

Project Report

Arush Verma – Neural Networks and guessing drawings

June 2019 - October 2019

Summary

One of the practical ventures that I have started and want to continue working on is an image recognition system. This project requires Python and Google Collab as the primary language and IDU, respectively. Using the knowledge that I learnt from a summer course in New York University (Summer 2019), I began a project similar to Google's sketch.io.

The project implements machine learning to train an algorithm (AdamOptimizer) to recognize different images. Using Google's data set, I trained the algorithm to recognize images of certain objects like a bee or a crocodile. Next, I drew some images of crocodiles and bees and uploaded to a drive, which I connected to the IDU. By using the information from the summer course, I attained 97% accuracy in a program that can guess what image is being shown to it. The application for this already exists, but it depends on users drawing something on the laptop.

The purpose of my project is to help students learning art at a very young age to draw. It acts like a test of sorts to see whether the student is able to draw accurately enough that the program can guess what has been drawn.

The most challenging part of this project will be implementing a GUI that allows small children to press a button and upload a photo of their drawing to the drive connected to the program and receive a result in that same GUI.

Code

The quickdraw dataset from google is available online for anyone to access. For my project, I chose 10 images that were easy for me to draw. Those images will be shown in the code below


```

bee = np.load('/content/drive/My Drive/full_numpy_bitmap_bee.npy').reshape(120890,28,28,1)
crocodile = np.load('/content/drive/My Drive/full_numpy_bitmap_crocodile.npy').reshape(127932,28,28,1)
dragon = np.load('/content/drive/My Drive/full_numpy_bitmap_dragon.npy').reshape(124362,28,28,1)
eye = np.load('/content/drive/My Drive/full_numpy_bitmap_eye.npy').reshape(125888,28,28,1)
fish = np.load('/content/drive/My Drive/full_numpy_bitmap_fish.npy').reshape(134150,28,28,1)
flamingo = np.load('/content/drive/My Drive/full_numpy_bitmap_flamingo.npy').reshape(124569,28,28,1)
hedgehog = np.load('/content/drive/My Drive/full_numpy_bitmap_hedgehog.npy').reshape(120527,28,28,1)
hotdog = np.load('/content/drive/My Drive/full_numpy_bitmap_hot_dog.npy').reshape(181999,28,28,1)
octopus = np.load('/content/drive/My Drive/full_numpy_bitmap_octopus.npy').reshape(150152,28,28,1)
telephone = np.load('/content/drive/My Drive/full_numpy_bitmap_telephone.npy').reshape(127885,28,28,1)

results = ['bee',
           'crocodile',
           'dragon',
           'eye',
           'fish',
           'flamingo',
           'hedgehog',
           'hotdog',
           'octopus',
           'telephone']

classes = 10
samples = 40000

bee = bee[0:samples]
crocodile = crocodile[0:samples]
dragon = dragon[0:samples]
eye = eye[0:samples]
fish = fish[0:samples]
flamingo = flamingo[0:samples]
hedgehog = hedgehog[0:samples]
hotdog = hotdog[0:samples]
octopus = octopus[0:samples]
telephone = telephone[0:samples]

X = np.concatenate((bee,
                    crocodile,
                    dragon,
                    eye,
                    fish,
                    flamingo,
                    hedgehog,
                    hotdog,
                    octopus,
                    telephone),
                  axis = 0)

print(X.shape)
#X = X.reshape(50000,28,28,1)
X = X/255
Y = np.empty((samples * classes, 1))

for i in range(0, classes):
    Y[i * samples : samples * (i + 1)] = i
print(Y.shape)

```

I split our dataset into a training set and a test set. The training set will be used to train the algorithm to recognize and label the different images while the test data determines the guessing accuracy.

In brief:

Here, the different layers are used in order to train the algorithm. An outline is that I alternated between a convolutional layer and a pooling layer to constantly resize the dataset. By doing this, I am allowing the algorithm to gain a deeper understanding of the training set. The `keras.models.Sequential()` is called through one of the imported libraries.

