**Ariel Siman Tov – 209449821, Tal Klein - 209234103**

# Deep Learning EX 2

**Data Description -** The "Labeled Faces in the Wild-a" (LFW-a) dataset is a collection of labeled grayscale face images designed to facilitate research on Face Recognition in unconstrained environments. In the assignment we were given a specific train/test split of the pairs of images (via text files) so that no image is shared by both sets.

**Dataset Overview:**

- Size - 5749 peoples, each one of them got between 1 to 407 face pictures in
- greyscale. In total there are 13233 images, 1680 people have more than 1 image.
- Train size - 2200 pairs of people. (some of them are the same person)
- Test size - 1000 pairs of people. (some of them are the same person)

For model training, we further divided the original training set into a new training set and validation set, maintaining an 80:20 split ratio, to optimize the training process.

**Example Images**

**Pre-Processing -** The preprocessing stage leverages the LFWpairs function, which parses and organizes image pair data from a specified file. Each pair represents two images (either of the same person or two different people).

**Data processing:**

A. **Data Parsing and Preparation**: Using the parse_pairs_file function, we extract pairs of images from the LFW dataset. The function handles the formatting of image file names for both same-person and different-person pairs, ensuring consistency in tuple generation.

B. **Image Processing**:
   - Images are converted to grayscale using PIL.
   - They are 250×250 pixels for consistency.
   - Normalization and tensor conversion are applied using PyTorch's transforms.

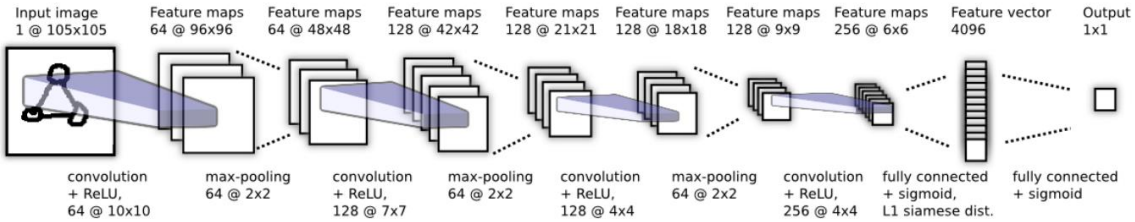
C. **Dataset Management**: The LFWDataset class handles data loading and applies additional transformations:
   - Each sample includes two images and a label (1 for same-person, 0 for different-person pairs).
   - Images are loaded dynamically using their file paths.
   - Default transformations, such as resizing and normalizing, are applied.

D. **Splitting the Dataset**:
   - The training dataset is split into training and validation sets with a 90:10 ratio. The split maintains an equal proportion of identical and different pairs in both sets.
   - The load_data function utilizes PyTorch's DataLoader to create iterable batches of image pairs and labels.

E. **Batch Configuration**: DataLoaders are initialized with a batch size of 32:
   - train_dataloader: Contains training batches of 32 pairs of images with corresponding labels.
   - val_dataloader: Contains validation batches structured similarly to the training set.
   - test_dataloader: Processes the test dataset with the same batch configuration.

## Model architecture:

As described in the paper, we followed the architecture presented in the next illustration:




**Paper General Architecture:** Number of layers: 8 | Activations: Relu | 3 Max polling operations | 1 fully - connected layer.

We modified the kernel size of the convolutional layers to accommodate 250x250 input images. Typical Siamese networks are designed with kernel sizes and pooling methods optimized for smaller images. With larger 250x250 images, smaller kernels struggle to effectively capture broader patterns and structures. By increasing the kernel size, each convolutional layer can cover a more extensive area of the image, enabling it to detect larger patterns more effectively.

```
# CNN architecture
self.cnn = nn.Sequential(
    nn.Conv2d(1, 64, kernel_size=25, stride=1, padding=0),
    nn.BatchNorm2d(64),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(2, stride=2),
    nn.Conv2d(64, 128, kernel_size=18, stride=1, padding=0),
    nn.BatchNorm2d(128),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(2, stride=2),
    nn.Conv2d(128, 128, kernel_size=10, stride=1, padding=0),
    nn.BatchNorm2d(128),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(2, stride=2),
    nn.Conv2d(128, 256, kernel_size=10, stride=1, padding=0),
    nn.BatchNorm2d(256),
    nn.ReLU(inplace=True),
    nn.Flatten()
)

# Fully connected layers
self.fc1 =  nn.Linear(256 * 10 * 10, 4096)
self.fc2 = nn.Linear(4096, 1)
```

**Experimental setup:**

- **Batch size:** In our trial-and-error phase, we tried 32, 64, but 32 produced the best results.
- **BatchNorm:** Using BatchNorm before each Max polling helped to get better results in our experiments.
- **Max epochs:** for each model: 15
- **Loss function:** Binary Cross Entropy by torch.nn because the model classification is 0 or 1.
- **Early stopping criteria**: we used Early stopping regularization (stepLR).
- **Learning rate**: we set with 0.01.
- **Optimizer:** we used SGD and Adam (SGD gave us the best performance).

