Computer Vision – Final Project

By: Adi Levy, 037336146

Preface And Motivation

The Aim of the project is to test our ability to provide a winning solution to the image classification problem.

The classification is of ABC letters and words, scattered in images, with different fonts, where the aim of the classification is to find the font that a word is written at.

The image DB is generated by SynthText.

Running the Project

Project files are located here:

https://drive.google.com/drive/folders/1o-Wc1i5LDThAgSVx561wbCJMuPoQIWIK?usp=sharing

Running in Google Colab

1. Open the fontClass.ipynb file in google colab
2. Configure path of all the other py files in line 2:

```
sys.path.append('/content/drive/MyDrive/Colab Notebooks')
```

3. In the following line configure input h5 test file and output csv file

```
#args = parser.parse_args(args = ["-p", "-i", "/content/drive/MyDrive/Colab
Notebooks/SynthText_test.h5", "-o", "/content/drive/MyDrive/Colab
Notebooks/result.csv"])
```

4. Run the code

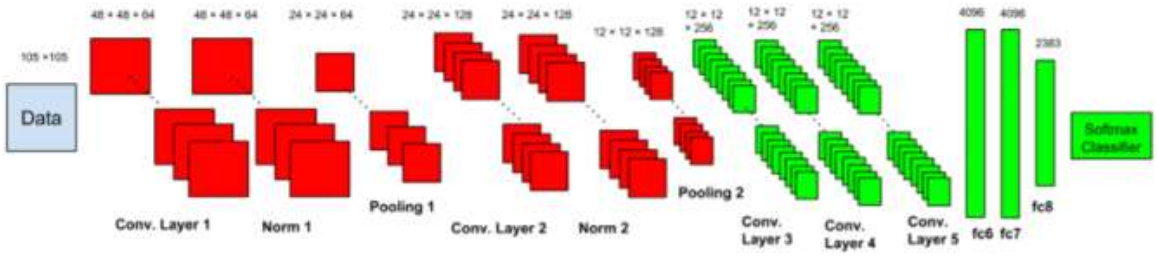
Running on PC

1. Make sure WSL (For Windows Machines), CUDA, and Nvidia drivers are installed.
2. From WSL or other linux console:
    a. `conda activate tf`
    b. `pip install -r requirements`
    c. `python fontClass.py -p -i <path>/SynthText_test.h5 -o <path>/result.csv`

Exploration for Solution

The first thing I thought is that someone has already done that, So I searched the web for similar things like google lens or an OCR.

I came across a research done by adobe, called "DeepFont[1]".
The research consists of a model, which achieves an accuracy of more than 80%.

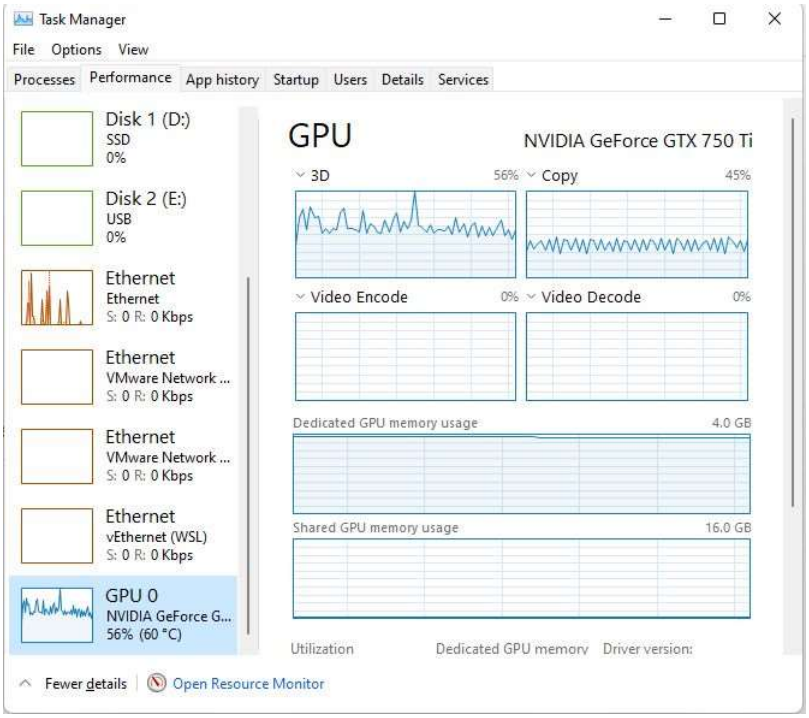


To my understanding, the model classifies font in words. Theoretically, fits the requirements of the project.

