Importing Libraries

In [ ]:

```
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
from torch.nn import LeakyReLU,ReLU,Tanh,Sigmoid,Softmax
import torch.nn.functional as F
from torch import optim
from torch.utils.data import Dataset,DataLoader
import torchvision.models as models
!pip install timm
import timm

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib
import joblib
import cv2
import os
from tqdm.notebook import tqdm
import time
import random
from PIL import Image,ImageOps
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report


import warnings
warnings.filterwarnings("ignore")
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/col
ab-wheels/public/simple/
Requirement already satisfied: timm in /usr/local/lib/python3.7/dist-packa
ges (0.6.11)
Requirement already satisfied: huggingface-hub in /usr/local/lib/python3.
7/dist-packages (from timm) (0.10.0)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dist-pac
kages (from timm) (6.0)
Requirement already satisfied: torchvision in /usr/local/lib/python3.7/dis
t-packages (from timm) (0.13.1+cu113)
Requirement already satisfied: torch>=1.7 in /usr/local/lib/python3.7/dist
-packages (from timm) (1.12.1+cu113)
Requirement already satisfied: typing-extensions in /usr/local/lib/python
3.7/dist-packages (from torch>=1.7->timm) (4.1.1)
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.
7/dist-packages (from huggingface-hub->timm) (21.3)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python
3.7/dist-packages (from huggingface-hub->timm) (5.0.0)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-p
ackages (from huggingface-hub->timm) (2.23.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-p
ackages (from huggingface-hub->timm) (3.8.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packa
ges (from huggingface-hub->timm) (4.64.1)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/
python3.7/dist-packages (from packaging>=20.9->huggingface-hub->timm) (3.
0.9)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-
packages (from importlib-metadata->huggingface-hub->timm) (3.8.1)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/usr/local/lib/python3.7/dist-packages (from requests->huggingface-hub->ti
mm) (1.24.3)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python
3.7/dist-packages (from requests->huggingface-hub->timm) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/di
st-packages (from requests->huggingface-hub->timm) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python
3.7/dist-packages (from requests->huggingface-hub->timm) (2022.9.24)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-pack
ages (from torchvision->timm) (1.21.6)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/pyt
hon3.7/dist-packages (from torchvision->timm) (7.1.2)
```

Setting Seed

In [ ]:

```python
def seed_everything(SEED=42):
    random.seed(SEED)
    np.random.seed(SEED)
    torch.manual_seed(SEED)
    torch.cuda.manual_seed(SEED)
    torch.cuda.manual_seed_all(SEED)
    torch.backends.cudnn.benchmark = True
SEED=42
seed_everything(SEED=SEED)
```

In [ ]:

```
if torch.cuda.is_available():
    device = 'cuda'
else:
    device = 'cpu'
```

Dataset Creation

In [ ]:

```
torchvision.datasets.Caltech101(root = 'Data', download = True)
```

```
Files already downloaded and verified
```

Out[ ]:

```
Dataset Caltech101
    Number of datapoints: 8677
    Root location: Data/caltech101
    Target type: ['category']
```

In [ ]:

```
path = []
label = []

parent = os.path.join('Data','caltech101','101_ObjectCategories')

for i,j in enumerate(os.listdir(parent)):

  category = os.path.join(parent,j)

  for k in os.listdir(category):
    path.append(os.path.join(category,k))
    label.append(i)

df = pd.DataFrame({'Image':path,'Label':label})
```

In [ ]:

```
df = df.groupby('Label').apply(lambda x: x.sample(30))
```

In [ ]:

```
df_train,df_test = train_test_split(df, test_size=0.2, random_state=SEED,stratify = df.
Label)
df_train, df_val = train_test_split(df_train, test_size=0.2, random_state=SEED,stratify
= df_train.Label)
```

