```python
#a. Import the necessary packages
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import numpy as np
```

```
Matplotlib is building the font cache; this may take a moment.
```

```python
#b. Load the training and testing data (MNIST/CIFAR10
train_data_dir = 'mnist-jpg/train'
test_data_dir = 'mnist-jpg/test'

# Image data generator for training data
train_datagen = ImageDataGenerator(
rescale=1.0/255
)

# Image data generator for testing data
test_datagen = ImageDataGenerator(
rescale=1.0/255
)

# Create data generators
train_batch_size = 10000
train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(28, 28),  # Resize images to 28x28
    batch_size=train_batch_size,
    class_mode='categorical',
     color_mode='grayscale',# Use 'categorical' for one-hot encoded
labels
    shuffle=True,
)

# Load test data without labels (class_mode=None)
test_batch_size = 2000
test_generator = test_datagen.flow_from_directory(
    test_data_dir,
    target_size=(28, 28),  # Resize images to 28x28
    batch_size=test_batch_size,
    class_mode='categorical',  # Use 'categorical' for one-hot encoded
labels
      color_mode='grayscale',
    shuffle=True,
)
```

```
Found 60000 images belonging to 10 classes.
Found 10000 images belonging to 10 classes.
```

```
#Selecting first batch containing 10000 images
x_train, y_train = train_generator[0]
x_test, y_test = test_generator[0]

#c. Define the network architecture using Keras

model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28, 1)),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
```

```
C:\ProgramData\anaconda3\Lib\site-packages\keras\src\layers\reshaping\
flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
#d.Train the model using SGD
model.compile(optimizer='sgd', loss='categorical_crossentropy',
metrics=['accuracy'])
history = model.fit(x_train, y_train, epochs=8,
validation_data=(x_test, y_test))
```

```
Epoch 1/8
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.6612 - loss:
1.3873 - val_accuracy: 0.8045 - val_loss: 0.8389
Epoch 2/8
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.8414 - loss:
0.6829 - val_accuracy: 0.8510 - val_loss: 0.5828
Epoch 3/8
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.8701 - loss:
0.5188 - val_accuracy: 0.8685 - val_loss: 0.4948
Epoch 4/8
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.8841 - loss:
0.4458 - val_accuracy: 0.8720 - val_loss: 0.4505
Epoch 5/8
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.8932 - loss:
0.4027 - val_accuracy: 0.8855 - val_loss: 0.4168
Epoch 6/8
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.9032 - loss:
0.3728 - val_accuracy: 0.8935 - val_loss: 0.3944
Epoch 7/8
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.9044 - loss:
0.3510 - val_accuracy: 0.8940 - val_loss: 0.3792
Epoch 8/8
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.9115 - loss:
0.3338 - val_accuracy: 0.9010 - val_loss: 0.3661
```

