week - 1

Course name : Build a Convolution Neural Network for Image Recognition

Program:

```
[1] import numpy as np
    import tensorflow as tf
    from tensorflow.keras import layers,models
    from tensorflow.keras.datasets import mnist
```

```
[2] (train_img, train_label),(test_img, test_label) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 1s 0us/step
```

```
[3] train_img , test_img = train_img/255.0, test_img/255.0
    train_label = tf.keras.utils.to_categorical(train_label, 10)
    test_label = tf.keras.utils.to_categorical(test_label, 10)
```



```
[4] #Building the CNN model
    model = models.Sequential()
    model.add(layers.Conv2D(32,(3,3),activation ='relu', input_shape = (28, 28, 1)))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3,3), activation ='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Flatten())
    model.add(layers.Dense(64,activation = 'relu'))
    model.add(layers.Dense(10,activation = 'softmax'))
```

```
    model.compile(optimizer = 'adam', loss ='categorical_crossentropy',metrics = ['accuracy'])
```

```
#Train the model on the training data
model.fit(train_img.reshape(-1, 28, 28, 1), train_label, epochs=5, batch_size=64)
```

```
Epoch 1/5
938/938 [==============================] - 49s 51ms/step - loss: 0.1802 - accuracy: 0.9467
Epoch 2/5
938/938 [==============================] - 49s 52ms/step - loss: 0.0496 - accuracy: 0.9851
Epoch 3/5
938/938 [==============================] - 48s 51ms/step - loss: 0.0351 - accuracy: 0.9894
Epoch 4/5
938/938 [==============================] - 48s 51ms/step - loss: 0.0276 - accuracy: 0.9914
Epoch 5/5
938/938 [==============================] - 47s 51ms/step - loss: 0.0217 - accuracy: 0.9931
<keras.src.callbacks.History at 0x7c13f0934e50>
```

```
#Evaluate the model on the test data
test_loss , test_accuracy = model.evaluate(test_img.reshape(-1,28,28,1),test_label)
print("Test accuracy : ",test_accuracy)
```

```
313/313 [==============================] - 3s 8ms/step - loss: 0.0328 - accuracy: 0.9902
Test accuracy :  0.9901999831199646
```

Week - 2

Module name : Understanding and Using ANN : Identifying age group of an actor

Exercise : Design Artificial Neural Networks for Identifying and Classifying an actor using Kaggle Dataset.

Program:

```
from tensorflow.keras.models import load_model
from PIL import Image
import numpy as np

image_height = 128
image_width = 128
num_channels = 3

model = load_model('trained_model_NEW_2_2_Dataset.h5')

new_face_path = '/content/hjhhh.jpg'
new_face = Image.open(new_face_path)

display(new_face)
new_face = new_face.resize((image_width, image_height))
new_face = np.array(new_face)
new_face = np.expand_dims(new_face, axis=0)
predictions = model.predict(new_face)
predicted_age_group = np.argmax(predictions)
print("predictions are ",predictions)
print("Predicted Age Group:", predicted_age_group)
age_mapping = {0: 'YOUNG', 1: 'MIDDLE', 2: 'OLD'}
predicted_age_group_label = age_mapping[predicted_age_group]

print("Predicted Age Group:", predicted_age_group_label)
```

OUTPUT:

```
1/1 [==============================] - 0s 428ms/step predictions are
[[0.6614267 0.12491771 0.21365556]] Predicted Age Group: 0 Predicted Age
Group: YOUN
```

Week - 6
Module Name: Advanced Deep Learning Architectures
Exercise: Implement Object Detection Using YOLO.

Program:

```
!wget https://raw.githubusercontent.com/pjreddie/darknet/master/data/coco.names
```

```
--2023-11-17 08:57:51--  https://raw.githubusercontent.com/pjreddie/darknet/master/data/coco.names
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.110.133, 185.199.111.133,
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.110.133|:443... connected
HTTP request sent, awaiting response... 200 OK
Length: 625 [text/plain]
Saving to: 'coco.names'

coco.names          100%[===================>]     625  --.-KB/s    in 0s

2023-11-17 08:57:52 (20.3 MB/s) - 'coco.names' saved [625/625]
```

```
!wget https://pjreddie.com/media/files/yolov3.weights
```

```
--2023-11-17 09:10:36--  https://pjreddie.com/media/files/yolov3.weights
Resolving pjreddie.com (pjreddie.com)... 128.208.4.108
Connecting to pjreddie.com (pjreddie.com)|128.208.4.108|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 248007048 (237M) [application/octet-stream]
Saving to: 'yolov3.weights'

yolov3.weights      100%[===================>] 236.52M  46.0MB/s    in 5.6s

2023-11-17 09:10:42 (42.0 MB/s) - 'yolov3.weights' saved [248007048/248007048]
```

```
!wget https://raw.githubusercontent.com/pjreddie/darknet/master/cfg/yolov3.cfg
```

```
--2023-11-17 09:11:02--  https://raw.githubusercontent.com/pjreddie/darknet/master/cfg/yolov3.cfg
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133,
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected
HTTP request sent, awaiting response... 200 OK
Length: 8342 (8.1K) [text/plain]
Saving to: 'yolov3.cfg'

yolov3.cfg          100%[===================>]   8.15K  --.-KB/s    in 0s

2023-11-17 09:11:02 (72.7 MB/s) - 'yolov3.cfg' saved [8342/8342]
```

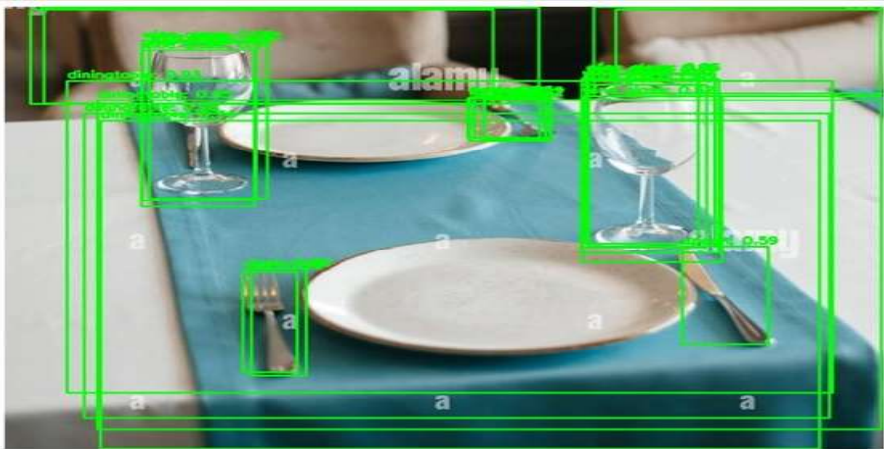
```
[4]  import cv2
     import numpy as np
```

```
[5]  net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")
```

```
with open("coco.names","r") as f:
  classes=f.read().strip().split("\n")
  image = cv2.imread("/content/dine.jpg") #Upload any image
  height, width = image.shape[:2]
  blob = cv2.dnn.blobFromImage(image, 1/255.0, (416, 416), swapRB=True, crop=False)
  net.setInput(blob)
  layer_names = net.getUnconnectedOutLayersNames()
  detections = net.forward(layer_names)
```

```
for detection in detections:
  for obj in detection:
    scores = obj[5:]
    class_id = np.argmax(scores)
    confidence = scores[class_id]
    if confidence >0.5: # Adjust the confidence threshold as needed
      center_x = int(obj[0] * width)
      center_y = int(obj[1] * height)
      w = int(obj[2] * width)
      h = int(obj[3] * height)
      # Calculate bounding box coordinates
      x = int(center_x - w / 2)
      y = int(center_y - h / 2)
      # Draw bounding box and label on the image
      cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
      label = f"{classes[class_id]}: {confidence:.2f}";
      cv2.putText(image, label, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
```

```
from google.colab.patches import cv2_imshow
cv2_imshow(image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Week - 8
Module name: Advanced CNN
Exercise: Build AlexNet using Advanced CNN

Program:

```
import torch
import torch.nn as nn
class AlexNet(nn.Module):
  def __init__(self,num_classes=1000):
    super(AlexNet,self).__init__()
    self.features=nn.Sequential(
      nn.Conv2d(3,64,kernel_size=11,stride=4,padding=2),
      nn.ReLU(inplace=True),
      nn.MaxPool2d(kernel_size=3,stride=2),
      nn.Conv2d(64,192,kernel_size=5,padding=2),
      nn.ReLU(inplace=True),
      nn.MaxPool2d(kernel_size=3,stride=2),
      nn.Conv2d(192,384,kernel_size=3,padding=1),
      nn.ReLU(inplace=True),
      nn.Conv2d(384,256,kernel_size=3,padding=1),
      nn.ReLU(inplace=True),
      nn.Conv2d(256,256,kernel_size=3,padding=1),
      nn.ReLU(inplace=True),
      nn.MaxPool2d(kernel_size=3,stride=2),
    )
    self.avgpool=nn.AdaptiveAvgPool2d((6,6))
    self.classifier=nn.Sequntial(
      nn.Dropout(),
      nn.Linear(256*6*6,4096),
      nn.ReLU(inplace=True),
      nn.Dropout(),
      nn.Linear(4096,4096),
      nn.ReLU(inplace=True),
      nn.Linear(4096,num_classes),
     )
    def forward(self,x):
      x=self.features(x)
      x=self.avgpool(x)
      x=torch.flatten(x,1)
      x=self.classifier(x)
      return x

    model=AlexNet()
    print(model)
```

```python
import torch
from torchvision import models,transforms
from PIL import Image
import requests
import io
import matplotlib.pyplot as plt
preprocess=transforms.Compose([
  transforms.Resize(256),
  transforms.CenterCrop(224),
  transforms.ToTensor(),
  transforms.Normalize(mean=[0.485,0.456,0.406],std=[0.229,0.224,0.225]),
])

model=models.alexnet(pretrained=True)
model.eval()
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning:
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/alexnet-owt-7be5be79.pth" to /root/.
100%|██████████| 233M/233M [00:02<00:00, 101MB/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)
```

```
url="https://yellowverandah.in/cdn/shop/products/IMG_1660_810x.jpg?v=1664791983"
 #url for a vase (we can give any link of the images related in LABELS_URL)
response=requests.get(url)
img=Image.open(io.BytesIO(response.content))
img_tensor=preprocess(img)
img_tensor=img_tensor.unsqueeze(0)
with torch.no_grad():
  output=model(img_tensor)
```

```
[6] LABELS_URL="https://raw.githubusercontent.com/anishathalye/imagenet-simple-labels/master/imagenet-simple-labels.json"
    labels = requests.get(LABELS_URL).json()
    _,predicted_idx=torch.max(output,1)
    predicted_label=labels[predicted_idx.item()]
```

```
plt.imshow(img)
plt.title(f"predicted Label:{predicted_label}")
plt.axis('off')
plt.show()
```

Week - 10
Module name: Advanced GANs
Exercise: Demonstration of GAN.

Program:

```
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
import torch
import torch.nn as nn
device=torch.device("cuda" if torch.cuda.is_available() else "cpu")
n=1000
first_column=torch.rand(n,1).to(device)
second_column=2*first_column
third_column=2*second_column
data=torch.cat([first_column,second_column,third_column],dim=1)

class Generator(nn.Module):
  def __init_(self):
    super(Generator,self)._init_()
    self.model=nn.Sequential(
        nn.Linear(3,50),
        nn.ReLU(),
        nn.Linear(50,3)
    )
  def forward(self,x):
    return self.model(x)

class Discriminator(nn.Module):
  def __init_(self):
    super(Discriminator,self)._init_()
    self.model=nn.Sequential(
       nn.Linear(3,50),
       nn.ReLU(),
       nn.Linear(50,1),
       nn.Sigmoid()
    )
  def forward(self,x):
    return self.model(x)
```

```
generator=Generator().to(device)
discriminator=Discriminator().to(device)

criterion=nn.BCELoss()
optimizer_g=torch.optim.Adam(generator.parameters(),lr=0.001)
optimizer_d=torch.optim.Adam(discriminator.parameters(),lr=0.001)

num_epochs=5000
for epoch in range(num_epochs):
  optimizer_d.zero_grad()
  real_data=data
  real_labels=torch.ones(n,1).to(device)
  outputs=discriminator(real_data)
  d_loss_real=criterion(outputs,real_labels)
  noise=torch.randn(n,3).to(device)
  fake_data=generator(noise)
  fake_labels=torch.zeros(n,1).to(device)
  outputs=discriminator(fake_data.detach())
  d_loss_fake=criterion(outputs,fake_labels)
  d_loss=d_loss_real + d_loss_fake
  d_loss.backward()
  optimizer_d.step()
  optimizer_g.zero_grad()
  outputs=discriminator(fake_data)
  g_loss=criterion(outputs,real_labels)
  g_loss.backward()
  optimizer_g.step()
  if(epoch+1)%1000==0:

print(f"Epoch[{epoch+1}/{num_epochs}],d_loss:{d_loss.item():.4f},g_loss:{g_l
oss.item():.4f}")
```

```
Epoch[1000/5000],d_loss:1.3726,g_loss:0.7051
Epoch[2000/5000],d_loss:1.3824,g_loss:0.6924
Epoch[3000/5000],d_loss:1.3824,g_loss:0.6726
Epoch[4000/5000],d_loss:1.3874,g_loss:0.7106
Epoch[5000/5000],d_loss:1.3864,g_loss:0.7042
```

```
with torch.no_grad():
  test_noise=torch.randn(n, 3).to(device)
  generated_data=generator(test_noise).cpu().numpy()
```

```
print("Generated Data(First 10 rows):")
for i in range(10):
  print(generated_data[i])
```

```
Generated Data(First 10 rows):
[0.82988614 1.7327724  3.5752788 ]
[0.06838156 0.1616691  0.37241268]
[0.6004674 1.2439212 2.5554523]
[0.36136454 0.7555432  1.5671514 ]
[0.8590026 1.8045118 3.7318196]
[0.6465259 1.3427644 2.761265 ]
[0.663725  1.3844492 2.864199 ]
[0.84645295 1.7821548  3.6963816 ]
[0.86263525 1.8166327  3.768263  ]
[0.8562646 1.8023043 3.7431922]
```

```
print("\nValidation (For the first 10 rows):")
for i in range(10):
        print(f"First:{generated_data[i][0]:.4f},
        Expected Second:{2*generated_data[i][0]:.4f},
        Actual Second: {generated_data[i][1]:.4f}")
        print(f"Second:{generated_data[i][1]:.4f},
        Expected Third:{2*generated_data[i][1]:.4f},
        Actual Third:{generated_data[i][2]:.4f}\n")
```

```
Validation (For the first 10 rows):
First:0.8299,Expected Second:1.6598,Actual Second: 1.7328
Second:1.7328,Expected Third:3.4655, Actual Third:3.5753

First:0.0684,Expected Second:0.1368,Actual Second: 0.1617
Second:0.1617,Expected Third:0.3233, Actual Third:0.3724

First:0.6005,Expected Second:1.2009,Actual Second: 1.2439
Second:1.2439,Expected Third:2.4878, Actual Third:2.5555

First:0.3614,Expected Second:0.7227,Actual Second: 0.7555
Second:0.7555,Expected Third:1.5111, Actual Third:1.5672

First:0.8590,Expected Second:1.7180,Actual Second: 1.8045
Second:1.8045,Expected Third:3.6090, Actual Third:3.7318

First:0.6465,Expected Second:1.2931,Actual Second: 1.3428
Second:1.3428,Expected Third:2.6855, Actual Third:2.7613

First:0.6637,Expected Second:1.3275,Actual Second: 1.3844
Second:1.3844,Expected Third:2.7689, Actual Third:2.8642

First:0.8465,Expected Second:1.6929,Actual Second: 1.7822
Second:1.7822,Expected Third:3.5643, Actual Third:3.6964

First:0.8626,Expected Second:1.7253,Actual Second: 1.8166
Second:1.8166,Expected Third:3.6333, Actual Third:3.7683

First:0.8563,Expected Second:1.7125,Actual Second: 1.8023
Second:1.8023,Expected Third:3.6046, Actual Third:3.7432
```

Week 7
Module Name: Optimization of Training in Deep Learning
Exercise :Design a Deep learning Network for Robust Bi-Tempered Logistic Loss.

Program:

Step-1: Define the sigmoid function

```
import math
def sigmoid_func(x):
  return 1.0/(1+math.exp(-x))
sigmoid_func(100)
sigmoid_func(-100)
sigmoid_func(0)
```

Output:

```
1.0
3.7200759760208356e-44
0.5
```

Step-2: Create an array series

```
import pandas as pd
import numpy as np
x = pd.Series(np.arange(-8, 8, 0.5))
y = x.map(sigmoid_func)
print(x)
```

Output:

```
0    -8.0
1    -7.5
2    -7.0
3    -6.5
4    -6.0
5    -5.5
6    -5.0
7    -4.5
8    -4.0
9    -3.5
10   -3.0
11   -2.5
12   -2.0
13   -1.5
14   -1.0
15   -0.5
16    0.0
```

```
17    0.5
18    1.0
19    1.5
20    2.0
21    2.5
22    3.0
23    3.5
24    4.0
25    4.5
26    5.0
27    5.5
28    6.0
29    6.5
30    7.0
31    7.5
dtype: float64
print(y)
```

Output:

```
0     0.000335
1     0.000553
2     0.000911
3     0.001501
4     0.002473
5     0.004070
6     0.006693
7     0.010987
8     0.017986
9     0.029312
10    0.047426
11    0.075858
12    0.119203
13    0.182426
14    0.268941
15    0.377541
16    0.500000
17    0.622459
18    0.731059
19    0.817574
20    0.880797
21    0.924142
22    0.952574
23    0.970688
```

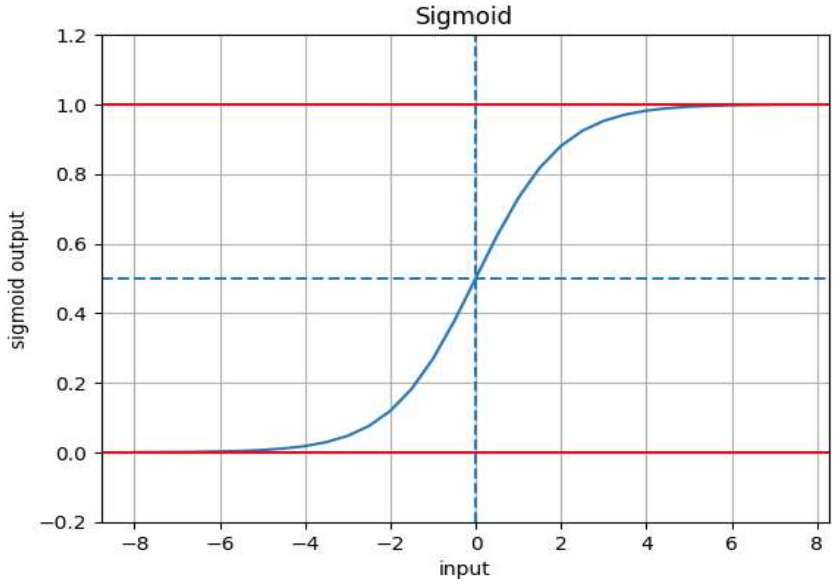
```
24    0.982014
25    0.989013
26    0.993307
27    0.995930
28    0.997527
29    0.998499
30    0.999089
31    0.999447
dtype: float64
```

Step-3: Plot the above generated series

```
import matplotlib.pyplot as plt

plt.plot(x, y)
plt.ylim(-0.2, 1.2)
plt.xlabel("input")
plt.ylabel("sigmoid output")
plt.grid(True)

plt.axvline(x=0, ymin=0, ymax=1, ls='dashed')
plt.axhline(y=0.5, xmin=0, xmax=10, ls='dashed')
plt.axhline(y=1.0, xmin=0, xmax=10, color='r')
plt.axhline(y=0.0, xmin=0, xmax=10, color='r')
plt.title("Sigmoid")
```

Output:


```
Text(0.5, 1.0, 'Sigmoid')
```



Step-6: Download the following dataset from kaggle
Link: https://www.kaggle.com/datasets/naveengowda16/logistic-regression-heart-disease-prediction

Step-7: import the dataset in google colab

```
import pandas as pd
df = pd.read_csv('/content/framingham_heart_disease.csv')
```

Step-8: Execute the straight line function

```
def straight_line(x):
  return 1.5046*x - 4.0777

def straight_line_weight(weight, x):
  return weight*x - 4.0777
```

Step-9: implement the sigmoid function on the column "cigsPerDay"

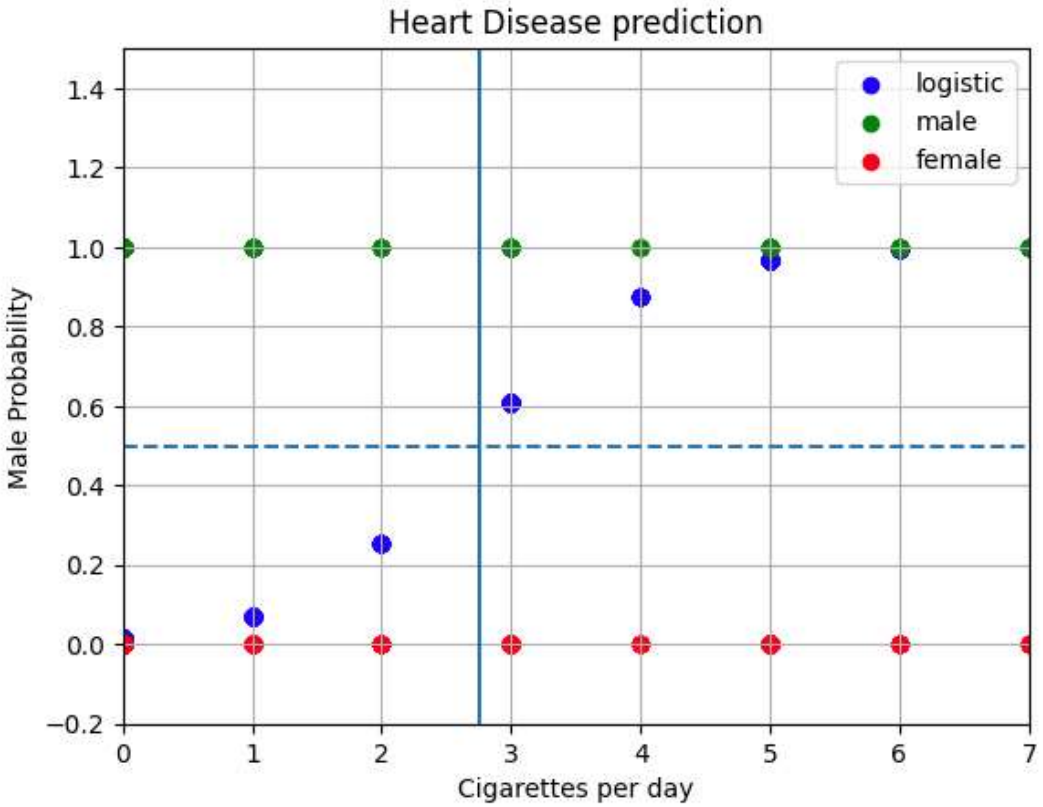
```
y_vals = df.cigsPerDay.map(straight_line).map(sigmoid_func)
```

Step-10: Plot the values

```
import matplotlib.pyplot as plt

plt.scatter(x=df.cigsPerDay, y=y_vals, color='b', label='logistic')
plt.scatter(x=df[df.male==1].cigsPerDay, y=df[df.male==1].male, color='g',
label='male')
plt.scatter(x=df[df.male==0].cigsPerDay, y=df[df.male==0].male, color='r',
label='female')
plt.title("Heart Disease prediction")
plt.xlabel("Cigarettes per day")
plt.ylabel("Male Probability")
plt.grid(True)
plt.legend()
plt.xlim((0, 7))
plt.ylim((-0.2, 1.5))
plt.axvline(x=2.75, ymin=0, ymax=1)
plt.axhline(y=0.5, xmin=0, xmax=6, label="cutoff at 0.5", ls='dashed')
```

Output:

```
<matplotlib.lines.Line2D at 0x7f913627f400>
```



Week - 4
Module name : Predicting Sequential Data
Exercise: Implement a Recurrence Neural Network for Predicting Sequential Data

Program:

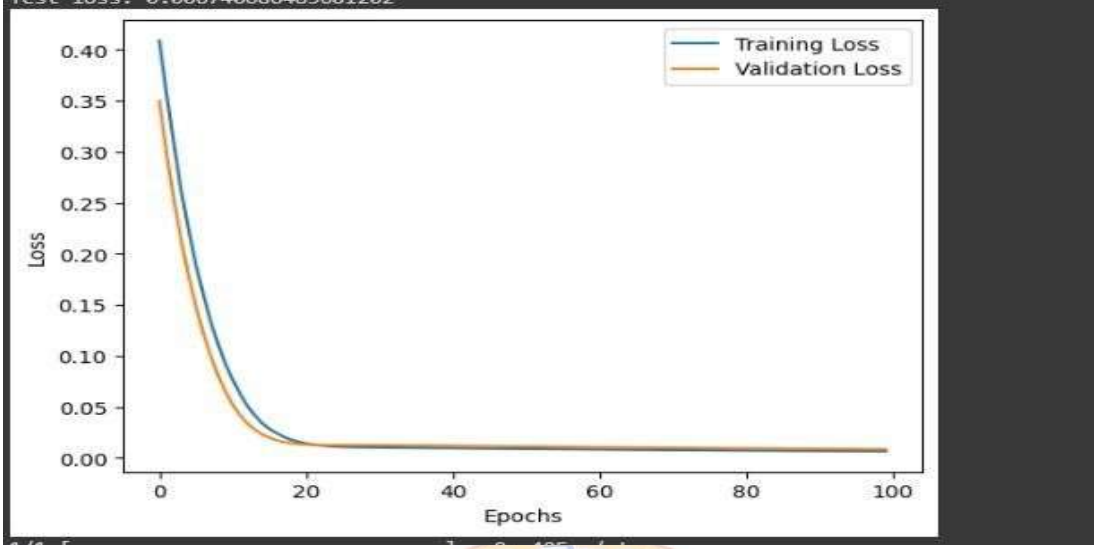
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Generate some example sequential data (you should replace this with
your own data)
# In this example, we'll use a simple sine wave.
sequence_length = 100
timesteps = np.linspace(0, 10, sequence_length)
data = np.sin(timesteps)

# Create sequences and labels
X = []
y = []
for i in range(sequence_length - 1):
    X.append(data[i:i+1])
    y.append(data[i+1])

X = np.array(X)
y = np.array(y)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Build the RNN model
model = Sequential()
model.add(SimpleRNN(50, activation='relu', input_shape=(1, 1))) # 50
units in the RNN layer
model.add(Dense(1)) # Output layer with 1 neuron

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')
```

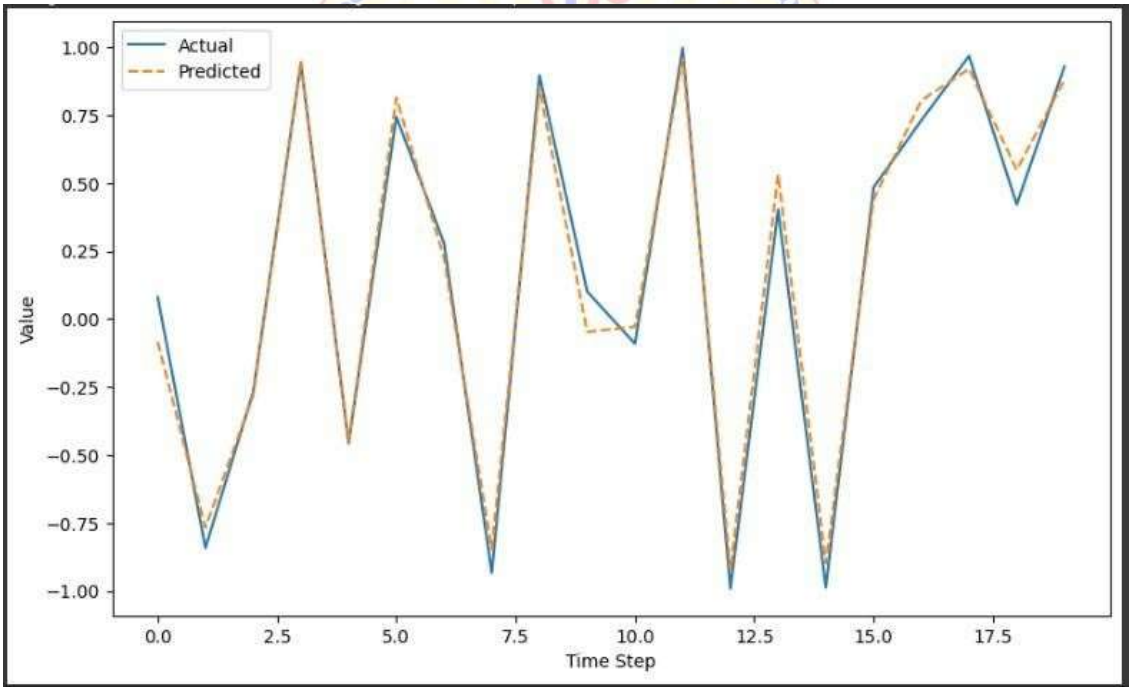
```
# Train the model
history = model.fit(X_train, y_train, epochs=100, batch_size=16,
validation_split=0.2)

# Evaluate the model
loss = model.evaluate(X_test, y_test)
print(f'Test loss: {loss}')

# Plot training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Predict using the trained model
predicted_values = model.predict(X_test)

# Plot the actual vs. predicted values
plt.figure(figsize=(10, 6))
plt.plot(y_test, label='Actual')
plt.plot(predicted_values, label='Predicted', linestyle='--')
plt.xlabel('Time Step')
plt.ylabel('Value')
plt.legend()
plt.show()
```

Output

```
Epoch 1/100
4/4 [==============================] - 2s 139ms/step - loss: 0.4087 - val_loss: 0.3496
Epoch 2/100
4/4 [==============================] - 0s 16ms/step - loss: 0.3562 - val_loss: 0.2980
Epoch 3/100
4/4 [==============================] - 0s 18ms/step - loss: 0.3069 - val_loss: 0.2524
Epoch 4/100
4/4 [==============================] - 0s 26ms/step - loss: 0.2610 - val_loss: 0.2128
Epoch 5/100
4/4 [==============================] - 0s 16ms/step - loss: 0.2233 - val_loss: 0.1782
Epoch 6/100
4/4 [==============================] - 0s 23ms/step - loss: 0.1890 - val_loss: 0.1481
Epoch 7/100
4/4 [==============================] - 0s 19ms/step - loss: 0.1601 - val_loss: 0.1222
```

```
Epoch 100/100
4/4 [==============================] - 0s 11ms/step - loss: 0.0066 - val_loss: 0.0085
1/1 [==============================] - 0s 366ms/step - loss: 0.0067
Test loss: 0.006740686483681202
```





WEEK-3
Module name : Understanding and Using CNN
Exercise: Design a CNN for Image Recognition which includes hyperparameter tuning.

Program:

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import cifar10
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import LearningRateScheduler

# Load and preprocess the CIFAR-10 dataset
(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()
train_images, test_images = train_images / 255.0, test_images / 255.0

# Split data into training and validation sets
train_images, val_images, train_labels, val_labels =
train_test_split(train_images, train_labels, test_size=0.1,
random_state=42)

# Define a CNN architecture
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32,
32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

# Define a learning rate schedule
def lr_schedule(epoch):
    initial_lr = 0.001
    if epoch >= 40:
        return initial_lr * 0.1
    return initial_lr

# Compile the model
```

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Create data augmentation generator
datagen = ImageDataGenerator(rotation_range=15,
                             width_shift_range=0.1,
                             height_shift_range=0.1,
                             horizontal_flip=True,
                             fill_mode='nearest')

# Train the model with data augmentation and learning rate schedule
history = model.fit(datagen.flow(train_images, train_labels,
batch_size=64),
                    epochs=50,
                    steps_per_epoch=len(train_images) // 64,
                    validation_data=(val_images, val_labels),
                    callbacks=[LearningRateScheduler(lr_schedule)])

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)
```

OUTPUT:

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-
python.tar.gz
170498071/170498071 [==============================] - 3s
0us/step
Epoch 1/50
703/703 [==============================] - 105s 148ms/step - loss:
1.6865 - accuracy: 0.1086 - val_loss: 1.3433 - val_accuracy: 0.0788 - lr:
0.0010
Epoch 2/50
703/703 [==============================] - 100s 142ms/step - loss:
1.3505 - accuracy: 0.0979 - val_loss: 1.2421 - val_accuracy: 0.0778 - lr:
0.0010
.
.
.
Epoch 50/50
Test accuracy: 0.10419999808073044
```

Week – 9
Module name: Auto Encoders Advanced
Exercise: Demonstration of Application of Auto Encoders

Program:

Step-1: Install the tensor flow library

pip install tensorflow

Step-2: import the required libraries.
import numpy as np import

matplotlib.pyplot as plt from

tensorflow.keras.datasets import mnist

from tensorflow.keras.models import

Model

from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D

Step-3: Load and preprocess the MNIST dataset
(x_train, _), (x_test, _) = mnist.load_data()

Step-4: Normalize the pixel values between 0

and 1 x_train = x_train.astype('float32') / 255.0

x_test = x_test.astype('float32') / 255.0

Step-5: Add random noise to the images

noise_factor = 0.5

x_train_noisy = x_train + noise_factor *

np.random.normal(loc=0.0,scale=1.0, size=x_train.shape)

x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_test.shape)

Step-6: Clip the pixel values between 0 and 1

```
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
 x_test_noisy = np.clip(x_test_noisy, 0., 1.)
```

Step-7: Define the autoencoder architecture

```
 input_img = Input(shape=(28, 28, 1))
 x = Conv2D(32, (3, 3), activation='relu',
padding='same')(input_img)
x = MaxPooling2D((2, 2), padding='same')(x)
 x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
 encoded = MaxPooling2D((2, 2), padding='same')(x)
 x = Conv2D(64, (3, 3), activation='relu',
padding='same')(encoded)
 x = UpSampling2D((2, 2))(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(1, (3, 3), activation='sigmoid',
padding='same')(x) autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adam',
loss='binary_crossentropy')
```

Step-8: Train the autoencoder

```
autoencoder.fit(x_train_noisy.reshape(-1, 28, 28, 1),
x_train.reshape(-1, 28, 28, 1),
 epochs=10,
 batch_size=128,
 shuffle=True,
 validation_data=(x_test_noisy.reshape(-1, 28, 28, 1), x_test.reshape(1,
28, 28, 1)))
```

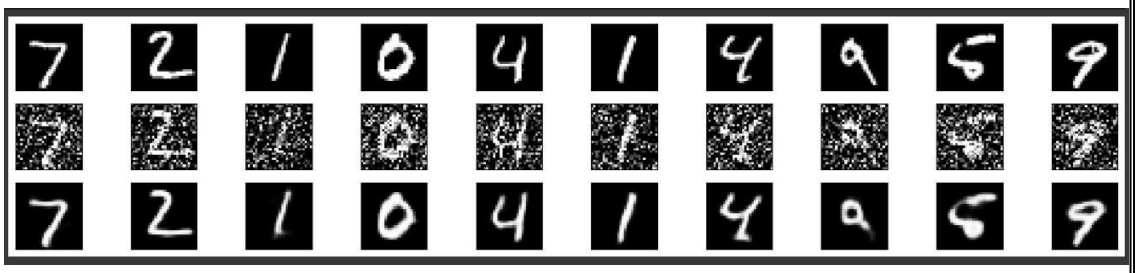
Step-9: Denoise test images using the trained autoencoder
denoised_images = autoencoder.predict(x_test_noisy.reshape(-1, 28, 28, 1))

Step-10: Display original, noisy, and denoised images

```
n = 10 # Number of images to display
plt.figure(figsize=(20, 4))
for i in range(n):
    # Original Images
    ax = plt.subplot(3, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    # Noisy Input Images
    ax = plt.subplot(3, n, i + 1 + n)
    plt.imshow(x_test_noisy[i].reshape(28,28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    # Denoised Images
    ax = plt.subplot(3, n, i + 1 + 2 * n)
    plt.imshow(denoised_images[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
```

plt.show()

Output:



Week - 5
Module Name: Removing noise from the images

Exercise: Implement Multi-Layer Perceptron algorithm for Image denoising hyperparameter tuning

Program:

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch.utils.data as data
import torchvision.transforms as transforms
import torchvision.datasets as datasets
from sklearn import metrics
from sklearn import decomposition
from sklearn import manifold
from tqdm.notebook import trange, tqdm
import matplotlib.pyplot as plt
import numpy as np
import copy
import random
import time
SEED = 1234
random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed(SEED)
torch.backends.cudnn.deterministic = True
ROOT = '.data'
train_data = datasets.MNIST(root=ROOT,
 train=True,
 download=True)
mean = train_data.data.float().mean() / 255
std = train_data.data.float().std() / 255
print(f'Calculated mean: {mean}')
print(f'Calculated std: {std}')
Calculated mean: 0.13066048920154572
Calculated std: 0.30810779333114624
train_transforms = transforms.Compose([
transforms.RandomRotation(5, fill=(0,)),
transforms.RandomCrop(28, padding=2),
transforms.ToTensor(),
transforms.Normalize(mean=[mean], std=[std])
```

```
 ])
test_transforms =  transforms.Compose([
 transforms.ToTensor(),
 transforms. Normalize(mean=[mean], std=[std])
 ])
train_data = datasets.MNIST(root=ROOT,
 train=True,
 download=True,
 transform=train_transforms)
test_data = datasets.MNIST(root=ROOT,
 train=False,
 download=True,
 transform=test_transforms)
print(f'Number of training examples: {len(train_data)}')
print(f'Number of testing examples: {len(test_data)}')
Number of training examples: 60000
Number of testing examples: 10000
def plot_images(images):
 n_images = len(images)
 rows = int(np.sqrt(n_images))
 cols = int(np.sqrt(n_images))
 fig = plt.figure()
 for i in range(rows*cols):
 ax = fig.add_subplot(rows, cols, i+1)
 ax.imshow(images[i].view(28, 28).cpu().numpy(), cmap='bone')
 ax.axis('off')
N_IMAGES = 25
images = [image for image, label in [train_data[i] for i in range(N_IMAGES)]]
plot_images(images)
VALID_RATIO = 0.9
n_train_examples = int(len(train_data) * VALID_RATIO)
n_valid_examples = len(train_data) - n_train_examples
train_data, valid_data = data.random_split(train_data,
[n_train_examples, n_valid_examples])
We can print out the number of examples again to check our splits are
correct.
print(f'Number of training examples: {len(train_data)}')
print(f'Number of validation examples: {len(valid_data)}')
print(f'Number of testing examples: {len(test_data)}')
Number of training examples: 54000
Number of validation examples: 6000
Number of testing examples: 10000
N_IMAGES = 25
images = [image for image, label in [valid_data[i] for i in range(N_IMAGES)]]
plot_images(images)
```

```
valid_data = copy.deepcopy(valid_data)
valid_data.dataset.transform = test_transforms
N_IMAGES = 25
images = [image for image, label in [valid_data[i] for i in range(N_IMAGES)]]
plot_images(images)
BATCH_SIZE = 64
train_iterator = data.DataLoader(train_data,
 shuffle=True,
 batch_size=BATCH_SIZE)
valid_iterator = data.DataLoader(valid_data,
 batch_size=BATCH_SIZE)
test_iterator = data.DataLoader(test_data,
 batch_size=BATCH_SIZE)
class MLP(nn.Module):
 def __init__(self, input_dim, output_dim):
 super().__init__()
 self.input_fc = nn.Linear(input_dim, 250)
 self.hidden_fc = nn.Linear(250, 100)
 self.output_fc = nn.Linear(100, output_dim)
 def forward(self, x):
 # x = [batch size, height, width]
 batch_size = x.shape[0]
 x = x.view(batch_size, -1)
 # x = [batch size, height * width]
 h_1 = F.relu(self.input_fc(x))
 # h_1 = [batch size, 250]
 h_2 = F.relu(self.hidden_fc(h_1))
 # h_2 = [batch size, 100]
 y_pred = self.output_fc(h_2)
 # y_pred = [batch size, output dim]
 return y_pred, h_2
INPUT_DIM = 28 * 28
OUTPUT_DIM = 10
model = MLP(INPUT_DIM, OUTPUT_DIM)
def count_parameters(model):
 return sum(p.numel() for p in model.parameters() if p.requires_grad)
print(f'The model has {count_parameters(model):,} trainable parameters')
The model has 222,360 trainable parameters
criterion = nn.CrossEntropyLoss()device = torch.device('cuda' if
torch.cuda.is_available() else 'cpu')
def calculate_accuracy(y_pred, y):
 top_pred = y_pred.argmax(1, keepdim=True)
 correct = top_pred.eq(y.view_as(top_pred)).sum()
 acc = correct.float() / y.shape[0]
 return acc
```

```
def train(model, iterator, optimizer, criterion, device):
 epoch_loss = 0
 epoch_acc = 0
 model.train()
 for (x, y) in tqdm(iterator, desc="Training", leave=False):
 x = x.to(device)
 y = y.to(device)
 optimizer.zero_grad()
 y_pred, _ = model(x)
 loss = criterion(y_pred, y)
 acc = calculate_accuracy(y_pred, y)
 loss.backward()
 optimizer.step()
 epoch_loss += loss.item()
 epoch_acc += acc.item()
 return epoch_loss / len(iterator), epoch_acc / len(iterator)
def evaluate(model, iterator, criterion, device):
 epoch_loss = 0
 epoch_acc = 0
 model.eval()
 with torch.no_grad():
 for (x, y) in tqdm(iterator, desc="Evaluating", leave=False):
 x = x.to(device)
 y = y.to(device)
 y_pred, _ = model(x)
 loss = criterion(y_pred, y)
 acc = calculate_accuracy(y_pred, y)
 epoch_loss += loss.item()
 epoch_acc += acc.item()
 return epoch_loss / len(iterator), epoch_acc / len(iterator)
```

