```
pip install torchviz
```

```
Collecting torchviz
  Downloading torchviz-0.0.2.tar.gz (4.9 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: torch in /opt/conda/lib/python3.7/site-packages (from torchviz) (1.9.1)
Requirement already satisfied: graphviz in /opt/conda/lib/python3.7/site-packages (from torchviz) (0.8.4)
Requirement already satisfied: typing-extensions in /opt/conda/lib/python3.7/site-packages (from torch->torchviz) (3.10.0.2)
Building wheels for collected packages: torchviz
  Building wheel for torchviz (setup.py) ... done
  Created wheel for torchviz: filename=torchviz-0.0.2-py3-none-any.whl size=4151 sha256=e44e606e2c1a0086c8c0ad9247c56522bfe74825a8e89de542245ae8d7fc2925
  Stored in directory: /root/.cache/pip/wheels/04/38/f5/dc4f85c3909051823df49901e72015d2d750bd26b086480ec2
Successfully built torchviz
Installing collected packages: torchviz
Successfully installed torchviz-0.0.2
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
Note: you may need to restart the kernel to use updated packages.
```

```
pip install hiddenlayer
```

```
Collecting hiddenlayer
  Downloading hiddenlayer-0.3-py3-none-any.whl (19 kB)
Installing collected packages: hiddenlayer
Successfully installed hiddenlayer-0.3
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
Note: you may need to restart the kernel to use updated packages.
```

+ Code + Markdown

```python
# Import Libraries
import os
import numpy as np
import matplotlib.pyplot as plt
import tarfile
from tqdm.notebook import tqdm
from sklearn.metrics import f1_score, accuracy_score, confusion_matrix, classification_report
import seaborn as sns
import pandas as pd
import time
from torchviz import make_dot
import hiddenlayer as hl
from collections import Counter
from imblearn.over_sampling import SMOTE

import torch
import torchvision
import torch.nn as nn
import torch.nn.functional as F
from torchvision.datasets.utils import download_url
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader
import torchvision.transforms as tt
from torch.utils.data import random_split
from torchvision.utils import make_grid
from torchvision import transforms
import torchvision.models as models
from torch.utils.data.sampler import SubsetRandomSampler, WeightedRandomSampler
```

```
!pip install opendatasets
```

```
Collecting opendatasets
  Downloading opendatasets-0.1.20-py3-none-any.whl (14 kB)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.7/site-packages (from opendatasets) (4.62.3)
Requirement already satisfied: kaggle in /opt/conda/lib/python3.7/site-packages (from opendatasets) (1.5.12)
Requirement already satisfied: click in /opt/conda/lib/python3.7/site-packages (from opendatasets) (8.0.3)
Requirement already satisfied: importlib-metadata in /opt/conda/lib/python3.7/site-packages (from click->opendatasets) (4.8.2)
Requirement already satisfied: python-slugify in /opt/conda/lib/python3.7/site-packages (from kaggle->opendatasets) (5.0.2)
Requirement already satisfied: requests in /opt/conda/lib/python3.7/site-packages (from kaggle->opendatasets) (2.25.1)
Requirement already satisfied: python-dateutil in /opt/conda/lib/python3.7/site-packages (from kaggle->opendatasets) (2.8.0)
Requirement already satisfied: six>=1.10 in /opt/conda/lib/python3.7/site-packages (from kaggle->opendatasets) (1.16.0)
Requirement already satisfied: certifi in /opt/conda/lib/python3.7/site-packages (from kaggle->opendatasets) (2021.10.8)
Requirement already satisfied: urllib3 in /opt/conda/lib/python3.7/site-packages (from kaggle->opendatasets) (1.26.7)
Requirement already satisfied: zipp>=0.5 in /opt/conda/lib/python3.7/site-packages (from importlib-metadata->click->opendatasets) (3.6.0)
Requirement already satisfied: typing-extensions>=3.6.4 in /opt/conda/lib/python3.7/site-packages (from importlib-metadata->click->opendatasets) (3.10.0.2)
Requirement already satisfied: text-unidecode>=1.3 in /opt/conda/lib/python3.7/site-packages (from python-slugify->kaggle->opendatasets) (1.3)
Requirement already satisfied: idna<3,>=2.5 in /opt/conda/lib/python3.7/site-packages (from requests->kaggle->opendatasets) (2.10)
Requirement already satisfied: chardet<5,>=3.0.2 in /opt/conda/lib/python3.7/site-packages (from requests->kaggle->opendatasets) (4.0.0)
Installing collected packages: opendatasets
Successfully installed opendatasets-0.1.20
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
```

```python
import opendatasets as od

#volatille
#b06bccd3d1795d29ff0b071f635f2d79

od.download("https://www.kaggle.com/paultimothymooney/breast-histopathology-images", force=True) #Wczesniej wczytywałem poprzez sciezkie w systemie, po migracji na c
```

```
Please provide your Kaggle credentials to download this dataset. Learn more: http://bit.ly/kaggle-creds
Your Kaggle username:
  volatille
Your Kaggle Key:
  ································
Downloading breast-histopathology-images.zip to ./breast-histopathology-images
100%|██████████| 3.10G/3.10G [00:48<00:00, 68.1MB/s]
```

```python
[7]:
#Tworzenie bazy

data_dir = './breast-histopathology-images/'
folder_name = "IDC_regular_ps50_idx5"
image_folders = os.path.join(data_dir, folder_name)

transform = transforms.Compose([transforms.Resize((50, 50)), transforms.ToTensor()]) #Funkcja robie resize każdego pliku (50x50) oraz rzutuję na = torch.FloatTensor 

zdjecia_idc = []
for file in os.listdir(image_folders):
    zdjecia_idc.append(ImageFolder(os.path.join(image_folders, file), transform=transform)) #pętla, alokacja wczesniej przyjętych zmian dla plików.
datasets = torch.utils.data.ConcatDataset(zdjecia_idc) #upchanie wszystkiego do datasets
```

