Offline Handwritten Letter Recognition

Project Overview

In this project, we compared the performance of CNN models constructed under different algorithms for the handwritten letter recognition task. We used a simple neural network model as our baseline model for comparison. Our results show that both CNN models outperform the baseline while the AlexNet algorithm gives the highest accuracy. The AlexNet CNN model achieves a 98% accuracy and the Keras model gives over 95%, while the baseline model is at about 90%. We believe that the results are acceptable considering the limited time and resources such as training data, computing power, etc.

In this paper, we will first provide a detailed introduction to the problem and dataset of our project, along with its potential applications. Next, we present the previous literatures that inspired us with starting points and ideas of certain techniques we used in the project. Next, we give the breakdown of the experimental process, including the higher-level concept of each algorithm we use. After that, we will describe the key coding part of our model along with other difficulties we experienced in this project and how we overcame the problems. The last parts present the result of our project and the future works we believe this project can lead into.

Problem Introduction

The goal of our project is to develop a solid and stable offline machine learning model to correctly recognize handwritten letters. In detail, our project can be separated into two parts. In part one, which is also the main body of this experiment, we wish to make a comparison among various models and evaluate the performance, advantages and drawbacks of each model. Our target model is set to be the convolutional neural network because it has been used in many state-of-the-art researches in the field. The second part of the project is to gain some brief insights on what and how the CNN model is accomplishing the tasks in supervised image analysis.

We choose image analysis tasks out of the mutual interest among all group members. This particular problem of handwritten letter recognition caught our attention as a listed project on Kaggle.com; and it was finally chosen because the scope and level of difficulty is suitable to the resources and time limit we have. Offline handwriting recognition also has a lot of useful applications in the real world, such as banking and postal automation, old document parsing, etc.

We made several restrictions to the scope of our project. First, we decide to do an offline model. An online model, which captures tip movement as signal for classification [4], relies heavily on the environment setting and is out of scope of this class. Next, we limit the space

of our classification model to the uppercase English alphabet. We exclude all non-alphabet characters because it is too large a space but has relatively low application value. We also exclude lowercase letters since the best dataset we can find didn't include lowercase and it won't be possible to prepare a large enough training sample by ourselves.

Previous Works in Literature

Many works have been done in the field of image processing, text recognition. Some of the most famous models that show outstanding performance include CNN, Tensorflow, OpenCV, ImageNet, etc. [5][6] We were also reminded while reading past papers that one-hot encoding is the essential part of all neural network models which lead us to look further into how it functions and how to implement it. [6][10] Another idea inspired by previous literature is to do a comparison among various models, which has been performed in text semantic classification problems, etc. [7] We believe such activity can help us better understand the strengths and weakness of different models. Some other techniques that caught our attention that may be useful to our project include the concept of bagging, cross-validation, etc. [8]

Project Decomposition

Our project can be decomposed into the following parts. Each part is assigned to one or more group members.

Data gathering and processing

The first step to a good machine learning project is to find a proper dataset for training and validation. Taotao is mainly in charge of this part. Our original dataset at the project checkpoint was the public dataset "handwritten_data_785.csv" provided with the problem "Handwritten A-Z" on Kaggle.com. However, we decided to change it because the sample inputs are too well-written and size of the data is relatively small. With such training set, our model won't perform well if we are to test it on some "oddly written" samples. Our final data is "emnist-letters-train.csv" and "emnist-letters-test.csv" from the publicly available dataset "EMNIST".

This part of the project also includes preprocessing the data before it can be fed to models we constructed. The preprocessing of data include reading and visualizing the raw data, transforming the dimension of the data, parsing the labels to the proper machine learning setting etc. The one-hot encoding function for parsing the labels is written by Yingyin following example of Gulli, et al [10].

Reference and previous work research

This part is mainly the responsibility of Emily. To build the models more efficiently, we need the help from previous researches and other helping sources such as online tutorials and

books covering the topic. The task requires large amount of reading and is time-consuming. Emily's research provided us with various ideas and starting points such as bagging, cross-validation, one-hot encoding, AlexNet, SVM, to name a few. Many related works in the field are mentioned in the previous section of this report.

Model construction and comparison

With the above research and preparation, the next step is to build our own models and conduct a comparison among them. We decided to build four different models: a baseline model, a CNN model using Keras package, a CNN model using AlexNet and an SVM model.

Chenhao is in charge of the baseline model. For the choice of baseline, one option is to simply use random guessing. However, we would like to set the benchmark a little more challenging and see if we can beat it. The final choice is a simply neural network model. This model consists of a single fully connected layer with ReLU activation, a dropout layer and softmax output layer.

The Keras CNN model is implemented by Taotao. The model consists of 3 hidden layers, each include a convolution layer, a max-pooling layer and a dropout layer, followed by a fully connected layer after flattening the input. And a softmax layer is added to the end to obtain output.

The AlexNet CNN model is implemented by Yingyin. The model consists of 5 hidden layers and a fully connected layer. This model is the hardest to implement because many built-in functions available in Keras are not available for AlexNet and hence require manual implementation. This model is also heavily resource-consuming which will be discussed in more detail in the next section.

The last model we were trying to implement and compare is the SVM model. However, after several attempts, we found that the SVM model runs extremely slow and reports very low classification accuracy. As a result, we decide to drop it from our comparison.

The visualization of performance of each model is done by the constructor while Taotao wrote the function to show the confusion matrix using external code of Fisher, G [11] as a reference.

CNN model insight

The next part is to look further into the CNN model and gain insight about how the convolutional layers work. This part is mainly done by Chenhao following the tutorial of Gabriel Pierobon [12]. This part requires saving the model weights and extracting each layer as a single activation model. Then the output of each layer is visualized for one single input to see if we can give any human interpretation.

Poster and project report

The final part of our project is to construct the demonstration poster and the project report. The poster is done by Chenhao and the report in gathered by Emily and finalized by Chenhao. In addition, Yingyin and Chenhao worked on the final cleanup of all codes.

Coding and Problem Solving

The project decomposition and high-level model construction concept has been introduced in the previous section. In this section, we present our experience, some of the difficulties we came into and how we overcame them.

Data processing

The first difficulty we experienced is the preprocessing of data. As lots of experienced engineers said, 80% of the time is spent on data cleaning and only 20% is devoted to the actual model implementation and testing.

We tried to use a dataset that use .png files as inputs because the sample includes both upper and lower cases. However, the most difficult part is to modified the original sample picture and convert them to the format that we can use and do the training. The main goal is to resize the image from 128x128 to 28x28 to reduce file size and increase computation efficiency. The challenge is to reduce the size without losing too much information from original image. Because some of the letters are very small in the center of the image, the resized letter would become blurred and even smaller if we apply simple algorithms such as sliding a window and taking the median. We first apply the Gaussian filter with sigma=1 to image, then threshold image. A bounding rectangle is fitted to the letter in the image then extracted. To avoid the character touch the boundary of image, we add 2 pixels to four sides of image. This process is able to make better use of empty space in the image. [1] Although, in the end, we did not use this processed data since we limited our scope to only capital letters, we believe this is a valuable experience to us and worth sharing.

Keras for CNN

The next experience is how powerful the Keras library is. All the major functions that are involved in the construction of a CNN model are already implemented in the library. This makes the coding process straight forward and easy to understand. With this advantage, we were able to spend more time in tuning the parameters and experimenting the best setting of the CNN model. We went through the tutorial by Brownlee, J. [14] as a guidance for our coding process.

AlexNet for CNN

For AlexNet, it was originally designed for the input of 224*224 of RGB images, in other words, the input requires an N*224*224*3 array, which is quite different on our sample gray-scale images with the dimension of N*784. Thus, we need to modify the original AlexNet code to fit our model construction. This makes the construction of the model much more complicated compared to Keras. The model was manually built using the tutorial by Birbeck, E. [13] as a reference.

Another experience with AlexNet is that we found this algorithm is highly resource consuming. For our 5-layer CNN, we tried to run it with all training samples in the dataset (about 89K) on Kaggle.com using Notebook kernel. The training failed with the error

message “resource exhausted”. Such property can also be seen from the training time of various models. While the baseline model and Keras model takes only 1 second and 5 seconds respectively for each full iteration, AlexNet take about 1 minute. This can be critical in a limited-resource situation.

Improving performance

We try to use some techniques to improve the performance of our models.

For each algorithm, we use cross-validation in the training process and finally test the performance on a leave-out validation set. This technique makes sure that the models do not have overfitting problems. Another technique we used to avoid overfitting is to add dropout layers in the CNN models. Such a technique also has the additional benefit of reducing parameters and hence reduces computation complexity.

