Deep Learning – Report

Implementation Details

1. Introduction

This project aims to implement and evaluate a Siamese Network for the task of face verification using the Labeled Faces in the Wild (LFW) dataset. The purpose of this project is to provide hands-on experience in building and applying a convolutional neural network (CNN) using PyTorch for facial recognition in a one-shot learning context, guided by the methodology of the Siamese Neural Networks for One-shot Image Recognition paper.

By leveraging the strengths of CNNs in a one-shot learning framework, we aim to push the boundaries of what machines can recognize with minimal input, simulating a more intuitive, human-like way of learning.

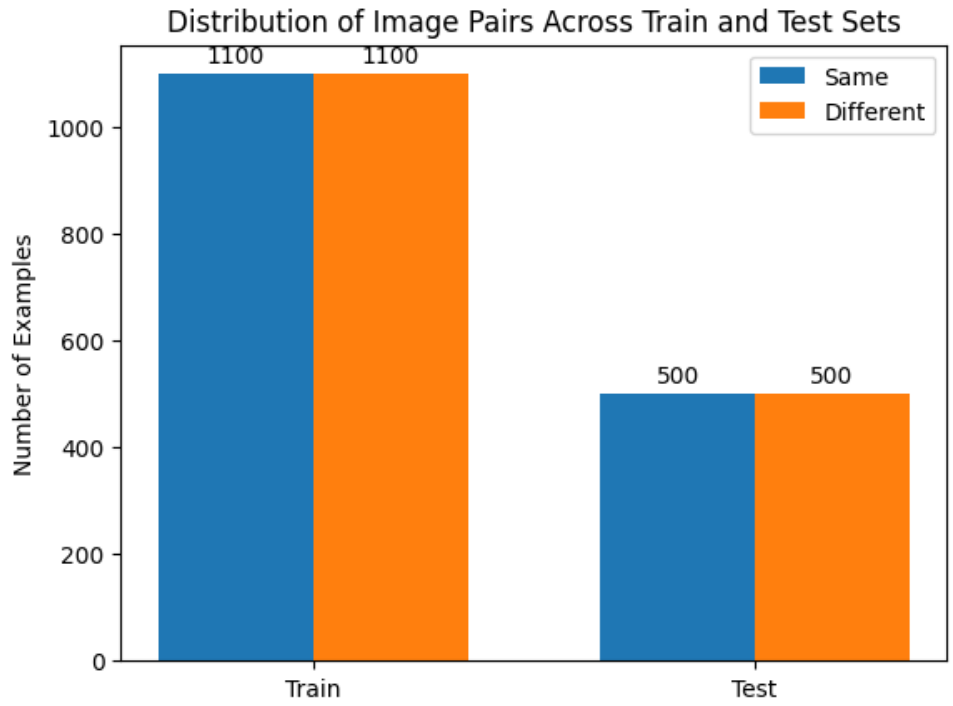
2. Dataset

For training a Siamese Network, the dataset structure is particularly organized into pairs of images:

The dataset includes text files specifying pairs of images for training and testing. These pairs are used to train and test the network on the task of verifying whether two images are of the same person or different people.
Same Person Pairs: These are pairs where both images are of the same person. They are used to teach the network the concept of similarity.
Different Person Pairs: These are pairs where the two images are of different people. They are used to teach the network the concept of dissimilarity.

pairsDevTrain.txt and pairsDevTest.txt: These files list pairs of images along with a label indicating whether the pair is of the same person or different people. The structure of the entries in these files can be either:
Two images of the same person: person_name image_number1 image_number2
Two images of different people: person1_name image_number1 person2_name image_number2.

The distribution of Image Pairs Across Train and Test Sets:



In each dataset, the samples are **evenly** split between 'same' and 'different' categories.
The test set constitutes **31.25%** of the total data.

Summary table:

Dataset	Total Pairs	Same Pairs	Different Pairs	Total Images
Training	2200	1100	1100	4400
Testing	1000	500	500	2000
Total	3200	1600	1600	6400

The original image size is: [3, 250, 250].