After implementing the model, I have discovered that:

1. The model is very heavy for a PC (With a GPU) to run on.
2. The model is very slow – Even on google colab.
3. The model didn't go over 30% accuracy after 25 epochs on the data that was provided.

[1] https://research.adobe.com/publication/deepfont-identify-your-font-from-an-image/, 2015



Then, I decided to try other solutions, such as training a model based on VGG16, with couple of changes:

1. Increase amount of data that feeds the training of the model – That is train using letters, rather than words.
2. The above decision led to a smaller input size (32 x 32, compared to 105 x 105 in the previous trial), and thus a smaller filters and faster training time (That event fits my PC).
3. After 10 epochs, The model achieved an accuracy of 74%.

Model Activation

Augmentation

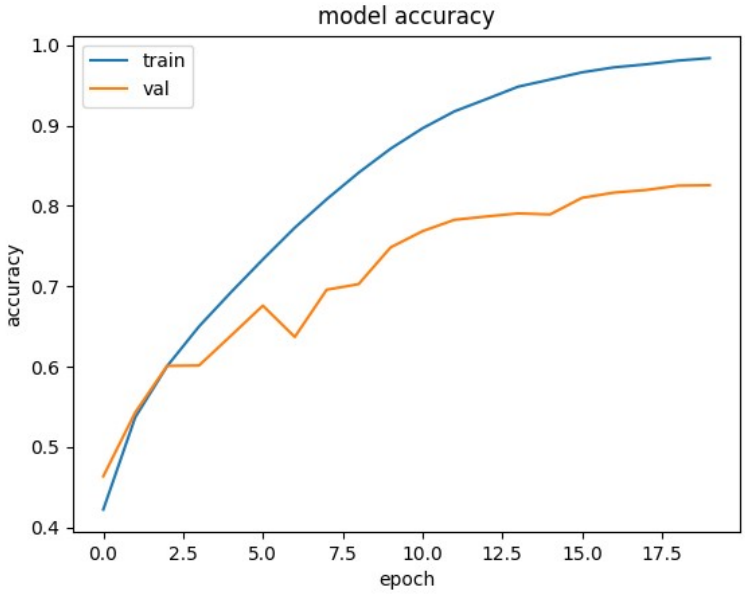
The Augmentation that is applied to images:

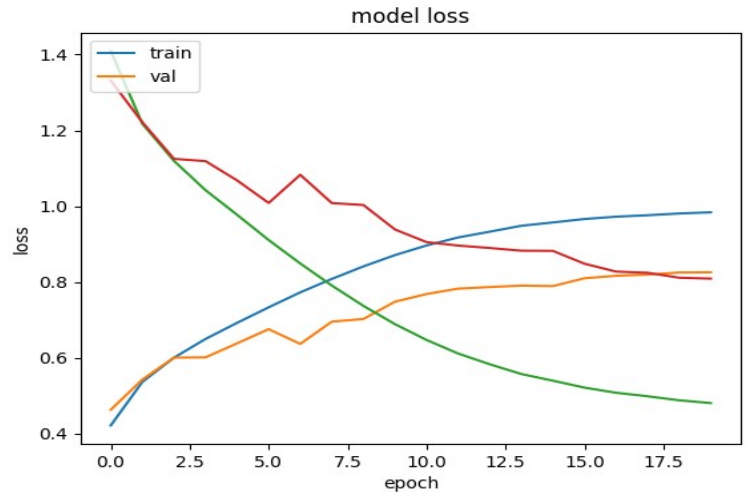
- Convert Images to grayscale.
- Sharpen images.
- Rotation by 20%
- Normalization of input count of fonts

Run Parameters

- 20 Epochs
- Batch Size 128
- Train/Validation ratio: 25%
- Learning Rate: 0.01
- Decay: 1e-6

- Momentum 0.9


model accuracy
train
val
accuracy
epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


model loss
train
val
loss
epoch
1.4
1.2
1.0
0.8
0.6
0.4
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

```
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d (Conv2D)              (None, 32, 32, 32)        320

 conv2d_1 (Conv2D)            (None, 32, 32, 64)        18496

 batch_normalization (BatchN  (None, 32, 32, 64)        256
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 16, 16, 64)        0
 )

 conv2d_2 (Conv2D)            (None, 16, 16, 128)       73856

 batch_normalization_1 (Batc  (None, 16, 16, 128)       512
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 128)         0
 2D)

 dropout (Dropout)            (None, 8, 8, 128)         0

 conv2d_3 (Conv2D)            (None, 6, 6, 256)         295168

 batch_normalization_2 (Batc  (None, 6, 6, 256)         1024
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 3, 3, 256)         0
 2D)

 flatten (Flatten)            (None, 2304)              0

 dense (Dense)                (None, 256)               590080

 batch_normalization_3 (Batc  (None, 256)               1024
 hNormalization)

 dense_1 (Dense)              (None, 128)               32896

 class (Dense)                (None, 5)                 645

=================================================================
Total params: 1,014,277
Trainable params: 1,012,869
Non-trainable params: 1,408
_________________________________________________________________
```

Attempt #1

Method

So, I have tried to do better, and achieve better results

In order to do that, I have written a method to augment and balance the dataset.