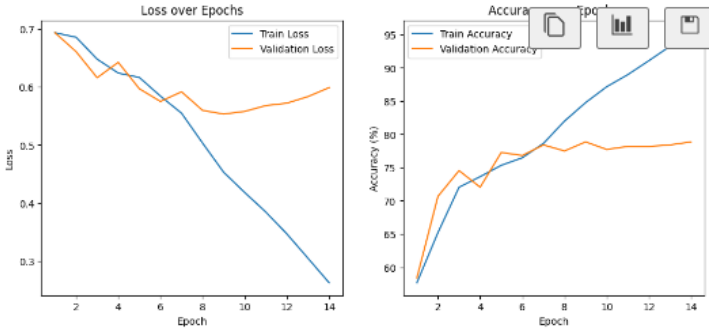
## Experiments:

### Experiment 1:

Setup: Uses SGD optimizer, batch size 32, and no batch normalization.

Purpose: Establishes a baseline to compare the effects of adding batch normalization.

```
Epoch [14/15], Train Loss: 0.2631, Train Acc: 95.28%
Validation Loss: 0.5989, Validation Acc: 78.86%
Early stopping triggered after 14 epochs.
```




```
Test Loss: 0.6633, Test Accuracy: 68.70%
```

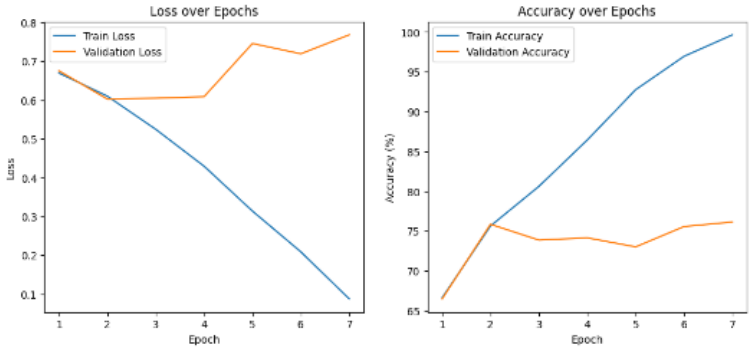
**Experiment 2:**

Setup: Uses SGD optimizer, batch size 32, and batch normalization.

Purpose:

- Directly compares to Experiment 1 to evaluate the impact of batch normalization on training stability and convergence.
- Acts as the base (later) for Experiment 6 with added Dropout layers for regularization.

```
Epoch [7/15], Train Loss: 0.0874, Train Acc: 99.64%
Validation Loss: 0.7678, Validation Acc: 76.14%
Early stopping triggered after 7 epochs.
```




```
Test Loss: 0.6671, Test Accuracy: 72.90%
```

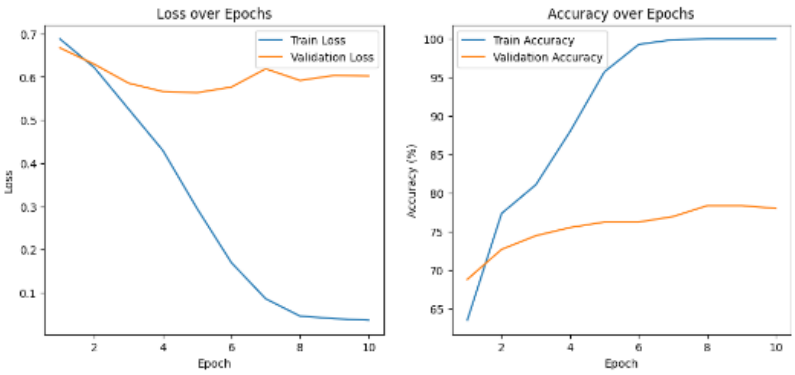
**Experiment 3:**

Setup: Uses SGD optimizer, batch size 64, and batch normalization.