For the final step, I used Dense layers. The numbers signify the dimensionality of the output shape, and relu is the activation function (more information about these layers can be found at <https://keras.io/layers/core/>)

```

(400000, 28, 28, 1)
(400000, 1)

[ ] 1 from sklearn.model_selection import train_test_split
    2 xtr , xts , ytr, yts = train_test_split(X,Y, test_size = 0.2, random_state = 42)
    3 print(xts.shape)

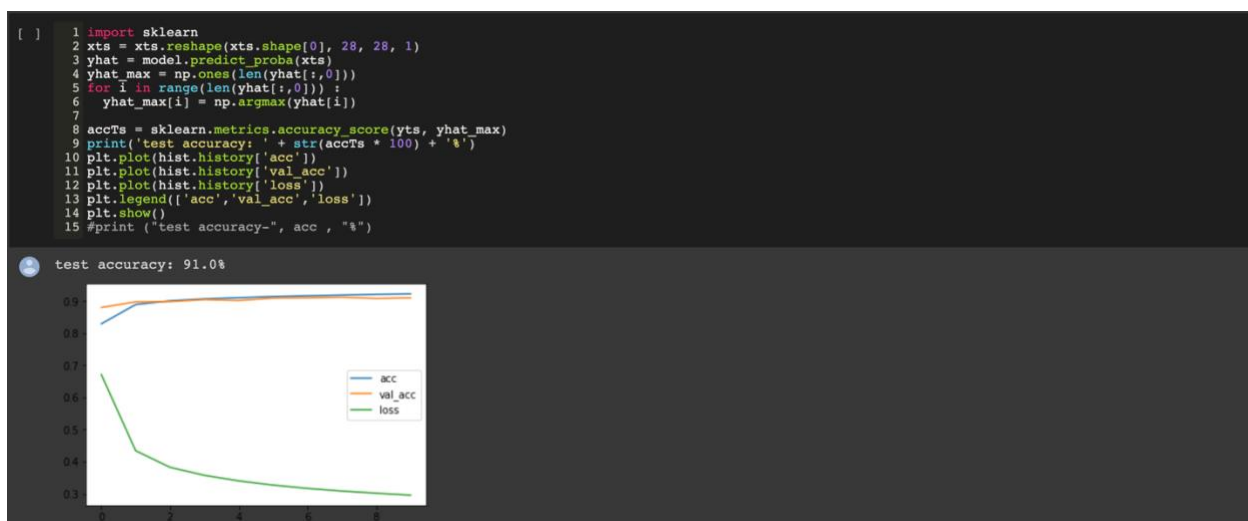
(80000, 28, 28, 1)

[ ] 1 K.clear_session()
    2 lr=0.001
    3 model = tf.keras.models.Sequential()
    4 model.add(Conv2D(64, (5,5), activation = 'relu', padding = 'valid', input_shape = (28,28,1)))
    5 model.add(MaxPooling2D((2,2)))
    6 model.add(Conv2D(32, (5,5), activation = 'relu', padding = 'valid'))
    7 model.add(MaxPooling2D((2,2)))
    8 model.add(Flatten())
    9 model.add(Dense(120,activation = 'relu', kernel_regularizer = l2(1)))
    10 model.add(Dense(84,activation = 'relu', kernel_regularizer = l2(1)))
    11 model.add(Dense(classes,activation = 'softmax'))
    12 opt = tf.keras.optimizers.Adam(lr = 0.001)
    13 model.compile(optimizer = opt, loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])
    14 hist = model.fit(xtr,ytr,validation_split = 0.2, batch_size = 100, epochs = 10)

Train on 256000 samples, validate on 64000 samples
Epoch 1/10
256000/256000 [=====] - 14s 55us/sample - loss: 0.6716 - acc: 0.8295 - val_loss: 0.4814 - val_acc: 0.8805
Epoch 2/10
256000/256000 [=====] - 14s 53us/sample - loss: 0.4341 - acc: 0.8897 - val_loss: 0.4026 - val_acc: 0.8980
Epoch 3/10
256000/256000 [=====] - 14s 53us/sample - loss: 0.3824 - acc: 0.9013 - val_loss: 0.3869 - val_acc: 0.8988
Epoch 4/10
256000/256000 [=====] - 13s 53us/sample - loss: 0.3574 - acc: 0.9073 - val_loss: 0.3647 - val_acc: 0.9051

```

Here, the training happens. The *AdamOptimizer* is used in order to train the algorithm, and I do not yet fully understand how the optimizer works, since it is irrelevant at this stage. After instantiating the optimizer, I decide how the data will be trained. 320,000 samples can be used to train the data, and this is divided into 256,000 samples to train on, and 64,000 samples to validate on. The *batch size* is 100, which means the algorithm takes the first 100 samples of the dataset and trains the neural network, then the next 100 and so on until 256,000 samples have been used. This will repeat 10 times (10 epochs), in order to improve the accuracy of the network. The way I decide to number the *batch size* and epochs are purely through trial and error, and constant guessing to see which gives me the highest accuracy.



