CS412 - Machine Learning: Homework 4 Report

Barış Pome
31311
Sabanci University

May 3, 2025

Contents

Jupyter Notebook Link: Notebook on Google Drive

1 Introduction

In this homework, we address the problem of binary gender classification on face images from the CelebA dataset subset. We leverage transfer learning on a pretrained VGG-16 convolutional neural network to compare two fine-tuning strategies and two learning rates. The goal is to determine which combination yields the best classification accuracy on unseen test data.

2 Methodology

This section describes dataset preparation, exploratory analysis, dataset splitting, model modifications, and training procedures. Corresponding notebook cells are noted.

2.1 Dataset Preparation

We use the provided `CelebA30k.csv` file to load 30,000 face images annotated for gender. Images are resized to 224 x 224 and normalized using ImageNet mean and standard deviation. (Starter Notebook: Data Loading and Preprocessing cells)

2.2 Dataset Exploration and Visualization

Gender labels are encoded as `-1` for females and `1` for males. The dataset comprises 30,000 images, with 17,320 female images (57.73%) and 12,680 male images (42.27%). See Figure 1 for distribution.

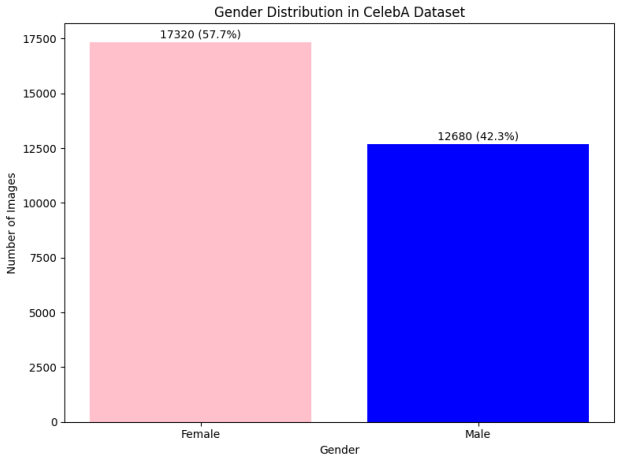

Figure 1: Gender distribution in the CelebA dataset.

To qualitatively assess the data, we randomly selected five images along with their ground-truth labels (Figure 2).

Figure 2: Five randomly selected images from the dataset annotated by gender.

2.3 Dataset Splitting

We partitioned the 30,000-image dataset into training (80%), validation (10%), and test (10%) sets using scikit-learn's `train_test_split` function with a fixed random seed (42) and stratification on gender labels. This resulted in 24,000 training, 3,000 validation, and 3,000 test examples. All splits maintained class proportions: 42.27% male and 57.73% female.

Subset	Number of Examples
Training	24,000
Validation	3,000
Test	3,000

Table 1: Dataset split sizes.

Each subset maintained the original class proportions: 42.27% male and 57.73% female (Table 2).

Subset	Male (%)	Female (%)
Training	42.27	57.73
Validation	42.27	57.73
Test	42.27	57.73

Table 2: Class distribution across data splits.

2.4 Data Preparation and Loading

We implemented the data pipeline using PyTorch. For training, images undergo random horizontal flips, resizing to 224*224, tensor conversion, and normalization (mean and std of 0.5). Validation and test sets use identical resizing and normalization without augmentation. The code is shown below:

```
train_transform = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5))
])

val_transform = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.ToTensor(),
    transforms.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5))
])
```

A custom `CelebADataset` class reads images, applies transforms, and maps labels $\{-1, 1\}$ to $\{0, 1\}$. We instantiated datasets for training (24,000), validation (3,000), and test (3,000) splits and created `DataLoader`s with batch size 64 (shuffle enabled for training). Each loader uses two worker processes.

Table 3 summarizes dataset sizes and batch counts, while Table 4 reports the shape of one training batch.

Subset	**Samples**	**Batches**
Training	24,000	375
Validation	3,000	47
Test	3,000	47

Table 3: Dataset splits and DataLoader batch counts (batch size 64).

Tensor	**Shape**
Images (one batch)	[64, 3, 224, 224]
Labels (one batch)	[64, 1]

Table 4: Shape of a single training batch.

Sample inputs from a training batch are visualized in Figure 3, confirming correct loading and label assignment.

