Lydia Jin
DATS 6203-12
Individual Final Report

Introduction

This project addresses an image classification problem, and further application of the image classification explored in class. The IAM Handwriting Dataset chosen for this project is comprised of a collection of handwritten passages by various writers. The goal of this project is to use deep learning to classify the writers by their writing styles. The results of this project can be used in future applications, such as identifying criminals by signature in fraudulent cases.

Our group decided to use a Convolutional Neural Network (CNN) for this image classification problem. Together we worked on the pre-processing of the data, which consisted of code found online and our own work. We found code for a generator function that would generate random equally sized crops from each sentence. We also worked together on ways to visualize our results. Then to make comparisons between various optimizers and number of layers, we each individually took over a model. I worked on the 3-layer model with an Adam optimizer, and each of my group members worked on a 3-layer CNN model with SGD optimizer or a 4-layer CNN model with Adam optimizers. Together, we created a 4-layer CNN model with an SGD optimizer. Our group met multiple times to write the final report, revise our code, and create our presentation.

Individual Work Description

I used a Convolutional Neural Network (CNN) as my model because it is one of the main categories for performing image recognition and classification. CNN image classification takes an input image, processes it and classifies it under certain categories. A 3-layer CNN architecture is shown below:

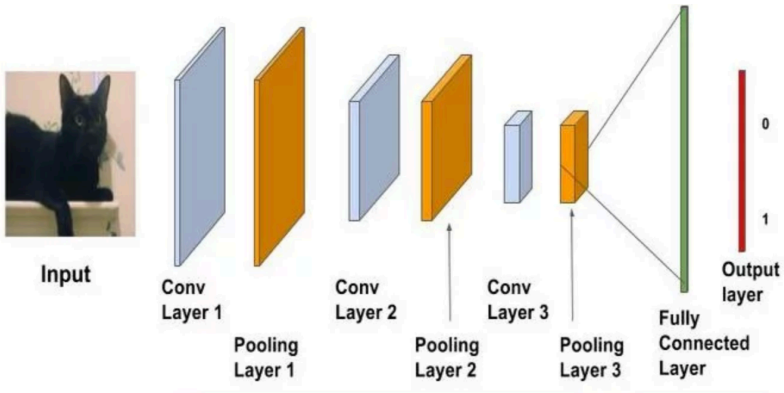

Fig1: Convolution Neural Network Architecture

My model is similar, in that there are 3 convolution layers and 3 max pooling layers. But my model has 3 fully connected layers with dropout layers to reduce the network. This type of CNN model with an Adam optimizer proved to be the best of the 4 models we experimented with.

Project Description

I worked on a variation of the 3-layer CNN model, with an Adam optimizer. There were three main parts to the code: pre-processing, model, and data accuracy/performance.

Pre-processing:
First, I created a target list that included the writer's number according to their handwriting. Then I label encoded the target names, from 000 to 0 and 001 to 1. Then I split the data into 3 sets (training, validation, testing) with a ratio of 7:1.5:1.5. Since the image files from our dataset are of different sizes, we have to resize and crop them to get equal sized patches from each image. For this, I used the generator code we found. It has multiple functions, such as scanning through each sentence and generating random patches of size 113*113. It is also used to get the pixel values of the images in the training, validation and testing sets and perform a grayscale normalization to reduce the effect of illumination's differences. As part of the code, it divided the X_train images files by 255 to allow the CNN to converge faster on [0..1] data compared to the original [0..255] data.

Model:
To create the model, I decided to use the Keras framework, which is the best for deep learning models with stacked layers. My group members are utilizing Keras as well, since it makes it easy to add or change layers. For judging performance, Keras also has functions such as fit_generator that can return accuracy and loss values.

