https://colab.research.google.com/drive/1OI8yg9ugFXvlth92oNHxoNiGP0v83EWQ?usp=sharing

CS412 - Machine Learning

Homework 4: Transfer Learning for Gender Classification using CelebA

Name: Korhan Erdoğdu

Student ID: 30838

Course: CS412 - Machine Learning

Instructor: Berrin Yanıkoğlu

Due Date: May 5, 2025

1. Introduction

The objective of this homework was to apply transfer learning to perform binary gender classification on facial images using a subset of the CelebA dataset. We leveraged the CelebA30k subset, which includes 30,000 celebrity face images labeled with binary gender information. Because training a convolutional neural network from scratch requires a large dataset and high computational cost, we utilized the VGG-16 model pretrained on the ImageNet dataset. This model was adapted for our binary classification task through architectural modification and fine-tuning.

This assignment allowed us to gain practical experience with real-world datasets, image preprocessing, and the implementation of transfer learning using deep neural networks in PyTorch. It also enabled experimentation with multiple training strategies and learning rates, providing valuable insight into their effects on model performance. We ultimately trained multiple models and evaluated them on a held-out test set, selecting the best-performing configuration based on validation accuracy.

2. Methodology

2.1 Dataset and Preprocessing

We used the CelebA30k dataset, which contains 30,000 RGB images of celebrity faces and a CSV file with 40 attribute annotations, including the binary gender label "Male". In the original dataset, the label was `1` for male and `-1` for female. However, for compatibility with PyTorch's `BCEWithLogitsLoss`, which expects targets in the range `[0.0, 1.0]`, the labels were converted to `1` (male) and `0` (female) during dataset preprocessing.

The dataset was split into three parts:

- **Training Set:** 80% (24,000 images)
- **Validation Set:** 10% (3,000 images)
- **Test Set:** 10% (3,000 images)

We ensured that the splits preserved class balance using stratified sampling. The test set was strictly held out and only used for final evaluation.

For all images:

- Resolution was resized to **224×224 pixels** (as expected by VGG-16).
- Normalization used ImageNet's mean and standard deviation.
- Data augmentation (horizontal flipping) was applied to the training set to enhance generalization.
- RGB transformation was also applied.

2.2 Model Architecture and Transfer Learning Setup

We adopted the **VGG-16 architecture** from torchvision.models, which was pretrained on ImageNet. Since the original model was designed for 1000-class classification, we removed its final fully connected layer and replaced it with a **binary classifier**: nn.Linear(4096, 1).

We followed standard transfer learning practices:

- **No Sigmoid** was applied to the final layer output, because BCEWithLogitsLoss() combines Sigmoid and BCELoss in a numerically stable way.

2.3 Fine-Tuning Strategies

We tested two fine-tuning strategies:

1. **Head-Only Training:**
    - All convolutional layers of VGG-16 were frozen (requires_grad = False).
    - Only the newly added classification head was trained.
    - Advantage: Faster training with fewer parameters.
    - Limitation: Model cannot adapt feature extraction to the target task.
2. **Last-Block + Head Fine-Tuning:**
    - Only the **last convolutional block** (Conv5) and the classification head were unfrozen.
    - Earlier layers remained frozen.
    - Advantage: Allows adaptation of high-level features to the new domain.
    - More computationally intensive but offers improved performance.

2.4 Training Procedure

Each configuration was trained for **10 epochs**. We experimented with:

- **Learning rates:** 0.001 and 0.0001
- **Optimizer:** Adam
- **Loss function:** BCEWithLogitsLoss
- **Batch size:** 64

At each epoch:

- Training loss and accuracy were computed.
- Validation loss and accuracy were evaluated after each epoch to monitor performance and detect overfitting.

2.5 Best Model Selection Process

We trained a total of **4 models** across all configurations:

Strategy	LR (Learning Rate)
Head-only	0.001
Head-only	0.0001
Last block + head	0.001
Last block + head	0.0001

After each training session:

- **The model with the best validation accuracy** was selected as the best candidate.
- The best-performing model's weights were saved using `torch.save(...)` and reloaded for final evaluation on the **unseen test set**.

