Machine Learning II DATS_6203_12
Date: April 21st, 2019

Individual Final Project Report

Yuyang Wang

1. Introduction.

- This project is aimed to classify writers based on their handwriting styles. The IAM Handwriting Dataset chosen for this project is comprised of a collection of handwritten passages by various writers. The dataset contains 4899 handwriting images with different sizes, written by 600 writers. For this project the top 50 writers' data are selected, considering the computation time and the data has been sufficient in classification.
- The components of this project are background research, data pre-processing, architecture construction, and model evaluation. Each group members do the search about how to use Keras to implement convolution neural network. In addition, members work on the data processing code together.

2. Description of your individual work.

- In this project, my individual work focus image reading and model testing. The original data are in PNG format but computers read images as pixels. My initial idea is to convert PNG image to pixels format and store the new data into a csv file. Then read the csv file as input. However, there are around 15000 columns for each image which takes a very long time to store and re-read data again. After learning from others' example code of converting images, I find directly store the pixels as variables will be much faster. In addition, I perform a grayscale normalization to pixels to reduce the effect of illumination's difference. That's also because convolution neural network converges faster in the range of 0 to 1.

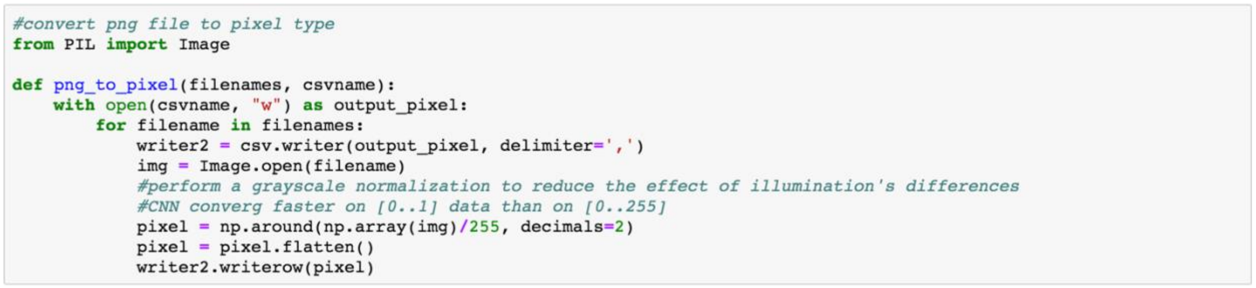
- CNN has a hierarchical processing pipeline, where the features in lower layers are primitive, while those in upper layers are high-level abstract feature made from combinations of lower-level features. Therefore, I think using more kernels in the upper layers will make more sense, because they contain richer information than previous layers. Thus, I try to use three layers with SGD as optimizer, and the kernels for each layer are 32, 64, 128 respectively. SGD has simple computation formula shown as below:

$$\eta_t = \alpha \cdot g_t$$

SGD optimizer includes support for momentum, learning rate decay, and Nesteroy momentum.

3. Describe the portion of the work that you did on the project in detail.

- Original image conversion code:

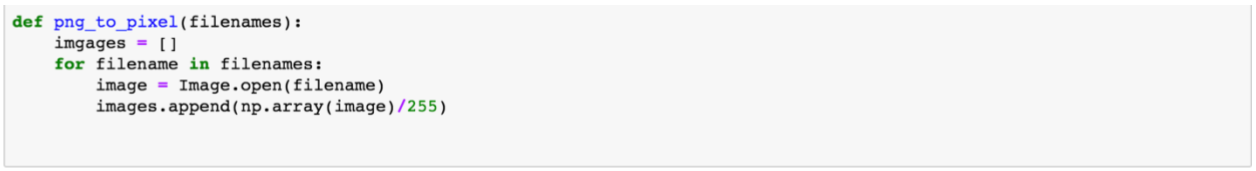


```
#convert png file to pixel type
from PIL import Image

def png_to_pixel(filenames, csvname):
    with open(csvname, "w") as output_pixel:
        for filename in filenames:
            writer2 = csv.writer(output_pixel, delimiter=',')
            img = Image.open(filename)
            #perform a grayscale normalization to reduce the effect of illumination's differences
            #CNN converg faster on [0..1] data than on [0..255]
            pixel = np.around(np.array(img)/255, decimals=2)
            pixel = pixel.flatten()
            writer2.writerow(pixel)
```

