206477952
207547183

Assignment 2 - Facial Recognition using one-shot learning

Project Objective:
The goal of this project is to develop a Convolutional Neural Network (CNN) for facial feature recognition. The model is designed to take two images of different individuals and predict whether they depict the same person.

To achieve this, we divided our training data into pairs: pairs of similar images and pairs of dissimilar images, enabling the model to learn how to distinguish between the two cases.

We utilized the "Labeled Faces in the Wild" (LFW) dataset, which contains labeled images for learning purposes. Additionally, we used two text files, "pairsDevTest" and "pairsDevTrain," to split the dataset into test and train sets.

After analyzing the data, we obtained the following results:

```
Total number of samples: 3200
Number of samples in the Train Set: 2200
Number of same people in the Train Set: 1100
Number of different people in the Train Set: 1100
Number of samples in the Test Set: 1000
Number of same people in the Test Set: 500
Number of different people in the Test Set: 500
```

We can observe that half of the images in both files (training and test sets) belong to pairs of similar images, while the other half belong to pairs of dissimilar images.

Dataset Preparation:
After analyzing the structure of the dataset, we created image paths and assigned labels to each pair:
Label = 1 for identical image pairs (same person).
Label = 0 for non-identical image pairs (different people).
The labeling process involved the following:
Checking if a line in the pairsDevTrain or pairsDevTest file contains 3 entries (indicating a pair of similar images of the same person) or 4 entries (indicating a pair of dissimilar images of two different people).
Based on the number of entries, the appropriate image paths were constructed, and the label was assigned.

Pre-processing:
The images were resized to 105x105 pixels, a standard dimension used for training Siamese Networks, as described in the article by Koch et al. They were then transformed into tensors to ensure compatibility with the model. Additionally, invalid or corrupted images were excluded during the pre-processing stage.

We applied this process to the dataset along with the labels. Specifically, we ran the preprocess_image function on each image pair and retained only valid pairs (where both images were successfully processed).

206477952
207547183

Dataset Splitting:

Following preprocessing, the data was split into:
Training Set (80%): Used to train the model and learn facial features.
Validation Set (20%): Used for monitoring the model's performance during training.
Test Set: Held out for final evaluation of the trained model.

```
Number of training samples: 1760
Number of validation samples: 440
Number of test samples: 1000
```

Model Implementation:

The first model we ran (this is not the final one)

```python
class SiameseNetwork(nn.Module):
    def __init__(self):
        super(SiameseNetwork, self).__init__()
        self.cnn = nn.Sequential(
            nn.Conv2d(1, 64, kernel_size=10, stride=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2),  # Output: 64 @ 48x48

            nn.Conv2d(64, 128, kernel_size=7, stride=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2),  # Output: 128 @ 21x21

            nn.Conv2d(128, 128, kernel_size=4, stride=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2),  # Output: 128 @ 9x9

            nn.Conv2d(128, 256, kernel_size=4, stride=1),
            nn.ReLU(inplace=True),                  # Output: 256 @ 6x6
        )

        self.fc1 = nn.Linear(256 * 6 * 6, 4096)  # Fully connected layer: 4096
        self.fc2 = nn.Sequential(
            nn.Linear(4096, 1),
            nn.Sigmoid()
        )

    def flatten_one(self, input):
        """
        Flattens the input tensor into (batch_size, -1).
        """
        return input.view(input.size(0), -1)

    def forward(self, one, two):
        # Forward pass through CNN
        output1 = self.cnn(one)
        output2 = self.cnn(two)

        # Flatten the outputs
        output1 = self.flatten_one(output1)
        output2 = self.flatten_one(output2)

        # Pass through the first fully connected layer
        output1 = self.fc1(output1)
        output2 = self.fc1(output2)

        # Compute absolute difference and pass through final FC layer
        similarity = self.fc2(torch.abs(output1 - output2))
        return similarity
```

The network we created aligns well with the architecture and methodology described in the paper.

1. Convolutional Layers:

The CNN component of the model includes convolutional layers with increasing depth (256 ,128 ,64).

Each convolutional layer is followed by ReLU activations and max-pooling layers, which reduce the dimensions of the feature maps.

This approach, consistent with the paper, allows the model to learn hierarchical feature representations while reducing computational complexity.

2. Flattening Operation:

After extracting feature maps through the convolutional layers, the flatten_one function reshapes the feature maps into a flat vector of shape (batch_size, -1).

This operation is crucial for connecting the convolutional layers to the fully connected layers, which operate on 1D feature vectors.

3. Fully Connected Layers:

The fully connected layers transform the flattened CNN output into a 4096-dimensional feature vector, representing the learned high-level information about the images.

The final fully connected layer computes the similarity score by comparing the absolute difference between the feature vectors of the two input images, as described in the paper.

4. Similarity Calculation:

The absolute difference between the two feature vectors is computed and passed through a sigmoid activation function, which produces a probability score indicating whether the two images are of the same person.

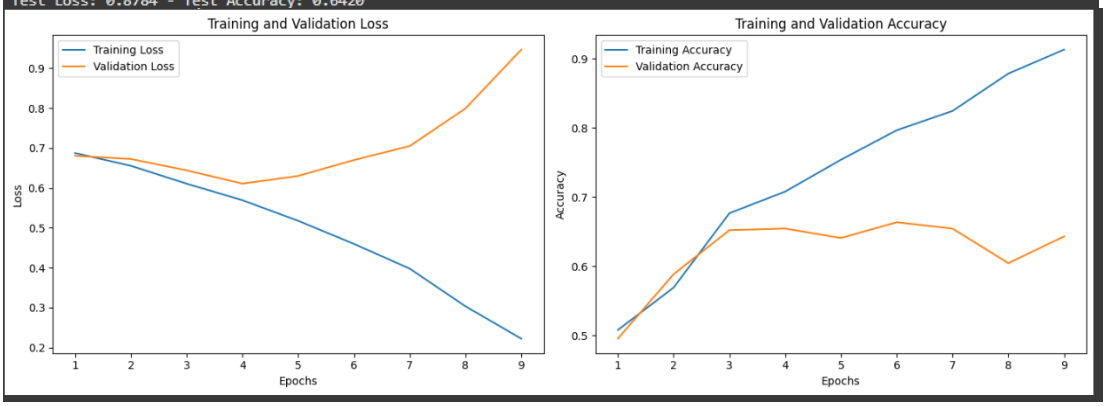
- A simple model without batch normalization or dropout was evaluated.
- Learning Rate: The paper specifies that the learning rate should range between 0 and 0.01. We adhered to this range during our experiments, specifically setting the learning rate to 0.0001, ensuring stable convergence during training.
  To optimize the training process, we utilized a Step Learning Rate Scheduler, which reduces the learning rate by a factor of 0.1 every 10 epochs. By dynamically adjusting the learning rate, the scheduler ensures that the model learns effectively throughout the training process.
- Loss Function: Binary Cross Entropy Loss (BCELoss) was used to measure the error in binary classification (same or different pairs)
- Batch size = 128 (like the article) - The first set of experiments started with a batch size of 128.
- number of epochs = 20.
- Early Stopping: Early stopping was employed to prevent overfitting and save computational resources. The training process monitored validation loss after each epoch, saving the model whenever the validation loss improved. If the validation loss did not improve for 5 consecutive epochs (patience=5), the training was halted, ensuring that the best-performing model was preserved for final evaluation.

