https://colab.research.google.com/drive/1leWcI2paXIfwYtjvtSnM0ZK76xcTwzBE?usp=sharing

import the dataset

```
import os
os.environ['CUDA_LAUNCH_BLOCKING'] = '1'
```

use API

```
from google.colab import files
files.upload()
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

选择文件 未选择任何文件 Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
```

install and unzip data

```
!kaggle datasets download -d birdy654/cifake-real-and-ai-generated-synthetic-images
!unzip cifake-real-and-ai-generated-synthetic-images.zip
```

```
  inflating: train/REAL/4553 (6).jpg
  inflating: train/REAL/4553 (7).jpg
  inflating: train/REAL/4553 (8).jpg
```

Import library we need

```
!pip install torchinfo
```

```
Collecting torchinfo
  Downloading torchinfo-1.8.0-py3-none-any.whl (23 kB)
Installing collected packages: torchinfo
Successfully installed torchinfo-1.8.0
```

```
import os
import json
import numpy as np
import pandas as pd
import torchvision.models as models
from PIL import Image
import matplotlib.pyplot as plt

import copy
import torch
import torchvision
from torch import nn
from torch import optim
import torch.nn.functional as F
from torchvision import transforms
from torchvision.transforms import InterpolationMode
from torch.utils.data import DataLoader
from torch.utils.data import Dataset
from torch.utils.data import random_split
from torchinfo import summary

from sklearn.model_selection import train_test_split
from torchvision.transforms import v2
from tqdm import tqdm
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score, precision_recall_curve, ConfusionMatrixDisplay, roc_cur
```

```
!pip install torchmetrics

from torchmetrics.functional.classification import binary_f1_score
```

```
Collecting torchmetrics
  Downloading torchmetrics-1.3.2-py3-none-any.whl (841 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 841.5/841.5 kB 16.2 MB/s eta 0:00:00
Requirement already satisfied: numpy>1.20.0 in /usr/local/lib/python3.10/dist-packages (from torchmetrics) (1.25.2)
Requirement already satisfied: packaging>17.1 in /usr/local/lib/python3.10/dist-packages (from torchmetrics) (24.0)
Requirement already satisfied: torch>=1.10.0 in /usr/local/lib/python3.10/dist-packages (from torchmetrics) (2.2.1+cu121)
Collecting lightning-utilities>=0.8.0 (from torchmetrics)
  Downloading lightning_utilities-0.11.2-py3-none-any.whl (26 kB)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from lightning-utilities>=0.8.0->torchmetrics) (67.7.2)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from lightning-utilities>=0.8.0->torchmetrics) (4.1
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->torchmetrics) (3.13.4)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->torchmetrics) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->torchmetrics) (3.3)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->torchmetrics) (3.1.3)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->torchmetrics) (2023.6.0)
Collecting nvidia-cuda-nvrtc-cu12==12.1.105 (from torch>=1.10.0->torchmetrics)
  Using cached nvidia_cuda_nvrtc_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (23.7 MB)
Collecting nvidia-cuda-runtime-cu12==12.1.105 (from torch>=1.10.0->torchmetrics)
  Using cached nvidia_cuda_runtime_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (823 kB)
Collecting nvidia-cuda-cupti-cu12==12.1.105 (from torch>=1.10.0->torchmetrics)
  Using cached nvidia_cuda_cupti_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (14.1 MB)
Collecting nvidia-cudnn-cu12==8.9.2.26 (from torch>=1.10.0->torchmetrics)
  Using cached nvidia_cudnn_cu12-8.9.2.26-py3-none-manylinux1_x86_64.whl (731.7 MB)
Collecting nvidia-cublas-cu12==12.1.3.1 (from torch>=1.10.0->torchmetrics)
  Using cached nvidia_cublas_cu12-12.1.3.1-py3-none-manylinux1_x86_64.whl (410.6 MB)
Collecting nvidia-cufft-cu12==11.0.2.54 (from torch>=1.10.0->torchmetrics)
  Using cached nvidia_cufft_cu12-11.0.2.54-py3-none-manylinux1_x86_64.whl (121.6 MB)
Collecting nvidia-curand-cu12==10.3.2.106 (from torch>=1.10.0->torchmetrics)
  Using cached nvidia_curand_cu12-10.3.2.106-py3-none-manylinux1_x86_64.whl (56.5 MB)
Collecting nvidia-cusolver-cu12==11.4.5.107 (from torch>=1.10.0->torchmetrics)
  Using cached nvidia_cusolver_cu12-11.4.5.107-py3-none-manylinux1_x86_64.whl (124.2 MB)
Collecting nvidia-cusparse-cu12==12.1.0.106 (from torch>=1.10.0->torchmetrics)
  Using cached nvidia_cusparse_cu12-12.1.0.106-py3-none-manylinux1_x86_64.whl (196.0 MB)
Collecting nvidia-nccl-cu12==2.19.3 (from torch>=1.10.0->torchmetrics)
  Using cached nvidia_nccl_cu12-2.19.3-py3-none-manylinux1_x86_64.whl (166.0 MB)
Collecting nvidia-nvtx-cu12==12.1.105 (from torch>=1.10.0->torchmetrics)
  Using cached nvidia_nvtx_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (99 kB)
Requirement already satisfied: triton==2.2.0 in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->torchmetrics) (2.2.0)
Collecting nvidia-nvjitlink-cu12 (from nvidia-cusolver-cu12==11.4.5.107->torch>=1.10.0->torchmetrics)
  Using cached nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (21.1 MB)
```

```
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch>=1.10.0->torchmetrics) (2.1.5)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch>=1.10.0->torchmetrics) (1.3.0)
Installing collected packages: nvidia-nvtx-cu12, nvidia-nvjitlink-cu12, nvidia-nccl-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvidia-cuda-run
Successfully installed lightning-utilities-0.11.2 nvidia-cublas-cu12-12.1.3.1 nvidia-cuda-cupti-cu12-12.1.105 nvidia-cuda-nvrtc-cu12-12.1.105 nv
```

