

# What is Colaboratory?

Colaboratory, or 'Colab' for short, allows you to write and execute Python in your browser, with

- Zero configuration required
- Free access to GPUs
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch [Introduction to Colab](#) to find out more, or just get started below!

## ▼ Getting started

The document that you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute code.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable and prints the result:

```
seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day

86400
```

To execute the code in the above cell, select it with a click and then either press the play button to the left of the code, or use the keyboard shortcut 'Command/Ctrl+Enter'. To edit the code, just click the cell and start editing.

Variables that you define in one cell can later be used in other cells:

```
seconds_in_a_week = 7 * seconds_in_a_day
seconds_in_a_week

604800
```

Colab notebooks allow you to combine **executable code** and **rich text** in a single document, along with **images**, **HTML**, **LaTeX** and more. When you create your own Colab notebooks, they are stored in your Google Drive account. You can easily share your Colab notebooks with co-workers or friends, allowing them to comment on your notebooks or even edit them. To find out more, see [Overview of Colab](#). To create a new Colab notebook you can use the File menu above, or use the following link: [Create a new Colab notebook](#).

Colab notebooks are Jupyter notebooks that are hosted by Colab. To find out more about the Jupyter project, see [jupyter.org](https://jupyter.org).

## ▼ Data science

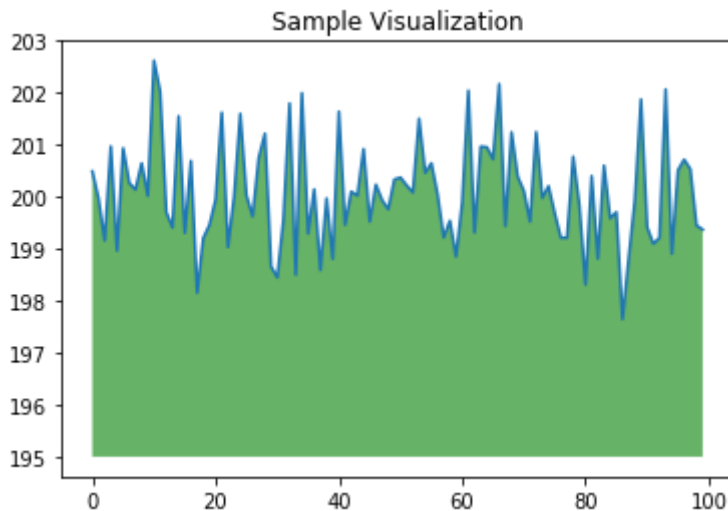
With Colab you can harness the full power of popular Python libraries to analyse and visualise data. The code cell below uses **numpy** to generate some random data, and uses **matplotlib** to visualise it. To edit the code, just click the cell and start editing.

```
import numpy as np
from matplotlib import pyplot as plt

ys = 200 + np.random.randn(100)
x = [x for x in range(len(ys))]

plt.plot(x, ys, '-')
plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g', alpha=0.6)

plt.title("Sample Visualization")
plt.show()
```



You can import your own data into Colab notebooks from your Google Drive account, including from spreadsheets, as well as from GitHub and many other sources. To find out more about importing data, and how Colab can be used for data science, see the links below under [Working with data](#).

## ▼ Machine learning

With Colab you can import an image dataset, train an image classifier on it, and evaluate the model, all in just [a few lines of code](#). Colab notebooks execute code on Google's cloud servers, meaning you can leverage the power of Google hardware, including [GPUs and TPUs](#), regardless of the power of your machine. All you need is a browser.


Colab is used extensively in the machine learning community with applications including:

- Getting started with TensorFlow
- Developing and training neural networks
- Experimenting with TPUs
- Disseminating AI research
- Creating tutorials

To see sample Colab notebooks that demonstrate machine learning applications, see the [machine learning examples](#) below.

## More resources

### Working with notebooks in Colab

- [Overview of Colaboratory](#)
- [Guide to markdown](#)
- [Importing libraries and installing dependencies](#)
- [Saving and loading notebooks in GitHub](#)
- [Interactive forms](#)
- [Interactive widgets](#)
-  [TensorFlow 2 in Colab](#)

### Working with data

- [Loading data: Drive, Sheets and Google Cloud Storage](#)
- [Charts: visualising data](#)
- [Getting started with BigQuery](#)

### Machine learning crash course

These are a few of the notebooks from Google's online machine learning course. See the [full course website](#) for more.

