Implementing Feed-forward neural networks with Keras and TensorFlow

a. Import the necessary packages

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
```

b. Load the training and testing data (MNIST/CIFAR10)

```
# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

#normalize the input data to range [0, 1]
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

#flatten the input data (28 X 28 images to 784 pixels)
x_train = x_train.reshape(-1, 784)
x_test = x_test.reshape(-1, 784)
```

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

c. Define the network architecture using Keras

```
model = Sequential()
model.add(Dense(128, activation='relu', input_shape=(784,)))
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

d. Train the model using SGD

```
# from keras.src.engine.training import optimizer
# compile the model
model.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(x_train, y_train, epochs=10, batch_size=64, validation_split=0.2)
```

```
Epoch 1/10
750/750 [==============================] - 7s 3ms/step - loss: 1.0122 - accuracy: 0.7379 - val_loss: 0.4530 - val_accuracy: 0.8848
Epoch 2/10
750/750 [==============================] - 3s 4ms/step - loss: 0.4081 - accuracy: 0.8876 - val_loss: 0.3333 - val_accuracy: 0.9087
Epoch 3/10
750/750 [==============================] - 2s 3ms/step - loss: 0.3307 - accuracy: 0.9064 - val_loss: 0.2933 - val_accuracy: 0.9152
Epoch 4/10
750/750 [==============================] - 3s 3ms/step - loss: 0.2942 - accuracy: 0.9165 - val_loss: 0.2718 - val_accuracy: 0.9220
Epoch 5/10
750/750 [==============================] - 2s 3ms/step - loss: 0.2690 - accuracy: 0.9232 - val_loss: 0.2534 - val_accuracy: 0.9277
Epoch 6/10
750/750 [==============================] - 2s 3ms/step - loss: 0.2501 - accuracy: 0.9285 - val_loss: 0.2336 - val_accuracy: 0.9338
Epoch 7/10
750/750 [==============================] - 3s 4ms/step - loss: 0.2339 - accuracy: 0.9332 - val_loss: 0.2220 - val_accuracy: 0.9369
Epoch 8/10
750/750 [==============================] - 3s 4ms/step - loss: 0.2198 - accuracy: 0.9367 - val_loss: 0.2105 - val_accuracy: 0.9407
Epoch 9/10
750/750 [==============================] - 3s 3ms/step - loss: 0.2073 - accuracy: 0.9408 - val_loss: 0.2015 - val_accuracy: 0.9447
Epoch 10/10
750/750 [==============================] - 2s 3ms/step - loss: 0.1963 - accuracy: 0.9441 - val_loss: 0.1923 - val_accuracy: 0.9463
```

e. Evaluate the network

```
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f'Test Loss: {test_loss:.4f}')
print(f'Test Accuracy: {test_accuracy:.4f}')
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.1941 - accuracy: 0.9451
Test Loss: 0.1941
Test Accuracy: 0.9451
```