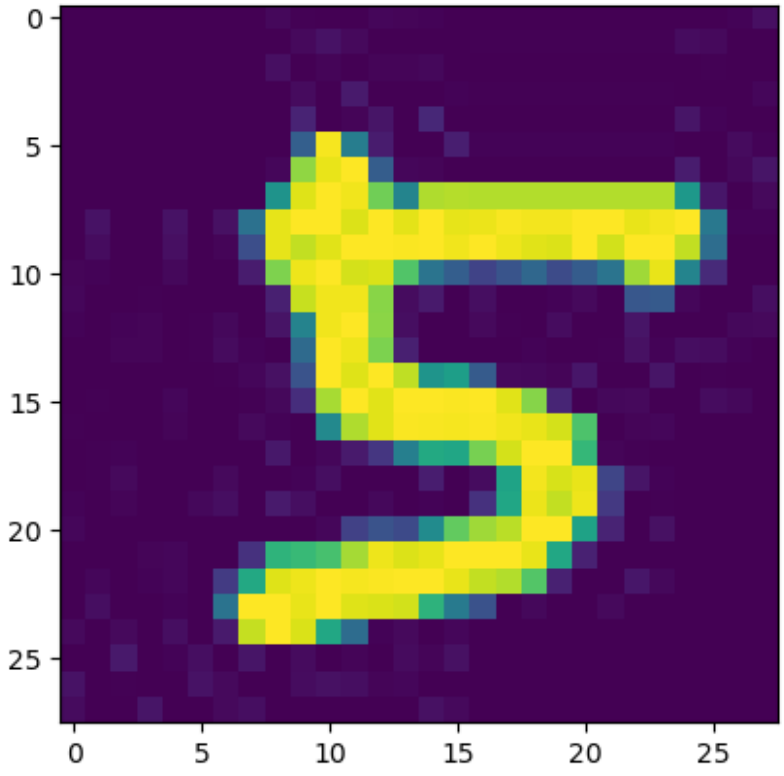
```
#e. Evaluate the network
```

```
test_loss, test_acc = model.evaluate(x_test, y_test)
print("Loss: ", test_loss)
print("Accuracy: ", test_acc)
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9010 - loss:
0.3661
Loss:  0.36612075567245483
Accuracy:  0.9010000228881836
```

```
n = 20
plt.imshow(x_test[n])
predicted_value = model.predict(x_test)
print("Actual Number: ",np.argmax(y_test[n]))
print("Predicted Number: ", np.argmax(predicted_value[n]))
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
Actual Number:  5
Predicted Number:  5
```