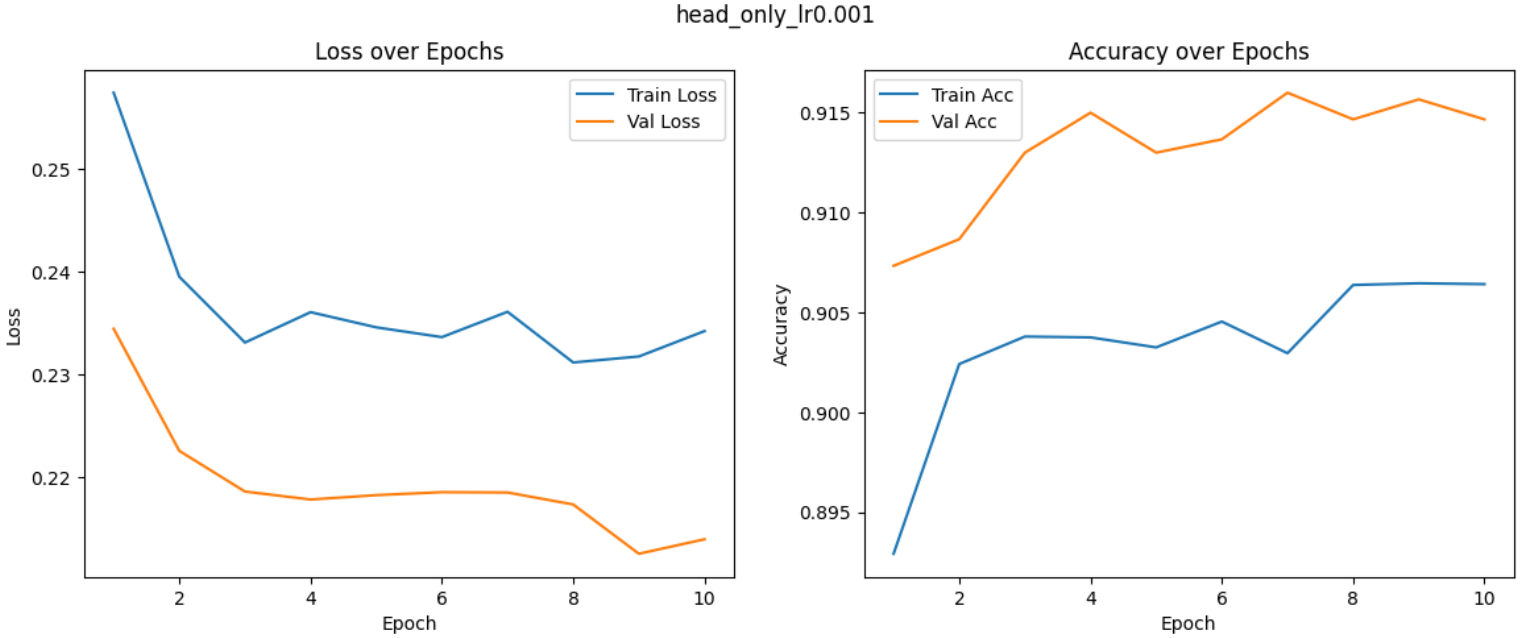
We then passed the test set through the selected model to compute final **accuracy**, **confusion matrix**, and a **classification report** with precision, recall, and F1-score.