206477952
207547183

Runs and Experiments:

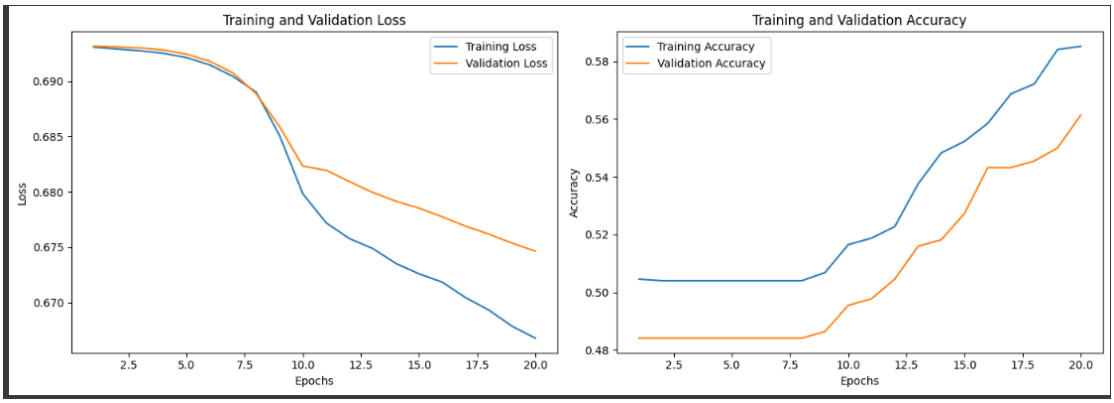
The first test was conducted using the ADAM optimization algorithm with the parameters mentioned above.

```
Running Experiment with batch_size=128, learning_rate=0.0001
Epoch 1/20 - training_loss: 0.6875 - training_accuracy: 0.5080 - validation_loss: 0.6806 - validation_accuracy: 0.4955
Epoch 2/20 - training_loss: 0.6556 - training_accuracy: 0.5693 - validation_loss: 0.6727 - validation_accuracy: 0.5886
Epoch 3/20 - training_loss: 0.6109 - training_accuracy: 0.6767 - validation_loss: 0.6443 - validation_accuracy: 0.6523
Epoch 4/20 - training_loss: 0.5694 - training_accuracy: 0.7080 - validation_loss: 0.6110 - validation_accuracy: 0.6545
Epoch 5/20 - training_loss: 0.5178 - training_accuracy: 0.7540 - validation_loss: 0.6302 - validation_accuracy: 0.6409
Epoch 6/20 - training_loss: 0.4600 - training_accuracy: 0.7966 - validation_loss: 0.6701 - validation_accuracy: 0.6636
Epoch 7/20 - training_loss: 0.3979 - training_accuracy: 0.8244 - validation_loss: 0.7051 - validation_accuracy: 0.6545
Epoch 8/20 - training_loss: 0.3035 - training_accuracy: 0.8784 - validation_loss: 0.7994 - validation_accuracy: 0.6045
Epoch 9/20 - training_loss: 0.2225 - training_accuracy: 0.9131 - validation_loss: 0.9467 - validation_accuracy: 0.6432
Early stopping triggered!
Experiment completed in 37.75 seconds
Evaluating on Test Data...
Test Loss: 0.8784 - Test Accuracy: 0.6420
```



The second test was conducted using the SGD optimization algorithm.