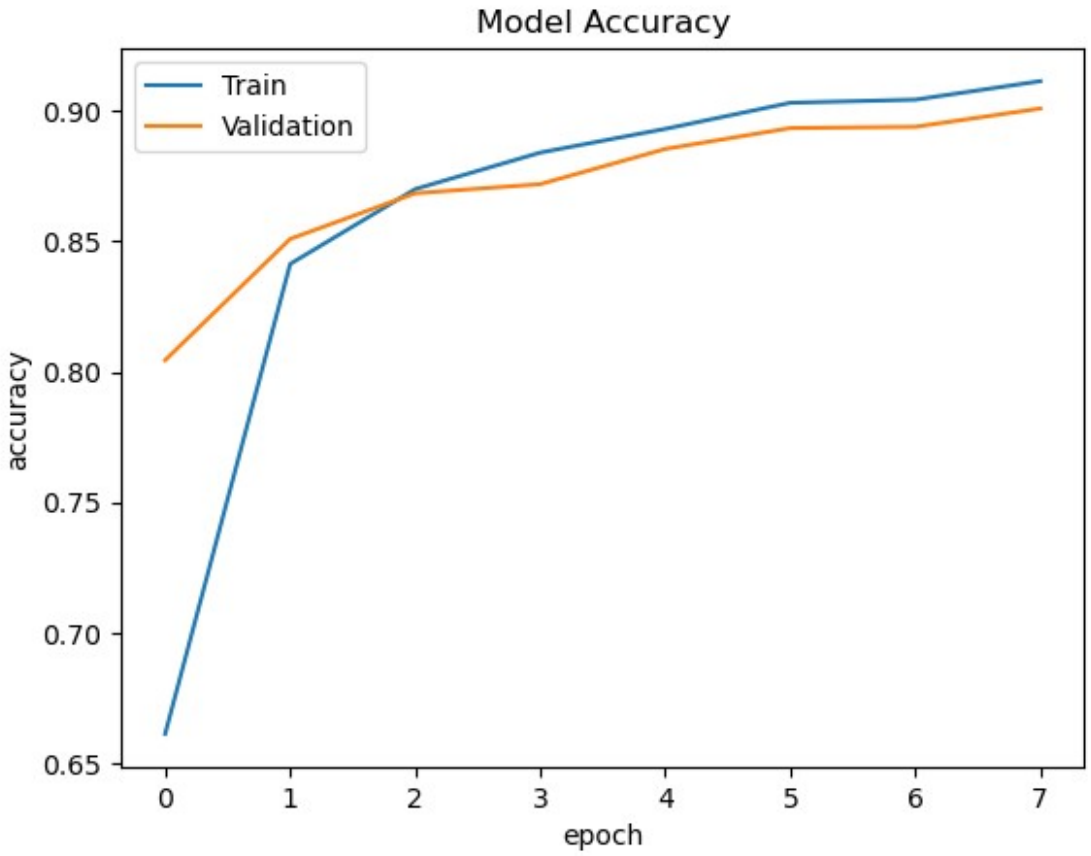

```
# Plot the training loss and accuracy

history = history.history
history.keys()
```

```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

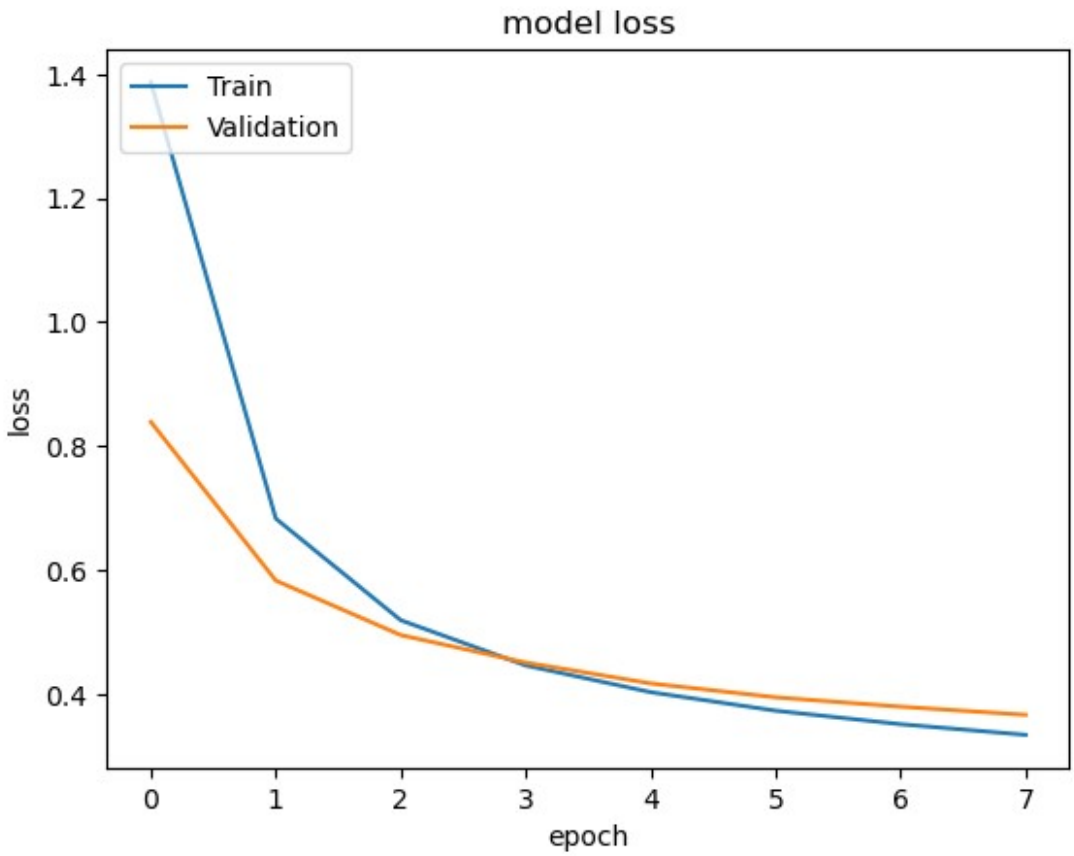
```
plt.plot(history['accuracy'])
plt.plot(history['val_accuracy'])
plt.title("Model Accuracy")
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Train', "Validation"], loc='upper left')
```