Purpose:

- Compares to Experiment 2 to assess how increasing the batch size affects training performance while keeping the optimizer and batch normalization constant.
- Tests the interaction between batch size and batch normalization.

```
Epoch [10/15], Train Loss: 0.0365, Train Acc: 100.00%
Validation Loss: 0.6021, Validation Acc: 78.01%
Early stopping triggered after 10 epochs.
```




```
Test Loss: 0.5975, Test Accuracy: 72.00%
```

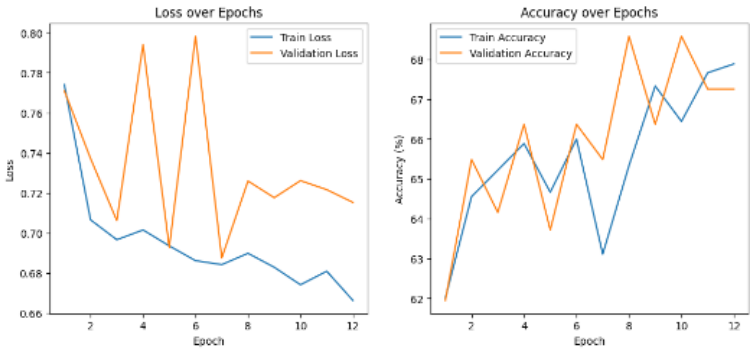
**Experiment 4:**

Setup: Uses Adam optimizer, batch size 32, and batch normalization.

Purpose:

- Directly compares to Experiment 2 to analyze the effect of changing the optimizer from SGD to Adam while maintaining the same batch size and batch normalization.
- Explores optimizer differences with a smaller batch size.

```
Epoch [12/15], Train Loss: 0.6662, Train Acc: 67.89%
Validation Loss: 0.7152, Validation Acc: 67.26%
Early stopping triggered after 12 epochs.
```




```
Test Loss: 0.6876, Test Accuracy: 54.20%
```

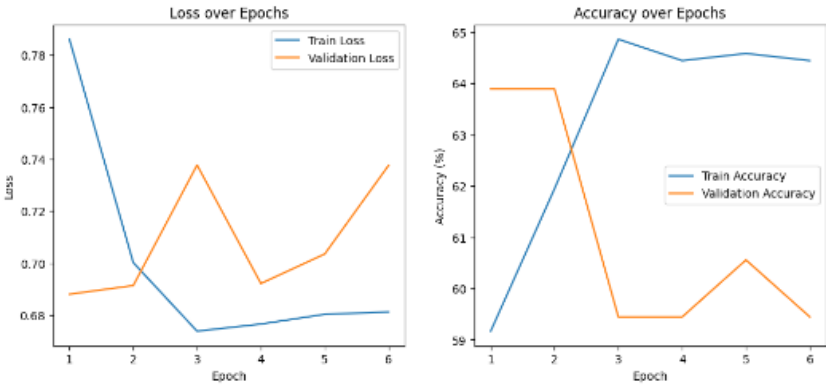
**Experiment 5:**

Setup: Uses Adam optimizer, batch size 64, and batch normalization.

Purpose:

- Builds on Experiment 4 by increasing the batch size, allowing for a comparison of the Adam optimizer's behavior with different batch sizes.
- Compares to Experiment 3 to evaluate the combined effects of optimizer choice and batch size.

```
Epoch [6/15], Train Loss: 0.6812, Train Acc: 64.44%
Validation Loss: 0.7376, Validation Acc: 59.44%
Early stopping triggered after 6 epochs.
```




```
Test Loss: 0.7255, Test Accuracy: 52.20%
```

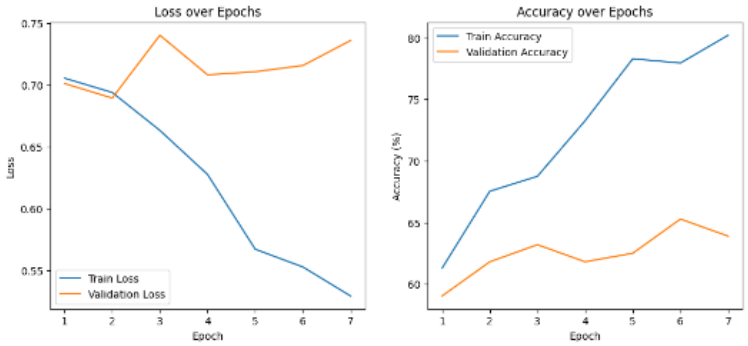
**Experiment 6:**

Setup: Adds Dropout layers (rate = 0.4) after each max-pooling layer based on Experiment 2.