In preparing the image data for the neural network model, several transformations were applied across the entire dataset to ensure uniformity and optimize model performance. Initially, all images were resized to a fixed dimension of 105x105 pixels and images were converted to grayscale. This resizing aligns with the input specifications of the model derived from the referenced paper and ensures that all input images maintain a consistent size.

The images underwent normalization using pre-determined mean and standard deviation values specific to single-channel images. Normalization is essential for stabilizing the training process; it ensures that the input features have similar data distributions, which helps in speeding up the learning process and leads to faster convergence. This step adjusts the pixel values so that the dataset features have a mean of 0.485 and a standard deviation of 0.229, values that are generally derived from empirical data and are tailored to adjust the pixel value ranges to be centered around zero, thus enhancing the model's ability to learn efficiently from the dataset.
The transformed image size is: [1, 105, 105].

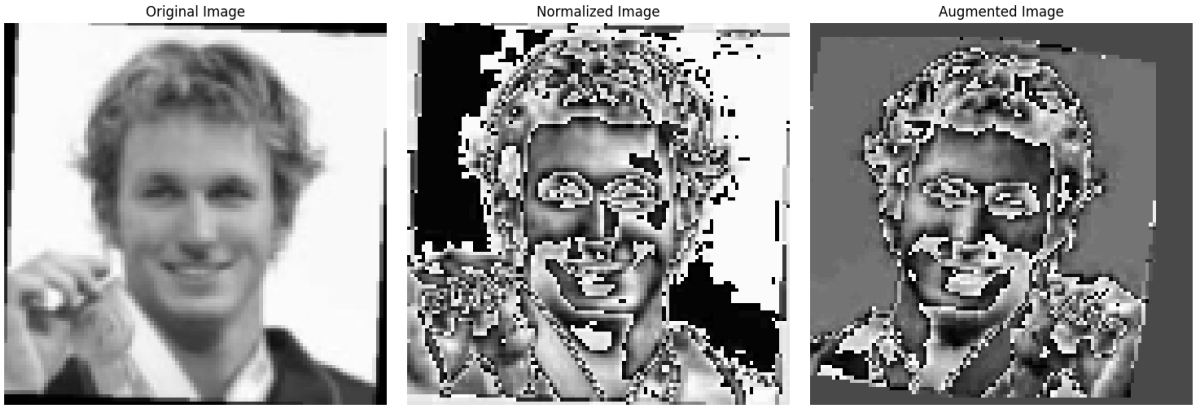
Data Augmentation

In addition to basic preprocessing, our training employs various data augmentation techniques to enhance model robustness and generalization, crucial due to limited training data in one-shot learning.

- Random Horizontal Flip: Images flip horizontally randomly (50% probability), aiding in learning invariant features.
- Random Rotation: Images rotate within +/- 15 degrees, introducing orientation variability.
- Random Affine Transformations: Applies translations and scale adjustments, mimicking object size and position changes.
- Color Jitter: Adjusts brightness and contrast pre-grayscale conversion, making the model less sensitive to lighting variations.

These augmentations occur after resizing and before conversion to tensors, ensuring manipulation without altering standardized inputs. This approach produces a more flexible neural network, capable of handling diverse operational conditions.

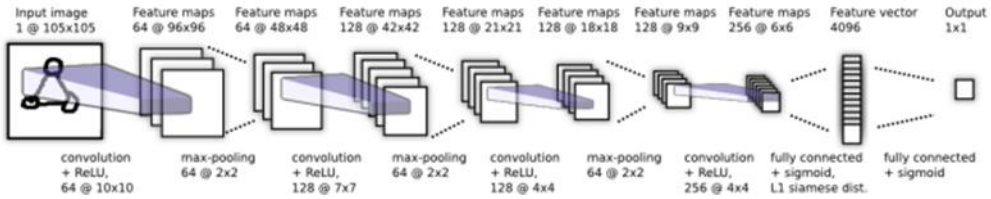
One example of an image in three different modes: the original image, the normalized image, and the augmented image.