The final step before training is to define a small function to tell us how long an epoch took.

```
def epoch_time(start_time, end_time):
 elapsed_time = end_time - start_time
 elapsed_mins = int(elapsed_time / 60)
 elapsed_secs = int(elapsed_time - (elapsed_mins * 60))
 return elapsed_mins, elapsed_secs
EPOCHS = 10
best_valid_loss = float('inf')
for epoch in trange(EPOCHS):
 start_time = time.monotonic()
 train_loss, train_acc = train(model, train_iterator, optimizer, criterion,
device)
 valid_loss, valid_acc = evaluate(model, valid_iterator, criterion, device)
 if valid_loss < best_valid_loss:
```

```
 best_valid_loss = valid_loss
 torch.save(model.state_dict(), 'tut1-model.pt')
 end_time = time.monotonic()
 epoch_mins, epoch_secs = epoch_time(start_time, end_time)
print(f'Epoch: {epoch+1:02} | Epoch Time: {epoch_mins}m {epoch_secs}s')
print(f'\tTrain Loss: {train_loss:.3f} | Train Acc: {train_acc*100:.2f}%')
print(f'\t Val. Loss: {valid_loss:.3f} | Val. Acc: {valid_acc*100:.2f}%')
model.load_state_dict(torch.load('tut1-model.pt'))
test_loss, test_acc = evaluate(model, test_iterator, criterion, device)
print(f'Test Loss: {test_loss:.3f} | Test Acc: {test_acc*100:.2f}%')
Test Loss: 0.052 | Test Acc: 98.37%
def get_predictions(model, iterator, device):
 model.eval()
 images = []
 labels = []
 probs = []
 with torch.no_grad():
 for (x, y) in iterator:
 x = x.to(device)
 y_pred, _ = model(x)
 y_prob = F.softmax(y_pred, dim=-1)
 images.append(x.cpu())
 labels.append(y.cpu())
 probs.append(y_prob.cpu())
 images = torch.cat(images, dim=0)
 labels = torch.cat(labels, dim=0)
 probs = torch.cat(probs, dim=0)
 return images, labels, probs
images, labels, probs = get_predictions(model, test_iterator, device)
pred_labels = torch.argmax(probs, 1)
def plot_confusion_matrix(labels, pred_labels):
 fig = plt.figure(figsize=(10, 10))
 ax = fig.add_subplot(1, 1, 1)
 cm = metrics.confusion_matrix(labels, pred_labels)
 cm = metrics.ConfusionMatrixDisplay(cm, display_labels=range(10))
cm.plot(values_format='d', cmap='Blues', ax=ax)
plot_confusion_matrix(labels, pred_labels)
corrects = torch.eq(labels, pred_labels)
incorrect_examples = []
for image, label, prob, correct in zip(images, labels, probs, corrects):
 if not correct:
 incorrect_examples.append((image, label, prob))
incorrect_examples.sort(reverse=True,
key=lambda x: torch.max(x[2], dim=0).values)
```

```
def plot_most_incorrect(incorrect, n_images):
 rows = int(np.sqrt(n_images))
 cols = int(np.sqrt(n_images))
 fig = plt.figure(figsize=(20, 10))
 for i in range(rows*cols):
 ax = fig.add_subplot(rows, cols, i+1)
 image, true_label, probs = incorrect[i]
 true_prob = probs[true_label]
 incorrect_prob, incorrect_label = torch.max(probs, dim=0)
 ax.imshow(image.view(28, 28).cpu().numpy(), cmap='bone')
 ax.set_title(f'true label: {true_label} ({true_prob:.3f})\n'
 f'pred label: {incorrect_label} ({incorrect_prob:.3f})')
 ax.axis('off')
 fig.subplots_adjust(hspace=0.5)
N_IMAGES = 25
plot_most_incorrect(incorrect_examples, N_IMAGES)
def get_representations(model, iterator, device):
model.eval()
 outputs = []
 intermediates = []
 labels = []
 with torch.no_grad():
 for (x, y) in tqdm(iterator):
 x = x.to(device)
 y_pred, h = model(x)
 outputs.append(y_pred.cpu())
 intermediates.append(h.cpu())
 labels.append(y)
 outputs = torch.cat(outputs, dim=0)
 intermediates = torch.cat(intermediates, dim=0)
 labels = torch.cat(labels, dim=0)
 return outputs, intermediates, labels
outputs, intermediates, labels = get_representations(model,
 train_iterator,
 device)
def get_pca(data, n_components=2):
 pca = decomposition.PCA()
 pca.n_components = n_components
 pca_data = pca.fit_transform(data)
 return pca_data
def plot_representations(data, labels, n_images=None):
 if n_images is not None:
 data = data[:n_images]
 labels = labels[:n_images]
 fig = plt.figure(figsize=(10, 10))
```

```
 ax = fig.add_subplot(111)
 scatter = ax.scatter(data[:, 0], data[:, 1], c=labels, cmap='tab10')
handles, labels = scatter.legend_elements()
ax.legend(handles=handles, labels=labels)
intermediate_pca_data = get_pca(intermediates)
plot_representations(intermediate_pca_data, labels)
def get_tsne(data, n_components=2, n_images=None):
 if n_images is not None:
 data = data[:n_images]
 tsne = manifold.TSNE(n_components=n_components, random_state=0)
 tsne_data = tsne.fit_transform(data)
 return tsne_data
N_IMAGES = 5_000
output_tsne_data = get_tsne(outputs, n_images=N_IMAGES)
plot_representations(output_tsne_data, labels, n_images=N_IMAGES)
intermediate_tsne_data = get_tsne(intermediates, n_images=N_IMAGES)
plot_representations(intermediate_tsne_data, labels, n_images=N_IMAGES
def imagine_digit(model, digit, device, n_iterations=50_000):
 model.eval()
 best_prob = 0
 best_image = None
 with torch.no_grad():
 for _ in trange(n_iterations):
 x = torch.randn(32, 28, 28).to(device)
 y_pred, _ = model(x)
 preds = F.softmax(y_pred, dim=-1)
 _best_prob, index = torch.max(preds[:, digit], dim=0)
 if _best_prob > best_prob:
 best_prob = _best_prob
 best_image = x[index]
 return best_image, best_prob
DIGIT = 3
best_image, best_prob = imagine_digit(model, DIGIT, device)
print(f'Best image probability: {best_prob.item()*100:.2f}%')
Best image probability: 99.98%
plt.imshow(best_image.cpu().numpy(), cmap='bone')
plt.axis('off')
def plot_weights(weights, n_weights):
 rows = int(np.sqrt(n_weights))
 cols = int(np.sqrt(n_weights))
 fig = plt.figure(figsize=(20, 10))
 for i in range(rows*cols):
 ax = fig.add_subplot(rows, cols, i+1)
 ax.imshow(weights[i].view(28, 28).cpu().numpy(), cmap='bone')
 ax.axis('off')
```

```
N_WEIGHTS = 25
weights = model.input_fc.weight.data
plot_weights(weights, N_WEIGHTS)
```

Output: