[CAPTCHA Recognition using CNN Gitlink]

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
Mounted at /content/drive
```

```
#importing libraries
import numpy as np

%matplotlib inline
#to use as command line calls #using inline graphs will come next to code

import matplotlib.pyplot as plt #for graphs
import os #for operating system dependent fucntionality
from keras import layers #for building layers of neural net
from keras.models import Model
from keras.models import load_model
from keras import callbacks #for training logs, saving to disk periodically
import cv2 #OpenCV(Open Source computer vision lib), containg CV algos
import string
```

```
#print images in dataset
os.listdir("/content/drive/My Drive/captcha_dataset/samples")
```

```
 'mpp2y.png',
 'dn5df.png',
 '6cm6m.png',
 'n265y.png',
 '23n88.png',
 'e72cd.png',
 'ep85x.png',
 'y4n6m.png',
 'w6yne.png',
 'en32e.png',
 'dn2ym.png',
 'dcnp8.png',
 'xxw44.png',
 'e2mg2.png',
 'yfdn7.png',
 'mfc35.png',
 'b84xc.png',
 '8gecm.png',
 'wnmyn.png',
 'ppwyd.png',
 'nbcgb.png',
 'y3c58.png',
 'f4fn2.png',
 '2w4y7.png',
 'wmpmp.png',
 'neggn.png',
 '3xcgg.png',
 'dc436.png',
 'bdbb3.png',
 'gc2wd.png',
```

```
 '68wfd.png',
 'g2fnw.png',
 'gmmne.jpg',
 '6mn8n.png',
 '243mm.png',
 'e8dxn.png',
 '88bgx.png',
 'm457d.png',
 '3xng6.png',
 '6end3.png',
 '65ebm.png',
 '6pfy4.png',
 '8b735.png',
 '34b84.png',
 'dn26n.png',
 '8n62n.png',
 '2pfpn.png',
 'fp382.png',
 '8n5pn.png',
 '57b27.png',
 'c482b.png',
 'pmd3w.png',
 '264m5.png',
 'ny8np.png',
 'nfcb5.png',
 'n3ffn.png',
 '25257.png',
 'p5g5m.png',
 ...]
```

```
#total no of images in dataset
n=len(os.listdir("/content/drive/My Drive/captcha_dataset/samples"))
n
```

```
1070
```

```
#defining size of image
imgshape=(50,200,1) #50-height, 200-width, 1-no of channels
```

```
character= string.ascii_lowercase + "0123456789" # All symbols captcha can contain
nchar = len(character) #total number of char possible
nchar
```

```
36
```

```
#preprocesss image
def preprocess():
  X = np.zeros((n,50,200,1)) #1070*50*200 array with all entries 0
  y = np.zeros((5,n,nchar)) #5*1070*36(5 letters in captcha) with all entries 0

  for i, pic in enumerate(os.listdir("/content/drive/My Drive/captcha_dataset/samp
  #i represents index no. of image in directory
  #pic contains the file name of the particular image to be preprocessed at a time

    img = cv2.imread(os.path.join("/content/drive/My Drive/captcha_dataset/samples
```

```
      pic_target = pic[:-4]#this drops the .png extension from file name and contains

      if len(pic_target) < 6: #captcha is not more than 5 letters
        img = img / 255.0 #scales the image between 0 and 1
        img = np.reshape(img, (50, 200, 1)) #reshapes image to width 200 , height 50

        target=np.zeros((5,nchar)) #creates an array of size 5*36 with all entries 0

        for j, k in enumerate(pic_target):
        #j iterates from 0 to 4(5 letters in captcha)
        #k denotes the letter in captcha which is to be scanned
           index = character.find(k) #index stores the position of letter k of captcl
           target[j, index] = 1 #replaces 0 with 1 in the target array at the positi

        X[i] = img #stores all the images
        y[:,i] = target #stores all the info about the letters in captcha of all imag

    return X,y

  #create model
  def createmodel():
      img = layers.Input(shape=imgshape) # Get image as an input of size 50,200,1
      conv1 = layers.Conv2D(16, (3, 3), padding='same', activation='relu')(img) #50*
      mp1 = layers.MaxPooling2D(padding='same')(conv1)  # 25*100
      conv2 = layers.Conv2D(32, (3, 3), padding='same', activation='relu')(mp1)
      mp2 = layers.MaxPooling2D(padding='same')(conv2)  # 13*50
      conv3 = layers.Conv2D(32, (3, 3), padding='same', activation='relu')(mp2)
      bn = layers.BatchNormalization()(conv3) #to improve the stability of model
      mp3 = layers.MaxPooling2D(padding='same')(bn)  # 7*25

