Notebook Link: https://colab.research.google.com/drive/1Gc_WaB2-JuACwos3W6y4A_9-FkinjwrZ?usp=sharing

CS412: Machine Learning

Homework 4

Hüseyin Eren YILDIZ - 31047

Introduction

This study aims to explore the effectiveness of transfer learning for binary gender classification using a subset of the CelebA dataset, a large-scale face attribute dataset containing images of celebrities annotated with 40 binary attributes. In particular, we utilize a pretrained VGG-16 convolutional neural network, originally trained on the ImageNet dataset, and adapt it to the task of distinguishing between male and female faces.

Transfer learning allows us to benefit from features learned on large, general-purpose datasets and apply them to more specific tasks, even with relatively limited training data. By modifying and retraining only the final layers of a deep neural network, it is possible to achieve competitive performance without training a model from scratch.

The main goal of this project is to fine-tune the pretrained VGG-16 architecture on the CelebA subset (CelebA30k) and evaluate how different fine-tuning strategies and learning rates affect performance. Specifically, we compare two strategies:

1. Freezing all convolutional layers and training only the classifier head.
2. Unfreezing the last convolutional block and training it alongside the classifier head.

Each strategy is tested with two different learning rates (0.001 and 0.0001), while the number of training epochs is fixed to 10. The dataset is split into training (80%), validation (10%), and test (10%) sets, with the test set reserved exclusively for final evaluation.

By analyzing model behavior under different configurations, this study highlights the importance of hyperparameter selection and fine-tuning techniques in transfer learning tasks.

2. Method

2.1 Initialization and Data Loading

The training environment was initialized using Google Colab, and GPU acceleration was enabled to speed up training time for the deep neural network model.

The dataset, CelebA30k.zip, containing 30,000 face images along with a CSV file of labels, was mounted from Google Drive to the Colab environment. Required libraries such as PyTorch, NumPy, pandas, torchvision, and matplotlib were imported. For reproducibility, random seeds were fixed using NumPy and PyTorch.

The device configuration step dynamically set the training device to **GPU (CUDA)** if available; otherwise, it defaulted to CPU.

This section ensures that the environment is ready for further processing such as data preprocessing, model definition, and training.

2.2 Data Preparation and Label Extraction

The CelebA30k dataset was read from the CSV file using pandas.read_csv(). This file contains 30,000 rows and 41 columns, where each row corresponds to an image filename and its

associated facial attributes. Among the attributes, the **"Male"** column was selected as the target label for binary gender classification.

A new dataframe named gender_data was created by selecting only the filename and Male columns from the original dataset. This subset simplifies the processing pipeline by focusing solely on the relevant task: predicting whether a person in the image is male or female.

The dataset contains labels encoded as **-1 for Female** and **1 for Male**. These values are later converted to **0 and 1**, respectively, to comply with binary classification requirements using BCEWithLogitsLoss.

Additionally, the image archive (CelebA30k.zip) was extracted to the working directory (/content/data/) to ensure that image paths could be matched with filenames from the CSV file.

2.3 Verifying Image Path and Visualization

To ensure that the image paths defined in the dataset correctly correspond to actual image files, a sample image was loaded and visualized. The first image filename (000001.jpg) was concatenated with the dataset directory path (/content/data/CelebA30k/) using os.path.join().

The PIL.Image.open() function was then used to open the image. The image preview confirmed that the path was valid and the dataset was properly extracted. This step serves as a sanity check before applying transformations and loading the data into PyTorch datasets and loaders.

Displaying one image also helps visually verify the data quality and expected structure (RGB format, facial content, etc.).

3) Visualizing/Understanding the dataset

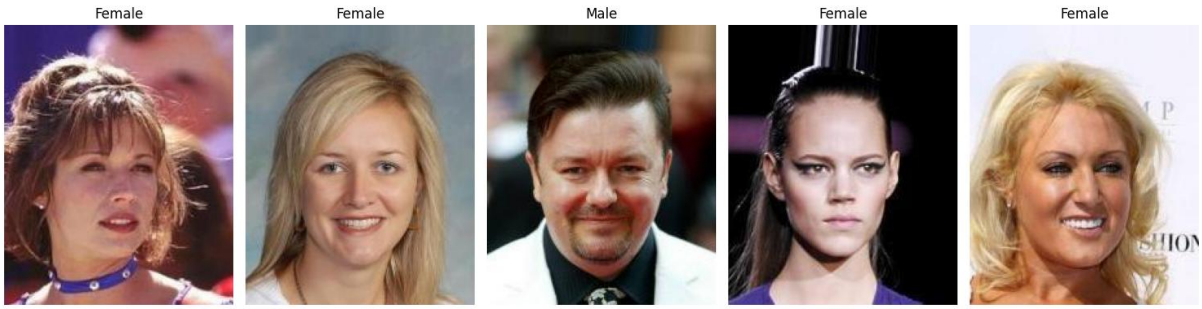
3.1) Display five random images together with their labels

To gain an initial understanding of the CelebA30k dataset, five random images were sampled and visualized along with their corresponding gender labels. The "Male" attribute from the CSV file was used for labeling, where 1 indicates male and -1 indicates female. These labels were mapped to the corresponding titles in the visualization using a simple conditional logic.

