Adi Maman – adimaman22@gmail.com

Deep Learning Project - Siamese Neural Networks

Project Purpose

The purpose of this project was to use convolutional neural networks (CNNs) to perform facial recognition. Specifically, we implemented a one-shot classification solution using Siamese Neural Networks, as described in the paper “Siamese Neural Networks for One-shot Image Recognition” (Koch et al., 2015).

Data Description

The "Labeled Faces in the Wild-a" (LFW-a) dataset is a collection of labeled face images designed for studying face recognition in unconstrained settings.

For this assignment, we were provided with a predefined train/test split via text files, ensuring no image overlaps between the two sets. This design ensures the learning task aligns with one-shot learning principles.

The training set contains 2200 pairs (1100 positive and 1100 negative), and the test set contains 1000 pairs (500 positive and 500 negative).

To enhance the training process, and due to the relatively small size of the original dataset, we split the original train set into a new train set and a validation set with a **90:10 ratio**. This decision was made to maximize the number of pairs available for training while still maintaining a small validation set for hyperparameter tuning and monitoring model performance.

Data distribution:

	Total pairs	*Positive pairs (same person)*	*Negative pairs (different people)*
Training Set	1980	985	995
Validation Set	220	115	105
Test Set	1000	500	500

Preprocessing:

- Dataset Download and Extraction:
  The dataset, "Labeled Faces in the Wild-a" (LFW-a), was downloaded as a .zip file from the given Link. The extraction process was verified to ensure all files were successfully unzipped.

- Dataset Class:
  A custom PyTorch Dataset class (LFWPairsDataset) was implemented to handle the LFW dataset's unique structure. The dataset pairs (1 for same person, 0 for different people) were defined by pairsDevTrain.txt and pairsDevTest.txt . Each pair was parsed, and the corresponding image file paths were generated.

- Data Transformations:
  The following transformations were applied to all images:
  - Resizing: Each image was resized to 105×105 pixels to match the input size described in the paper.
  - Grayscale Conversion: Images were converted to grayscale to reduce the computational complexity of training and align with the experimental setup described in the paper.
  - Tensor Conversion: Images were converted to PyTorch tensors for compatibility with the model.

- Train/Validation Split:
  The split, as described previously, maintained a balanced proportion of positive and negative pairs in both subsets.
  Note: A fixed random seed was used to ensure reproducibility of the split.

- Data Loaders:
  PyTorch DataLoader was used to batch and shuffle the dataset for efficient processing during training, validation, and testing.
  - **Batch Size**: Each batch contains 32 pairs of images along with their labels.

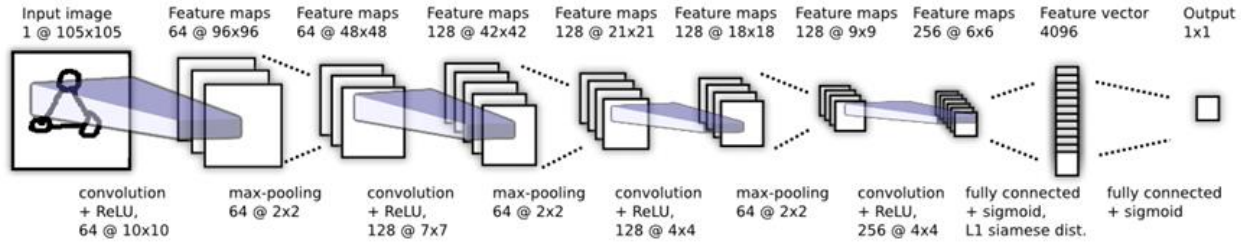
Examples of images after pre-processing:

Below, we provide examples of image pairs from the dataset. Each pair includes two grayscale images resized to 105×105 pixels. Positive pairs represent two images of the same person, while negative pairs consist of images of two different individuals.

These examples illustrate the diversity of the dataset and the complexity of the one-shot learning task, as the images often include variations in lighting, pose, and background.

Initial model architecture:

First, we implemented the model according to the architecture presented in the article and in the following illustration:



Paper General Architecture:

1) Number of Layers: 8
   4 convolutional layers, 3 max-pooling, and 1 fully connected layer.
2) Activations: ReLU (used after every convolutional layer)
3) Pooling: Max pooling applied after the first three convolutional layers.
4) Fully Connected Layer: Outputs a 4096-dimensional vector for similarity calculation.
5) Weight Initialization:
   - Convolutional Layers:
   Normal distribution with mean 0 and standard deviation 0.01.
   - Fully Connected Layers:
   Normal distribution with mean 0 and standard deviation 0.2.
   - Biases (all layers):
   Normal distribution with mean 0.5 and standard deviation 0.01.

Final Configuration

After testing various configurations on the validation set, we selected the following setup for the best performance:

1. Max Epochs:
   Training was capped at 100 epochs to ensure the model did not overfit (compared to the paper, in which the network trains for a maximum of 200 epochs).

2. Batch Size:
    - During the trial-and-error phase, sizes of 32, 64, and 128 were tested.
    - Batch size **64** yielded the most stable results in the final experiment.

3. Weight Initialization:
   We used **Kaiming Uniform Initialization** for weight initialization, which is specifically designed for layers with ReLU activations. This method ensures that the variance of the inputs to each layer remains stable throughout the network, facilitating better convergence and improved performance during the final experiment.

4. Optimization:
   The paper uses stochastic gradient descent (SGD) as the optimization method while our implementation uses **Adam**.
   Adam is often preferred for its adaptive learning rates and faster convergence, especially on smaller datasets.

5. Batch Normalization:
   Batch normalization was applied before each max-pooling operation.
   This addition consistently improved model convergence and stability in all experiments.

6. Loss Function:
   Binary Cross-Entropy Loss (torch.nn.BCELoss) was used as the model classifies whether two images belong to the same class (output: 0 or 1).

7. Early Stopping Criteria (Convergence Check):
   Training was stopped if the validation loss showed no improvement for 5 (default value) consecutive epochs.

8. Learning Rate:
   As described in the paper, we reduced the learning rate by 1% after each epoch using a multiplicative scheduler.

9. Dropout:
    - Dropout is applied in our architecture after each convolutional layer with p=0.3 (30% of neurons dropped during training).
    - Also applied before the fully connected layer in part1 of the network to further regularize the dense feature representations.

Experimental Setup

early_stopping_threshold=0.0001 (Early stopping threshold for validation loss changes)
patience=5 (Number of epochs to wait for improvement before stopping)

Experiment 1: Original paper architecture

(SGD optimizer, no Batch Normalization , no Dropout)

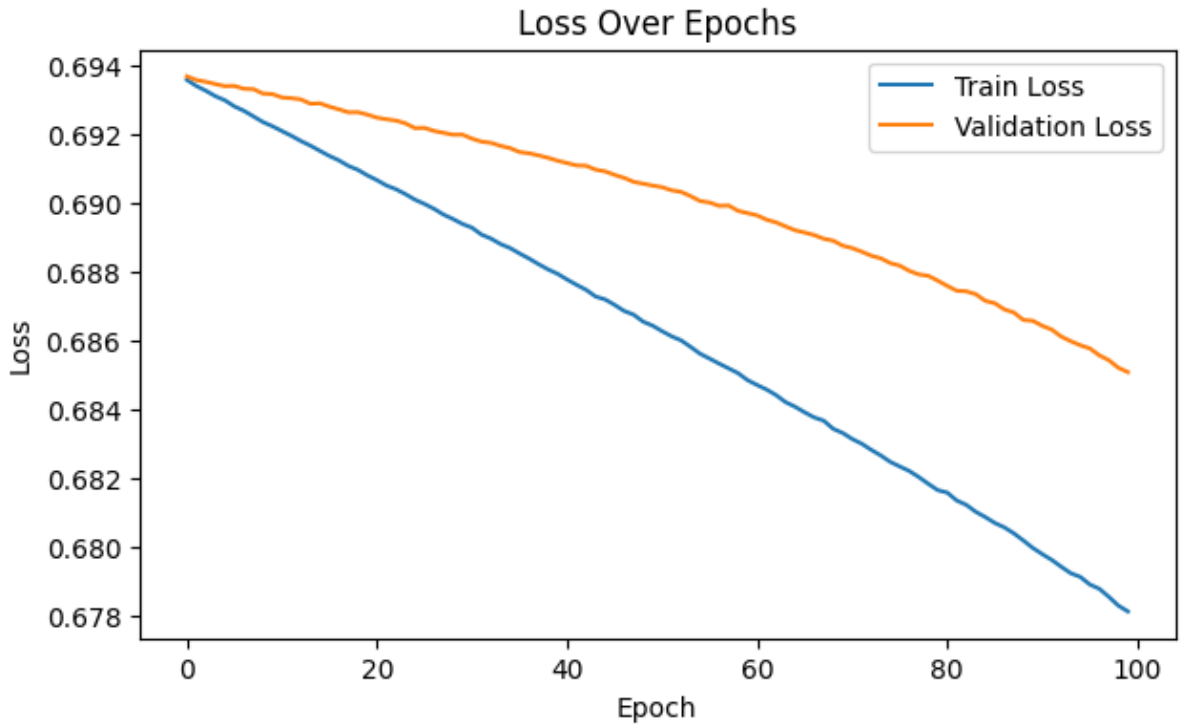
```
Siamese_Model(
  (part1): Sequential(
    (0): Conv2d(1, 64, kernel_size=(10, 10), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 128, kernel_size=(7, 7), stride=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(128, 128, kernel_size=(4, 4), stride=(1, 1))
    (7): ReLU()
    (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (9): Conv2d(128, 256, kernel_size=(4, 4), stride=(1, 1))
    (10): ReLU()
    (11): Flatten(start_dim=1, end_dim=-1)
    (12): Linear(in_features=9216, out_features=4096, bias=True)
  )
  (part2): Sequential(
    (0): Linear(in_features=4096, out_features=1, bias=True)
    (1): Sigmoid()
  )
)
```

