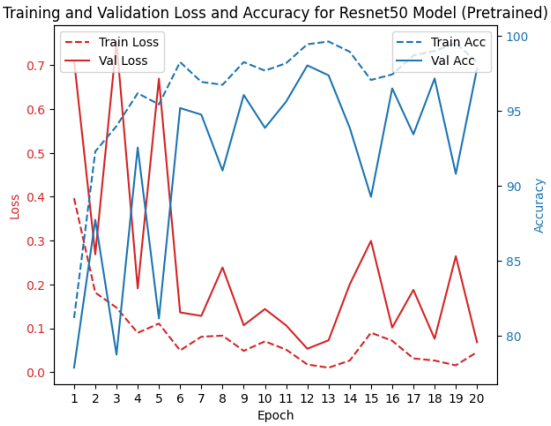
Loss function: `nn.CrossEntropyLoss()`

Optimiser: `optim.Adam(model.parameters())`

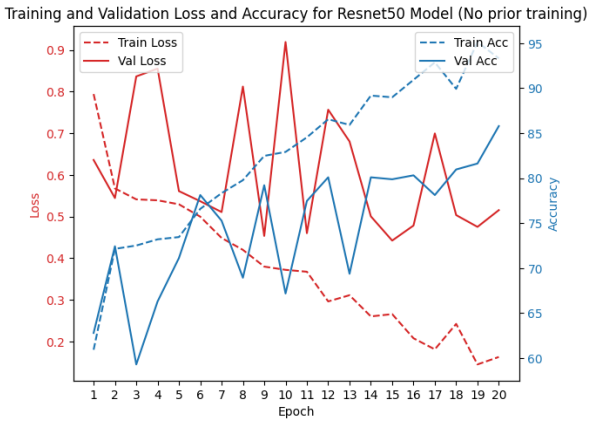
4 Results

Figure 4 depicts training and validation loss and accuracies for each of the following models.

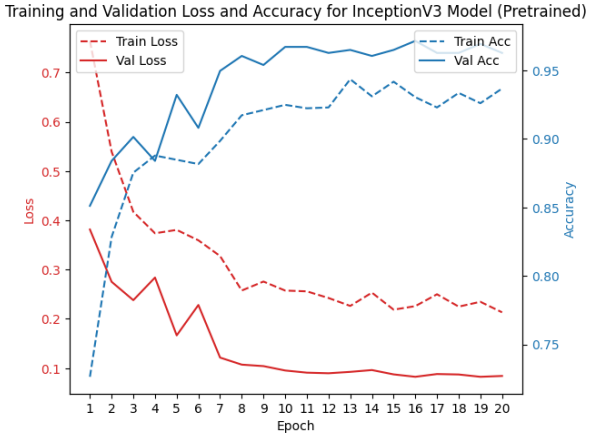
From the graph depicted in Figure 4a, it can be inferred that the model performs well on the training data. This is because both the training loss and accuracy stablise relatively quickly. In comparison however, the validation accuracy and loss both fluctuate as the number of epochs increases. This may indicate some instability or overfitting.



(a) ResNet50 Results (Pretrained)



(b) ResNet50 Results (Untrained Weights)



(c) InceptionV3 Results

Figure 4: Comparison of Model Results

Figure 4b depicts the performance of the ResNet50 model without using pre-trained weights. Similar to Figure 4a, both the validation accuracy and loss fluctuate significantly. Moreover, whilst the model is learning, it learns at a slower rate than its trained counterpart. This reduction in learning quality could be attributed to weights not being assigned prior to training. It is likely that this model also suffers from overfitting.

Figure 4c depicts the performance of the InceptionV3 model. Overall, it demonstrates efficient learning with low training and validation losses. Moveover, both the validation and training accuracies are high and stable, demonstrating strong performance and proper generalisation.

Over 20 epochs, Tables 1, 2 and 3 show the performance of ResNet50 (trained and untrained) and InceptionV3 on the training data.

Comparing the results in the three tables, ResNet50 with pre-trained weights performs the best with 96.61 % on the testing dataset and a minimal training loss of 0.0662. ResNet50 without pre-trained weights performs the worst on this dataset, with a testing accuracy

	Accuracy	Loss
Training Dataset	99.25%	0.0221
Testing Dataset	96.51%	0.0662

Table 1: Training and Testing Accuracy and Loss for ResNet50 (pretrained)

	Accuracy	Loss
Training Dataset	92.55 %	0.2486
Testing Dataset	95.63 %	0.1089

Table 2: Training and Testing Accuracy and Loss for InceptionV3 (pretrained)

	Accuracy	Loss
Training Dataset	95.12 %	0.1209
Testing Dataset	86.03 %	0.3832

Table 3: Training and Testing Accuracy and Loss for ResNet50 (untrained)

of 86.03 % and loss of 0.3832. A main reason for this massive disparity in performance is that ResNet50 untrained must learn all the features from scratch, since it will randomly initialise its weights upon testing. This in turn, leads to slower learning, higher computational requirements and poorer performance on the task at hand.

