ASL Fingerspelling Recognition Using Convolutional Neural Networks

Brian Neldon
Khaled Jabr

Abstract

Sign language fingerspelling is the main building block for designing and developing an American Sign Language (ASL) interpreter. In this project, we used the University of Massey ASL fingerspelling dataset containing five different fingerspelling to build an ASL Fingerspelling classification system by training a convolutional neural network (CNN) on it. Our CNN had an architecture similar to AlexNet, and we trained it on 32 classes of the alphabet. Having used different data splitting configurations and hyper parameters, we achieved our best result of 83.37% accuracy with a training set of 4 signers and test set of 1 signer. We conclude that CNNs are an appropriate approach for such a challenge, and that an intelligent data split and training could substantially increase learning and performance.

1. Introduction

It is estimated that there are around 70 million deaf people around the world who use sign language as their main way of communication. Worldwide, there are at least 25 sign languages, one of which is American Sign Language (ASL). The challenge of developing an autonomous ASL interpreter has been a long sought goal by the linguistics and computer science community. With recent advances in image processing, pattern recognition, and convolutional neural networks, the challenge has received much attention and research in the past few years, and there has been some notable advancements and results. In doing this project, we were motivated by these advances, and by the social benefit of such project, as it not only contributes to the academic field but also to the deaf community.

In this project, our domain is the ASL fingerspelling, which is the classification of the static ASL alphabet consisting of letters and numbers. ASL fingerspelling is a challenging problem, and it requires the understanding of the underlying informational model of ASL and its edge cases, alongside with limits to which it can be automated and modeled using computer science techniques. This report's structure and scope will be similar to that of a research paper. The report starts by highlighting related work, then it shifts attention to our work, explaining our approach and methodology, then results, along with an analysis and a discussion.

Most of the work related to ASL fingerspelling has been on image and pattern recognition. Bergh et al. [1], proposed a hand gesture system that extracts Haar wavelets features, and then classifies the image by performing a search on a database of feature, finding the nearest match. Their system was developed over six features only. Starner et al. [2], proposed a system to hand-gesture recognition that used a Bayesian network, using Hidden Markov Models and 3D gloves to track hand movements, and were able to achieve 99.2% accuracy of classifying ASL fingerspelling on their test dataset. Others works used those advancements to apply them directly to the ASL fingerspelling challenge. Pigou et al. [3], developed a system for recognizing Italian sign language with 20 classes, achieving an accuracy of 95.68% of a colored image dataset, though in his paper, there was no split between training, validation, and testing data . Isaac et al. [4], proposed a system for recognizing ASL finger spelling using neural networks that train on wavelet features . Pugeault et al. [5], proposed a real-time system for ASL fingerspelling recognition using gabor filters, and random forests. Their system was able to correctly classify 24 classes with a max accuracy of 75%. In their work, they used both color and depth images for ASL fingerspelling. Kang et al. [6] , proposed a system using convolutional neural networks and trained only on static depth images that were acquired using a depth camera. Their system made use of image processing techniques and CNNs to classify 31 classes, achieving an accuracy of 83% for a new signer .

Our project is similar to previous works in the fact that we used convolutional neural networks (CNN) to image recognition. CNNs have been proven to perform well with images to extract important features that are learned, and then used to classify ASL letters. For our project, we used an architecture based on the AlexNet CNN, with the ASL image dataset from the University of Massey to train on. We also used three different data split configurations, inspired by Kang et al, and two different learning rates. We were able to achieve an accuracy of 83.37%. We concluded that Convolutional Neural Networks are the best approach to learn the most important features on ASL images to be able to correctly classify them. We have also concluded that splitting the dataset intelligently could increase the accuracy results.

2. Approach

2.1 Preprocessing

We acquired our training from a public dataset published by the University of Massey. The dataset consists of 2524 close-up images of ASL fingerspelling with black backgrounds. All the images were different sizes, so preprocessing was necessary to provide the images as inputs to the CNN. We scaled all the images to 28 by 28 pixels, while preserving the aspect ratio between the width and height of the image. We also padded the images borders to help our CNNs detect border and edge features.