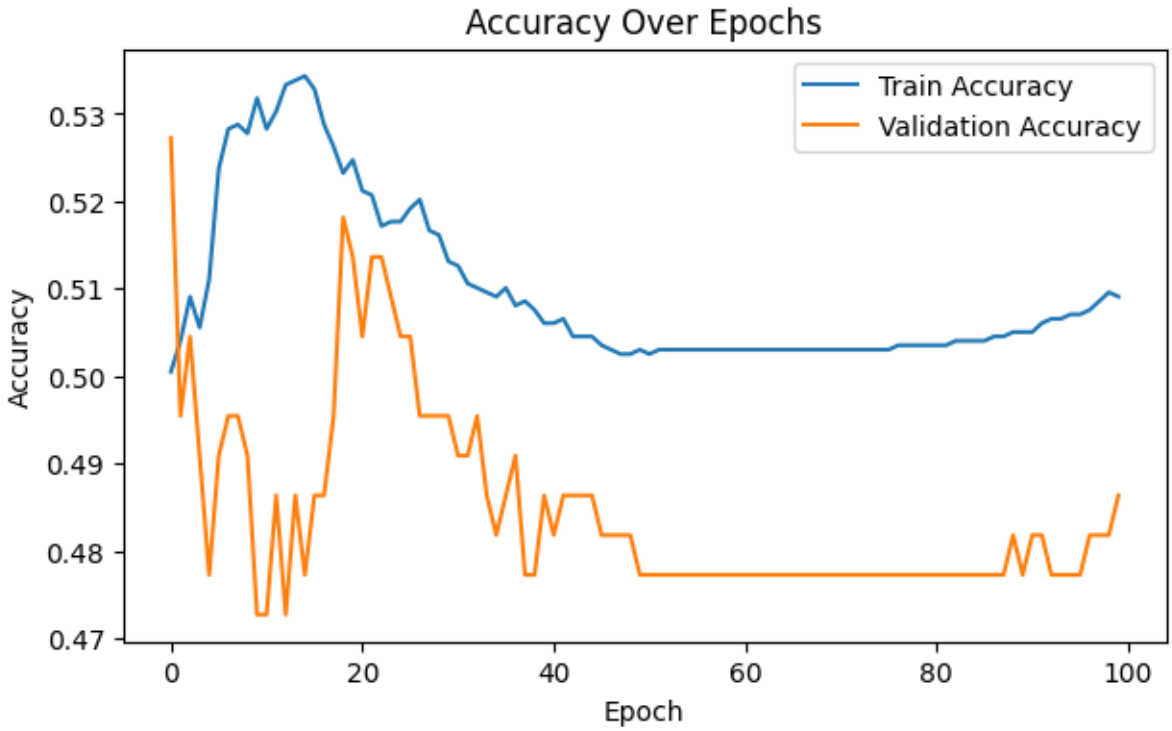
The Results:

```
Epoch 99/100:
Train Loss: 0.6783, Train Accuracy: 0.5096
Validation Loss: 0.6852, Validation Accuracy: 0.4818
Epoch 100/100:
Train Loss: 0.6781, Train Accuracy: 0.5091
Validation Loss: 0.6851, Validation Accuracy: 0.4864
Total Training Time: 800.87 seconds
```

```
Test Loss: 0.679149, Test Accuracy: 0.504000
```

Loss & Accuracy Plots:





Experiment 2: Original paper architecture with Batch Normalization

(SGD optimizer, no Dropout)

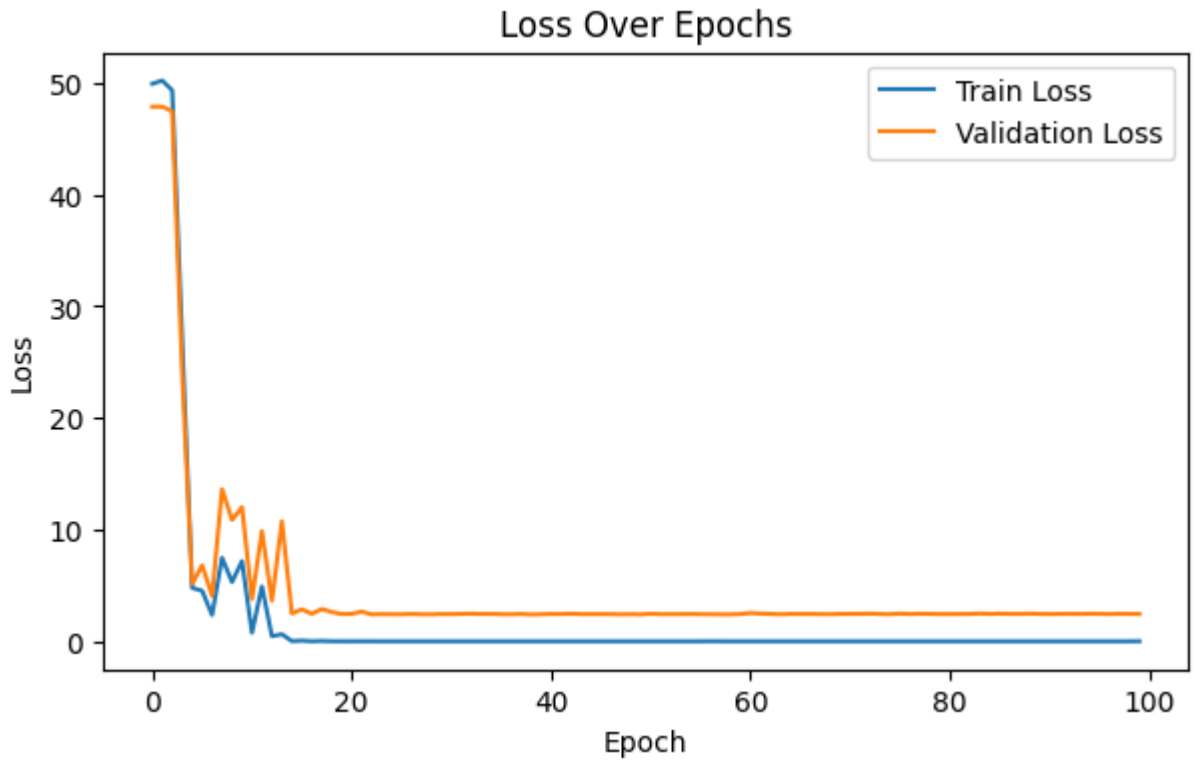
```
Siamese_Model(
  (part1): Sequential(
    (0): Conv2d(1, 64, kernel_size=(10, 10), stride=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(64, 128, kernel_size=(7, 7), stride=(1, 1))
    (5): ReLU()
    (6): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Conv2d(128, 128, kernel_size=(4, 4), stride=(1, 1))
    (9): ReLU()
    (10): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (12): Conv2d(128, 256, kernel_size=(4, 4), stride=(1, 1))
    (13): ReLU()
    (14): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (15): Flatten(start_dim=1, end_dim=-1)
    (16): Linear(in_features=9216, out_features=4096, bias=True)
  )
  (part2): Sequential(
    (0): Linear(in_features=4096, out_features=1, bias=True)
    (1): Sigmoid()
  )
)
```