GPU status

```
# Check and allocate GPU usage
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cuda
```

Data Pre-processing

```
!pip install albumentations torch torchvision
```

```
Requirement already satisfied: albumentations in /usr/local/lib/python3.10/dist-packages (1.3.1)
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.2.1+cu121)
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-packages (0.17.1+cu121)
Requirement already satisfied: numpy>=1.11.1 in /usr/local/lib/python3.10/dist-packages (from albumentations) (1.25.2)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from albumentations) (1.11.4)
Requirement already satisfied: scikit-image>=0.16.1 in /usr/local/lib/python3.10/dist-packages (from albumentations) (0.19.3)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.10/dist-packages (from albumentations) (6.0.1)
Requirement already satisfied: qudida>=0.0.4 in /usr/local/lib/python3.10/dist-packages (from albumentations) (0.0.4)
Requirement already satisfied: opencv-python-headless>=4.1.1 in /usr/local/lib/python3.10/dist-packages (from albumentations) (4.9.0.80)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch) (3.13.4)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch) (4.11.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch) (3.3)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch) (3.1.3)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch) (2023.6.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch) (12.1.105)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch) (12.1.105)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch) (12.1.105)
Requirement already satisfied: nvidia-cudnn-cu12==8.9.2.26 in /usr/local/lib/python3.10/dist-packages (from torch) (8.9.2.26)
Requirement already satisfied: nvidia-cublas-cu12==12.1.3.1 in /usr/local/lib/python3.10/dist-packages (from torch) (12.1.3.1)
Requirement already satisfied: nvidia-cufft-cu12==11.0.2.54 in /usr/local/lib/python3.10/dist-packages (from torch) (11.0.2.54)
Requirement already satisfied: nvidia-curand-cu12==10.3.2.106 in /usr/local/lib/python3.10/dist-packages (from torch) (10.3.2.106)
Requirement already satisfied: nvidia-cusolver-cu12==11.4.5.107 in /usr/local/lib/python3.10/dist-packages (from torch) (11.4.5.107)
Requirement already satisfied: nvidia-cusparse-cu12==12.1.0.106 in /usr/local/lib/python3.10/dist-packages (from torch) (12.1.0.106)
Requirement already satisfied: nvidia-nccl-cu12==2.19.3 in /usr/local/lib/python3.10/dist-packages (from torch) (2.19.3)
Requirement already satisfied: nvidia-nvtx-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch) (12.1.105)
Requirement already satisfied: triton==2.2.0 in /usr/local/lib/python3.10/dist-packages (from torch) (2.2.0)
Requirement already satisfied: nvidia-nvjitlink-cu12 in /usr/local/lib/python3.10/dist-packages (from nvidia-cusolver-cu12==11.4.5.107->torch) (
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.10/dist-packages (from torchvision) (9.4.0)
Requirement already satisfied: scikit-learn>=0.19.1 in /usr/local/lib/python3.10/dist-packages (from qudida>=0.0.4->albumentations) (1.2.2)
Requirement already satisfied: imageio>=2.4.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.16.1->albumentations) (2.31.6)
Requirement already satisfied: tifffile>=2019.7.26 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.16.1->albumentations) (2024.
Requirement already satisfied: PyWavelets>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.16.1->albumentations) (1.6.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.16.1->albumentations) (24.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch) (2.1.5)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch) (1.3.0)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.19.1->qudida>=0.0.4->albumentation
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.19.1->qudida>=0.0.4->albumen
```

Transformation

```
import os
from PIL import Image
import numpy as np
import torch
from torch.utils.data import Dataset, DataLoader
import albumentations as A
from albumentations.pytorch import ToTensorV2

class AlbumentationsDataset(Dataset):
        def __init__(self, root_dir, transform=None):
                self.root_dir = root_dir
                self.transform = transform
                self.images = []
                self.labels = []

                for label, cls_folder in enumerate(os.listdir(root_dir)):
                        cls_path = os.path.join(root_dir, cls_folder)
                        for img_file in os.listdir(cls_path):
                                self.images.append(os.path.join(cls_path, img_file))
                                self.labels.append(label)

        def __len__(self):
                return len(self.images)

        def __getitem__(self, idx):
                image_path = self.images[idx]
                image = np.array(Image.open(image_path).convert("RGB"))
                label = self.labels[idx]

                if self.transform:
                        augmented = self.transform(image=image)
                        image = augmented['image']


                return image, label

# Define transformations for training and testing
train_transform = A.Compose([
        A.Resize(224, 224),
        A.HorizontalFlip(p=0.5),
        A.RandomBrightnessContrast(p=0.2),
        A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
        ToTensorV2()
])

test_transform = A.Compose([
        A.Resize(224, 224),
        A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
        ToTensorV2()
])

# Create dataset
train_dataset = AlbumentationsDataset(root_dir='/content/train', transform=train_transform)
test_dataset = AlbumentationsDataset(root_dir='/content/test', transform=test_transform)
```