3. Network Architecture

Architecture Overview

The model used in this project is a Siamese Network, which consists of two identical sub-networks (referred to as the twin networks). Each sub-network processes one of the input images, and the outputs are then compared to determine if the images belong to the same person. The network architecture based on the paper Siamese Neural Networks for One-shot Image Recognition with some implementations of our own.



Network Layers

The Siamese Network consists of several convolutional layers followed by fully connected layers:

1. Convolutional Layers: These layers extract features from the input images.

    - conv1: 64 filters, kernel size 10, followed by BatchNorm, ReLU activation, and Dropout.
    - conv2: 128 filters, kernel size 7, followed by BatchNorm, ReLU activation, and Dropout.
    - conv3: 128 filters, kernel size 4, followed by BatchNorm, ReLU activation, and Dropout.
    - conv4: 256 filters, kernel size 4, followed by BatchNorm, ReLU activation, and Dropout.

2. Fully Connected Layers: These layers combine the features extracted by the convolutional layers.

    - fc1: 4096 units, followed by ReLU activation and Dropout.
    - out: 1 unit, followed by a Sigmoid activation function to output the final similarity score.

Forward Pass

The Siamese Network processes the two input images one by one through the identical sub-networks. Each image is independently passed through the convolutional and fully connected layers of the sub-network. This results in two feature vectors, one for each image. The absolute difference between the two feature vectors is calculated. This step is crucial as it measures the element-wise distance between the feature representations of the two images. The resulting difference vector is then passed through the final fully connected layer (out) with a Sigmoid activation function to output a similarity score between 0 and 1.
The output similarity score indicates the likelihood that the two input images belong to the same person, with values closer to 1 indicating higher similarity and values closer to 0 indicating lower similarity.

4. Experiments:

The foundation of our experimental setup is to ensure that every stage—from data handling and model configuration to the rigorous validation processes—is meticulously planned and executed. This careful preparation sets the stage for a detailed exploration of hyperparameter optimization, allowing us to refine our model based on empirical evidence gathered through

structured testing and validation. Now, let's delve into the hyperparameter optimization process, where we explore various configurations to identify the most effective model parameters for face verification tasks.

Experiments setup

In our Siamese Network project for face verification, we implement several key components to optimize performance:

- **Weights and biases** are initialized using a truncated normal distribution, with values described in the paper.
- **The loss function** used in this project is Binary Cross Entropy Loss (like in the paper), which is suitable for binary classification tasks. It measures the discrepancy between the predicted similarity score and the actual label (1 for matching pairs, 0 for non-matching pairs).
- **The Adam optimizer** is used, known for its adaptive learning rate capabilities, which helps in managing sparse gradients and ensuring efficient convergence.
- For **regularization** we followed the paper by implementing L2 regularization with 0.01 penalty value.
- **Batch normalization** is incorporated into each convolutional layer to standardize the inputs, accelerating training and improving stability by reducing internal covariate shift.
- **Early stopping** is implemented to halt training if the validation loss sees minimal improvement over 8 epochs (instead of 20 like in the paper), effectively preventing overfitting and ensuring the model generalizes well to new data.
- Training is set for 111 **epochs**, with early stopping after 8 epochs.
- We integrated a learning rate **scheduler** that methodically reduces the learning rate by multiplying it by 0.99 after each epoch, allowing for finer adjustments in the training process as it progresses.

Hyperparameter Optimization

To find the best configuration for the Siamese Network, we perform a grid search over the following hyperparameters:

- **Batch Size**: 16, 32
- **Learning Rate (lr)**: 1e-3, 1e-4
- **Number of Augmentations**: 0(without augmentation), 4, 9
- **Dropout Rate**: 0(without dropout), 0.3

We use 5-fold cross-validation to evaluate each combination of hyperparameters. The performance of each configuration is measured using validation accuracy.