In [ ]:

```
class ImageData(Dataset):
    def __init__(self,df,transform=None):
        self.df = df
        self.transform = transform

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

    def __getitem__(self,index):
        img_path = self.df.iloc[index,0]
        labels = torch.tensor(self.df.iloc[index,1],dtype=torch.long)
        image = (Image.open(img_path)).convert('RGB')
        if self.transform is not None:
            image = self.transform(image)
        return image, labels

transform = transforms.Compose([transforms.Resize(size = (224,224)),transforms.ToTensor
(), transforms.Normalize(mean = [0.485, 0.456, 0.406],std = [0.229, 0.224, 0.225])])#

train_generator = DataLoader(ImageData(df_train,transform=transform),batch_size = 100,s
huffle = True)
val_generator = DataLoader(ImageData(df_val,transform=transform),batch_size = 100,shuff
le = True)
test_generator = DataLoader(ImageData(df_test,transform=transform),batch_size = 100,shu
ffle = True)
```

Model Creation

In [ ]:

```
class Model(nn.Module):
    def __init__(self, model_name = 'mobilenetv3_large_100', pretrained = True):
        super().__init__()
        self.model_name = model_name
        self.cnn = timm.create_model(self.model_name, pretrained = pretrained, num_clas
ses = 102)

    def forward(self, x):
        x = self.cnn(x)
        return x
```

Train, Validation and Test Functions

In [ ]:

```
def train_fn(train_loader, model, criterion, optmizer, device):
  model.train()

  size = len(train_loader.dataset)
  num_batches = len(train_loader)

  loss, correct = 0, 0

  for batch, (X, y) in tqdm(enumerate(train_loader)):

    start = time.time()

    device = torch.device(device)
    X, y = X.to(device), y.to(device)

    optimizer.zero_grad()
    pred = model(X)
    loss = criterion(pred, y.long())
    current = batch * len(X)

    loss.backward()
    optimizer.step()

    y_pred, y_true = torch.argmax(pred, axis=1), y.long().squeeze()
    correct += (y_pred == y_true).type(torch.float).sum().item()

    end = time.time()
    time_delta = np.round(end - start, 3)

    loss, current = np.round(loss.item(), 5), batch * len(X)

  correct /= size
  loss /= num_batches

  print(f"Train: Accuracy: {(100*correct):>0.2f}%, Avg loss: {loss:>5f} \n")

  return loss, correct
```

In [ ]:

```
def valid_fn(valid_loader, model, criterion, device):
  model.eval()

  size = len(valid_loader.dataset)
  num_batches = len(valid_loader)

  loss, correct = 0, 0


  with torch.no_grad():
    for batch, (X, y) in enumerate(valid_loader):

      start = time.time()

      device = torch.device(device)
      X, y = X.to(device), y.to(device)

      pred = model(X)
      loss = criterion(pred, y.long().squeeze())
      current = batch * len(X)

      y_pred, y_true = torch.argmax(pred, axis=1), y.long().squeeze()
      correct += (y_pred == y_true).type(torch.float).sum().item()

      end = time.time()
      time_delta = np.round(end - start, 3)

      loss, current = np.round(loss.item(), 5), batch * len(X)

  correct /= size
  loss /= num_batches

  print(f"Valid: Accuracy: {(100*correct):>0.2f}%, Avg loss: {loss:>5f} \n")

  return loss, correct
```

In [ ]:

```
def test_fn(test_loader, model, criterion, device):
  model.eval()

  size = len(test_loader.dataset)
  num_batches = len(test_loader)

  loss, correct = 0, 0

  with torch.no_grad():
    for batch, (X, y) in enumerate(test_loader):

      start = time.time()

      device = torch.device(device)
      X, y = X.to(device), y.to(device)

      pred = model(X)
      loss = criterion(pred, y.long().squeeze())
      current = batch * len(X)

      y_pred, y_true = torch.argmax(pred, axis=1), y.long().squeeze()
      correct += (y_pred == y_true).type(torch.float).sum().item()

      end = time.time()
      time_delta = np.round(end - start, 3)

      loss, current = np.round(loss.item(), 5), batch * len(X)

  correct /= size
  loss /= num_batches

  print(f"Test: Accuracy: {(100*correct):>0.2f}%, Avg loss: {loss:>5f} \n")

  print(classification_report(y_true,y_pred))

  return loss, correct
```