[8]:

```
from collections import Counter

#Obliczanie ile jest danych w zbiorze (IDC)
j=0
for dataset in tqdm(datasets.datasets):
    if j == 0:
        result = Counter(dataset.targets)
        j += 1
    else:
        result += Counter(dataset.targets)


Pliki_2 = os.listdir(os.path.join(data_dir, folder_name))
print(" \n Liczba Pacjentów:", len(Pliki_2))

print("""\n Liczba zdjęć:
    IDC_0 (Brak IDC)  : {}
    IDC_i (Obecne IDC): {}""".format(result[0], result[1]))
```

100% 279/279 [00:00<00:00, 5293.95it/s]

```
Liczba Pacjentów: 279

Liczba zdjęć:
   IDC_0 (Brak IDC)  : 198738
   IDC_i (Obecne IDC): 78786
```

[9]:

```
# Przygotowywanie modelu,danych itp.
random_seed = 42
torch.manual_seed(random_seed)

test_size = 38000
train_size = len(datasets) - test_size
train_ds, test_ds = random_split(datasets, [train_size, test_size])

val_size = 38000
train_size = len(train_ds) - val_size
train_ds, val_ds = random_split(train_ds, [train_size, val_size])

len(train_ds), len(val_ds), len(test_ds)
```

[9]: (201524, 38000, 38000)

[10]:

```
#Dane do treningu, validacji oraz testu
#   shuffle=True, przy false może nastąpić większy błąd val_loss (w treningu sieci)
#   pin_memory to wstawka dla conda cpu

train_data = DataLoader(train_ds, shuffle=True, num_workers=4, pin_memory=True)
val_data = DataLoader(val_ds, shuffle=True, num_workers=4, pin_memory=True)
test_data = DataLoader(test_ds, shuffle=True, num_workers=4, pin_memory=True)
```

```
/opt/conda/lib/python3.7/site-packages/torch/utils/data/dataloader.py:481: UserWarning: This DataLoader will create 4 worker processes in total. Our suggested max number
m is 2, which is smaller than what this DataLoader is going to create. Please be aware that excessive worker creation might get DataLoader running slow or even freeze, lo
void potential slowness/freeze if necessary.
  cpuset_checked))
```

[11]:

```
# Zapis danych dysk

def save_data(data, mode="train"):
    i = 0
    for img, label in data:
        folder_path = os.path.join(os.path.join(os.getcwd(), mode), str(np.array(label)[0]))
        if not os.path.exists(folder_path):
            os.makedirs(folder_path)
        i += 1
        image = transforms.ToPILImage()(img[0, :, :, :])
        image.save(os.path.join(folder_path, mode+"_"+str(i)+".jpg"), "JPEG")
```