Fig1. Screenshot original image conversion code

- Improved image conversion code:

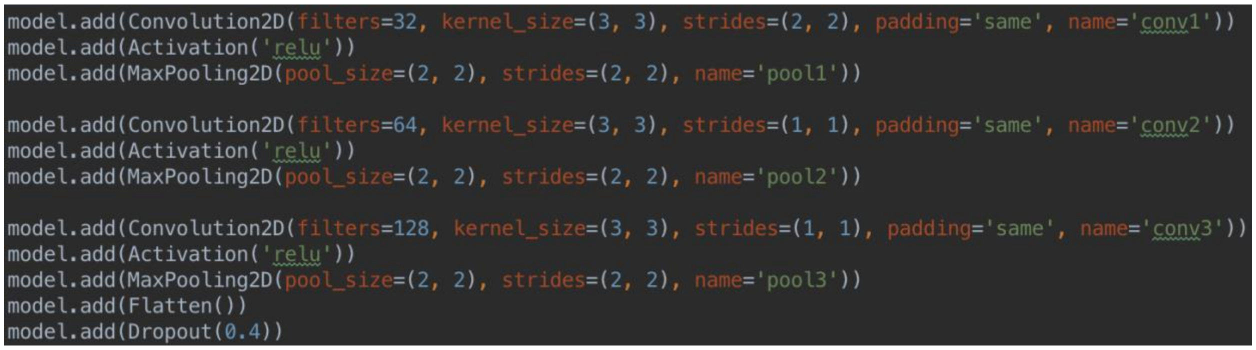


```
def png_to_pixel(filenames):
    imgages = []
    for filename in filenames:
        image = Image.open(filename)
        images.append(np.array(image)/255)
```

Fig2. Screenshot of improved image conversion code

- Model construction

The sequential model is used in this project. It allows us to easily stack sequential layers of the network in order from input to output. Thus, we first declare the model type by command *model = Sequential( )*. Next three 2D convolutional layers are added to process the input images. The first argument passed to *the Conv2D()* layer function is the number of output channels. The next input is the kernel size, which in this case I choose 3x3 moving window, followed by the strides in x and y directions (2,2). Next, activation function is a rectified linear unit and finally supply the model with the size of the input to the layer. Next, a 2D max pooling layer is added after. I simply specify the size of the pooling in the x and y directions as (2,2) in our case and the strides. In the end of the convolution layers, we want to flatten the output to enter the fully connected layers.



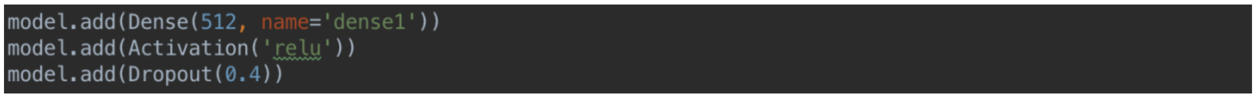
```
model.add(Convolution2D(filters=32, kernel_size=(3, 3), strides=(2, 2), padding='same', name='conv1'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), name='pool1'))

model.add(Convolution2D(filters=64, kernel_size=(3, 3), strides=(1, 1), padding='same', name='conv2'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), name='pool2'))

model.add(Convolution2D(filters=128, kernel_size=(3, 3), strides=(1, 1), padding='same', name='conv3'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), name='pool3'))
model.add(Flatten())
model.add(Dropout(0.4))
```

Fig3. CNN model construction part 1

The following lines declare the fully connected layers using *Dense()* layer in Keras. The declaration is easy and we just need to specify the size in line with our architecture. We specified 512 nodes, each activated by a ReLu function.