- [Intro to Pandas DataFrame](#)
- [Linear regression with tf.keras using synthetic data](#)

### Using accelerated hardware

- [TensorFlow with GPUs](#)
- [TensorFlow with TPUs](#)

## ▼ Machine learning examples

To see end-to-end examples of the interactive machine-learning analyses that Colaboratory makes possible, take a look at these tutorials using models from [TensorFlow Hub](#).

A few featured examples:

- [Retraining an Image Classifier](#): Build a Keras model on top of a pre-trained image classifier to distinguish flowers.
- [Text Classification](#): Classify IMDB film reviews as either *positive* or *negative*.
- [Style Transfer](#): Use deep learning to transfer style between images.
- [Multilingual Universal Sentence Encoder Q&A](#): Use a machine-learning model to answer questions from the SQuAD dataset.
- [Video Interpolation](#): Predict what happened in a video between the first and the last frame.

```
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 functionality
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/samples")
```

```

xmx5.png ,
'58pnp.png',
'dbfen.png',
'ec6pm.png',
'52447.png',
'nbnwn.png',
'4dw3w.png',
'5n728.png',
'6wg4n.png',
'm23bp.png',
'6fg8c.png',
'574d7.png',
'ddmyg.png',
'y3m3.png',
'x38fn.png',
'f5cm2.png',
'y5dpp.png',
'8fexn.png',
'w4cnn.png',
'vdd3a.png'

```

```

'33n73.png',
'pmf5w.png',
'pm47f.png',
'xnd3y.png',
'gwn53.png',
'8c2wy.png',
'mdyp7.png',
'3den6.png',
'5ywwf.png',
'5325m.png',
'pcm7f.png',
'2mpnn.png',
'en4n4.png',
'excmn.png',
'fgb36.jpg',
'e2d66.png',
'662bw.png',
'g2fnw.png',
'6n5fd.png',
'pdc4.png',
'by5y3.png',
'ywn6f.png',
'cwmny.png',
'nbp3e.png',
'ncfgeb.png',
'y3c58.png',
'mmc5n.png',
'm67b3.png',
'2enf4.png',
'nxn4f.png',
'n7ff2.png',
'ddxpp.jpg',
'w52fn.png',
'mcyfx.png',
'efx34.png',
'xngxc.png',
'b5nmm.png',
'664dn.png',
'p4pde.png',

```

```

#total no of images in dataset
n=len(os.listdir("/content/drive/My Drive/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/samples")):
        #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/samples", pic), cv2.IMR
        pic_target = pic[:-4]#this drops the .png extension from file name and contain

        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 captc
                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 ima

    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=create_model();
model.summary();
```

Layer (type)	Output Shape	Param #	Connected
input_2 (InputLayer)	[(None, 50, 200, 1)]	0	[]
conv2d_3 (Conv2D)	(None, 50, 200, 16)	160	['input_2[
max_pooling2d_3 (MaxPooling2D)	(None, 25, 100, 16)	0	['conv2d_3
conv2d_4 (Conv2D)	(None, 25, 100, 32)	4640	['max_pool
max_pooling2d_4 (MaxPooling2D)	(None, 13, 50, 32)	0	['conv2d_4
conv2d_5 (Conv2D)	(None, 13, 50, 32)	9248	['max_pool
batch_normalization_1 (Batch Normalization)	(None, 13, 50, 32)	128	['conv2d_5
max_pooling2d_5 (MaxPooling2D)	(None, 7, 25, 32)	0	['batch_no
flatten_1 (Flatten)	(None, 5600)	0	['max_pool
dense_10 (Dense)	(None, 64)	358464	['flatten_
dense_12 (Dense)	(None, 64)	358464	['flatten_
dense_14 (Dense)	(None, 64)	358464	['flatten_
dense_16 (Dense)	(None, 64)	358464	['flatten_
dense_18 (Dense)	(None, 64)	358464	['flatten_
dropout_5 (Dropout)	(None, 64)	0	['dense_10
dropout_6 (Dropout)	(None, 64)	0	['dense_12
dropout_7 (Dropout)	(None, 64)	0	['dense_14
dropout_8 (Dropout)	(None, 64)	0	['dense_16
dropout_9 (Dropout)	(None, 64)	0	['dense_18
dense_11 (Dense)	(None, 36)	2340	['dropout_
dense_13 (Dense)	(None, 36)	2340	['dropout_
dense_15 (Dense)	(None, 36)	2340	['dropout_
dense_17 (Dense)	(None, 36)	2340	['dropout_
dense_19 (Dense)	(None, 36)	2340	['dropout_