Figure 3: Random sample of training images with their labels after preprocessing.

2.5 Transfer Learning with VGG-16

We leverage a pretrained VGG-16 model from `torchvision.models` as a feature extractor. We remove its original 1,000-class head and replace it with a single-output layer compatible with `BCEWithLogitsLoss`. Two strategies are evaluated:

1. **Head-only Training:** Freeze all convolutional layers; train only the new classifier head.

2. **Last-Block Fine-Tuning:** Unfreeze and train the last convolutional block (features[28:]) along with the classifier head.

Listing 1: VGG-16 transfer learning setup

```
def setup_vgg16(freeze_features=True, fine_tune_last_block=
    False):
     model = models.vgg16(pretrained=True)
     model.classifier[6] = nn.Linear(model.classifier[6].
        in_features, 1)
     if freeze_features:
         for p in model.features.parameters():
             p.requires_grad = False
     if fine_tune_last_block:
         for p in model.features[28:].parameters():
             p.requires_grad = True
     return model

# Head-only: 119,549,953 params (89.04%)
# Last-block:121,909,761 params (90.80%)
```

Strategy	Trainable Params	% of Total
Head-only Training	119,549,953	89.04%
Last-Block Fine-Tuning	121,909,761	90.80%

Table 5: Trainable parameter counts per fine-tuning strategy.

2.6 Model Architecture and Fine-Tuning

We adapt the pretrained VGG-16 model by replacing its classifier head with a binary output layer. Two strategies are evaluated:

1. **Head-only Training:** All convolutional layers frozen; only the new classifier head is trained.

2. **Last-Block Fine-Tuning:** All convolutional layers frozen except the last convolutional block, plus the classifier head.

These strategies test the trade-off between feature reuse and adaptation. (Starter Notebook: Model Definition)

2.7 Training Procedure and Hyperparameters

Models are trained for 10 epochs under two learning rates (0.001 and 0.0001) using `Adam` optimizer and `BCEWithLogitsLoss`. Batch size is set to 64. Performance is monitored on the validation set after each epoch. (Starter Notebook: Training Loop)

2.8 Fine-Tuning and Training the Model

We structured our training and evaluation into modular Python functions:

- `train_one_epoch`: Performs a single training epoch, returning average loss and accuracy.
- `validate_one_epoch`: Evaluates the model on validation data without gradient updates.
- `run_training`: Orchestrates training over multiple epochs, tracks history, and saves best-performing weights.
- `plot_history`: Plots loss and accuracy curves per epoch for train/validation sets.
- `evaluate_on_loader`: Computes F1 scores and confusion matrices for each configuration
- `plot_confusion_matrix`:Plots confusion matrix
- `experiment_suite`: Runs multiple configurations (different fine-tuning strategies and learning rates), collects final accuracies.
- `test_model`: Computes overall accuracy on the held-out test set.

Listing 2 shows the skeleton of these functions:

Listing 2: Training and evaluation functions

```
def train_one_epoch(...):
    # training logic per batch

def validate_one_epoch(...):
    # validation logic per batch

def run_training(...):
    # loops over epochs, tracks history, saves best model

def plot_history(...):
```

```
    # generates loss and accuracy plots

def evaluate_on_loader(model, loader, device):
    # To compute F1 scores and confusion matrices for each
       configuration, we define the following utility that
       collects predictions and labels over a dataset loader
       and prepares them for metric computation:

def plot_confusion_matrix(cm, class_names, title='Confusion␣
   Matrix', cmap='Blues'):
    # generates confusion matrix plots

def experiment_suite(configs, ...):
    # runs each config, returns summary dataframe

def test_model(model, loader, device):
    # computes test-set accuracy
```

Using `experiment_suite`, we evaluated four configurations. The final train/validation accuracies are summarized in Table 6.

Configuration	Train Acc (%)	Val Acc (%)
Head-only, lr=0.001, Adam	98.04	95.17
Head-only, lr=0.0001, Adam	99.24	95.37
Head+conv5_3, lr=0.001, Adam	98.68	95.40
Head+conv5_3, lr=0.0001, Adam	99.55	96.43

Table 6: Final training and validation accuracies after 10 epochs.

