Deepfake Image Classification

Andrei Ștefan

Contents

1. Introduction

1.1 Brief overview

The purpose of this project is creating and training an AI model that classifies deepfake images generated by deep generative models. This is a multi-way classification task in which an image must be classified into one of five classes.

The goal is to maximize classification accuracy on the test set.

1.2 Dataset

Name	train	validation	test	train.csv	validation.csv	test.csv
Size	12500	1250	6500	-	-	-
Description	Train image set	Validation image set	Test image set	.csv file containing labels for train images	.csv file containing labels for validation images	.csv file containing only image names

1.3 Approached models

The models implemented and evaluated are: Convolutional Neural Network and K-Nearest Neighbors, with CNN achieving the higher accuracy and being the focus of this paper.

Implementation

For the Convolutional Neural Network, I utilized the **TensorFlow** framework. Early versions used **PyTorch**, but I settled for TensorFlow due to its better logging and monitoring capabilities during model training. Both libraries seemed to perform the same in terms of performance.

For the KNN classifier I decided to use the **scikit-learn** library which provided a reliable implementation of the learning algorithm.

2. Data Augmentation and Preprocessing

2.1 Data Augmentation

To improve the generalization capability of the models, data augmentation was applied to the training images.

In order to reduce the training time for different versions of the model, the augmentation was applied offline, meaning the augmented versions of each training image was generated and stored before training.

This increased the size of the dataset and reduced the risk of overfitting.

2.2 Data Preprocessing

Prior to training, all input images are resized to a resolution of 100x100 pixels.

Pixel values for all three channels (RGB) were normalized to the range [0, 1] by dividing their values by 255.

3. Feature Representation

The two main types of inputs the model uses are the normalized pixel values which are directly fed into the convolutional layers and color histograms which are fed later into Dense layers of the network.

The color histograms are useful for extracting color-based features. The image is first converted into HSV format before computing histograms for each of the three channels using 16 bins per channel. This results in a 48-dimensional color descriptor. The histograms are also normalized to sum to one.

The following histograms show how hue, saturation or value spikes can become clues to what category an image belongs to.

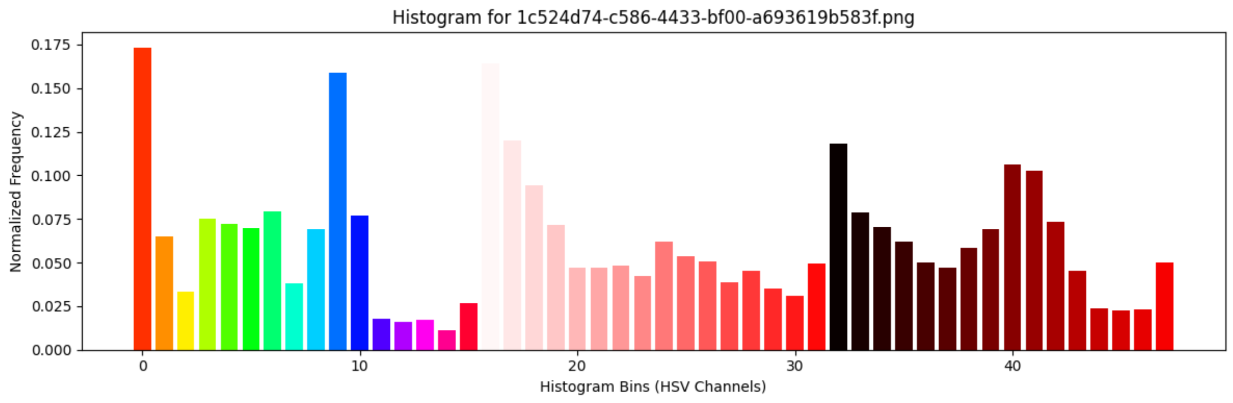
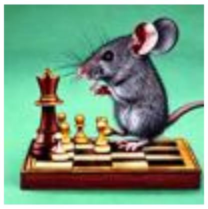

Histogram for 1c524d74-c586-4433-bf00-a693619b583f.png
Normalized Frequency
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0
10
20
30
40
Histogram Bins (HSV Channels)

Figure 1 Histogram



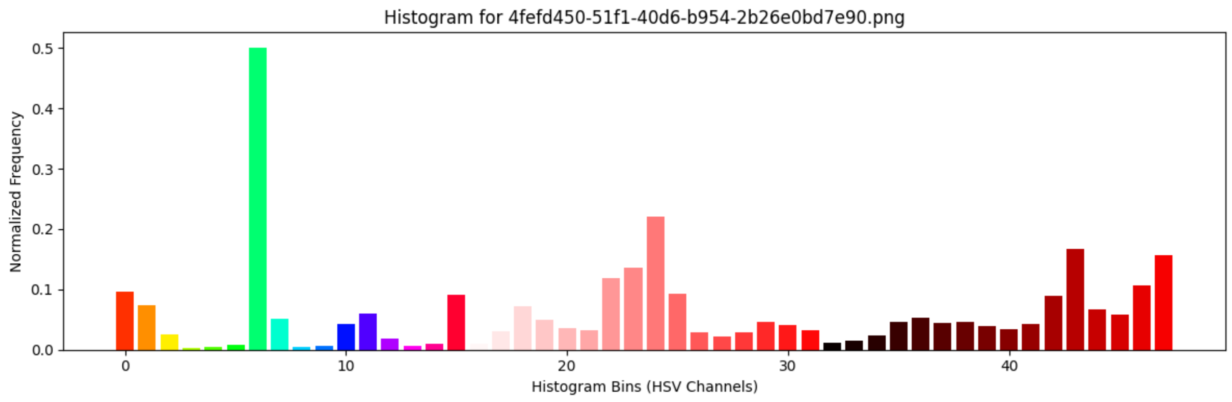
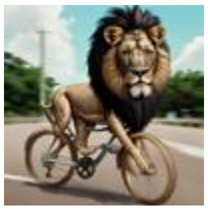