We also considered including normalization layers in our models to improve performance. But the result is not significant since ReLU activation is not sensitive to normalization.

In addition, we implemented a checkpoint to record the model’s weights of the iteration with highest accuracy and reapply these weights to obtain the final model. With enough number of full iterations, the model performance will go ups and downs instead of strictly increasing. The benefit of this technique is to ensure we are using the optimal version seen in the training process.

Results and Comparison

Evaluating performance by validation accuracy, both our CNN models outperform the baseline model. The baseline model reports an accuracy of about 90%, while CNN models with Keras and AlexNet are both over 95%. The AlexNet model can obtain as high as 98.55% accuracy in our testing process.

Further insight may be obtained from the confusion matrices. From Figure 2 and 3, we can see that the most misclassifications occur when the model confuses “I” with “L”. This really

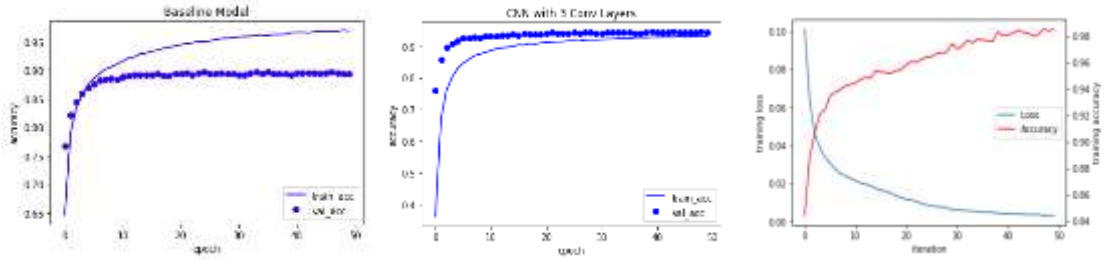

Figure 1: Comparison of Model Accuracy

Left: baseline mode, Middle: Keras CNN model, Right: AlexNet CNN model

makes sense since the two letters have very similar structures. The handwriting can be hard to recognize even for humans. Another examples like this are misclassifying “G” as “O”, or “V” as “Y”, to give a few.

The confusion matrix for AlexNet gives us a hint of why the model gives such high accuracy, much higher than we would have expected. One reason may be that the testing sample is much smaller than the testing set of the other two models. This choice is forced by the fact that our computing power is not enough to run the complete testing set. Another reason, which we believe is the main cause, is that the test set is highly unbalanced. It contains very few instances of the letters that are hard to classify such as “I” and “V”. This would significantly reduce the difficulty of the task. We actually obtained the testing set by randomly sample from the complete data. But for some reason, this is the outcome we have.

