```
#Importing important modules
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.a

Enter your authorization code:
··········
Mounted at /content/drive
```

```
import h5py
import numpy as np

# Open the file as readonly. The file should be present inside a directory called "data" in the same folder as code
h5f = h5py.File('/content/drive/My Drive/AIML/data/SVHN_single_grey1.h5', 'r')

# Load the training, test and validation set
x_train = h5f['X_train'][:]
y_train = h5f['y_train'][:]
x_test = h5f['X_test'][:]
y_test = h5f['y_test'][:]


# Close this file
h5f.close()
```

```
hsf.close()


%matplotlib inline

import matplotlib.pyplot as plt
w=10
h=10
fig=plt.figure(figsize=(8, 8))
columns = 10
rows = 10
for i in range(1, columns*rows +1):
    img = x_test[i]
    fig.add_subplot(rows, columns, i)
    plt.imshow(img, cmap='gray')
plt.show()
```

```
print(x_train.shape)
```

```
(42000, 32, 32)
```

```
print(y_train.shape)
```

```
(42000,)
```

```
x_train = x_train[0:22000,:]
y_train = y_train[0:22000]
```

```
y_train = y_train[0:22000]
x_test = x_test[0:8000,:]
y_test = y_test[0:8000]


print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(22000, 32, 32)
(22000,)
(8000, 32, 32)
(8000,)
```

```
print(x_train.dtype)
```

```
float32
```

```
#Importing opencv module for the resizing function
import cv2

#Create a resized dataset for training and testing inputs with corresponding size. Here we are resizing it to 28X28 (same input size
x_train_res = np.zeros((x_train.shape[0],28,28), dtype=np.float32)
for i in range(x_train.shape[0]):
  #using cv2.resize to resize each train example to 28X28 size using Cubic interpolation
  x_train_res[i,:,:] = cv2.resize(x_train[i], dsize=(28, 28), interpolation=cv2.INTER_CUBIC)

x_test_res = np.zeros((x_test.shape[0],28,28), dtype=np.float32)
for i in range(x_test.shape[0]):
  #using cv2.resize to resize each test example to 28X28 size using Cubic interpolation
  x_test_res[i,:,:] = cv2.resize(x_test[i], dsize=(28, 28), interpolation=cv2.INTER_CUBIC)

#We don't need the original dataset anynmore so we can clear up memory consumed by original dataset
del x_train
del x_test

print(x_train_res.shape)
print(x_test_res.shape)
```

```
print(x_test_res.shape)
```

```
(22000, 28, 28)
(8000, 28, 28)
```

```
# input image dimensions
img_rows, img_cols = 28, 28


#Keras expects data to be in the format (N_E.N_H,N_W,N_C)  N_E = Number of Examples, N_H = height, N_W = Width, N_C = Number of Chann
x_train = x_train_res.reshape(x_train_res.shape[0], img_rows, img_cols, 1)
x_test = x_test_res.reshape(x_test_res.shape[0], img_rows, img_cols, 1)
input_shape = (img_rows, img_cols, 1)


x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

#Normalizing the input
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```
x_train shape: (22000, 28, 28, 1)
22000 train samples
8000 test samples
```

```
batch_size = 128
num_classes = 10
epochs = 50


print(y_train[0])
```

```
2
```

```
import keras
```

```
import keras
# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)


print(y_train[0])
```

```
[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
```

```
#Initialize the model
model = Sequential()

#Add a Convolutional Layer with 32 filters of size 3X3 and activation function as 'ReLU'
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape,name='conv_1'))

#Add a Convolutional Layer with 64 filters of size 3X3 and activation function as 'ReLU'
model.add(Conv2D(64, (3, 3), activation='relu',name='conv_2'))

#Add a MaxPooling Layer of size 2X2
model.add(MaxPooling2D(pool_size=(2, 2),name='max_1'))

#Apply Dropout with 0.25 probability
model.add(Dropout(0.25,name='drop_1'))

#Flatten the layer
model.add(Flatten())

#Add Fully Connected Layer with 128 units and activation function as 'ReLU'
model.add(Dense(128, activation='relu',name='dense_1'))

#Apply Dropout with 0.5 probability
model.add(Dropout(0.5,name='drop_2'))

#Add Fully Connected Layer with 10 units and activation function as 'softmax'
model.add(Dense(num_classes, activation='softmax',name='dense_2'))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:66: The name tf.get_default_g

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:541: The name tf.placeholder

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:4432: The name tf.random_unif

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:4267: The name tf.nn.max_pool

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:148: The name tf.placeholder_

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:3733: calling dropout (from t
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
```

```
from keras.optimizers import Adam
from keras.losses import categorical_crossentropy

#To use adam optimizer for learning weights with learning rate = 0.001
optimizer = Adam(lr=0.001)
#Set the loss function and optimizer for the model training
model.compile(loss=categorical_crossentropy,
              optimizer=optimizer,
              metrics=['accuracy'])
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:793: The name tf.train.Optimizer is deprecate

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:3576: The name tf.log is depr
```

```
#Training on the dataset
model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=20,
          verbose=1,
          validation_data=(x_test, y_test))
```

```
Train on 22000 samples, validate on 8000 samples
Epoch 1/20
22000/22000 [==============================] - 58s 3ms/step - loss: 0.3620 - acc: 0.8796 - val_loss: 0.3807 - val_acc: 0.8910
Epoch 2/20
22000/22000 [==============================] - 58s 3ms/step - loss: 0.3626 - acc: 0.8796 - val_loss: 0.3906 - val_acc: 0.8902
Epoch 3/20
22000/22000 [==============================] - 58s 3ms/step - loss: 0.3429 - acc: 0.8852 - val_loss: 0.3857 - val_acc: 0.8920
Epoch 4/20
22000/22000 [==============================] - 58s 3ms/step - loss: 0.3444 - acc: 0.8847 - val_loss: 0.3877 - val_acc: 0.8920
Epoch 5/20
22000/22000 [==============================] - 58s 3ms/step - loss: 0.3324 - acc: 0.8917 - val_loss: 0.3862 - val_acc: 0.8912
Epoch 6/20
22000/22000 [==============================] - 58s 3ms/step - loss: 0.3263 - acc: 0.8905 - val_loss: 0.4017 - val_acc: 0.8886
Epoch 7/20
22000/22000 [==============================] - 58s 3ms/step - loss: 0.3175 - acc: 0.8930 - val_loss: 0.3918 - val_acc: 0.8929
Epoch 8/20
22000/22000 [==============================] - 58s 3ms/step - loss: 0.3045 - acc: 0.8981 - val_loss: 0.3909 - val_acc: 0.8952
Epoch 9/20
22000/22000 [==============================] - 58s 3ms/step - loss: 0.3053 - acc: 0.8945 - val_loss: 0.3836 - val_acc: 0.8952
Epoch 10/20
22000/22000 [==============================] - 57s 3ms/step - loss: 0.3029 - acc: 0.8985 - val_loss: 0.3978 - val_acc: 0.8935
Epoch 11/20
22000/22000 [==============================] - 58s 3ms/step - loss: 0.2864 - acc: 0.9043 - val_loss: 0.3895 - val_acc: 0.8958
Epoch 12/20
22000/22000 [==============================] - 57s 3ms/step - loss: 0.2863 - acc: 0.9028 - val_loss: 0.4009 - val_acc: 0.8931
Epoch 13/20
22000/22000 [==============================] - 58s 3ms/step - loss: 0.2834 - acc: 0.9061 - val_loss: 0.3942 - val_acc: 0.8938
Epoch 14/20
22000/22000 [==============================] - 57s 3ms/step - loss: 0.2711 - acc: 0.9072 - val_loss: 0.3876 - val_acc: 0.8984
Epoch 15/20
22000/22000 [==============================] - 58s 3ms/step - loss: 0.2692 - acc: 0.9076 - val_loss: 0.3878 - val_acc: 0.8970
Epoch 16/20
22000/22000 [==============================] - 58s 3ms/step - loss: 0.2689 - acc: 0.9070 - val_loss: 0.4074 - val_acc: 0.8926
Epoch 17/20
22000/22000 [==============================] - 57s 3ms/step - loss: 0.2560 - acc: 0.9122 - val_loss: 0.3870 - val_acc: 0.8975
Epoch 18/20
22000/22000 [==============================] - 58s 3ms/step - loss: 0.2527 - acc: 0.9128 - val_loss: 0.3933 - val_acc: 0.8974
Epoch 19/20
22000/22000 [==============================] - 58s 3ms/step - loss: 0.2451 - acc: 0.9157 - val_loss: 0.3964 - val_acc: 0.8955
Epoch 20/20
22000/22000 [==============================] - 58s 3ms/step - loss: 0.2479 - acc: 0.9139 - val_loss: 0.3919 - val_acc: 0.8964
<keras.callbacks.History at 0x7f5e6b5dd4a8>
```

```
#Testing the model on test set
score = model.evaluate(x_test, y_test)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
8000/8000 [==============================] - 6s 694us/step
Test loss: 0.39186338885873556
Test accuracy: 0.896375
```

```
import tensorflow as tf
#Initialize the model
model = Sequential()

#Add a Convolutional Layer with 32 filters of size 3X3 and activation function as 'ReLU'
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape,name='conv_1'))

#Add a Convolutional Layer with 64 filters of size 3X3 and activation function as 'ReLU'
model.add(Conv2D(64, (3, 3), activation='relu',name='conv_2'))

#Add a MaxPooling Layer of size 2X2
model.add(MaxPooling2D(pool_size=(2, 2),name='max_1'))

#Apply Dropout with 0.25 probability
model.add(Dropout(0.25,name='drop_1'))

#Flatten the layer
model.add(Flatten())

#Add Fully Connected Layer with 128 units and activation function as 'ReLU'
model.add(Dense(128, activation='relu',name='dense_1'))
```

```
#Normalize the data
model.add(keras.layers.BatchNormalization())
#Apply Dropout with 0.5 probability
model.add(Dropout(0.5,name='drop_2'))

#Add Fully Connected Layer with 10 units and activation function as 'softmax'
model.add(Dense(num_classes, activation='softmax',name='dense_2'))
model.compile(loss=categorical_crossentropy,
              optimizer=optimizer,
              metrics=['accuracy'])


#Training on the dataset
model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=20,
          verbose=1,
          validation_data=(x_test, y_test))
```

```
Train on 22000 samples, validate on 8000 samples
Epoch 1/20
22000/22000 [==============================] - 60s 3ms/step - loss: 1.2916 - acc: 0.5777 - val_loss: 0.6269 - val_acc: 0.8236
Epoch 2/20
22000/22000 [==============================] - 58s 3ms/step - loss: 0.7226 - acc: 0.7793 - val_loss: 0.5763 - val_acc: 0.8281
Epoch 3/20
22000/22000 [==============================] - 59s 3ms/step - loss: 0.6323 - acc: 0.8081 - val_loss: 0.5090 - val_acc: 0.8449
Epoch 4/20
22000/22000 [==============================] - 59s 3ms/step - loss: 0.5902 - acc: 0.8203 - val_loss: 0.4663 - val_acc: 0.8620
Epoch 5/20
22000/22000 [==============================] - 62s 3ms/step - loss: 0.5552 - acc: 0.8292 - val_loss: 0.4467 - val_acc: 0.8688
Epoch 6/20
22000/22000 [==============================] - 62s 3ms/step - loss: 0.5337 - acc: 0.8381 - val_loss: 0.4283 - val_acc: 0.8774
Epoch 7/20
22000/22000 [==============================] - 61s 3ms/step - loss: 0.5120 - acc: 0.8454 - val_loss: 0.4377 - val_acc: 0.8746
Epoch 8/20
22000/22000 [==============================] - 62s 3ms/step - loss: 0.4911 - acc: 0.8477 - val_loss: 0.4215 - val_acc: 0.8764
Epoch 9/20
22000/22000 [==============================] - 62s 3ms/step - loss: 0.4685 - acc: 0.8554 - val_loss: 0.4374 - val_acc: 0.8721
Epoch 10/20
22000/22000 [==============================] - 62s 3ms/step - loss: 0.4572 - acc: 0.8593 - val_loss: 0.4101 - val_acc: 0.8818
Epoch 11/20
22000/22000 [==============================] - 61s 3ms/step - loss: 0.4531 - acc: 0.8603 - val_loss: 0.4328 - val_acc: 0.8752
Epoch 12/20
22000/22000 [==============================] - 62s 3ms/step - loss: 0.4431 - acc: 0.8640 - val_loss: 0.4136 - val_acc: 0.8836
Epoch 13/20
22000/22000 [==============================] - 61s 3ms/step - loss: 0.4223 - acc: 0.8675 - val_loss: 0.3970 - val_acc: 0.8858
Epoch 14/20
22000/22000 [==============================] - 62s 3ms/step - loss: 0.4170 - acc: 0.8685 - val_loss: 0.3950 - val_acc: 0.8881
Epoch 15/20
22000/22000 [==============================] - 62s 3ms/step - loss: 0.4131 - acc: 0.8702 - val_loss: 0.3886 - val_acc: 0.8922
Epoch 16/20
22000/22000 [==============================] - 61s 3ms/step - loss: 0.3984 - acc: 0.8753 - val_loss: 0.4062 - val_acc: 0.8848
Epoch 17/20
22000/22000 [==============================] - 62s 3ms/step - loss: 0.3913 - acc: 0.8776 - val_loss: 0.4022 - val_acc: 0.8822
Epoch 18/20
22000/22000 [==============================] - 61s 3ms/step - loss: 0.3965 - acc: 0.8736 - val_loss: 0.3775 - val_acc: 0.8935
Epoch 19/20
22000/22000 [==============================] - 62s 3ms/step - loss: 0.3821 - acc: 0.8784 - val_loss: 0.3848 - val_acc: 0.8902
Epoch 20/20
22000/22000 [==============================] - 61s 3ms/step - loss: 0.3882 - acc: 0.8783 - val_loss: 0.3829 - val_acc: 0.8909
<keras.callbacks.History at 0x7f5e4e10b710>
```

```
#Testing the model on test set
score = model.evaluate(x_test, y_test)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
8000/8000 [==============================] - 6s 797us/step
Test loss: 0.3829143206179142
Test accuracy: 0.890875
```