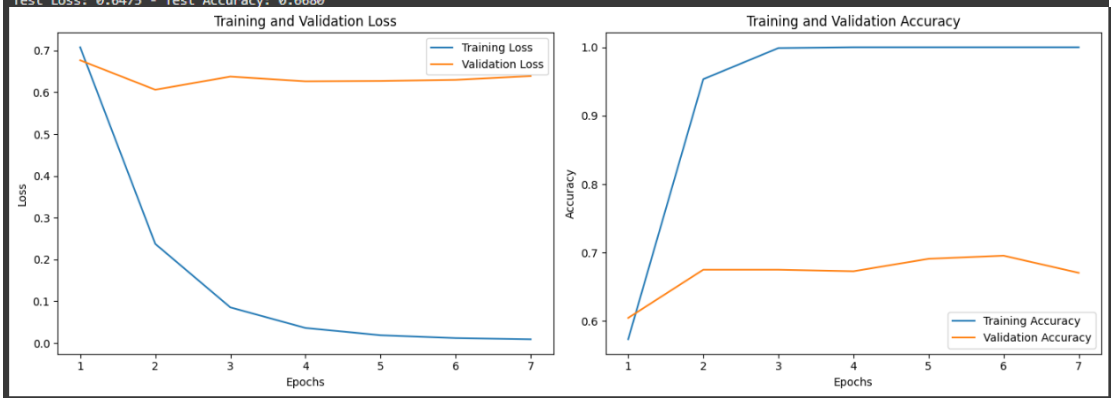
```
Running Experiment with SGD and batch_size=128
Epoch 1/20 - SGD - training_loss: 0.6931 - training_accuracy: 0.5045 - validation_loss: 0.6931 - validation_accuracy: 0.4841
Epoch 2/20 - SGD - training_loss: 0.6929 - training_accuracy: 0.5040 - validation_loss: 0.6931 - validation_accuracy: 0.4841
Epoch 3/20 - SGD - training_loss: 0.6927 - training_accuracy: 0.5040 - validation_loss: 0.6930 - validation_accuracy: 0.4841
Epoch 4/20 - SGD - training_loss: 0.6925 - training_accuracy: 0.5040 - validation_loss: 0.6928 - validation_accuracy: 0.4841
Epoch 5/20 - SGD - training_loss: 0.6921 - training_accuracy: 0.5040 - validation_loss: 0.6924 - validation_accuracy: 0.4841
Epoch 6/20 - SGD - training_loss: 0.6915 - training_accuracy: 0.5040 - validation_loss: 0.6918 - validation_accuracy: 0.4841
Epoch 7/20 - SGD - training_loss: 0.6904 - training_accuracy: 0.5040 - validation_loss: 0.6907 - validation_accuracy: 0.4841
Epoch 8/20 - SGD - training_loss: 0.6890 - training_accuracy: 0.5040 - validation_loss: 0.6888 - validation_accuracy: 0.4841
Epoch 9/20 - SGD - training_loss: 0.6851 - training_accuracy: 0.5068 - validation_loss: 0.6859 - validation_accuracy: 0.4864
Epoch 10/20 - SGD - training_loss: 0.6798 - training_accuracy: 0.5165 - validation_loss: 0.6823 - validation_accuracy: 0.4955
Epoch 11/20 - SGD - training_loss: 0.6772 - training_accuracy: 0.5188 - validation_loss: 0.6820 - validation_accuracy: 0.4977
Epoch 12/20 - SGD - training_loss: 0.6758 - training_accuracy: 0.5227 - validation_loss: 0.6809 - validation_accuracy: 0.5045
Epoch 13/20 - SGD - training_loss: 0.6749 - training_accuracy: 0.5375 - validation_loss: 0.6800 - validation_accuracy: 0.5159
Epoch 14/20 - SGD - training_loss: 0.6735 - training_accuracy: 0.5483 - validation_loss: 0.6792 - validation_accuracy: 0.5182
Epoch 15/20 - SGD - training_loss: 0.6726 - training_accuracy: 0.5523 - validation_loss: 0.6785 - validation_accuracy: 0.5273
Epoch 16/20 - SGD - training_loss: 0.6719 - training_accuracy: 0.5585 - validation_loss: 0.6777 - validation_accuracy: 0.5432
Epoch 17/20 - SGD - training_loss: 0.6705 - training_accuracy: 0.5687 - validation_loss: 0.6769 - validation_accuracy: 0.5432
Epoch 18/20 - SGD - training_loss: 0.6693 - training_accuracy: 0.5722 - validation_loss: 0.6762 - validation_accuracy: 0.5455
Epoch 19/20 - SGD - training_loss: 0.6679 - training_accuracy: 0.5841 - validation_loss: 0.6754 - validation_accuracy: 0.5500
Epoch 20/20 - SGD - training_loss: 0.6668 - training_accuracy: 0.5852 - validation_loss: 0.6747 - validation_accuracy: 0.5614
Experiment with SGD completed in 87.13 seconds
Evaluating on Test Data with SGD...
Test Loss: 0.6636 - Test Accuracy: 0.5870
```



Adam demonstrates significantly better performance compared to SGD, achieving higher accuracy and a faster reduction in loss during the initial epochs. For Adam, the training loss rapidly decreases, and validation accuracy improves substantially within the first few epochs, indicating quicker convergence. Adam reaches a test accuracy of 64.2% compared to SGD's 58.7%. Additionally, the time taken for the Adam experiment (37.75 seconds) is considerably less than that for SGD (87.13 seconds). While SGD shows a gradual improvement in both loss and accuracy over 20 epochs, it converges more slowly and requires more epochs to achieve comparable results. This underscores Adam's advantages in optimization speed and early accuracy gains, making it the preferable choice.

After concluding that Adam performed better, we proceeded with it and incorporated batch normalization into the model to evaluate if it would enhance the results.

```
Running Experiment with learning_rate=0.0001, batch_size=128 (no dropout)
Epoch 1/20 - learning_rate=0.0001 - training_loss: 0.7079 - training_accuracy: 0.5733 - validation_loss: 0.6767 - validation_accuracy: 0.6045
Epoch 2/20 - learning_rate=0.0001 - training_loss: 0.2374 - training_accuracy: 0.9534 - validation_loss: 0.6063 - validation_accuracy: 0.6750
Epoch 3/20 - learning_rate=0.0001 - training_loss: 0.0854 - training_accuracy: 0.9989 - validation_loss: 0.6377 - validation_accuracy: 0.6750
Epoch 4/20 - learning_rate=0.0001 - training_loss: 0.0362 - training_accuracy: 1.0000 - validation_loss: 0.6264 - validation_accuracy: 0.6727
Epoch 5/20 - learning_rate=0.0001 - training_loss: 0.0188 - training_accuracy: 1.0000 - validation_loss: 0.6273 - validation_accuracy: 0.6909
Epoch 6/20 - learning_rate=0.0001 - training_loss: 0.0120 - training_accuracy: 1.0000 - validation_loss: 0.6300 - validation_accuracy: 0.6955
Epoch 7/20 - learning_rate=0.0001 - training_loss: 0.0090 - training_accuracy: 1.0000 - validation_loss: 0.6390 - validation_accuracy: 0.6705
Early stopping triggered!
Experiment with learning_rate=0.0001 completed in 33.40 seconds
Evaluating on Test Data with learning_rate=0.0001...
Test Loss: 0.6475 - Test Accuracy: 0.6680
```



Compared to the first experiment, The inclusion of batch normalization significantly improved the performance of the model.

Stability in Training: With batch normalization, the training loss decreased more consistently, and the training accuracy reached 100% by the 5th epoch. This indicates that the network could optimize better with batch normalization, addressing the issue of internal covariate shift and stabilizing the learning process.

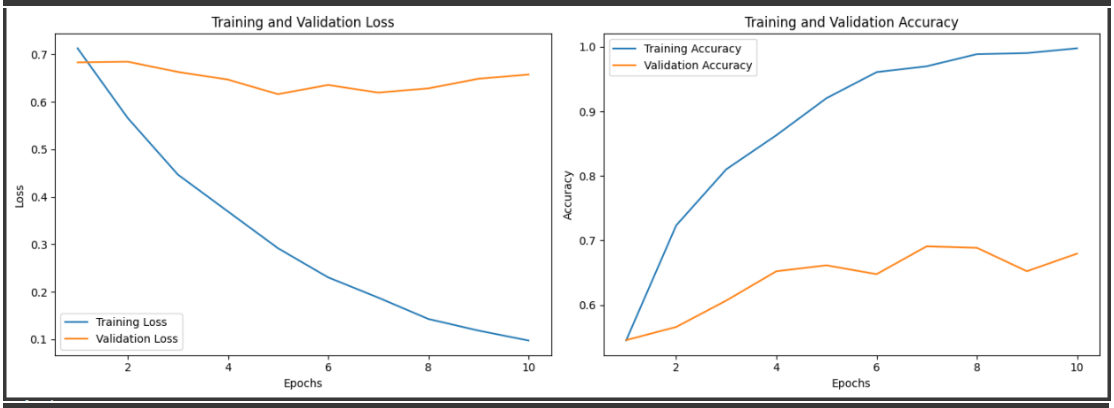
Validation Performance: While the model without batch normalization achieved a test accuracy of 64.2%, the version with batch normalization improved the validation and test accuracy to 66.8%. This improvement demonstrates that batch normalization not only helps in achieving better training accuracy but also improves the generalization of the model.

Test Loss: The test loss with batch normalization also decreased compared to the model without it (0.6475 vs. 0.8784), indicating better alignment of predictions with the ground truth.

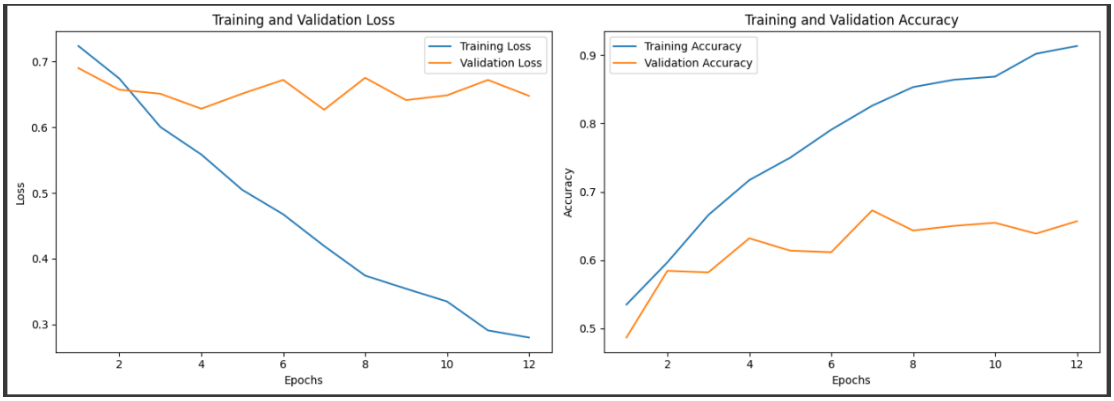
206477952
207547183

After concluding that batch normalization improved the results, we proceeded to test whether adding dropout would further enhance the model's performance. To evaluate this, we conducted experiments with dropout rates of 0.1 and 0.2

```
Running Experiment with dropout_rate=0.1, learning_rate=0.0001, batch_size=128
Epoch 1/20 - train_loss: 0.7130 - train_acc: 0.5449 - val_loss: 0.6833 - val_acc: 0.5455
Epoch 2/20 - train_loss: 0.5662 - train_acc: 0.7233 - val_loss: 0.6848 - val_acc: 0.5659
Epoch 3/20 - train_loss: 0.4470 - train_acc: 0.8102 - val_loss: 0.6632 - val_acc: 0.6068
Epoch 4/20 - train_loss: 0.3693 - train_acc: 0.8631 - val_loss: 0.6471 - val_acc: 0.6523
Epoch 5/20 - train_loss: 0.2917 - train_acc: 0.9205 - val_loss: 0.6163 - val_acc: 0.6614
Epoch 6/20 - train_loss: 0.2304 - train_acc: 0.9608 - val_loss: 0.6358 - val_acc: 0.6477
Epoch 7/20 - train_loss: 0.1877 - train_acc: 0.9699 - val_loss: 0.6196 - val_acc: 0.6909
Epoch 8/20 - train_loss: 0.1428 - train_acc: 0.9886 - val_loss: 0.6286 - val_acc: 0.6886
Epoch 9/20 - train_loss: 0.1183 - train_acc: 0.9903 - val_loss: 0.6488 - val_acc: 0.6523
Epoch 10/20 - train_loss: 0.0974 - train_acc: 0.9977 - val_loss: 0.6577 - val_acc: 0.6795
Early stopping triggered!
Experiment completed in 46.91 seconds
Evaluating on Test Data...
Test Loss: 0.6487 - Test Accuracy: 0.6630
```




```
Running Experiment with dropout_rate=0.2, learning_rate=0.0001, batch_size=128
Epoch 1/20 - train_loss: 0.7234 - train_acc: 0.5347 - val_loss: 0.6898 - val_acc: 0.4864
Epoch 2/20 - train_loss: 0.6738 - train_acc: 0.5966 - val_loss: 0.6569 - val_acc: 0.5841
Epoch 3/20 - train_loss: 0.6003 - train_acc: 0.6659 - val_loss: 0.6508 - val_acc: 0.5818
Epoch 4/20 - train_loss: 0.5585 - train_acc: 0.7170 - val_loss: 0.6280 - val_acc: 0.6318
Epoch 5/20 - train_loss: 0.5047 - train_acc: 0.7500 - val_loss: 0.6509 - val_acc: 0.6136
Epoch 6/20 - train_loss: 0.4676 - train_acc: 0.7909 - val_loss: 0.6718 - val_acc: 0.6114
Epoch 7/20 - train_loss: 0.4194 - train_acc: 0.8261 - val_loss: 0.6265 - val_acc: 0.6727
Epoch 8/20 - train_loss: 0.3744 - train_acc: 0.8534 - val_loss: 0.6750 - val_acc: 0.6432
Epoch 9/20 - train_loss: 0.3544 - train_acc: 0.8642 - val_loss: 0.6411 - val_acc: 0.6500
Epoch 10/20 - train_loss: 0.3350 - train_acc: 0.8687 - val_loss: 0.6484 - val_acc: 0.6545
Epoch 11/20 - train_loss: 0.2910 - train_acc: 0.9023 - val_loss: 0.6718 - val_acc: 0.6386
Epoch 12/20 - train_loss: 0.2802 - train_acc: 0.9136 - val_loss: 0.6476 - val_acc: 0.6568
Early stopping triggered!
Experiment completed in 56.57 seconds
Evaluating on Test Data...
Test Loss: 0.6524 - Test Accuracy: 0.6350
```



