Importing Libraries

In [1]:

```
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 plotly.express as px

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: torchvision in /usr/local/lib/python3.7/dis
t-packages (from timm) (0.13.1+cu113)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dist-pac
kages (from timm) (6.0)
Requirement already satisfied: torch>=1.7 in /usr/local/lib/python3.7/dist
-packages (from timm) (1.12.1+cu113)
Requirement already satisfied: huggingface-hub in /usr/local/lib/python3.
7/dist-packages (from timm) (0.10.0)
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: requests in /usr/local/lib/python3.7/dist-p
ackages (from huggingface-hub->timm) (2.23.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packa
ges (from huggingface-hub->timm) (4.64.1)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python
3.7/dist-packages (from huggingface-hub->timm) (5.0.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-p
ackages (from huggingface-hub->timm) (3.8.0)
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.
7/dist-packages (from huggingface-hub->timm) (21.3)
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: idna<3,>=2.5 in /usr/local/lib/python3.7/di
st-packages (from requests->huggingface-hub->timm) (2.10)
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: certifi>=2017.4.17 in /usr/local/lib/python
3.7/dist-packages (from requests->huggingface-hub->timm) (2022.9.24)
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: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/pyt
hon3.7/dist-packages (from torchvision->timm) (7.1.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-pack
ages (from torchvision->timm) (1.21.6)
```

Setting Seed

In [2]:

```
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 # keep True if all the input have same size.
SEED=42
seed_everything(SEED=SEED)
```

In [3]:

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

Dataset Creation

In [4]:

```
a = []
b = []
c = []
for i in tqdm(range(10000)):
  l = random.randint(2,10)
  x = [random.uniform(0,1) for _ in range(l)]
  y = [0 for _ in range (l)]
  i1,i2 = random.sample(range(l),2)
  y[i1] = 1
  y[i2] = 1
  z = sum(np.array(x)*np.array(y))
  a.append(x)
  b.append(y)
  c.append(z)

df = pd.DataFrame({'A':a,'B':b,'Target':c})

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

In [5]:

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

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

    def __getitem__(self,index):

        x = torch.tensor(np.array([i for i in zip(self.df.iloc[index,0],self.df.iloc[in
dex,1])]),dtype=torch.float)
        y = torch.tensor(df.iloc[index,2],dtype=torch.float)
        return x,y

train_generator = DataLoader(Data(df_train),batch_size = 1,shuffle = True)
val_generator = DataLoader(Data(df_val),batch_size = 1,shuffle = True)
test_generator = DataLoader(Data(df_test),batch_size = 1,shuffle = True)
```

Model Creation

In [6]:

```
class Model(nn.Module):
  def __init__(self, model_name, input_size = 2, hidden_size = 1, num_layers = 1 , num_
classes = 1):
    super(Model, self).__init__()
    self.model_name = model_name
    self.num_layers = num_layers
    self.hidden_size = hidden_size

    if model_name == 'RNN':
      self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True)
    elif model_name == 'LSTM':
      self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
    elif model_name =='GRU':
      self.gru = nn.GRU(input_size, hidden_size, num_layers, batch_first=True)

    # self.fc = nn.Linear(hidden_size, num_classes)

    print('Initialized', model_name)

  def forward(self, x):

    if self.model_name == 'RNN':
      h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device)

      out, _ = self.rnn(x, h0)
      out = out[:, -1, :]
      # out = self.fc(out)
      return out

    elif self.model_name == 'LSTM':
      h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device)
      c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device)

      out, _ = self.lstm(x, (h0,c0))
      out = out[:, -1, :]
      return out

    elif self.model_name =='GRU':
      h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device)

      out, _ = self.gru(x, h0)
      out = out[:, -1, :]
      return out
```

Train, Validation and Test Functions

In [7]:

```
def train_fn(model,criterion,optimizer,num_epochs,train_generator,val_generator):
  avg_train_loss = []
  avg_val_loss = []
  for epoch in tqdm(range(num_epochs)):

    train_loss = []

    for i, (x,y) in enumerate(train_generator):

      model.train()
      outputs = model(x)
      loss = criterion(outputs, y)
      train_loss.append(loss)
      optimizer.zero_grad()
      loss.backward()
      optimizer.step()

    val_loss = valid_fn(model,criterion,val_generator)

    avg_train_loss.append((sum(train_loss)/len(train_loss)).item())
    avg_val_loss.append((sum(val_loss)/len(val_loss)).item())

    print('Epoch {}/{} | Average Training Loss = {:.3f} | Average Validation Loss = {:.
3f}'.format(epoch+1,num_epochs,avg_train_loss[-1],avg_val_loss[-1]))

  return avg_train_loss, avg_val_loss
```