The sklearn library is used, which allows me to plot the progression of the algorithm. As epochs increase (X axis) the different values vary. Loss decreases, and *val Acc* and *acc* increase to 91% meaning that my program has a potential margin of error of only 9%.

This block of code is used so that the program can retrieve the files uploaded to the drive by me. These are the photos that I draw by hand and upload to the drive, and using a loop the program retrieves all the *.png* images. Every time I run the program a random image from the ones I draw is taken and the program attempts to guess it.

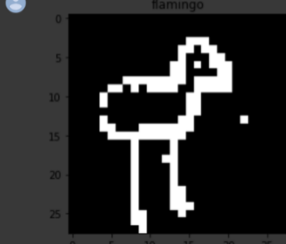
```
[ ] 1 from google.colab import drive
    2 drive.mount('/content/drive',force_remount = True)

Mounted at /content/drive

[ ] 1 import skimage
    2 inputSamples = 10
    3 inputSample = 6
    4 testSample = 3
    5 inputs = [0]* inputSamples
    6 inputdr = [0]* inputSample
    7 inputtest = [0]* inputSample
    8 i = 0
    9 j = 0
   10
   11 for i in range (0,inputSamples):
   12     j = i+1
   13     inputs[i] = skimage.io.imread('/content/drive/My Drive/input/img' + str(j) + '.png')
   14     k = 0
   15     for k in range (0,inputSample):
   16         z = k+1
   17         if z == 2:
   18             z = z+1
   19         inputdr[k] = skimage.io.imread('/content/drive/My Drive/input/papimg' + str(z) + '.png')
   20     for q in range (0,inputSample):
   21         n = q+1
   22         inputtest[q] = skimage.io.imread('/content/drive/My Drive/input/testimg' + str(n) + '.png')
   23
```

The image is then displayed on the screen using a series of commands and then reshaped to fit the screen.

```
[ ] 1 def test(img):
    2     img = img[:, :, 0]
    3     img = img.reshape(1,28,28,1)
    4     yhat = model.predict_proba(img)
    5     img = img.reshape(28,28)
    6     plt.title(results[np.argmax(yhat)])
    7     plt.imshow(img,'gray')
    8     plt.show()
    9
   10 def testdraw(img):
   11     img = img[:, :, 0]
   12     img = img.reshape(1,28,28,1)
   13     img = img/255
   14     img[img>.6] =1
   15     img[img<.7] =0
   16     img = 1-img
   17     yhat = model.predict_proba(img)
   18     img = img.reshape(28,28)
   19     plt.title(results[np.argmax(yhat)])
   20     plt.imshow(img, 'gray')
   21
   22 for cim in range(len(inputtest)) :
   23     test(compInputs[cim])
```



This is the result of the program, guessed by the network. As is visible, the program picked up an image that I drew of a flamingo and it was successfully able to guess it using the data given to it from the training set.

Next Steps

This project has a lot of possible uses. One of the things I learned from this project was denoising images to display them on the GUI, and this in itself could be an interesting project. By training an algorithm with images that are noisy and then denoised, I could create a network that can denoise any image given to it. This image guessing program can be used in a variety of ways, from teaching kids drawing to a game similar to Google's 'sketch.io'. In many ways, the possibilities are endless, and the opportunities provided by this project cannot be taken for granted.

I wish to continue working on this project in my own time and believe that I could turn it into something far more productive as I learn more things about neural networks. A lot of the code used in this project was 'black-boxed' since I am not advanced enough to understand how every little thing works (such as the AdamOptimizer). For example, in the future, rather than just know that I should use pooling and convolutional layers to train a network, I would want to learn more about exactly how using these layers actually train a program.

Overall, this process was quite intriguing and difficult since it was a brand new skill, but there is still a lot of work to be done if I want to turn this project into something viable from an end-user deployment standpoint.