2.9 Test Set Evaluation

The best-performing configuration (*Head+conv5_3, lr=0.0001, Adam*) was retrained with its best weights and then evaluated on the test set. It achieved an accuracy of 96.43% as shown in Listing 3.

Listing 3: Test-set evaluation

```
test_accuracy = test_model(best_model, test_loader, device)
# Output:
# Best Validation Accuracy: 0.9667
# Test Accuracy: 0.9643 (2893/3000)
```

3 Results

This section presents the performance of each configuration across training and validation metrics. We provide epoch-level loss and accuracy curves, confusion matrices, and a summary bar chart for a comparative overview.

3.1 Loss and Accuracy Curves by Configuration

Each configuration was trained for 10 epochs. The following figures show how the loss and accuracy evolved over epochs on both the training and validation sets.

Figure 4: Loss curves (Head-only, LR=0.001).

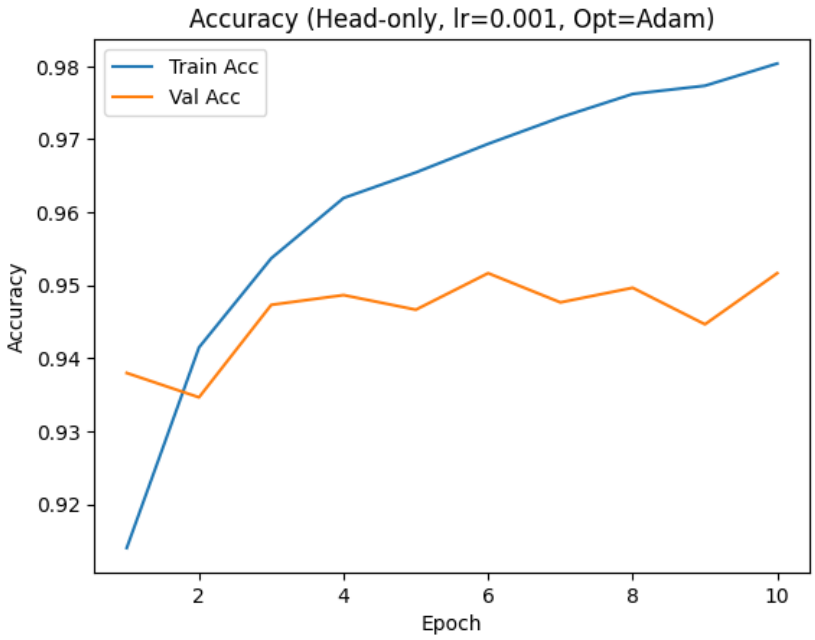

Figure 5: Accuracy curves (Head-only, LR=0.001).

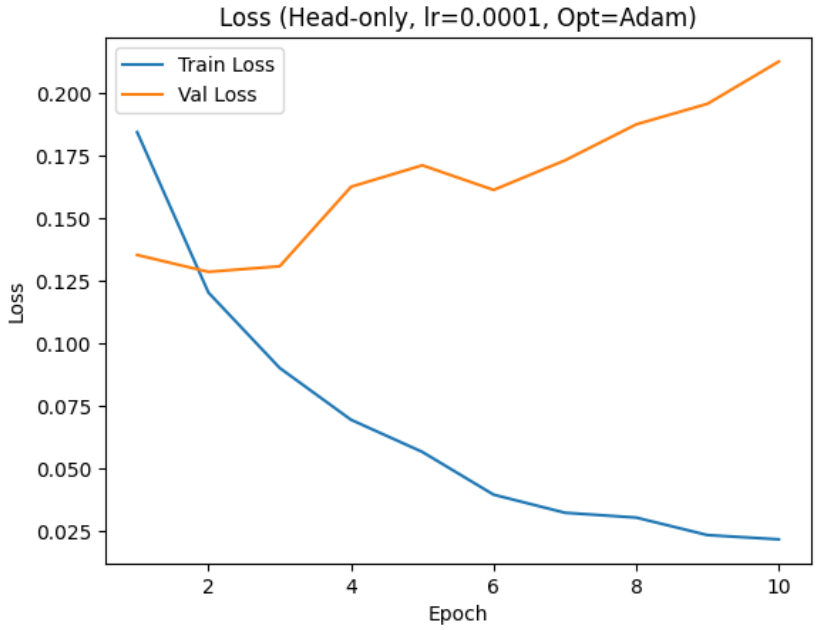

Figure 6: Loss curves (Head-only, LR=0.0001).