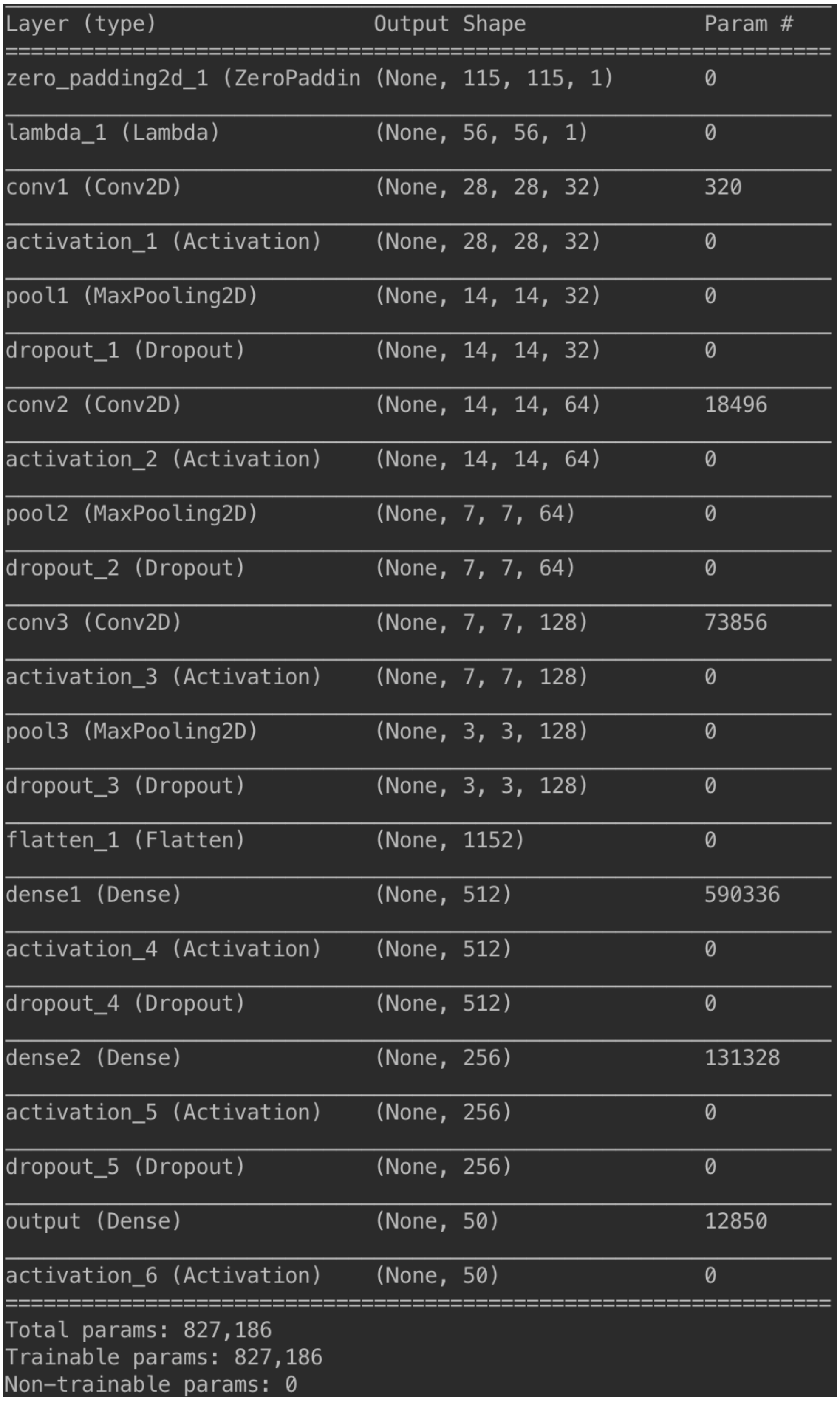


```
model.add(Dense(512, name='dense1'))
model.add(Activation('relu'))
model.add(Dropout(0.4))
```