5 GradCAM visualisation

Grad-CAM (Gradient-weighted Class Activation Mapping) is a technique used to produce visual explanations for the decisions made by CNNs [2]. It highlights the important regions in an input image that contribute most to the model's prediction, providing insights into what the model is focusing on, which is the altered facial regions in the deepfake detection model.

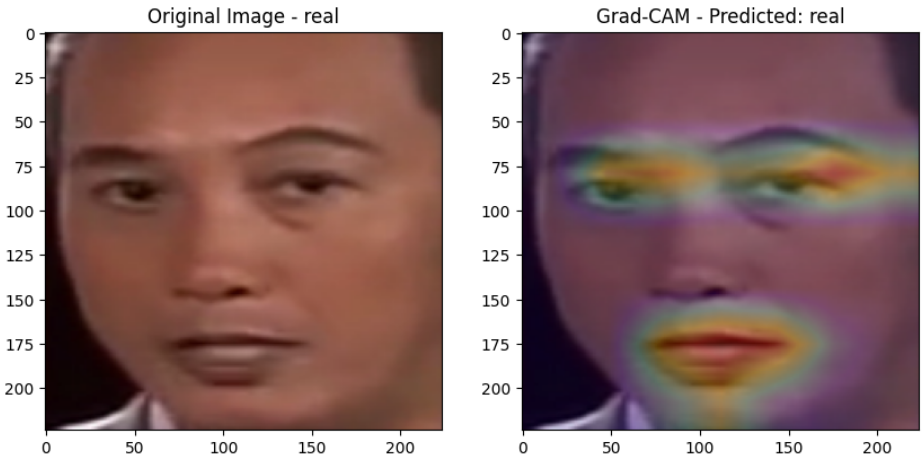


Figure 5: Original image detection using ResNet50 pretrained

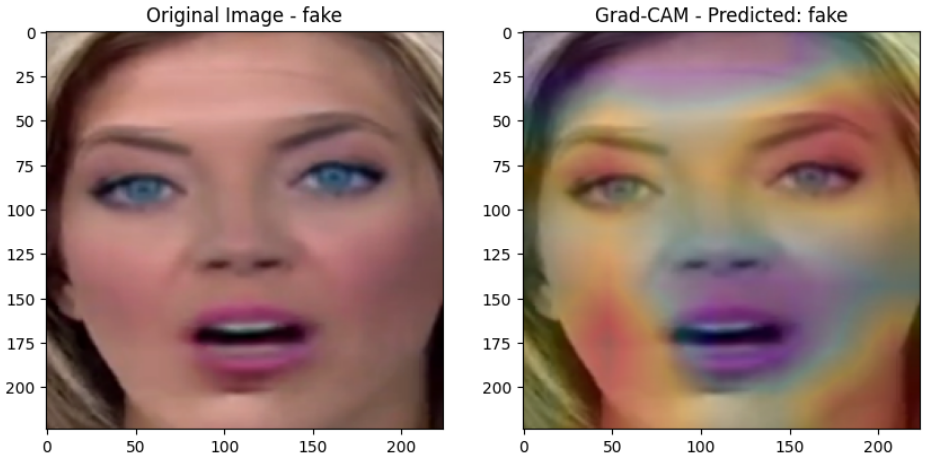


Figure 6: Manipulated image detection using ResNet50 pretrained

As per table 1, the pretrained ResNet50 model had the highest testing accuracy. The target layer for GradCam visualisation was set to the last convolution layer in the model as that is where most of the defining high level features are usually extracted. The visualisation shows that for the original image, key parts of the target face such as eyes and mouth are highlighted as these features likely had a large impact on the model's decision making process. For the manipulated image, the model manages to highlight the parts of the target face that had undergone manipulation and also seems to detect the borders of the manipulation successfully.