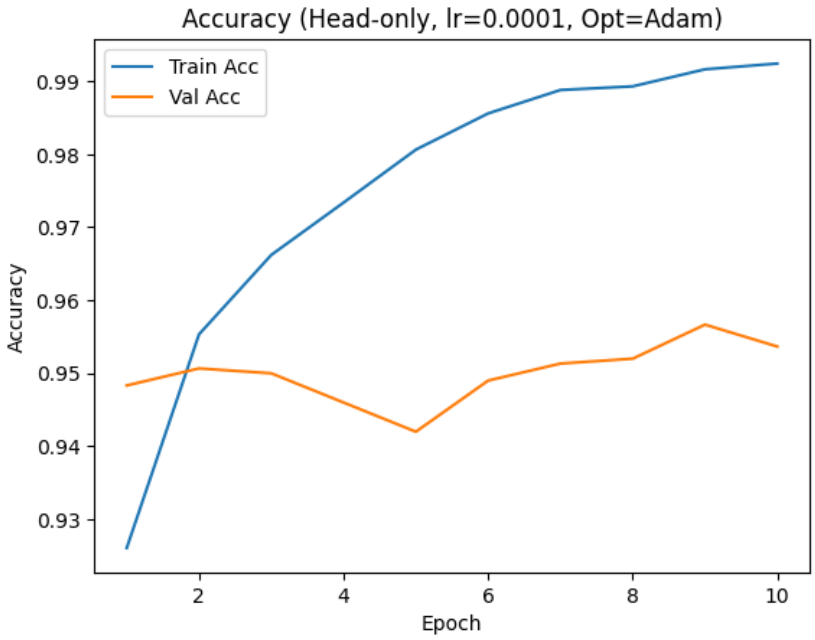

Figure 7: Accuracy curves (Head-only, LR=0.0001).

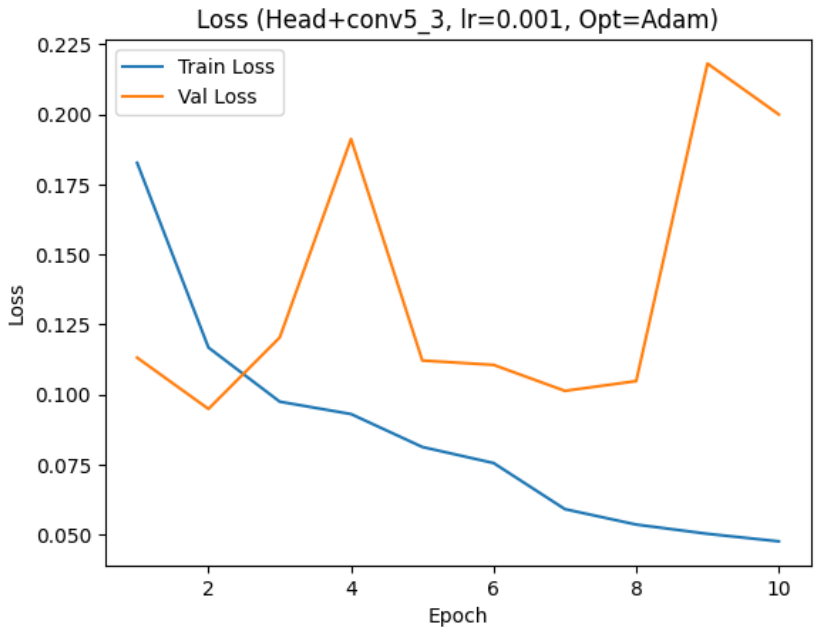

Figure 8: Loss curves (Last-Block Fine-Tune, LR=0.001).