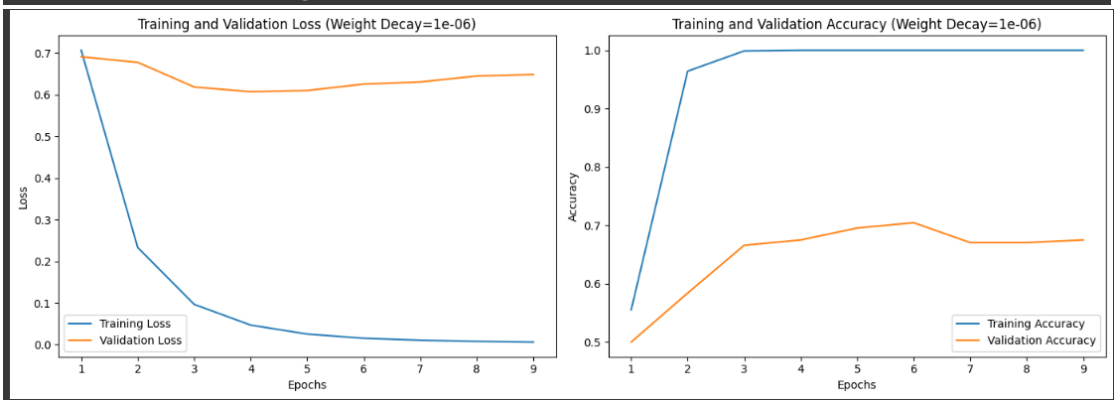
Neither of them improved the results compared to the batch normalization-only model.

We decided to conduct a final series of experiments to refine the model further. These tests aimed to determine the optimal batch size and evaluate the impact of L2 regularization (weight decay). We conducted 12 experiments in total, testing four different batch sizes (32, 64, 128, 256) with three weight decay values (1e-4, 1e-5, 1e-6) for each batch size.

- To streamline our analysis, we report only the top-performing configuration for each batch size.

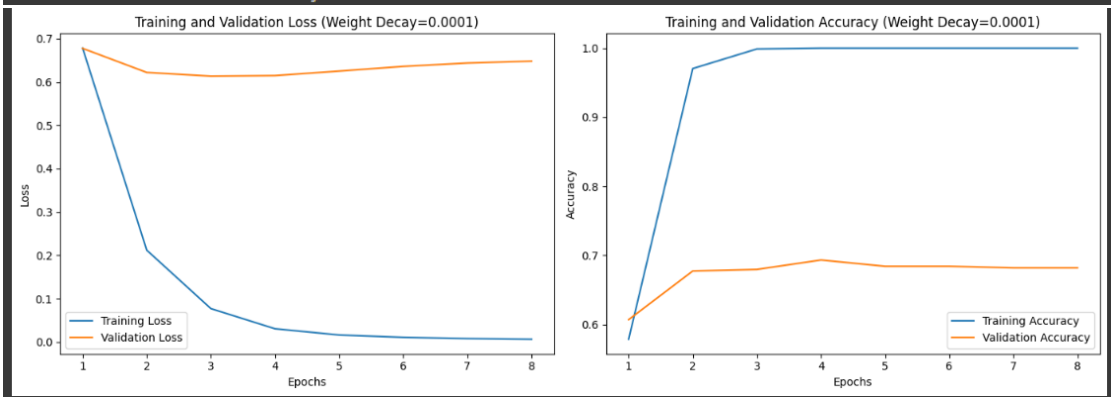
  Weight decay=0.000001, batch size=256:

```
Running Experiment with weight_decay=1e-06, learning_rate=0.0001, batch_size=256
Epoch 1/20 - weight_decay=1e-06 - training_loss: 0.7071 - training_accuracy: 0.5551 - validation_loss: 0.6914 - validation_accuracy: 0.5000
Epoch 2/20 - weight_decay=1e-06 - training_loss: 0.2332 - training_accuracy: 0.9642 - validation_loss: 0.6780 - validation_accuracy: 0.5841
Epoch 3/20 - weight_decay=1e-06 - training_loss: 0.0965 - training_accuracy: 0.9989 - validation_loss: 0.6188 - validation_accuracy: 0.6659
Epoch 4/20 - weight_decay=1e-06 - training_loss: 0.0468 - training_accuracy: 1.0000 - validation_loss: 0.6075 - validation_accuracy: 0.6750
Epoch 5/20 - weight_decay=1e-06 - training_loss: 0.0253 - training_accuracy: 1.0000 - validation_loss: 0.6104 - validation_accuracy: 0.6955
Epoch 6/20 - weight_decay=1e-06 - training_loss: 0.0151 - training_accuracy: 1.0000 - validation_loss: 0.6258 - validation_accuracy: 0.7045
Epoch 7/20 - weight_decay=1e-06 - training_loss: 0.0105 - training_accuracy: 1.0000 - validation_loss: 0.6309 - validation_accuracy: 0.6705
Epoch 8/20 - weight_decay=1e-06 - training_loss: 0.0077 - training_accuracy: 1.0000 - validation_loss: 0.6452 - validation_accuracy: 0.6705
Epoch 9/20 - weight_decay=1e-06 - training_loss: 0.0059 - training_accuracy: 1.0000 - validation_loss: 0.6490 - validation_accuracy: 0.6750
Early stopping triggered!
Experiment with weight_decay=1e-06 completed in 38.47 seconds
Evaluating on Test Data with weight_decay=1e-06...
Test Loss: 0.6212 - Test Accuracy: 0.6830
```



Weight decay=0.0001, batch size=128:

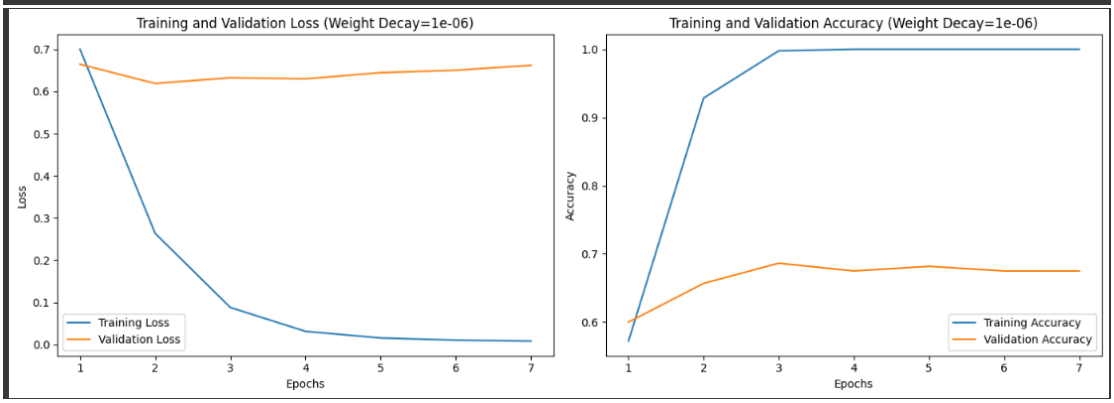
```
Running Experiment with weight_decay=0.0001, learning_rate=0.0001, batch_size=128
Epoch 1/20 - weight_decay=0.0001 - train_loss: 0.6783 - train_acc: 0.5784 - val_loss: 0.6779 - val_acc: 0.6068
Epoch 2/20 - weight_decay=0.0001 - train_loss: 0.2119 - train_acc: 0.9705 - val_loss: 0.6222 - val_acc: 0.6773
Epoch 3/20 - weight_decay=0.0001 - train_loss: 0.0771 - train_acc: 0.9989 - val_loss: 0.6138 - val_acc: 0.6795
Epoch 4/20 - weight_decay=0.0001 - train_loss: 0.0305 - train_acc: 1.0000 - val_loss: 0.6150 - val_acc: 0.6932
Epoch 5/20 - weight_decay=0.0001 - train_loss: 0.0161 - train_acc: 1.0000 - val_loss: 0.6254 - val_acc: 0.6841
Epoch 6/20 - weight_decay=0.0001 - train_loss: 0.0107 - train_acc: 1.0000 - val_loss: 0.6362 - val_acc: 0.6841
Epoch 7/20 - weight_decay=0.0001 - train_loss: 0.0078 - train_acc: 1.0000 - val_loss: 0.6440 - val_acc: 0.6818
Epoch 8/20 - weight_decay=0.0001 - train_loss: 0.0062 - train_acc: 1.0000 - val_loss: 0.6484 - val_acc: 0.6818
Early stopping triggered!
Experiment with weight_decay=0.0001 completed in 37.07 seconds
Evaluating on Test Data with weight_decay=0.0001...
Test Loss: 0.6146 - Test Accuracy: 0.6960
```



206477952
207547183

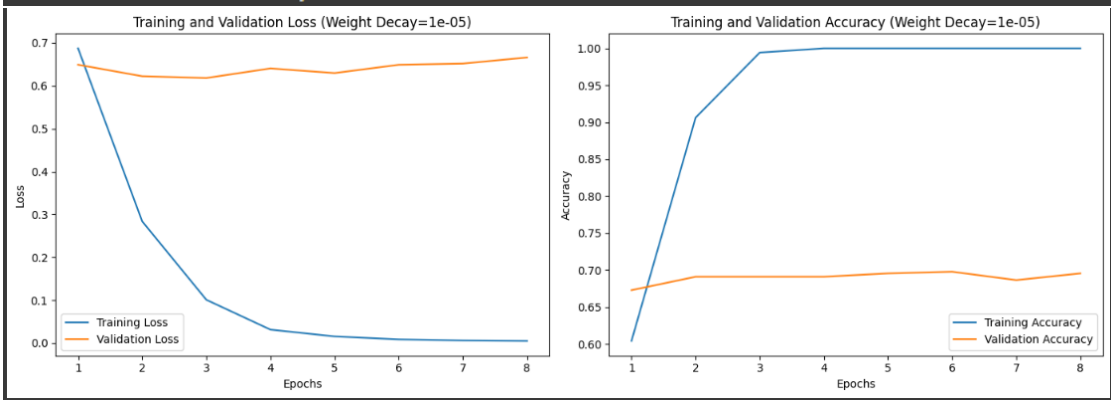
Weight decay=0.000001, batch size=64:

```
Running Experiment with weight_decay=1e-06, learning_rate=0.0001, batch_size=64
Epoch 1/20 - weight_decay=1e-06 - train_loss: 0.7002 - train_acc: 0.5722 - val_loss: 0.6647 - val_acc: 0.6000
Epoch 2/20 - weight_decay=1e-06 - train_loss: 0.2637 - train_acc: 0.9284 - val_loss: 0.6191 - val_acc: 0.6568
Epoch 3/20 - weight_decay=1e-06 - train_loss: 0.0881 - train_acc: 0.9977 - val_loss: 0.6327 - val_acc: 0.6864
Epoch 4/20 - weight_decay=1e-06 - train_loss: 0.0314 - train_acc: 1.0000 - val_loss: 0.6301 - val_acc: 0.6750
Epoch 5/20 - weight_decay=1e-06 - train_loss: 0.0157 - train_acc: 1.0000 - val_loss: 0.6447 - val_acc: 0.6818
Epoch 6/20 - weight_decay=1e-06 - train_loss: 0.0105 - train_acc: 1.0000 - val_loss: 0.6504 - val_acc: 0.6750
Epoch 7/20 - weight_decay=1e-06 - train_loss: 0.0082 - train_acc: 1.0000 - val_loss: 0.6622 - val_acc: 0.6750
Early stopping triggered!
Experiment with weight_decay=1e-06 completed in 43.05 seconds
Evaluating on Test Data with weight_decay=1e-06...
Test Loss: 0.6343 - Test Accuracy: 0.6870
```



Weight decay=0.00001, batch size=32:

```
Running Experiment with weight_decay=1e-05, learning_rate=0.0001, batch_size=32
Epoch 1/20 - weight_decay=1e-05 - train_loss: 0.6874 - train_acc: 0.6040 - val_loss: 0.6493 - val_acc: 0.6727
Epoch 2/20 - weight_decay=1e-05 - train_loss: 0.2839 - train_acc: 0.9062 - val_loss: 0.6224 - val_acc: 0.6909
Epoch 3/20 - weight_decay=1e-05 - train_loss: 0.1010 - train_acc: 0.9943 - val_loss: 0.6182 - val_acc: 0.6909
Epoch 4/20 - weight_decay=1e-05 - train_loss: 0.0314 - train_acc: 1.0000 - val_loss: 0.6406 - val_acc: 0.6909
Epoch 5/20 - weight_decay=1e-05 - train_loss: 0.0155 - train_acc: 1.0000 - val_loss: 0.6298 - val_acc: 0.6955
Epoch 6/20 - weight_decay=1e-05 - train_loss: 0.0086 - train_acc: 1.0000 - val_loss: 0.6491 - val_acc: 0.6977
Epoch 7/20 - weight_decay=1e-05 - train_loss: 0.0060 - train_acc: 1.0000 - val_loss: 0.6519 - val_acc: 0.6864
Epoch 8/20 - weight_decay=1e-05 - train_loss: 0.0049 - train_acc: 1.0000 - val_loss: 0.6662 - val_acc: 0.6955
Early stopping triggered!
Experiment with weight_decay=1e-05 completed in 55.25 seconds
Evaluating on Test Data with weight_decay=1e-05...
Test Loss: 0.6299 - Test Accuracy: 0.7040
```