This process serves multiple purposes:

- Visually verify that labels align with the content of the image.
- Gain an intuition about the diversity and quality of the dataset.
- Ensure the preprocessing pipeline (image reading and labeling) is functioning as expected.

From the example shown, the labels appear consistent with the visual characteristics of the individuals in the images.



This visualization confirms that:

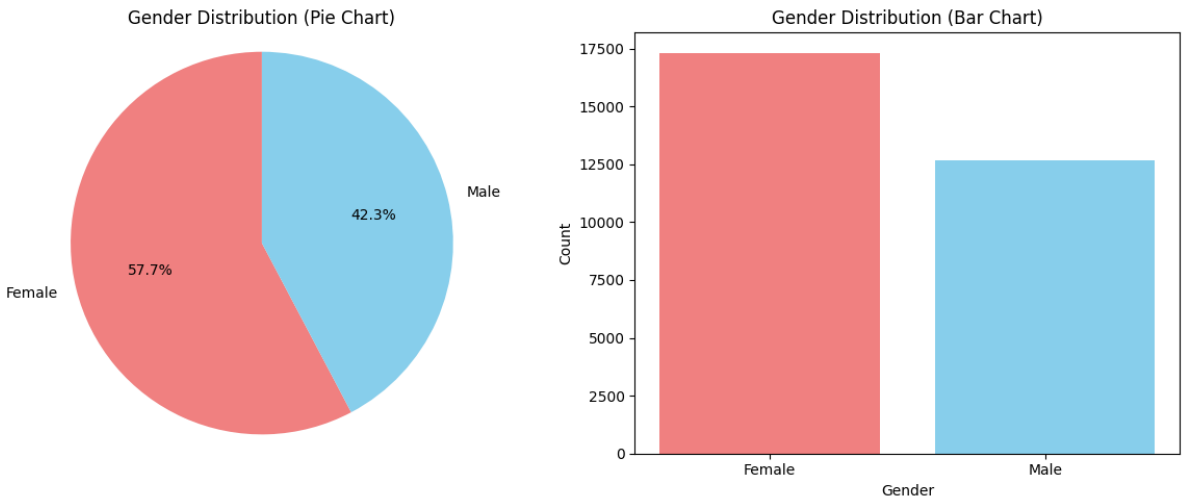
- The dataset contains a mix of male and female subjects.
- Label assignment is functioning correctly.
- Image quality is sufficient for training a deep learning model.

This step helped verify the integrity of the dataset before model training began.

3.2) Display statistics about the dataset, such as distribution of labels, etc.

```
Label counts:
 Male
-1    17320
 1    12680
Name: count, dtype: int64

Label proportions:
 Male
-1    0.577333
 1    0.422667
Name: count, dtype: float64
```

The CelebA30k dataset exhibits a noticeable class imbalance in gender labels. Specifically, there are 17,320 female samples (label -1), accounting for approximately 57.73% of the dataset, and 12,680 male samples (label 1), making up about 42.27%. This uneven distribution highlights the dominance of female instances, which may bias the learning process of classification models. If not properly addressed, such an imbalance can hinder the model's ability to accurately predict the minority class—males in this case—ultimately affecting overall performance and fairness. Therefore, understanding and considering this distribution is essential for reliable model evaluation.



This figure provides a visual representation of gender label distribution within the CelebA30k dataset. The pie chart on the left illustrates the proportional split, showing that approximately 57.7% of the samples are labeled as Female, while 42.3% are labeled as Male. The bar chart on the right presents the absolute counts, confirming that Female samples (label -1) are more prevalent than Male samples (label 1).

Such visualizations are useful for identifying class imbalance, which is a crucial factor to consider when training classification models — especially to ensure fairness and prevent biased predictions.