      flat = layers.Flatten()(mp3) #convert the layer into 1-D

      outs = []
      for _ in range(5): #for 5 letters of captcha
          dens1 = layers.Dense(64, activation='relu')(flat)
          drop = layers.Dropout(0.5)(dens1) #drops 0.5 fraction of nodes
          res = layers.Dense(nchar, activation='sigmoid')(drop)

          outs.append(res) #result of layers

      # Compile model and return it
      model = Model(img, outs) #create model
      model.compile(loss='categorical_crossentropy', optimizer='adam',metrics=["accu
      return model


  #Create model
  model=createmodel();
  model.summary();
```

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #     Connected t
=================================================================
 input_1 (InputLayer)         [(None, 50, 200, 1)  0           []

                              ]
```

```
conv2d (Conv2D)                (None, 50, 200, 16)  160         ['input_1[0
max_pooling2d (MaxPooling2D)   (None, 25, 100, 16)  0           ['conv2d[0
conv2d_1 (Conv2D)              (None, 25, 100, 32)  4640        ['max_pool
max_pooling2d_1 (MaxPooling2D)  (None, 13, 50, 32)  0           ['conv2d_1
conv2d_2 (Conv2D)              (None, 13, 50, 32)   9248        ['max_pool
batch_normalization (BatchNorm  (None, 13, 50, 32)  128         ['conv2d_2
alization)
max_pooling2d_2 (MaxPooling2D)  (None, 7, 25, 32)   0           ['batch_no
flatten (Flatten)              (None, 5600)         0           ['max_pool
dense (Dense)                  (None, 64)           358464      ['flatten[0
dense_2 (Dense)                (None, 64)           358464      ['flatten[0
dense_4 (Dense)                (None, 64)           358464      ['flatten[0
dense_6 (Dense)                (None, 64)           358464      ['flatten[0
dense_8 (Dense)                (None, 64)           358464      ['flatten[0
dropout (Dropout)              (None, 64)           0           ['dense[0]
dropout_1 (Dropout)            (None, 64)           0           ['dense_2[0
dropout_2 (Dropout)            (None, 64)           0           ['dense_4[0
dropout_3 (Dropout)            (None, 64)           0           ['dense_6[0
dropout_4 (Dropout)            (None, 64)           0           ['dense_8[0
dense_1 (Dense)                (None, 36)           2340        ['dropout[0
dense_3 (Dense)                (None, 36)           2340        ['dropout_1
dense_5 (Dense)                (None, 36)           2340        ['dropout_2
dense_7 (Dense)                (None, 36)           2340        ['dropout_3
dense_9 (Dense)                (None, 36)           2340        ['dropout_4
==================================================================================
Total params: 1,818,196
Trainable params: 1,818,132
Non-trainable params: 64
__________________________________________________________________________________
```

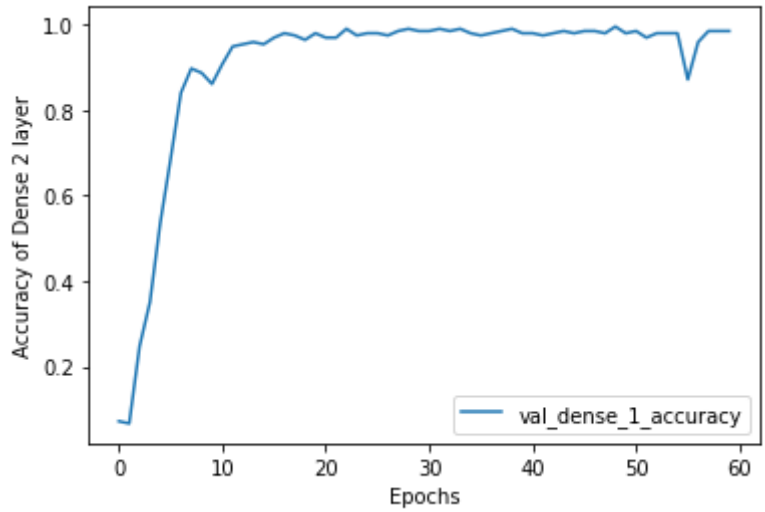
```
X,y=preprocess()


#split the 1070 samples where 970 samples will be used for training purpose
X_train, y_train = X[:970], y[:, :970]
X_test, y_test = X[970:], y[:, 970:]
```

```
#Applying the model
hist = model.fit(X_train, [y_train[0], y_train[1], y_train[2], y_train[3], y_train
#batch size- 32 defines no. of samples per gradient update
#Validation split=0.2 splits the training set in 80-20% for training nd testing
```