[12]:

```
#Save Train Data
save_data(train_data, mode="train")

# Save Validation Data
save_data(val_data, mode="validation")

# Save Test Data
save_data(test_data, mode="test")
```

+ Code + Markdown

[13]:

```
# Transformacja danych /tt.ToTensor
train_tfms = tt.Compose([tt.ToTensor()])
valid_tfms = tt.Compose([tt.ToTensor()])
```

[14]:

```
data_dir = "./"

train_file = os.path.join(data_dir, "train")
val_file = os.path.join(data_dir, "validation")
test_file = os.path.join(data_dir, "test")

train_ds = ImageFolder(train_file, train_tfms)
val_ds = ImageFolder(val_file, valid_tfms)
test_ds = ImageFolder(test_file, valid_tfms)
```

```
#     Batch_size - liczba próbek propagowanych przez uczącą się sieć. Czyli jesli równe 1000, t
#     problemem z batch size może być liczba próbek niepodzielna przez parametr ustawiony przez


batch_size = 100
```

+ Code   + Markdown

```
# Data loadery. Podajemy dane dla treningu, walidacji, testu.
#     Numworkers = 3, tutaj experymentalnie 3.
#     Numworkers = 0 to znaczy że jest to główny proces
#     Numworkers = 1, oznacza że proces podzielony i ma jednego "podwykonawcę" (wedługp rzeczyt
#     Im więcej num_workers tym większe zasoby zuzywamy, a często nic nie zyskujemy. Możliwa ob

#     Shuffle, tasujemy dane treningowe.

train_dl = DataLoader(train_ds, batch_size, shuffle=True, num_workers=3, pin_memory=True)
val_dl = DataLoader(val_ds, batch_size*2, num_workers=3, pin_memory=True)
test_dl = DataLoader(test_ds, batch_size*2, num_workers=3, pin_memory=True)
```

```
/opt/conda/lib/python3.7/site-packages/torch/utils/data/dataloader.py:481: UserWarning: This DataLoader will creat
m is 2, which is smaller than what this DataLoader is going to create. Please be aware that excessive worker creat
void potential slowness/freeze if necessary.
  cpuset_checked))
```

```
#   Funkcja jest testem na dokładność. Wartość 1.0 oznacza najwyższą precyzję

#Funkcja oblicza wartości:
# TP = True Positives
# TN = True Negatives
# FP = False Positives
# FN = False Negatives

def F_score(output, label, threshold=0.5, beta=1):
    prob = output > threshold
    label = label > threshold

    TP = (prob & label).sum(1).float()
    TN = ((~prob) & (~label)).sum(1).float()
    FP = (prob & (~label)).sum(1).float()
    FN = ((~prob) & label).sum(1).float()

    precision = torch.mean(TP / (TP + FP + 1e-12))
    recall = torch.mean(TP / (TP + FN + 1e-12))
    F2 = (1 + beta**2) * precision * recall / (beta**2 * precision + recall + 1e-12)
    return F2.mean(0)
```

```
#Klasa PyTorch zawierająca funkcję do kroku treningu modułu, kroku walidacji modułu.
# @training_step() zwraca train_loss
# @validation_step() zwraca validation_loss
# @validation_epoch_end()


class ImageClassificationBase(nn.Module):
    def training_step(self, batch):
        images, targets = batch
        targets = torch.reshape(targets.type(torch.cuda.FloatTensor), (len(targets), 1))
        out = self(images)
        loss = F.binary_cross_entropy(out, targets)
        return loss

    def validation_step(self, batch):
        images, targets = batch
        targets = torch.reshape(targets.type(torch.cuda.FloatTensor), (len(targets), 1))
        out = self(images)                              # Generate predictions
        loss = F.binary_cross_entropy(out, targets)   # Calculate loss
        score = F_score(out, targets)
        return {'val_loss': loss.detach(), 'val_score': score.detach() }

    def validation_epoch_end(self, outputs):
        batch_losses = [x['val_loss'] for x in outputs]
        epoch_loss = torch.stack(batch_losses).mean()   # Combine losses
        batch_scores = [x['val_score'] for x in outputs]
        epoch_score = torch.stack(batch_scores).mean()      # Combine accuracies
        return {'val_loss': epoch_loss.item(), 'val_score': epoch_score.item()}

    def epoch_end(self, epoch, result):
        print("Epoch [{}], last_lr: {:.4f}, train_loss: {:.4f}, val_loss: {:.4f}, val_score: {:.4f}".format(
            epoch, result['lrs'][-1], result['train_loss'], result['val_loss'], result['val_score']))
```

