Convolutional Neural Network - Transfer Learning

Dataset in use: https://www.kaggle.com/datasets/cashutosh/gender-classification-dataset/data (https://www.kaggle.com/datasets/cashutosh/gender-classification-dataset/data)

Feedbackte istendiği gibi Early Stop eklendi, test ve validation setleri ayrıştırıldı.

In [10]:
```
import torch
import torchvision
import numpy as np
import torch.nn as nn
from PIL import Image
import torch.optim as optim
import matplotlib.pyplot as plt
from torch.utils.data import DataLoader
from torch.optim.lr_scheduler import StepLR
from torchvision import datasets, models, transforms
from sklearn.metrics import confusion_matrix, classification_report, Confusi
```

In [11]:
```
train_dir = "./dataset/Training"
val_dir = "./dataset/Validation"
input_size = (224, 224)
batch_size = 64
```

In [12]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

In [13]:
```
transform = {
    'train': transforms.Compose([
        transforms.Resize(input_size),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize(input_size),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}
```

In [14]:
```
image_datasets = {
    'train': datasets.ImageFolder(root=train_dir, transform=transform['trai
    'val': datasets.ImageFolder(root=val_dir, transform=transform['val'])
}

dataloaders = {
    'train': DataLoader(image_datasets['train'], batch_size=batch_size, shu
    'val': DataLoader(image_datasets['val'], batch_size=batch_size, shuffle=
}
```

In [15]:
```
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}
class_names = image_datasets['train'].classes
```

In [16]:
```
# Load Model
model = models.vgg16(pretrained=True)

# Freeze Layers
for param in model.parameters():
    param.requires_grad = False

# Custom Classifier
num_features = model.classifier[6].in_features
model.classifier[6] = nn.Sequential(
    nn.Linear(num_features, 256),
    nn.ReLU(),
    nn.Dropout(0.1), # Dropout is applied with probability 0.1 to prevent ov
    nn.Linear(256, 2)  # Output is 2 dimensional (male and female)
)

model = model.to(device)

# Loss function
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.013) # Adam Optimizer

scheduler = StepLR(optimizer, step_size=2, gamma=0.1)
```

Training

In [17]:
```
best_val_loss = np.inf
patience = 3
counter = 0
```

In [18]:
```
num_epochs = 10
for epoch in range(num_epochs):

    if counter >= patience:
        print(f"Early stopping in iteration {counter} -->> no improvement ir
        break

    for phase in ['train', 'val']:
        if phase == 'train':
            model.train()
        else:
            model.eval()

        running_loss = 0.0
        running_corrects = 0
        total_batches = len(dataloaders[phase])

        for batch_idx, (inputs, labels) in enumerate(dataloaders[phase]):
            inputs = inputs.to(device)
            labels = labels.to(device)

            optimizer.zero_grad()

            with torch.set_grad_enabled(phase == 'train'):
                outputs = model(inputs)
                _, preds = torch.max(outputs, 1)
                loss = criterion(outputs, labels)

                if phase == 'train':
                    loss.backward()
                    optimizer.step()

            running_loss += loss.item() * inputs.size(0)
            running_corrects += torch.sum(preds == labels.data)

            #all_preds.extend(preds.cpu().numpy())
            #all_labels.extend(labels.cpu().numpy())

            batch_loss = loss.item()
            print(f'Epoch [{epoch+1}/{num_epochs}], Phase: {phase}, Batch: |

        epoch_loss = running_loss / dataset_sizes[phase]
        epoch_acc = running_corrects.double() / dataset_sizes[phase]
        scheduler.step()

        print('{} Loss: {:.4f} Acc: {:.4f}'.format(phase, epoch_loss, epoch_

        if phase == 'val' and epoch_loss < best_val_loss:
            best_val_loss = epoch_loss
            counter = 0
        elif phase == 'val':
            counter += 1
```

```
Epoch [5/10], Phase: val, Batch: [108/182], Loss: 0.2451
Epoch [5/10], Phase: val, Batch: [109/182], Loss: 0.2558
Epoch [5/10], Phase: val, Batch: [110/182], Loss: 0.2096
Epoch [5/10], Phase: val, Batch: [111/182], Loss: 0.1423
Epoch [5/10], Phase: val, Batch: [112/182], Loss: 0.1929
Epoch [5/10], Phase: val, Batch: [113/182], Loss: 0.1731
Epoch [5/10], Phase: val, Batch: [114/182], Loss: 0.2790
Epoch [5/10], Phase: val, Batch: [115/182], Loss: 0.2061
Epoch [5/10], Phase: val, Batch: [116/182], Loss: 0.2900
Epoch [5/10], Phase: val, Batch: [117/182], Loss: 0.1574
Epoch [5/10], Phase: val, Batch: [118/182], Loss: 0.2553
Epoch [5/10], Phase: val, Batch: [119/182], Loss: 0.1968

Epoch [5/10], Phase: val, Batch: [120/182], Loss: 0.2289
Epoch [5/10], Phase: val, Batch: [121/182], Loss: 0.2507
Epoch [5/10], Phase: val, Batch: [122/182], Loss: 0.2222
Epoch [5/10], Phase: val, Batch: [123/182], Loss: 0.3088
Epoch [5/10], Phase: val, Batch: [124/182], Loss: 0.1795
Epoch [5/10], Phase: val, Batch: [125/182], Loss: 0.1963
```