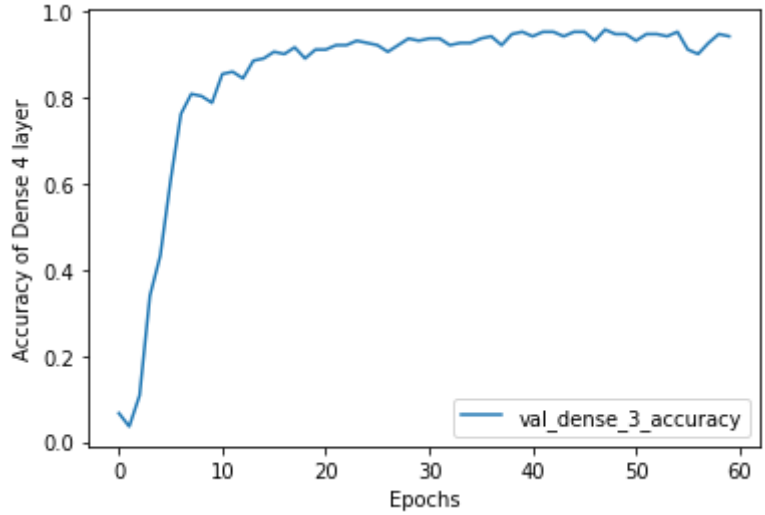
```
Epoch 32/60
25/25 [==============================] - 5s 200ms/step - loss: 1.9726 - dens
Epoch 33/60
25/25 [==============================] - 5s 200ms/step - loss: 1.9420 - dens
Epoch 34/60
25/25 [==============================] - 5s 200ms/step - loss: 1.9329 - dens
Epoch 35/60
25/25 [==============================] - 5s 200ms/step - loss: 1.8960 - dens
Epoch 36/60
25/25 [==============================] - 5s 200ms/step - loss: 1.7936 - dens
Epoch 37/60
25/25 [==============================] - 5s 200ms/step - loss: 1.7459 - dens
Epoch 38/60
25/25 [==============================] - 5s 200ms/step - loss: 1.7410 - dens
Epoch 39/60
25/25 [==============================] - 5s 200ms/step - loss: 1.7810 - dens
Epoch 40/60
25/25 [==============================] - 5s 200ms/step - loss: 1.6462 - dens
Epoch 41/60
25/25 [==============================] - 5s 200ms/step - loss: 1.6713 - dens
Epoch 42/60
25/25 [==============================] - 5s 199ms/step - loss: 1.5966 - dens
Epoch 43/60
25/25 [==============================] - 5s 201ms/step - loss: 1.7150 - dens
Epoch 44/60
25/25 [==============================] - 5s 199ms/step - loss: 1.6162 - dens
Epoch 45/60
25/25 [==============================] - 5s 199ms/step - loss: 1.5823 - dens
Epoch 46/60
25/25 [==============================] - 5s 200ms/step - loss: 1.5966 - dens
Epoch 47/60
25/25 [==============================] - 5s 201ms/step - loss: 1.5390 - dens
Epoch 48/60
25/25 [==============================] - 5s 199ms/step - loss: 1.5552 - dens
Epoch 49/60
25/25 [==============================] - 5s 200ms/step - loss: 1.4261 - dens
Epoch 50/60
25/25 [==============================] - 5s 199ms/step - loss: 1.4741 - dens
Epoch 51/60
25/25 [==============================] - 5s 199ms/step - loss: 1.3588 - dens
Epoch 52/60
25/25 [==============================] - 5s 200ms/step - loss: 1.4019 - dens
Epoch 53/60
25/25 [==============================] - 5s 200ms/step - loss: 1.3805 - dens
Epoch 54/60
25/25 [==============================] - 5s 199ms/step - loss: 1.3815 - dens
Epoch 55/60
25/25 [==============================] - 5s 199ms/step - loss: 1.3844 - dens
Epoch 56/60
25/25 [==============================] - 5s 200ms/step - loss: 1.4518 - dens
Epoch 57/60
25/25 [==============================] - 5s 201ms/step - loss: 1.2935 - dens
Epoch 58/60
25/25 [==============================] - 5s 201ms/step - loss: 1.3622 - dens
Epoch 59/60
```

```
Epoch 59/60
25/25 [==============================] - 5s 201ms/step - loss: 1.2814 - dens
Epoch 60/60
25/25 [==============================] - 5s 201ms/step - loss: 1.2899 - dens
```

```
#graph of accuracy of dense_2 vs epochs
for label in ["val_dense_1_accuracy"]:
 plt.plot(hist.history[label],label=label)
plt.legend()
plt.xlabel("Epochs")
plt.ylabel("Accuracy of Dense 2 layer")
plt.show()
```

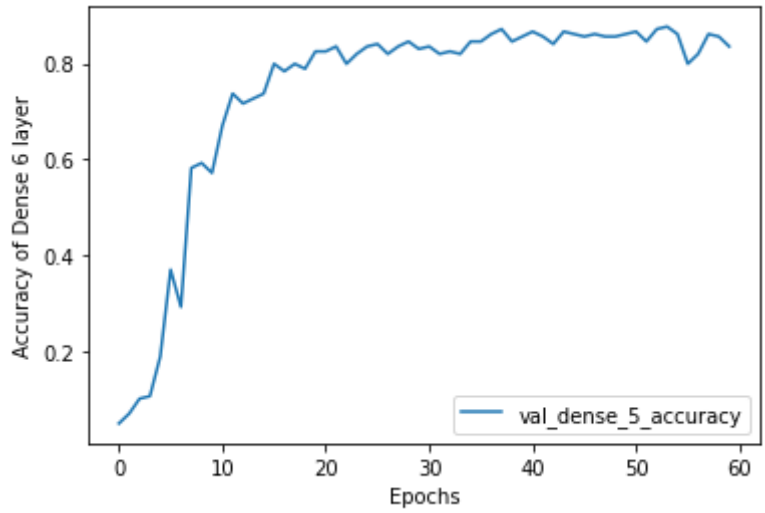



```
#graph of accuracy of dense_4 vs epochs
for label in ["val_dense_3_accuracy"]:
  plt.plot(hist.history[label],label=label)
plt.legend()
plt.xlabel("Epochs")
plt.ylabel("Accuracy of Dense 4 layer")
plt.show()
```

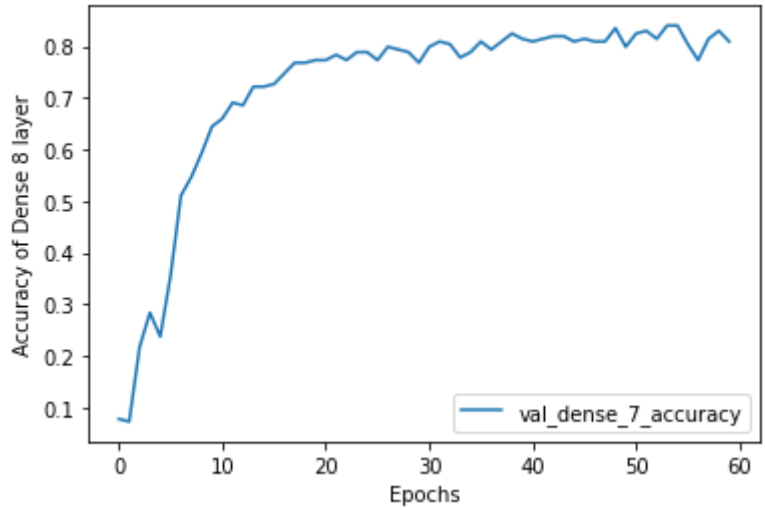



```
#graph of accuracy of dense_6 vs epochs
for label in ["val_dense_5_accuracy"]:
```