[21]:

```python
class BreastCancerResnet34(ImageClassificationBase):
    def __init__(self):
        super().__init__()
        # Use a pretrained model
        self.network = models.resnet34(pretrained=True)
        # Replace last layer
        num_ftrs = self.network.fc.in_features
        self.network.fc = nn.Linear(num_ftrs, 1)

    def forward(self, xb):
        return torch.sigmoid(self.network(xb))

    def freeze(self):
        # To freeze the residual layers
        for param in self.network.parameters():
            param.require_grad = False
        for param in self.network.fc.parameters():
            param.require_grad = True

    def unfreeze(self):
        # Unfreeze all layers
        for param in self.network.parameters():
            param.require_grad = True
```

[22]:

```python
def get_default_device():
    """Pick GPU if available, else CPU"""
    if torch.cuda.is_available():
        return torch.device('cuda')
    else:
        return torch.device('cpu')

def to_device(data, device):
    """Move tensor(s) to chosen device"""
    if isinstance(data, (list,tuple)):
        return [to_device(x, device) for x in data]
    return data.to(device, non_blocking=True)

class DeviceDataLoader():
    """Wrap a dataloader to move data to a device"""
    def __init__(self, dl, device):
        self.dl = dl
        self.device = device

    def __iter__(self):
        """Yield a batch of data after moving it to device"""
        for b in self.dl:
            yield to_device(b, self.device)

    def __len__(self):
        """Number of batches"""
        return len(self.dl)

device = get_default_device()
device
```

[22… device(type='cuda')

[23]:

```python
train_dl = DeviceDataLoader(train_dl, device)
val_dl = DeviceDataLoader(val_dl, device)
test_dl = DeviceDataLoader(test_dl, device)
```

[24]:

```python
def evaluate(model, val_loader):
    model.eval()
    outputs = [model.validation_step(batch) for batch in val_loader]
    return model.validation_epoch_end(outputs)

def get_lr(optimizer):
    for param_group in optimizer.param_groups:
        return param_group['lr']

def fit_one_cycle(epochs, max_lr, model, train_loader, val_loader,
                  weight_decay=0, grad_clip=None, opt_func=torch.optim.SGD):
    torch.cuda.empty_cache()
    history = []

    # Set up cutom optimizer with weight decay
    optimizer = opt_func(model.parameters(), max_lr, weight_decay=weight_decay)
    # Set up one-cycle learning rate scheduler
    sched = torch.optim.lr_scheduler.OneCycleLR(optimizer, max_lr, epochs=epochs,
                                                steps_per_epoch=len(train_loader))

    for epoch in range(epochs):
        # Training Phase
        model.train()
        train_losses = []
        lrs = []
        for batch in tqdm(train_loader):
            loss = model.training_step(batch)
            train_losses.append(loss)
            loss.backward()

            # Gradient clipping
            if grad_clip:
                nn.utils.clip_grad_value_(model.parameters(), grad_clip)

            optimizer.step()
            optimizer.zero_grad()

            # Record & update learning rate
            lrs.append(get_lr(optimizer))
            sched.step()

        # Validation phase
        result = evaluate(model, val_loader)
        result['train_loss'] = torch.stack(train_losses).mean().item()
        result['lrs'] = lrs
        model.epoch_end(epoch, result)
        history.append(result)
    return history
```

[25]:
```
model = to_device(BreastCancerResnet34(), device)
```
```
Downloading: "https://download.pytorch.org/models/resnet34-b627a593.pth" to /root/.cache/torch/hub/checkpoints/resnet34-b627a593.pth
100% 83.3M/83.3M [00:03<00:00, 31.6MB/s]
```

[26]:
```
history = [evaluate(model, val_dl)]
history
```
```
[26… [{'val_loss': 0.6686992049217224, 'val_score': 0.018263157457113266}]
```

[28]:
```
#Resnet z metodą nauki 1 warstwy z zamrożeniem

model.freeze()
```

[29]:
```
epochs = 15
max_lr = 0.01
grad_clip = 0.1
weight_decay = 1e-4
opt_func = torch.optim.Adam
```