Model 1 - Transfer Learning

In [ ]:

```
start = time.time()

loss_fn = nn.CrossEntropyLoss()

device = torch.device(device)
model = Model()

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

for param in model.cnn.classifier.parameters():
    param.requires_grad = True

optimizer = optim.Adam(model.parameters(), lr=1e-3,amsgrad = False)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min')

train_loss_history = []
train_acc_history = []
valid_loss_history = []
valid_acc_history = []
LR_history = []

best_loss = np.inf
best_epoch_loss = 0
best_acc = 0
best_epoch_acc = 0

print('Starting Training...\n')

start_train_time = time.time()

EPOCH = 10
for epoch in range(0, EPOCH):
  print(f"\n-------------------------------   Epoch {epoch + 1}   ---------------------
----------\n")
  start_epoch_time = time.time()

  train_loss, train_acc = train_fn(train_generator, model, loss_fn, optimizer, device)
  train_loss_history.append(train_loss)
  train_acc_history.append(train_acc)

  valid_loss, valid_acc = valid_fn(val_generator, model, loss_fn, device)
  valid_loss_history.append(valid_loss)
  valid_acc_history.append(valid_acc)


  scheduler.step(valid_loss)

  for param_group in optimizer.param_groups:
    LR_history.append(param_group['lr'])

  if valid_loss < best_loss:
    best_epoch_loss = epoch + 1
    best_loss = valid_loss
    torch.save(model.state_dict(), './' + f"Model_ep{best_epoch_loss}.pth")

  if valid_acc > best_acc:
    best_epoch_acc = epoch + 1
    best_acc = valid_acc
    torch.save(model.state_dict(), './' + f"Model_ep{best_epoch_acc}.pth")

  end_epoch_time = time.time()
  time_delta = np.round(end_epoch_time - start_epoch_time, 3)
  print("\n\nEpoch Elapsed Time: {} s".format(time_delta))

test_fn(test_generator, model, loss_fn, device)

end_train_time = time.time()
print("\n\nTotal Elapsed Time: {} min".format(np.round((end_train_time - start_train_time)/60, 3)))
print("Done!")
```

```
Starting Training...

-------------------------------   Epoch 1   ------------------------------
-


Train: Accuracy: 11.90%, Avg loss: 0.185967

Valid: Accuracy: 33.47%, Avg loss: 0.714884


Epoch Elapsed Time: 130.311 s

-------------------------------   Epoch 2   ------------------------------
-


Train: Accuracy: 59.55%, Avg loss: 0.112360

Valid: Accuracy: 61.02%, Avg loss: 0.471320


Epoch Elapsed Time: 124.49 s

-------------------------------   Epoch 3   ------------------------------
-


Train: Accuracy: 83.25%, Avg loss: 0.061283

Valid: Accuracy: 73.06%, Avg loss: 0.350450


Epoch Elapsed Time: 125.133 s

-------------------------------   Epoch 4   ------------------------------
-


Train: Accuracy: 90.96%, Avg loss: 0.031755

Valid: Accuracy: 77.14%, Avg loss: 0.246896


Epoch Elapsed Time: 118.539 s

-------------------------------   Epoch 5   ------------------------------
-
```

```
Train: Accuracy: 94.33%, Avg loss: 0.020068

Valid: Accuracy: 79.39%, Avg loss: 0.185212


Epoch Elapsed Time: 118.755 s

-------------------------------   Epoch 6   ------------------------------
-


Train: Accuracy: 97.09%, Avg loss: 0.013282

Valid: Accuracy: 81.43%, Avg loss: 0.194176


Epoch Elapsed Time: 119.797 s

-------------------------------   Epoch 7   ------------------------------
-


Train: Accuracy: 97.91%, Avg loss: 0.019956

Valid: Accuracy: 82.45%, Avg loss: 0.167180


Epoch Elapsed Time: 121.559 s

-------------------------------   Epoch 8   ------------------------------
-


Train: Accuracy: 99.13%, Avg loss: 0.009669

Valid: Accuracy: 83.06%, Avg loss: 0.192636


Epoch Elapsed Time: 121.29 s

-------------------------------   Epoch 9   ------------------------------
-
```

```
Train: Accuracy: 99.39%, Avg loss: 0.009702

Valid: Accuracy: 83.88%, Avg loss: 0.162574



Epoch Elapsed Time: 121.847 s

-------------------------------   Epoch 10   -----------------------------
--



Train: Accuracy: 99.59%, Avg loss: 0.009562

Valid: Accuracy: 83.67%, Avg loss: 0.143920



Epoch Elapsed Time: 120.479 s
Test: Accuracy: 83.50%, Avg loss: 0.090801

              precision    recall  f1-score   support

           2       1.00      1.00      1.00         1
           3       1.00      1.00      1.00         1
           7       0.00      0.00      0.00         1
          21       1.00      1.00      1.00         1
          22       0.00      0.00      0.00         1
          30       1.00      1.00      1.00         1
          31       1.00      1.00      1.00         1
          51       0.00      0.00      0.00         0
          54       1.00      1.00      1.00         1
          58       0.00      0.00      0.00         0
          75       1.00      1.00      1.00         1
          85       1.00      1.00      1.00         1
          92       1.00      1.00      1.00         1
          96       1.00      1.00      1.00         1

    accuracy                           0.83        12
   macro avg       0.71      0.71      0.71        12
weighted avg       0.83      0.83      0.83        12



Total Elapsed Time: 20.771 min
Done!
```