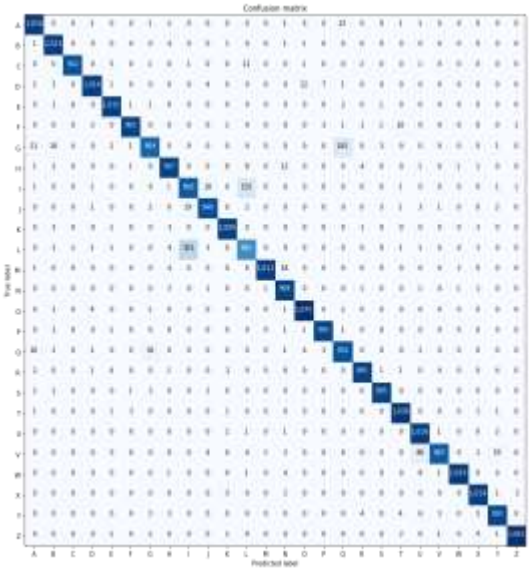

Figure 2: Confusion Matrix of Baseline Model

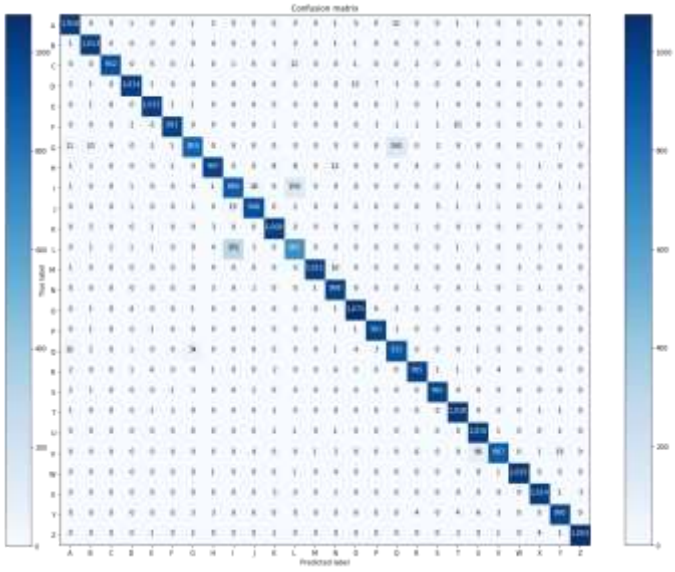

Figure 3: Confusion Matrix of Keras CNN Model

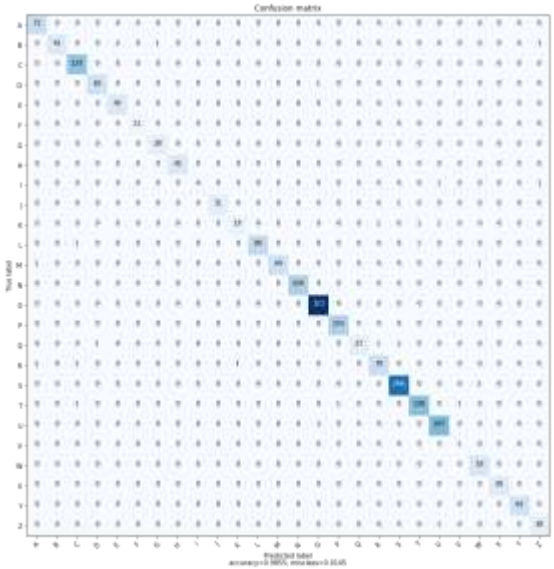

Figure 4: Confusion Matrix of AlexNet CNN Model

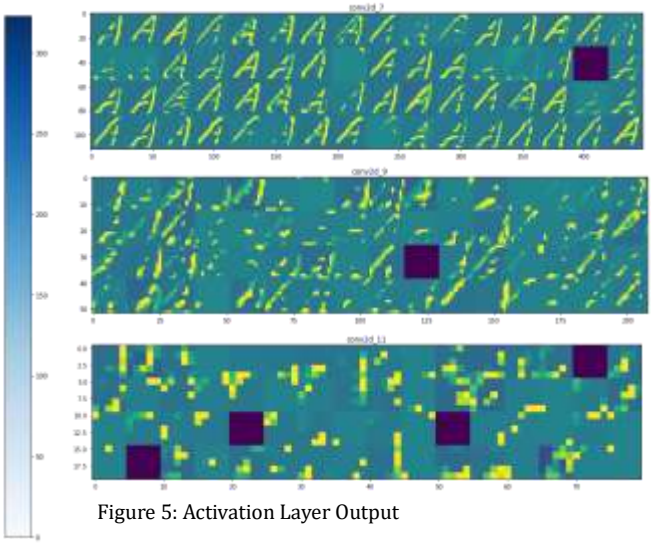

Figure 5: Activation Layer Output

Top: 1st hidden layer, Middle: 2nd hidden layer, Bottom: 3rd hidden layer

Now, we would like to have some insight on how the convolutional layers are actually extracting features within a raw image input, and how the features change among layers. We use a single image input of a letter “A” and show the result after the convolutional step of

each hidden layer. As we can see from Figure 5, the earliest layers show recognizable traits of the original image but later layers more and more on abstract and local properties. These insights are based on Keras CNN model for simplicity.

Future Works

Our project is just a starting point in the field of handwriting recognition and can be taken further in many directions. Here we discuss a few of them.

The first thing we can do is to include lower case letters into the model. One approach to tackle this problem is to train upper case and lower case separately and an additional model for classifying if the given character is upper or lower case. With such a setting, a new input will first be classified as either upper or lower case, and then went through the next model accordingly.

One obvious extension is to include other characters such as digits, other Latin-originated language alphabets, commonly used signs, etc. In the end, it could even extend to other language systems and complex mathematical symbols. The major difficulty of such extension is that the problem space is too large and the classification of which subset problem it is (i.e. telling whether the given character is English or Greek) can already be a stand-alone research topic.

