Problem Set 4: Application of Convolutional Neural Network - Lung Images Classification

Jianing (Julia) Chen

MS Applied Data Science, University of Southern California,

Los Angeles, California 90089, USA

(Dated: March 17, 2021)

Abstract

This report works on over ten thousand 64*64 pixels lung images and utilizes convolutional neural networks algorithm to build classification. To successfully classify x-ray or CT-scans of lungs into three categories will help hospitals deciding on giving treatments. As a result, the model with original labels works well on the training and validation set but cannot distinguish the testing set. It is because of the imbalanced nature of the dataset. As I weight the classes, results get lower accuracy score on training and validation set but getting a higher score on Kaggle. Finally, I would like to use LeNet-5 with tuning hyperparameters and data augmentation as the final model since it works the best in both situations. In reality, it is also necessary to weight classes' labels to achieve a better result.

I. INTRODUCTION

In this problem set, I aim to use convolutional neural networks to build image classifiers. In the beginning, I preprocessing the datasets, then I ultilize both LeNet-5 and VGG-16 architecture. I used the structure of LeNet-5 and VGG-16 that we introduced in the lecture to build base models and modified it, such as adding more layers and changing the learning rate. I build four models, but I fit both the original data and the weighted data in model fitting. To measure models' performance, I plot out the training and validation accuracy to prevent over-fitting and report micro, macro F1 score, recall rate for each label and testing accuracy (calculated by Kaggle).

II. DATA EXPLORATION & DATA PREPROCESSING

For this problem set, we are given training image data with labels and testing data. The training data have 13260 pieces of 64*64 pixels images. The labels range from 0 to 2, and represent the status for each lung. Such as healthy, which does not need any medication, and pre-existing conditions are not very serious but have few problems such as aortic enlargement. Another category is a severe condition and needs immediate attention. We need to notice the imbalance of classes: there are 10506 label 0, 2372 label 1, 382 label 2.

FIG. 1. Example of lung condition image, reference Problem Set 4 guideline

The training data has two parts: the images (in npy file), and the other is the labels(in csv file). Few steps I did for the data preprocessing:

- cast image data from uint8 to float64, the length of dimensions for image data is (13260, 64, 64, 1).

- remove the ID column in label data and cast to float64, the length of dimensions for image data is (13260,).
- Split into training and validation set. Then normalize the image data and test data by dividing 255. In total, we will have 5 sets of data. They are train_x, valid_x, train_y, valid_y and test.
- Create a more balanced weight by using counts of labels divided by all labels' sum.

III. MODEL SELECTION

A. LeNet-5 from Lecture note

The two base models I used are LeNet-5 and VGG-16. The first model I create has detailed architecture below on FIG. 2. The input image has an activation shape (64,64,1). The first convolution layer has 20 filters. The usage of the filter is used to detect the patterns and local information on the images[1]. I also listed the activation shape for each layer after the input. In the convolution layer, how it is calculated is by (N-f+1)/s, where "s" is the stride size [1], "f" is the filter size. So the shape can be write as ((N-f+1)/s, (N-f+1)/s, number of filters)[2]. The activation size is just simply multiplied each term in the activation shape.

In the pooling layer, when the padding value is "same", activation shape is calculated by (N, N, number of filters). In contrast, when padding value is "valid", activation shape will be shrink to ((N-f+1)/s, (N-f+1)/s, number of filters) again. The difference between "same" and "valid" will also impact how the model is leaning the border's information. Be specific, what padding is generally to add columns and rows of zeroes to keep the spatial sizes constant after the convolution layer. When it is set to "same", the convolution will use zero paddings; when it is set to "valid", the convolution layer does not use zero paddings and may ignore[1]. The reason I apply "valid" padding is because by looking at the FIG. 1 on lung, the borders of pictures do not significantly determine whether a person is healthy.

Besides the convolution layer and pooling layer, the model also has Flatten layer and Dense layer. Flatten makes the input into a one-dimensional list for input to the dense layers. Then using ReLU activation on the first dense layer, Softmax on the second dense

[1] Stride: the number of pixels we skip when traversing the input horizontally and vertically

CNN LeNet-5 From Lecture note				
Layer	Number of Filters	Padding	Activation Shape	Activation Size
Input Image	-	-	(64, 64, 1)	4096
Con2d(f=5, s=1)	20	Valid	(60, 60, 20)	72000
MaxPool(f=2, s=2)	-	Valid	(30, 30, 20)	18000
Con2d(f=5, s=1)	20	Valid	(26, 26, 20)	13520
MaxPool(f=2, s=2)	-	Valid	(13, 13, 20)	3380
Flatten	-	-	(3380, 1)	3380
Dense	-	-	(500, 1)	500
Softmax	-	-	(10, 1)	10

FIG. 2. 1st CNN LeNet-5 Architecture

layer. The Dense layer manages its weight and will connect all weights between the neurons and inputs.

