## Assignment 1:

```
pip install tensorflow
import tensorflow as tf
print(tf.__version__)
print(tf.reduce_sum(tf.random.normal([1000,1000])))
pip install keras
from tensorflow import keras
from keras import datasets
(train_images,train_labels),(test_images,test_labels)=datasets.mnist.load_data()
train_images.shape,test_images.shape
pip install theano
import numpy
import theano.tensor as T
from theano import function
x = T.dscalar('x')
y = T.dscalar('y')
z = x + y
f = function([x,y],z)
f(5,7)
pip install torch
import torch
import torch.nn as nn
print(torch.__version__)
```

## Assignment 2

```
import tensorflow as tf
from tensorflow import keras
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random
```

```
%matplotlib inline


#import dataset and split into train and test
mnist = tf.keras.datasets.mnist
(x_train,y_train),(x_test,y_test) = mnist.load_data()


#to see how first image looks
plt.matshow(x_train[0])

x_train = x_train / 255
x_test = x_test / 255

model = keras.Sequential([
   keras.layers.Flatten(input_shape=(28,28)),
   keras.layers.Dense(128,activation = 'relu'),
   keras.layers.Dense(10,activation = 'softmax')
])

model.summary()

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

history = model.fit(x_train,y_train,validation_data = (x_test,y_test), epochs = 10)

test_loss,test_acc = model.evaluate(x_test,y_test)
print("Loss = %.3f" %test_loss)
print("Accuracy = %.3f" %test_acc)

n = random.randint(0,9999)
```

```
plt.imshow(x_test[n])
plt.show()
```

```
test_predict = model.predict(x_test)
#get classification labels
test_predict_labels = np.argmax(test_predict,axis = 1)
confusion_matrix = tf.math.confusion_matrix(labels = y_test, predictions = test_predict_labels)
print('Confusion Matrix of the Test Set :\n' , confusion_matrix)
```

## assignment3

```
import numpy as np
import pandas as pd
import random
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten,Conv2D,Dense,MaxPooling2D
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist
```

```
(X_train,y_train),(X_test,y_test) = mnist.load_data()
```

```
print(X_train.shape)
```

```
X_train[0].min(), X_train[0].max()

X_train = (X_train - 0.0) / (255.0 - 0.0)
X_test = (X_test - 0.0) / (255.0 - 0.0)
X_train[0].min(), X_train[0].max()


def plot_digit(image, digit, plt, i):
 plt.subplot(4, 5, i + 1)
 plt.imshow(image, cmap=plt.get_cmap('gray'))
 plt.title(f"Digit: {digit}")
 plt.xticks([])
 plt.yticks([])
plt.figure(figsize=(16, 10))
for i in range(20):
 plot_digit(X_train[i], y_train[i], plt, i)
plt.show()

X_train = X_train.reshape((X_train.shape + (1,)))
X_test = X_test.reshape((X_test.shape + (1,)))
y_train[0:20]

model = Sequential([
 Conv2D(32, (3, 3), activation="relu", input_shape=(28, 28, 1)),
 MaxPooling2D((2, 2)),
 Flatten(),
 Dense(100, activation="relu"),
 Dense(10, activation="softmax")
])
```

```
optimizer = SGD(learning_rate=0.01, momentum=0.9)
model.compile(
 optimizer=optimizer,
 loss="sparse_categorical_crossentropy",
 metrics=["accuracy"]
)
model.summary()



history = model.fit(X_train, y_train,validation_data = (x_test,y_test), epochs=10)

plt.figure(figsize=(16, 10))
for i in range(20):
 image = random.choice(X_test).squeeze()
 digit = np.argmax(model.predict(image.reshape((1, 28, 28, 1)))[0], axis=-1)
 plot_digit(image, digit, plt, i)
plt.show()

predictions = np.argmax(model.predict(X_test), axis=-1)
accuracy_score(y_test, predictions)

score = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score[0])    #Test Loss
print('Test accuracy:', score[1])    #Test Accuracy

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = 10
```

```
epochs_range = range(epochs)


import os
# plotting the metrics
fig = plt.figure()
plt.figure(figsize=(8, 8))

plt.subplot(2,1,1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='lower right')


plt.subplot(2,1,2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')
plt.tight_layout()



#Save the model
# serialize model to JSON
model_digit_json = model.to_json()
```

```
with open("model_digit.json", "w") as json_file:
 json_file.write(model_digit_json)
# serialize weights to HDF5
model.save_weights("model_digit.h5")
print("Saved model to disk")
```