[30]:
```
%%time
start_time = time.time()
history += fit_one_cycle(epochs, max_lr, model, train_dl, val_dl,
                         grad_clip=grad_clip,
                         weight_decay=weight_decay,
                         opt_func=opt_func)

train_time = time.time() - start_time
```
```
100% 2016/2016 [02:03<00:00, 18.55it/s]
Epoch [0], last_lr: 0.0015, train_loss: 0.3165, val_loss: 0.4039, val_score: 0.1346
100% 2016/2016 [01:59<00:00, 18.59it/s]
Epoch [1], last_lr: 0.0044, train_loss: 0.3179, val_loss: 0.3451, val_score: 0.1832
100% 2016/2016 [01:58<00:00, 18.52it/s]
Epoch [2], last_lr: 0.0076, train_loss: 0.3235, val_loss: 0.4990, val_score: 0.1280
100% 2016/2016 [01:56<00:00, 15.95it/s]
Epoch [3], last_lr: 0.0097, train_loss: 0.3256, val_loss: 0.3365, val_score: 0.1760
100% 2016/2016 [01:56<00:00, 18.64it/s]
Epoch [4], last_lr: 0.0099, train_loss: 0.3229, val_loss: 0.4014, val_score: 0.1503
100% 2016/2016 [01:58<00:00, 18.37it/s]
Epoch [5], last_lr: 0.0095, train_loss: 0.3211, val_loss: 0.4361, val_score: 0.1549
100% 2016/2016 [01:58<00:00, 18.27it/s]
Epoch [6], last_lr: 0.0087, train_loss: 0.3189, val_loss: 0.3276, val_score: 0.1998
100% 2016/2016 [01:59<00:00, 16.12it/s]
Epoch [7], last_lr: 0.0075, train_loss: 0.3159, val_loss: 0.3697, val_score: 0.1837
100% 2016/2016 [02:01<00:00, 18.24it/s]
Epoch [8], last_lr: 0.0061, train_loss: 0.3123, val_loss: 0.4203, val_score: 0.2461
100% 2016/2016 [02:01<00:00, 18.42it/s]
Epoch [9], last_lr: 0.0046, train_loss: 0.3085, val_loss: 0.3146, val_score: 0.2206
100% 2016/2016 [02:01<00:00, 18.51it/s]
Epoch [10], last_lr: 0.0032, train_loss: 0.3045, val_loss: 0.3352, val_score: 0.2304
100% 2016/2016 [02:01<00:00, 18.57it/s]
Epoch [11], last_lr: 0.0019, train_loss: 0.2981, val_loss: 0.3149, val_score: 0.1884
100% 2016/2016 [01:59<00:00, 18.46it/s]
Epoch [12], last_lr: 0.0009, train_loss: 0.2933, val_loss: 0.2944, val_score: 0.1982
100% 2016/2016 [02:01<00:00, 15.66it/s]
Epoch [13], last_lr: 0.0002, train_loss: 0.2883, val_loss: 0.2864, val_score: 0.2167
100% 2016/2016 [02:01<00:00, 18.11it/s]
Epoch [14], last_lr: 0.0000, train_loss: 0.2858, val_loss: 0.2845, val_score: 0.2139
CPU times: user 27min 3s, sys: 1min 4s, total: 28min 8s
```

