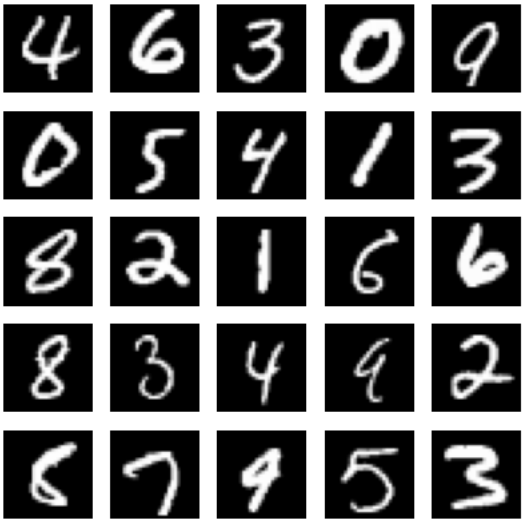
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)


import matplotlib.pyplot as plt # plotting library
%matplotlib inline


from keras.models import Sequential
from keras.layers import Dense , Activation, Dropout
from keras.optimizers import Adam ,RMSprop
from keras import  backend as K
```

```
Start coding or generate with AI.
```

```
from keras.datasets import mnist


# load dataset
(x_train, y_train),(x_test, y_test) = mnist.load_data()



# count the number of unique train labels
unique, counts = np.unique(y_train, return_counts=True)
print("Train labels: ", dict(zip(unique, counts)))



# count the number of unique test labels
unique, counts = np.unique(y_test, return_counts=True)
print("\nTest labels: ", dict(zip(unique, counts)))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
Train labels:  {0: 5923, 1: 6742, 2: 5958, 3: 6131, 4: 5842, 5: 5421, 6: 5918, 7: 6265, 8: 5851, 9: 5949}

Test labels:  {0: 980, 1: 1135, 2: 1032, 3: 1010, 4: 982, 5: 892, 6: 958, 7: 1028, 8: 974, 9: 1009}
```

```
indexes = np.random.randint(0, x_train.shape[0], size=25)
images = x_train[indexes]
labels = y_train[indexes]


# plot the 25 mnist digits
plt.figure(figsize=(5,5))
for i in range(len(indexes)):
    plt.subplot(5, 5, i + 1)
    image = images[i]
    plt.imshow(image, cmap='gray')
    plt.axis('off')

plt.show()
plt.savefig("mnist-samples.png")
plt.close('all')
```



```
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from keras.utils import to_categorical, plot_model


num_labels = len(np.unique(y_train))


y_train = to_categorical(y_train)
y_test = to_categorical(y_test)


image_size = x_train.shape[1]
input_size = image_size * image_size
input_size
```

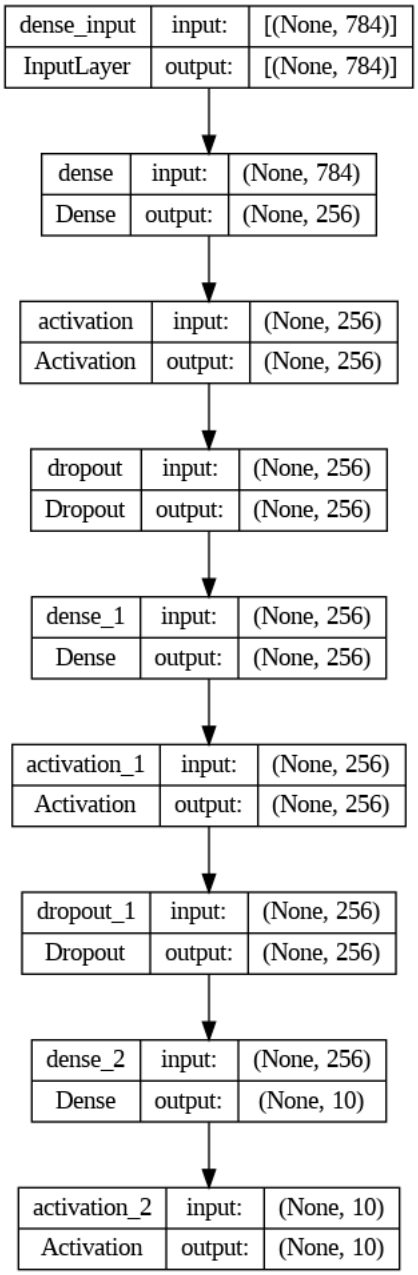
```
784
```