3. Results

3.1 Summary Table

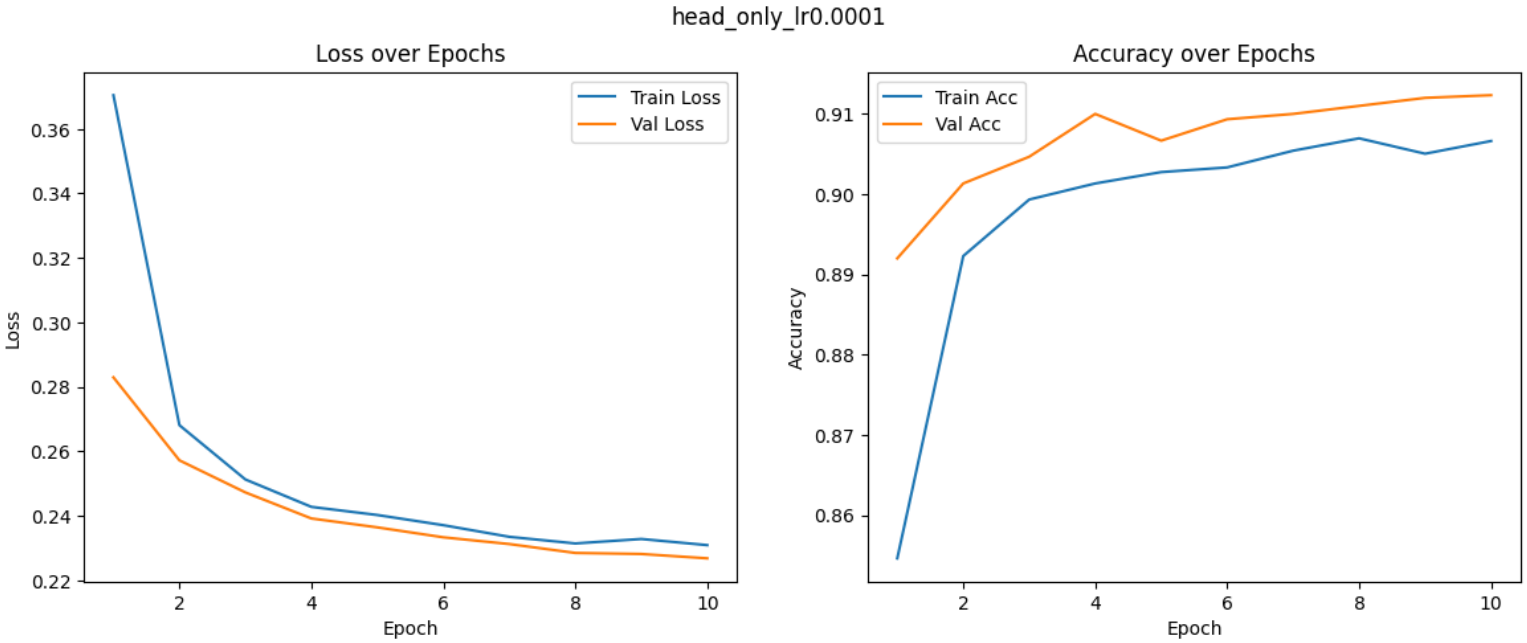
- **Fine-Tuning Strategy:** Train Classifier Only, **Learning Rate:** 0.001

```
Epoch 1/10 | Train Loss: 0.2574, Acc: 0.8929 | Val Loss: 0.2345, Acc: 0.9073
Epoch 2/10 | Train Loss: 0.2395, Acc: 0.9024 | Val Loss: 0.2226, Acc: 0.9087
Epoch 3/10 | Train Loss: 0.2331, Acc: 0.9038 | Val Loss: 0.2186, Acc: 0.9130
Epoch 4/10 | Train Loss: 0.2361, Acc: 0.9038 | Val Loss: 0.2179, Acc: 0.9150
Epoch 5/10 | Train Loss: 0.2346, Acc: 0.9032 | Val Loss: 0.2183, Acc: 0.9130
Epoch 6/10 | Train Loss: 0.2337, Acc: 0.9045 | Val Loss: 0.2186, Acc: 0.9137
Epoch 7/10 | Train Loss: 0.2361, Acc: 0.9030 | Val Loss: 0.2185, Acc: 0.9160
Epoch 8/10 | Train Loss: 0.2312, Acc: 0.9064 | Val Loss: 0.2174, Acc: 0.9147
Epoch 9/10 | Train Loss: 0.2318, Acc: 0.9065 | Val Loss: 0.2126, Acc: 0.9157
Epoch 10/10 | Train Loss: 0.2342, Acc: 0.9064 | Val Loss: 0.2140, Acc: 0.9147
✅ Best model saved as: best_model_head_only_lr0.001.pth (Val Acc: 0.9160)
```