```
<matplotlib.legend.Legend at 0x174309d86e0>
```



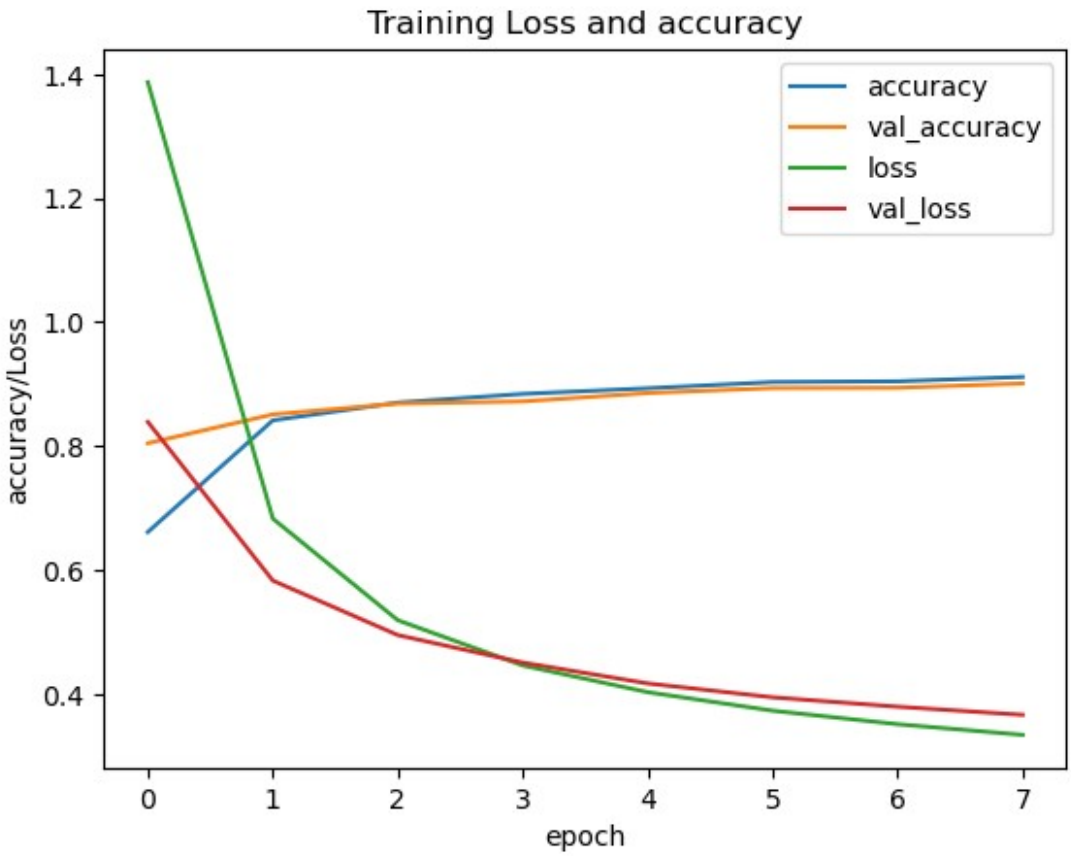

```
plt.plot(history['loss'])
plt.plot(history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
```

```
<matplotlib.legend.Legend at 0x174313ce0d0>
```




```
plt.plot(history['accuracy'])
plt.plot(history['val_accuracy'])
plt.plot(history['loss'])
plt.plot(history['val_loss'])
plt.title('Training Loss and accuracy')
plt.ylabel('accuracy/Loss')
plt.xlabel('epoch')
plt.legend(['accuracy', 'val_accuracy','loss','val_loss'])
```

```
<matplotlib.legend.Legend at 0x174324be710>
```


Training Loss and accuracy
accuracy
val_accuracy
loss
val_loss
accuracy/Loss
epoch
1.4
1.2
1.0
0.8
0.6
0.4
0
1
2
3
4
5
6
7