Histogram for 4fefd450-51f1-40d6-b954-2b26e0bd7e90.png
Normalized Frequency
0.0
0.1
0.2
0.3
0.4
0.5
0
10
20
30
40
Histogram Bins (HSV Channels)

Figure 2 Histogram



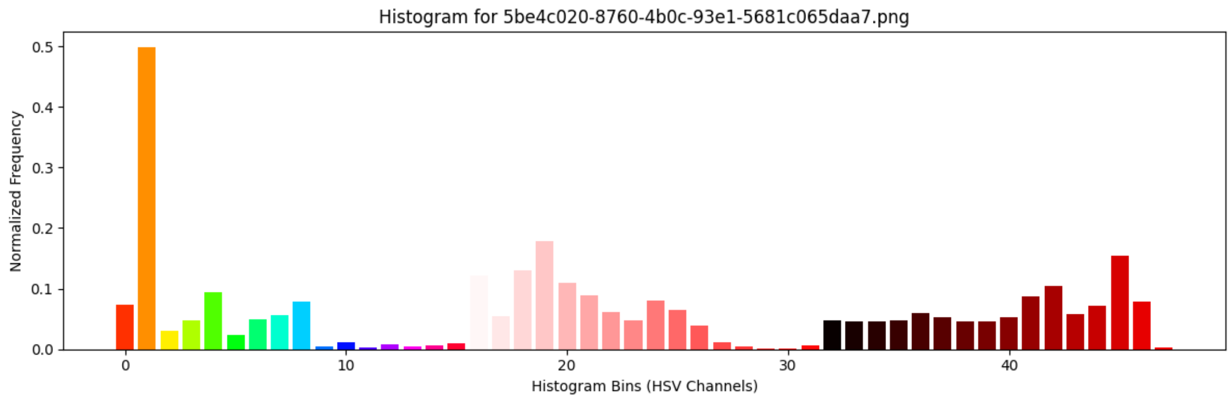

Histogram for 5be4c020-8760-4b0c-93e1-5681c065daa7.png
Normalized Frequency
0.0
0.1
0.2
0.3
0.4
0.5
0
10
20
30
40
Histogram Bins (HSV Channels)

Figure 3 Histogram

4. Machine Learning Models

4.1. K-Nearest Neighbors (KNN)

With KNN having limited performance due to its sensitivity to noise and high-dimensional data, not a lot of effort was put into optimizing such a model.

However, the model managed to obtain an accuracy on the validation set of ~38%. Interestingly, by using only the color histograms described before as inputs for the model, the accuracy on the validation set goes up to 59% for a K value of 11.

The distance metric used is: Manhattan

The value K is chosen by running the model for K = 3, 5, 7, 9, 11 and choosing the best performing one.

4.2. Convolutional Neural Network

4.2.1. Model Architecture

The model accepts two parallel inputs: an RGB image of shape 100x100x3 and a 48-dimensional HSV color histogram vector of the same image. These will be referred to as 'image input' and 'histogram input'.

The image input is processed through 5 convolutional blocks.

Each convolution block starts with a **2D convolution layer** that uses **'same' padding** and the **he_normal kernel initializer**. The padding adds zeros to the border of the input so that the convolution filter can be applied at the edges. This also helps keeping the output feature map the same spatial size as the input.

This is followed by **batch normalization** and a **LeakyReLU** activation function. Batch normalization normalizes the outputs of the previous convolution layer for each batch.

LeakyReLU is a variation of the ReLU activation function which introduces a small, but non-zero gradient for negative input values. This helps prevent "dying ReLU" problem, where neurons can get deactivated by getting stuck and never recovering.

ReLU function: f(x) = {x if x > 0; 0 otherwise}

LeakyReLU function: f(x) = {x if x > 0; αx otherwise} where α is typically a small value (like 0.1)

Max Pooling layers are used after the first three convolutional blocks to downsample the feature maps. A pooling window of 2x2 slides over the input and outputs the maximum value in that region. This effectively halves the feature map size from 100x100 -> 50x50 -> 25x25.

Residual connections between blocks are implemented as follows: The output of the first block is added to the output of the third block. The resulting output (from block 3 concatenated with the output of block 1 through the residual connection) is then also added to the output of the fifth block. This proved to be a useful addition to the model, allowing later layers to potentially reuse features from earlier ones.

Global average pooling is used to downsample the last convolution block, reducing each feature map to its average value.

This output is concatenated with the histogram input which is passed through a **dense layer** with 64 activated neurons.

The combined feature vector (the output of the convolution blocks and the histogram dense layer) is fed into a dense layer with 512 neurons. Batch normalization, LeakyReLU activation and a dropout of 0.5 is applied.

A **softmax output layer** gives the class probabilities over the five target categories.

Optimizer

The optimizer chosen is **AdamW**, a variant of Adam optimizer that instead of applying the weight decay to the learning rate itself, it applies it directly on the parameters after the Adam update.

Loss Function

The loss function is **Sparse Categorical Crossentropy**. It is a good choice for multi-class classification tasks where labels are integer encoded.