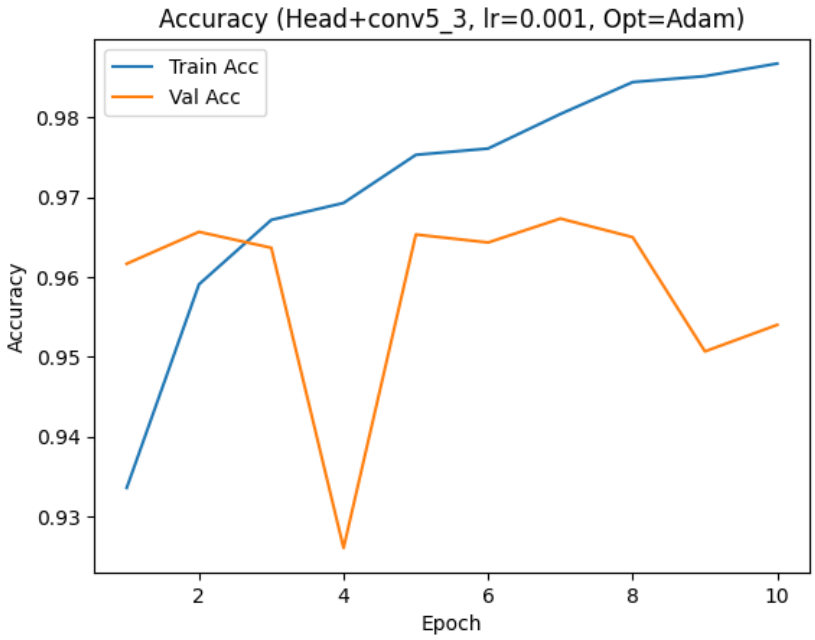

Figure 9: Accuracy curves (Last-Block Fine-Tune, LR=0.001).

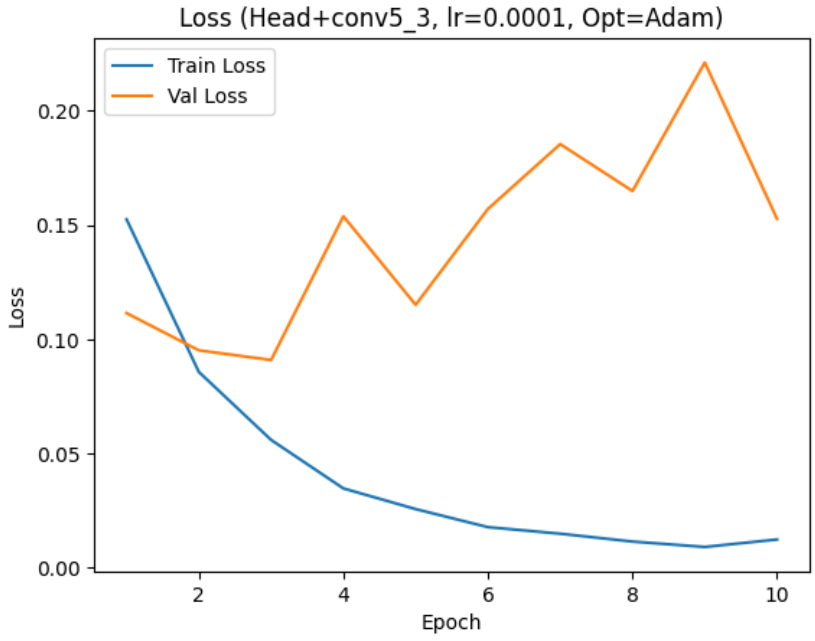

Figure 10: Loss curves (Last-Block Fine-Tune, LR=0.0001).