```
x_train = np.reshape(x_train, [-1, input_size])
x_train = x_train.astype('float32') / 255
x_test = np.reshape(x_test, [-1, input_size])
x_test = x_test.astype('float32') / 255


batch_size = 128
hidden_units = 256
dropout = 0.45


model = Sequential()
model.add(Dense(hidden_units, input_dim=input_size))
model.add(Activation('relu'))
model.add(Dropout(dropout))
model.add(Dense(hidden_units))
model.add(Activation('relu'))
model.add(Dropout(dropout))
model.add(Dense(num_labels))
model.add(Activation('softmax'))


model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 256)               200960

 activation (Activation)     (None, 256)               0

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 256)               65792

 activation_1 (Activation)   (None, 256)               0

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 10)                2570

 activation_2 (Activation)   (None, 10)                0
```

```
=================================================================
Total params: 269322 (1.03 MB)
Trainable params: 269322 (1.03 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
plot_model(model, to_file='mlp-mnist.png', show_shapes=True)
```




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

```
model.fit(x_train, y_train, epochs=20, batch_size=batch_size)
```

```
Epoch 1/20
469/469 [==============================] - 5s 8ms/step - loss: 0.4242 - accuracy: 0.8694
Epoch 2/20
469/469 [==============================] - 4s 8ms/step - loss: 0.1931 - accuracy: 0.9420
Epoch 3/20
469/469 [==============================] - 5s 10ms/step - loss: 0.1487 - accuracy: 0.9550
Epoch 4/20
469/469 [==============================] - 4s 8ms/step - loss: 0.1293 - accuracy: 0.9612
Epoch 5/20
469/469 [==============================] - 4s 8ms/step - loss: 0.1125 - accuracy: 0.9657
Epoch 6/20
469/469 [==============================] - 4s 9ms/step - loss: 0.1036 - accuracy: 0.9683
Epoch 7/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0962 - accuracy: 0.9699
Epoch 8/20
469/469 [==============================] - 4s 8ms/step - loss: 0.0868 - accuracy: 0.9733
Epoch 9/20
```

```
469/469 [==============================] - 5s 10ms/step - loss: 0.0809 - accuracy: 0.9744
Epoch 10/20
469/469 [==============================] - 4s 8ms/step - loss: 0.0779 - accuracy: 0.9755
Epoch 11/20
469/469 [==============================] - 4s 8ms/step - loss: 0.0744 - accuracy: 0.9763
Epoch 12/20
469/469 [==============================] - 4s 10ms/step - loss: 0.0712 - accuracy: 0.9775
Epoch 13/20
469/469 [==============================] - 4s 8ms/step - loss: 0.0680 - accuracy: 0.9779
Epoch 14/20
469/469 [==============================] - 4s 8ms/step - loss: 0.0653 - accuracy: 0.9783
Epoch 15/20
469/469 [==============================] - 5s 10ms/step - loss: 0.0638 - accuracy: 0.9798
Epoch 16/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0622 - accuracy: 0.9805
Epoch 17/20
469/469 [==============================] - 4s 8ms/step - loss: 0.0586 - accuracy: 0.9809
Epoch 18/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0566 - accuracy: 0.9818
Epoch 19/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0534 - accuracy: 0.9829
Epoch 20/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0526 - accuracy: 0.9830
<keras.src.callbacks.History at 0x7992b6655180>
```

```
loss, acc = model.evaluate(x_test, y_test, batch_size=batch_size)
print("\nTest accuracy: %.1f%%" % (100.0 * acc))
```

```
79/79 [==============================] - 0s 3ms/step - loss: 0.0648 - accuracy: 0.9829

Test accuracy: 98.3%
```

```
from keras.regularizers import l2
model.add(Dense(hidden_units,
                kernel_regularizer=l2(0.001),
                input_dim=input_size))
```