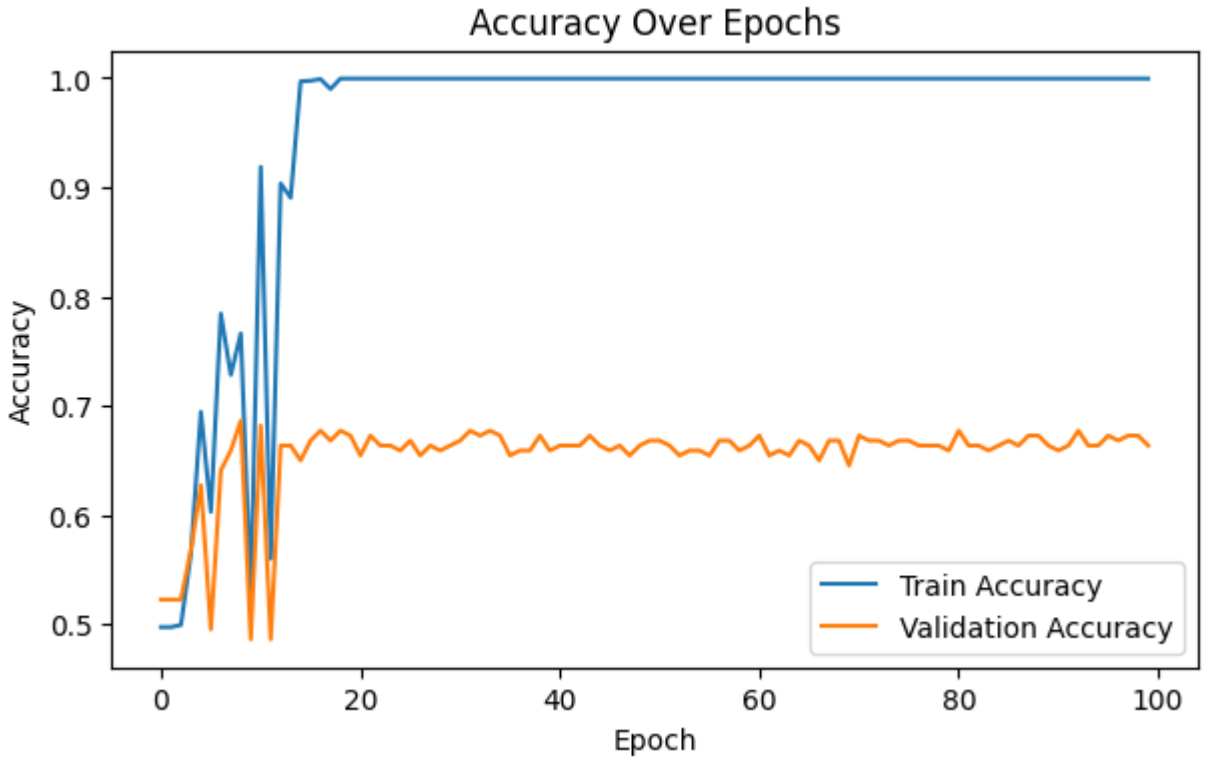
The Results:

```
Epoch 99/100:
Train Loss: 0.0504, Train Accuracy: 0.9995
Validation Loss: 2.4996, Validation Accuracy: 0.6727
Epoch 100/100:
Train Loss: 0.0576, Train Accuracy: 0.9995
Validation Loss: 2.4875, Validation Accuracy: 0.6636
Total Training Time: 819.51 seconds
```

```
Test Loss: 4.417677, Test Accuracy: 0.671000
```

Loss & Accuracy Plots:





It is evident that the experiment with Batch Normalization yielded better results, so we will proceed with that approach.
Next, we examined the effect of dropout by testing dropout rates of [0.1, 0.2, 0.3, 0.4, 0.5, 0,6] and observed a decrease in results.

```
Test Loss: 47.597377, Test Accuracy: 0.500000
```

Let's examine the effect of the optimizer used:

Experiment 3: Original paper architecture with ADAM optimizer

(with Batch Normalization, no Dropout)

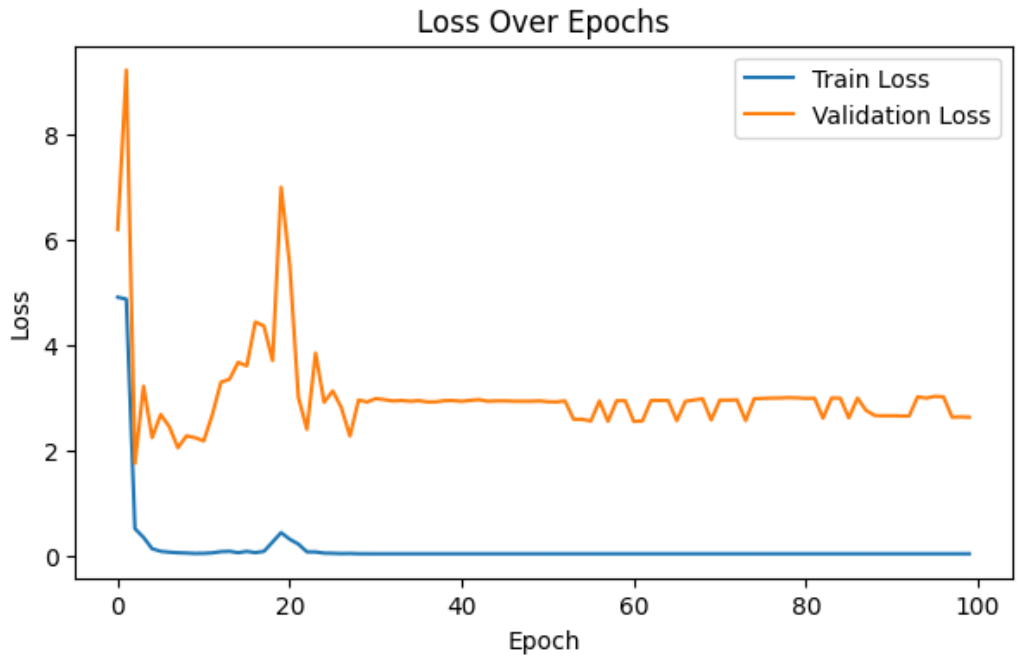
```
Siamese_Model(
  (part1): Sequential(
    (0): Conv2d(1, 64, kernel_size=(10, 10), stride=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(64, 128, kernel_size=(7, 7), stride=(1, 1))
    (5): ReLU()
    (6): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Conv2d(128, 128, kernel_size=(4, 4), stride=(1, 1))
    (9): ReLU()
    (10): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (12): Conv2d(128, 256, kernel_size=(4, 4), stride=(1, 1))
    (13): ReLU()
    (14): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (15): Flatten(start_dim=1, end_dim=-1)
    (16): Linear(in_features=9216, out_features=4096, bias=True)
  )
  (part2): Sequential(
    (0): Linear(in_features=4096, out_features=1, bias=True)
    (1): Sigmoid()
  )
)
```

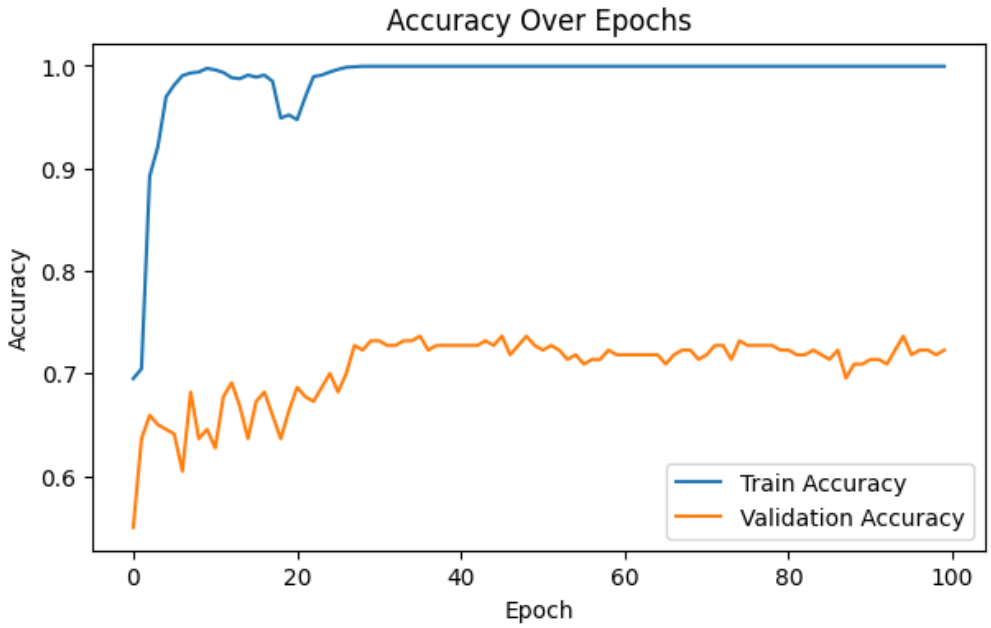
The Results:

```
Epoch 98/100:
Train Loss: 0.0504, Train Accuracy: 0.9995
Validation Loss: 2.6371, Validation Accuracy: 0.7227
Epoch 99/100:
Train Loss: 0.0504, Train Accuracy: 0.9995
Validation Loss: 2.6410, Validation Accuracy: 0.7182
Epoch 100/100:
Train Loss: 0.0504, Train Accuracy: 0.9995
Validation Loss: 2.6360, Validation Accuracy: 0.7227
Total Training Time: 838.04 seconds
```

```
Test Loss: 5.952237, Test Accuracy: 0.654000
```

Loss & Accuracy Plots:





We see that under identical conditions, the SGD optimizer yielded slightly better results than ADAM.