Purpose:

- Extends Experiment 2 by testing the impact of regularization through Dropout on reducing overfitting.
- Serves as a model to test the synergy between batch normalization and Dropout.

```
Epoch [7/15], Train Loss: 0.5291, Train Acc: 80.21%
Validation Loss: 0.7360, Validation Acc: 63.89%
Early stopping triggered after 7 epochs.
```




```
Test Loss: 0.6758, Test Accuracy: 59.00%
```

**Key Comparisons Methodologies:**

**Batch normalization impact:** Compare Experiment 1 vs. Experiment 2 to quantify the benefits of batch normalization in terms of faster convergence and reduced sensitivity to learning rate.

**Batch size effect:** Compare Experiment 2 vs. Experiment 3 (SGD) and Experiment 4 vs. Experiment 5 (Adam) to assess the influence of batch size on performance and generalization.

**Optimizer effect:** Compare SGD (Experiments 2 and 3) vs. Adam (Experiments 4 and 5) to explore optimizer efficiency and generalization, especially for different batch sizes.

**Regularization (dropout):** Compare Experiment 2 vs. Experiment 6 to understand the added impact of Dropout on mitigating overfitting, particularly in conjunction with batch normalization.

This experimental design allows for a step-by-step investigation of how optimizers, batch sizes, batch normalization, and regularization interact to affect model performance. Comparison between experiments facilitates a deeper understanding of their individual and combined impacts, enabling informed decisions about hyperparameter tuning and model architecture.

## Conclusion

| Num_Epochs | batch_size | Optimizer | Time_Taken | Train_Loss | Train_Accuracy | Val_Loss | Val_Accuracy | Test_Loss | Test_Accuracy |
|---|---|---|---|---|---|---|---|---|---|
| 14 | 32 | SGD | 104.835 | 0.506 | 80.11 | **0.598** | 75.422 | 0.663 | 68.7 |
| 7 | 32 | SGD | 47.134 | 0.406 | 85.522 | 0.675 | 73.58 | 0.667 | **72.9** |
| 10 | 64 | SGD | 50.642 | **0.294** | **90.488** | 0.6 | **75.567** | **0.597** | 72 |
| 12 | 32 | Adam | 52.579 | 0.695 | 65.509 | 0.733 | 65.966 | 0.688 | 54.2 |
| 6 | 64 | Adam | 19.484 | 0.7 | 63.241 | 0.708 | 61.111 | 0.725 | 52.2 |
| 7 | 32 | SGD | 18.307 | 0.62 | 72.47 | 0.715 | 62.5 | 0.676 | 59 |

- We can observe from this comparison that Experiment 2 achieved the highest test accuracy, reaching 72.9%.
- In all experiments, we initially set the number of epochs to 15. However, training stopped earlier for each experiment due to early stopping criteria, which monitored validation loss to prevent overfitting and unnecessary computation.

## Possible Explanations for Performance Differences Across Experiments:

### Experiment 1 (SGD, 32 batch size, no batch normalization):

**Performance:** Test Accuracy: 68.7% | Train Accuracy: 80.11% | Train-Test Gap: ~11.4%

**Explanation:**

- Without batch normalization, the model may struggle with stable gradients during training, leading to suboptimal generalization.
- The relatively high train accuracy indicates potential overfitting, as the test accuracy is significantly lower.
- The longer training time (104.835 units) suggests slower convergence due to the lack of batch normalization.

### Experiment 2 (SGD, 32 batch size, batch normalization):

**Performance:** Test Accuracy: 72.9% | Train Accuracy: 85.522% | Train-Test Gap: ~12.6%

**Explanation:**

- Batch normalization improves gradient flow and training stability, allowing the model to achieve better accuracy within fewer epochs (7 epochs vs. 14 in Experiment 1).
- A smaller training time (47.134 units) supports faster convergence.
- The larger train-test gap suggests overfitting, but better performance compared to Experiment_1 indicates that batch normalization reduces sensitivity to hyperparameters.

### Experiment 3 (SGD, 64 batch size, batch normalization):

**Performance:** Test Accuracy: 72% | Train Accuracy: 90.488% | Train-Test Gap: ~18.5%

**Explanation:**

- The larger batch size improves the stability of gradient updates but can sometimes hinder generalization compared to smaller batch sizes.
- The high train accuracy and increased train-test gap suggest overfitting is exacerbated with the larger batch size, despite batch normalization.