Fig4. CNN model construction part II

4. Results.

- The following table shows the summary of the model created



```
Layer (type)                 Output Shape              Param #
=================================================================
zero_padding2d_1 (ZeroPaddin (None, 115, 115, 1)       0
_________________________________________________________________
lambda_1 (Lambda)            (None, 56, 56, 1)         0
_________________________________________________________________
conv1 (Conv2D)               (None, 28, 28, 32)        320
_________________________________________________________________
activation_1 (Activation)    (None, 28, 28, 32)        0
_________________________________________________________________
pool1 (MaxPooling2D)         (None, 14, 14, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 32)        0
_________________________________________________________________
conv2 (Conv2D)               (None, 14, 14, 64)        18496
_________________________________________________________________
activation_2 (Activation)    (None, 14, 14, 64)        0
_________________________________________________________________
pool2 (MaxPooling2D)         (None, 7, 7, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 7, 7, 64)          0
_________________________________________________________________
conv3 (Conv2D)               (None, 7, 7, 128)         73856
_________________________________________________________________
activation_3 (Activation)    (None, 7, 7, 128)         0
_________________________________________________________________
pool3 (MaxPooling2D)         (None, 3, 3, 128)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 3, 3, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1152)              0
_________________________________________________________________
dense1 (Dense)               (None, 512)               590336
_________________________________________________________________
activation_4 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense2 (Dense)               (None, 256)               131328
_________________________________________________________________
activation_5 (Activation)    (None, 256)               0
_________________________________________________________________
dropout_5 (Dropout)          (None, 256)               0
_________________________________________________________________
output (Dense)               (None, 50)                12850
_________________________________________________________________
activation_6 (Activation)    (None, 50)                0
=================================================================
Total params: 827,186
Trainable params: 827,186
Non-trainable params: 0
```

Table1. 3 layers CNN model summary

- Accuracy and loss are calculated in each epoch. Accuracy is the proportion of correctly classified samples by the model to measure how our model performs and whether good enough for our task or not. Although accuracy is truly comprehensible, however the problem is that it cannot be directly used by the learning process of out model to tune the parameters of the model. Thus, we need loss function, which we can us in the training process. We expect the loss value to be low so that we would have a high accuracy.

Epoch	Accuracy	Loss
1	0.2015	3.6172
2	0.2057	3.5805
3	0.2030	3.5718
4	0.2059	3.5118
5	0.2048	3.3455
6	0.2055	3.2651
7	0.2061	3.1112
8	0.2068	3.0849

Table2. 3 layers CNN model loss and accuracy

- The following graph is the visualization of accuracy change over epochs.

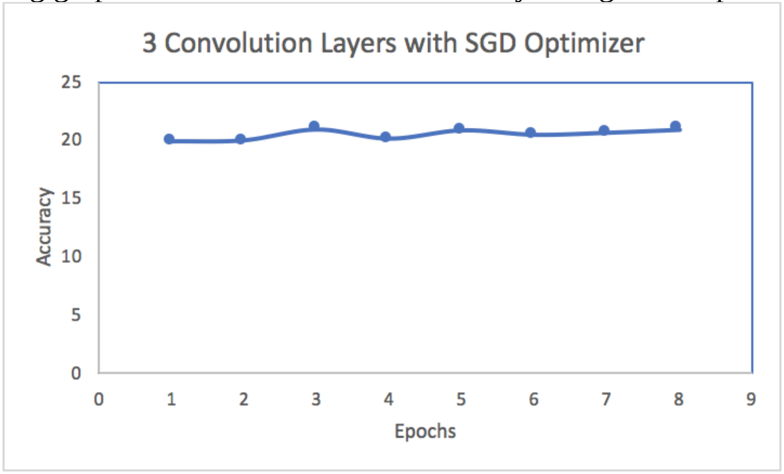


Fig5. Accuracy of 3 CNN layers with SGD optimizer

5. **Summary and conclusions.**

- Based on the accuracy of each epoch, we can tell that the performance of this model is really bad. When we use exact the same model construction with Adam optimizer instead, the final accuracy is around 80% which is much better. It can be told that SGD is not a good optimization for this data. Because SGD updates frequently, sometimes significant fluctuations may occur; the model may get stuck at the local minima or saddle point, where it is difficult for SGD to escape. For our project this is the case. While, for Adam, it usually works well even with very little hyperparameter tuning. Adam combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems.

6. **Calculate the percentage of the code that you found or copied from the internet.**

- $\frac{31-3}{31+132} = 17.17\%$

7. References.

Introduction to CNN Keras:
https://www.kaggle.com/yassineghouzam/introduction-to-cnn-keras-0-997-top-6/notebook

IAM Handwriting Database:
http://www.fki.inf.unibe.ch/databases/iam-handwriting-database

Keras Tutorial:
https://adventuresinmachinelearning.com/keras-tutorial-cnn-11-lines/

What's fully connected layer:
https://zhuanlan.zhihu.com/p/33841176

Checkpoint code:
https://machinelearningmastery.com/check-point-deep-learning-models-keras/