The configuration with a weight decay of 1e-5 and a batch size of 32 yielded the best results observed during the experiments. The model achieved a validation accuracy of 69.7% after early stopping and attained a test accuracy of 70.4%, which marks the highest performance across all tested configurations. The low test loss of 0.6299 indicates that the model was not only accurate but also made predictions with a relatively small margin of error.

Our final model:

Our final model incorporates the insights gained from previous experiments, such as the use of batch normalization and L2 regularization and was trained with a batch size of 32 and a learning rate of 0.0001.

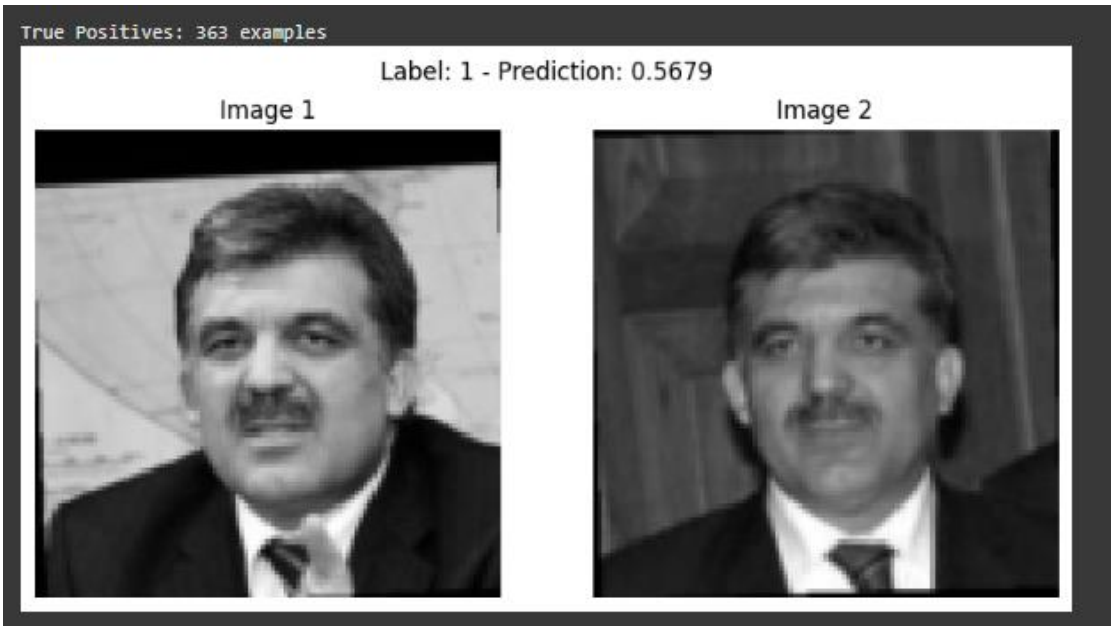
```
class SiameseNetwork(nn.Module):
    def __init__(self):
        super(SiameseNetwork, self).__init__()
        self.cnn = nn.Sequential(
            nn.Conv2d(1, 64, kernel_size=10, stride=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2),  # Output: 64 @ 48x48

            nn.Conv2d(64, 128, kernel_size=7, stride=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2),  # Output: 128 @ 21x21

            nn.Conv2d(128, 128, kernel_size=4, stride=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2),  # Output: 128 @ 9x9

            nn.Conv2d(128, 256, kernel_size=4, stride=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True)  # Final convolutional block
        )

        self.fc1 = nn.Sequential(
            nn.Linear(256 * 6 * 6, 4096),
            nn.BatchNorm1d(4096),
            nn.ReLU(),
        )

        self.fc2 = nn.Sequential(
            nn.Linear(4096, 1),
            nn.Sigmoid()
        )
```

```
    def flatten_one(self, input):
        """
        Flattens the input tensor into (batch_size, -1).
        """
        return input.view(input.size(0), -1)

    def forward(self, one, two):
        # Forward pass through CNN
        output1 = self.cnn(one)
        output2 = self.cnn(two)

        # Flatten the outputs
        output1 = self.flatten_one(output1)
        output2 = self.flatten_one(output2)

        # Pass through the first fully connected layer
        output1 = self.fc1(output1)
        output2 = self.fc1(output2)

        # Compute absolute difference and pass through final FC layer
        similarity = self.fc2(torch.abs(output1 - output2))
        return similarity
```

206477952
207547183

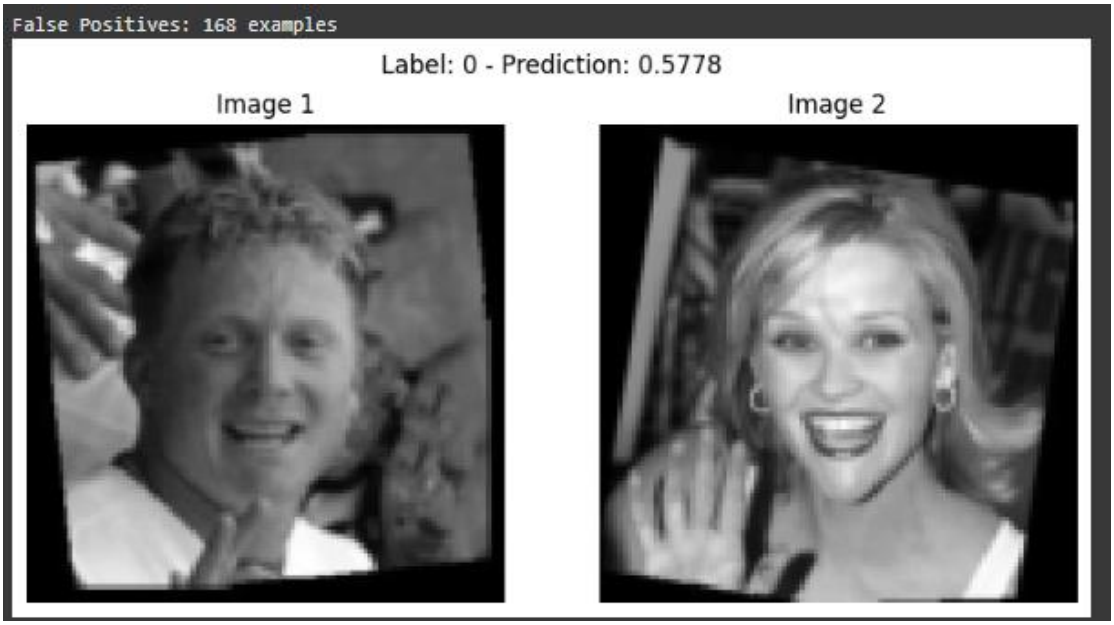
Prediction results:

- **True Positive:**



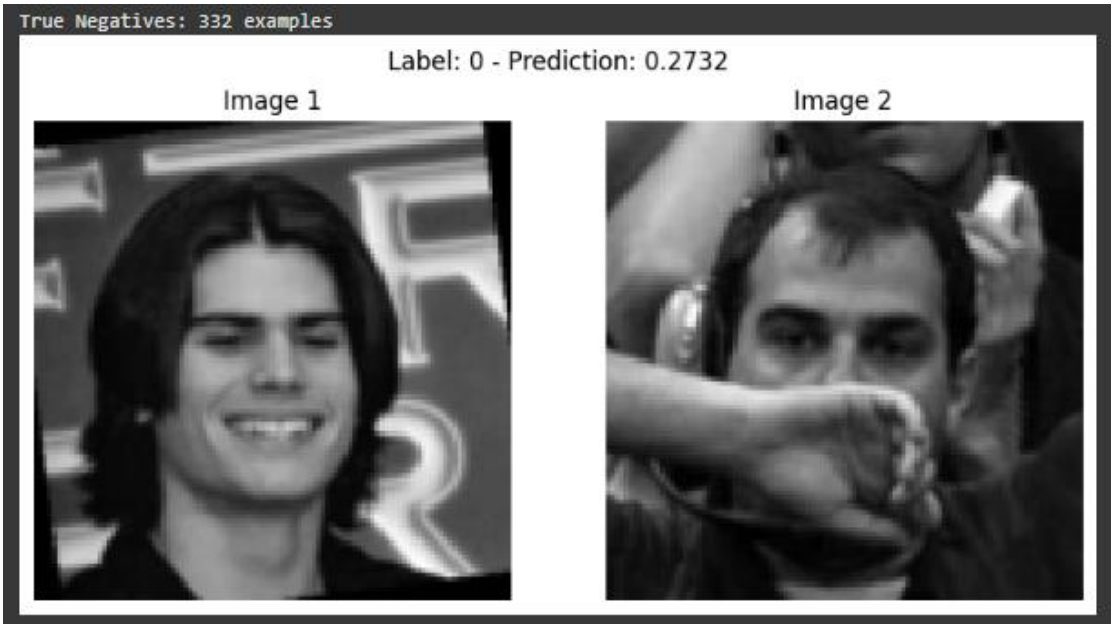
The model successfully classified this pair of images as True Positives with a prediction probability of 0.5679, indicating that the model identified these two images as belonging to the same person with a moderate level of confidence.
The ground truth label for this pair is 1, which means that both images do indeed depict the same person. The model's output score of 0.5679 is above the decision threshold of 0.5, resulting in a correct classification.
While the model correctly classified this example, the prediction score (0.5679) implies a certain level of uncertainty. This can be attributed to:
Changes in lighting or angle between the two images.
Subtle changes in facial expressions or environmental conditions.
Finally, despite the uncertainty, the model successfully classified the images as belonging to the same person. Observing the images, we can see that they both clearly depict the same individual, even though the right image is darker, and the left image shows the person with squinted eyes. The model did indeed correctly identify that it is the same person.

False Positive:



This pair of images was incorrectly classified as False Positives with a prediction probability of 0.5778, indicating that the model mistakenly identified these two images as depicting the same person.
The ground truth label for this pair is 0, which means that the two images belong to different individuals. However, the model's prediction score of 0.5778 exceeded the decision threshold of 0.5, resulting in a misclassification.
We assume that this misclassification occurred because both individuals are shown with similar expressions—both are smiling with their teeth exposed. Additionally, the hand gesture in both images is the same, and both individuals have light skin tones, which might have further contributed to the confusion.
In this case, while the two images clearly belong to different individuals, the model was unable to differentiate effectively due to the shared visual features that influenced the classification.

- **True Negative:**



This pair of images was correctly classified as True Negatives with a prediction probability of 0.2732, indicating that the model accurately identified these two images as belonging to different individuals.
The ground truth label for this pair is 0, meaning the two images represent different people. The model's prediction score of 0.2732 is below the decision threshold of 0.5, resulting in a correct classification.
This accurate classification can be attributed to the distinct visual differences between the two images: The individuals have noticeably different facial structures, the individual in Image 1 has longer hair styled down, while the individual in Image 2 has shorter hair, which contributes to their visual distinction, The individual in Image 1 is smiling, while the individual in Image 2 has a neutral expression and their face is partially obscured by their arm, The backgrounds and settings in the two images are dissimilar, further reinforcing the distinction. In this case, the model effectively distinguished between the two individuals by leveraging these diverse and meaningful features, demonstrating its robustness in classification.

- **False Negative:**

This pair of images was incorrectly classified as False Negatives with a prediction probability of 0.1162, indicating that the model failed to identify these two images as depicting the same person. The ground truth label for this pair is 1, which means both images represent the same individual. However, the model's prediction score of 0.1162 is significantly below the decision threshold of 0.5, resulting in a misclassification. The misclassification could be due to the following factors: In Image 1, the individual is facing directly towards the camera, whereas in Image 2, the person is shown in profile. This side angle makes it challenging for the model to capture and compare facial features effectively.
The angle of the photo in Image 2 affects the visibility of the face significantly, as only half of the face is visible, for example we can see only one eye. This limits the key features the model can use for comparison. Additionally, the background and surrounding individuals in Image 2 may have introduced noise, further complicating the model's ability to focus on the primary subject.