$\mathbf{L = -log(p_y)}$ where p_y is the predicted probability of the true class y

Learning Rate Scheduler

ReduceLROnPlateau scheduler was used. It monitors validation accuracy with a patience of 4 and a reducing factor of 0.5. It is combined with an **early stopping** callback that has patience 13.

4.2.2. Hyperparameter tuning and Design Choices

While many optimization attempts were conducted, only the most representative ones will be documented in this section.

4.2.2.1. 8 bins vs 16 bins vs 24 bins Histogram Extraction

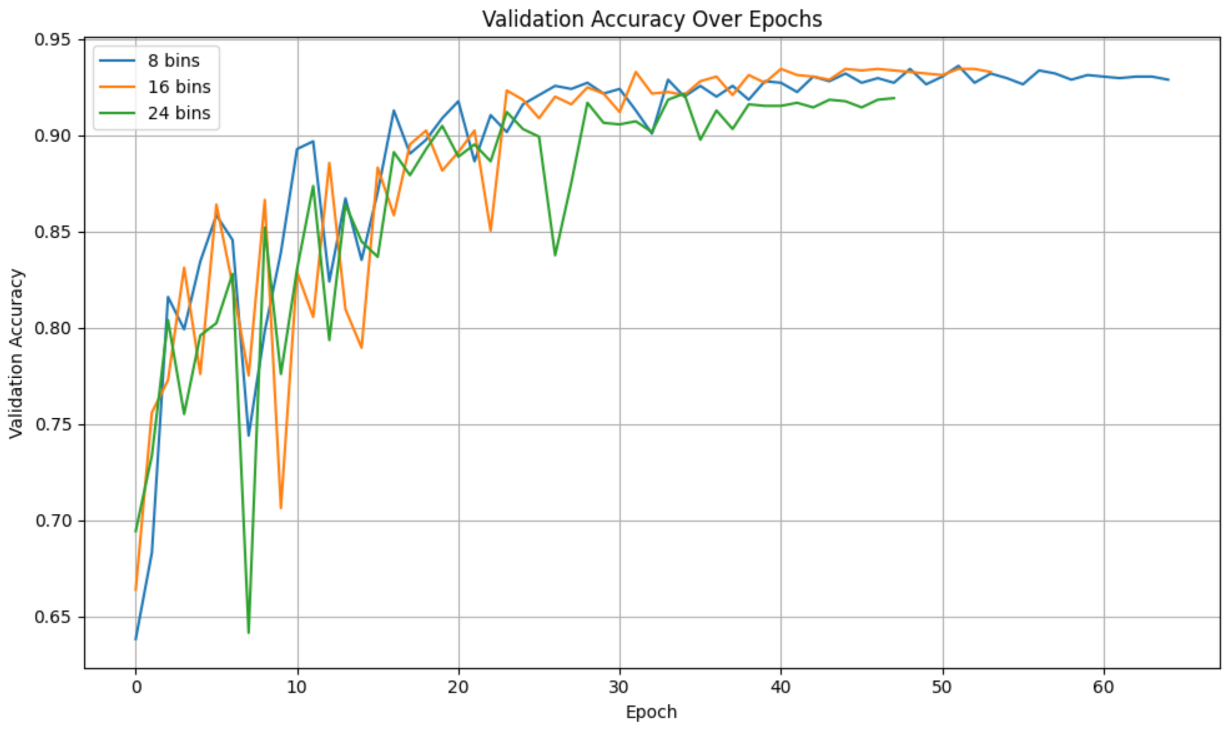


Figure 4 Validation Accuracy - different amount of histogram bins

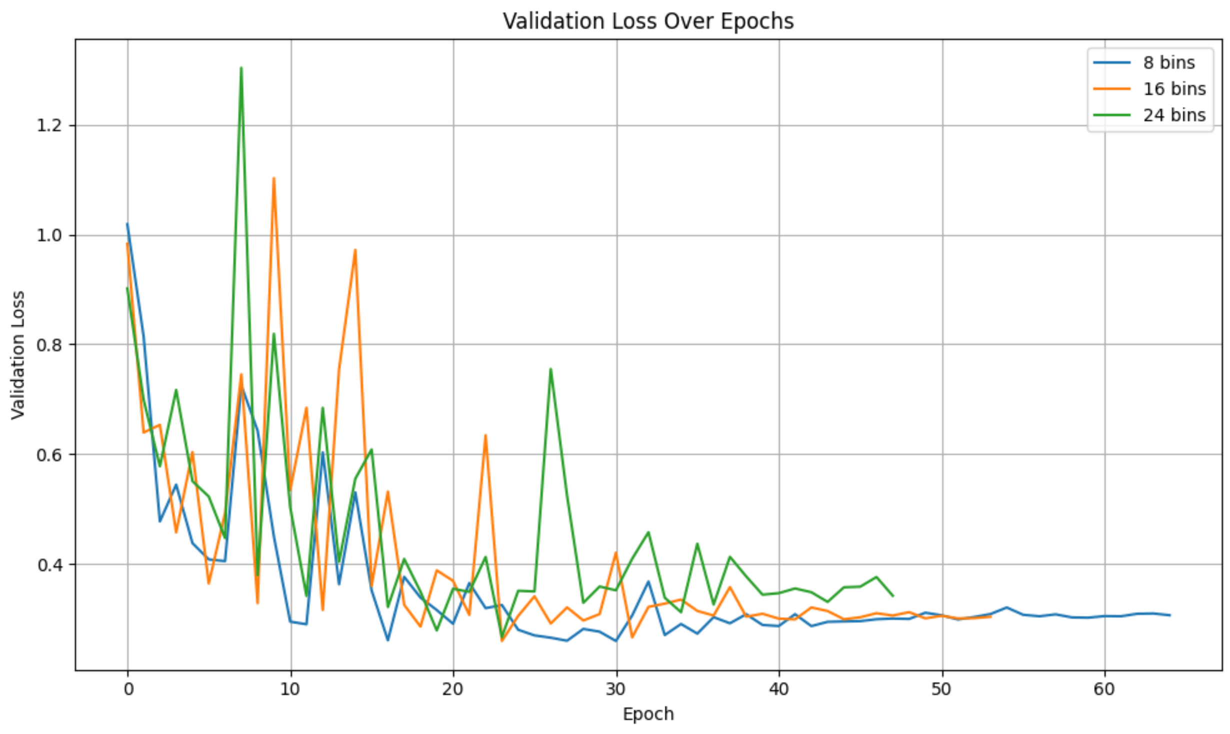


Figure 5 Validation Loss - different amount of histogram bins

Feature extraction through histograms proves helpful to the model. Using histograms with 8 and 16 bins provide good results, but increasing to 24 bins starts harming the validation loss and accuracy, even with a stronger model (with more dense layers).