This method actually generates more variations from the same images I already have, and balances the classes.

Results

Thing is that I have found out that the validation loss and validation accuracy get better until 6th to 7th epochs, and then get (surprise) get worst!

```
Epoch 8: val_loss improved from 1.19898 to 1.17151, saving model to top_model.h5
293/293 [==============================] - 21s 71ms/step - loss: 1.0025 - accuracy: 0.5864
- val_loss: 1.1715 - val_accuracy: 0.4993
Epoch 9/20
293/293 [==============================] - ETA: 0s - loss: 0.9636 - accuracy: 0.6068
```

```
Epoch 9: val_loss improved from 1.17151 to 1.17093, saving model to top_model.h5
293/293 [==============================] - 21s 72ms/step - loss: 0.9636 - accuracy: 0.6068
- val_loss: 1.1709 - val_accuracy: 0.5006
Epoch 10/20
293/293 [==============================] - ETA: 0s - loss: 0.9273 - accuracy: 0.6240
Epoch 10: val_loss improved from 1.17093 to 1.15262, saving model to top_model.h5
293/293 [==============================] - 21s 72ms/step - loss: 0.9273 - accuracy: 0.6240
- val_loss: 1.1526 - val_accuracy: 0.5100
Epoch 11/20
293/293 [==============================] - ETA: 0s - loss: 0.8898 - accuracy: 0.6413
Epoch 11: val_loss did not improve from 1.15262
293/293 [==============================] - 21s 71ms/step - loss: 0.8898 - accuracy: 0.6413
- val_loss: 1.1593 - val_accuracy: 0.5115
Epoch 12/20
293/293 [==============================] - ETA: 0s - loss: 0.8566 - accuracy: 0.6561
Epoch 12: val_loss did not improve from 1.15262
293/293 [==============================] - 21s 71ms/step - loss: 0.8566 - accuracy: 0.6561
- val_loss: 1.1614 - val_accuracy: 0.5091
```

At this point, I had two options:
1. Stick to my model, and fine tune the hyper parameters
2. Change the model and hyper parameters.

I have decided to do both.

Attempt #2… #n

I found it quite hard to make the model predict at least more 55%. I consider it a "weak learner", and additional way to provide more accurate results will involve majority voting (Will be explained later)

Changes that made a great impact, and provided higher rate of validation accuracy:

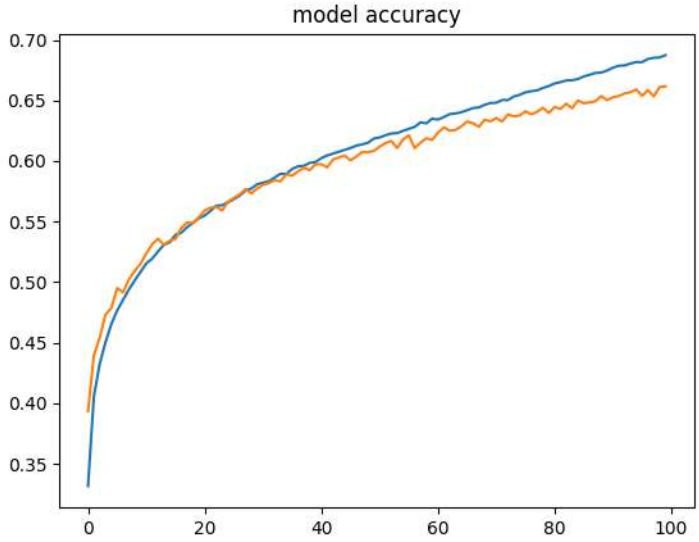
1. Set Learning Rate to 1e-3
2. 100 epochs
3. 40000 total items for each class, done with augmentation.