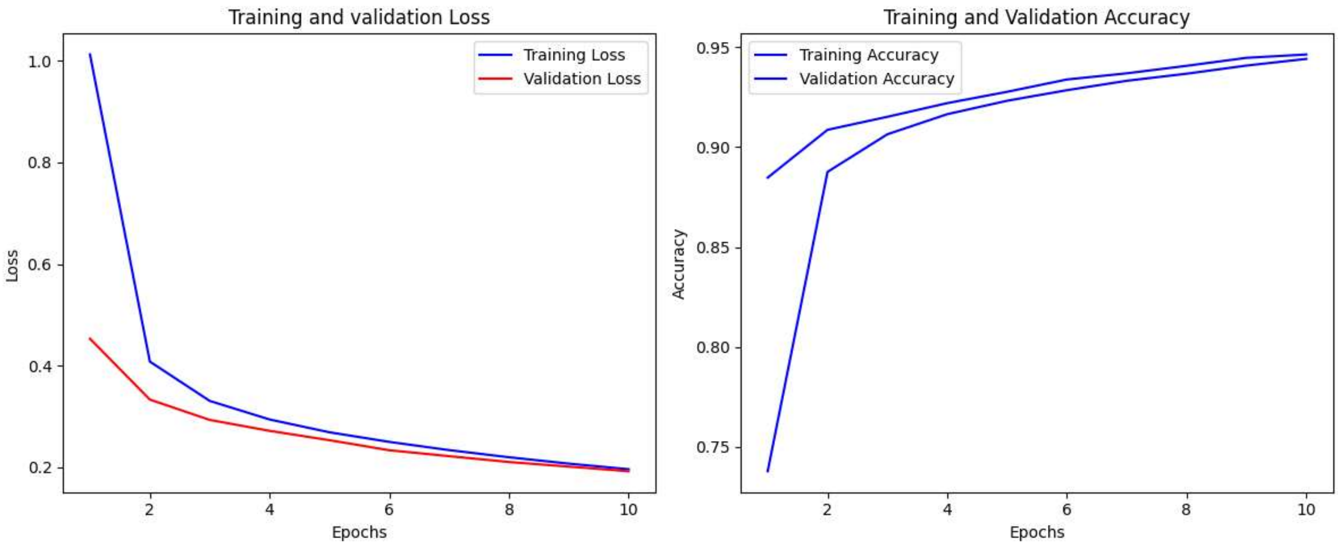
f. Plot the training loss and accuracy

```
train_loss = history.history['loss']
val_loss = history.history['val_loss']
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
epochs = range(1, len(train_loss) + 1)


plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs, train_loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'r', label='Validation Loss')
plt.title('Training and validation Loss')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epochs, train_acc, 'b', label='Training Accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```



Build the Image classification model by dividing the model into the following fourstages:

a. Loading and preprocessing the image data

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical

# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Preprocess the image data
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

#one hot encoding
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 3s 0us/step
```

b. Defining the model's architecture

```
# Define the model architecture
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

c. Training the model

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

# Train the model
history = model.fit(x_train, y_train, epochs=10, batch_size=64, validation_data=(x_test, y_test))
```

```
Epoch 1/10
782/782 [==============================] - 12s 6ms/step - loss: 1.5589 - accuracy: 0.4357 - val_loss: 1.2763 - val_accuracy: 0.5453
Epoch 2/10
782/782 [==============================] - 4s 5ms/step - loss: 1.1791 - accuracy: 0.5815 - val_loss: 1.0912 - val_accuracy: 0.6115
Epoch 3/10
782/782 [==============================] - 5s 7ms/step - loss: 1.0284 - accuracy: 0.6385 - val_loss: 1.0530 - val_accuracy: 0.6310
Epoch 4/10
782/782 [==============================] - 4s 5ms/step - loss: 0.9383 - accuracy: 0.6691 - val_loss: 0.9637 - val_accuracy: 0.6710
Epoch 5/10
782/782 [==============================] - 5s 7ms/step - loss: 0.8625 - accuracy: 0.6981 - val_loss: 0.9030 - val_accuracy: 0.6923
Epoch 6/10
782/782 [==============================] - 4s 6ms/step - loss: 0.8039 - accuracy: 0.7182 - val_loss: 0.9372 - val_accuracy: 0.6791
Epoch 7/10
782/782 [==============================] - 4s 6ms/step - loss: 0.7536 - accuracy: 0.7349 - val_loss: 0.9106 - val_accuracy: 0.6888
Epoch 8/10
782/782 [==============================] - 5s 6ms/step - loss: 0.7048 - accuracy: 0.7536 - val_loss: 0.8679 - val_accuracy: 0.7083
Epoch 9/10
782/782 [==============================] - 4s 6ms/step - loss: 0.6709 - accuracy: 0.7664 - val_loss: 0.8758 - val_accuracy: 0.7097
Epoch 10/10
782/782 [==============================] - 4s 6ms/step - loss: 0.6265 - accuracy: 0.7795 - val_loss: 0.8569 - val_accuracy: 0.7179
```

d. Estimating the model's performance

```
# Evaluate the model
test_loss, test_accuracy = model.evaluate(x_test, y_test)

print(f'Test loss: {test_loss:.4f}')
print(f'Test accuracy: {test_accuracy:.4f}')
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.8569 - accuracy: 0.7179
Test loss: 0.8569
Test accuracy: 0.7179
```

