American Sign Language Recognition

Tianchang Li
tli289@wisc.edu

Hao Wang
hwang779@wisc.edu

Jing Wen
jwen29@wisc.edu

Abstract

Nowadays, American Sign Language (ASL) is used predominantly in the United States and in many parts of Canada by deaf communities. Even though ASL facilitates the communication among deaf people, the communication between deaf community and normal people is still limited by rare usage of ASL among ordinary people. By figuring out algorithms that performs computer version ASL recognition, we hope that ASL of deaf community can be directly translated to English letters or classes for SPACE, DELETE and NOTHING. With ResNet18, Inception v-3, and Autoencoder, three different models, our group develops high accuracy algorithm to translate American Sign Language from pictures to letters.

1. Introduction

Sign language is a type of language important to certain types of people. Because of limited usage of sign language,the communication using sign language should be paid more attention. American Sign Language uses hand gestures to represent 26 letters. People's names, places, titles, brands, new foods, and uncommon animals or plants all fall broadly under the finger spelling alphabet, and this list is by no means exhaustive. Due to this reason, the recognition process for each individual letter plays quite a crucial role in its interpretation [2]. Research on ASL was widely conducted with deep learning. Human body pose estimation and hand detection are two important tasks for systems that perform computer vision-based sign language recognition(SLR)[3]. Our project will focus on recognizing images of the 26 letters in American Sign Language and 3 classes for SPACE, DELETE and NOTHING. This will help people who cannot recognize American Sign Language to smoothly communicate with people with disability through computer version ASL recognition. The translation of ASL will also help beginners of this language to learn in the same way as subtitles of foreign movies do. We hope this algorithm to break the communication barrier between people with speaking disability and the rest. Moreover, our group hope to apply knowledge from class to real

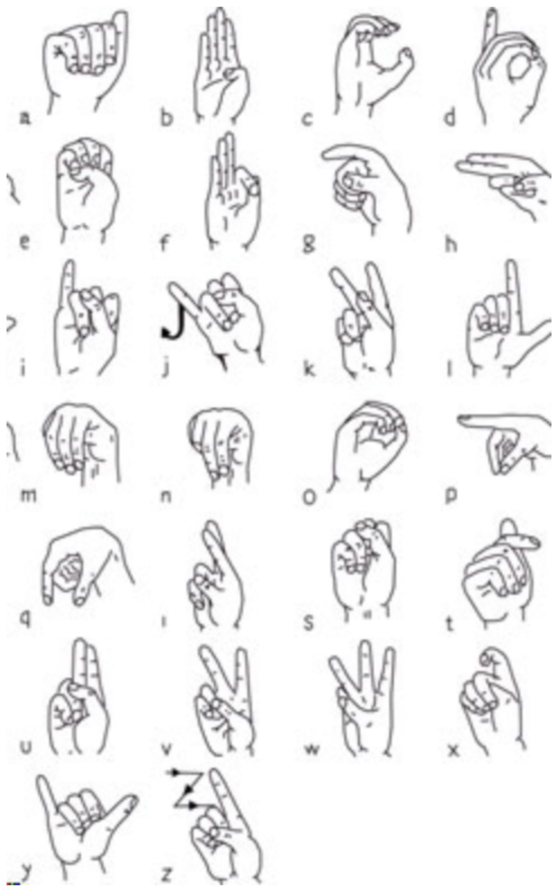


Figure 1. American Sign Language Alphabet

life problems. Examining ASL with deep learning will also prove the capacity of convolutional neural network recognizing complicated gesture images with low error rates. except knowledge learnt from class, our group hope to learn more by optimizing our model without class Knowledge. Our group will convert sign pictures into numerical data set and obtain bias and weights by constructing a multiple-layer Convolutional Neural Network. After achieving sufficient accuracy on this step, we will then try to build a text analysis model which takes in a sequence of classifications from the first step and identify multiple-letter words and even sentences. Our group finally developed three models to solve the problem.

2. Related Work

Oyebade K. Oyedotun and Adnan Khashman proposed applying deep learning to the problem of hand gesture recognition for the whole 24 hand gestures obtained from Thomas Moeslund's gesture recognition database [4]. Their experiment showed that more biologically inspired and deep neural networks such as convolutional neural networks and stacked denoising autoencoders are capable of learning the complex hand gesture classification task with lower error rates [3].