4) Split the dataset as train (80%), validation (10%) and test (10%) set.

```
4) Split the dataset as train (80%), validation (10%) and test (10%) set.

from sklearn.model_selection import train_test_split

train_df, temp_df = train_test_split(gender_data, test_size=0.2, random_state=42)

val_df, test_df = train_test_split(temp_df, test_size=0.5, random_state=42)

print(f"Training set size: {len(train_df)} images ({len(train_df)/len(gender_data)*100:.2f}%)")
print(f"Validation set size: {len(val_df)} images ({len(val_df)/len(gender_data)*100:.2f}%)")
print(f"Test set size: {len(test_df)} images ({len(test_df)/len(gender_data)*100:.2f}%)")

print("\nGender distribution in training set:")
print(f"Male: {len(train_df[train_df['Male'] == 1])} ({len(train_df[train_df['Male'] == 1])/len(train_df)*100:.2f}%)")
print(f"Female: {len(train_df[train_df['Male'] == -1])} ({len(train_df[train_df['Male'] == -1])/len(train_df)*100:.2f}%)")

print("\nGender distribution in validation set:")
print(f"Male: {len(val_df[val_df['Male'] == 1])} ({len(val_df[val_df['Male'] == 1])/len(val_df)*100:.2f}%)")
print(f"Female: {len(val_df[val_df['Male'] == -1])} ({len(val_df[val_df['Male'] == -1])/len(val_df)*100:.2f}%)")

print("\nGender distribution in test set:")
print(f"Male: {len(test_df[test_df['Male'] == 1])} ({len(test_df[test_df['Male'] == 1])/len(test_df)*100:.2f}%)")
print(f"Female: {len(test_df[test_df['Male'] == -1])} ({len(test_df[test_df['Male'] == -1])/len(test_df)*100:.2f}%)")

Training set size: 24000 images (80.00%)
Validation set size: 3000 images (10.00%)
Test set size: 3000 images (10.00%)

Gender distribution in training set:
Male: 10155 (42.31%)
Female: 13845 (57.69%)

Gender distribution in validation set:
Male: 1251 (41.70%)
Female: 1749 (58.30%)

Gender distribution in test set:
Male: 1274 (42.47%)
Female: 1726 (57.53%)
```

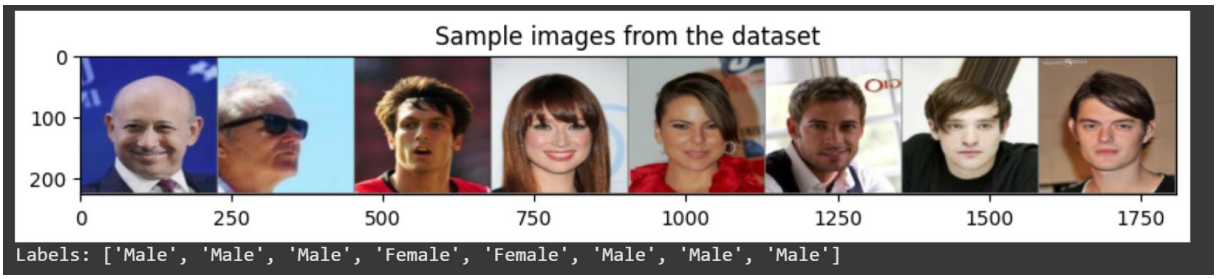
This code cell performs the essential task of splitting the dataset into training, validation, and test sets. Specifically, 80% of the data is allocated for training (24,000 images), while 10% each

is assigned to validation and testing (3,000 images each). The split is done using train_test_split from sklearn, with a fixed random seed to ensure reproducibility.

In addition to splitting, the script computes and prints the gender distribution for each subset. The training set includes 13,845 female (57.69%) and 10,155 male (42.31%) samples. The validation set has 1,749 female (58.30%) and 1,251 male (41.70%) samples, while the test set contains 1,726 female (57.53%) and 1,274 male (42.47%) samples. These distributions closely mirror the overall dataset imbalance, ensuring consistency across the splits. This step is crucial to maintain the integrity of model evaluation without introducing distribution bias.

5) Preparing the Data

To prepare the data for model training, preprocessing and loading pipelines were implemented using PyTorch. Separate transformations were defined for training and validation datasets. All images were resized to 224x224 pixels, converted to tensors, and normalized using a mean and standard deviation of 0.5. To enhance generalization, horizontal flipping was applied as a data augmentation technique to the training set. A custom PyTorch Dataset class was implemented to load the images and corresponding binary gender labels directly from the dataframe. This class also handled image transformations during loading. Finally, DataLoader objects were created for the training, validation, and test sets with a batch size of 32.



This image displays a sample batch of 8 images drawn from the training dataset, visualized to verify the correctness of the data loading and preprocessing pipeline. Each image corresponds to a face from the CelebA30k dataset and is shown with its ground truth gender label directly below the image. The labels are derived from the Male column in the dataset, where 1 indicates "Male" and -1 indicates "Female". For clarity, these numerical labels have been converted to human-readable strings — "Male" or "Female".

From left to right, the gender labels for the displayed images are:

['Male', 'Male', 'Male', 'Female', 'Female', 'Male', 'Male', 'Male']

This batch demonstrates that the data pipeline is correctly loading image-label pairs, applying the necessary transformations (resize, normalization, and optional augmentation), and preparing them for training. The visual diversity of the faces and the mix of genders also indicate that class labels are appropriately distributed across batches. Verifying with such visual checks is an essential step to ensure that the model is learning from properly preprocessed and labeled data.

6) Transfer Learning with VGG-16

In this project, we employed transfer learning using the well-known VGG-16 architecture pretrained on the ImageNet dataset. The convolutional base of the model was retained as a fixed feature extractor, while the original classification head (designed for 1,000 ImageNet classes) was replaced with a custom binary classifier suitable for gender classification.