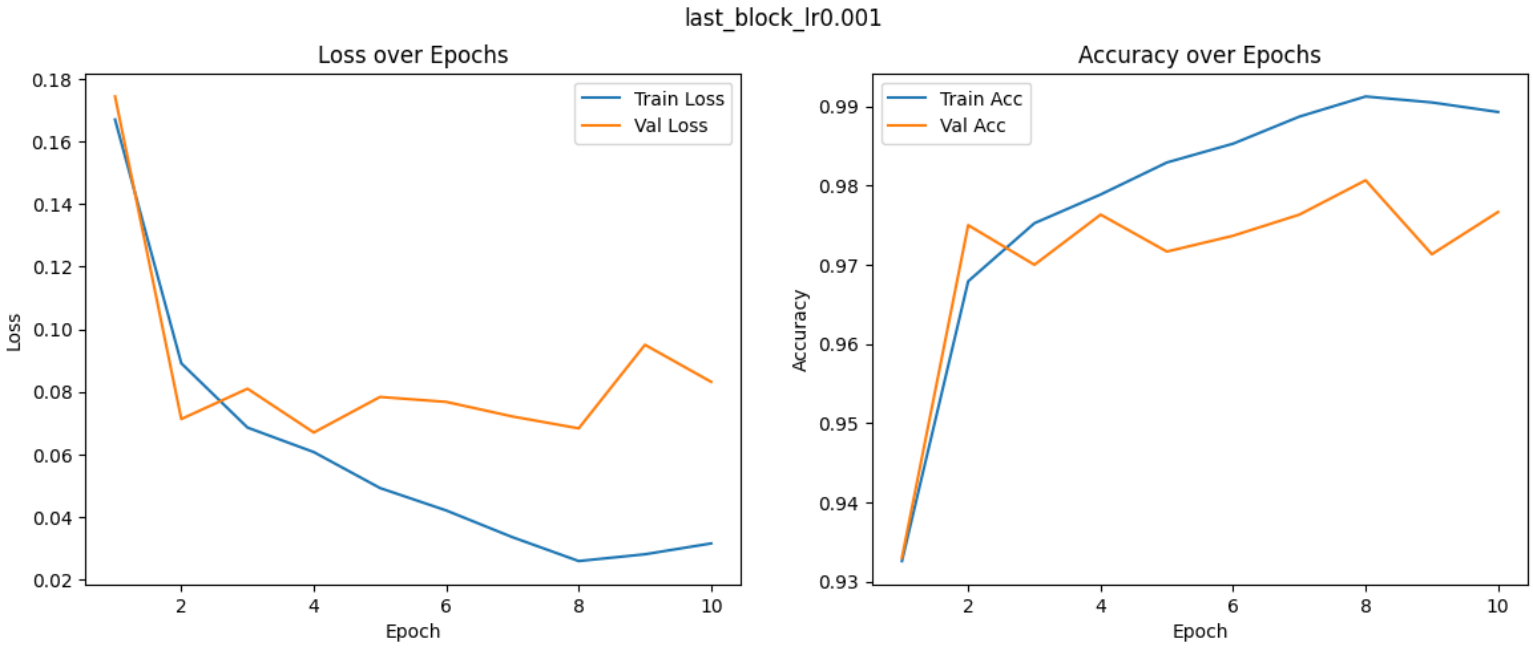
- **Fine-Tuning Strategy:** Train Classifier Only,
  Learning Rate: 0.0001

```
Epoch 1/10 | Train Loss: 0.3705, Acc: 0.8546 | Val Loss: 0.2829, Acc: 0.8920
Epoch 2/10 | Train Loss: 0.2681, Acc: 0.8923 | Val Loss: 0.2572, Acc: 0.9013
Epoch 3/10 | Train Loss: 0.2513, Acc: 0.8993 | Val Loss: 0.2472, Acc: 0.9047
Epoch 4/10 | Train Loss: 0.2428, Acc: 0.9013 | Val Loss: 0.2392, Acc: 0.9100
Epoch 5/10 | Train Loss: 0.2402, Acc: 0.9028 | Val Loss: 0.2364, Acc: 0.9067
Epoch 6/10 | Train Loss: 0.2371, Acc: 0.9033 | Val Loss: 0.2333, Acc: 0.9093
Epoch 7/10 | Train Loss: 0.2335, Acc: 0.9054 | Val Loss: 0.2312, Acc: 0.9100
Epoch 8/10 | Train Loss: 0.2314, Acc: 0.9070 | Val Loss: 0.2285, Acc: 0.9110
Epoch 9/10 | Train Loss: 0.2328, Acc: 0.9050 | Val Loss: 0.2281, Acc: 0.9120
Epoch 10/10 | Train Loss: 0.2309, Acc: 0.9066 | Val Loss: 0.2268, Acc: 0.9123
✅ Best model saved as: best_model_head_only_lr0.0001.pth (Val Acc: 0.9123)
```