B. LeNet-5 Tune hyperparameters

The second model is just to modify the hyperparameters on the first model. The detailed architecture below on FIG.3. The first thing I changed is the filter numbers. Instead of using 20 filters, I first set 16 filters then 32 filters to collect information. This change will also change the activation shape as shown in the table. The second change I make is adding a dropout layer with 50% dropout rate on a fully connected network to prevent overfitting. The third change I do is to change the learning rate. Based on Brownlee's article[3], the learning rate may be the most important hyperparameter when configuring neural networks. The higher the learning rate, the bigger the steps are, and the quicker the convergence is. However, the sampling is inferior with a high learning rate, and the optimizer could probably meet some problems to find the local minimum. A slightly lower learning rate will make it easier to find the local minimum. The first model has a default "Adam" optimizer, and the learning rate is 0.001. So I switch to 0.0008 in my second model.

CNN LeNet-5 Tune hyperparameters				
Layer	Number of Filters	Padding	Activation Shape	Activation Size
Input Image	-	-	(64, 64, 1)	4096
Con2d(f=5, s=1)	16	Valid	(60, 60, 16)	57600
MaxPool(f=2, s=2)	-	Valid	(30, 30, 16)	14400
Con2d(f=5, s=1)	32	Valid	(26, 26, 32)	21632
MaxPool(f=2, s=2)	-	Valid	(13, 13, 32)	5408
Flatten	-	-	(5408, 1)	5408
Dense	-	-	(500, 1)	500
Dropout	-	-	(500, 1)	500
Softmax	-	-	(10, 1)	10

FIG. 3. 2nd CNN LeNet-5 Architecture

C. LeNet-5 Tune hyperparameters and Data Augmentation

The third model keeps the same LeNet-5 architecture as the second model but perform data augmentation on both training and test dataset. The intention of doing this is because I want to increase the diversity of the training set and increase training images. Here, I apply both rotation and zooming.

D. VGG-16 from Lecture note

The difference between LeNet and VGG architecture is that LeNet architecture usually has 1 convolution layer and a pooling layer and keeping this, but VGG architecture has 2 or 3 convolution layers and a pooling layer, then again 2 or 3 convolution layers and a pooling layer, and so on. The information on architecture shows on FIG.4.

CNN VGG-16 From Lecture note				
Layer	Number of Filters	Padding	Activation Shape	Activation Size
Input Image	-	-	(64, 64, 1)	4096
Con2d(f=3, s=1)	32	Valid	(62, 62, 32)	123008
MaxPool(f=2, s=2)	-	Valid	(31, 31, 32)	30752
Con2d(f=3, s=1)	64	Valid	(29, 29, 64)	53824
Con2d(f=3, s=1)	64	Valid	(27, 27, 64)	46656
MaxPool(f=2, s=2)	-	Valid	(13, 13, 64)	10816
Con2d(f=3, s=1)	128	Valid	(11, 11, 128)	15488
Con2d(f=3, s=1)	128	Valid	(9, 9, 128)	10368
Dropout	-	-	(9, 9, 128)	10368
MaxPool(f=2, s=2)	-	Valid	(4, 4, 128)	2048
Flatten	-	-	(2048, 1)	2048
Dense	-	-	(512, 1)	512
Softmax	-	-	(10, 1)	10

FIG. 4. 3rd CNN VGG-16 Architecture

IV. MODEL EVALUATION

There are total 4 models, but I fit the original data and the weighted data when performing the model fitting. The weighted data use each label counts divided by the total labels. I utilize Micro F1, Macro F1, and the Kaggle result. The result for original data shows on TABLE I, and the result for weighted label data shows on TABLE II.

Before we getting to know the results, I will introduce the Micro F1 and Macro F1. In the binary classification confusion matrix, we will have a 2*2 matrix, so the F1 score is calculated by $\frac{2*Precision*Recall}{Precision+Recall}$, but in multi-class classification, we do not calculate an overall

TABLE I. Result for CNN Models With Original Data

Model	Micro F1	Macro F1	Recall 0	Recall 1	Recall 2	Kaggle
LeNet-5	0.8914	0.5524	0.9429	0.7818	0.0000	0.56296
LeNet-5 Tune Hyperparameters	0.8861	0.54331	0.9795	0.6165	0.0000	0.58518
LeNet-5 Hyperpara + Augmentation	0.7342	0.5061	0.7493	0.7542	0.2051	0.61481
VGG-16	0.9076	0.5723	0.9738	0.7627	0.0000	0.54814

TABLE II. Result for CNN Models With Original Data

Model	Micro F1	Macro F1	Recall 0	Recall 1	Recall 2	Kaggle
LeNet-5	0.8325	0.6000	0.8887	0.6377	0.5000	0.65185
LeNet-5 Tune Hyperparameters	0.7919	0.5636	0.8573	0.5233	0.6538	0.65925
LeNet-5 Hyperpara + Augmentation	0.6109	0.3275	0.7374	0.0254	0.7436	0.67407
VGG-16	0.7892	0.5487	0.8492	0.5487	0.5897	0.65925

F-1 score. Instead, we use the F-1 score per class. The micro F1 calculates values globally and takes the weight of the metric towards the largest one. While the macro F1 tends to average each class's F-1 score, macro F1 is insensitive to the classes' imbalance.

In addition, I calculate recall rate for each class. The reason for me to do this is because if someone needs treatment, we do not want to say he/she does not need it, especially for label 2 which represent the most severe situation which need the medication or treatment immediately. That is to say, three models with original data are fail to detect the any label 2. It is a bad indicator, therefore it is necessary to weight classes or do the data augmentation.

In summary, we will purpose to use the third model as the final result. It achieves the highest recall rate for label 2 in both cases, also have the highest score on the Kaggle result.

V. INTERPRETATION

What we can see from two tables (I),(II), Micro F1 scores on original data are higher than weighted label data. This is because the Micro F1 scores are averaging to weight metric towards the largest one. Another finding is based on the Table is all Kaggle results on weighted data are higher than original data. This is because the training data we used is highly imbalanced, while the measurement in Kaggle probably has more balance labels for each category.

In the last, we observed that the Kaggle result on the third model, LeNet-5 architecture with tuning hyperparameters and adding data augmentation, in both cases, is the best (See red scores on two tables). It is because after doing data augmentation, the number of training data increases and improves the ability of the fit models to generalize what model have learned to new images.

VI. CONCLUSIONS

In the final analysis, LeNet-5 architecture with tuning hyperparameters and adding data augmentation gives the best result. Surprisingly, the VGG-16 is a bit complicated than Lenet-5, but based on my result, it shows that a complicated model may not necessarily bring a better outcome.

For the data augmentation, I perform both rotation range and zoom range. The main reason for doing this is to increase the diversity of my dataset in training. The hyperparameters I change in LeNet-5 include:

- Modifying the numbers of filters to capture patterns
- Adding a dropout layer for preventing overfitting
- Reducing the learning rate to reach the global minimum of the loss function efficiently

My final result on the original unweighted data has 0.7342 Micro F1 score, 0.5061 Macro F1 score, 0.2051 recall rate for label 2, and 0.61481 in the Kaggle result. I would highly recommend the hospital weigh the class of each label first before using the suggested model. Because the result shows on Kaggle improve from 61481 to 0.67407. Even though the Micro F1 and Macro F1 scores are lower than the original data, we will have the highest recall rate on label 2, 0.7436.

VII. ADDITIONAL REFLECTIONS

This is my first neural network homework. I had many difficulties when doing this problem set, such as I think data augmentation should help increase my model's performance. But I realized I could not use ImageDataGenerator to generate data for a certain class, for example, to generate more images for class 1 and class 2, since we already have enough class

0. I also want to incorporate more models, such as tuning hyperparameters on VGG-16, but I do not have much time left. In the last, I still think it is worth finishing this problem set.

DATA AVAILABILITY

Data is available at: Github

CODE AVAILABILITY

Code is available at: Github

---

[1] A. Gron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 1st ed. (O'Reilly Media, Inc., 2017).

[2] S. Ramesh, *A guide to an efficient way to build neural network architectures- Part II: Hyper-parameter selection and tuning for Convolutional Neural Networks using Hyperas on Fashion-MNIST* (2018).

[3] J. Brownlee, *Understand the Impact of Learning Rate on Neural Network Performance* (2019).