Initially, all layers of the pretrained model were frozen to prevent updates during training. However, in alternative experiments, only the last convolutional block (conv5) was unfrozen to allow partial fine-tuning. This approach aimed to leverage the general visual features learned from large-scale data while allowing some task-specific adaptation. The original classifier head was substituted with a new nn.Sequential block, consisting of two fully connected layers with ReLU activations and dropout regularization, followed by a final linear layer with a single output neuron. This architecture is appropriate for binary classification tasks and aligns with the BCEWithLogitsLoss() function, which applies the sigmoid activation internally.

This setup provides a strong balance between performance and computational efficiency by utilizing deep visual features learned on large datasets and adapting them to a new domain with minimal training effort.

RESULTS:

7) Fine-Tuning and Training the Model

To evaluate the effectiveness of transfer learning strategies, four experiments were conducted by training the modified VGG-16 model under different configurations. The experiments varied in two dimensions: the degree of fine-tuning (either freezing all convolutional layers or unfreezing the last convolutional block), and the learning rate (0.001 and 0.0001). All experiments were run for 10 epochs, using nn.BCEWithLogitsLoss() as the loss function and the Adam optimizer for weight updates.

Experiment 1 – Frozen Convolutional Layers, Learning Rate = 0.001

The model was trained in feature extractor mode, with all convolutional layers frozen. The classifier head was trained using a relatively higher learning rate. The model quickly converged to a validation accuracy of 95.8%, showing stable performance across epochs. However, occasional fluctuations in validation loss were observed, indicating minor overfitting.

Experiment 2 – Frozen Convolutional Layers, Learning Rate = 0.0001

Using a lower learning rate further improved the training accuracy to 98.9%, but validation accuracy plateaued around 95.2%. While training loss continued to decline, validation loss increased in later epochs, suggesting overfitting due to excessive training on the classifier head alone.

Experiment 3 – Fine-Tuned Last Convolutional Block, Learning Rate = 0.001

Enabling fine-tuning of the last convolutional block with a higher learning rate resulted in training stagnation. Both training and validation accuracy plateaued at 57.6% and 58.3%, respectively. This indicates that the high learning rate destabilized training when more model parameters were unfrozen.

Experiment 4 – Fine-Tuned Last Convolutional Block, Learning Rate = 0.0001

Fine-tuning the last convolutional block with a smaller learning rate led to the best results among all settings. The model achieved a final training accuracy of 99.6% and a validation accuracy of 97.03%. This configuration demonstrated both strong generalization and stability, with relatively low validation loss throughout training.