Present train and test loader

```
train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True, num_workers=0)
test_loader = DataLoader(test_dataset, batch_size=128, shuffle=True, num_workers=0)
```

Quick check by inspect the label of the dataset

```
def inspect_labels(dataloader):
        for i, (inputs, labels) in enumerate(dataloader):
                print(f'Batch {i + 1} labels: {labels.tolist()}')
                if i == 2:  # Check the first 3 batches
                        break

# Call the inspection function for both loaders
print("Training Loader Labels Inspection:")
inspect_labels(train_loader)
print("\nTest Loader Labels Inspection:")
inspect_labels(test_loader)
```

```
Training Loader Labels Inspection:
Batch 1 labels: [0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1,
```

```
Batch 2 labels: [0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1,
Batch 3 labels: [1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1,

Test Loader Labels Inspection:
Batch 1 labels: [1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0,
Batch 2 labels: [1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0,
Batch 3 labels: [0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0,
```

Build Resnet50

```
def build_resnet(num_classes=1, pretrained=True):
    # Load a pre-trained ResNet50 model
    model = models.resnet50(pretrained=pretrained)

    # Modify the fully connected layer to the number of desired classes
    num_features = model.fc.in_features
    model.fc = nn.Linear(num_features, num_classes)

    return model

resnet = build_resnet()
resnet.to(device)
```

```
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
      )
      (3): Bottleneck(
        (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
      )
      (4): Bottleneck(
        (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
      )
      (5): Bottleneck(
        (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
      )
    )
    (layer4): Sequential(
      (0): Bottleneck(
        (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (downsample): Sequential(
          (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (1): Bottleneck(
        (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
```

Built Alexnet

```
def build_alexnet(num_classes=1, pretrained=True):
    # Load a pre-trained AlexNet model
    model = models.alexnet(pretrained=pretrained)

    # Modify the classifier to output only one class for binary classification
    num_features = model.classifier[6].in_features
    model.classifier[6] = nn.Linear(num_features, num_classes)

    return model

# Create the model
alexnet = build_alexnet()
alexnet.to(device)
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weights'
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/alexnet-owt-7be5be79.pth" to /root/.cache/torch/hub/checkpoints/alexnet-owt-7be5be79.pth
100%|██████████| 233M/233M [00:01<00:00, 186MB/s]
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=1, bias=True)
  )
)
```

Built Letnet

```
class LeNet(nn.Module):
    def __init__(self, num_classes=1):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)  # Change input channels to 3 for RGB images
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 53 * 53, 120)  # Adjust the input features to match image size
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, num_classes)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 53 * 53)  # Flatten the output and adjust size accordingly
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = LeNet(num_classes=1).to(device)
```

Define Training Function

```
def train(model, dataloader, optimizer, criterion, device):
    model.train()
    total_loss = 0
    total_correct = 0

    for inputs, labels in dataloader:
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels.float().unsqueeze(1))
        loss.backward()
        optimizer.step()

        total_loss += loss.item() * inputs.size(0)
        predictions = torch.sigmoid(outputs).round()
        total_correct += predictions.eq(labels.float().unsqueeze(1)).sum().item()

    avg_loss = total_loss / len(dataloader.dataset)
    accuracy = total_correct / len(dataloader.dataset)
    return avg_loss, accuracy
```

Define Evaluate Function

```
def evaluate(model, dataloader, criterion, device):
    model.eval()  # Set the model to evaluation mode
    total_loss = 0
    total_correct = 0

    with torch.no_grad():  # No gradients needed for evaluation, which saves memory and computations
        for inputs, labels in dataloader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels.float().unsqueeze(1))

            total_loss += loss.item() * inputs.size(0)
            predictions = torch.sigmoid(outputs).round()
            total_correct += predictions.eq(labels.float().unsqueeze(1)).sum().item()

    avg_loss = total_loss / len(dataloader.dataset)
    accuracy = total_correct / len(dataloader.dataset)
    return avg_loss, accuracy
```