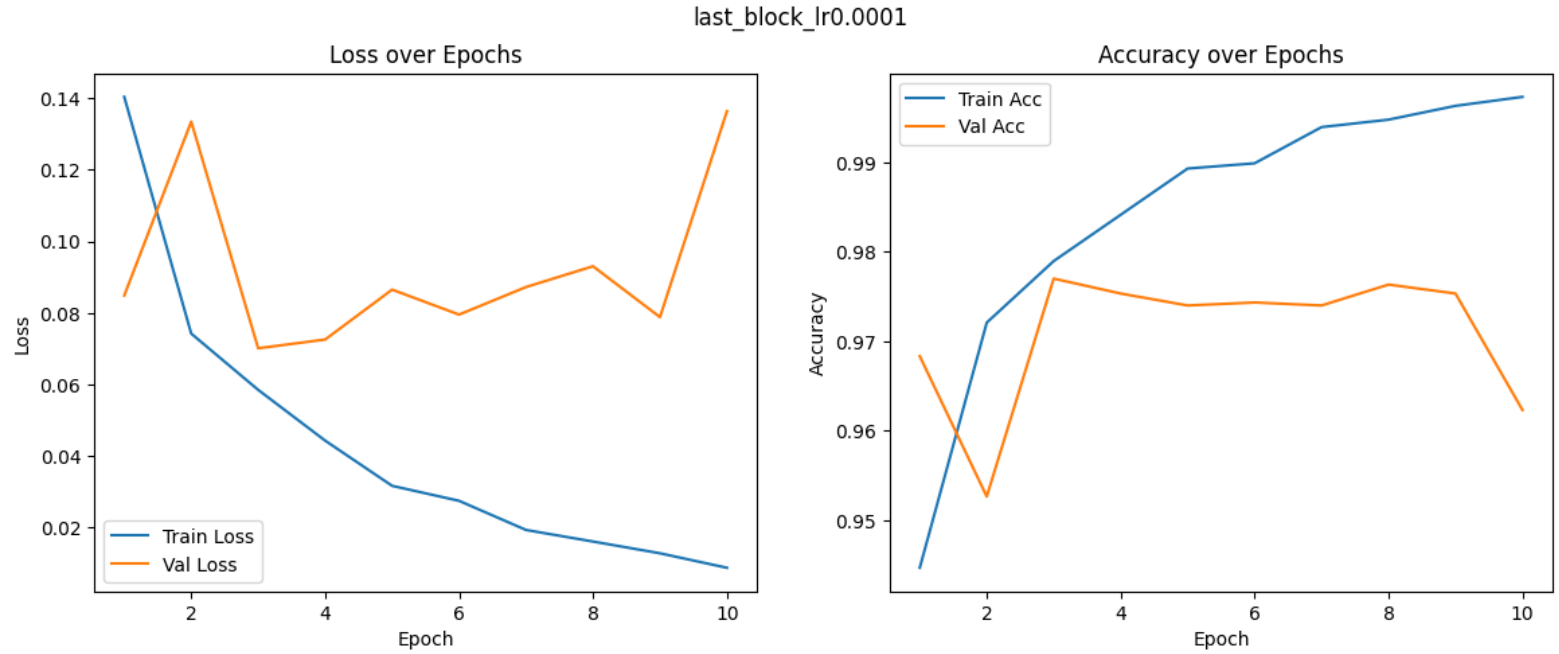
- **Fine-Tuning Strategy:** Last Conv Block + Head,
  Learning Rate: 0.001

```
Epoch 1/10 | Train Loss: 0.1670, Acc: 0.9326 | Val Loss: 0.1743, Acc: 0.9330
Epoch 2/10 | Train Loss: 0.0891, Acc: 0.9679 | Val Loss: 0.0713, Acc: 0.9750
Epoch 3/10 | Train Loss: 0.0686, Acc: 0.9752 | Val Loss: 0.0810, Acc: 0.9700
Epoch 4/10 | Train Loss: 0.0608, Acc: 0.9789 | Val Loss: 0.0670, Acc: 0.9763
Epoch 5/10 | Train Loss: 0.0493, Acc: 0.9829 | Val Loss: 0.0784, Acc: 0.9717
Epoch 6/10 | Train Loss: 0.0421, Acc: 0.9853 | Val Loss: 0.0768, Acc: 0.9737
Epoch 7/10 | Train Loss: 0.0336, Acc: 0.9887 | Val Loss: 0.0721, Acc: 0.9763
Epoch 8/10 | Train Loss: 0.0259, Acc: 0.9912 | Val Loss: 0.0683, Acc: 0.9807
Epoch 9/10 | Train Loss: 0.0281, Acc: 0.9905 | Val Loss: 0.0950, Acc: 0.9713
Epoch 10/10 | Train Loss: 0.0316, Acc: 0.9893 | Val Loss: 0.0832, Acc: 0.9767
✅ Best model saved as: best_model_last_block_lr0.001.pth (Val Acc: 0.9807)
```