- Faster convergence and reduced training time (50.642 units) are consistent with the use of batch normalization.

**Experiment 4 (Adam, 32 batch size, batch normalization):**

**Performance:** Test Accuracy: 54.2% | Train Accuracy: 65.509% | Train-Test Gap: ~11.3%

**Explanation:**

- Adam optimizer is typically faster but may lead to suboptimal generalization when learning rates are not well-tuned.
- The relatively low train and test accuracies suggest that Adam did not converge effectively, possibly due to an inappropriate learning rate or hyperparameter mismatch.
- The slower convergence (52.579 units) compared to other Adam experiments might reflect difficulty in adapting to the dataset.

**Experiment 5 (Adam, 64 batch size, batch normalization):**

**Performance:** Test Accuracy: 52.2% | Train Accuracy: 63.241% | Train-Test Gap: ~11%

**Explanation:**

- Similar to Experiment 4, Adam fails to achieve competitive accuracy, which may be due to an unsuitable learning rate or the optimizer being less effective in this particular task.
- The larger batch size may further hinder generalization, as it averages gradients over a larger sample, potentially smoothing out smaller, beneficial updates.
- The fastest training time (19.484 units) is consistent with Adam's behavior, prioritizing computational efficiency but sometimes at the cost of performance.

**Experiment 6 (SGD, 32 batch size, batch normalization, Dropout):**

**Performance:** Test Accuracy: 59% | Train Accuracy: 72.47% | Train-Test Gap: ~13.5%

**Explanation:**

- Adding Dropout introduces regularization, helping to reduce overfitting but potentially limiting the model's ability to memorize the training data.

- The reduced training accuracy and slightly lower test accuracy suggest that Dropout at a 0.4 rate may be too aggressive, leading to under-utilization of the model's capacity.
- The shortest training time (18.307 units) could indicate that Dropout reduces the model's reliance on redundant features, thereby streamlining training.

## Summary of Key Observations:

1. **Batch Normalization:**
   - Improves convergence speed and generalization (Experiments 2 and 3 compared to Experiment 1).
   - However, combined with a larger batch size (Experiment 3), it can increase overfitting.
2. **Optimizer Choice:**
   - SGD generally outperforms Adam for this task, possibly due to Adam's sensitivity to hyperparameters and tendency to over-adapt to noisy gradients.
3. **Regularization (Dropout):**
   - Regularization through Dropout in Experiment 6 mitigates overfitting but can limit performance when applied too aggressively.
4. **Batch Size:**
   - Smaller batch sizes (Experiments 1, 2, and 4) generally result in better generalization compared to larger batch sizes (Experiments 3 and 5), though at the cost of longer training times.
5. **Preprocessing Choices:**
   - Grayscale Conversion: Converting images to grayscale simplifies computations while retaining sufficient features for facial recognition, which are often shape- and contrast-based rather than color-dependent.
   - Normalization: Ensuring consistent pixel intensity distributions helped improve model convergence and stability during training.

6. **Adaptation for Larger Input Size:**
    - The use of larger kernels (e.g., 25x25, 18x18, 10x10) tailored the architecture to the 250x250 pixel input images. This adjustment allowed the model to capture meaningful spatial features at various scales, from global structures to finer details.

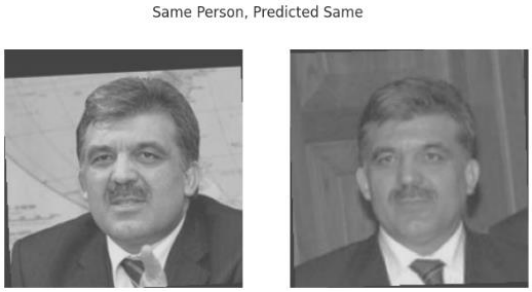
**Future Improvements:**

- Additional experimentation with advanced optimizers (e.g., RMSProp or Lookahead) and data augmentation could further enhance model performance.
- Exploring more sophisticated architectures like ResNet-based Siamese networks might improve feature extraction for one-shot learning tasks.

**Prediction Examples:**

**1) Correct Classifications:**

True-positive example: classify two images of the same person (class 1) for two images of the same person (class 1).




True-negative example: classify as two different people (class 0) for two images of the different people (class 0).

## 2) Misclassifications:

False-negative example: classify two images of the same person (class 1) as two different people (class 0)


False-positive example: classify two different people (class 0) as the same person (class 1).