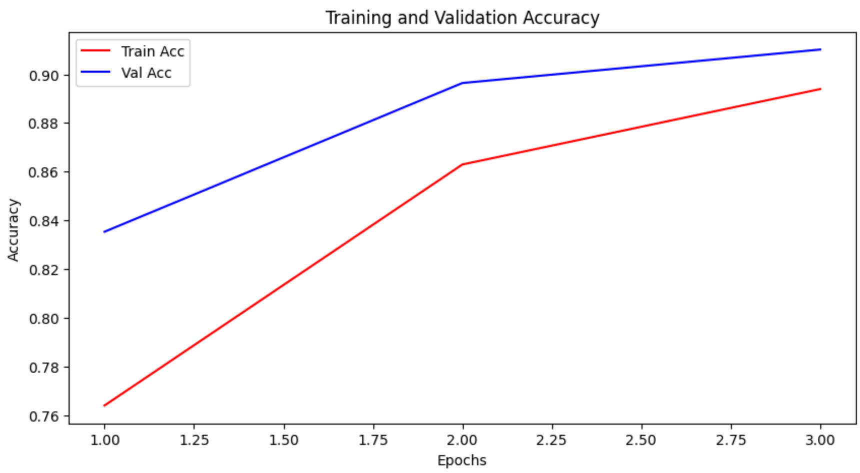
Training and Evaluate models

Letnet training,evaluating, and visualization

```
import torch.nn as nn

model1 = LeNet(num_classes=1).to(device)  # Make sure the model is instantiated and moved to GPU if available

# Define the optimizer
# The optimizer should be linked to the parameters of the specific model instance you're training
optimizer = torch.optim.SGD(model1.parameters(), lr=0.01, momentum=0.9)


# Define the loss function
criterion = nn.BCEWithLogitsLoss()
```

```
import matplotlib.pyplot as plt

num_epochs = 3  # Define the number of training epochs

# Initialize lists to store the per-epoch train and validation metrics
train_losses = []
train_accuracies = []
val_losses = []
val_accuracies = []

for epoch in range(num_epochs):
    # Train the model for one epoch
    train_loss, train_accuracy = train(model1, train_loader, optimizer, criterion, device)

    # Evaluate the model using the test dataset
    val_loss, val_accuracy = evaluate(model1, test_loader, criterion, device)

    # Append the metrics to the lists
    train_losses.append(train_loss)
    train_accuracies.append(train_accuracy)
    val_losses.append(val_loss)
    val_accuracies.append(val_accuracy)

    # Print training and validation results
    print(f'Epoch {epoch + 1}/{num_epochs}:')
    print(f'Train Loss: {train_loss:.4f}, Train Accuracy: {train_accuracy:.4f}')
    print(f'Validation Loss: {val_loss:.4f}, Validation Accuracy: {val_accuracy:.4f}')

# After the training loop, plot the training and validation metrics
plt.figure(figsize=(10, 5))
plt.plot(range(1, num_epochs+1), train_accuracies, 'r', label='Train Acc')
plt.plot(range(1, num_epochs+1), val_accuracies, 'b', label='Val Acc')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

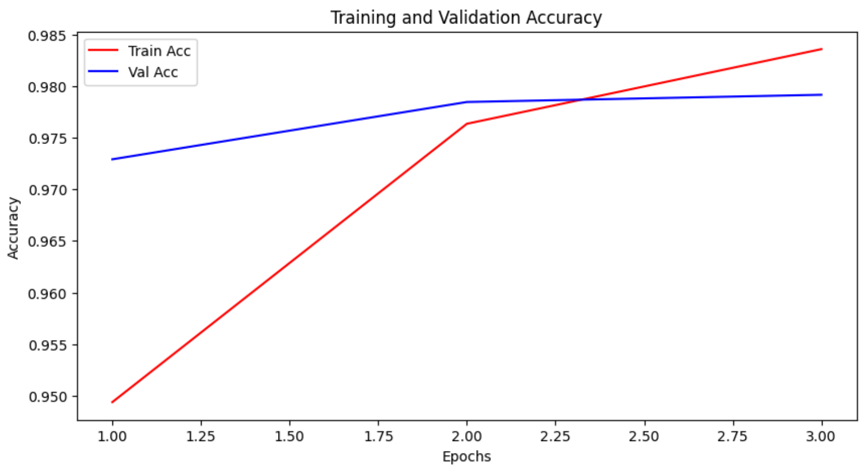
```
Epoch 1/3:
Train Loss: 0.4723, Train Accuracy: 0.7639
Validation Loss: 0.3674, Validation Accuracy: 0.8353
Epoch 2/3:
Train Loss: 0.3198, Train Accuracy: 0.8629
Validation Loss: 0.2530, Validation Accuracy: 0.8964
Epoch 3/3:
Train Loss: 0.2582, Train Accuracy: 0.8939
Validation Loss: 0.2272, Validation Accuracy: 0.9102
```



Resnet

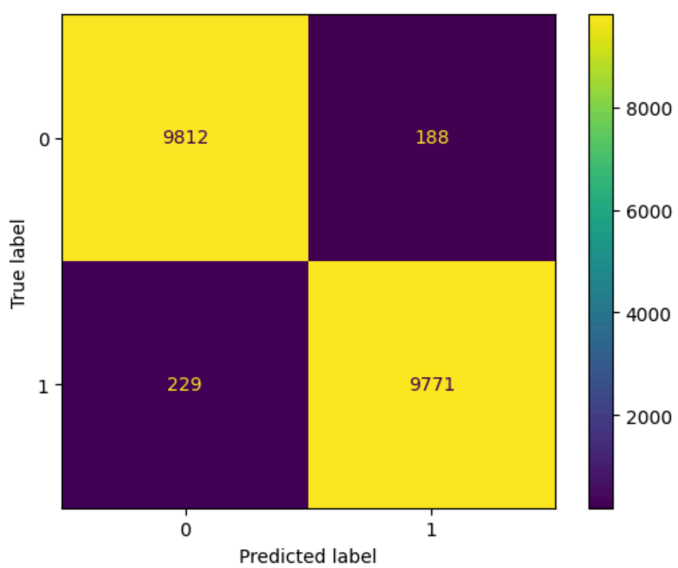
```
model2 = resnet.to(device)   # Make sure the model is instantiated and moved to GPU if available
optimizer = torch.optim.SGD(model2.parameters(), lr=0.01, momentum=0.9)
criterion = nn.BCEWithLogitsLoss()


num_epochs = 3   # Define the number of training epochs

# Initialize lists to store the per-epoch train and validation metrics
train_losses = []
train_accuracies = []
val_losses = []
val_accuracies = []

for epoch in range(num_epochs):
        # Train the model for one epoch
        train_loss, train_accuracy = train(model2, train_loader, optimizer, criterion, device)