Note: Even with dropout rates of [0.3, 0.6], the overall performance declined. However, the overfitting was less pronounced, indicating that the dropout had a positive regularization effect.

The original architecture is very large, and with only 1980 pairs available for training, the network quickly overfit the training set. Specifically, the training accuracy reached 99.99%, while the validation accuracy remained at only 66%. Additionally, the training loss decreased sharply to near zero within just 5 epochs, whereas the validation loss began to increase, indicating overfitting. To address this issue, we significantly reduced the number of parameters in each layer to improve generalization and prevent overfitting.

We tested the reduced architecture using both optimizers: SGD and Adam.

Experiment 4: New Reduced Architecture with SGD Optimizer and Batch Normalization

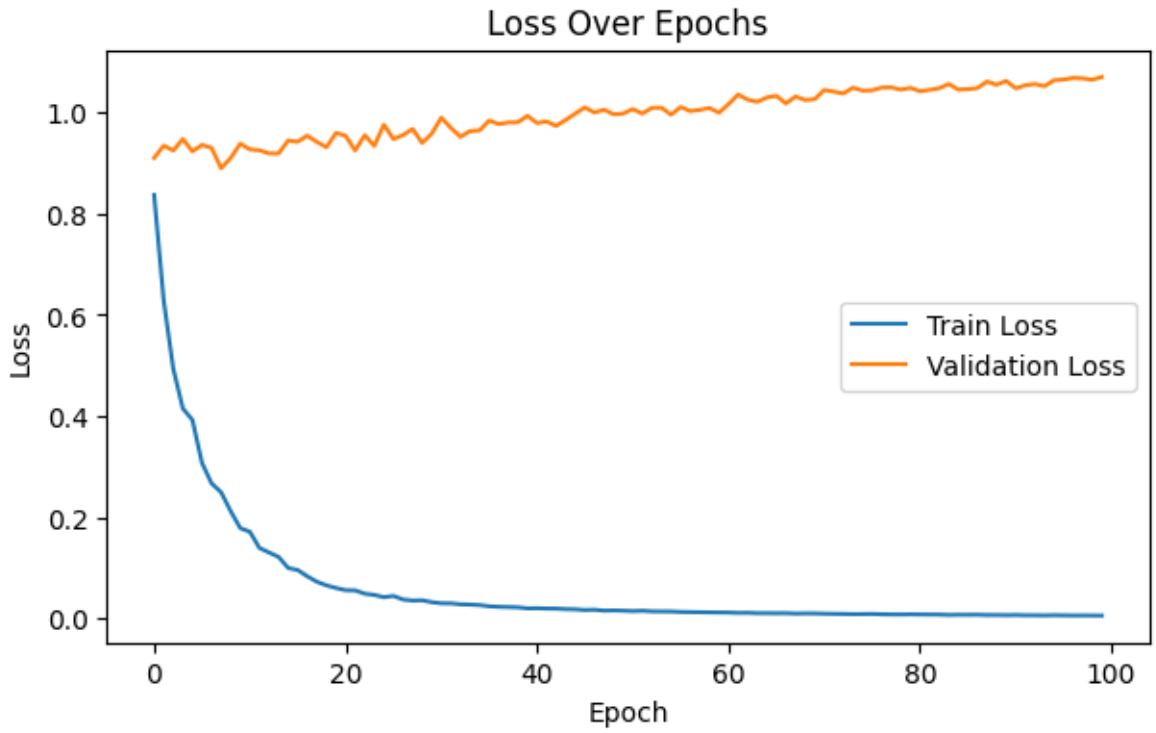
```
Siamese_Model(
  (part1): Sequential(
    (0): Conv2d(1, 32, kernel_size=(10, 10), stride=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(32, 32, kernel_size=(7, 7), stride=(1, 1))
    (5): ReLU()
    (6): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Conv2d(32, 32, kernel_size=(4, 4), stride=(1, 1))
    (9): ReLU()
    (10): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (12): Conv2d(32, 32, kernel_size=(4, 4), stride=(1, 1))
    (13): ReLU()
    (14): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (15): Flatten(start_dim=1, end_dim=-1)
    (16): Linear(in_features=1152, out_features=32, bias=True)
  )
  (part2): Sequential(
    (0): Linear(in_features=32, out_features=1, bias=True)
    (1): Sigmoid()
  )
)
```

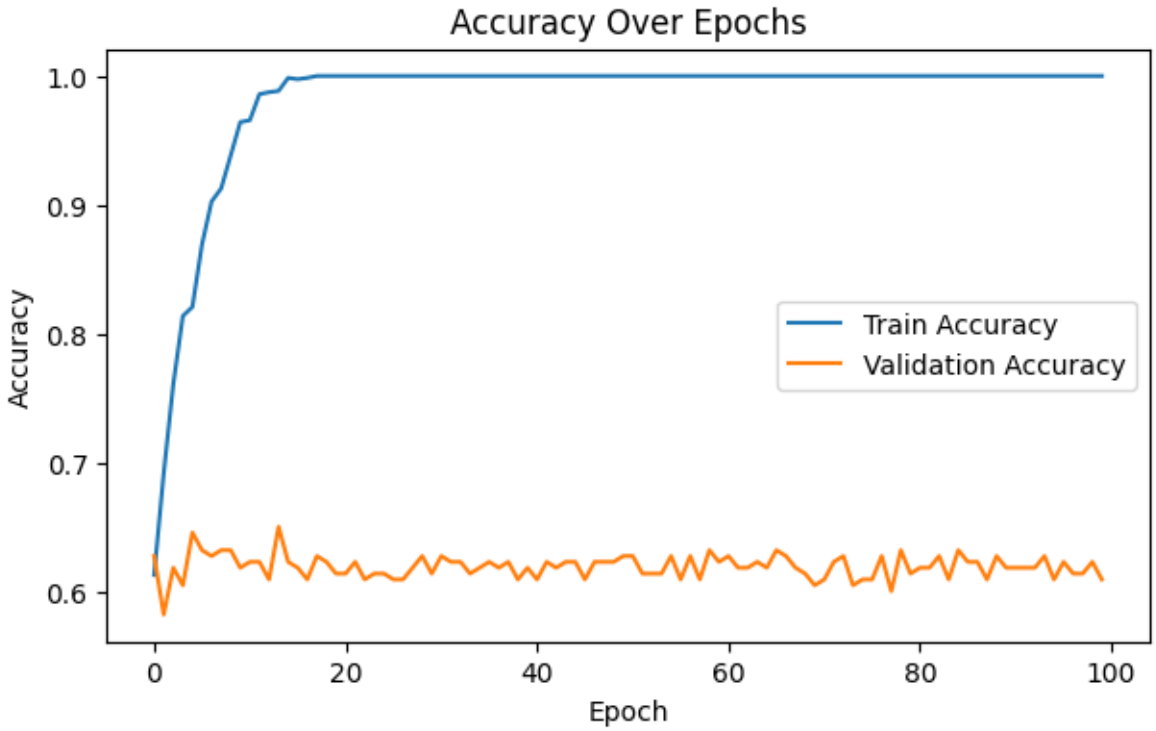
The Results:

```
Epoch 99/100:
Train Loss: 0.0063, Train Accuracy: 1.0000
Validation Loss: 1.0647, Validation Accuracy: 0.6227
Epoch 100/100:
Train Loss: 0.0062, Train Accuracy: 1.0000
Validation Loss: 1.0702, Validation Accuracy: 0.6091
Total Training Time: 757.14 seconds
```

```
Test Loss: 1.342467, Test Accuracy: 0.569000
```

Loss & Accuracy Plots:





Although the test loss was lower, the overall results were less favorable.