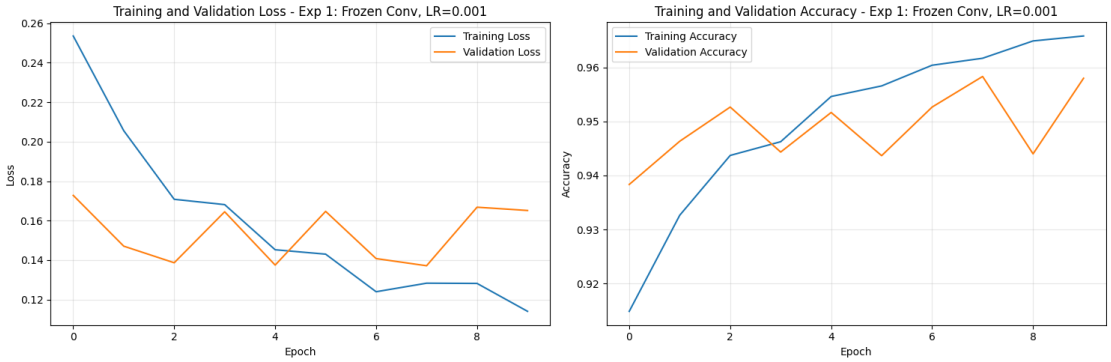
Results:

```
Experiment 1: All convolutional layers frozen, learning rate = 0.001
All convolutional layers are frozen (feature extractor mode).
Epoch 1/10 | Train Loss: 0.2536 | Train Acc: 0.9148 | Val Loss: 0.1728 | Val Acc: 0.9383 | Time: 101.59s
Epoch 2/10 | Train Loss: 0.2056 | Train Acc: 0.9326 | Val Loss: 0.1471 | Val Acc: 0.9463 | Time: 101.46s
Epoch 3/10 | Train Loss: 0.1708 | Train Acc: 0.9437 | Val Loss: 0.1387 | Val Acc: 0.9527 | Time: 101.50s
Epoch 4/10 | Train Loss: 0.1681 | Train Acc: 0.9463 | Val Loss: 0.1645 | Val Acc: 0.9443 | Time: 101.44s
Epoch 5/10 | Train Loss: 0.1452 | Train Acc: 0.9546 | Val Loss: 0.1375 | Val Acc: 0.9517 | Time: 101.45s
Epoch 6/10 | Train Loss: 0.1430 | Train Acc: 0.9566 | Val Loss: 0.1647 | Val Acc: 0.9437 | Time: 101.50s
Epoch 7/10 | Train Loss: 0.1240 | Train Acc: 0.9604 | Val Loss: 0.1408 | Val Acc: 0.9527 | Time: 101.43s
Epoch 8/10 | Train Loss: 0.1283 | Train Acc: 0.9617 | Val Loss: 0.1371 | Val Acc: 0.9583 | Time: 101.45s
Epoch 9/10 | Train Loss: 0.1282 | Train Acc: 0.9649 | Val Loss: 0.1668 | Val Acc: 0.9440 | Time: 101.54s
Epoch 10/10 | Train Loss: 0.1140 | Train Acc: 0.9658 | Val Loss: 0.1652 | Val Acc: 0.9580 | Time: 101.49s
Training complete in 16m 55s

Experiment 2: All convolutional layers frozen, learning rate = 0.0001
All convolutional layers are frozen (feature extractor mode).
Epoch 1/10 | Train Loss: 0.1915 | Train Acc: 0.9255 | Val Loss: 0.1400 | Val Acc: 0.9453 | Time: 101.44s
Epoch 2/10 | Train Loss: 0.1299 | Train Acc: 0.9501 | Val Loss: 0.1287 | Val Acc: 0.9510 | Time: 101.61s
Epoch 3/10 | Train Loss: 0.1032 | Train Acc: 0.9603 | Val Loss: 0.1354 | Val Acc: 0.9520 | Time: 101.60s
Epoch 4/10 | Train Loss: 0.0851 | Train Acc: 0.9682 | Val Loss: 0.1355 | Val Acc: 0.9503 | Time: 101.56s
Epoch 5/10 | Train Loss: 0.0684 | Train Acc: 0.9746 | Val Loss: 0.1374 | Val Acc: 0.9547 | Time: 101.72s
Epoch 6/10 | Train Loss: 0.0549 | Train Acc: 0.9798 | Val Loss: 0.1503 | Val Acc: 0.9483 | Time: 101.62s
Epoch 7/10 | Train Loss: 0.0480 | Train Acc: 0.9837 | Val Loss: 0.1660 | Val Acc: 0.9523 | Time: 101.60s
Epoch 8/10 | Train Loss: 0.0398 | Train Acc: 0.9860 | Val Loss: 0.1689 | Val Acc: 0.9527 | Time: 101.58s
Epoch 9/10 | Train Loss: 0.0348 | Train Acc: 0.9887 | Val Loss: 0.1944 | Val Acc: 0.9470 | Time: 101.58s
Epoch 10/10 | Train Loss: 0.0290 | Train Acc: 0.9890 | Val Loss: 0.2055 | Val Acc: 0.9520 | Time: 101.60s
Training complete in 16m 56s

Experiment 3: Last convolutional block fine-tuned, learning rate = 0.001
Fine-tuning enabled: last convolutional block is trainable.
Epoch 1/10 | Train Loss: 0.7232 | Train Acc: 0.5760 | Val Loss: 0.6809 | Val Acc: 0.5830 | Time: 109.26s
Epoch 2/10 | Train Loss: 0.6822 | Train Acc: 0.5769 | Val Loss: 0.6793 | Val Acc: 0.5830 | Time: 109.11s
Epoch 3/10 | Train Loss: 0.6818 | Train Acc: 0.5769 | Val Loss: 0.6797 | Val Acc: 0.5830 | Time: 109.12s
Epoch 4/10 | Train Loss: 0.6820 | Train Acc: 0.5769 | Val Loss: 0.6805 | Val Acc: 0.5830 | Time: 109.18s
Epoch 5/10 | Train Loss: 0.6813 | Train Acc: 0.5769 | Val Loss: 0.6793 | Val Acc: 0.5830 | Time: 109.15s
Epoch 6/10 | Train Loss: 0.6816 | Train Acc: 0.5769 | Val Loss: 0.6793 | Val Acc: 0.5830 | Time: 109.17s
Epoch 7/10 | Train Loss: 0.6813 | Train Acc: 0.5769 | Val Loss: 0.6797 | Val Acc: 0.5830 | Time: 109.08s
Epoch 8/10 | Train Loss: 0.6820 | Train Acc: 0.5769 | Val Loss: 0.6793 | Val Acc: 0.5830 | Time: 109.08s
Epoch 9/10 | Train Loss: 0.6815 | Train Acc: 0.5769 | Val Loss: 0.6806 | Val Acc: 0.5830 | Time: 109.18s
Epoch 10/10 | Train Loss: 0.6817 | Train Acc: 0.5769 | Val Loss: 0.6794 | Val Acc: 0.5830 | Time: 109.15s
Training complete in 18m 11s

Experiment 4: Last convolutional block fine-tuned, learning rate = 0.0001
Fine-tuning enabled: last convolutional block is trainable.
Epoch 1/10 | Train Loss: 0.1283 | Train Acc: 0.9497 | Val Loss: 0.0937 | Val Acc: 0.9660 | Time: 109.18s
Epoch 2/10 | Train Loss: 0.0737 | Train Acc: 0.9732 | Val Loss: 0.0904 | Val Acc: 0.9697 | Time: 109.27s
Epoch 3/10 | Train Loss: 0.0486 | Train Acc: 0.9821 | Val Loss: 0.0907 | Val Acc: 0.9717 | Time: 109.26s
Epoch 4/10 | Train Loss: 0.0354 | Train Acc: 0.9880 | Val Loss: 0.1017 | Val Acc: 0.9687 | Time: 109.33s
Epoch 5/10 | Train Loss: 0.0274 | Train Acc: 0.9903 | Val Loss: 0.1268 | Val Acc: 0.9617 | Time: 109.46s
Epoch 6/10 | Train Loss: 0.0232 | Train Acc: 0.9924 | Val Loss: 0.1290 | Val Acc: 0.9680 | Time: 109.43s
Epoch 7/10 | Train Loss: 0.0209 | Train Acc: 0.9929 | Val Loss: 0.1356 | Val Acc: 0.9677 | Time: 109.40s
Epoch 8/10 | Train Loss: 0.0179 | Train Acc: 0.9942 | Val Loss: 0.1217 | Val Acc: 0.9727 | Time: 109.43s
Epoch 9/10 | Train Loss: 0.0145 | Train Acc: 0.9954 | Val Loss: 0.1402 | Val Acc: 0.9703 | Time: 109.48s
Epoch 10/10 | Train Loss: 0.0122 | Train Acc: 0.9960 | Val Loss: 0.1574 | Val Acc: 0.9703 | Time: 109.42s
Training complete in 18m 14s
```