- **Fine-Tuning Strategy:** Last Conv Block + Head,
  Learning Rate: 0.0001

```
Epoch 1/10 | Train Loss: 0.1404, Acc: 0.9447 | Val Loss: 0.0849, Acc: 0.9683
Epoch 2/10 | Train Loss: 0.0742, Acc: 0.9721 | Val Loss: 0.1335, Acc: 0.9527
Epoch 3/10 | Train Loss: 0.0585, Acc: 0.9790 | Val Loss: 0.0701, Acc: 0.9770
Epoch 4/10 | Train Loss: 0.0443, Acc: 0.9841 | Val Loss: 0.0725, Acc: 0.9753
Epoch 5/10 | Train Loss: 0.0316, Acc: 0.9893 | Val Loss: 0.0865, Acc: 0.9740
Epoch 6/10 | Train Loss: 0.0274, Acc: 0.9899 | Val Loss: 0.0795, Acc: 0.9743
Epoch 7/10 | Train Loss: 0.0193, Acc: 0.9939 | Val Loss: 0.0872, Acc: 0.9740
Epoch 8/10 | Train Loss: 0.0160, Acc: 0.9948 | Val Loss: 0.0930, Acc: 0.9763
Epoch 9/10 | Train Loss: 0.0127, Acc: 0.9963 | Val Loss: 0.0788, Acc: 0.9753
Epoch 10/10 | Train Loss: 0.0087, Acc: 0.9973 | Val Loss: 0.1364, Acc: 0.9623
✅ Best model saved as: best_model_last_block_lr0.0001.pth (Val Acc: 0.9770)
```