        # Evaluate the model using the test dataset
        val_loss, val_accuracy = evaluate(model2, test_loader, criterion, device)

        # Append the metrics to the lists
        train_losses.append(train_loss)
        train_accuracies.append(train_accuracy)
        val_losses.append(val_loss)
        val_accuracies.append(val_accuracy)

        # Print training and validation results
        print(f'Epoch {epoch + 1}/{num_epochs}:')
        print(f'Train Loss: {train_loss:.4f}, Train Accuracy: {train_accuracy:.4f}')
        print(f'Validation Loss: {val_loss:.4f}, Validation Accuracy: {val_accuracy:.4f}')

# After the training loop, plot the training and validation metrics
plt.figure(figsize=(10, 5))
plt.plot(range(1, num_epochs+1), train_accuracies, 'r', label='Train Acc')
plt.plot(range(1, num_epochs+1), val_accuracies, 'b', label='Val Acc')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
Epoch 1/3:
Train Loss: 0.1282, Train Accuracy: 0.9494
Validation Loss: 0.0723, Validation Accuracy: 0.9729
Epoch 2/3:
Train Loss: 0.0631, Train Accuracy: 0.9763
Validation Loss: 0.0598, Validation Accuracy: 0.9785
Epoch 3/3:
Train Loss: 0.0437, Train Accuracy: 0.9836
Validation Loss: 0.0558, Validation Accuracy: 0.9791
```



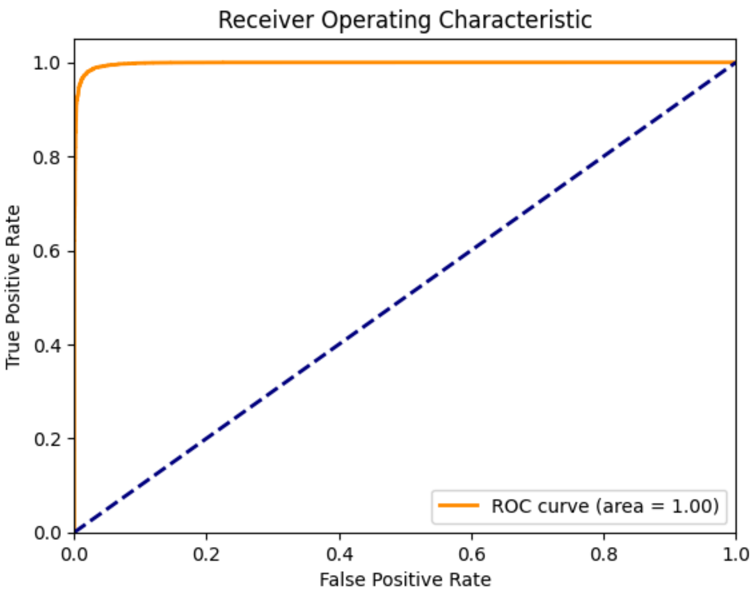

```
# Save the entire model
torch.save(model2, 'final_resnet_model.pth')
torch.save(model2, 'model_weights.pth')
```

```
def evaluate(model, dataloader, device):
        y_true = []
        y_pred = []

        model.eval()
        with torch.no_grad():
                for inputs, labels in dataloader:
                        inputs = inputs.to(device)
                        labels = labels.to(device)
                        outputs = model(inputs)


                        predicted_probs = torch.sigmoid(outputs)
                        predicted_labels = torch.round(predicted_probs)

                        y_true.extend(labels.tolist())
                        y_pred.extend(predicted_labels.squeeze().tolist())

        return y_true, y_pred


y_true, y_pred = evaluate(model2, test_loader, device)


from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_true, y_pred, labels=[0, 1])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])
disp.plot()
plt.show()
```

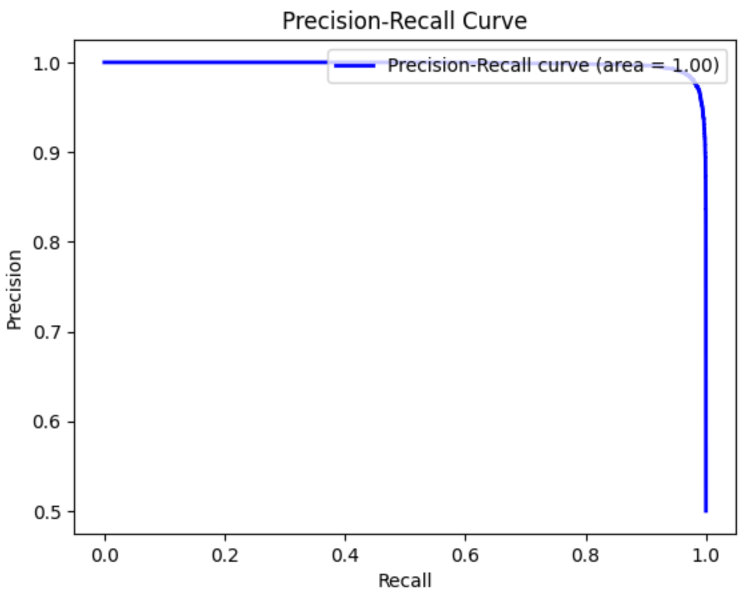



```
def evaluate(model, dataloader, device):
        model.eval()
        y_true = []
        y_scores = []

        with torch.no_grad():
                for inputs, labels in dataloader:
                        inputs = inputs.to(device)
                        labels = labels.to(device)
                        outputs = model(inputs)


                        scores = torch.sigmoid(outputs).squeeze()