Cross-Validation

The dataset is systematically divided into five folds to facilitate cross-validation. In each iteration, the model is trained on four of these folds and validated on the remaining one. This process is meticulously repeated for each combination of hyperparameters, ensuring a thorough evaluation under varied conditions. The results from each fold are then averaged to ascertain the most effective configuration.

To optimize training effectiveness and prevent data leakage, different transformations are applied to the data depending on its role in the training cycle:

Training Data: During training, we employ a set of data augmentation techniques. The use of these augmentations is strictly confined to the training data of each fold.

Validation and Test Data: For the validation set within each fold and the final test set, only basic transformations are applied. By restricting the validation and test data to only basic transformations, we ensure that the evaluation of the model's performance is both fair and rigorous, avoiding any biases that might be introduced by the varied augmentations applied to the training data.

Retraining with Best Configuration

Once the best configuration is identified based on the hyperparameters with cross-validation results, the model is retrained on the entire training dataset using this configuration. The final model is then tested on the test set to evaluate its performance.

5. Results and Discussion

Cross-Validation Results for Hyperparameters

params	avg_acc	avg_time(s)
{'batch_size': 16, 'lr': 0.001, 'augmentations': 0, 'dropout': 0.0}	0.6718	170.28
{'batch_size': 16, 'lr': 0.001, 'augmentations': 0, 'dropout': 0.3}	0.5005	65.04
{'batch_size': 16, 'lr': 0.001, 'augmentations': 4, 'dropout': 0.0}	0.6836	893.03
{'batch_size': 16, 'lr': 0.001, 'augmentations': 4, 'dropout': 0.3}	0.4882	208.07
{'batch_size': 16, 'lr': 0.001, 'augmentations': 9, 'dropout': 0.0}	0.6423	1556.35
{'batch_size': 16, 'lr': 0.001, 'augmentations': 9, 'dropout': 0.3}	0.4882	419.02
{'batch_size': 16, 'lr': 0.0001, 'augmentations': 0, 'dropout': 0.0}	0.7023	50.06
{'batch_size': 16, 'lr': 0.0001, 'augmentations': 0, 'dropout': 0.3}	0.5886	46.26
{'batch_size': 16, 'lr': 0.0001, 'augmentations': 4, 'dropout': 0.0}	0.7405	524.15
{'batch_size': 16, 'lr': 0.0001, 'augmentations': 4, 'dropout': 0.3}	0.6205	343.47
{'batch_size': 16, 'lr': 0.0001, 'augmentations': 9, 'dropout': 0.0}	0.7377	917.68
{'batch_size': 16, 'lr': 0.0001, 'augmentations': 9, 'dropout': 0.3}	0.6295	771.63
{'batch_size': 32, 'lr': 0.001, 'augmentations': 0, 'dropout': 0.0}	0.6595	120.9
{'batch_size': 32, 'lr': 0.001, 'augmentations': 0, 'dropout': 0.3}	0.5518	85.19
{'batch_size': 32, 'lr': 0.001, 'augmentations': 4, 'dropout': 0.0}	0.6777	1030.55
{'batch_size': 32, 'lr': 0.001, 'augmentations': 4, 'dropout': 0.3}	0.4873	218.93
{'batch_size': 32, 'lr': 0.001, 'augmentations': 9, 'dropout': 0.0}	0.6764	1202.21
{'batch_size': 32, 'lr': 0.001, 'augmentations': 9, 'dropout': 0.3}	0.4873	461.16
{'batch_size': 32, 'lr': 0.0001, 'augmentations': 0, 'dropout': 0.0}	0.7036	52.7
{'batch_size': 32, 'lr': 0.0001, 'augmentations': 0, 'dropout': 0.3}	0.5659	46.21
{'batch_size': 32, 'lr': 0.0001, 'augmentations': 4, 'dropout': 0.0}	0.7327	345.71
{'batch_size': 32, 'lr': 0.0001, 'augmentations': 4, 'dropout': 0.3}	0.61	338.01
{'batch_size': 32, 'lr': 0.0001, 'augmentations': 9, 'dropout': 0.0}	0.7373	588.11
{'batch_size': 32, 'lr': 0.0001, 'augmentations': 9, 'dropout': 0.3}	0.6377	706.89

The results table above displays the average accuracy (avg_acc) and average training time in seconds (avg_time(s)) for a subset of hyperparameter combinations. Each entry in the params

column succinctly represents a unique combination of hyperparameters, providing an overview of the tested scenarios.