After processing our images, we did further processing to deal with two edge cases in ASL fingerspelling. First case: in ASL, letters **J** and **Z** are not static, thus they require more temporal information to represent and require more attention to train; thus, we removed them from the training dataset. This is a common issue, and all the related works that we discussed dealt with it the same way. Second Case: number **0** and letter **o** have almost the same representation in ASL. This is also the case for number **2** and letter **V**. To handle, we decided to deal with all instances of letters **O** and **V**, as examples for numbers **0** and **2**, respectively.This could have been done the other way around, it is a matter of preference. Thus our system classifies a total of 32 classes: letters A-Z except for {J,Z,O,V}, and numbers 0-9.

2.2 Architecture

Once we obtained our preprocessed images, they were ready to train the convolutional neural network. The architecture of our net can be seen in **Figure 1**. The first layer is the original image, with a volume of 28 by 28 by 3 representing the width, height, and RGB channels of the image, respectively. This image is then convoluted with 64 different kernels with stride 1. At the 'conv1' layer, the height and width has not changed due to the stride size of 1, but now there are 64 layers corresponding to the 64 kernels applied to the image. After convolution, a max-pooling is done with a stride of 2 so that the image height and width shrink by half. This significantly reduces the processing time. Then local response normalization (LRN) and dropouts of probability 0.8 are applied. The cycle of convolution, maxpooling, LRN, and dropouts is repeated two more times with 128 and 256 convolutional kernels.

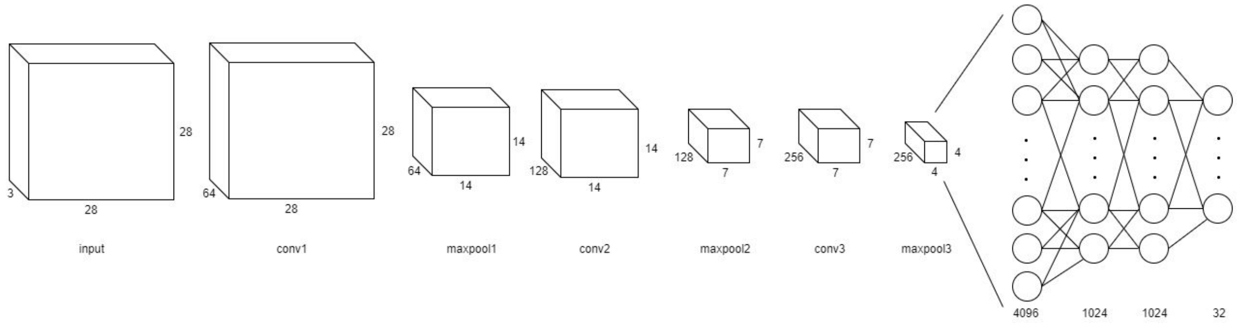


Figure 1: CNN Architecture

After the convolutional layers, three fully connected layers classify the image as one of the 32 gestures. The input to the fully connected layers is the third convolutional layer reshaped into a 4096 by 1 array. Then there are two hidden 1024-node layers, and the output layer with 32 nodes. The outputs are based on probabilities from the softmax distribution.

2.3 Training

The way the networks learns is by updating the weights in the convolutional kernels and in the fully-connected network. We decided to use gradient descent as the update rule with cross-entropy being our cost function. We also had batch updating with a batch size of 20. The algorithm ran 90 epochs through the training data but would stop if it detected an upward trend in the error on the validation data. This helped to prevent overfitting.

3. Experiments and Methodology

We decided to conduct several experiments concerning the learning rate of the update rule and the split between training, validation, and test sets. The learning rates we used were 0.01 and 0.0001. The different splits we used were three signers in training, one in validation, and one in testing (notated 3-1-1); four signers in training and one in testing (notated 4-1); and a random uniformly distributed selection of 60% data for training, 20% for validation, and 20% for testing (notated 60-20-20). In total, we ran six experiments: each of the three data splits using both learning rates. We decided to measure performance on these experiments with two metrics: top-1 accuracy, the percentage that the learning model correctly classified the gesture, and top-5 accuracy, the percentage that the correct class was in the top five most likely classes given by the learning model.

4.Results

Overall, we achieved better results than we expected, shown in **Table 1**. Our best-performing learning model was Experiment 2, which on the test set achieved accuracy of 83.37% and top-5 accuracy of 99.79%. Shown in **Figure 2**, it trained for all 90 epochs since it did not have a validation set. Our worst-performing model was Experiment 4, which had 40.84% top-1 accuracy and 78.74% top-5 accuracy. It stopped training at epoch 50 when the validation error increased, as shown in **Figure 3.**

Experiment	**Learning Config**	**Learnin g Rate**	**Test Top-1 Accuracy**	**Test Top-5 Accuracy**

1	3-1-1	0.001	0.566316	0.901053
2	4-1	0.001	0.833684	0.997895
3	60-20-20	0.001	0.698947	0.966316
4	3-1-1	0.0001	0.408421	0.787368
5	4-1	0.0001	0.522105	0.886316
6	60-20-20	0.0001	0.492632	0.867368

Table 1: Top-1 and Top-5 Accuracies

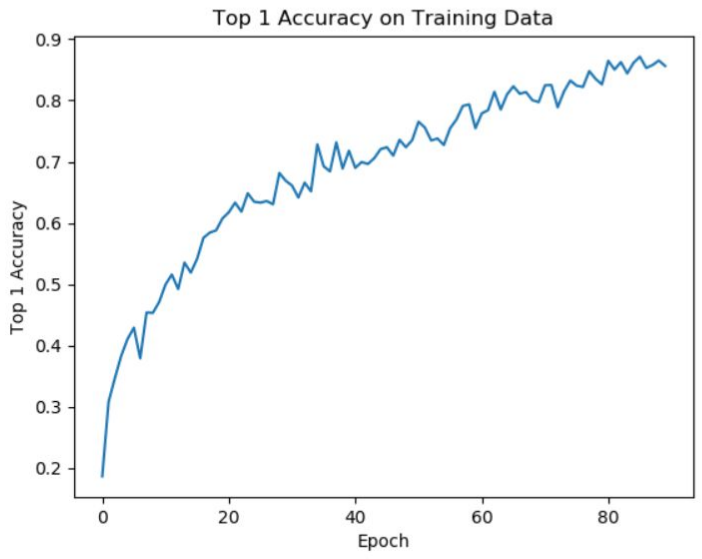


Figure 2: Top-1 Accuracy Learning Curve (Experiment 2)

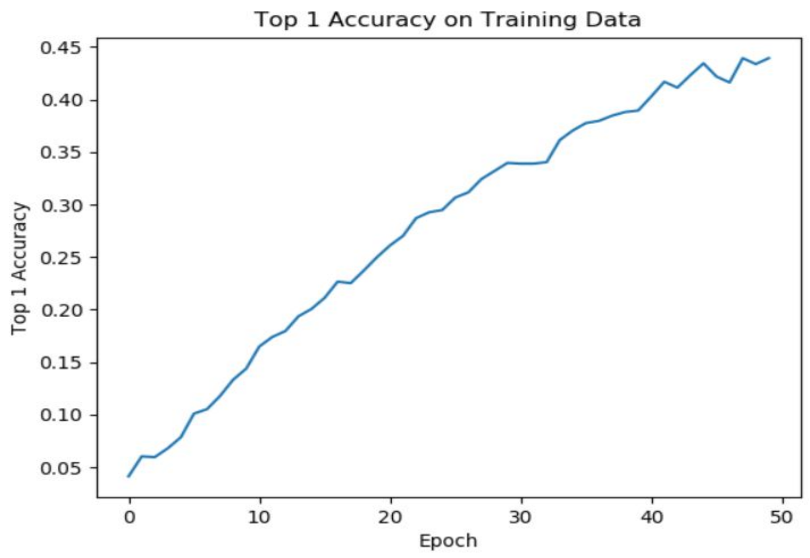


Figure 3: Top-1 Accuracy Learning Curve (Experiment 4)

All of our image processing, data manipulation, training, and metrics were implemented in Python 3.6 with TensorFlow, numpy, opencv, and matplotlib.

6. Discussion

Since our dataset contained only 2524 images, we were concerned that it would not be able to generalize very well. We believe it performed well because the test data was very similar to the training data, with the black background and little noise. If the model were tested on more noisy images, such as a photo of someone gesturing in a coffee shop, the model might require many more training examples to better generalize the classes.

The data splits were interesting because, while normally we would split randomly by percentage, we hypothesized that training on certain signers and testing on a different signer would be more realistic to applications for this model. For instance, the model would have to classify a completely new person's gestures when used in practice. The model trained on four signers and tested on the fifth turned out to perform the best, for reasons we will discuss below.

We believe our worst result, Experiment 4, performed badly because of two factors. First, the learning rate was too low to effectively update the weights in the 50 epochs that it trained. The 0.0001 learning rate had obvious negative effects on the performance of all the models that used it. Second, it may have stopped too early because of the error increase on the validation set. It could have been some stochasticity in the system that made the validation error increase instead of a general trend of overfitting. The fact that our best result trained for 90 epochs supports this hypothesis.

Our best result possibly had a better learning rate and a better data split. The 0.001 learning rate on average had a difference of 22.5% more accuracy than its 0.0001 counterpart. As well, we believe that having four instead of three signers in the training set was beneficial to the model and allowed it to generalize to the test set better.

Looking to previous works, many of them classified a smaller amount of gestures. However, Kang et al. in [6] used CNNs on depth images to determine 31 classes with an accuracy of 83%. Although variability in datasets rules out direct comparison, our results achieved over 83% accuracy on 32 classes of two-dimensional images.

7. Conclusions

In this project, we confirmed our hypothesis of using convolutional neural networks to accurately classify two-dimensional images of ASL signs. We conclude that performing the training using different splitting configurations, especially with a limited dataset, could affect the amount of learning a CNN can do, thus affecting its accuracy. In addition, we found that the learning rate is critical for performance. A too-small learning rate will never converge the algorithm. However, a too-large learning rate may overshoot the global minima of the cost function, leading to an oscillating state that also does not converge. Finding the right learning rate can be achieved with careful experimentation and search.

8. Future Work

For future work, there is room for improvement and addition to our project. First improvement would be to fix the issues of resemblance between letters **0** and **V** and numbers **0** and **2**, which leads us to our second possible improvement: getting more training data. For this project we had a public, and relatively small, dataset; however, there are others such as the ASL dataset by Surrey University, which contains 65000 images. We believe that acquiring a larger dataset, or a merge of multiple datasets, would improve the learning ability of our CNNs, and would make it more able to differentiate more subtle differences, such as the one between letter **O** and number **0**. We did not use the Surrey dataset for this project because the dataset was very noisy and had many edge cases, which we believed would have impeded our limited timeline.

Lastly, once a good ASL fingerspelling system is implemented, we could be able to train a system to classify a sequence of ASL signs, and learn to classify it to the nearest possible word, for a given dataset. This would be great start for designing and implementing an autonomous ASL interpreter.

Bibliography

[1] T.Starner and A. Pentland. Real-Time American Sign Language Recognition from Video Using Hidden Markov Models. Computational Imaging and Vision, 9(1); 227-243, 1997.

[2] M. Van den Bergh and L. Van Gool. Combining rgb and tof cameras for real-time 3d hand gesture interaction. In Applications of Computer Vision (WACV), 2011 IEEE Workshop on, pages 66–72, Jan 2011.

[3] L. Pigou, S. Dieleman, P.-J. Kindermans, and B. Schrauwen. Sign language recognition using convolutional neural networks. In Computer Vision - ECCV 2014 Workshops, pages 572–578, 2015.

[4] J. Isaacs and S. Foo. Hand pose estimation for american sign language recognition. In System Theory, 2004. Proceedings of the Thirty-Sixth Southeastern Symposium on, pages 132–136, 2004.

[5] N. Pugeault and R. Bowden. Spelling it out: Real-time asl fingerspelling recognition. In Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on, pages 1114–1119, Nov 2011.

[6] Kang, B., Tripathi, S., & Nguyen, T. Q. (2015). Real-time sign language fingerspelling recognition using convolutional neural networks from depth map. 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR). doi:10.1109/acpr.2015.7486481[1] Kang, B., Tripathi, S., & Nguyen, T. Q. (2015). Real-time sign language fingerspelling recognition using convolutional neural networks from depth map. 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR). doi:10.1109/acpr.2015.7486481

APPENDIX

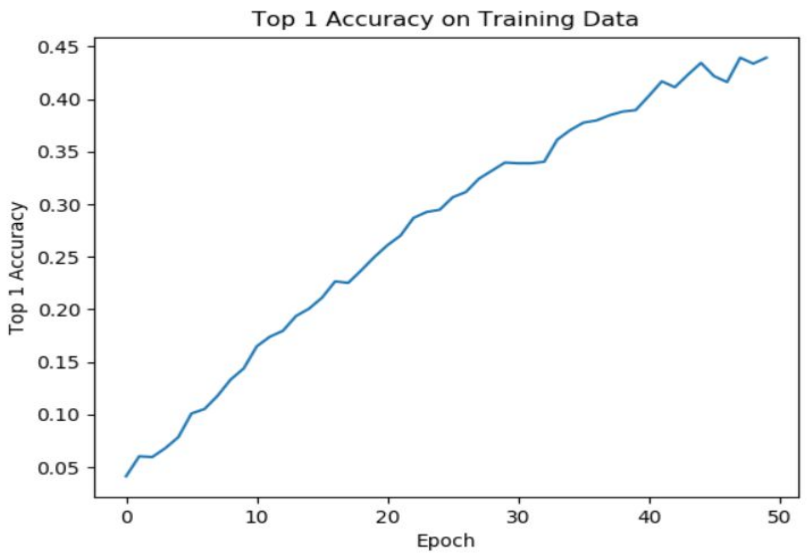
Experiments results

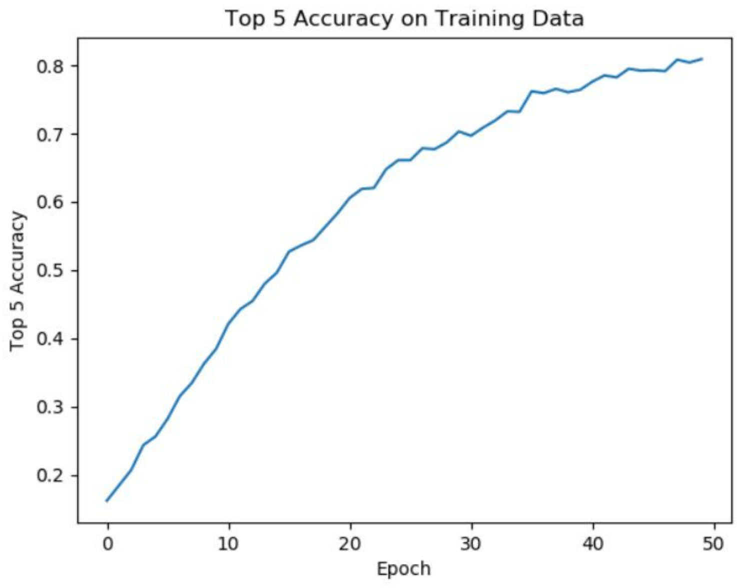
Key :

Experiment #	Learning Configuration	Learning Rate

1	LC: 3-1-1	LR = 0.0001

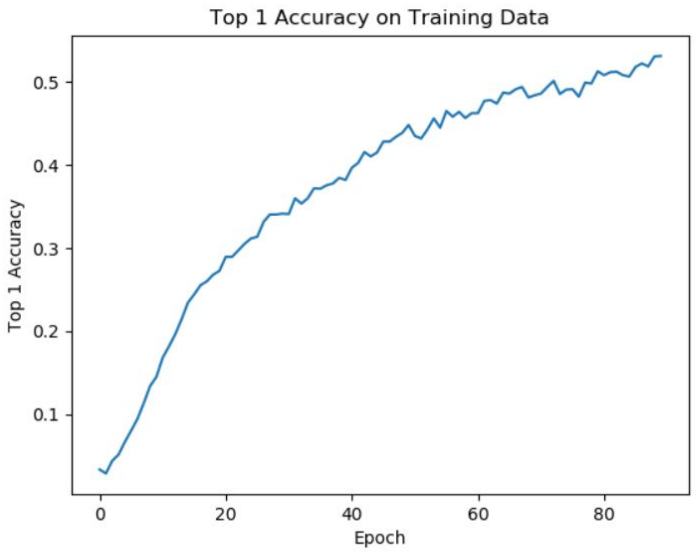
Top 1 Accuracy on Training Set = 0.4392982456140351
Top 5 Accuracy on Training Set = 0.8091228070175439
Accuracy on Validation Set = 0.4147368371486664
Validation accuracy has decreased (0.41684210300445557 -> 0.4147368371486664)
Stopping training after Epoch 50 to prevent overfitting.
Top 1 Accuracy on Test Data: 0.408421
Top 5 Accuracy on Test Data: 0.787368

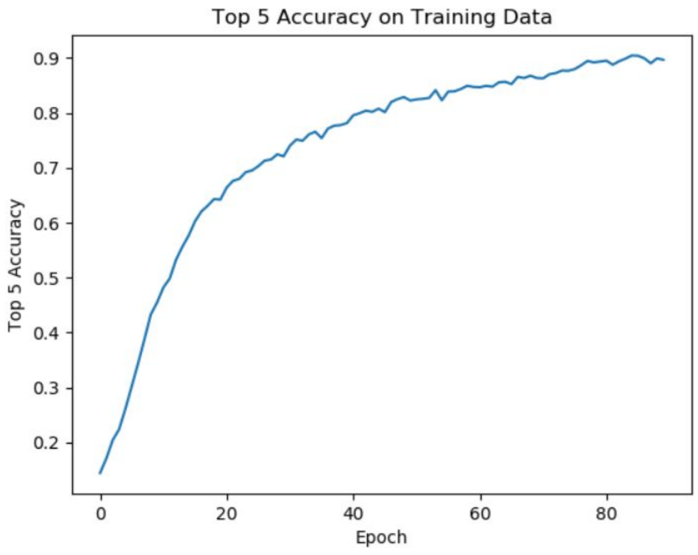




2	LC: 4-1	LR = 0.0001

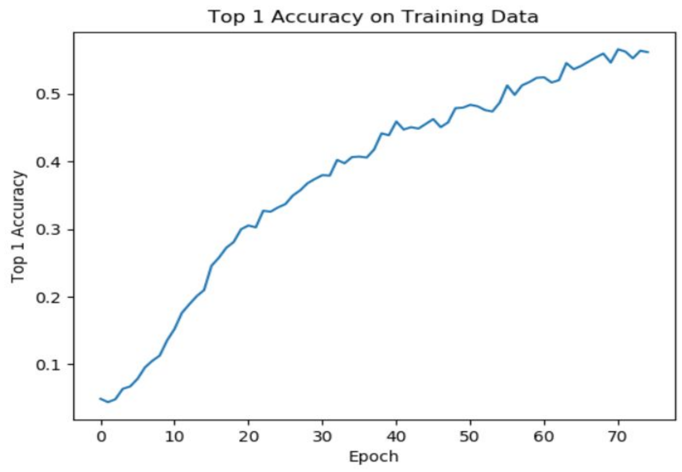
Top 1 Accuracy on Training Set = 0.531578947368421
Top 5 Accuracy on Training Set = 0.8968421052631579
Top 1 Accuracy on Test Data: 0.522105
Top 5 Accuracy on Test Data: 0.886316

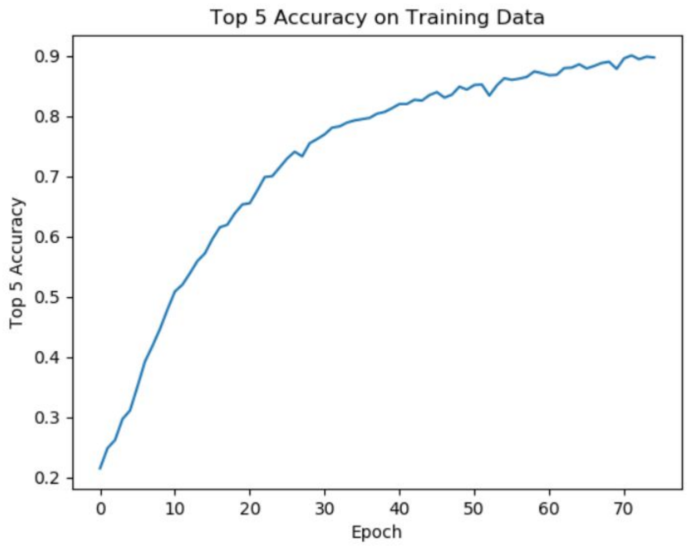




3	LC: Percentage	LR = 0.0001

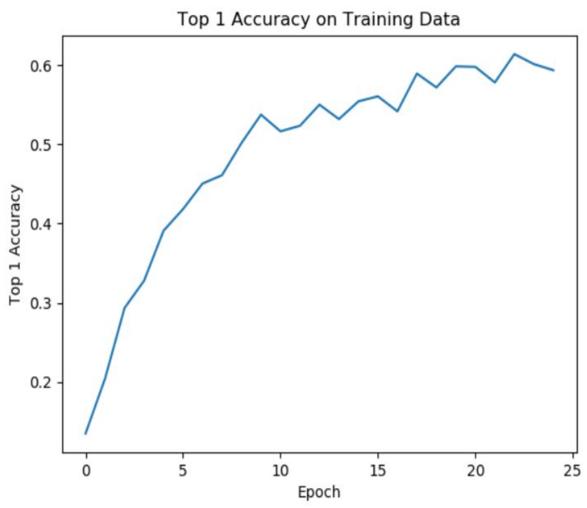
Top 1 Accuracy on Training Set = 0.5614035087719298
Top 5 Accuracy on Training Set = 0.8975438596491228
Accuracy on Validation Set = 0.503157913684845
Validation accuracy has decreased (0.5073684453964233 -> 0.503157913684845)
Stopping training after Epoch 75 to prevent overfitting.
Top 1 Accuracy on Test Data: 0.492632
Top 5 Accuracy on Test Data: 0.867368

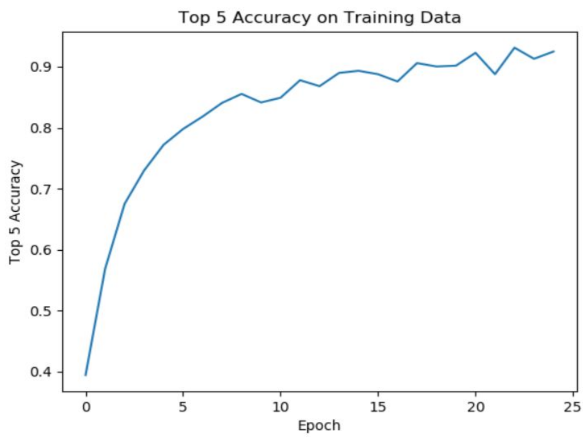




4	LC: 3-1-1	LR = 0.001

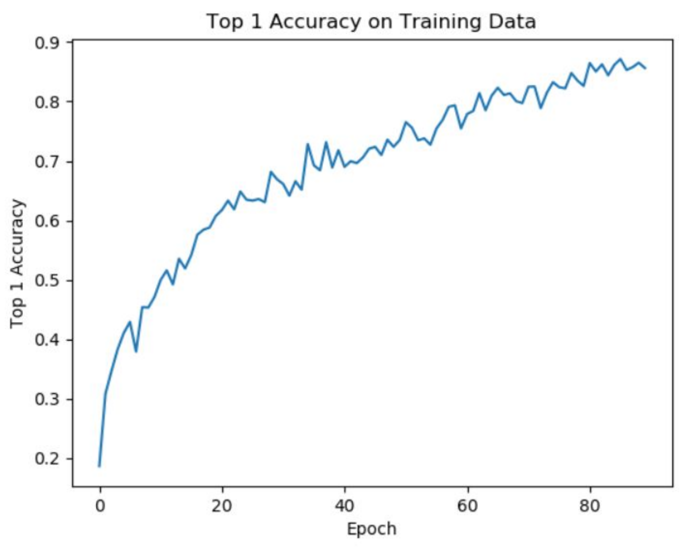
```
learning_rate = 0.001
usePersonSplit2 = True
Top 1 Accuracy on Training Set = 0.5936842105263158
Top 5 Accuracy on Training Set = 0.9249122807017544
Accuracy on Validation Set = 0.551578938961029
Validation accuracy has decreased (0.5557894706726074 -> 0.551578938961029)
Stopping training after Epoch 25 to prevent overfitting.
Top 1 Accuracy on Test Data:  0.566316
Top 5 Accuracy on Test Data:  0.901053
```

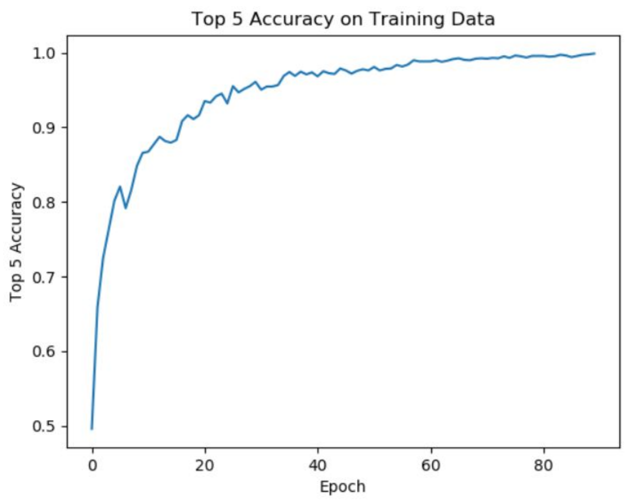




5	LC: 4-1	LR = 0.001

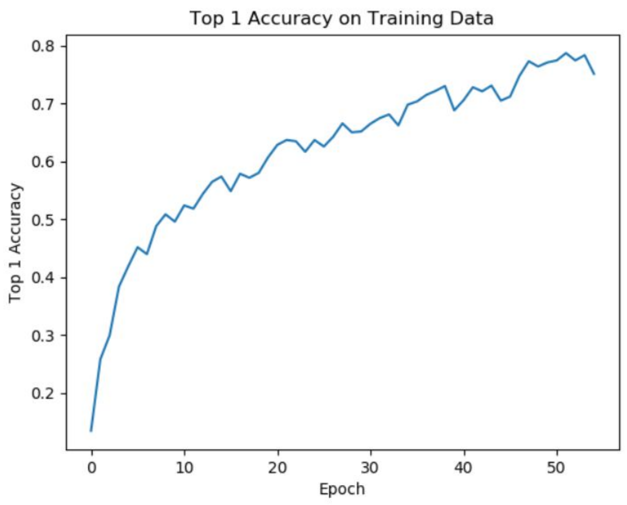
learning_rate = 0.001
usePersonSplit1 = True
Top 1 Accuracy on Training Set = 0.8563157894736843
Top 5 Accuracy on Training Set = 0.9989473684210526
Top 1 Accuracy on Test Data: 0.833684
Top 5 Accuracy on Test Data: 0.997895

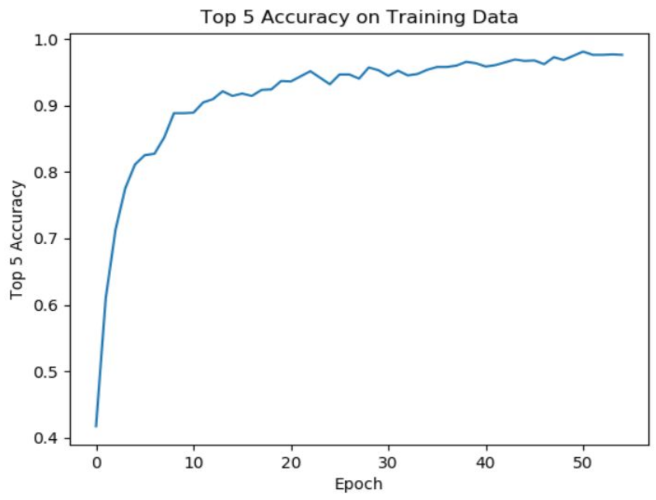




6	LC: Percentage	LR = 0.001

learning_rate = 0.001
usePercentage = True
Top 1 Accuracy on Training Set = 0.7515789270401001
Top 5 Accuracy on Training Set = 0.9761403799057007
Accuracy on Validation Set = 0.7010526061058044
Validation accuracy has decreased (0.7136842012405396 -> 0.7010526061058044)
Stopping training after Epoch 55 to prevent overfitting.
Top 1 Accuracy on Test Data: 0.698947
Top 5 Accuracy on Test Data: 0.966316


Top 1 Accuracy on Training Data
Top 1 Accuracy
Epoch


Top 5 Accuracy on Training Data
Top 5 Accuracy
Epoch