The proper formatting for a CNN model is a convolution layer followed by an activation function and a max pooling layer. For my model, I repeated this format 3 times with a chosen number of kernels (filters) and kernel size. Next, I flattened the data before adding a Dropout layer. Then the model transitioned into Dense/fully connected layers. Finally, the model was compiled with a cross entropy loss function and an Adam optimizer. The model with all its layers is shown below:

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
conv2 (Conv2D)	(None, 14, 14, 64)	18496
activation_2 (Activation)	(None, 14, 14, 64)	0
pool2 (MaxPooling2D)	(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
flatten_1 (Flatten)	(None, 1152)	0
dropout_1 (Dropout)	(None, 1152)	0
dense1 (Dense)	(None, 512)	590336
activation_4 (Activation)	(None, 512)	0
dropout_2 (Dropout)	(None, 512)	0
dense2 (Dense)	(None, 256)	131328
activation_5 (Activation)	(None, 256)	0
dropout_3 (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

Fig2: 3-Layer CNN model with Adam optimizer

In terms of the parameters I chose for this model, after experimentation with various batch sizes, a batch size of approximately 16 was found to be the best for this model. My group members found that a similar batch size worked for best for their models as well.

For the Dropout layer, I found that a probability of 0.4-0.5 was the best for the Dense/Fully Connected layers. I did some experimentations with Dropout layers after each convolution layer but found that a probability of 0.1-0.2 worked best. But unfortunately, the accuracy decreased quite heavily compared to when there were no Dropout layers between convolution layers.

And of course, since I decided to work with the Adam optimizer, I experimented with a variety of different learning rates and beta values, but ultimately found that the default learning rate (lr=0.001, beta_1=0.9, beta_2=0.999) returned the best results.

Accuracy/Performance:
I trained the model with 8 epochs and 1000 samples per epoch. Originally, I intended to run 3000 samples per epoch, but even on GPU, the running time was too long. I used the Keras function fit_generator to get the accuracy and loss values. As a part of the code, I used the Keras ModelCheckpoint function to save the epoch number and accuracy. Comparing the accuracy values of the 8th epoch for the 4 models we created, we found that the 3-layer Adam had the best accuracy at around 80% accuracy.

I also wrote code to plot the accuracy and loss values between the training and validation sets to identify overfitting and model accuracy. The code for the plot is shown below:

```
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, nb_epoch), history_object.history["acc"], label="train_acc")
plt.plot(np.arange(0, nb_epoch), history_object.history["val_acc"], label="val_acc")
plt.plot(np.arange(0, nb_epoch), history_object.history["loss"], label="train_loss")
plt.plot(np.arange(0, nb_epoch), history_object.history["val_loss"], label="val_loss")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Accuracy/Loss")
plt.legend(loc="lower left")
plt.show()
```

Fig3: Code for plotting accuracy and loss of Training and Validation sets

Results
The Keras function fit_generator returns the accuracy and loss values for each of the 8 epochs. To calculate the accuracy, the test samples are fed into the model created, and the number of mistakes are recorded by comparing to the true target. This allows a percentage based on misclassification to be calculated. In the case of the 3-level CNN model with Adam optimizer, the accuracy on the final epoch was approximately 80%. The accuracy is easy to understand, but it has a downside, in that it cannot be used to tune the model parameters during the learning process.

The loss value is calculated from the training and validation sets. It interprets how well the model is doing for the two sets, as a summation of the errors made for each sample in the training or validation sets. Therefore, loss is not a percentage as with the accuracy. The main objective of a learning model is to minimize the loss values since a better model has lower loss; the only exception being if the model has over-fitted. As a result, loss values will typically decrease for each epoch.

```
Epoch 1/8
1000/1000 [==============================] - 2065s 2s/step - loss: 2.5778 - acc: 0.3068 - val_loss: 1.6233 - val_acc: 0.5028

Epoch 00001: saving model to check-01-1.6233.hdf5
Epoch 2/8
1000/1000 [==============================] - 1813s 2s/step - loss: 1.4534 - acc: 0.5520 - val_loss: 1.0423 - val_acc: 0.6724

Epoch 00002: saving model to check-02-1.0423.hdf5
Epoch 3/8
1000/1000 [==============================] - 1799s 2s/step - loss: 1.0960 - acc: 0.6599 - val_loss: 0.8940 - val_acc: 0.7204

Epoch 00003: saving model to check-03-0.8940.hdf5
Epoch 4/8
1000/1000 [==============================] - 1794s 2s/step - loss: 0.9058 - acc: 0.7189 - val_loss: 0.7573 - val_acc: 0.7642

Epoch 00004: saving model to check-04-0.7573.hdf5
Epoch 5/8
1000/1000 [==============================] - 1789s 2s/step - loss: 0.7845 - acc: 0.7559 - val_loss: 0.6939 - val_acc: 0.7839

Epoch 00005: saving model to check-05-0.6939.hdf5
Epoch 6/8
1000/1000 [==============================] - 1807s 2s/step - loss: 0.6902 - acc: 0.7852 - val_loss: 0.6555 - val_acc: 0.7984

Epoch 00006: saving model to check-06-0.6555.hdf5
Epoch 7/8
1000/1000 [==============================] - 1797s 2s/step - loss: 0.6219 - acc: 0.8066 - val_loss: 0.5794 - val_acc: 0.8222

Epoch 00007: saving model to check-07-0.5794.hdf5
Epoch 8/8
1000/1000 [==============================] - 1836s 2s/step - loss: 0.5696 - acc: 0.8222 - val_loss: 0.5786 - val_acc: 0.8194
```

Fig4: 3-Layer CNN model with Adam optimizer training model output (8 epochs)

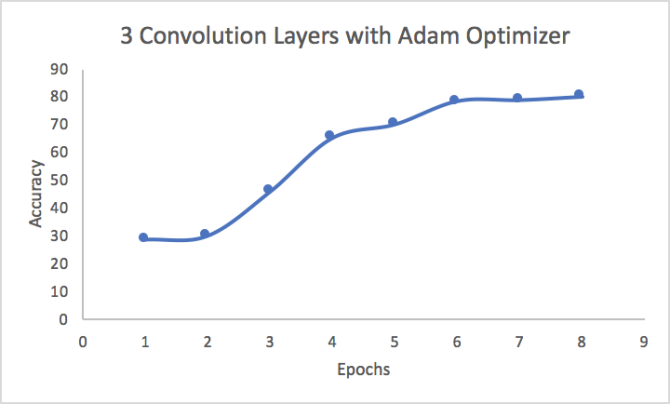

Fig5: Visualization of accuracy over 8 epochs. There is a steady increase before leveling off at around 80% accuracy.

Summary and Conclusions

Summarize the results you obtained, explain what you have learned, and suggest improvements that could be made in the future.

The results from this 3-layer CNN model with Adam optimizer shows that the performance of this model is really good overall. In comparison with the other 3 models that were tested, there are some conclusions that can be made. For classifying this type of handwriting image data, an SGD optimizer is a very bad choice, and a 3-layer CNN works better than a 4-layer CNN. Dropouts are necessary in the fully connected layers, and possibly in the convolution layers, to prevent overfitting; but it is necessary to experiment with different dropout ratios. Deciding on the number of batches, epochs, samples per epoch, kernels, and the kernel size are important to getting more accurate models.

I learned a lot about the different ways of manipulating layers using the Keras framework. This also allowed me to understand how flexible a CNN algorithm can be. But most

importantly, this process of building our models and putting our ideas and conclusions into a report allowed me to better connect the aspects that we learned in class.

In terms of suggestions for improvements to this analysis, I would definitely think about trying to classify 10 classes instead of the 50 our models are currently classifying. There are some more comprehensive visualizations, such as confusion matrixes or visualizing the gradients for each layer. Also, if we were given more time and higher computing power, it would be good to consider running more samples per epoch.

Percentage
Our group took 31 lines of code from an online source in order to create a generator for our data. We didn't make any modifications to those lines of code, and the total lines of code for mywork.py is 156.

$$\frac{31 - 0}{31 + 136} = 18.56\%$$

References
Dataset: https://www.kaggle.com/tejasreddy/iam-handwriting-top50
CNN: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2
Checkpoint code: https://machinelearningmastery.com/check-point-deep-learning-models-keras/
Generator code: https://www.kaggle.com/tejasreddy/offline-handwriting-recognition-cnn/notebook
CNN Background Information: https://www.learnopencv.com/image-classification-using-convolutional-neural-networks-in-keras/
Save/Load Keras models: http://faroit.com/keras-docs/2.0.2/models/about-keras-models/