3. Proposed Method

In this project, we implemented three state-of-art deep learning models, ResNet18, Inception v-3, and Autoencoder with transpose convolution.

3.1. ResNet18

ResNet18 includes 4 layers for convolution and a fully connected layer for classification. Figure [2] illustrates the architecture of ResNet34, a model with similar within and between-block structure. Within each convolutional layer of ResNet18, there are two basic blocks each of which consists of two sets of a 2-d convolutional layer (3x3 kernel) and a BatchNorm layer, connected by a ReLU activation function in the middle. The activation of each first block is added to the activation of the second block, as shown in figure [3], which will be passed on to another ReLU function. The two activations from two basic blocks together make up the output for one layer. This technique prevents the vanishing gradient problem in neural networks with large architecture. If we denote a(i) as the activations for i th set, W(i), b(i) as the weights and bias for i th set, and z(i) as the weighted sum for i th block, the output/activation of the (l+2) th set can be represented as

$$a^{(l+2)} = \sigma * (z^{(l+2)} + a^{(l)})$$

$$= \sigma * (a^{(l+1)} * W^{(l+2)} + b^{(l+2)} + a^{(l)})$$

When weight W and bias b approaches zero, the final output of this layer approaches the activation of the first block.

By doing this, even if the second block does not learn to perform well on the feature extraction task, it would not affect training for the later on layers. To match the output size for two connected blocks, the activation from the previous block is downsampled by another convolutional and BatchNorm layer (as shown in figure [3]). In this project we hoped to implement a deep net with a number of convolutional layers to extract the outline of hand gestures. ResNet fits this purpose well.

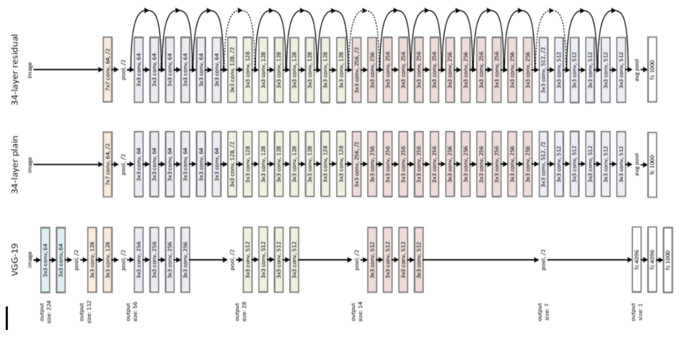


Figure 2. ResNet34

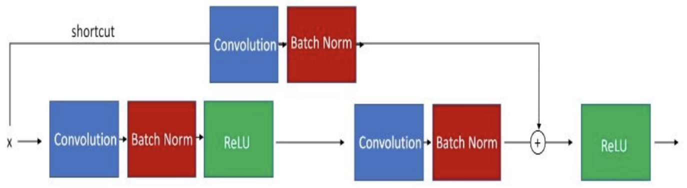


Figure 3. basic block

3.2. Inception v-3

This part mainly focuses on methods and techniques we apply to recognize American Sign Language, including tensor flow, transfer learning and Inception-v3 model.

Tensor Flow Tensor Flow is an artificial neural network with more than three layers which consists of one input, one output and multiple hidden layers. As a deep learning framework firstly developed by Google, it can control every neuron in the network and also possesses several libraries suitable for image processing. The weights of the neural network can be modified to achieve better performance in image recognition.

Transfer Learning Transfer Learning, as an important Machine Learning technique, makes it possible to train and develop a model for one task and then reuse previous results on another similar task. It refers to the situation whereby what has been learnt in one setting is exploited to improve optimization in another setting. Transfer Learning is usually used when a new dataset smaller than the original dataset is applied to train the pre-trained model. Here Inception-v3 is first trained on a base dataset and learn features (or transfer them), then trained on a new dataset. Transfer Learning enables us to start with the learned features on the base dataset and then adjust these features or even the structure of the model to suit a new dataset rather than start a new learning process on the data with random weight initialization. TensorFlow facilitates Transfer Learning of the CNN pre-trained model.

Inception v-3 Inception-v3 is one of the pre-trained models on the TensorFlow. It is a rethinking for the initial structure of computer vision after Inception-v1, Inception-v2 in 2015. The Inception-v3 model is trained on the training datasets, containing the information that can identify 29 classes. In training dataset, the error rate is about 15 percents. TensorFlow also allows us to retrain Inception's final Layer for new categories using transfer learning. Figure.1 describes a Inception-v3 model which processes convolution, pooling, softmax and fully connected operations.

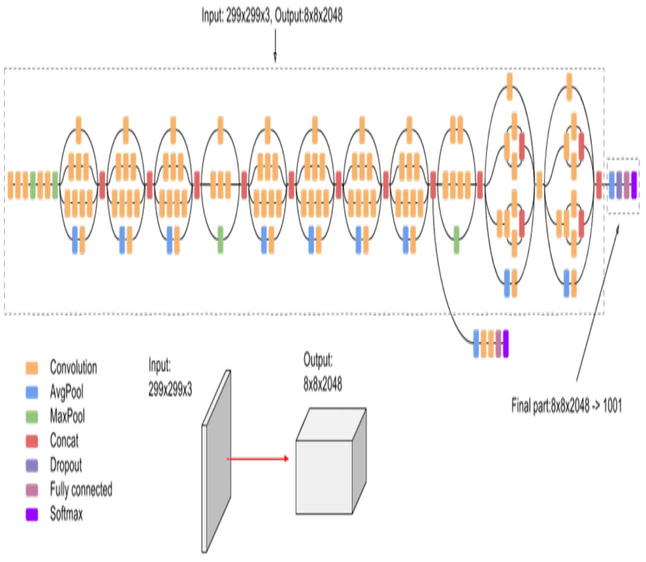


Figure 4. inceptionv3

3.3. Autoencoder

Due to the large size of images used in this project, we explored to reduce dimensionality by extracting important representation of the images with the encoder part of Autoencoder. Autoencoder is a neural network used to compress and reconstruct images (figure [5]). In the encoder phase, a few layers of convolutions compress the flatten images into small vectors, named latent space. In the decoder phase, the transpose convolutions that mirror the ones in the previous phase are implemented to reconstruct the images. By back propagating the mean squared error (MSE) between the reconstructed and original images, the model learns to refine the representations at the bottleneck. Feeding the representations in traditional machine learning models such as Random Forest, we are able to compare this technique with common deep nets.

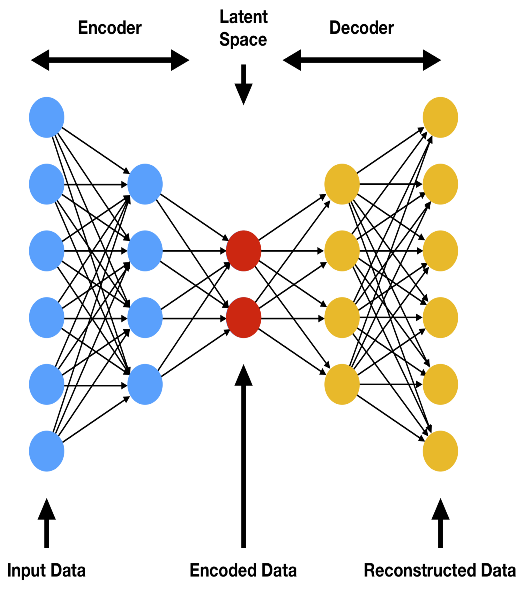


Figure 5. Autoencoder

4. Experiment

Our goal is to accurately predict the alphabet in each hand gesture picture. The model outputs are set to 29 classes. We used cross-entropy to calculate the loss of each batch. Mini-batch gradient descent was used in all models. To maximize the computation efficiency, we set the mini batch size to 256. We applied adaptive learning rate via the adaptive moment estimation (ADAM) which is a widely used method combining the momentum method and RMSProp. Throughout the experiments, we used the 3-way holdout validation method where the dataset is divided into train, validation, and test set in the ratio of 7:2:1.