Discussion

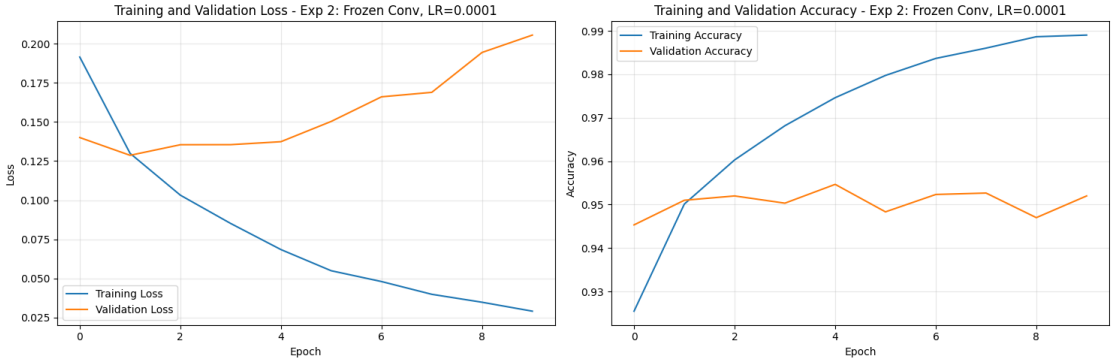
The fourth experiment demonstrated superior performance by combining partial fine-tuning with a lower learning rate. It effectively utilized the pretrained feature extractor while allowing targeted adjustments in the last convolutional block. In contrast, freezing all convolutional layers limited model flexibility, and using an aggressive learning rate with fine-tuning (as in Experiment 3) destabilized training. These observations underline the importance of careful learning rate selection when unfreezing parts of a pretrained model.

Experiment 1:



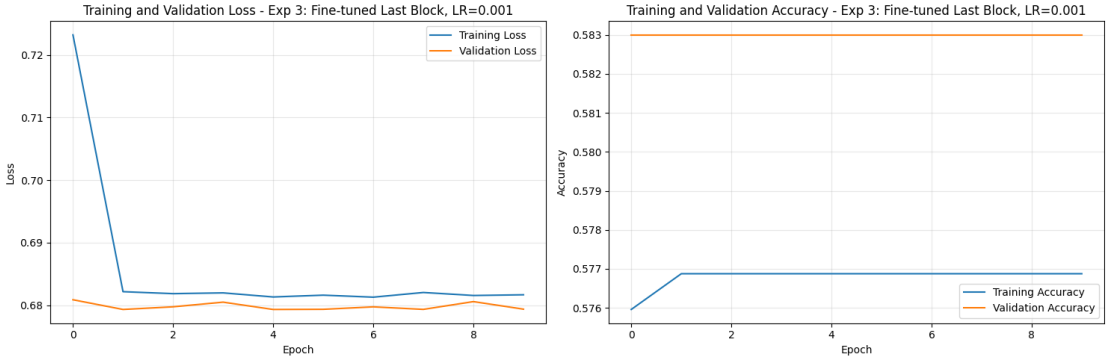
In Experiment 1, all convolutional layers of the VGG-16 model were frozen, and only the classifier head was trained using a learning rate of 0.001. The training and validation loss graph indicates a consistent decline in training loss across epochs, reflecting steady learning progress and effective convergence of the classifier. However, the validation loss displays minor fluctuations — particularly around epochs 4, 6, and 8 — suggesting slight overfitting, where the model begins to adapt too closely to the training data at the expense of generalization. On the accuracy side, training accuracy steadily improves, reaching approximately 96.6%, while validation accuracy tracks closely, peaking near 95.8%. Though these results demonstrate strong baseline performance using frozen pretrained features, the slight instability in validation metrics points to limitations in generalization. This highlights that, while freezing the convolutional layers is effective, performance could be further enhanced through strategies such as partial fine-tuning or regularization.