[31]:
```
model.unfreeze()
```

[32]:

```python
%%time
start_time = time.time()
history += fit_one_cycle(epochs, 0.001, model, train_dl, val_dl,
                         grad_clip=grad_clip,
                         weight_decay=weight_decay,
                         opt_func=opt_func)
train_time += time.time() - start_time
```

```
100% 2016/2016 [02:03<00:00, 18.34it/s]
Epoch [0], last_lr: 0.0002, train_loss: 0.2850, val_loss: 0.2855, val_score: 0.2187
100% 2016/2016 [02:04<00:00, 15.91it/s]
Epoch [1], last_lr: 0.0004, train_loss: 0.2859, val_loss: 0.2862, val_score: 0.2187
100% 2016/2016 [02:05<00:00, 18.16it/s]
Epoch [2], last_lr: 0.0008, train_loss: 0.2869, val_loss: 0.2943, val_score: 0.2257
100% 2016/2016 [02:03<00:00, 17.40it/s]
Epoch [3], last_lr: 0.0010, train_loss: 0.2880, val_loss: 0.2908, val_score: 0.2052
100% 2016/2016 [02:03<00:00, 18.65it/s]
Epoch [4], last_lr: 0.0010, train_loss: 0.2880, val_loss: 0.3211, val_score: 0.1792
100% 2016/2016 [02:03<00:00, 18.26it/s]
Epoch [5], last_lr: 0.0010, train_loss: 0.2875, val_loss: 0.2870, val_score: 0.2185
100% 2016/2016 [02:01<00:00, 15.89it/s]
Epoch [6], last_lr: 0.0009, train_loss: 0.2868, val_loss: 0.2929, val_score: 0.2231
100% 2016/2016 [02:02<00:00, 18.19it/s]
Epoch [7], last_lr: 0.0008, train_loss: 0.2850, val_loss: 0.2882, val_score: 0.2231
100% 2016/2016 [02:03<00:00, 18.24it/s]
Epoch [8], last_lr: 0.0006, train_loss: 0.2842, val_loss: 0.2862, val_score: 0.2082
100% 2016/2016 [02:03<00:00, 16.49it/s]
Epoch [9], last_lr: 0.0005, train_loss: 0.2820, val_loss: 0.2853, val_score: 0.2121
100% 2016/2016 [02:02<00:00, 18.24it/s]
Epoch [10], last_lr: 0.0003, train_loss: 0.2801, val_loss: 0.2859, val_score: 0.2077
100% 2016/2016 [02:03<00:00, 18.41it/s]
Epoch [11], last_lr: 0.0002, train_loss: 0.2790, val_loss: 0.2823, val_score: 0.2148
100% 2016/2016 [02:02<00:00, 18.26it/s]
Epoch [12], last_lr: 0.0001, train_loss: 0.2774, val_loss: 0.2831, val_score: 0.2103
100% 2016/2016 [02:02<00:00, 18.29it/s]
Epoch [13], last_lr: 0.0000, train_loss: 0.2767, val_loss: 0.2813, val_score: 0.2139
100% 2016/2016 [02:03<00:00, 18.36it/s]
Epoch [14], last_lr: 0.0000, train_loss: 0.2750, val_loss: 0.2815, val_score: 0.2178
CPU times: user 27min 50s, sys: 1min 10s, total: 29min 1s
Wall time: 34min 4s
```

```python
#Blokujemy gradient, obliczanie pred. danych

@torch.no_grad()
def predict_dl(dl, model, threshold=0.5):
    torch.cuda.empty_cache()
    batch_probs = []
    for xb, _ in tqdm(dl):
        probs = model(xb)
        batch_probs.append(probs.cpu().detach())
    batch_probs = torch.cat(batch_probs)
    return [int(x) for x in batch_probs>threshold]
```

+ Code + Markdown

34]:

```python
# Analyze Prediction Results
test_preds = predict_dl(test_dl, model)
actual_label = test_dl.dl.dataset.targets
```

```
100% 190/190 [00:13<00:00, 18.84it/s]
```

[35]:

```
#Macierz błedów. Obliczanie

f1 = f1_score(actual_label, test_preds)
f_score = float(np.array(F_score(torch.tensor(np.array(test_preds).reshape(len(test_preds), 1)), torch.
accuracy = accuracy_score(actual_label, test_preds)
cm = confusion_matrix(actual_label, test_preds)
report = classification_report(actual_label, test_preds)

print("Model F-Score (Test Data): ", f_score)
print("Model F1-Score (Test Data): ", f1)
print("Model Accuracy: ", accuracy)
print("Confusion Matrix:\n", cm)
print("\nClassification Report:\n", report)

# Plot Confusion Matrix
df_cm = pd.DataFrame(cm, index = [i for i in "01"], columns = [i for i in "01"])
plt.figure(figsize = (10,7))
sns.set(font_scale=1.4)
sns.heatmap(df_cm, cmap="Oranges", annot=True, annot_kws={"size": 16})
plt.title("Plot of Confusion Matrix")
plt.show()
plt.savefig("ResNet34_CM")
```

```
Model F-Score (Test Data):  0.2200789451599121
Model F1-Score (Test Data):  0.782137011924246
Model Accuracy:  0.8773947368421052
Confusion Matrix:
 [[24978  2119]
 [ 2540  8363]]

Classification Report:
               precision    recall  f1-score   support

           0       0.91      0.92      0.91     27097
           1       0.80      0.77      0.78     10903

    accuracy                           0.88     38000
   macro avg       0.85      0.84      0.85     38000
weighted avg       0.88      0.88      0.88     38000
```