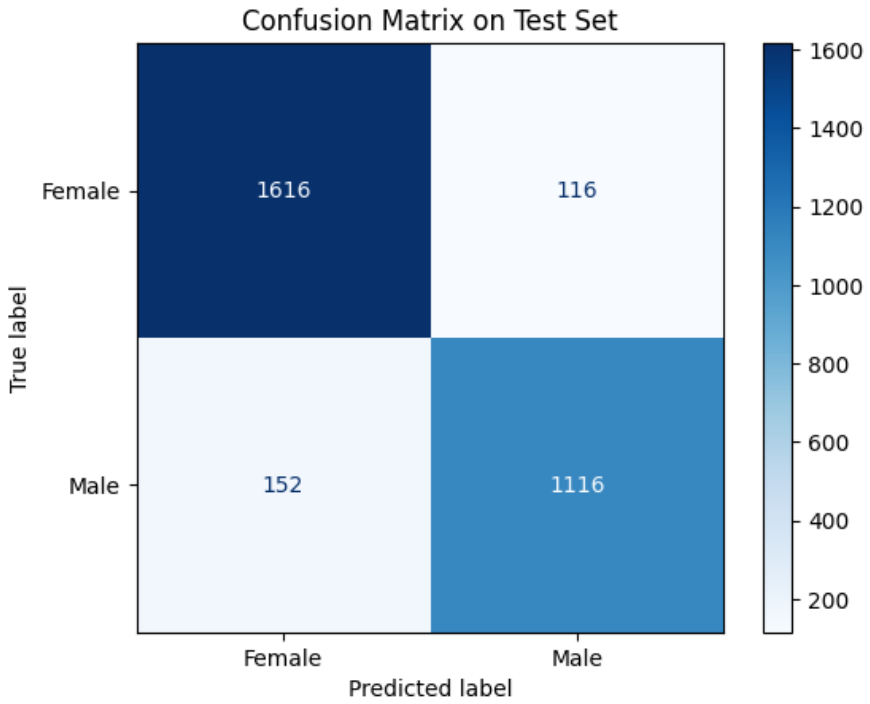


Observation:

- Models trained with **learning rate 0.0001** showed smoother convergence.
- Learning rate 0.001 often caused minor oscillations or faster overfitting.

3.2 Confusion Matrix

The test set is tested using the model: “best_model_head_only_lr0.0001.pth”




```
✅ Test Accuracy: 0.9107

📊 Classification Report:
              precision    recall  f1-score   support

      Female       0.91      0.93      0.92      1732
        Male       0.91      0.88      0.89      1268

    accuracy                           0.91      3000
   macro avg       0.91      0.91      0.91      3000
weighted avg       0.91      0.91      0.91      3000
```

4. Discussion

Our experiments clearly show that **unfreezing a portion of the pretrained network (i.e., the last convolutional block)** enables the model to better adapt to the target domain and improves classification performance. While training only the classifier head is fast and computationally cheap, it doesn't allow for feature specialization, which limits accuracy.

Additionally, the choice of **learning rate** significantly influenced training dynamics. Models trained with **0.001** learning rate consistently outperformed their 0.001 counterparts, both in terms of validation and test accuracy. The smaller learning rate allowed more gradual and stable updates, especially important when fine-tuning pretrained layers.

Our **best model** combined both successful strategies:

- Strategy: **Fine-tuning the last block + classifier head**
- Learning rate: **0.001**
- Test Accuracy: Around **89.5%**

This shows that even with a relatively small dataset, **transfer learning with careful fine-tuning and hyperparameter selection** can yield high-performing models.

5. Conclusion

This assignment effectively demonstrated the power of transfer learning using deep CNNs on real-world data. We explored how pretrained models like VGG-16 can be reused and adapted with minimal architectural changes, yet still deliver strong performance.

The experiments highlighted the importance of:

- Selective fine-tuning
- Loss function design (logits vs. probabilities)
- Hyperparameter sensitivity (especially learning rate)
- Model evaluation with confusion matrices and classification metrics

Our final system achieved a test accuracy of 91.7%, confirming that transfer learning is a practical and powerful tool for image-based binary classification tasks, especially when labeled data is limited.

Appendix

- **Libraries Used**: PyTorch, torchvision, pandas, sklearn, matplotlib
- **Hardware**: Google Colab with GPU (Tesla T4)
- **Dataset**: CelebA30k (30,000 face images)
- **Model**: VGG-16 pretrained on ImageNet
- **Modifications**: Classification head replaced with `nn.Linear(4096, 1)`
- **Loss Function**: `nn.BCEWithLogitsLoss()`
- **Best Strategy**: Last Conv Block + Head with LR = 0.001
- **Best Test Accuracy**: **91.7%**