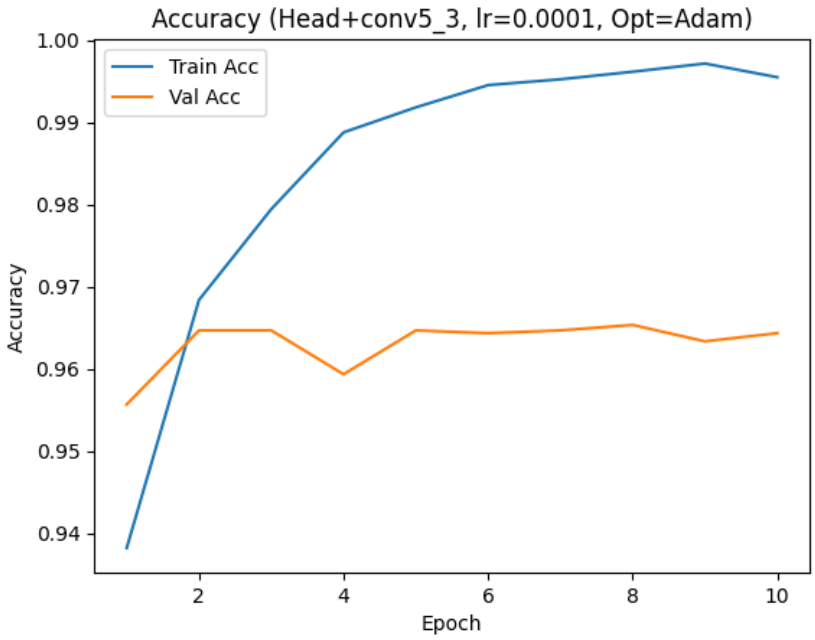

Figure 11: Accuracy curves (Last-Block Fine-Tune, LR=0.0001).

3.2 Confusion Matrices

The confusion matrices below show prediction performance on the test set for each configuration.

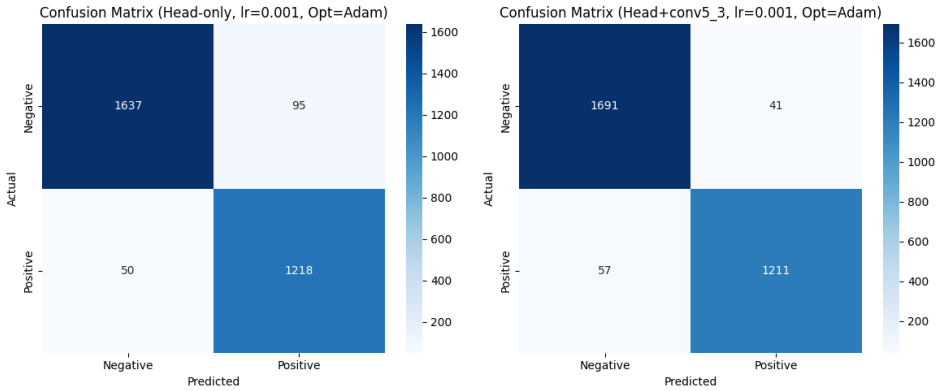

Figure 12: Confusion matrices (LR=0.001): Left – Head-only, Right – Fine-tune last block.

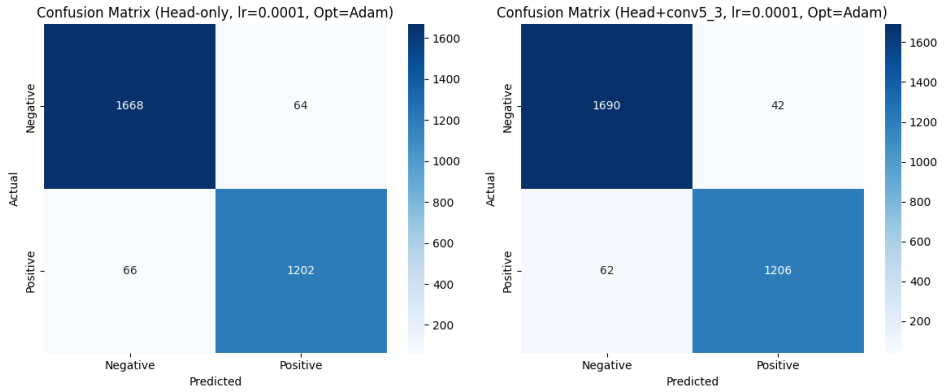

Figure 13: Confusion matrices (LR=0.0001): Left – Head-only, Right – Fine-tune last block.

3.3 Test Set Performance

Configuration	**Accuracy (%)**	**F1 Score**
Head-only, LR=0.001	95.17	0.9438
Fine-tune last block, LR=0.001	95.40	0.9611
Head-only, LR=0.0001	95.36	0.9487
Fine-tune last block, LR=0.0001	96.43	0.9587

Table 7: Classification performance on the test set for each experimental setup.