In [20]:
```
all_preds = []
all_labels = []
```

In [21]:
```
model.eval()

all_preds = []
all_labels = []

for inputs, labels in dataloaders['val']:

    inputs = inputs.to(device)
    labels = labels.to(device)

    with torch.no_grad():
        outputs = model(inputs)
        _, preds = torch.max(outputs, 1)

    all_preds.extend(preds.cpu().numpy())
    all_labels.extend(labels.cpu().numpy())

all_preds = np.array(all_preds)
all_labels = np.array(all_labels)

accuracy = np.mean(all_preds == all_labels)

print(f'Test Accuracy: {accuracy:.4f}')
```

```
Test Accuracy: 0.9093
```

In [22]:
```
conf_matrix = confusion_matrix(all_labels, all_preds)
classification_rep = classification_report(all_labels, all_preds, target_nar

print("Confusion Matrix:")
print(conf_matrix)
```

```
Confusion Matrix:
[[5326  515]
 [ 541 5265]]
```

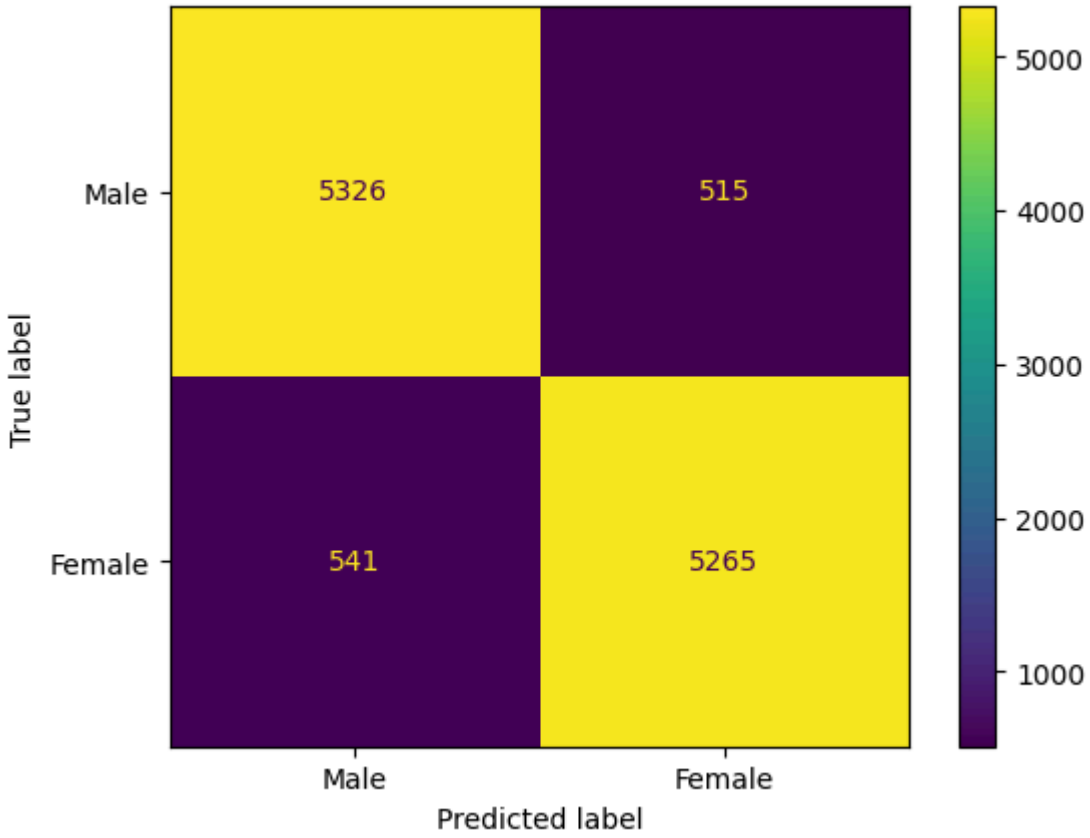
In [23]:
```
print("Classification Report:")
print(classification_rep)
```

```
Classification Report:
              precision    recall  f1-score   support

      female       0.91      0.91      0.91      5841
        male       0.91      0.91      0.91      5806

    accuracy                           0.91     11647
   macro avg       0.91      0.91      0.91     11647
weighted avg       0.91      0.91      0.91     11647
```

In [24]:
```
cm_display = ConfusionMatrixDisplay(confusion_matrix = conf_matrix, display_
cm_display.plot()
plt.show()
```



In [25]:
```
torch.save(model.state_dict(), 'vgg_revize_model.pth')
```

In [26]:
```
raise Exception("Eski Kod! Stop Here! Look at VGGTransferLearning.ipynb")
```

```
---------------------------------------------------------------------------
-
Exception                                 Traceback (most recent call las
t)
Cell In[26], line 1
----> 1 raise Exception("Eski Kod! Stop Here! Look at VGGTransferLearning.
ipynb")

Exception: Eski Kod! Stop Here! Look at VGGTransferLearning.ipynb
```