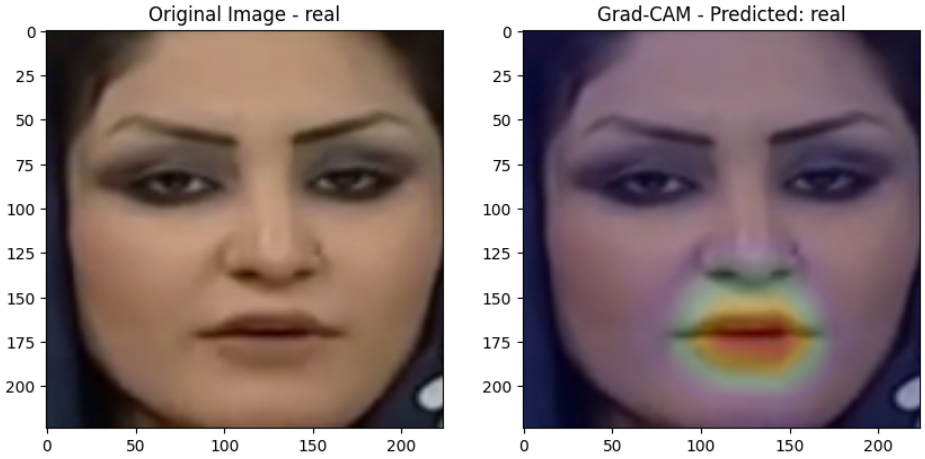


Figure 7: Original image detection using ResNet50 non pretrained

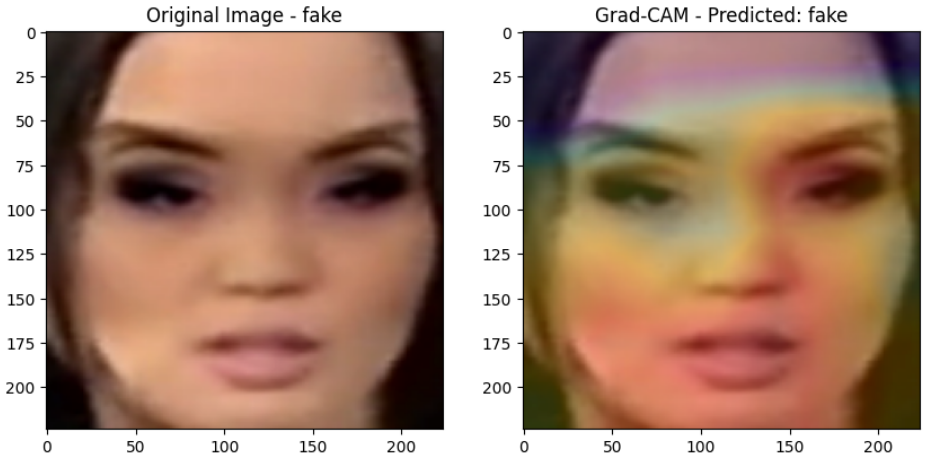


Figure 8: Manipulated image detection using ResNet50 non pretrained

The non trained ResNet50 model had the lowest accuracy as it had to conduct the training from scratch. As the training dataset is not considerably large specially compared to the pretrained models, this inferiority is expected. The model still manages to identify target's mouth as a key feature of the decision making process. For the manipulated image, it manages to highlight the manipulated section of the face along with accurate boundaries. Overall, the explainability of the visualisation seems satisfactory despite the lower accuracy levels.