=====  
Total params: 1,818,196

Trainable params: 1,818,122

```
trainable params: 1,010,152
```

```
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
```

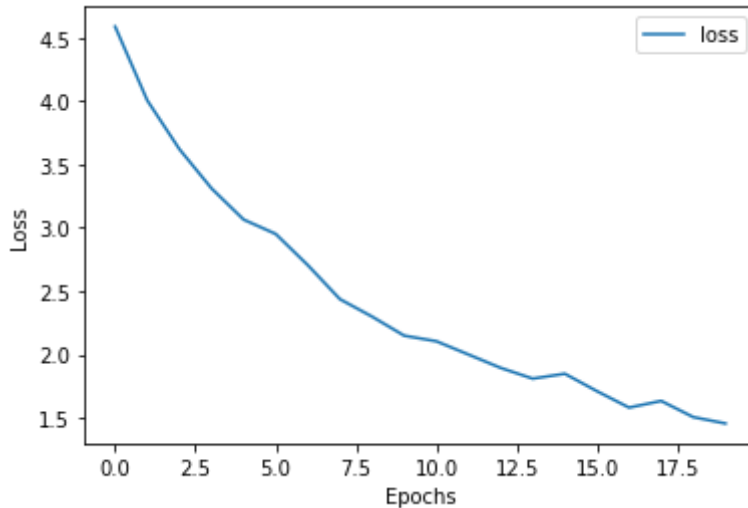
```
#batch size- 32 defines no. of samples per gradient update
```

```
#Validation split=0.2 splits the training set in 80-20% for training and testing
```

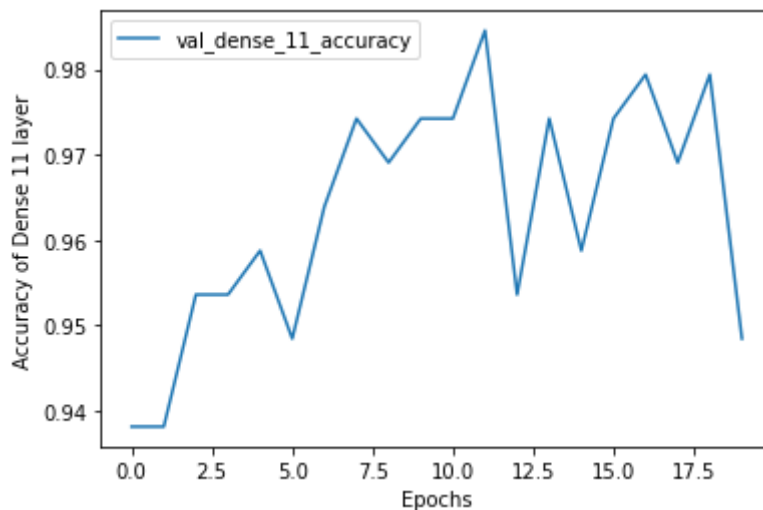
```
1.2705 - dense_19_loss: 1.1971 - dense_11_accuracy: 0.8286 - dense_13_accu
1.1305 - dense_19_loss: 1.0531 - dense_11_accuracy: 0.8621 - dense_13_accu
1.0125 - dense_19_loss: 0.9639 - dense_11_accuracy: 0.8892 - dense_13_accu
0.9107 - dense_19_loss: 0.8446 - dense_11_accuracy: 0.8943 - dense_13_accu
0.8372 - dense_19_loss: 0.7895 - dense_11_accuracy: 0.8982 - dense_13_accu
0.8195 - dense_19_loss: 0.7825 - dense_11_accuracy: 0.8750 - dense_13_accu
0.7343 - dense_19_loss: 0.7070 - dense_11_accuracy: 0.8995 - dense_13_accu
0.7015 - dense_19_loss: 0.6466 - dense_11_accuracy: 0.9162 - dense_13_accu
0.5717 - dense_19_loss: 0.6355 - dense_11_accuracy: 0.9162 - dense_13_accu
0.5774 - dense_19_loss: 0.5417 - dense_11_accuracy: 0.9072 - dense_13_accu
0.5581 - dense_19_loss: 0.6351 - dense_11_accuracy: 0.9253 - dense_13_accu
0.5482 - dense_19_loss: 0.5627 - dense_11_accuracy: 0.9162 - dense_13_accu
0.5314 - dense_19_loss: 0.5428 - dense_11_accuracy: 0.9137 - dense_13_accu
0.4602 - dense_19_loss: 0.5140 - dense_11_accuracy: 0.9407 - dense_13_accu
0.4539 - dense_19_loss: 0.5031 - dense_11_accuracy: 0.9253 - dense_13_accu
0.4342 - dense_19_loss: 0.4696 - dense_11_accuracy: 0.9433 - dense_13_accu
0.4060 - dense_19_loss: 0.4512 - dense_11_accuracy: 0.9446 - dense_13_accu
0.4383 - dense_19_loss: 0.4408 - dense_11_accuracy: 0.9240 - dense_13_accu
0.4451 - dense_19_loss: 0.4075 - dense_11_accuracy: 0.9485 - dense_13_accu
0.3888 - dense_19_loss: 0.3956 - dense_11_accuracy: 0.9472 - dense_13_accu
```