                        y_true.extend(labels.tolist())
                        y_scores.extend(scores.cpu().tolist())

        return y_true, y_scores



# Get true labels and predicted scores
y_true, y_scores = evaluate(model2, test_loader, device)
```

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

fpr, tpr, _ = roc_curve(y_true, y_scores)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```




```
from sklearn.metrics import precision_recall_curve, auc
import matplotlib.pyplot as plt

precision, recall, thresholds = precision_recall_curve(y_true, y_scores)
pr_auc = auc(recall, precision)

plt.figure()
plt.plot(recall, precision, color='blue', lw=2, label=f'Precision-Recall curve (area = {pr_auc:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc="upper right")
plt.show()
```



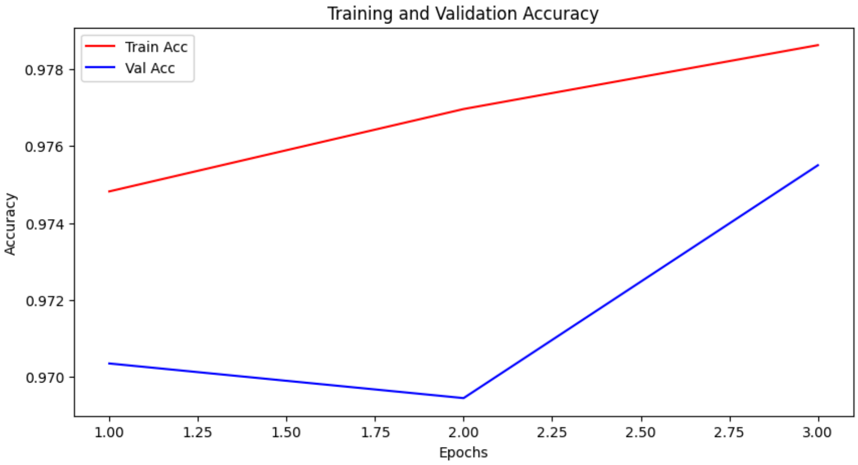
Alexnet

```
model3 = alexnet.to(device)  # Make sure the model is instantiated and moved to GPU if available
optimizer = torch.optim.SGD(model3.parameters(), lr=0.01, momentum=0.9)
criterion = nn.BCEWithLogitsLoss()
```

```
num_epochs = 3  # Define the number of training epochs

# Initialize lists to store the per-epoch train and validation metrics
train_losses = []
train_accuracies = []
val_losses = []
val_accuracies = []

for epoch in range(num_epochs):
    # Train the model for one epoch
    train_loss, train_accuracy = train(model3, train_loader, optimizer, criterion, device)

    # Evaluate the model using the test dataset
    val_loss, val_accuracy = evaluate(model3, test_loader, criterion, device)

    # Append the metrics to the lists
    train_losses.append(train_loss)
    train_accuracies.append(train_accuracy)
    val_losses.append(val_loss)
    val_accuracies.append(val_accuracy)

    # Print training and validation results
    print(f'Epoch {epoch + 1}/{num_epochs}:')
    print(f'Train Loss: {train_loss:.4f}, Train Accuracy: {train_accuracy:.4f}')
    print(f'Validation Loss: {val_loss:.4f}, Validation Accuracy: {val_accuracy:.4f}')
```

```
Epoch 1/3:
Train Loss: 0.0678, Train Accuracy: 0.9748
Validation Loss: 0.0837, Validation Accuracy: 0.9704
Epoch 2/3:
Train Loss: 0.0616, Train Accuracy: 0.9770
Validation Loss: 0.0844, Validation Accuracy: 0.9695
Epoch 3/3:
Train Loss: 0.0576, Train Accuracy: 0.9786
Validation Loss: 0.0670, Validation Accuracy: 0.9755
```

```
# After the training loop, plot the training and validation metrics
plt.figure(figsize=(10, 5))
plt.plot(range(1, num_epochs+1), train_accuracies, 'r', label='Train Acc')
plt.plot(range(1, num_epochs+1), val_accuracies, 'b', label='Val Acc')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



Hyperparameters Tuning on Resnet(since this is the best model among 3 of them)

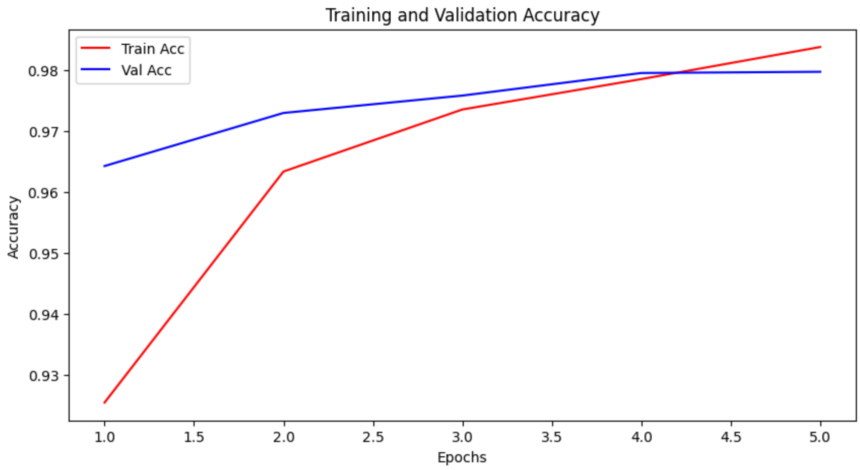
Experiment 1:

lower learning rate and increase the number of epoch

Trying to find a better local minima.

```
model4 = resnet.to(device)  # Make sure the model is instantiated and moved to GPU if available
optimizer = torch.optim.SGD(model4.parameters(), lr=0.001, momentum=0.9)#lower learning rate from 0.1 to 0.01
criterion = nn.BCEWithLogitsLoss()
num_epochs = 5  # increase number of epochs

# Initialize lists to store the per-epoch train and validation metrics
train_losses = []
train_accuracies = []
val_losses = []
val_accuracies = []

for epoch in range(num_epochs):
        # Train the model for one epoch
        train_loss, train_accuracy = train(model4, train_loader, optimizer, criterion, device)

        # Evaluate the model using the test dataset
        val_loss, val_accuracy = evaluate(model4, test_loader, criterion, device)

        # Append the metrics to the lists
        train_losses.append(train_loss)
        train_accuracies.append(train_accuracy)
        val_losses.append(val_loss)
        val_accuracies.append(val_accuracy)

        # Print training and validation results
        print(f'Epoch {epoch + 1}/{num_epochs}:')
        print(f'Train Loss: {train_loss:.8f}, Train Accuracy: {train_accuracy:.8f}')
        print(f'Validation Loss: {val_loss:.8f}, Validation Accuracy: {val_accuracy:.8f}')

# After the training loop, plot the training and validation metrics
plt.figure(figsize=(10, 5))
plt.plot(range(1, num_epochs+1), train_accuracies, 'r', label='Train Acc')
plt.plot(range(1, num_epochs+1), val_accuracies, 'b', label='Val Acc')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
Epoch 1/5:
Train Loss: 0.18617602, Train Accuracy: 0.92543000
Validation Loss: 0.09552688, Validation Accuracy: 0.96420000
Epoch 2/5:
Train Loss: 0.09546666, Train Accuracy: 0.96330000
Validation Loss: 0.07543346, Validation Accuracy: 0.97290000
Epoch 3/5:
Train Loss: 0.07119428, Train Accuracy: 0.97347000
Validation Loss: 0.06546338, Validation Accuracy: 0.97575000
Epoch 4/5:
Train Loss: 0.05757303, Train Accuracy: 0.97845000
Validation Loss: 0.05876442, Validation Accuracy: 0.97945000
Epoch 5/5:
Train Loss: 0.04477836, Train Accuracy: 0.98371000
Validation Loss: 0.05642315, Validation Accuracy: 0.97965000
```




```
# Save the entire model
torch.save(model4, 'final_resnet_model.pth')
```

```
torch.save(model4, 'model_weights.pth')
```

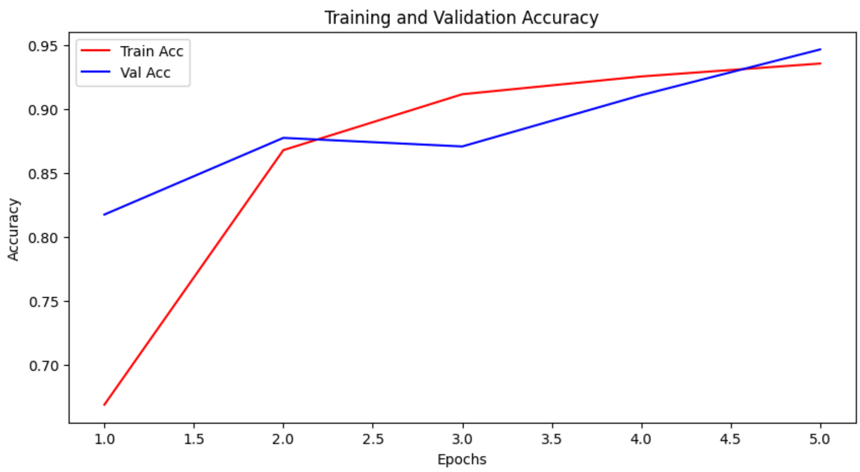
Experiment 2:

increase the learning rate and increase the number of epoch

Trying to find another local minima.

