```
from zipfile import ZipFile
file_name="data.zip"
with ZipFile(file_name,'r') as zip:
  zip.extractall()
  print('done')
```

```
done
```

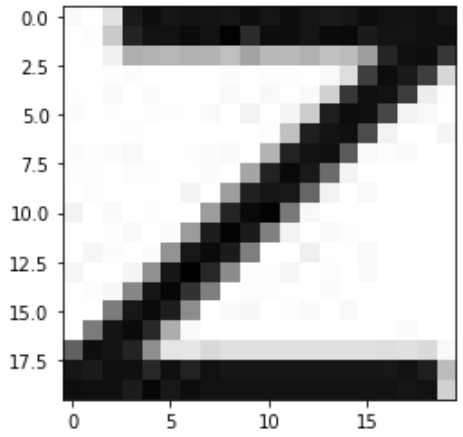
```
import os
import numpy as np
import cv2 as cv


letters = [
            '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D',
            'E', 'F', 'G', 'H', 'J', 'K', 'L', 'M', 'N', 'P', 'Q', 'R', 'S', 'T',
            'U', 'V', 'W', 'X', 'Y', 'Z'
        ]

def read_training_data(training_directory):
    image_data = []
    target_data = []
    for i in letters:
        for j in range(10):
            image_path = os.path.join(training_directory,i,i + '_' + str(j) + '.jpg')
            img = cv.imread(image_path,0)

            resized = img.reshape(-1)
            image_data.append(resized)
            target_data.append(i)

    return (np.array(image_data), np.array(target_data))


print('reading data.......')
training_dataset_dir = 'train20X20'
image_data, target_data = read_training_data(training_dataset_dir)
print('.....completed')
```

```
reading data.......
.....completed
```

```
print(target_data.shape)
print(image_data.shape)
```

```
(340,)
(340, 400)
```

```
import matplotlib.pyplot as plt
img = image_data[338]
img = img.reshape((20,20))
plt.imshow(img,cmap='gray')
#plt.title(train.iloc[0,0])
plt.axis("on")
plt.show()
```




```
# Normalize the data
image_data = image_data / 255.0
#test = test / 255.0
print(image_data[7].min(),"-",image_data[7].max())
```

```
0.0 - 1.0
```

```
image_data=image_data.reshape(-1,20,20,1)
#target_data = target_data.reshape(-1,20,20,1)
print("Image_data shape: ",image_data.shape)
#print("test shape: ",test.shape)
#print(target_data)
```

```
Image_data shape:  (340, 20, 20, 1)
```

```
from numpy import array
from numpy import argmax
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
values = array(target_data)
print(values)
# integer encode
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(values)
print(integer_encoded)
# binary encode
onehot_encoder = OneHotEncoder(sparse=False)
integer_encoded = integer_encoded.reshape(len(integer_encoded), 1)
onehot_encoded = onehot_encoder.fit_transform(integer_encoded)
print(onehot_encoded)
print(onehot_encoded.shape)
```

```
['0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '1' '1' '1' '1' '1' '1' '1' '1'
 '1' '1' '2' '2' '2' '2' '2' '2' '2' '2' '2' '2' '3' '3' '3' '3' '3' '3'
 '3' '3' '3' '3' '4' '4' '4' '4' '4' '4' '4' '4' '4' '4' '5' '5' '5' '5'
 '5' '5' '5' '5' '5' '5' '6' '6' '6' '6' '6' '6' '6' '6' '6' '6' '7' '7'
 '7' '7' '7' '7' '7' '7' '8' '8' '8' '8' '8' '8' '8' '8' '8' '8'
 '9' '9' '9' '9' '9' '9' '9' '9' '9' '9' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A'
 'A' 'A' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'C' 'C' 'C' 'C' 'C' 'C'
 'C' 'C' 'C' 'C' 'D' 'D' 'D' 'D' 'D' 'D' 'D' 'D' 'D' 'D' 'E' 'E' 'E' 'E'
 'E' 'E' 'E' 'E' 'F' 'F' 'F' 'F' 'F' 'F' 'F' 'F' 'F' 'F' 'G' 'G'
 'G' 'G' 'G' 'G' 'G' 'G' 'H' 'H' 'H' 'H' 'H' 'H' 'H' 'H' 'H' 'H'
 'J' 'J' 'J' 'J' 'J' 'J' 'J' 'J' 'J' 'J' 'K' 'K' 'K' 'K' 'K' 'K' 'K' 'K'
 'K' 'K' 'L' 'L' 'L' 'L' 'L' 'L' 'L' 'L' 'L' 'L' 'M' 'M' 'M' 'M' 'M' 'M'
 'M' 'M' 'M' 'M' 'N' 'N' 'N' 'N' 'N' 'N' 'N' 'N' 'N' 'N' 'P' 'P' 'P' 'P'
 'P' 'P' 'P' 'P' 'P' 'P' 'Q' 'Q' 'Q' 'Q' 'Q' 'Q' 'Q' 'Q' 'Q' 'Q' 'R' 'R'
 'R' 'R' 'R' 'R' 'R' 'R' 'S' 'S' 'S' 'S' 'S' 'S' 'S' 'S' 'S' 'S'
 'T' 'T' 'T' 'T' 'T' 'T' 'T' 'T' 'T' 'T' 'U' 'U' 'U' 'U' 'U' 'U' 'U' 'U'
 'U' 'U' 'V' 'V' 'V' 'V' 'V' 'V' 'V' 'V' 'V' 'V' 'W' 'W' 'W' 'W' 'W' 'W'
 'W' 'W' 'W' 'W' 'X' 'X' 'X' 'X' 'X' 'X' 'X' 'X' 'X' 'X' 'Y' 'Y' 'Y' 'Y'
 'Y' 'Y' 'Y' 'Y' 'Y' 'Y' 'Z' 'Z' 'Z' 'Z' 'Z' 'Z' 'Z' 'Z' 'Z' 'Z']
[ 0  0  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1  1  1  2  2  2  2
  2  2  2  2  2  2  3  3  3  3  3  3  3  3  3  3  4  4  4  4  4  4  4  4
  4  4  5  5  5  5  5  5  5  5  5  5  6  6  6  6  6  6  6  6  6  6  7  7
  7  7  7  7  7  7  8  8  8  8  8  8  8  8  8  8  9  9  9  9  9  9
  9  9  9  9 10 10 10 10 10 10 10 10 10 10 11 11 11 11 11 11 11 11 11 11
 12 12 12 12 12 12 12 12 12 12 13 13 13 13 13 13 13 13 13 13 14 14 14 14
 14 14 14 14 14 14 15 15 15 15 15 15 15 15 15 15 16 16 16 16 16 16 16 16
 16 16 17 17 17 17 17 17 17 17 17 17 18 18 18 18 18 18 18 18 18 18 19 19
 19 19 19 19 19 19 19 20 20 20 20 20 20 20 20 20 20 21 21 21 21 21 21
 21 21 21 21 22 22 22 22 22 22 22 22 22 22 23 23 23 23 23 23 23 23 23 23
 24 24 24 24 24 24 24 24 24 24 25 25 25 25 25 25 25 25 25 25 26 26 26 26
 26 26 26 26 27 27 27 27 27 27 27 27 27 27 28 28 28 28 28 28 28 28
 28 28 29 29 29 29 29 29 29 29 29 29 30 30 30 30 30 30 30 30 30 30 31 31
 31 31 31 31 31 31 31 32 32 32 32 32 32 32 32 32 32 33 33 33 33 33 33
 33 33 33 33]
[[1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 1.]]
(340, 34)
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/_encoders.py:415: FutureWarning: T
If you want the future behaviour and silence this warning, you can specify "categories='auto'".
In case you used a LabelEncoder before this OneHotEncoder to convert the categories to integers
  warnings.warn(msg, FutureWarning)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(image_data,onehot_encoded, test_size = 0.2, rand
print("x_train shape",X_train.shape)
print("x_test shape",X_test.shape)
print("y_train shape",Y_train.shape)
print("y_test shape",Y_test.shape)
```

```
x_train shape (272, 20, 20, 1)
x_test shape (68, 20, 20, 1)
y_train shape (272, 34)
y_test shape (68, 34)
```

```
from keras.models import Sequential
from keras.layers import Dense, Dropout,Conv2D,MaxPooling2D, Flatten
#create model
model = Sequential()
```

```
#add model layers
model.add(Conv2D(filters=64, kernel_size = (3,3),padding='Same', activation="relu", input_shape=(20,
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(filters=128, kernel_size = (3,3),padding='Same', activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512,activation="relu"))
model.add(Dropout(0.50))

model.add(Dense(34,activation="softmax"))

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



model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_17 (Conv2D)           (None, 20, 20, 64)        640
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 10, 10, 64)        0
_________________________________________________________________
dropout_25 (Dropout)         (None, 10, 10, 64)        0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 10, 10, 128)       73856
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 5, 5, 128)         0
_________________________________________________________________
dropout_26 (Dropout)         (None, 5, 5, 128)         0
_________________________________________________________________
flatten_9 (Flatten)          (None, 3200)              0
_________________________________________________________________
dense_17 (Dense)             (None, 512)               1638912
_________________________________________________________________
dropout_27 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_18 (Dense)             (None, 34)                17442
=================================================================
Total params: 1,730,850
Trainable params: 1,730,850
Non-trainable params: 0
_________________________________________________________________
```

```
model.fit(X_train,Y_train,epochs=40,verbose=2,validation_data=(X_test,Y_test))
```

```
Train on 272 samples, validate on 68 samples
Epoch 1/40
 - 2s - loss: 3.5782 - acc: 0.0184 - val_loss: 3.5664 - val_acc: 0.0147
Epoch 2/40
 - 1s - loss: 3.4989 - acc: 0.0588 - val_loss: 3.5581 - val_acc: 0.0294
Epoch 3/40
 - 1s - loss: 3.4423 - acc: 0.0846 - val_loss: 3.6023 - val_acc: 0.1029
Epoch 4/40
 - 1s - loss: 3.3725 - acc: 0.1213 - val_loss: 3.5440 - val_acc: 0.1029
Epoch 5/40
 - 1s - loss: 3.1966 - acc: 0.1287 - val_loss: 3.3107 - val_acc: 0.2059
Epoch 6/40
 - 1s - loss: 2.8327 - acc: 0.2684 - val_loss: 2.9373 - val_acc: 0.2794
Epoch 7/40
 - 1s - loss: 2.3563 - acc: 0.4007 - val_loss: 2.4193 - val_acc: 0.3971
Epoch 8/40
 - 1s - loss: 1.8325 - acc: 0.5000 - val_loss: 1.8247 - val_acc: 0.5735
Epoch 9/40
 - 1s - loss: 1.3220 - acc: 0.6287 - val_loss: 1.4840 - val_acc: 0.5735
Epoch 10/40
 - 1s - loss: 1.0902 - acc: 0.7022 - val_loss: 1.0731 - val_acc: 0.7353
Epoch 11/40
 - 1s - loss: 0.9130 - acc: 0.7169 - val_loss: 0.9562 - val_acc: 0.7500
Epoch 12/40
 - 1s - loss: 0.6856 - acc: 0.7904 - val_loss: 0.5629 - val_acc: 0.8971
Epoch 13/40
 - 1s - loss: 0.5175 - acc: 0.8419 - val_loss: 0.5021 - val_acc: 0.8676
Epoch 14/40
 - 1s - loss: 0.4285 - acc: 0.8787 - val_loss: 0.3164 - val_acc: 0.9559
Epoch 15/40
 - 1s - loss: 0.3314 - acc: 0.8824 - val_loss: 0.3161 - val_acc: 0.8824
Epoch 16/40
 - 1s - loss: 0.3631 - acc: 0.9044 - val_loss: 0.3528 - val_acc: 0.8971
Epoch 17/40
 - 1s - loss: 0.2737 - acc: 0.9191 - val_loss: 0.2549 - val_acc: 0.9265
Epoch 18/40
 - 1s - loss: 0.2550 - acc: 0.9191 - val_loss: 0.2331 - val_acc: 0.9265
Epoch 19/40
 - 1s - loss: 0.1938 - acc: 0.9485 - val_loss: 0.2268 - val_acc: 0.9412
Epoch 20/40
 - 1s - loss: 0.1653 - acc: 0.9559 - val_loss: 0.1978 - val_acc: 0.9118
Epoch 21/40
 - 1s - loss: 0.1886 - acc: 0.9412 - val_loss: 0.1313 - val_acc: 0.9559
Epoch 22/40
 - 1s - loss: 0.1671 - acc: 0.9449 - val_loss: 0.2307 - val_acc: 0.9265
Epoch 23/40
 - 1s - loss: 0.2068 - acc: 0.9375 - val_loss: 0.1412 - val_acc: 0.9265
Epoch 24/40
 - 1s - loss: 0.1507 - acc: 0.9596 - val_loss: 0.1296 - val_acc: 0.9706
Epoch 25/40
 - 1s - loss: 0.1378 - acc: 0.9559 - val_loss: 0.1110 - val_acc: 0.9853
Epoch 26/40
 - 1s - loss: 0.1113 - acc: 0.9743 - val_loss: 0.1008 - val_acc: 0.9853
Epoch 27/40
 - 1s - loss: 0.0908 - acc: 0.9816 - val_loss: 0.0986 - val_acc: 0.9853
Epoch 28/40
 - 1s - loss: 0.0853 - acc: 0.9779 - val_loss: 0.1452 - val_acc: 0.9412
Epoch 29/40
 - 1s - loss: 0.0891 - acc: 0.9779 - val_loss: 0.0891 - val_acc: 0.9706
Epoch 30/40
 - 1s - loss: 0.0860 - acc: 0.9779 - val_loss: 0.1025 - val_acc: 0.9559
Epoch 31/40
 - 1s - loss: 0.0896 - acc: 0.9853 - val_loss: 0.0894 - val_acc: 0.9853
Epoch 32/40
 - 1s - loss: 0.1104 - acc: 0.9596 - val_loss: 0.1132 - val_acc: 0.9706
Epoch 33/40
 - 1s - loss: 0.0801 - acc: 0.9743 - val_loss: 0.0900 - val_acc: 0.9706
Epoch 34/40
```

```
 - 1s - loss: 0.0894 - acc: 0.9816 - val_loss: 0.1048 - val_acc: 0.9706
Epoch 35/40
 - 1s - loss: 0.0717 - acc: 0.9816 - val_loss: 0.1063 - val_acc: 0.9559
Epoch 36/40
 - 1s - loss: 0.0648 - acc: 0.9816 - val_loss: 0.0872 - val_acc: 0.9853
Epoch 37/40
 - 1s - loss: 0.0572 - acc: 0.9853 - val_loss: 0.1592 - val_acc: 0.9412
Epoch 38/40
 - 1s - loss: 0.0369 - acc: 0.9963 - val_loss: 0.0797 - val_acc: 0.9853
Epoch 39/40
 - 1s - loss: 0.0456 - acc: 0.9853 - val_loss: 0.1318 - val_acc: 0.9412
Epoch 40/40
 - 1s - loss: 0.0403 - acc: 0.9963 - val_loss: 0.0617 - val_acc: 0.9853
<keras.callbacks.History at 0x7fbd64d67dd8>
```

```
y_pred = model.predict(X_test)
X_test__ = X_test.reshape(X_test.shape[0], 20, 20)

fig, axis = plt.subplots(4, 4, figsize=(12, 14))
for i, ax in enumerate(axis.flat):
    ax.imshow(X_test__[i], cmap='binary')
    num=y_pred[i].argmax()
    if num <=33 and num>0:
      if (num==1):
        print(1)
      elif(num==2):
        print(2)
      elif(num==3):
        print(3)
      elif(num==4):
        print(4)
      elif(num==5):
        print(5)
      elif(num==6):
        print(6)
      elif(num==7):
        print(7)
      elif(num==8):
        print(8)
      elif(num==9):
        print(9)
      elif(num==10):
        print("A")
      elif(num==11):
        print("B")
      elif(num==12):
        print("C")
      elif(num==13):
        print("D")
      elif(num==14):
        print("E")
      elif(num==15):
        print("F")
      elif(num==16):
        print("G")
      elif(num==17):
        print("H")
      elif(num==18):
        print("J")
      elif(num==19):
        print("K")
```

```
      elif(num==20):
        print("L")
      elif(num==21):
        print("M")
      elif(num==22):
        print("N")
      elif(num==23):
        print("P")
      elif(num==24):
        print("Q")
      elif(num==25):
        print("R")
      elif(num==26):
        print("S")
      elif(num==27):
        print("T")
      elif(num==28):
        print("U")
      elif(num==29):
        print("V")
      elif(num==30):
        print("W")
      elif(num==31):
        print("X")
      elif(num==32):
        print("Y")
      else:
        print("Z")
    else:
     print(0)
     #ax.set(title = f"Real is {Y_test[i].argmax()}\nPredict is {y_pred[i].argmax()}")
```

5
U
K
0
6
7
6
9
E
E
9
F
6
L
V
H