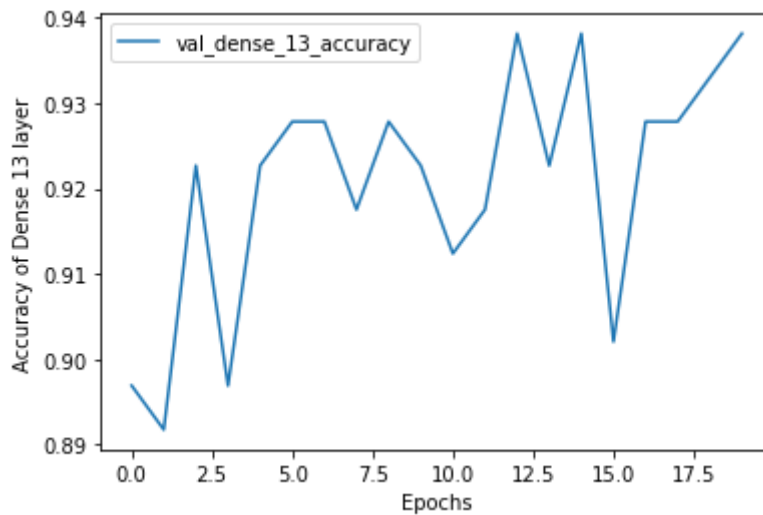
```
#graph of loss vs epochs
for label in ["loss"]:
    plt.plot(hist.history[label],label=label)
plt.legend()
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()
```



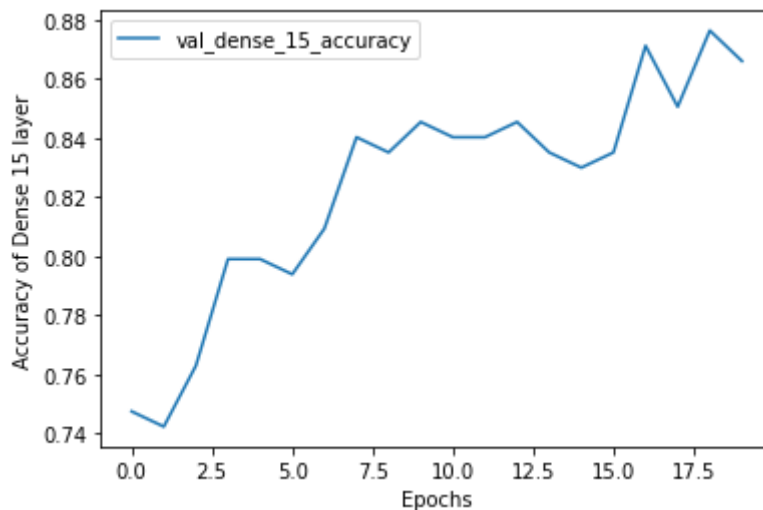
```
#graph of accuracy of dense_2 vs epochs
for label in ["val_dense_11_accuracy"]:
    plt.plot(hist.history[label],label=label)
plt.legend()
plt.xlabel("Epochs")
plt.ylabel("Accuracy of Dense 11 layer")
plt.show()
```



```
#graph of accuracy of dense_4 vs epochs
for label in ["val_dense_13_accuracy"]:
    plt.plot(hist.history[label],label=label)
plt.legend()
plt.xlabel("Epochs")
plt.ylabel("Accuracy of Dense 13 layer")
plt.show()
```