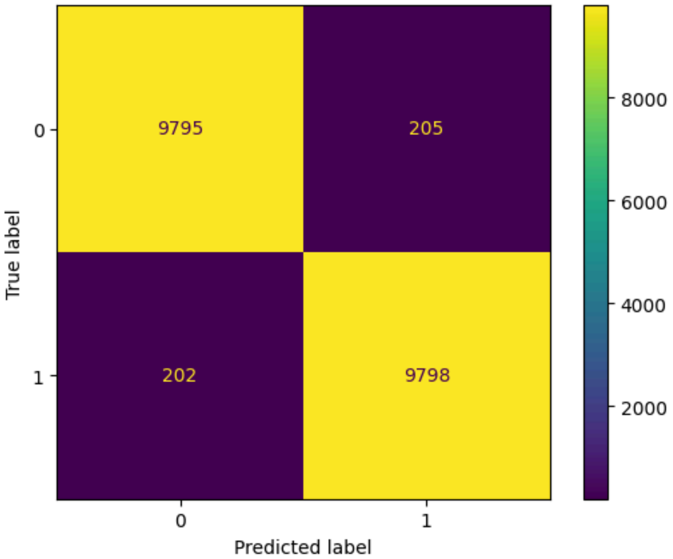
```
model5 = resnet.to(device)  # Make sure the model is instantiated and moved to GPU if available
optimizer = torch.optim.SGD(model5.parameters(), lr=0.1, momentum=0.9)#increase learning rate from 0.01 to 0.1
criterion = nn.BCEWithLogitsLoss()
num_epochs = 5  # increase number of epochs

# Initialize lists to store the per-epoch train and validation metrics
train_losses = []
train_accuracies = []
val_losses = []
val_accuracies = []

for epoch in range(num_epochs):
    # Train the model for one epoch
    train_loss, train_accuracy = train(model5, train_loader, optimizer, criterion, device)

    # Evaluate the model using the test dataset
    val_loss, val_accuracy = evaluate(model5, test_loader, criterion, device)

    # Append the metrics to the lists
    train_losses.append(train_loss)
    train_accuracies.append(train_accuracy)
    val_losses.append(val_loss)
    val_accuracies.append(val_accuracy)

    # Print training and validation results
    print(f'Epoch {epoch + 1}/{num_epochs}:')
    print(f'Train Loss: {train_loss:.8f}, Train Accuracy: {train_accuracy:.8f}')
    print(f'Validation Loss: {val_loss:.8f}, Validation Accuracy: {val_accuracy:.8f}')

# After the training loop, plot the training and validation metrics
plt.figure(figsize=(10, 5))
plt.plot(range(1, num_epochs+1), train_accuracies, 'r', label='Train Acc')
plt.plot(range(1, num_epochs+1), val_accuracies, 'b', label='Val Acc')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
Epoch 1/5:
Train Loss: 0.63398848, Train Accuracy: 0.66903000
Validation Loss: 0.39900069, Validation Accuracy: 0.81770000
Epoch 2/5:
Train Loss: 0.31472937, Train Accuracy: 0.86807000
Validation Loss: 0.30229282, Validation Accuracy: 0.87765000
Epoch 3/5:
Train Loss: 0.22272087, Train Accuracy: 0.91177000
Validation Loss: 0.30527460, Validation Accuracy: 0.87095000
Epoch 4/5:
Train Loss: 0.18963951, Train Accuracy: 0.92567000
Validation Loss: 0.21846669, Validation Accuracy: 0.91105000
Epoch 5/5:
Train Loss: 0.16496538, Train Accuracy: 0.93578000
Validation Loss: 0.14092445, Validation Accuracy: 0.94680000
```



Evaluate The Model(the model4)

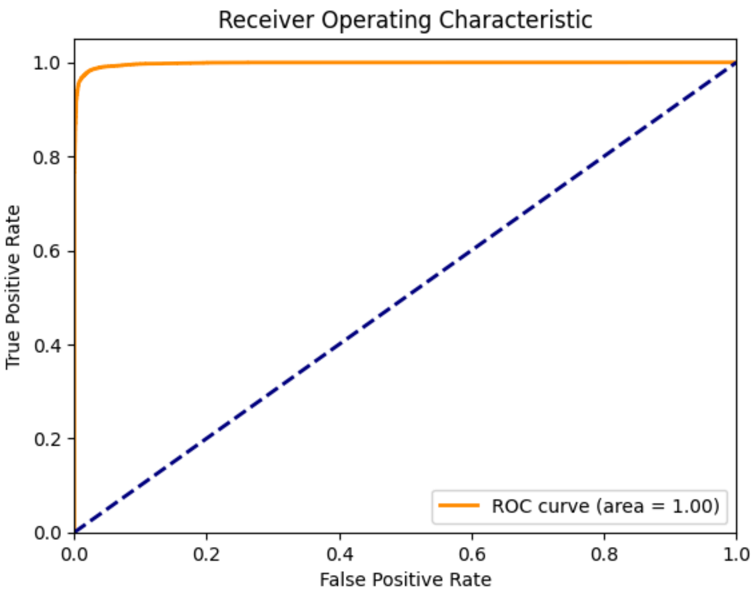
```
def evaluate(model, dataloader, device):
    y_true = []
    y_pred = []

    model.eval()
    with torch.no_grad():
        for inputs, labels in dataloader:
            inputs = inputs.to(device)
            labels = labels.to(device)
            outputs = model(inputs)


            predicted_probs = torch.sigmoid(outputs)
            predicted_labels = torch.round(predicted_probs)

            y_true.extend(labels.tolist())
            y_pred.extend(predicted_labels.squeeze().tolist())

    return y_true, y_pred


y_true, y_pred = evaluate(model4, test_loader, device)



from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_true, y_pred, labels=[0, 1])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])
disp.plot()
plt.show()
```




```
def evaluate(model, dataloader, device):
    model.eval()
    y_true = []
    y_scores = []

    with torch.no_grad():
        for inputs, labels in dataloader:
            inputs = inputs.to(device)
            labels = labels.to(device)
            outputs = model(inputs)


            scores = torch.sigmoid(outputs).squeeze()

            y_true.extend(labels.tolist())
            y_scores.extend(scores.cpu().tolist())

    return y_true, y_scores
```

```
# Get true labels and predicted scores
y_true, y_scores = evaluate(model4, test_loader, device)
```

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

fpr, tpr, _ = roc_curve(y_true, y_scores)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```




```
from sklearn.metrics import precision_recall_curve, auc
import matplotlib.pyplot as plt

precision, recall, thresholds = precision_recall_curve(y_true, y_scores)
pr_auc = auc(recall, precision)

plt.figure()
plt.plot(recall, precision, color='blue', lw=2, label=f'Precision-Recall curve (area = {pr_auc:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc="upper right")
plt.show()
```