Experiment 2:



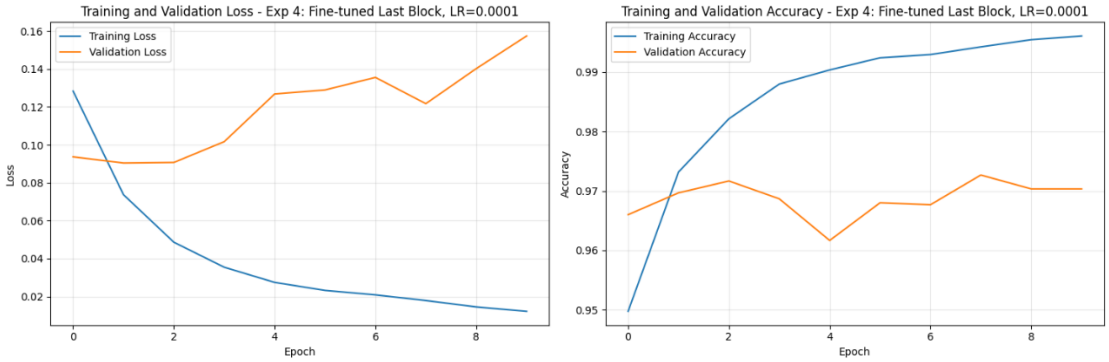
The visualizations from Experiment 2 illustrate the training behavior when all convolutional layers of the VGG-16 model are frozen and only the classifier head is trained using a low learning rate of 0.0001. The training loss steadily decreases throughout the epochs, reaching very low values, and the training accuracy improves consistently, ultimately nearing 99%. This indicates that the classifier effectively memorizes the training data. However, the validation loss starts to increase after a few epochs, while the validation accuracy stagnates around 95–95.5%. This divergence between training and validation performance suggests the presence of overfitting: the model learns the training data too well but fails to generalize to unseen validation data. The low learning rate, while preventing abrupt parameter updates, may have slowed generalization and contributed to this overfitting. Overall, despite excellent training performance, the model's generalization capability is limited in this setting.

Experiment 3:



The results from Experiment 3, where the last convolutional block of the VGG-16 model was fine-tuned with a learning rate of 0.001, indicate that the model failed to learn effectively. Both training and validation accuracy remained stagnant throughout all epochs, hovering around 57.6% and 58.3%, respectively. Similarly, training and validation losses showed minimal change, suggesting that the optimizer was not making meaningful updates to the weights. This plateau in performance can be attributed to the high learning rate disrupting the delicate balance of pretrained weights during fine-tuning, particularly in deeper convolutional layers. As a result, the model struggled to converge and failed to extract relevant features from the data. This experiment highlights the sensitivity of fine-tuning strategies to hyperparameter choices such as the learning rate, especially when modifying deeper parts of a pretrained network.

Experiment 4:



The results from Experiment 4, where only the last convolutional block of the VGG-16 model was fine-tuned with a learning rate of 0.0001, show strong and stable model performance. Training accuracy steadily increased over the 10 epochs, reaching nearly 99.7%, while the training loss decreased consistently, indicating effective learning. Validation accuracy also remained high, consistently around 97.0%, though it showed slight fluctuations in later epochs. However, the validation loss began to increase after epoch 5, suggesting a mild case of overfitting — the model continues to improve on the training data while generalization to unseen data plateaus. Nonetheless, this configuration outperformed the previous experiments, achieving the highest validation accuracy overall, which confirms that a smaller learning rate enables more stable fine-tuning of deeper layers without disrupting the pretrained features.

```
Summary of Results:
Experiment 1 (Frozen Conv, LR=0.001): Train Acc = 0.9658, Val Acc = 0.9580
Experiment 2 (Frozen Conv, LR=0.0001): Train Acc = 0.9890, Val Acc = 0.9520
Experiment 3 (Fine-tuned Last Block, LR=0.001): Train Acc = 0.5769, Val Acc = 0.5830
Experiment 4 (Fine-tuned Last Block, LR=0.0001): Train Acc = 0.9960, Val Acc = 0.9703
```