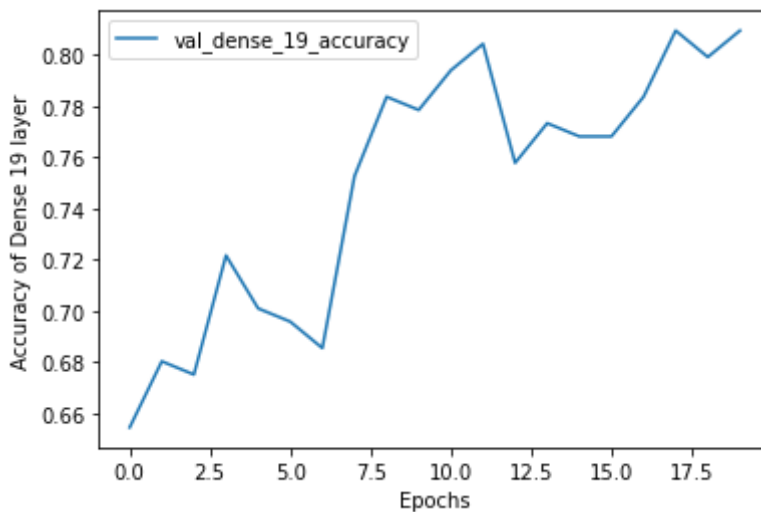
```
#graph of accuracy of dense_6 vs epochs
for label in ["val_dense_15_accuracy"]:
    plt.plot(hist.history[label],label=label)
plt.legend()
plt.xlabel("Epochs")
plt.ylabel("Accuracy of Dense 15 layer")
plt.show()
```



```
#graph of accuracy of dense_8 vs epochs
for label in ["val_dense_17_accuracy"]:
    plt.plot(hist.history[label],label=label)
plt.legend()
plt.xlabel("Epochs")
plt.ylabel("Accuracy of Dense 17 layer")
plt.show()
```



```
#graph of accuracy of dense_10 vs epochs
for label in ["val_dense_19_accuracy"]:
    plt.plot(hist.history[label],label=label)
plt.legend()
plt.xlabel("Epochs")
plt.ylabel("Accuracy of Dense 19 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 78ms/step - loss: 0.5599 - dense_
Loss on training set= 0.5598708391189575
```

```
#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 56ms/step - loss: 3.0772 - dense_11
Loss on testing set= 3.0772125720977783
```

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

    if img is not None: #image found at file path
        img = img / 255.0 #Scale image
    else:
```

```
print("Not detected");
```

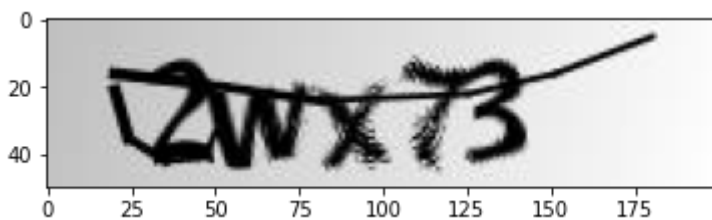
```
res = np.array(model.predict(img[np.newaxis, :, :, np.newaxis])) #np.newaxis=1
#added this bcoz x_train 970*50*200*1
#returns array of size 1*5*36
result = np.reshape(res, (5, 36)) #reshape the array
k_ind = []
probs = []
for i in result:
    k_ind.append(np.argmax(i)) #adds the index of the char 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/sample/2wx73.png',cv2.IMREAD_GRAYSCALE)
plt.imshow(img, cmap=plt.get_cmap('gray'))
```

```
<matplotlib.image.AxesImage at 0x7fb1bf054e10>
```



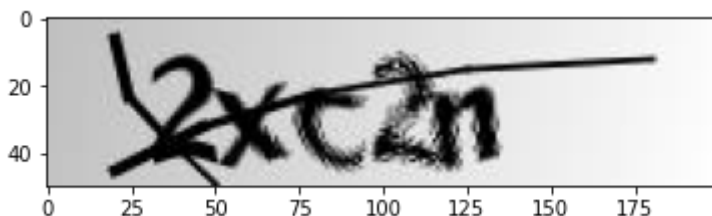
```
print("Predicted Captcha =",predict('/content/drive/My Drive/sample/2wx73.png'))
```

```
Predicted Captcha = 2wx73
```

```
#Sample 2
```

```
img=cv2.imread('/content/drive/My Drive/sample/2xc2n.png',cv2.IMREAD_GRAYSCALE)
plt.imshow(img, cmap=plt.get_cmap('gray'))
```

```
<matplotlib.image.AxesImage at 0x7fb1befd0810>
```



```
print("Predicted Captcha =",predict('/content/drive/My Drive/sample/2xc2n.png'))
```

```
Predicted Captcha = 2xc2n
```

---

✓ 0s completed at 11:53

● ✕