Best Configuration

- Parameters: {'batch_size': 16, 'lr': 0.0001, 'augmentations': 4, 'dropout': 0.0}
- Best Average Accuracy: 0.7405
- Training Time for Best Config: 524.15 seconds

This configuration represents the optimal balance of hyperparameters that achieved the highest average accuracy during our tests, indicating effective learning without excessive computational demand.

Discussion of Results

The cross-validation process has illuminated several crucial insights:

- Impact of Augmentations: The level of data augmentation significantly influences the model's accuracy, confirming its role in improving generalization. However, the benefits of augmentation reach a plateau beyond certain levels, suggesting there is an optimal point beyond which additional computation time does not yield proportional gains in performance. Importantly, increased data augmentation also impacts training time, as more complex, numerous transformations and more trainning data require additional processing, which can substantially extend the duration of the training sessions.

- Negative Impact of Dropout: Analysis of the hyperparameter configurations indicates that the inclusion of dropout layers consistently degraded the model's performance across different settings. Configurations with a dropout rate of 0.3 resulted in lower accuracy compared to those with a dropout rate of 0.0, aligning with findings from the original paper. This suggests that dropout, while generally effective at reducing overfitting, may not be beneficial for our specific model architecture and dataset. The model likely requires all available features for optimal performance, and reducing the feature set through dropout hinders its ability to generalize effectively.

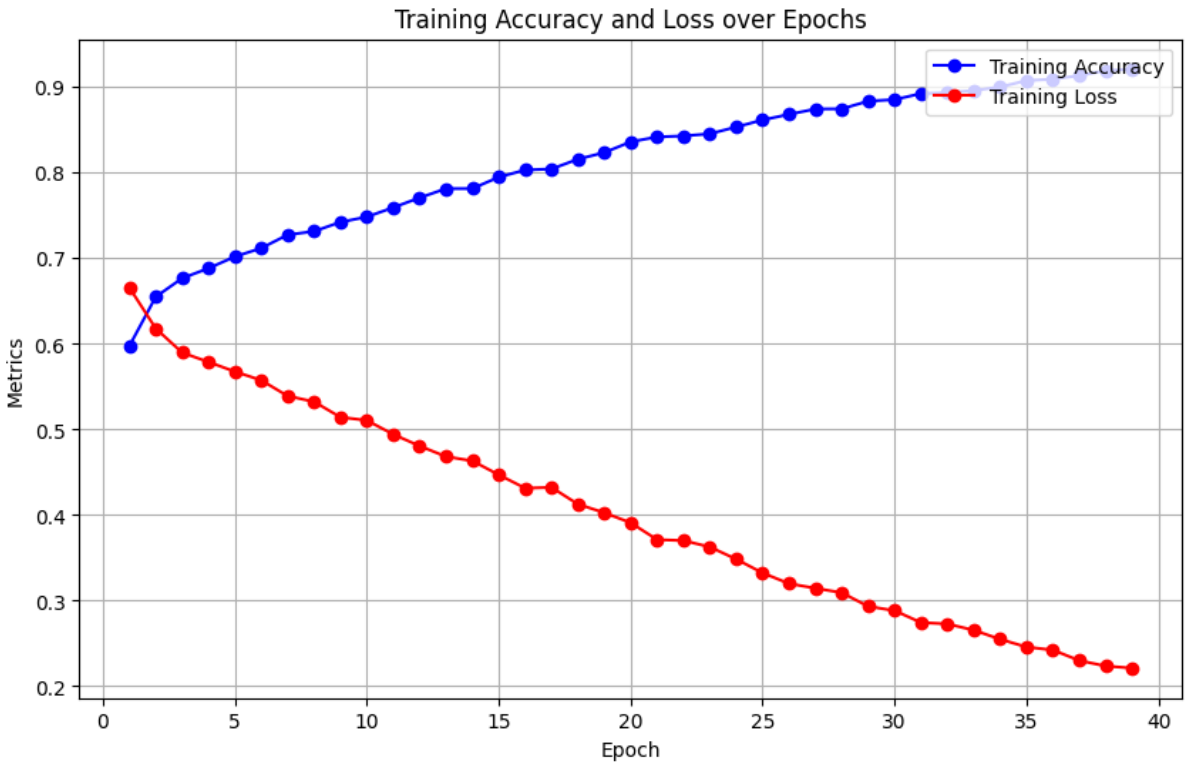
- Influence of Batch Size: Larger batch sizes typically provide a more stable and accurate estimate of the gradient but may result in slower convergence if too large. Smaller batch sizes offer faster convergence and can help escape local minima but might lead to unstable training processes. Our results indicate a balance where neither small nor large batch sizes significantly outperform one another.