## Assignmenet 4

```
import pandas as pd
import numpy as np
import pickle
import matplotlib.pyplot as plt
from scipy import stats
import tensorflow as tf
import seaborn as sns
from pylab import rcParams
from sklearn.model_selection import train_test_split
from keras.models import Model, load_model
from keras.layers import Input, Dense
from keras.callbacks import ModelCheckpoint, TensorBoard
from keras import regularizers
%matplotlib inline
sns.set(style='whitegrid', palette='muted', font_scale=1.5)
rcParams['figure.figsize'] = 14, 8
RANDOM_SEED = 42
LABELS = ["Normal", "Fraud"]

df =  pd.read_csv("D:\creditcard.csv")

count_classes = pd.value_counts(df['Class'], sort = True)
```

```
count_classes.plot(kind = 'bar', rot=0)
plt.title("Transaction class distribution")
plt.xticks(range(2), LABELS)
plt.xlabel("Class")
plt.ylabel("Frequency");


from sklearn.preprocessing import StandardScaler

data = df.drop(['Time'], axis=1)

data['Amount'] = StandardScaler().fit_transform(data['Amount'].values.reshape(-1, 1))

X_train, X_test = train_test_split(data, test_size=0.2, random_state=RANDOM_SEED)
X_train = X_train[X_train.Class == 0]
X_train = X_train.drop(['Class'], axis=1)

y_test = X_test['Class']
X_test = X_test.drop(['Class'], axis=1)

X_train = X_train.values
X_test = X_test.values

input_dim = X_train.shape[1]
encoding_dim = 14

input_layer = Input(shape=(input_dim, ))

encoder = Dense(encoding_dim, activation="tanh",
        activity_regularizer=regularizers.l1(10e-5))(input_layer)
encoder = Dense(int(encoding_dim / 2), activation="relu")(encoder)
```

```
decoder = Dense(int(encoding_dim / 2), activation='tanh')(encoder)

decoder = Dense(input_dim, activation='relu')(decoder)


autoencoder = Model(inputs=input_layer, outputs=decoder)



nb_epoch = 100

batch_size = 32

early_stop = tf.keras.callbacks. EarlyStopping( monitor= 'val_loss', min_delta=0.0001, patience=10,
verbose=1, mode='min',

                    restore_best_weights=True)


autoencoder.compile(optimizer='adam',

        loss='mean_squared_error',

        metrics=['accuracy'])


checkpointer = ModelCheckpoint(filepath="model.h5",

            verbose=0,

            save_best_only=True)

tensorboard = TensorBoard(log_dir='./logs',

          histogram_freq=0,

          write_graph=True,

          write_images=True)


history = autoencoder.fit(X_train, X_train,

        epochs=nb_epoch,

        batch_size=batch_size,

        shuffle=True,

        validation_data=(X_test, X_test),

        verbose=1,
```

```
                    callbacks=[checkpointer, early_stop]).history




plt.plot(history['loss'])
plt.plot(history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right');

predictions = autoencoder.predict(X_test)


mse = np.mean(np.power(X_test - predictions, 2), axis=1)
error_df = pd.DataFrame({'reconstruction_error': mse,
                        'true_class': y_test})


error_df.describe()


threshold = 50
groups = error_df.groupby('true_class')
fig, ax = plt.subplots()

for name, group in groups:
    ax.plot(group.index, group.reconstruction_error, marker='o', ms=3.5, linestyle='',
            label= "Fraud" if name == 1 else "Normal")
ax.hlines(threshold, ax.get_xlim()[0], ax.get_xlim()[1], colors="r", zorder=100, label='Threshold')
```

```
ax.legend()

plt.title("Reconstruction error for different classes")

plt.ylabel("Reconstruction error")

plt.xlabel("Data point index")

plt.show();




from sklearn.metrics import confusion_matrix,recall_score,accuracy_score,precision_score


y_pred = [1 if e > threshold else 0 for e in error_df.reconstruction_error.values]

conf_matrix = confusion_matrix(error_df.true_class, y_pred)


plt.figure(figsize=(12, 12))

sns.heatmap(conf_matrix, xticklabels=LABELS, yticklabels=LABELS, annot=True, fmt="d");

plt.title("Confusion matrix")

plt.ylabel('True class')

plt.xlabel('Predicted class')

plt.show()


error_df['pred'] = y_pred


# print Accuracy, precision and recall print(" Accuracy:


print("Accuracy:",accuracy_score (error_df['true_class'],error_df['pred']))

print(" Recall:",recall_score(error_df['true_class'],error_df['pred']))

print(" Precision:",precision_score(error_df['true_class'],error_df['pred']))
```

# Assignment 5

```
import numpy as np

import keras.backend as K

from keras.models import Sequential

from keras.layers import Dense, Embedding, Lambda

from keras.utils import np_utils

from keras.preprocessing import sequence

from keras.preprocessing.text import Tokenizer

import gensim

data=open('D:\corona.txt','r')

corona_data = [text for text in data if text.count(' ') >= 2]

vectorize = Tokenizer()

vectorize.fit_on_texts(corona_data)

corona_data = vectorize.texts_to_sequences(corona_data)

total_vocab = sum(len(s) for s in corona_data)

word_count = len(vectorize.word_index) + 1

window_size = 2

print(total_vocab)




def cbow_model(data, window_size, total_vocab):

   total_length = window_size*2

   for text in data:

      text_len = len(text)
```

```
    for idx, word in enumerate(text):
        context_word = []
        target   = []
        begin = idx - window_size
        end = idx + window_size + 1
        context_word.append([text[i] for i in range(begin, end) if 0 <= i < text_len and i != idx])
        target.append(word)
        contextual = sequence.pad_sequences(context_word, total_length=total_length)
        final_target = np_utils.to_categorical(target, total_vocab)
        yield(contextual, final_target)


model = Sequential()
model.add(Embedding(input_dim=total_vocab, output_dim=100, input_length=window_size*2))
model.add(Lambda(lambda x: K.mean(x, axis=1), output_shape=(100,)))
model.add(Dense(total_vocab, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')
for i in range(10):
    cost = 0
    for x, y in cbow_model(data, window_size, total_vocab):
        cost += model.train_on_batch(contextual, final_target)
    print(i, cost)


dimensions=100
vect_file = open('D:/vectors.txt' ,'w')
vect_file.write('{} {}\n'.format(total_vocab,dimensions))


weights = model.get_weights()[0]
for text, i in vectorize.word_index.items():
```

```
    final_vec = ' '.join(map(str, list(weights[i, :])))
    vect_file.write('{} {}\n'.format(text, final_vec))
vect_file.close()



cbow_output = gensim.models.KeyedVectors.load_word2vec_format('D:/vectors.txt', binary=True)
cbow_output.most_similar(positive=['the'])

cbow_output['the'].shape
```

# Assignment 6

```
# example of using a pre-trained model as a classifier
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input
from keras.applications.vgg16 import decode_predictions
from keras.applications.vgg16 import VGG16
# load an image from file
image = load_img('download.jpg', target_size=(224, 224))
# convert the image pixels to a numpy array
image = img_to_array(image)
# reshape data for the model
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
# prepare the image for the VGG model
image = preprocess_input(image)
# load the model
model = VGG16()
# predict the probability across all output classes
yhat = model.predict(image)
```

```
# convert the probabilities to class labels
label = decode_predictions(yhat)
# retrieve the most likely result, e.g. highest probability
label = label[0][0]
# print the classification
print('%s (%.2f%%)' % (label[1], label[2]*100))


# load an image from file
image = load_img('download2.png', target_size=(224, 224))
# convert the image pixels to a numpy array
image = img_to_array(image)
# reshape data for the model
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
# prepare the image for the VGG model
image = preprocess_input(image)
# load the model
model = VGG16()
# predict the probability across all output classes
yhat = model.predict(image)
# convert the probabilities to class labels
label = decode_predictions(yhat)
# retrieve the most likely result, e.g. highest probability
label = label[0][0]
# print the classification
print('%s (%.2f%%)' % (label[1], label[2]*100))


# load an image from file
image = load_img('download3.jpg', target_size=(224, 224))
```

```
# convert the image pixels to a numpy array
image = img_to_array(image)
# reshape data for the model
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
# prepare the image for the VGG model
image = preprocess_input(image)
# load the model
model = VGG16()
# predict the probability across all output classes
yhat = model.predict(image)
# convert the probabilities to class labels
label = decode_predictions(yhat)
# retrieve the most likely result, e.g. highest probability
label = label[0][0]
# print the classification
print('%s (%.2f%%)' % (label[1], label[2]*100))
```