Experiment 5: New Reduced Architecture with ADAM optimizer and Batch Normalization

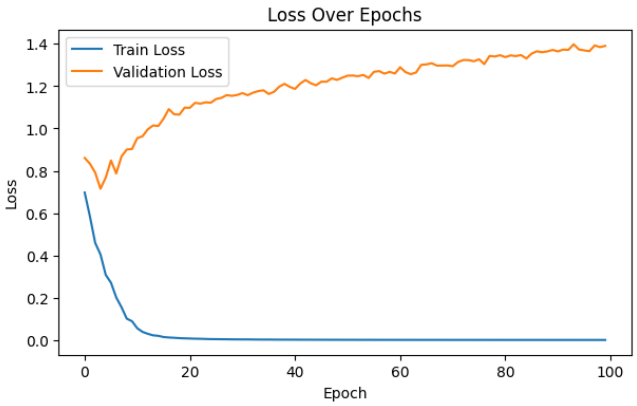
```
Siamese_Model(
  (part1): Sequential(
    (0): Conv2d(1, 32, kernel_size=(10, 10), stride=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(32, 32, kernel_size=(7, 7), stride=(1, 1))
    (5): ReLU()
    (6): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Conv2d(32, 32, kernel_size=(4, 4), stride=(1, 1))
    (9): ReLU()
    (10): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (12): Conv2d(32, 32, kernel_size=(4, 4), stride=(1, 1))
    (13): ReLU()
    (14): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (15): Flatten(start_dim=1, end_dim=-1)
    (16): Linear(in_features=1152, out_features=32, bias=True)
  )
  (part2): Sequential(
    (0): Linear(in_features=32, out_features=1, bias=True)
    (1): Sigmoid()
  )
)
```

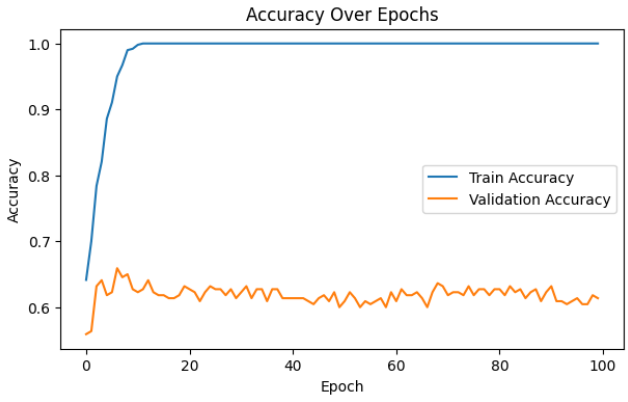
The Results:

```
Epoch 99/100:
Train Loss: 0.0001, Train Accuracy: 1.0000
Validation Loss: 1.3846, Validation Accuracy: 0.6182
Epoch 100/100:
Train Loss: 0.0001, Train Accuracy: 1.0000
Validation Loss: 1.3905, Validation Accuracy: 0.6136
Total Training Time: 945.02 seconds
```

```
Test Loss: 1.308112, Test Accuracy: 0.601000
```

Loss & Accuracy Plots:





So far, the most successful experiment has been achieved using the original architecture from the paper combined with Batch Normalization and the SGD optimizer (Experiment 2) but with high test loss.

At this stage, we plan to modify the weight initialization method: instead of following the approach described in the paper (using a normal distribution), we will use **Kaiming Uniform Initialization**. This method is specifically designed for layers with ReLU activations, as it helps maintain a stable variance of activations throughout the network, improving convergence and performance.

We observed that the combination of Kaiming Uniform Initialization with the **SGD optimizer** did not yield good results. However, when switching back to the **Adam optimizer**, we achieved improved performance. Despite the improvement, there was still a slight tendency toward overfitting, as indicated by the gap between training and validation performance. To mitigate this, we applied **Dropout with a rate of 0.3** [we tried: 0, 0.3, 0.5], which helped to reduce overfitting while maintaining the improved results.

Experiment 6: Original paper architecture with ADAM optimizer, Kaiming Uniform weight Initialization and Dropout=0.3 (with Batch Normalization)

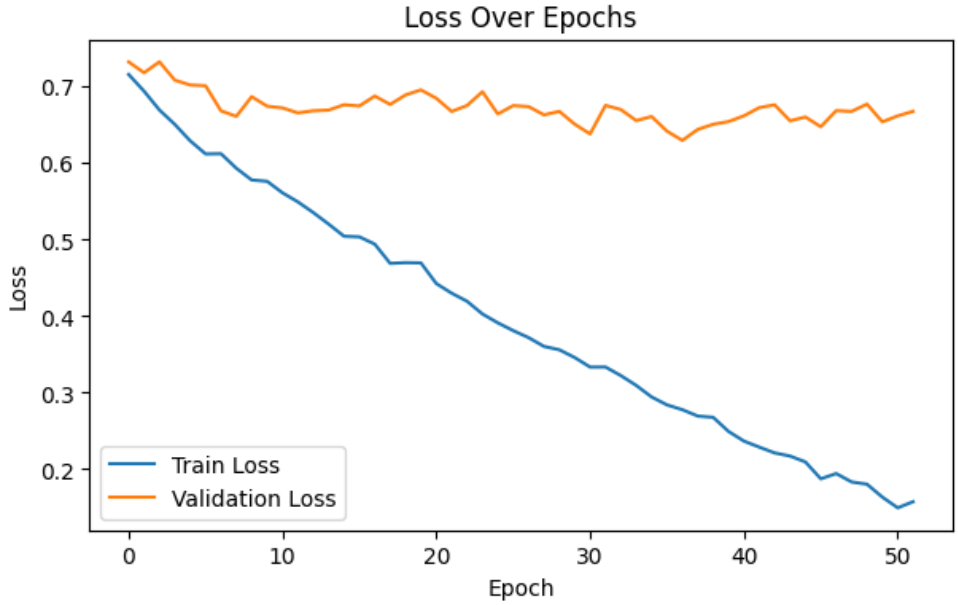
```
Siamese_Model(
  (part1): Sequential(
    (0): Conv2d(1, 64, kernel_size=(10, 10), stride=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.3, inplace=False)
    (5): Conv2d(64, 128, kernel_size=(7, 7), stride=(1, 1))
    (6): ReLU()
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (9): Dropout(p=0.3, inplace=False)
    (10): Conv2d(128, 128, kernel_size=(4, 4), stride=(1, 1))
    (11): ReLU()
    (12): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (14): Dropout(p=0.3, inplace=False)
    (15): Conv2d(128, 256, kernel_size=(4, 4), stride=(1, 1))
    (16): ReLU()
    (17): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (18): Dropout(p=0.3, inplace=False)
    (19): Flatten(start_dim=1, end_dim=-1)
    (20): Linear(in_features=9216, out_features=32, bias=True)
    (21): Dropout(p=0.3, inplace=False)
  )
  (part2): Sequential(
    (0): Linear(in_features=32, out_features=1, bias=True)
    (1): Sigmoid()
  )
)
```

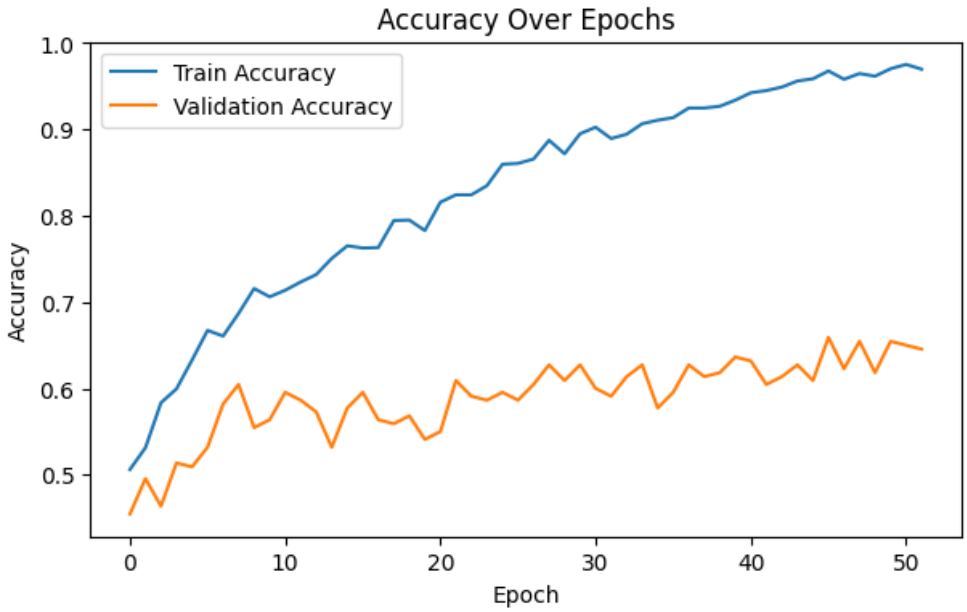
The Results:

```
Epoch 51/100:
Train Loss: 0.1493, Train Accuracy: 0.9753
Validation Loss: 0.6619, Validation Accuracy: 0.6500
Epoch 52/100:
Train Loss: 0.1574, Train Accuracy: 0.9697
Validation Loss: 0.6679, Validation Accuracy: 0.6455
Early stopping due to lack of improvement.
Total Training Time: 423.13 seconds
```

```
Test Loss: 0.702558, Test Accuracy: 0.632000
```

Loss & Accuracy Plots:





Throughout our experiments, we observed no significant differences in performance when varying the batch size, so all previously described experiments were conducted with a batch size of 32.

However, in this case, a batch size of 64 produced the best results:

Experiment 7: Original paper architecture with ADAM optimizer, Kaiming Uniform weight Initialization and Dropout=0.3

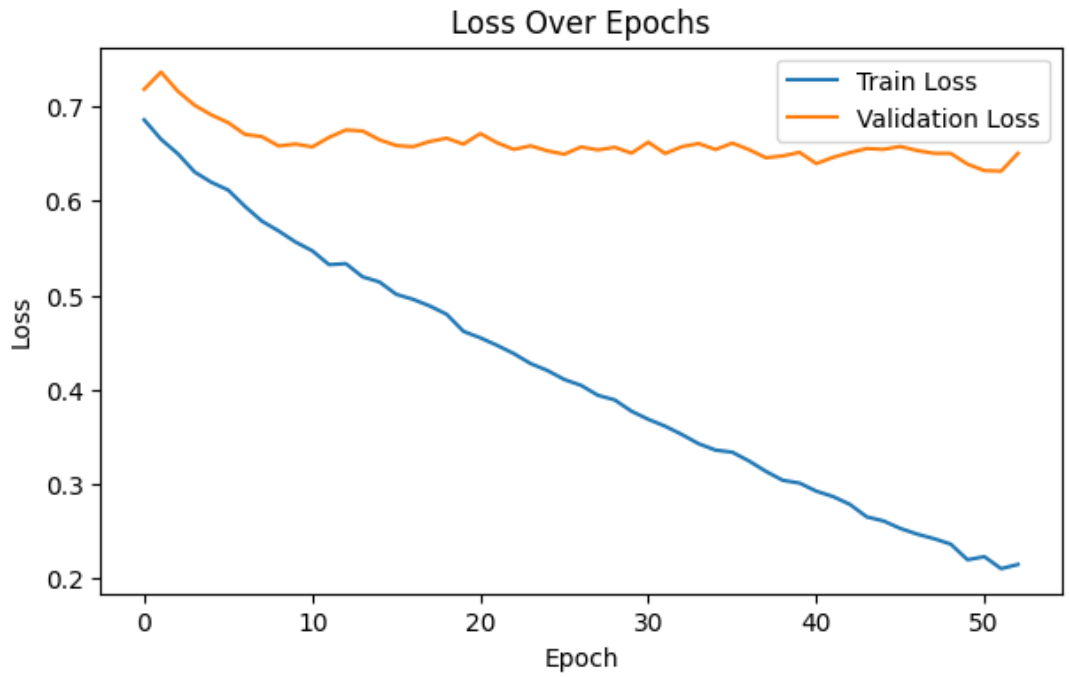
(with Batch Normalization and **Batch size of 64**)

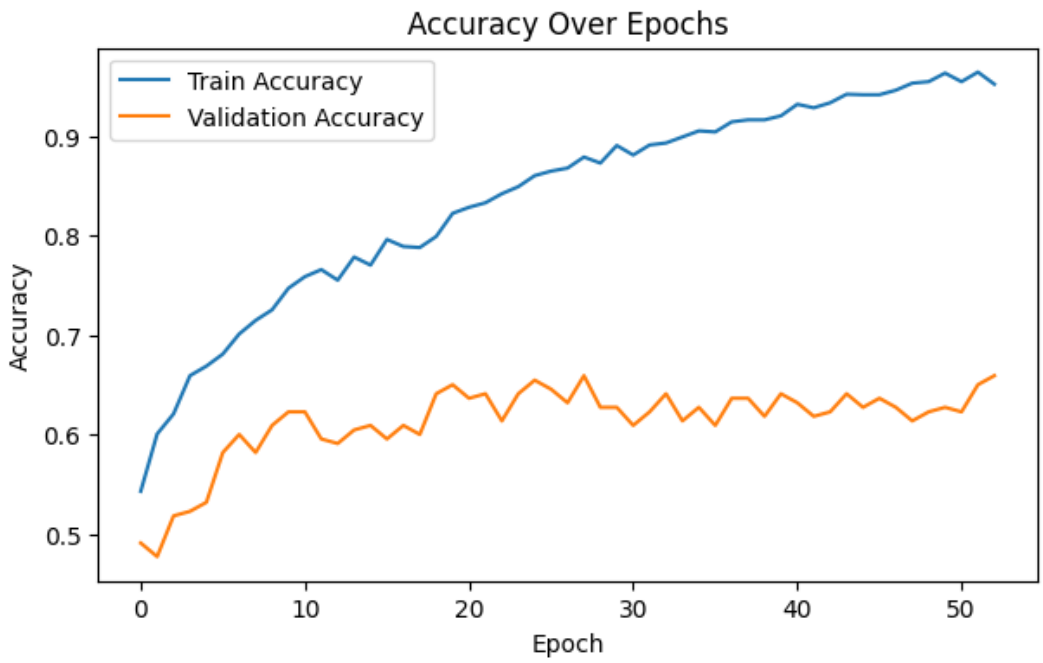
The Results:

```
Epoch 52/100:
Train Loss: 0.2106, Train Accuracy: 0.9641
Validation Loss: 0.6314, Validation Accuracy: 0.6500
Epoch 53/100:
Train Loss: 0.2151, Train Accuracy: 0.9520
Validation Loss: 0.6503, Validation Accuracy: 0.6591
Early stopping due to lack of improvement.
Total Training Time: 425.09 seconds
```

```
Test Loss: 0.639552, Test Accuracy: 0.659000
```

Loss & Accuracy Plots:





Finally,

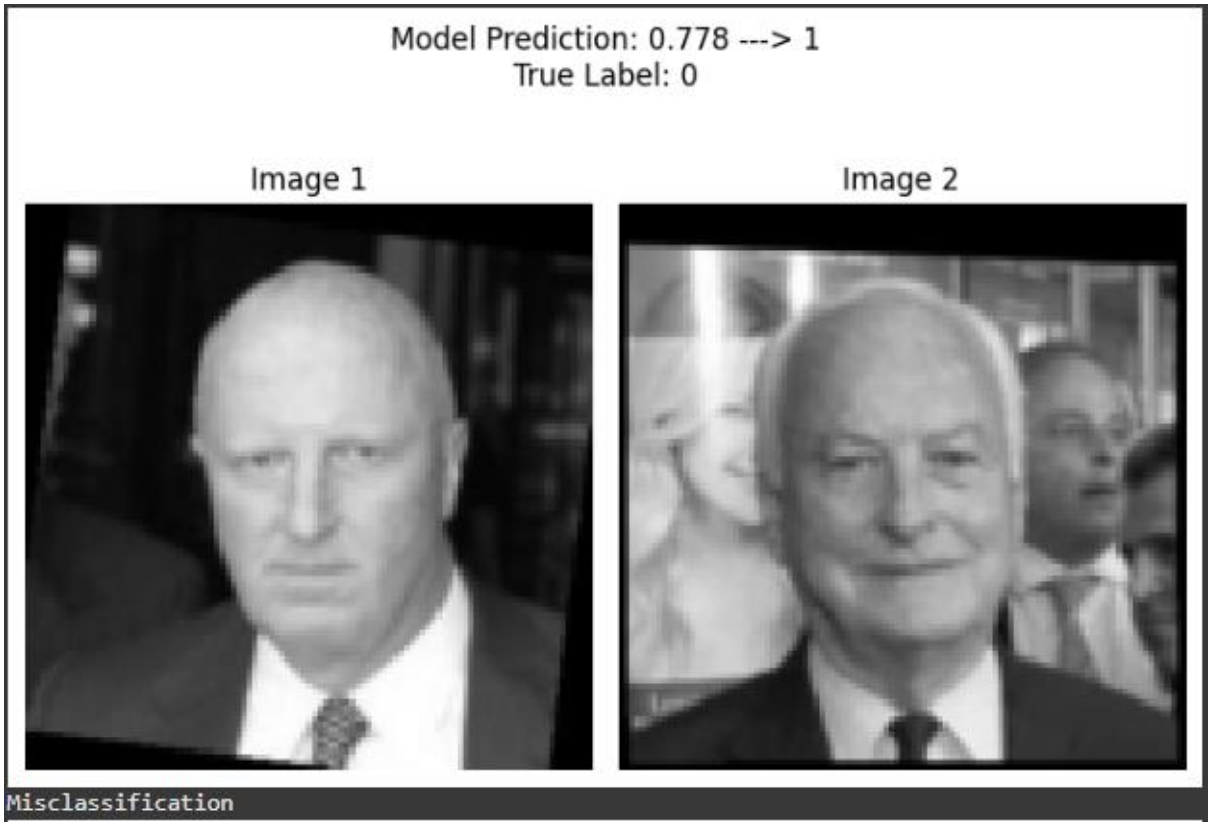
We conducted multiple experiments with various combinations and observed that this particular run yielded better results. Although the validation and test accuracy were slightly lower , the test loss was significantly reduced, and the model exhibited less overfitting overall.

Best Model Features:

- Original paper architecture
- ADAM optimizer
- Kaiming Uniform weight Initialization
- Dropout=0.3
- With Batch Normalization
- Batch size = 64

Examples of accurate and misclassifications Predictions

False-Positive prediction:



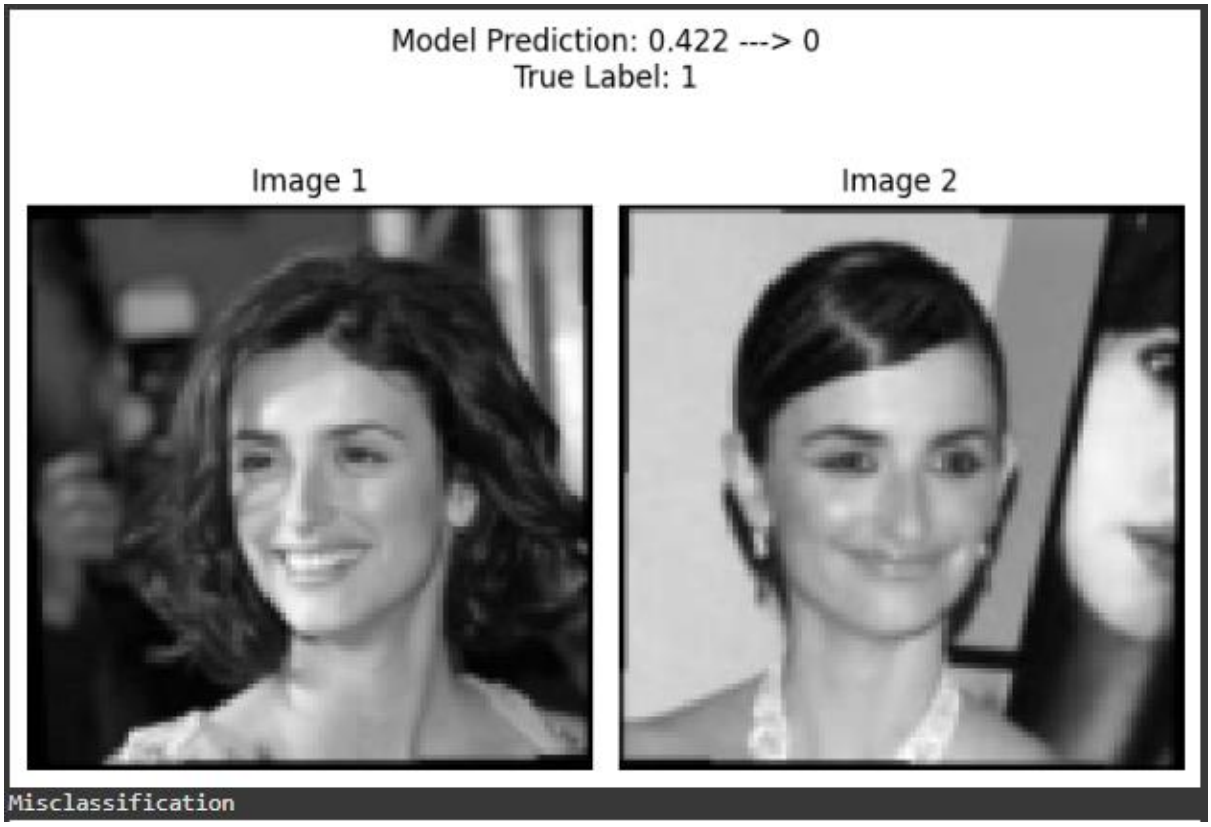
At first glance, the people in the pictures appear similar, but a closer examination reveals that they are different individuals. This similarity may have contributed to the model's error.

Both subjects share similar facial features, including head shape, hairstyle, and formal attire (suit and tie), which likely confused the model.

Furthermore, the consistent lighting, frontal pose, and cluttered background in image 2 may have led the model to extract similar features, resulting in misclassification.

This situation underscores the model's difficulty in recognizing subtle differences between individuals, particularly when superficial similarities are present.

False-Negative prediction:



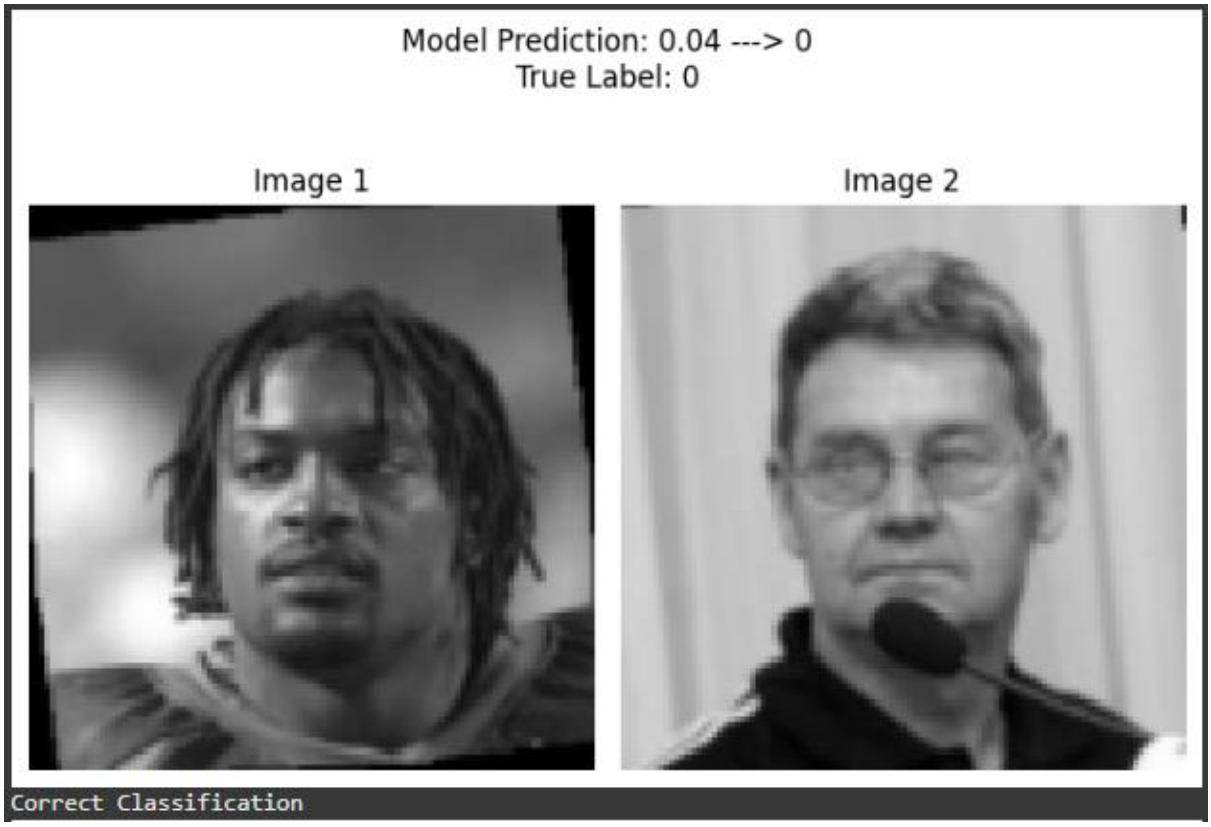
While the true label is 1, the model misclassified them as different. This error may have occurred due to subtle differences between the images:

- **Hair Style**: The hair in the first image is loose and curly, whereas in the second image, it is pulled back and sleek. This significant difference may have misled the model.
- **Pose and Angle**: The person's head tilt and pose differ between the two images, with the first image showing a more natural angle and the second more formal.
- **Facial Features Visibility**: The lighting and hairstyle in the second image partially obscure parts of the face, making it harder for the model to extract consistent features.
- **Expression Variation**: While both images show a smile, the smile in the first image appears broader and more relaxed, whereas the second one is more subtle.

These variations in hairstyle, pose, lighting, and expression likely contributed to the misclassification, highlighting the model's challenge in handling significant visual differences across images of the same person.

It is worth noting that the model's prediction score (0.422) is relatively close to 0.5, indicating a certain level of uncertainty in the classification. This suggests the model did recognize some similarity between the two images but was ultimately unable to confidently classify them as the same person.

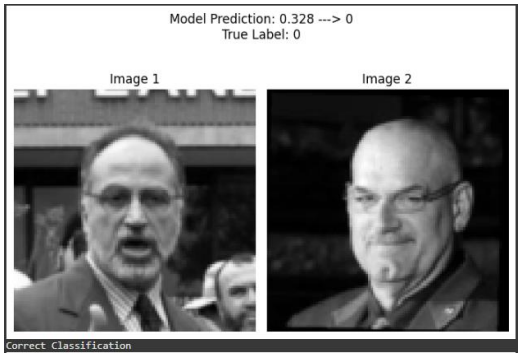
True-Negative prediction:



The model correctly classified the pair as different individuals. This successful classification can be attributed to the following factors:

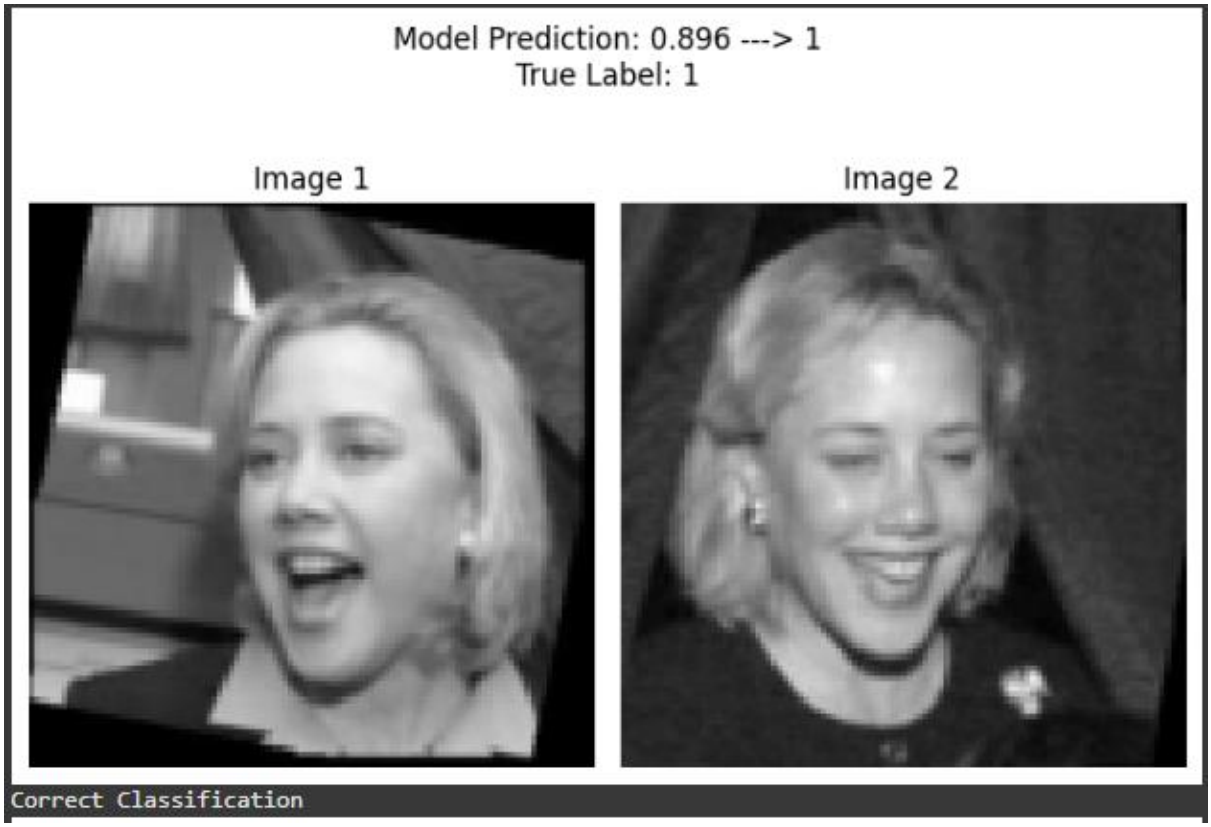
1. **Distinct Facial Features**: The two individuals have noticeably different facial structures and expressions, making it easier for the model to distinguish between them.
2. **Hair and Accessories**: The person in Image 1 has prominent dreadlocks, while the individual in Image 2 has short hair and wears glasses, providing clear distinguishing features.
3. **Pose and Background**: The differences in pose, attire (sports gear vs. casual clothing), and background further helped the model identify the pair as unrelated.

Additionally, the model's confidence score (0.04) shows a strong decision, indicating it recognized the clear dissimilarities between the two images.



It's worth noting that in this image, the model's confidence (0.328) is lower compared to the previous image (0.04), indicating greater uncertainty in this classification. Despite the correct prediction, the model found fewer distinct differences between the individuals here than in the previous example.

True-Positive prediction:



The model correctly classified the pair as the same person with a high confidence score of **0.896**.

This successful classification can be attributed to the following factors:

1. **Facial Features**: Despite the differences in pose and expression, the facial features such as the shape of the eyes, nose, and mouth are clearly similar.
2. **Expression Similarity**: Both images show the subject smiling broadly, which likely helped the model identify similarities.
3. **Lighting and Hair**: The lighting conditions differ slightly, but the hairstyle and overall appearance remain consistent, aiding the model in making a confident prediction.

It is notable that the model succeeded in this classification despite the visible differences between the two images, such as the angle of the face and variations in lighting. This suggests the model's ability to extract robust and consistent features even in challenging scenarios.

Summary

The experiments conducted in this assignment were guided by a process of trial and error to evaluate the performance of the Siamese network for one-shot learning.

Our results demonstrate that key design choices significantly influenced the model's performance:

- **Batch Normalization**: Adding batch normalization after each convolutional layer improved the model's accuracy and convergence.
- **Optimizer Choice**: We observed that the SGD optimizer performed better initially but required careful tuning, while Adam provided more stable results overall.
- **Dropout**: Incorporating dropout successfully reduced overfitting, although its impact depended on the specific value used.

Throughout the experimentation process, we learned that finding the optimal combination of hyperparameters—such as learning rate, batch size, and weight initialization—can be time-consuming, as small changes can have significant impacts on performance.

The misclassification examples revealed interesting insights. In some cases, errors were caused by variations in the images, such as changes in pose, lighting, background details, or facial expressions. These findings highlight the challenge of generalizing well on images with subtle visual differences.

For future work, we suggest experimenting with models that focus solely on facial features, possibly using attention mechanisms to minimize the influence of irrelevant background information. Additionally, training on larger datasets with cleaner and more standardized images could further enhance the model's robustness and accuracy.

This assignment provided valuable learning opportunities:

- We gained hands-on experience in building and training a convolutional neural network (CNN) using PyTorch.
- We deepened our understanding of the architecture and training process of **Siamese networks**, particularly in the context of one-shot image recognition.
- We developed practical skills in hyperparameter tuning, regularization techniques, and evaluating model performance through structured experimentation.

Overall, this project gave us a deeper appreciation for the complexity and potential of CNN-based solutions in real-world image recognition tasks.