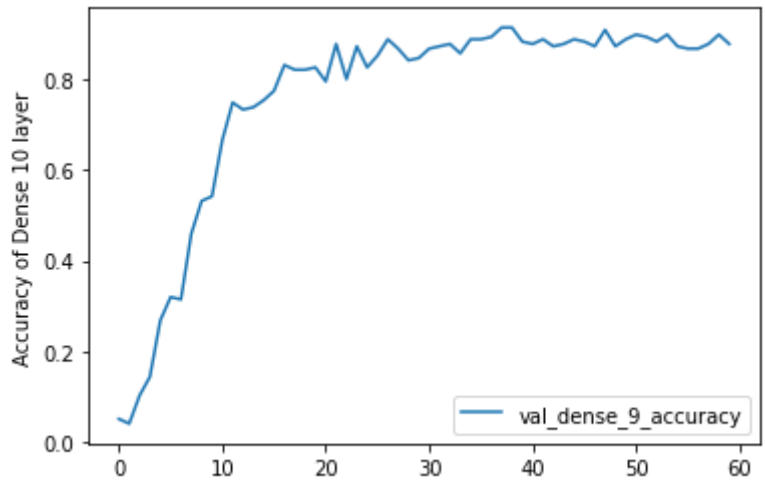
```
  plt.plot(hist.history[label],label=label)
plt.legend()
plt.xlabel("Epochs")
plt.ylabel("Accuracy of Dense 6 layer")
plt.show()
```




```
#graph of accuracy of dense_8 vs epochs
for label in ["val_dense_7_accuracy"]:
  plt.plot(hist.history[label],label=label)
plt.legend()
plt.xlabel("Epochs")
plt.ylabel("Accuracy of Dense 8 layer")
plt.show()
```




```
#graph of accuracy of dense_10 vs epochs
for label in ["val_dense_9_accuracy"]:
  plt.plot(hist.history[label],label=label)
plt.legend()
plt.xlabel("Epochs")
plt.ylabel("Accuracy of Dense 10 layer")
plt.show()
```




```
#Loss on training set
#Finding Loss on training set
preds = model.evaluate(X_train, [y_train[0], y_train[1], y_train[2], y_train[3], y
print ("Loss on training set= " + str(preds[0]))
```

```
31/31 [==============================] - 2s 58ms/step - loss: 0.5112 - dense_
Loss on training set= 0.5112058520317078
```

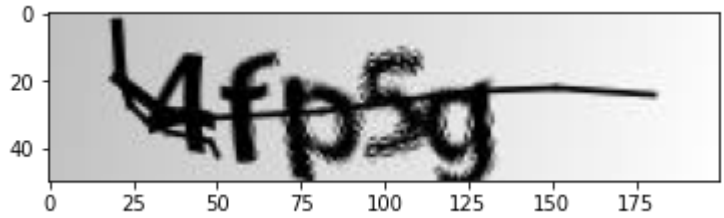
```
#Finding loss on test set
preds = model.evaluate(X_test, [y_test[0], y_test[1], y_test[2], y_test[3], y_test
print ("Loss on testing set= " + str(preds[0]))
```

```
4/4 [==============================] - 0s 43ms/step - loss: 1.9047 - dense_1_
Loss on testing set= 1.9046618938446045
```

```
#to predict captcha
def predict(filepath):
    img = cv2.imread(filepath, cv2.IMREAD_GRAYSCALE)

    if img is not None: #image foud at file path
        img = img / 255.0 #Scale image
    else:
        print("Not detected");

    res = np.array(model.predict(img[np.newaxis, :, :, np.newaxis])) #np.newaxis=1
    #added this bcoz x_train 970*50*200*1
    #returns array of size 1*5*36
    result = np.reshape(res, (5, 36)) #reshape the array
    k_ind = []
    probs = []
    for i in result:
        k_ind.append(np.argmax(i)) #adds the index of the char found in captcha

    capt = '' #string to store predicted captcha
    for k in k_ind:
        capt += character[k] #finds the char corresponding to the index
    return capt
```

```
#Check model on samples
img=cv2.imread('/content/drive/My Drive/captcha_dataset/4fp5g.png',cv2.IMREAD_GRAYS
plt.imshow(img, cmap=plt.get_cmap('gray'))
```

```
<matplotlib.image.AxesImage at 0x7f6c80e0fc10>
```

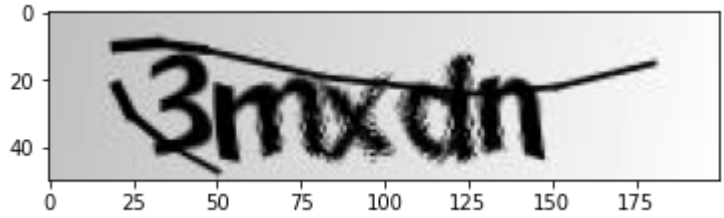



```
print("Predicted Captcha =",predict('/content/drive/My Drive/captcha_dataset/4fp5g
```

```
Predicted Captcha = 4fp5g
```

```
#Sample 2
img=cv2.imread('/content/drive/My Drive/captcha_dataset/3mxdn.png',cv2.IMREAD_GRAYS
plt.imshow(img, cmap=plt.get_cmap('gray'))
```

```
<matplotlib.image.AxesImage at 0x7f6c81f00050>
```




```
print("Predicted Captcha =",predict('/content/drive/My Drive/captcha_dataset/3mxdn
```

```
Predicted Captcha = 3mxdn
```

✓ 0s completed at 3:46 PM