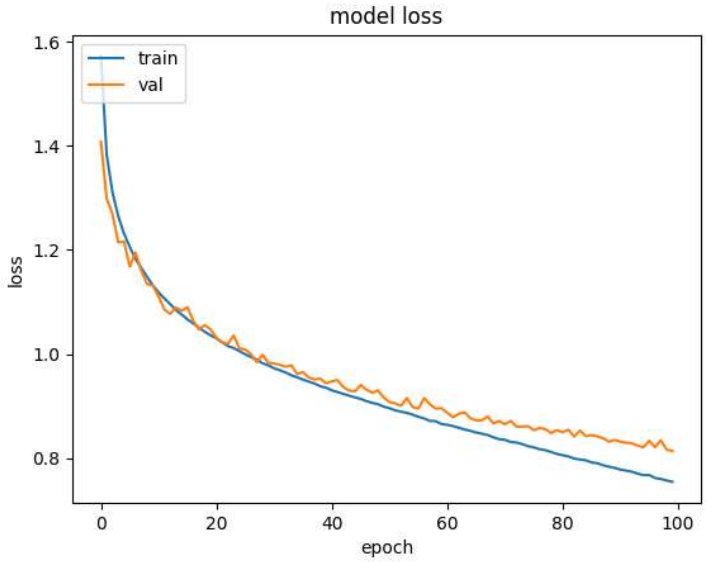
This configuration yielded a good enough result:

```
Test loss: 0.8133710026741028
Test accuracy: 0.6616106033325195
```

Here you can see training accuracy gain and loss during training session.

Also note that there is no overfitting, as the validation accuracy is close enough to the training accuracy.

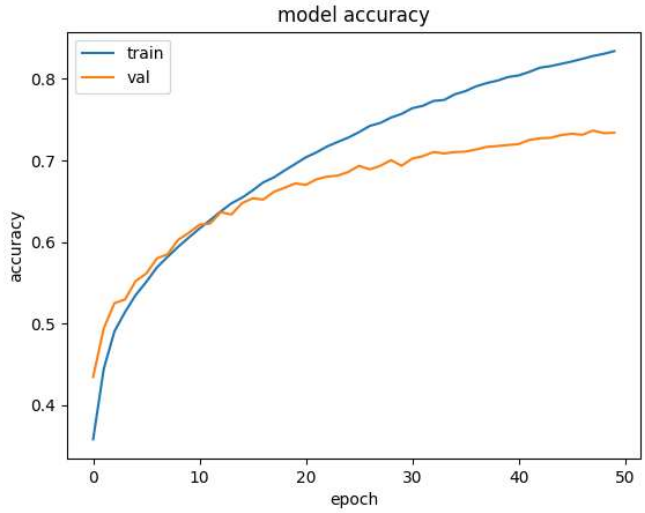




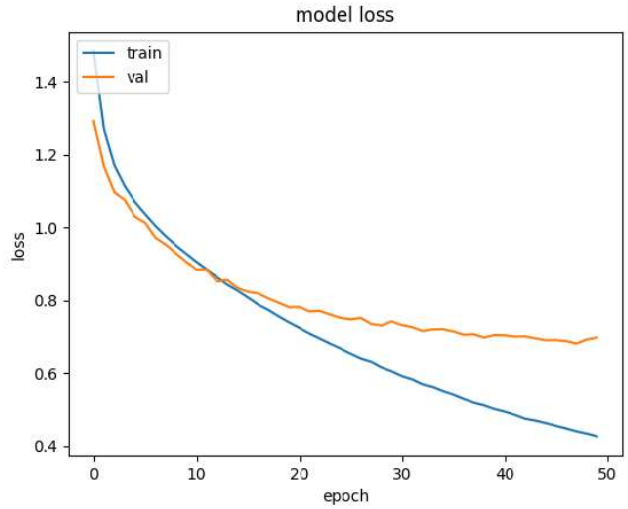
Attempt #n + 1

In this attempt, I changed the optimizer to 'Adam' optimizer,
I have decided to move forward with this optimizer, also after reading this negative opinion regarding it's lack of generalization, aggressive learning rates and lack on convergence.
To our needs at this project, this is more than enough.

As we can deduce from the graphs, there is a certain overfitting toward the higher epochs



This is noticeable also from loss graph



On top of these results, the total score of the model, with Adam optimizer looks promising:

```
Test loss: 0.69821697473526
Test accuracy: 0.7338399887084961
```

So, I'm going forward with that model.

Rest of the Code

Apart from the sections that are used to prepare the data, and the model.
The following lines are essential in helping raise the predictions accuracy.

First of, after getting the prediction result from the model, I'm running a argmax function, in order to find the correct class that the model predicted.

This is the conversion label/class conversion logic

```
0 == "Alex Brush"
1 == "Open Sans"
2 == "Sansation"
3 == "Ubuntu Mono"
4 =="Titillium Web"
```

After we get the predictions, I added an additional stage of majority voting.
The idea is to use the assumption on the data that all the chars that belong to the same word are of the same class.
Given that assumption, it is quite easy to loop through words, and assign the same class to all the letters of a word.