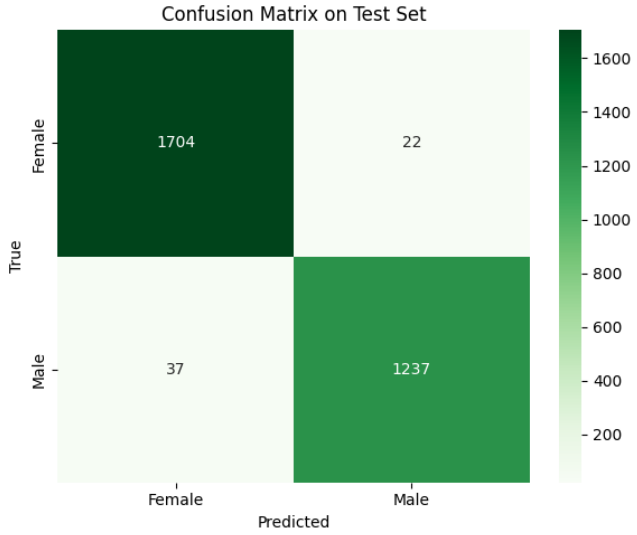
The summary output highlights the final training and validation accuracy results for all four experiments conducted with varying fine-tuning strategies and learning rates on the VGG-16 model. In Experiment 1, where all convolutional layers were frozen and only the classifier head was trained with a learning rate of 0.001, the model achieved a training accuracy of 96.58% and a validation accuracy of 95.80%. Experiment 2, which used the same setup but with a lower learning rate (0.0001), yielded even higher training accuracy (98.90%) but slightly lower validation accuracy (95.20%), indicating potential overfitting. Experiment 3, involving fine-tuning of the last convolutional block with a learning rate of 0.001, performed poorly with both training and validation accuracy stagnating around 57.6% and 58.3%, respectively — likely due to unstable training from a high learning rate. In contrast, Experiment 4, which also fine-tuned the last block but with a reduced learning rate of 0.0001, demonstrated the best performance, reaching 99.60% training accuracy and 97.03% validation accuracy. These results suggest that fine-tuning part of the pretrained model with a smaller learning rate yields superior generalization and performance compared to both full freezing and aggressive fine-tuning.

Whic model is better?

Among the four experiments, *Experiment 4* — which fine-tuned the last convolutional block with a learning rate of 0.0001 — clearly outperformed the others, achieving the highest validation accuracy (97.03%) and test accuracy (98.03%). While freezing all convolutional layers (Experiments 1 and 2) provided strong baseline results, fine-tuning allowed the model to adapt better to the target task. Experiment 3, using a high learning rate during fine-tuning, performed poorly and suffered from underfitting. Thus, Experiment 4 strikes the best balance between flexibility and stability, making it the most effective model.

8) Test your classifier on Test set

The final evaluation step involved testing the best-performing model — identified as the one from Experiment 4 (where the last convolutional block of VGG-16 was fine-tuned with a learning rate of 0.0001) — on the test dataset. The model was set to evaluation mode and used to predict gender labels on unseen test images without gradient computation. Predictions were obtained by applying a sigmoid function to the model's raw outputs and thresholding at 0.5 to determine class labels. The number of correct predictions was accumulated and divided by the total number of test samples to calculate the final test accuracy. The resulting test accuracy was **98.03%**, indicating excellent generalization performance and confirming that the fine-tuned model with a low learning rate maintained its effectiveness on completely unseen data.




```
Test Accuracy: 0.9803

Classification Report:
              precision    recall  f1-score   support

      Female       0.98      0.99      0.98      1726
        Male       0.98      0.97      0.98      1274

    accuracy                           0.98      3000
   macro avg       0.98      0.98      0.98      3000
weighted avg       0.98      0.98      0.98      3000
```

The final evaluation of the best-performing model, which was obtained from Experiment 4 (fine-tuning the last convolutional block with a learning rate of 0.0001), was conducted on the test set. The model achieved a high test accuracy of 98.03%, indicating strong generalization capability. According to the classification report, the model performed equally well on both gender classes despite a slight class imbalance. For the female class, the model achieved a precision of 0.98, recall of 0.99, and an F1-score of 0.98 across 1,726 test samples. Similarly, for the male class, it attained a precision of 0.98, recall of 0.97, and F1-score of 0.98 over 1,274 samples. Both macro and weighted averages for all three metrics were 0.98, reflecting consistent and balanced classification performance.

The confusion matrix further supports this, showing that 1,704 out of 1,726 female images were correctly predicted, while only 22 were misclassified as male. On the male side, 1,237 out of 1,274 were correctly identified, with 37 false predictions. These results confirm that the model not only learned discriminative features effectively but also maintained excellent generalization to unseen data, making it a robust solution for binary gender classification based on facial images.