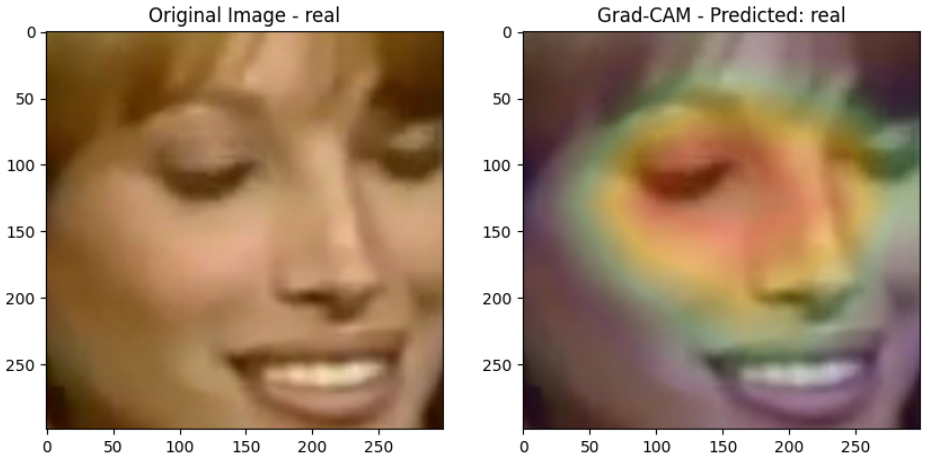


Figure 9: Original image detection using InceptionNet pretrained

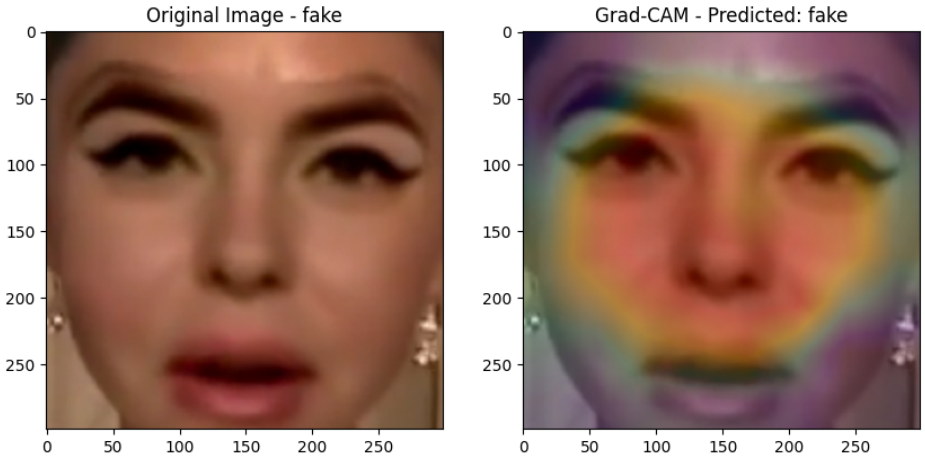


Figure 10: Manipulated image detection using InceptionNet pretrained

The name of the target layer used for Gradcam with InceptionV3 is `Mixed_7c`, it is one of the final layers in the InceptionV3 architechture, this means that it captures high level features that are highly relevant for the classification result. From Table 2, it unsurprisingly beat out untrained ResNet50, but has lower accuracy than pretrained ResNet50. One interpretation of this result can be interpreted with GradCam visualisations. Contrary to ResNet results, that highlighted different regions of the face across images (Mouth highlighted in figure 7, and outer region of face highlighted in 6), InceptionV3 tends to highlight the entire centre region of the face, without much differentiation. This generalised approach may explain the lower performance of InceptionV3 compared to ResNet.

6 Potential Future Extensions and Improvements

Whilst the current model can predict with a relatively high accuracy which images are manipulated, the reasoning behind each prediction decision can sometimes be visually abstract for human viewers. The combination of Vision Transformers (ViTs) and Momentum Constract (MoCo) learning can significantly enhance the clarity and depth of its prediction reasoning. Moreover, this combined approach works to produce a more transparent, interpretable and trustworthy system, as insight can be gained into which parts of the image are most important for the classification task at hand. A potential implementation method is provided below.

1. Use MoCo to pre-train a chosen ViT on a large image datset to learn rich feature representations, involving distance discrimination tasks where the model learns to distinguish between different images.
2. Refine the pre-trained ViT on the deepfake detection dataset.
3. Extract the attention maps from the ViT, so that the patches of the image which are being focused on can be visualised.

4. Apply GradCAM to the attention outputs of the ViT and overlay the produced mappings onto the original images.

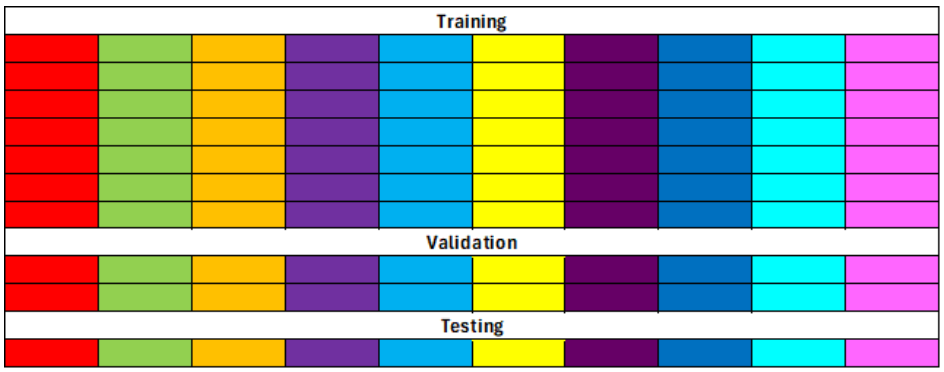


Figure 11: Train, validation and testing split

Figure 11 indicates how the cropped face data is split into 70% training, 20% validation and remaining for test sets. The colours indicate the videos. A potential problem, is that although there are minor changes between different frames of the same video, the images still overall look very similar. Having images in the test set that look similar to data in the training and validation sets may cause over-fitting and produce misleading performance metrics. To address this issue, the videos could firstly be distributed amongst the different datasets, after which, images can be extracted, ensuring model is not trained on testing and validation data.

References

[1] Iván de Paz Centeno. Mtcnn face detection implementation for tensorflow, as a pip package. `https://github.com/ipazc/mtcnn`, 2018. Accessed: 2024-05-24.

[2] Jacob Gildenblat and contributors. Pytorch library for cam methods. `https://github.com/jacobgil/pytorch-grad-cam`, 2021.

[3] Paperspace. Popular deep learning architectures: Resnet, inceptionv3, squeezenet, 2024. Accessed: 2024-05-24.

[4] Andreas Rössler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. FaceForensics++: Learning to detect manipulated facial images. In *International Conference on Computer Vision (ICCV)*, 2019.