3.4 Overall Summary

The figure below summarizes final training and validation accuracies for each experimental configuration:

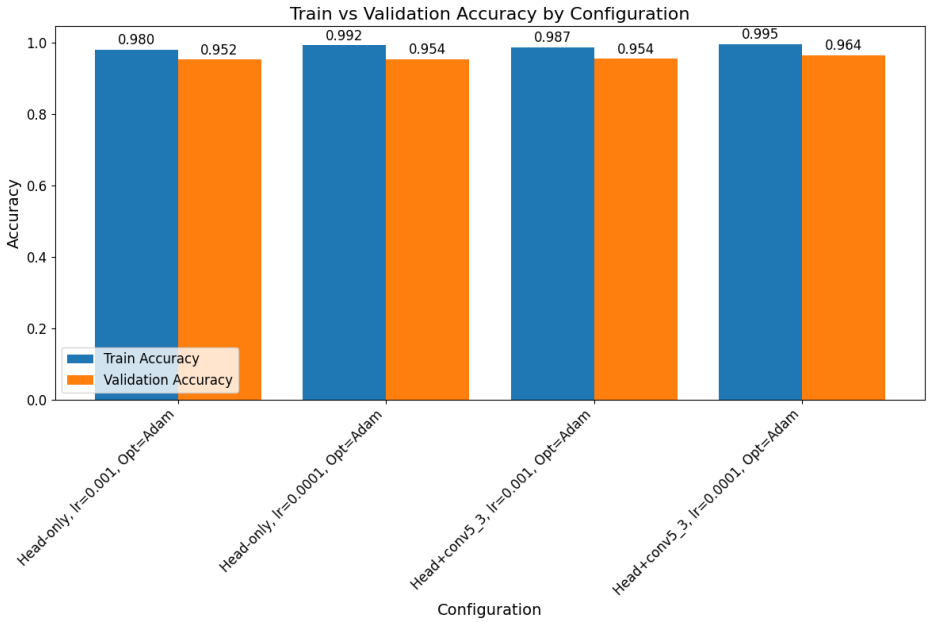

Figure 14: Comparison of training and validation accuracy across all configurations.

4 Discussion

The experimental results highlight the significant influence of both the **fine-tuning strategy** and **learning rate** on model performance.

The configuration that achieved the best results involved *fine-tuning the last convolutional block* of the VGG-16 model with a *learning rate of 0.0001*, resulting in a **test accuracy of 96.43%** and a strong **F1 score of 0.9587**. This suggests that allowing part of the convolutional base to adapt to the gender classification task helps the model better capture task-specific, high-level visual patterns.

In contrast, *head-only training*, where the convolutional base is frozen and only the classifier head is trained, yielded slightly lower performance. While this approach trains faster and requires fewer resources, it lacks the flexibility to tailor the feature representations for the new task. For instance, the head-only configuration with learning rate 0.001 achieved a test accuracy of 95.17% and an F1 score of 0.9438.

Another key observation is the role of *learning rate*. A smaller value (0.0001) generally resulted in better generalization, particularly when fine-tuning part of the model. This is likely because smaller steps avoid disrupting the pretrained weights too aggressively. **Key takeaways:**

- Fine-tuning just the classifier head is fast but less effective.
- Unfreezing even a small part of the convolutional base (e.g., the last block) significantly improves performance.
- Lower learning rates (e.g., 0.0001) help maintain the benefits of pretrained weights and lead to better generalization.

Overall, this study demonstrates that carefully designed fine-tuning strategies and optimization choices can substantially boost performance in transfer learning tasks.

5 Conclusion

This project demonstrated the effectiveness of **transfer learning** in addressing a real-world computer vision task: gender classification using face images from the CelebA dataset.

By leveraging a pretrained `VGG-16` model and adjusting its architecture for binary classification, we explored two approaches:

- Training only the new classification head.
- Fine-tuning the last convolutional block in addition to the head.

Our results showed that while head-only training performs reasonably well, allowing parts of the pretrained network to adapt to the new task significantly improves accuracy and F1 score. The best configuration (last block fine-tuned with a learning rate of 0.0001) achieved **96.43% test accuracy** and an F1 score of **0.9587**.

These findings emphasize the importance of:

- Choosing appropriate layers to fine-tune,
- Selecting stable learning rates,
- Balancing reuse of pretrained features with task-specific adaptation.

In conclusion, this assignment deepened our understanding of CNN-based classification, model fine-tuning, and practical ML workflows using PyTorch. It also highlighted how transfer learning can yield strong performance with relatively modest resources when careful tuning and experimentation are applied.