Use Autoencoder to implement anomaly detection. Build the model by using the following:

a. Import required libraries

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras.datasets import mnist
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

b. Upload/access the dataset

```
# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Preprocess the image data
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))


#split data into training and tesing
x_train, x_val = train_test_split(x_train, test_size=0.2, random_state=42)
```

c. The encoder converts it into a latent representation

```
input_layer = Input(shape=(x_train.shape[1],))
encoded = Dense(128, activation='relu')(input_layer)
encoded = Dense(64, activation='relu')(encoded)
encoded = Dense(32, activation='relu')(encoded)

encoder = Model(input_layer, encoded)
```

d. Decoder networks convert it back to the original input

```
decoded = Dense(64, activation='relu')(encoded)
decoded = Dense(128, activation='relu')(decoded)
decoded = Dense(x_train.shape[1], activation='sigmoid')(decoded)

autoencoder = Model(input_layer, decoded)
```

e. Compile the models with Optimizer, Loss, and Evaluation Metrics

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


#train the autoencoder on the training data
autoencoder.fit(x_train, x_train, epochs=10, batch_size=32, validation_data=(x_val, x_val))
```

```
Epoch 1/10
1500/1500 [==============================] - 8s 4ms/step - loss: 0.0359 - val_loss: 0.0215
Epoch 2/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0188 - val_loss: 0.0169
Epoch 3/10
1500/1500 [==============================] - 5s 4ms/step - loss: 0.0155 - val_loss: 0.0146
Epoch 4/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0138 - val_loss: 0.0134
Epoch 5/10
1500/1500 [==============================] - 5s 4ms/step - loss: 0.0127 - val_loss: 0.0125
```

```
Epoch 6/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0121 - val_loss: 0.0120
Epoch 7/10
1500/1500 [==============================] - 5s 4ms/step - loss: 0.0115 - val_loss: 0.0115
Epoch 8/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0111 - val_loss: 0.0111
Epoch 9/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0107 - val_loss: 0.0107
Epoch 10/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.0104 - val_loss: 0.0107
<keras.src.callbacks.History at 0x799e739f4b80>
```

```
#evauate the autoencoder on the training data
reconstruction_error = np.mean(np.square(x_test - autoencoder.predict(x_test)), axis=1)

#set the threshold anomaly detection
threshold = np.percentile(reconstruction_error, 95)

#detect the anomaly in the test data
anomalies = x_test[reconstruction_error > threshold]
```

```
313/313 [==============================] - 1s 2ms/step
```

```
#display the some detected anomalies
n_anomalies_to_display = 5
for i in range(n_anomalies_to_display):
    plt.subplot(1, n_anomalies_to_display, i + 1)
    plt.imshow(anomalies[i].reshape(28, 28), cmap='gray')
    plt.title(f'Anomaly {i + 1}')
    plt.axis('off')
plt.show()
```

Implement the Continuous Bag of Words (CBOW) Model. Stages can be:

a. Data preparation

```
corpus = [
    "I like to learn deep learning",
    "Deep learning is interesting",
    "I enjoy studying deep learning"
]
```

b. Generate training data

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import skipgrams

# Tokenize the corpus
tokenizer = Tokenizer()
tokenizer.fit_on_texts(corpus)
word2idx = tokenizer.word_index
idx2word = {v: k for k, v in word2idx.items()}

# Generate training data
vocab_size = len(word2idx) + 1
target_words, context_words = [], []
for sentence in corpus:
    tokenized = tokenizer.texts_to_sequences([sentence])[0]
    pairs, _ = skipgrams(tokenized, vocabulary_size=vocab_size, window_size=1, negative_samples=0)
    target, context = zip(*pairs)
    target_words.extend(target)
    context_words.extend(context)
```

c. Train model

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Dense, Reshape

embedding_dim = 100

model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=1))
model.add(Reshape((embedding_dim,)))
model.add(Dense(units=vocab_size, activation='softmax'))

model.compile(loss='sparse_categorical_crossentropy', optimizer='adam')
model.summary()

# Train the model
model.fit(x=target_words, y=context_words, epochs=50)
```

```
1/1 [==============================] - 0s 10ms/step - loss: 2.1081
Epoch 31/50
1/1 [==============================] - 0s 11ms/step - loss: 2.0972
Epoch 32/50
1/1 [==============================] - 0s 12ms/step - loss: 2.0861
Epoch 33/50
1/1 [==============================] - 0s 14ms/step - loss: 2.0750
Epoch 34/50
1/1 [==============================] - 0s 11ms/step - loss: 2.0638
Epoch 35/50
1/1 [==============================] - 0s 11ms/step - loss: 2.0524
Epoch 36/50
1/1 [==============================] - 0s 12ms/step - loss: 2.0410
Epoch 37/50
1/1 [==============================] - 0s 11ms/step - loss: 2.0294
Epoch 38/50
1/1 [==============================] - 0s 12ms/step - loss: 2.0178
Epoch 39/50
1/1 [==============================] - 0s 12ms/step - loss: 2.0060
Epoch 40/50
1/1 [==============================] - 0s 11ms/step - loss: 1.9941
Epoch 41/50
1/1 [==============================] - 0s 13ms/step - loss: 1.9821
Epoch 42/50
1/1 [==============================] - 0s 12ms/step - loss: 1.9700
Epoch 43/50
1/1 [==============================] - 0s 13ms/step - loss: 1.9578
Epoch 44/50
1/1 [==============================] - 0s 13ms/step - loss: 1.9455
Epoch 45/50
1/1 [==============================] - 0s 11ms/step - loss: 1.9331
Epoch 46/50
1/1 [==============================] - 0s 11ms/step - loss: 1.9206
Epoch 47/50
1/1 [==============================] - 0s 11ms/step - loss: 1.9080
Epoch 48/50
1/1 [==============================] - 0s 19ms/step - loss: 1.8953
Epoch 49/50
1/1 [==============================] - 0s 12ms/step - loss: 1.8825
Epoch 50/50
1/1 [==============================] - 0s 11ms/step - loss: 1.8696
<keras.src.callbacks.History at 0x799e7d7e3d30>
```

d. Output

```
word_to_lookup = "deep"
word_idx = word2idx[word_to_lookup]
word_embedding = model.layers[0].get_weights()[0][word_idx]

print(f"Embedding for '{word_to_lookup}': {word_embedding}")
```

```
Embedding for 'deep': [-0.08426171 -0.04487503  0.03471786  0.00877153  0.03311342 -0.05220547
 -0.10537343  0.05674294  0.064823   -0.0503858   0.08464649 -0.03675745
  0.09589159 -0.02757626  0.05152629  0.02986002  0.00835305  0.0891538
  0.0274698  -0.08283201  0.08814979 -0.02207836  0.07251184 -0.10077845
  0.01189684 -0.04852607 -0.03135568 -0.0921233   0.0509307   0.0990841
 -0.07088152 -0.05199756 -0.08785651  0.02239579  0.07870483 -0.08659703
  0.06364825 -0.0147792  -0.07328346 -0.00406267 -0.06556738 -0.05483082
  0.02189518 -0.06745288  0.06877382 -0.08121266 -0.00131394 -0.08789965
  0.01931682 -0.06161145  0.04886431  0.01397459  0.01659805  0.0495768
 -0.00831992 -0.06807461  0.0378589  -0.10111599  0.06906442  0.07710025
  0.01673487  0.05439958 -0.02330448  0.05384533 -0.05315861  0.09971622
 -0.02362644  0.02694047  0.00438797  0.00650261 -0.04028632  0.08963581
 -0.08902054 -0.00401343 -0.05719236  0.07904532  0.09287891  0.06366136
  0.07346602  0.00621467  0.04426254 -0.01764723 -0.00551786 -0.05514027
 -0.04602342  0.05532861 -0.07229256 -0.07894777  0.03305203  0.04546653
  0.07533833  0.05800479 -0.05717856  0.02284533 -0.09087887  0.07040332
 -0.03173155 -0.07509761 -0.0395535   0.05414165]
```

Object detection using Transfer Learning of CNN architectures

```
import torch
import torchvision
from torchvision import transforms as T
from PIL import Image
import cv2
from google.colab.patches import cv2_imshow
```

```
# Load the pre-trained model
model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated sinc
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None`
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/fasterrcnn_resnet50_fpn_coco-258fb6c6.pth" to /root/.cache/torch/hub/checkpoints/-
100%|██████████| 160M/160M [00:01<00:00, 160MB/s]
```

```
model.eval()
```

```
!wget 'http://images.cocodataset.org/val2017/000000037777.jpg'
```

```
--2023-10-19 13:34:44--  http://images.cocodataset.org/val2017/000000037777.jpg
Resolving images.cocodataset.org (images.cocodataset.org)... 52.216.49.97, 3.5.25.75, 3.5.7.189, ...
Connecting to images.cocodataset.org (images.cocodataset.org)|52.216.49.97|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 40833 (40K) [image/jpeg]
Saving to: '000000037777.jpg'

000000037777.jpg    100%[===================>]  39.88K  --.-KB/s    in 0.05s

2023-10-19 13:34:45 (842 KB/s) - '000000037777.jpg' saved [40833/40833]
```

```
ig = Image.open('/content/000000037777.jpg')
```

```
transform = T.ToTensor()
img = transform(ig)
```

```
with torch.no_grad():
    pred = model([img])
```

```
pred[0].keys()
```

```
dict_keys(['boxes', 'labels', 'scores'])
```

```
bboxex, labels, scores = pred[0]['boxes'], pred[0]['labels'], pred[0]['scores']
```

```
num = torch.argwhere(scores > 0.8).shape[0]
```

```
print(num)
```

```
11
```

```
coco_names = [
    "person" , "bicycle" , "car" , "motorcycle" , "airplane" , "bus" , "train" , "truck" , "boat" , "traffic light" ,
    "fire hydrant" , "street sign" , "stop sign" , "parking meter" , "bench" , "bird" , "cat" , "dog" , "horse" , "shee
    "cow" , "elephant" , "bear" , "zebra" , "giraffe" , "hat" , "backpack" , "umbrella" , "shoe" , "eye glasses" , "har
    "tie" , "suitcase" , "frisbee" , "skis" , "snowboard" , "sports ball" , "kite" , "baseball bat" ,
    "baseball glove" , "skateboard" , "surfboard" , "tennis racket" , "bottle" ,
    "plate" , "wine glass" , "cup" , "fork" , "knife" , "spoon" , "bowl" ,
    "banana" , "apple" , "sandwich" , "orange" , "broccoli" , "carrot" , "hot dog" ,
    "pizza" , "donut" , "cake" , "chair" , "couch" , "potted plant" , "bed" ,
    "mirror" , "dining table" , "window" , "desk" , "toilet" , "door" , "tv" ,
    "laptop" , "mouse" , "remote" , "keyboard" , "cell phone" , "microwave" ,
    "oven" , "toaster" , "sink" , "refrigerator" , "blender" , "book" ,
```

```
    "clock" , "vase" , "scissors" , "teddy bear" , "hair drier" , "toothbrush" , "hair brush"
]

font = cv2.FONT_HERSHEY_SIMPLEX


igg = cv2.imread('/content/000000037777.jpg')
for i in range(num):
    x1, y1, x2, y2 = bboxex[i].numpy().astype('int')
    class_name = coco_names[labels.numpy()[i] - 1]
    igg = cv2.rectangle(igg, (x1, y1), (x2, y2), (255, 0, 0), 1)
    igg = cv2.putText(igg, class_name, (x1, y1-10), font, 0.5, (255, 0, 0), 1, cv2.LINE_AA)


cv2_imshow(igg)
```