In [8]:

```
def valid_fn(model,criterion,val_generator):
  val_loss = []
  for i, (x,y) in enumerate(val_generator):
      model.eval()
      preds = model(x)
      loss = criterion(preds, y)
      val_loss.append(loss)
  return val_loss
```

In [9]:

```
def test_fn(model,criterion,test_generator):
  test_loss = []
  for i, (x,y) in enumerate(test_generator):
      model.eval()
      preds = model(x)
      loss = criterion(preds, y)
      test_loss.append(loss)
  print('Average Testing Loss =',(sum(test_loss)/len(test_loss)).item())
  # return test_loss
```

RNN

In [10]:

```
rnn = Model(model_name = 'RNN', input_size = 2, hidden_size = 1, num_layers = 1 , num_c
lasses = 1).to(device)
avg_train_loss_rnn, avg_val_loss_rnn = train_fn(model = rnn,criterion = nn.MSELoss(),op
timizer = torch.optim.Adam(rnn.parameters(), lr= 1e-4) ,num_epochs=5,train_generator =
train_generator,val_generator = val_generator)
test_fn(model = rnn,criterion = nn.MSELoss(),test_generator = test_generator)
```

```
Initialized RNN


Epoch 1/5 | Average Training Loss = 0.189 | Average Validation Loss = 0.16
9
Epoch 2/5 | Average Training Loss = 0.170 | Average Validation Loss = 0.16
3
Epoch 3/5 | Average Training Loss = 0.166 | Average Validation Loss = 0.16
1
Epoch 4/5 | Average Training Loss = 0.165 | Average Validation Loss = 0.16
1
Epoch 5/5 | Average Training Loss = 0.165 | Average Validation Loss = 0.16
0
Average Testing Loss = 0.16035017371177673
```

LSTM

In [11]:

```
lstm = Model(model_name = 'LSTM', input_size = 2, hidden_size = 1, num_layers = 1 , num
_classes = 1).to(device)
avg_train_loss_lstm, avg_val_loss_lstm = train_fn(model = lstm,criterion = nn.MSELoss
(),optimizer = torch.optim.Adam(lstm.parameters(), lr= 1e-4),num_epochs=5,train_generat
or = train_generator,val_generator = val_generator)
test_fn(model = lstm,criterion = nn.MSELoss(),test_generator = test_generator)
```

```
Initialized LSTM


Epoch 1/5 | Average Training Loss = 0.951 | Average Validation Loss = 0.35
6
Epoch 2/5 | Average Training Loss = 0.238 | Average Validation Loss = 0.18
4
Epoch 3/5 | Average Training Loss = 0.180 | Average Validation Loss = 0.16
8
Epoch 4/5 | Average Training Loss = 0.171 | Average Validation Loss = 0.16
4
Epoch 5/5 | Average Training Loss = 0.168 | Average Validation Loss = 0.16
3
Average Testing Loss = 0.1632545292377472
```

GRU

In [14]:

```
gru = Model(model_name = 'GRU', input_size = 2, hidden_size = 1, num_layers = 1 , num_c
lasses = 1).to(device)
avg_train_loss_gru, avg_val_loss_gru = train_fn(model = gru,criterion = nn.MSELoss(),op
timizer = torch.optim.Adam(gru.parameters(), lr= 1e-4),num_epochs=2,train_generator = t
rain_generator,val_generator = val_generator)
test_fn(model = gru,criterion = nn.MSELoss(),test_generator = test_generator)
```

```
Initialized GRU


Epoch 1/2 | Average Training Loss = 1.274 | Average Validation Loss = 0.33
5
Epoch 2/2 | Average Training Loss = 0.213 | Average Validation Loss = 0.17
0
Average Testing Loss = 0.1701526790857315
```

Benchmark

In [15]:

```
test_loss = []
for i, (x,y) in enumerate(test_generator):
    loss = nn.MSELoss()(torch.tensor(1), y)
    test_loss.append(loss)
print('Average Testing Loss (when sum is always predicted as 1) =',(sum(test_loss)/len(
test_loss)).item())
```

```
Average Testing Loss (when sum is always predicted as 1) = 0.1600903272628
7842
```

In [13]:

```

```