4.2.2.2. CosineDecay vs ReduceLROnPlateau

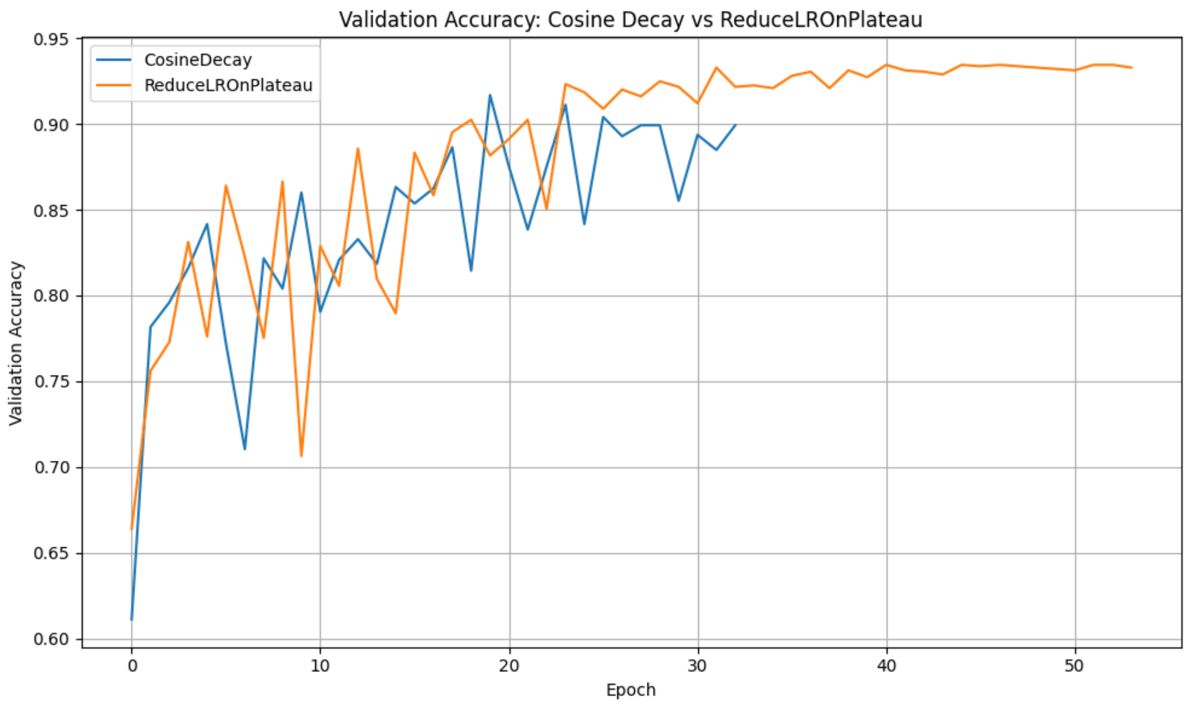


Figure 6 Validation Accuracy - CosineDecay vs ReduceLROnPlateau

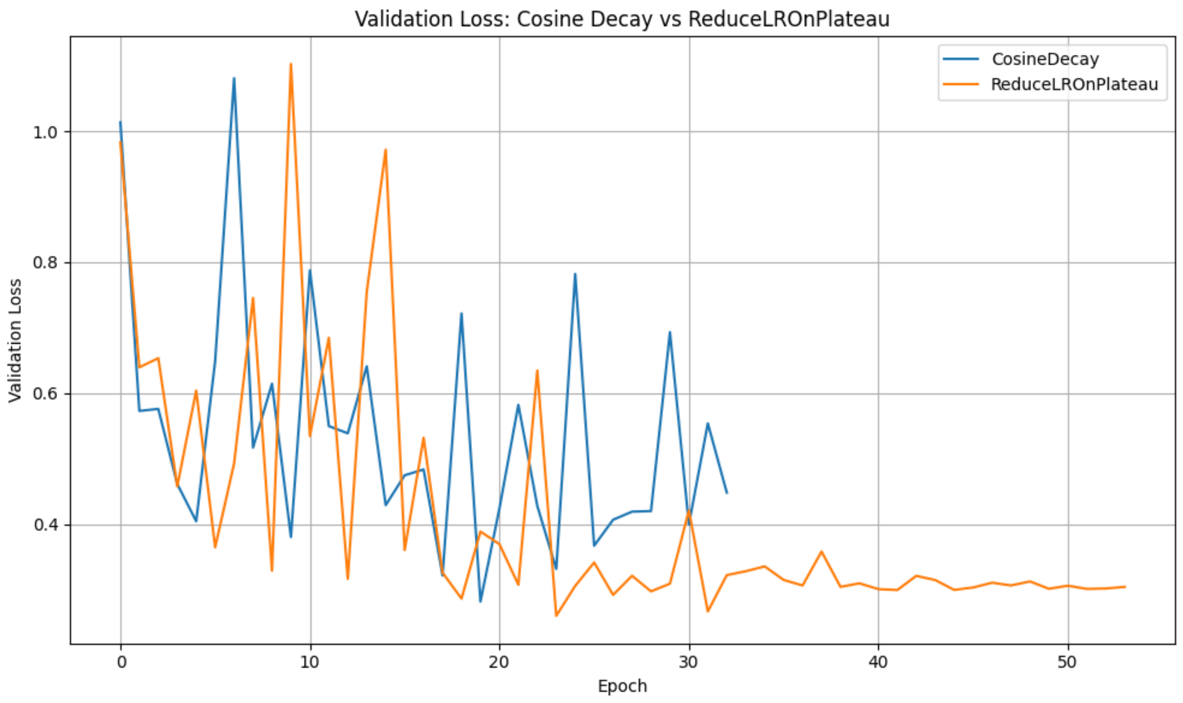


Figure 7 Validation Loss - CosineDecay vs ReduceLROnPlateau

4.2.2.3. 0.3 vs 0.5 dropout on dense layer

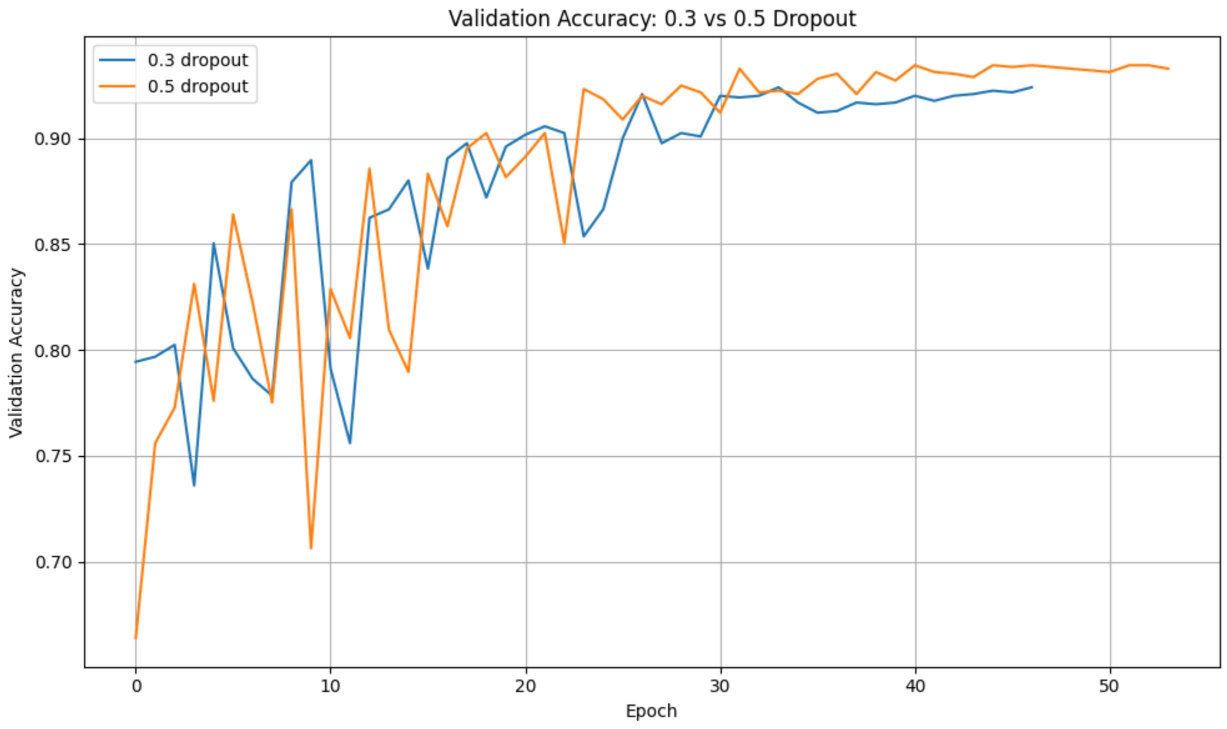


Figure 8 Validation Accuracy - 0.3 vs 0.5 dropout on dense layer

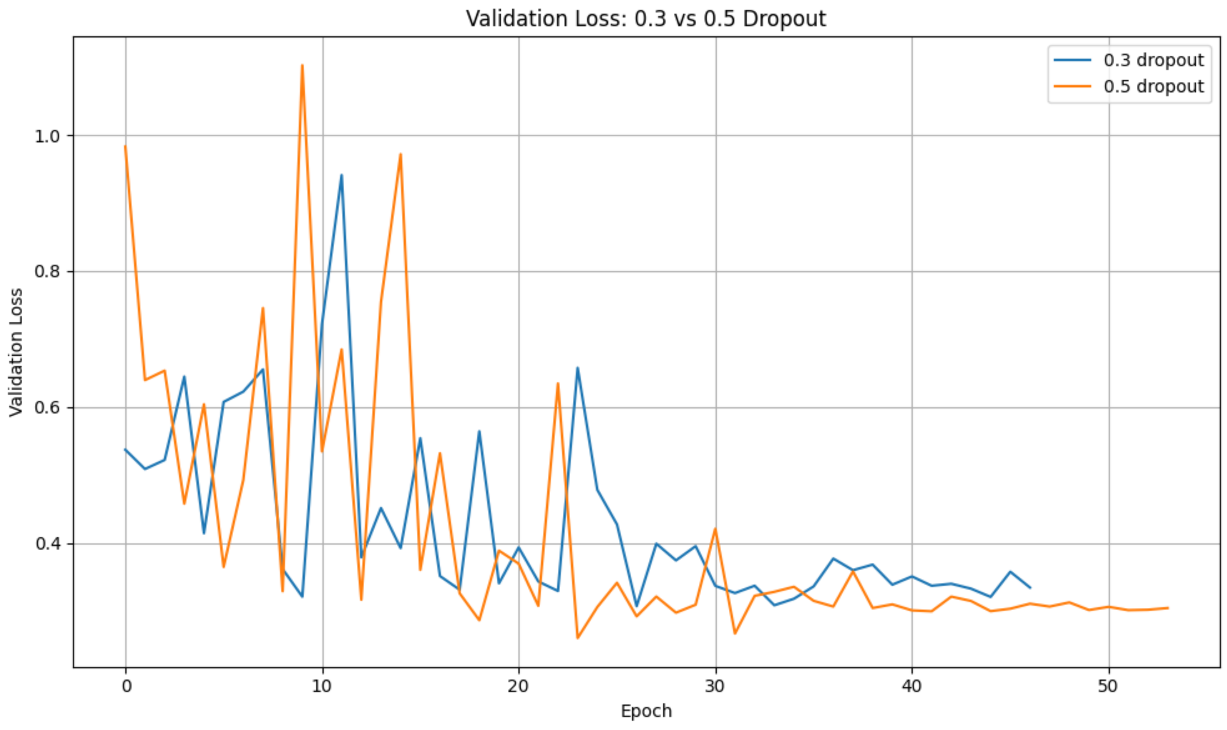


Figure 9 Validation Loss - 0.3 vs 0.5 dropout on dense layer

4.2.2.4. Flatten vs Global Average Pooling

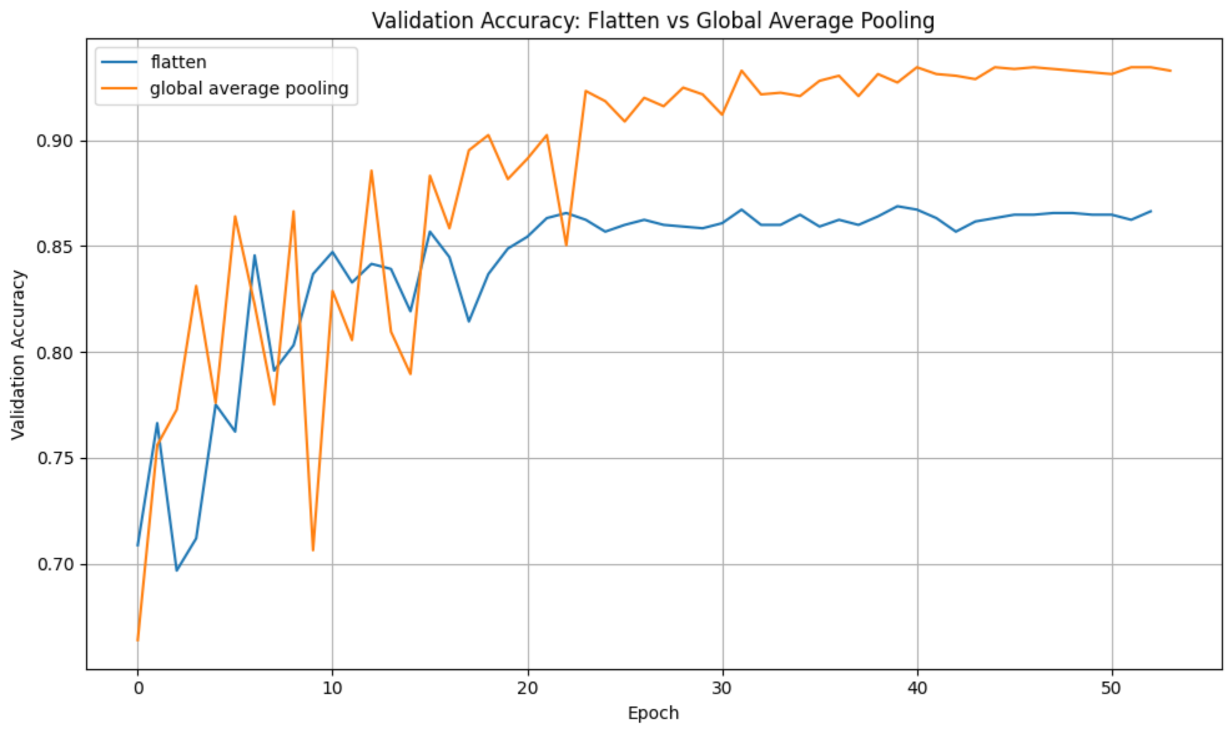


Figure 10 Validation Accuracy - Flatten vs Global Avg Pooling

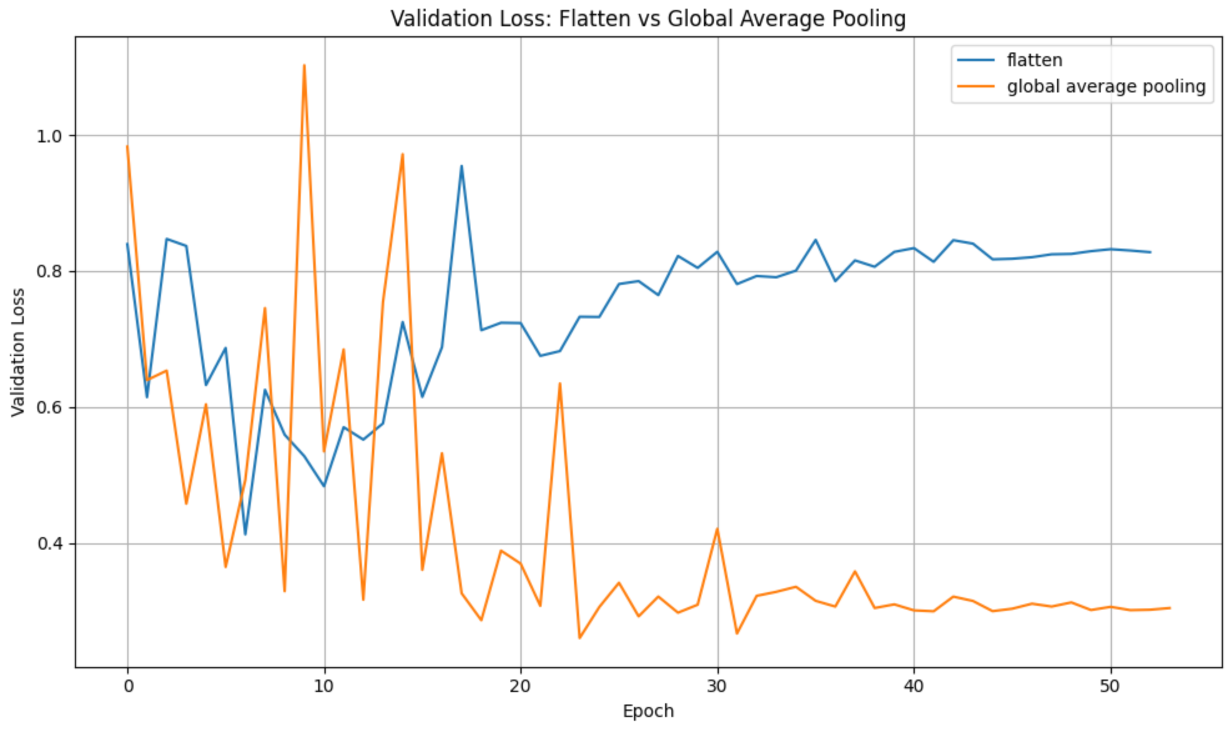


Figure 11 Validation Loss - Flatten vs Global Avg Pooling

5. Confusion matrices

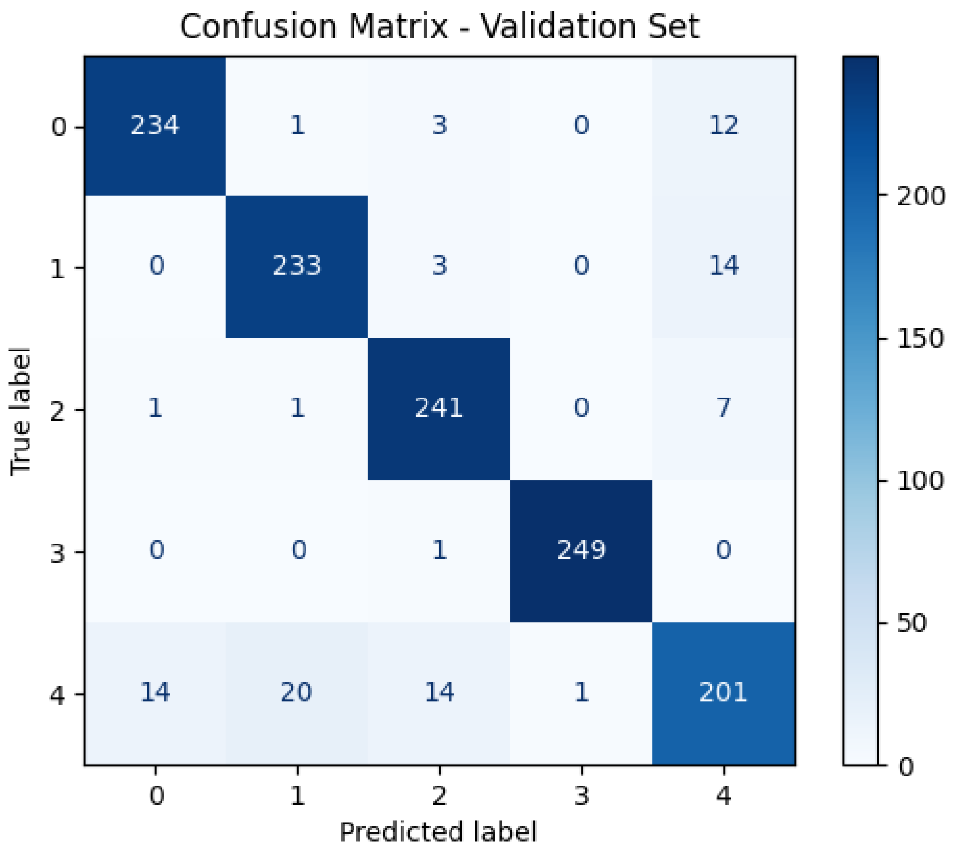


Figure 12 CNN Confusion Matrix

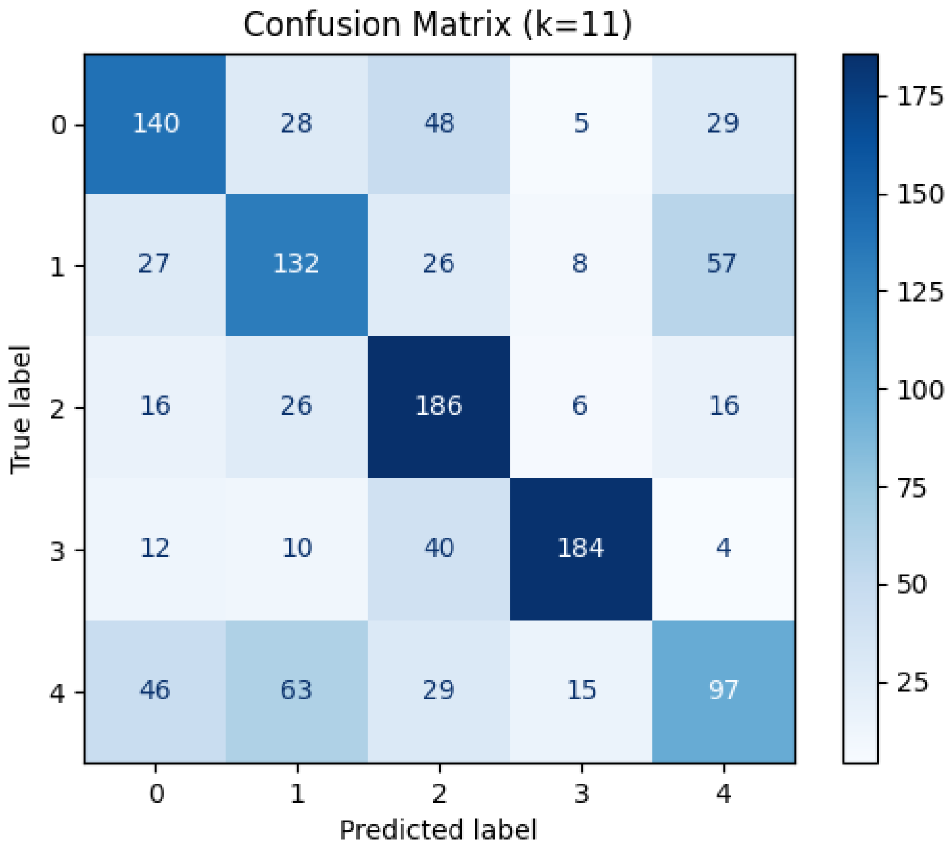


Figure 13 KNN Confusion Matrix

Both models consistently perform well in identifying **class 2 and 3**. Class 2 stands out due to its images containing high color contrasts, vibrant colors and unusual textures. This likely makes the images easier to classify.

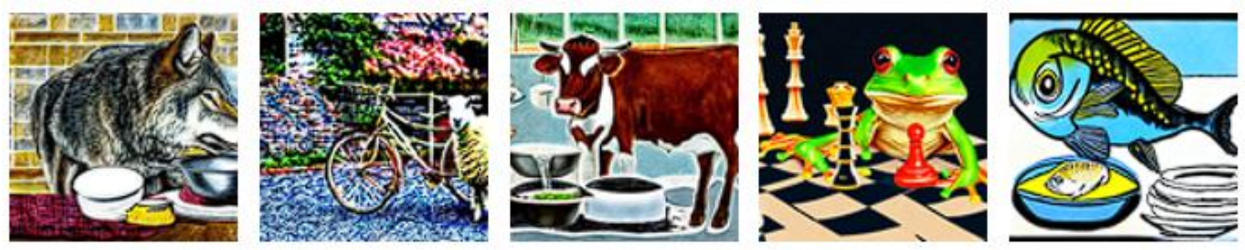