Model 2 - Randomly initializing weights

In [17]:

```
start = time.time()

loss_fn = nn.CrossEntropyLoss()

device = torch.device(device)
model2 = model = Model(pretrained = False)
# model2.classifier[3] = torch.nn.Linear(in_features=1024, out_features=102)

optimizer = optim.Adam(model2.parameters(), lr=1e-3,amsgrad = False)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min')

train_loss_history = []
train_acc_history = []
valid_loss_history = []
valid_acc_history = []
LR_history = []

best_loss = np.inf
best_epoch_loss = 0
best_acc = 0
best_epoch_acc = 0

print('Starting Training...\n')

start_train_time = time.time()

EPOCH = 10
for epoch in range(0, EPOCH):
  print(f"\n-------------------------------   Epoch {epoch + 1}   ---------------------
----------\n")
  start_epoch_time = time.time()

  train_loss, train_acc = train_fn(train_generator, model2, loss_fn, optimizer, device)
  train_loss_history.append(train_loss)
  train_acc_history.append(train_acc)

  valid_loss, valid_acc = valid_fn(val_generator, model2, loss_fn, device)
  valid_loss_history.append(valid_loss)
  valid_acc_history.append(valid_acc)


  scheduler.step(valid_loss)

  for param_group in optimizer.param_groups:
    LR_history.append(param_group['lr'])

  if valid_loss < best_loss:
    best_epoch_loss = epoch + 1
    best_loss = valid_loss
    torch.save(model2.state_dict(), './' + f"Model2_ep{best_epoch_loss}.pth")

  if valid_acc > best_acc:
    best_epoch_acc = epoch + 1
    best_acc = valid_acc
    torch.save(model2.state_dict(), './' + f"Model2_ep{best_epoch_acc}.pth")

  end_epoch_time = time.time()
  time_delta = np.round(end_epoch_time - start_epoch_time, 3)
  print("\n\nEpoch Elapsed Time: {} s".format(time_delta))
```

```
test_fn(test_generator, model2, loss_fn, device)

end_train_time = time.time()

print("\n\nTotal Elapsed Time: {} min".format(np.round((end_train_time - start_train_ti
me)/60, 3)))

print("Done!")
```

```
Starting Training...

-------------------------------   Epoch 1   ------------------------------
-


Train: Accuracy: 1.33%, Avg loss: 0.227358

Valid: Accuracy: 1.02%, Avg loss: 0.995036


Epoch Elapsed Time: 300.973 s

-------------------------------   Epoch 2   ------------------------------
-


Train: Accuracy: 4.49%, Avg loss: 0.229224

Valid: Accuracy: 4.69%, Avg loss: 0.926176


Epoch Elapsed Time: 283.015 s

-------------------------------   Epoch 3   ------------------------------
-


Train: Accuracy: 7.66%, Avg loss: 0.197466

Valid: Accuracy: 6.94%, Avg loss: 0.906132


Epoch Elapsed Time: 289.823 s

-------------------------------   Epoch 4   ------------------------------
-


Train: Accuracy: 19.92%, Avg loss: 0.193359

Valid: Accuracy: 9.59%, Avg loss: 0.853344


Epoch Elapsed Time: 282.731 s

-------------------------------   Epoch 5   ------------------------------
-
```

```
Train: Accuracy: 35.65%, Avg loss: 0.136618

Valid: Accuracy: 11.02%, Avg loss: 0.899154


Epoch Elapsed Time: 286.288 s

-------------------------------   Epoch 6   ------------------------------
-


Train: Accuracy: 58.94%, Avg loss: 0.077153

Valid: Accuracy: 7.76%, Avg loss: 1.204748


Epoch Elapsed Time: 280.705 s

-------------------------------   Epoch 7   ------------------------------
-


Train: Accuracy: 72.83%, Avg loss: 0.061795

Valid: Accuracy: 7.55%, Avg loss: 1.185458


Epoch Elapsed Time: 278.902 s

-------------------------------   Epoch 8   ------------------------------
-


Train: Accuracy: 80.18%, Avg loss: 0.056622

Valid: Accuracy: 10.00%, Avg loss: 1.313528


Epoch Elapsed Time: 277.843 s

-------------------------------   Epoch 9   ------------------------------
-
```

```
Train: Accuracy: 83.35%, Avg loss: 0.032853

Valid: Accuracy: 11.02%, Avg loss: 1.335812



Epoch Elapsed Time: 276.799 s

-------------------------------   Epoch 10   -----------------------------
--



Train: Accuracy: 88.82%, Avg loss: 0.026949

Valid: Accuracy: 11.02%, Avg loss: 1.354808



Epoch Elapsed Time: 276.922 s
Test: Accuracy: 12.58%, Avg loss: 0.955547

              precision    recall  f1-score   support

           1       0.00      0.00      0.00         0
          10       0.00      0.00      0.00         0
          15       0.00      0.00      0.00         1
          23       0.00      0.00      0.00         1
          25       0.00      0.00      0.00         1
          26       0.00      0.00      0.00         1
          31       0.00      0.00      0.00         1
          38       0.00      0.00      0.00         0
          45       0.00      0.00      0.00         1
          46       0.00      0.00      0.00         0
          47       0.00      0.00      0.00         0
          50       1.00      1.00      1.00         1
          57       0.00      0.00      0.00         0
          60       0.00      0.00      0.00         1
          70       0.00      0.00      0.00         0
          74       0.00      0.00      0.00         1
          75       0.00      0.00      0.00         0
          85       0.00      0.00      0.00         1
          89       0.00      0.00      0.00         1
          96       0.00      0.00      0.00         1
          97       0.00      0.00      0.00         0
          99       0.00      0.00      0.00         0

    accuracy                           0.08        12
   macro avg       0.05      0.05      0.05        12
weighted avg       0.08      0.08      0.08        12



Total Elapsed Time: 47.597 min
Done!
```

In [ ]: