Static Hand Gesture Recognition

Yiran Zhou
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA
yiranz1@andrew.cmu.edu

Lixue Xiao, Lin Li
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA
{lixuex, linl2}@andrew.cmu.edu

Abstract

Hand Gesture recognition provides a modern way of non-verbal communication and thus has a wide range of application in human computer interaction. We propose a deep learning based method using Convolutional Neural Networks (CNN) to identify static hand gestures. As for the dataset, we choose a challenging dataset, NUS-II hand posture dataset, which contains clutter in the background of hand posture images. We provide experimental results demonstrating good performance of our proposed method. After comprehensive tests of our method on 500 images, we achieved a high test accuracy rate of 94.6%.

1. Introduction

Expressive, meaningful body motions involving physical movements of the fingers, hands, arms, head, face, or body are called gestures [1]. Hand gestures are an integral part of communication which play an important role in our daily life. Gesture recognition can be used in sign language recognition, whose goal is to facilitate the machine to be able to "read" the language from the disabled people and thus make their life easier and better. Moreover, gesture recognition has the potential to enable users to interact with their electronic products such as laptop and iPad, without using any external devices such as the keyboard. Therefore, an automatic system for the detection, segmentation and recognition of multi-class hand postures against complex natural backgrounds is an exciting topic for developing intelligent computer vision system.

Hand gestures are either static or dynamic, in our project, we focus on the recognition and classification of static hand gestures. Since Deep learning has won numerous pattern recognition competitions in recent years and it requires less feature engineering, and Convolutional Neural Networks (CNN) can be trained effectively to recognize objects directly from their images with robustness, we decided to apply CNN to classify static hand gestures.

1.1 Dataset Description

NUS-II hand posture dataset [2] has 10 hand postures in complex natural backgrounds with varying hand shapes, sizes and ethnicities. Classes 1 to 10 are named with starting letters 'a' to 'j' respectively. At first we thought those gestures are real sign language, however after careful comparison we found that while some gestures correspond to sign language characters, others are not included in sign language alphabet. So we relabeled those hand gestures to be '0' to '9' in order to prevent confusion. The postures were collected using 40 subjects comprising of both male and female members in age group of 22 to 56 years from various ethnicities [2]. The subjects showed each pose 5 times. The whole dataset contains hand gestures in presence of clutter consisting of 2000 hand posture color images (40 subjects, 10 classes, 5 images per class per subject). Each image is of size 160 × 120 pixels with complex backgrounds. In our project, we will focus our work on the hand postures without human noise.

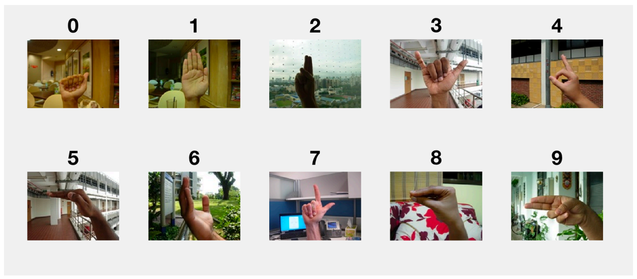

Figure 1: 10 hand postures in NUS-II dataset

From Figure 1, we could easily observe ten different hand gestures in ten different complicated environments. Their backgrounds contain many various elements such as buildings, plants, chairs, etc. which makes it really hard to detect the edge of the hand and recognize the correct hand gesture. There are other static hand gesture datasets such as the datasets proposed by Jochen Triesch [3] and Marcel [4].

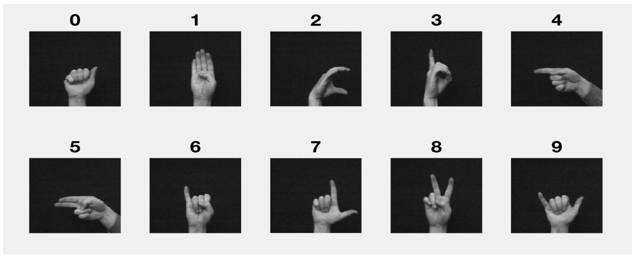

Figure 2: 10 hand postures in Triesch's dataset with dark background

Jochen Triesch's dataset has 10 different classes in uniform light, dark or white background. Because the background is uniform, it is easier to extract the hand from the images using technique like edge detection and perform classification after edge detection and feature extraction. The Marcel dataset has only 6 different classes separated into training data and testing data so the classification task is much simpler. Besides, most of their testing data is in a relatively uniform case, which makes it easier to correctly classify different hand gestures. Compared to those two hand gesture datasets, the NUS-II hand posture dataset is much more complex so we thought it would be more challenging to use that dataset in our final project.

2. Related work

Skin detection is the initial step to detect and recognize human hand postures [5]. However, there are some issues in skin detection, the first is that different ethnicities have different skin features, which make skin-based hand detection difficult. Besides, the complexity of the background is likely to produce erroneous results when doing skin detection. There are several other methods proposed to detect hand postures without skin detection. Ong and Bowden [5] proposed a boosted classifier tree-based method for hand detection and recognition. Kim and Fellner [5] detected fingertip locations in the dark environment.

Van-Toi NGUYEN [6], and other co-authors proposed a method to employ the Kernel Descriptor (KDES) to represent the hand postures. In that way, they could investigate Kernel Descriptor in diffident color channels such as HSV, RGB, Lab to find out which color space is the most suitable for representing hand postures. Meanwhile, they used two datasets to perform an extensive experiment to evaluate the performance of proposed framework. The first dataset was the NUS-II static hand posture dataset and the second dataset was their own dataset collected in the context of human-robot interaction in indoor environment containing cluttered background [6].

Sai Saketh Rambhatla and Rajiv Ranjan Sahay [7] proposed a CNN model to classify hand gestures in NUS-II dataset. The architecture of their CNN model is [CONV-POOL-CONV-POOL-FC-FC] and they trained on 1200 images and tested on 800 images. They were able to recognize hand gestures to a high degree and achieve a test accuracy of 89.1% by using dropout layers and ReLU as the activation function.

3. Proposed Method

The task of our project is to correctly classify static hand gestures, and the first classification method we could think of is k-Nearest Neighbor (k-NN) because it is very simple and intuitive, so we used k-NN algorithm on the dataset as a baseline. Next we implemented Convolutional Neural Network (CNN) model to classify the hand gestures. We started from a simple but classic CNN model LeNet-5, and built our own CNN architecture eventually. We got inspired from Professor Savvides that by implementing edge detection on original images, our k-NN algorithm could perform better, and we noticed that the filtered images produced by the convolution layer in our CNN model captured meaningful features from the original images and could be viewed as images with edge detection, so we also used k-NN method on those filtered images to observe the effects. Data preprocessing was done before we trained our model.

3.1 Data Segmentation

In machine learning, it is very common to learn from one dataset and make predictions on the real testing data. There are mainly three main datasets which are commonly used in different stages of the whole model. The first one is called training dataset which consists of a set of examples used to fit the parameters of the model under supervised learning methods [8]. Successively, the second dataset is called the validation dataset, which is used to provide responses of the observations for the fitted model [9]. The validation dataset provides an unbiased evaluation of the model while tuning the models hyper-parameters [10]. It can be used for regularization by early stopping. We should stop training when the error on the validation dataset increases, which is a sign of overfitting in the training data [11]. Lastly, the third dataset is called test dataset, which is used to provide final evaluation of the model. Because the test dataset is independent of the training dataset. If the model fits on the training dataset also fits on the test dataset, then overfitting is unlikely to happen.

Before we start the experiment, we manually spilt the whole data set into training set and testing set. For each class, we randomly pick 150 images to be training images and 50 images to be testing images, so there are 1500 images in our training set and 50 images in testing set.

For the CNN method, in the training set, we further pick 15 images per class from the training set as validation data so that we could keep track of training and validation error in order to better detect overfitting and under-fitting. As a result, for our CNN models, there are 135 training images per class, 15 validation images per class and 50 test images per class. In total there are 1350 training images, 150 validation images and 500 testing images. The model will only learn from those 1350 training images.

3.2 K Nearest Neighbor Algorithm

K Nearest Neighbors algorithm, or k-NN is a non-parametric method used for classification and regression [12]. A non-parametric method usually makes no assumptions on the underlying data distribution. Because most of the real data in practice does not obey some typical theoretical assumptions made. Furthermore, k-NN model is also a lazy algorithm. This means that the training phase in k-NN model is very fast because it does not use the training samples to do any generalization or prediction.

Specifically, the input of the k-NN algorithm is made up of k nearest training samples in the feature space. As a matter of fact, k-NN is the simplest algorithm among all machine learning methods. This is because k-NN is a type of instance-based learning, where the function only approximated locally and all computation is deferred until classification [13]. As a result, k-NN is very sensitive to the local structure of the dataset.

The algorithm can be divided into two part, the training phase and the classification phase. In the training phase, we are only storing the feature vectors and class label of the training dataset; in the classification phase, we assign a label on an unlabeled vector by finding out the most frequent used label among k nearest training samples to the specific query point [13]. In order to calculate the distance between data points, a commonly used distance is called Euclidean distance.

3.3 Convolutional Neural Network

Convolutional neural network is a class of deep, feed-forward artificial neural networks that has successfully been applied to analyzing visual imagery. An outstanding characteristic of CNN is that it requires much less need for the preprocessing of the input data.[14] It uses a variation of multilayer perceptrons, which are also known as shift invariant or space invariant artificial neural networks, based on their shared-weights architecture and translation invariance characteristics[15][16].

The basic structure and building blocks of CNNs are called layers, each layer has learnable parameters including weights and bias, it takes the output from its previous layer, performs some operations such as convolution and polling, and then produces an output. A softmax function is commonly used in the final layer to produce the probability of the input being in different classes. To train the model, we need to optimize an objective function over the parameters of all the layers and then use stochastic gradient descent (SGD) to update those parameters.

3.4 LeNet-5

LeNet-5 [17] is a simple but very classic CNN model, so we decided to start from using LeNet-5 on the NUS-II dataset. Figure 3 shows the basic architecture of LeNet-5; its structure is [CONV-POOL-CONV-POOL-FC-FC]. The size of input images we gave to LeNet-5 is 160 x 120 instead of 32 x32, and in our project we used 32 filters for the first convolution layer, 64 filters for the second convolution layer. The convolution layer is the core building block of CNNs. In the convolution layer, we would apply convolution operations with filters on input images and the goal is to extract different types of low-level features of that image. Polling layer is commonly used after the convolution layer to reduce the spatial size of feature maps and LeNet-5 implements max-polling which computes the maximum value within each feature window in our polling layer. The size of convolution filters and max pooling filters are 3 x 3 and 2 x 2 respectively. To study the effect of dropout layers, we also added three dropout layers in LeNet-5, which makes the architecture to be [CONV-POOL-DROP-CONV-POOL-DROP-FC-DROP-POOL].

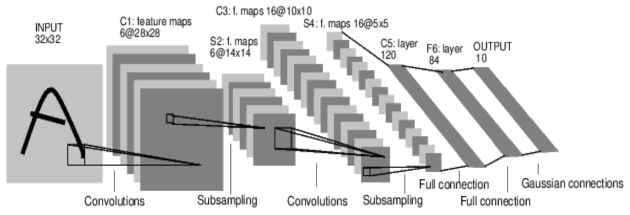

Figure 3: LeNet-5 architecture

The activation layer is used to model the non-linearity of real data and we use ReLU [18]:

$$relu(x) = \max(0, x) \qquad (1)$$

as our activation function since it has been found to work well in vision related problems. For classification task, we use a softmax function to assign probability to each class given the input feature map.

$$p = softmax(Wx + b) \qquad (2)$$

We used categorical cross entropy as our loss function, the goal was to minimize the logarithmic loss for each category. We used SGD to train the model. At every iteration we will take a mini batch of the training images and get the gradient of the parameters, we then update the parameter based on the gradients. We can use stochastic gradient with momentum [19] to update the parameters:

$$\theta = \mu\theta + \alpha\frac{\partial l}{\partial w}, w = w - \theta \qquad (3)$$

where θ accumulates the gradients over the history, μ determines how the gradients from previous steps contribute to current update and α is the learning rate at current step.

3.5 Our Own CNN Model

Due to the complexity of the backgrounds in the images, we need a CNN model that is more complex and deeper than LeNet-5. So in our project, we built our own CNN model. To study the effect of dropout layers, we have two different architectures:

1. Without Dropout Layers: [CONV-POOL-CONV-POOL-CONV-CONV-POOL-FC-FC]
2. With Dropout Layers: [CONV-POOL-DROP-CONV-POOL-DROP-CONV-CONV-POOL-DROP-FC-DROP-FC]

We used four convolution layers in our model, the first three has 32 filters each, and the last has 64 filters, the size the filters used in convolution layers is 3 x 3. We observed from the intermediate outputs that the filter in convolution layer acts pretty much like an edge detector which extracts low-level features from the input images. Figure 4 shows an intermediate output from the first convolution layer in the first epoch in our CNN model.

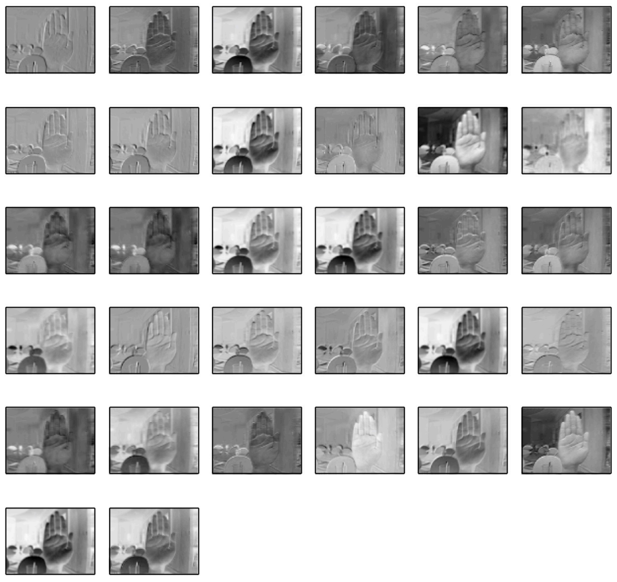

Figure 4: Intermediate outputs from convolution layer

To down-sample the output feature maps from convolution layers, we used three max pooling layers with the size 2 x 2. The activation layer and softmax function are the same as LeNet-5 model. For the parameter settings of SGD optimizer, we used a momentum of 0.9 to better find the global minimum; we started from a learning rate of 0.05 but resulted in training accuracy and validation accuracy both being 0.10 even after 100 epochs, it indicated that we need to take really small steps while training so we finally set the learning rate to be 0.001.

3.6 k-NN Method re-visited

We got inspired from Professor Savvides that by implementing edge detection on original images, our k-NN algorithm could perform better, so at first we decided to use canny edge detection technique to preprocess the hand posture image. However, the result of such method shown in Figure 5 was not very promising. Although it reduced unimportant background object, it also increased the misclassification possibility by zeroing out many other useful features of the hand posture other than the outline of the hand posture. What we did instead was to take advantage of the intermediate result from the 1st convolution layer of our CNN model from the first epoch. Specifically, we picked the 9th filter and used it as an image preprocessing method to preprocess all images and then fed those images into the k-NN model. One example of filtered images is shown in Figure 6.

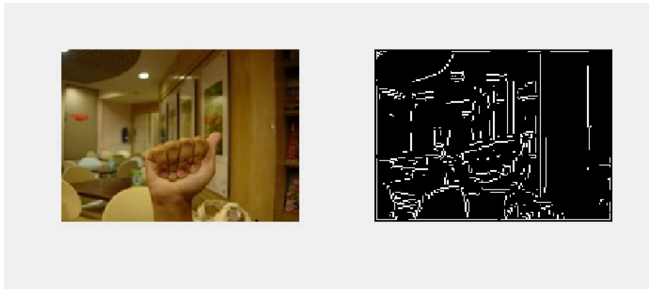

Figure 5: Canny edge detection

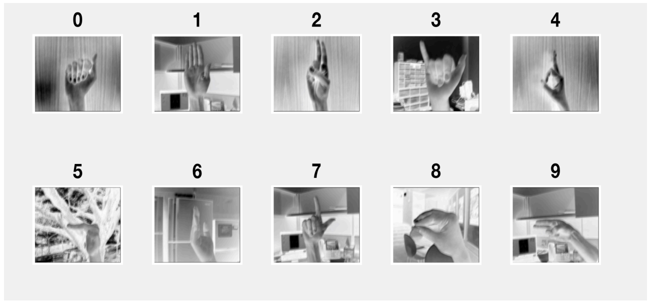

Figure 6: Intermediate outputs for 10 classes

3.7 Dropout

Dropout is a regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data [20]. The term ‘dropout’ refers to dropping our units (both hidden and visible) in a neural network [21]. Specifically, dropout means ignoring units, such as neurons, which is chosen randomly during the training phase. Therefore, for a specific node in a training stage, it is either dropped or kept with the probability of p and 1-p correspondingly.

The main reason for using dropout layers in the CNN model is to prevent the situation of overfitting. A fully connected layer contains most of the hyper-parameters, and the inter-dependency between the neurons during the training phase can lead to overfitting the training data. To illustrate the usage of dropout in detail, we explain the specific approach used in the training phase. In training phase, for each iteration, we zero out a random fraction of nodes and its corresponding activations in each hidden layer of each training sample.

With dropout, we can force a neural network to learn more robust features. Even though dropout may double the iteration time required to converge, each epoch costs less time during the training phase.

3.8 Data Augmentation

One of the major reasons for overfitting is that there is not enough data to train the network, and data augmentation has been proved to be an effective way to counter overfitting [22]. Data augmentation increases the size of training data and it acts like regularization which is "any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error." [23] so with data augmentation we are able to decrease a model's variance.

In our project, the CNN model could only see and learn from 1350 training images so we used data augmentation to increase the size of our training data to prevent overfitting. The main idea of data augmentation is to artificially create more images from the images you already have by changing the width, height, orientation etc. of the image. One example of data augmentation is shown in Figure 7. For each training image we used data augmentation to produce 9 more images, which means we have 13500 training images after data augmentation, 1350 images per class.

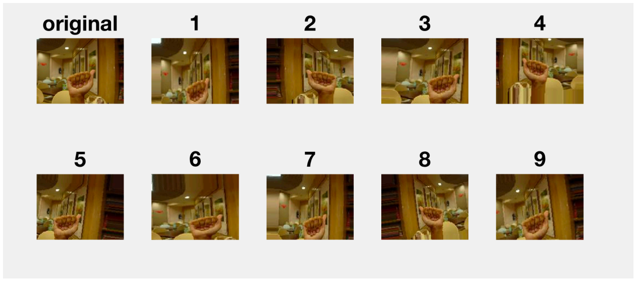

Figure 7: Images after data augmentation

4.Experiment and Results

In our project, we conducted ten experiments in total. First we implemented k-NN algorithm on the dataset, then we used LeNet-5 model to conduct four experiments: one without any dropout layers and data augmentation; one with three dropout layers only; one with data augmentation only and one with both dropout layers and data augmentation. Next we built our own CNN model and conducted the same four experiments as LeNet-5. Finally, we used the intermediate outputs from the first convolution layer in our CNN model as the "images with preprocessing", and then used k-NN algorithm on those images. The final results of the ten experiments are shown in Table 1.

Algorithm	**Training set**	**Validation set**	**Testing set**	**Test accuracy**
k-nn	1500	/	500	26.00%
k-nn + data preprocessing	1500	/	500	39.20%
LeNet-5	1350	150	500	40.40%
LeNet-5 + Dropout	1350	150	500	58.20%
LeNet-5 + Augmentation	13500	150	500	66.62%
LeNet-5 + Both	13500	150	500	77.80%
OurModel	1350	150	500	70.20%
OurModel + Dropout	1350	150	500	75.62%
OurModel + Augmentation	13500	150	500	90.06%
OurModel + Both	13500	150	500	94.60%

Table 1: Experiment results

4.1 Analysis for k-NN

k-NN is an instance-based learning, in other words, it is based on the memorization of the dataset. As a result, the number of parameters is unbounded and will grow as the size of the dataset increases. What's more, there's not even a model associated with the learning process and hence the cost of training is basically zero, and all the cost is spent on the computation of the prediction. k-NN is also a lazy-learning approach, it uses the local neighborhood to make prediction. This means that if we change the distance function, we can change the classification result. In our approach, we use Euclidean distance to compute the distance between data points.

The parameter k is also a factor which influence k-NN performance. As we know from the algorithm, it uses majority voting classification approach. A drawback of this method occurs when the class distribution is skewed. In other words, those frequent classes tend to dominate the prediction of the newly-come sample points. One way to avoid the problem is to assign weight to each class, another way is to use abstraction in data representation. In our algorithm, we choose k = 1, which has a higher classification accuracy than larger k values.

As we can see from the result, k-NN has the lowest performance among the ten methods. k-NN produced a test accuracy of 26.0%, which is only 16% better than making a random guess (10% accuracy). The reason is because the k-NN model focuses too much on the background instead of the hand gesture itself. After we fed the filtered images produced by our CNN model's convolution model to k-NN model, the k-NN model could focus more on the shape of the hand instead of the colors or the shape of the background. So with proper data preprocessing, k-NN algorithm was able to increase the test accuracy to 39.20%, but this result was still not satisfying and it required much more effort.

4.2 Analysis for LeNet-5

Figure 8 shows the accuracy curves produced by LeNet-5 model. As we can see from the accuracy curves, without using any dropout layers and data augmentation, the LeNet-5 model can only produce a test accuracy of 40.40% and there is a serious overfitting problem. We think the overfitting occurs due to the lack of our training samples—for each class, there are only 135 sample images for the CNN model to learn. This makes it much easier for the model to memorize the training images instead of learning from them. As a result, we need to find a way to address overfitting issue.

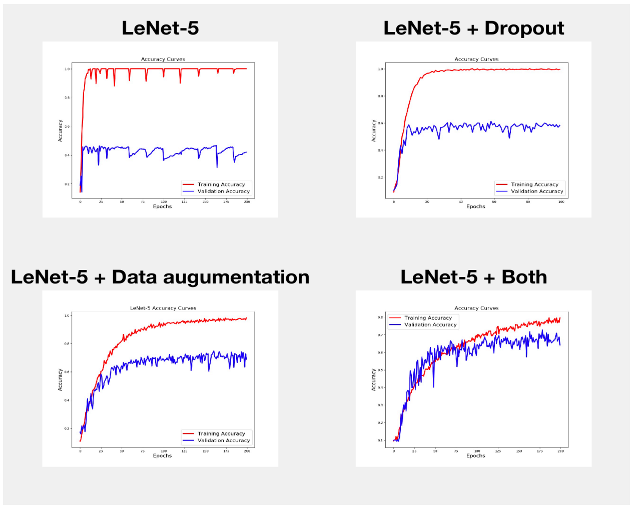

Figure 8: LeNet-5 Accuracy Curves

To avoid overfitting, we first added three dropout layers with a dropout rate of 25%, which means we forced the model to randomly drop out 25% of the nodes in those layers during each epoch, thus prevent the model from memorizing the training images. We can see from the result that by implementing dropout layers, the overfitting issue has been lightened and the LeNet-5 model is able to produce a test accuracy of 58.20%.

Data augmentation performed much better than dropout layers, it reduced overfitting even further and achieved a test accuracy of 66.62%. Since we have ten times more training data after data augmentation, it is more difficult for the CNN model to memorize and thus can address the overfitting problem.

From the accuracy curve we can see that when we implemented LeNet-5 model with both data augmentation and dropout layers, we were able to further boost the performance of LeNet-5 and reached a test accuracy of 77.80%. And we can see from the loss curve in Figure 9 that the combination of data augmentation and dropout layers can reduce the loss function below 1.0. As a result, the combination of data augmentation and dropout layers can address overfitting issue very well. However, the test accuracy is still not high enough, and we think the problem of LeNet-5 is that it is a shallow neural network, but the NUS-II dataset is a very complex data set which contains hand gesture images with clutter backgrounds, so we might need a deeper neural network with more parameters to learn and more low-level features to extract.

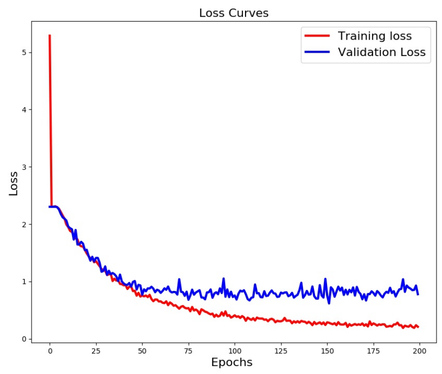

Figure 9: LeNet-5 + Both Loss Curves

4.3 Analysis for Our Model

The LeNet-5 model could reach a test accuracy of 77.80% on the dataset at most, which is not very satisfying, so we are motivated to build our own CNN model. Due to the complexity of the dataset, we decided to add more layers to our CNN model to make it deeper and thus extract more low-level features and learn more from the training images. From the accuracy curves shown in Figure 10 we can see that even without any dropout layers and data augmentation, our model was able to produce a test accuracy of 70.20%. This result outperforms the result produced by LeNet-5 model by a huge amount (70.20% compare to 40.40%) and it even outperforms the result produced by LeNet-5 with data augmentation. Therefore, we believe our CNN architecture can learn better and recognize testing images better and thus is more effective on the NUS-II dataset, so we continued our experiments using this model.

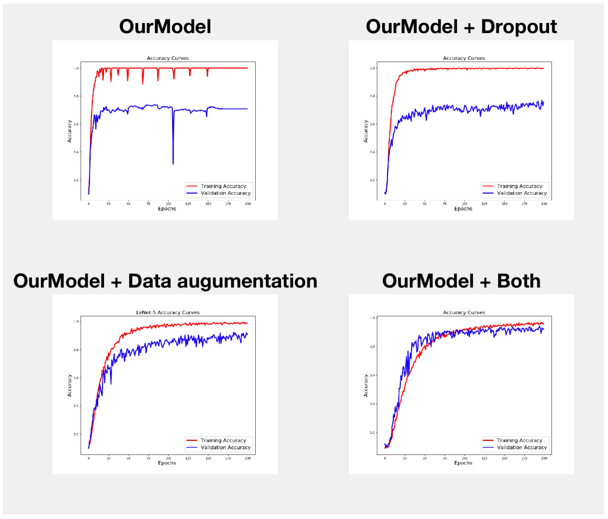

Figure 10: Our Model Accuracy Curves

With the help of dropout layers, our model was able to achieve 75.62% test accuracy, however, with the help of data augmentation, our model was able to correctly classified 90.06% of the testing images. So in our case, data augmentation has much more effect than dropout layers on addressing overfitting problem. We think there are several reasons for that phenomenon: (1) The dropout rate we used in our model is 0.25, so we still keep most of (75%) the nodes after each dropout layer thus the effect of dropout is relatively small. We decided not to use a high dropout rate because if the dropout rate is too high then there will be an under-fitting problem. (2) The most obvious problem in the dataset is the lack of training samples. There are 200 images per class originally, but after we segment the dataset there are only 135 training images per class so the size of our training data is very small, thus it is easier for the model to remember those training samples. Data augmentation can address this major issue by increasing the size of the training set from 1350 to 13500, so data augmentation could make it more difficult for the model to memorize training samples. For those two reasons above, it is reasonable to see that data augmentation outperforms dropout layers in our project.

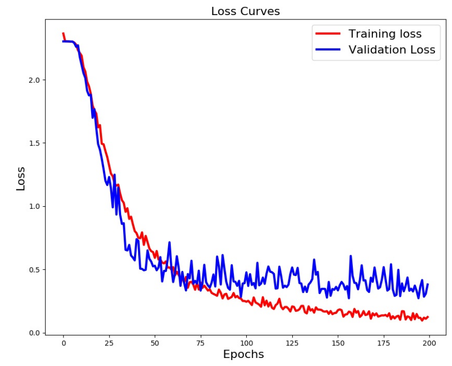

Figure 11: OurModel + Both Loss Curves

When we implemented our model with both data augmentation and dropout layers, the result was quite satisfying—our model was able to achieve 94.60% test accuracy, which means our model only misclassified 27 test images out of 500. Although our CNN model is deeper than the classic LeNet-5 model, with the help of data augmentation and dropout layers we could still prevent overfitting problem very well. From the loss curve shown in Figure 11 we can see that the combination of data augmentation and dropout layers can reduce the loss function below 0.5, which is lower than the best a LeNet-5 model could achieve (below 1.0), thus it indicated that our model is more effective in recognizing static hand gestures in NUS-II hand gesture dataset.

5. Conclusion and Future Work

5.1 Conclusion

The baseline method k-NN produced a test accuracy of 26.0%, which is only 16% better than making a random guess (10% accuracy). The filters in the convolution layer act very much like edge detectors, so we could use them to preprocess our data and then fed those data to the k-NN algorithm. With the help of proper data preprocessing, k-NN is able to perform better but the result is still not good enough, so k-NN is really not suitable in hand gesture recognition.

LeNet-5 is one simple but classic CNN model, one could use it as a good starting point for image classification tasks, but it is too simple and shallow to deal with images with clutter backgrounds. Therefore, LeNet-5 model is not a proper choice for static hand gesture recognition task using NUS-II hand posture dataset.

Our own CNN model outperforms LeNet-5 model and can reach a very high test accuracy of 94.60%. So our proposed CNN model has been demonstrated to be able to recognize static hand gestures to a high degree of accuracy on the NUS-II hand gesture dataset. One major advantage of our proposed CNN method is that with CNN we do not need to segment the hand in the input image or explicitly extract any features. Once we have trained the model, we can easily save the model and use it to classify any unknown images.

Both dropout layers and data augmentation are useful tools to address the overfitting problem. In the NUS-II hand gesture dataset, data augmentation turns out to be more effective than adding dropout layers. We could get the best result by combining data augmentation and the usage of dropout layers.

5.2 Future Work

There are several things we could do in the future. First of all, we could use our model for real sign language recognition. Since the hand gestures in the dataset we used are pretty much the same as sign language characters, we believe our model could perform effectively in sign language recognition. Secondly, we could test our model on hand gestures with human noise, we think those images will be more challenging because it will be more difficult for the CNN model to learn and judge if there is a human face behind a hand posture. As a result, we might need more complex CNN architecture to fulfill that task. Since we only focus on static hand gesture recognition in our project, so in the future we could extend our work to handle dynamic hand gesture recognition and effectively recognize hand gestures in real time.

6. References

[1] A.Mohanty, S.S.Rambhatla, and R.R.Sahay, “cx Deep Gesture: Static Hand Gesture Recognition Using CNN”, in Proceedings of International Conference on Computer Vision and Image Processing, pp.449-461,Spring,2017.

[2] https://www.ece.nus.edu.sg/stfpage/elepv/NUS-HandSet/

[3] J. Triesch and C. Von Der Malsburg, “Robust classification of hand postures against complex backgrounds,” in *fg*. IEEE, 1996, p. 170.

[4] S. Marcel, “Hand posture recognition in a body-face centered space,” in *CHI ’99 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA ’99. New York, NY, USA: ACM, 1999, pp. 302–303.

[5] P. K. Pisharady “Attention based detection and recognition of hand postures against complex backgrounds, ” International Journal of Computer Vision, vol.101,no.3,pp.403-419,2013.

[6] Van-Toi NGUYEN, , Thi-Lan LE , Thanh-Hai TRAN , Remy MULLOT , Vincent COURBOULAY, “Hand posture recognition using Kernel Descriptor”.

[7] Sai Saketh Rambhatla and Rajiv Ranjan Sahay “Deep Gesture: Static Hand Gesture Recognition Using CNN”, Proceedings of International Conference on Computer Vision and Image Processing pp.449-461

[8] Ripley, Brian (1996). Pattern Recognition and Neural Networks. Cambridge University Press. p. 354. ISBN 978-0521717700.

[9] James, Gareth (2013). An Introduction to Statistical Learning: with Applications in R. Springer. p. 176. ISBN 978-1461471370.

[10] Brownlee, Jason. "What is the Difference Between Test and Validation Datasets?". Retrieved 12 October2017.

[11] Prechelt, Lutz; Geneviève B. Orr (2012-01-01). "Early Stopping — But When?". In Grégoire Montavon; Klaus-Robert Müller. Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science. Springer Berlin Heidelberg.pp. 53–67. ISBN 978-3-642-35289-8. Retrieved 2013-12-15.

[12] Altman, N. S. (1992). "An introduction to kernel and nearest-neighbor nonparametric regression". The American Statistician. 46 (3):175-185. doi:10.1080/00031305.1992.10475879.

[13] https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

[14] LeCun, Yann. "LeNet-5, convolutional neural networks". Retrieved 16 November 2013.

[15] Zhang, Wei (1988). "Shift-invariant pattern recognition neural network and its optical architecture". Proceedings of annual conference of the Japan Society of Applied Physics.

[16] Zhang, Wei (1990). "Parallel distributed processing model with local space-invariant interconnections and its optical architecture". Applied Optics. 29(32): 4790–7. Bibcode:1990ApOpt..29.4790Z. doi:10.1364/AO.29.004790. PMID 20577468.

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition.*Proceedings of the IEEE*, november 1998.

[18] https://en.wikipedia.org/wiki/Rectifier_(neural_networks)#cite_note-Hahnloser2000-1

[19] http://ufldl.stanford.edu/tutorial/supervised/OptimizationStochasticGradientDescent/

[20] Hinton, Geoffrey E.; Srivastava, Nitish; Krizhevsky, Alex; Sutskever, Ilya; Salakhutdinov, Ruslan R. (2012). "Improving neural networks by preventing co-adaptation of feature detectors". arXiv:1207.0580 [cs.NE].

[21] "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". Jmlr.org. Retrieved July 26, 2015.

[22] http://www.learnopencv.com/image-classification-using-convolutional-neural-networks-in-keras/

[23] Goodfellow et al’s Deep Learning, Section 5.2.2