- Adjustments in Learning Rate: The learning rate is pivotal in controlling how rapidly a model adjusts its weights. A higher learning rate can lead the model to converge quickly but risks overshooting the minimum, while a too-low learning rate might cause the training process to stall. Our findings suggest that the chosen learning rates achieve a balance where neither extremely high nor low rates significantly outperform one another.

Testing and Model Performance Evaluation

Following the extensive cross-validation process and the identification of the optimal hyperparameter configuration, we proceeded to evaluate the model's performance on a separate test set. The best configuration, which included a **data augmentation factor of 4**, was used to train the model on the entire training dataset. This approach effectively increased the size of the training set fivefold.

Training Process: The model was trained for **39 epochs**, a duration determined to be optimal based on insights from the cross-validation phase. This specific number of epochs balanced the need for sufficient model training to converge on the best features, without overfitting, which is crucial given the expanded training dataset size through augmentation.



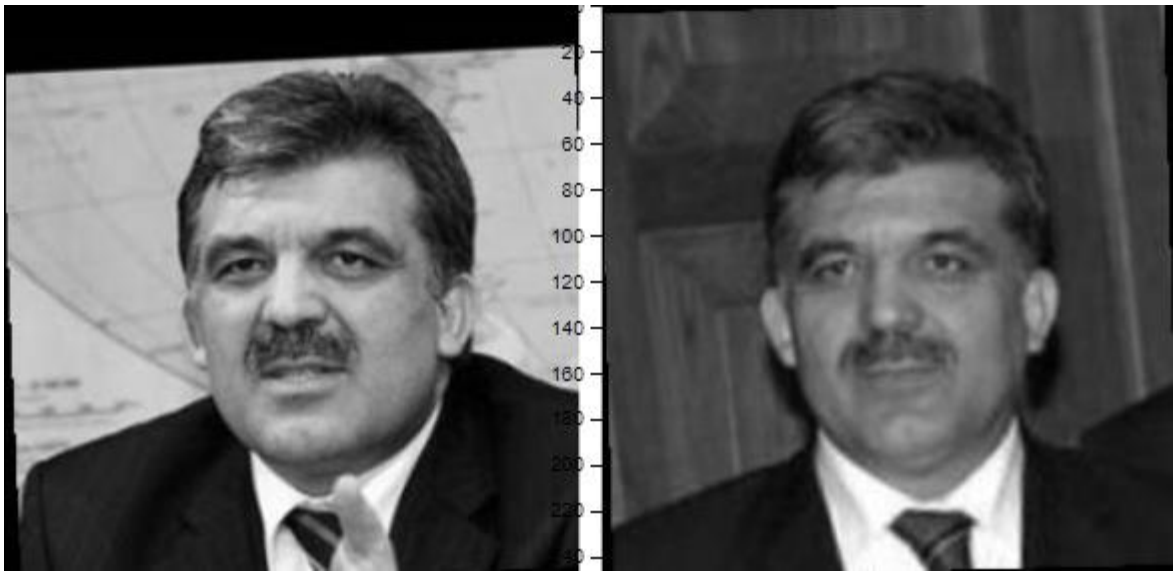
The plot below illustrates the evolution of training accuracy and loss over 39 epochs for the model trained using the optimal hyperparameters identified through our rigorous cross-validation process. As depicted, training accuracy consistently improves, indicating effective learning and adaptation by the model to the features of the dataset. Concurrently, the training loss decreases significantly, which is indicative of the model's increasing proficiency in making accurate predictions as training progresses. These trends are essential for evaluating the model's learning efficiency and ensuring that it is not overfitting or underfitting, as evidenced by the steady improvement in both metrics throughout the training period.

Model Testing and Final Results: The final test accuracy achieved was **0.784**, indicating a robust generalization capability when applied to unseen data. This result validates the effectiveness of the chosen hyperparameters, and the augmentation strategy employed during training.

Model Performance Analysis:

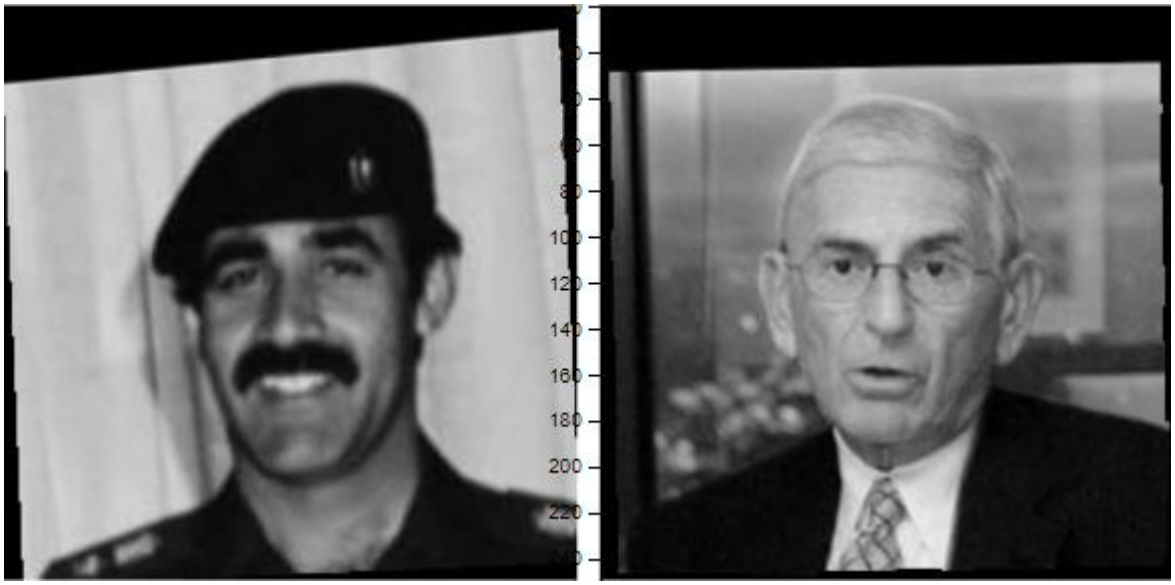
In this task, we delve into the model's performance by examining specific instances of both successful classifications and misclassifications. This analysis helps to identify patterns or characteristics that may influence the model's accuracy, providing insights into the model's practical functioning in real-world scenarios.