Figure 14 Images from Class 2

Class 4 is the most frequently misclassified by both models. This can attributed to its weaker visual hints, such as more muted colors or less distinct luminosity, and a heavier reliance on contextual or logical reasoning. For example, recognizing the implausibility of a tiger (or any other animal) riding a bicycle or playing chess.

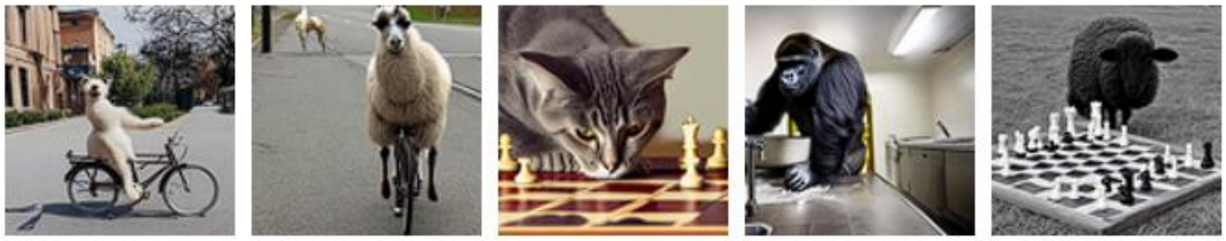


Figure 15 Images from Class 4

6. Conclusions

This works shows a Convolutional Neural Network augmented with color histograms can learn to effectively discriminate between five classes of deepfake generated images. The highest accuracy on the validation set was 93.60%. Compared to the KNN model, the CNN architecture consistently surpassed 80% accuracy, showing that it is able to learn complex features.