Another extension direction is to be able to recognize words instead of single characters. This is hard because each letter has different width and height. As a result, we won't be able to simply partition the word by fixed window size and reduce it to single character recognition.

Conclusion

Our findings for this project include that the use of a 3 layer CNN results in the best accuracy at around 96% and a 5 layer CNN results in the best accuracy at over 98%. The difficulties of our project include finding the correct parameters to use as features, finding helpful datasets, image resizing, having sufficient GPU, and increasing the accuracy. The cumulation of this project taught our group members about convolutional neural networks, AlexNet architecture, segmentation, image resizing, support vector machines, and using different open source machine learning libraries.

There are many different types of networks used in machine learning, and for a beginner it may be difficult to know which type of neural network to choose depending on what type of problem one is trying to solve. The method used in this project is called a Convolutional Neural Network (CNN). CNNs are used to break up a component into its subcomponents. In the case of this project the main component is the image with the handwritten letter and the subcomponents are the different objects in the image such as the outline of the letter or the

different lines and curves of the handwritten image. The deeper the model, the more localized and detailed the extracted features would be.

Convolutional Neural Networks can be useful for recognizing larger structures such as faces, objects and are ideal for image and video processing. CNNs also have applications in medical image analysis, recommender systems, and natural language processing. Today, the best offline text recognition methods use convolutional neural networks to extract visual features over several overlapping windows of a text line image which an RNN(Recurrent Neural Network) then uses to produce character probabilities.

Bibliography

Group Members of CS178 final project group 20

Yingyi Xu	Taotao Qian	Emily Kang	Chenhao Qian
Yingyx2@uci.edu	taotaoq@uci.edu	kangee@uci.edu	chenhq1@uci.edu
26382819	12129755	131013703	001035275

References

1. Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. Dataset retrieved from http://arxiv.org/abs/1702.05373.
2. Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.
3. S. N. Srihari and E. J. Keubert, "Integration of handwritten address interpretation technology into the United States Postal Service Remote Computer Reader System". `Proc. Int. Conf. Document Analysis and Recognition (ICDAR) 1997, IEEE-CS Press, pp. 892–896
4. Vaidya, Rohan, et al. "Handwritten Character Recognition Using Deep-Learning." 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT). IEEE, 2018.
5. Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.
6. Zhang, Xiang, Junbo Zhao, and Yann LeCun. "Character-level convolutional networks for text classification." Advances in neural information processing systems. 2015.
7. Gallo, Ignazio, Shah Nawaz, and Alessandro Calefati. "Semantic text encoding for text classification using convolutional neural networks." 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR). Vol. 5. IEEE, 2017.
8. Dong, Yan-Shi, and Ke-Song Han. "A comparison of several ensemble methods for text categorization." IEEE International Conference on Services Computing, 2004.(SCC 2004). Proceedings. 2004. IEEE, 2004.

9. Puigcerver, Joan. "Are Multidimensional Recurrent Layers Really Necessary for Handwritten Text Recognition?." Document Analysis and Recognition (ICDAR), 2017 14th IAPR International Conference on. Vol. 1. IEEE, 2017.

10. Gulli, Antonio, and Amita Kapoor. TensorFlow 1.x Deep Learning Cookbook: over 90 Unique Recipes to Solve Artificial-Intelligence Driven Problems with Python. Packt Publishing, 2017. Pg. 278

11. Fisher, George. “Plot a confusion matrix”, Kaggle.com, https://www.kaggle.com/grfiv4/plot-a-confusion-matrix

12. Pierobon, Gabriel. “Visualizing intermediate activation in convolutional neural networks with Keras”, TowardsDataScience.com, https://towardsdatascience.com/visualizing-intermediate-activation-in-convolutional-neural-networks-with-keras-260b36d60d0

13. Birbeck, Ellie. “How to build a neural network to recognize handwritten digits with TensorFlow”, DigitalOcean.com, https://www.digitalocean.com/community/tutorials/how-to-build-a-neural-network-to-recognize-handwritten-digits-with-tensorflow

14. Brownlee, J. “Handwritten Digit Recognition using Convolutional Neural Networks in Python with Keras”, MachineLearningMastery.com, https://machinelearningmastery.com/handwritten-digit-recognition-using-convolutional-neural-networks-python-keras/