Deepfake Classification: Project Report

Țacu Darius-Ștefan
Group: 231

May 27, 2025

1 Introduction

This report details the machine learning approaches used for deepfake image classification. We describe the chosen feature sets, data preprocessing, tested models, hyperparameter tuning, and present results with confusion matrices and performance tables/figures.

2 Data Preprocessing and Feature Representation

2.1 Feature Extraction

I choose to represent image features using color histograms choosing the RGB color space with 256 bins per channel.

Example code for feature extraction:

Example color histogram for figure **??**:

```
import cv2
import numpy as np
def extract_color_histogram(image_path):
    image = cv2.imread(image_path)
    histogram = cv2.calcHist([image], [0, 1, 2], None, [256, 256, 256], [0, 256, 0, 2
    histogram = cv2.normalize(histogram, histogram).flatten()
    return histogram
```

2.2 Dataset Overview

The dataset consists of images divided into training, validation, and test sets. Each image is labeled as one of five classes.

2.3 Preprocessing Steps

- **Resizing:** All images were resized to a fixed dimension.
- **Normalization:** Pixel values were normalized to the [0, 1] range.

Figure 1: Example Color Histogram for an Image

- **Flattening:** Images were flattened into 1D vectors for model input.
- **Augmentation:** (If used, describe augmentations such as flips, rotations, etc.)

3 Machine Learning Models

We tested at least two different models as required:

3.1 K-Nearest Neighbors (K-NN)

- **Description:** K-NN is a non-parametric method that classifies based on the majority label among the k nearest samples.
- **Hyperparameters:** Number of neighbors k, distance metric.
- **Implementation:** Used `sklearn.neighbors.KNeighborsClassifier`.

3.2 Support Vector Machine (SVM)

- **Description:** SVM finds the optimal hyperplane to separate classes in feature space.
- **Hyperparameters:** Regularization parameter C, kernel type.
- **Implementation:** Used `sklearn.svm.SVC`.

4 Hyperparameter Tuning

We performed grid search over key hyperparameters for each model.

4.1 SVM Hyperparameter Tuning

- **Parameter:** C (Regularization)
- **Values tested:** 0.01, 0.1, 1, 10, 100
- **Performance:** See Table **??** and Figure **??**.

Table 1: SVM Validation Scores for Different C Values

C	0.01	0.1	1	10	100
Score	0.5352	0.624	0.7296	0.7136	0.7152

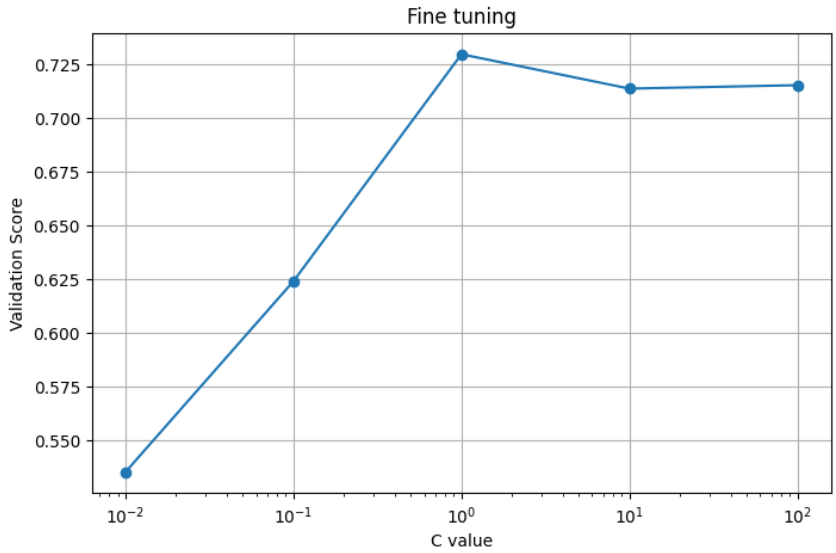

Figure 2: Validation Score vs. C for SVM

4.2 K-NN Hyperparameter Tuning

- **Parameter:** k (Number of neighbors)
- **Values tested:** 1, 3, 5, 7, 9
- **Performance:** (Insert table/figure with results)

5 Results and Evaluation

5.1 Confusion Matrices

We report confusion matrices for the validation set for each model.

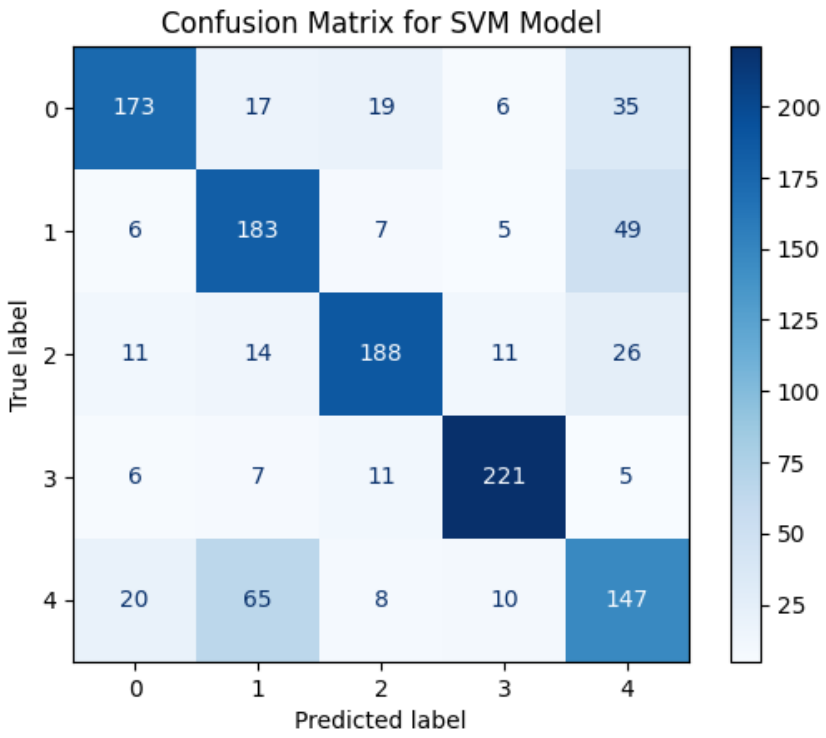

Figure 3: Confusion Matrix for SVM Model

5.2 Performance Comparison

Table 2: Validation Accuracy for Different Models

Model	Best Hyperparameters	Validation Accuracy
SVM	$C = 1$	0.7296
K-NN	$k = 5$	(insert value)

6 Discussion

- Discuss the impact of feature choices and preprocessing.
- Compare model performances and discuss why some approaches worked better.
- Document unsuccessful approaches and lessons learned.

7 Conclusion

Summarize findings, best performing model, and possible future improvements.

Appendix

Sample Python Code with Comments

```
# Load images and labels
train_image_dataset = load_images(train_dir)
# Train SVM model
model = svm.SVC(C=1)
model.fit(train_image_dataset.images, train_image_dataset.labels)
# Evaluate on validation set
score = model.score(validation_image_dataset.images, validation_image_dataset.labels)
print(f"Validation Score: {score}")
```