True Positive Prediction



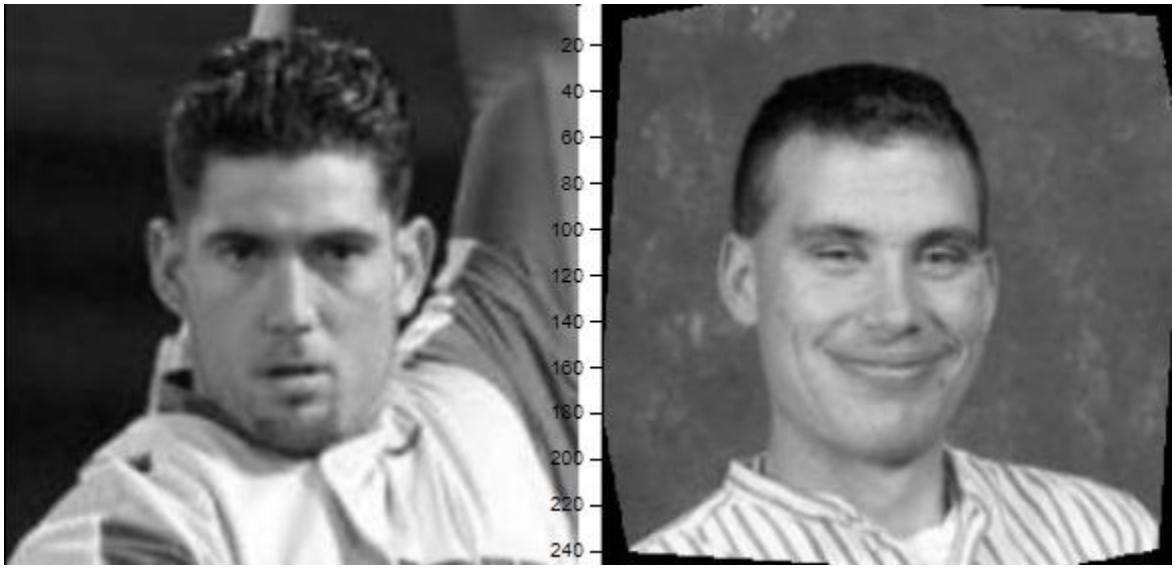
The displayed image pair shows the same individual in two different settings. Both photographs present the person with a similar pose and facial expression, which are key features that the model has been trained to recognize.

True Negative Prediction



The image pair displayed two different individuals. The model correctly identified these images as not matching, demonstrating its ability to discern between distinct individuals even under varying conditions.

False Positive Prediction



In this case, the model incorrectly identified two different individuals as the same person. This misclassification highlights challenges in distinguishing between similar-looking individuals, a common issue in facial recognition systems.

False Negative Prediction

This example illustrates a false negative where the model failed to identify the same individual across two images due to a significant obstruction—the presence of a tennis racket partially covering the face in one of the photos.

6. Conclusion

The comprehensive evaluation through cross-validation and the subsequent testing phase have validated the efficacy of our selected training approach. Utilizing a moderate level of data augmentation and an optimally determined number of training epochs. The training strategy, informed by the nuanced insights gained from our experimentation with various hyperparameters, ensured that the model could leverage the augmented data effectively without succumbing to overfitting. The high level of accuracy achieved on the test set not only attests to the model's robust performance in controlled settings but also underscores its capability to generalize well to real-world conditions.

The detailed examination of the examples from the model performance analysis provides valuable insights into the strengths and weaknesses of our model. By understanding where and why the model fails or succeeds, we can better guide our efforts to improve its accuracy and reliability in future iterations. Enhanced training strategies, expanded datasets, or adjusted model architectures may be necessary to address the identified shortcomings.

Challenges

Throughout this project, we encountered several challenges that provided learning opportunities and directions for further improvement:

- Model Overfitting: Initial iterations showed a tendency towards overfitting, with the model performing exceptionally well on the training data but less so on unseen data.

- Hyperparameter Tuning: Determining the optimal settings for hyperparameters such as dropout rate and learning rate required extensive testing and validation, highlighting the delicate balance necessary for optimal model training.

Future Work

Building on the insights gained, several areas present promising avenues for future research and development:

- Advanced Data Augmentation Techniques: Exploring more sophisticated data augmentation methods could further enhance the model’s ability to generalize from training to real-world scenarios.

- Model Performance Analysis: Leveraging detailed analysis of model performance and extracting some insights could illuminate strategies for refining algorithmic approaches, enhancing efficiency, and improving outcomes across various metrics.