The code for data cleaning and model implementation can be found on our GitHub site: https://github.com/LTCrazy/ASL_Recognition_DL

4.1. ResNet18

Firstly, we applied the pre-trained ResNet18 on the raw 200x200x3 images with an initial learning rate of 0.001, allowing the last fully connected layer to learn. The training and validation accuracy reached 96.89 percents and 96.43 percents by the eighth epoch. However, it took about 15 hours to finish the first eight epochs which makes this model hardly practice in real-life classification when we have tens of thousands images. But this trial confirmed the potential classification capacity of ResNet18 on our dataset. To reduce the dimension of input, we resize the images down to 64x64x3. We also augmented the original training images by random cropping with a ratio between 0.95 and 1.05. Figure [6] and [7] exemplify some outcome images. This enabled the model to learn from noisier images, which improved the classification ability on real-life application. Based on this model setting, we also manipulated initial learning rate and applied the exponential learning rate decay technique, aiming at breaking the local minima if any exist and better settling on the global minimum if reached. We explored initial learning rate of 0.0015 and 0.0012 with exponential decay (gamma = 0.85) In addition, given ResNet18 itself does not include dropout layers, we added 5

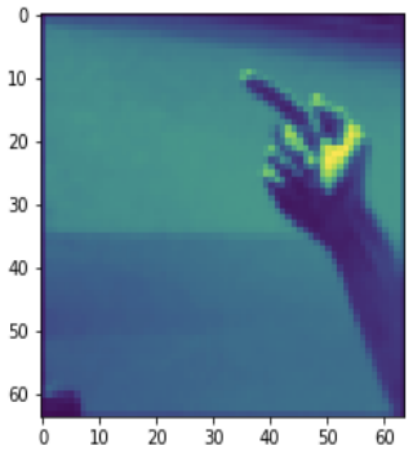


Figure 6. figure 1

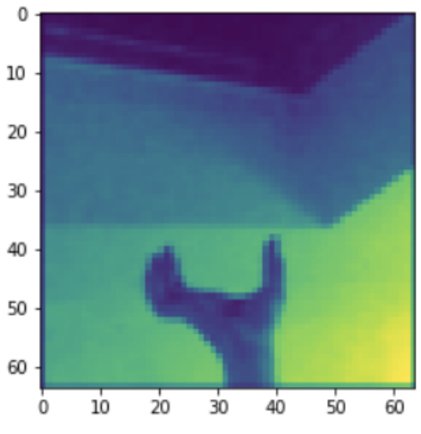


Figure 7. figure 2

4.2. Inception

This section mainly talks about our experimental procedure using Inception-v3 on Tensor Flow framework. In detail, this Inception-v3 model is divided into following three stages: image prepossessing, training, verification and testing.

Image Preprocessing Image pre-processing is a common procedure in image recognizing problem where the quality of image data can be improved by reducing incorrect or even useless information like undesired distortions and by enhancing useful information like some image features important for further processing at the lowest level of abstraction. As a preprocessing procedure, it certainly cannot increase image information content, instead, it just selectively takes the essence and discard the dross of these images on the basis of redundancy in images. Now that neighbouring pixels corresponding to one sign in collected images share a very similar brightness value, a distorted pixel can still be restored by taking the average value of neighbouring pixels if it is about to be discarded.

Training In the training process, we use an image dataset containing and every image is used multiple times throughout the training process. It takes a long time (about three to four hours) to compute the layers especially these convolutional layers just before the final output layer that sorts each image into a specific class. As long as the lower layers of the neural network have not been changed their outputs can be stored and reused.

Verification and Testing By testing, the system is evaluated in different conditions and its behavior as well performance is observed. As mentioned above, rather than using single image, multiple images are used for multiple times as input to the inception to detect its defects. On the other hand, by verification, a convincing argument that the model will not perform badly when applied under a broad range of circumstances, here we verify this by recognizing images taken by ourselves.

4.3. Autoencoder

In the encoder phase, we implemented a 2-d convolution layer taking in 64x64x3 with 3x3 kernel, stride of 1 and same padding. This outputs 64x64x4 nodes which were shrunk by half to 32x32x4 by 2-d max pooling with 2x2 kernel and stride of 2. Then we implemented another set of a convolution layer (output 32x32x8) and a max pooling layer (output 16x16x8), creating the bottleneck representation vectors. Then these steps were reversed in the decoder phase by a set of transpose convolutions that output 64x64x3 images. After sufficiently training this autoencoder model for 20 epochs, we applied the random forest classifier with 500 estimators on the bottleneck representation vectors and compared the accuracy with feeding the original 64x64x3 images.

4.4. Dataset

Our group decided to use image data set for alphabets in the American Sign Language from Kaggle [1]. The data set includes 87,000 colored images which are 200x200 pixels. Images were shot on hand gestures evenly distributed into 29 classes, of which 26 are for the letters A-Z and 3 classes for SPACE, DELETE and NOTHING. These three classes are very helpful in real time applications, and classification. The background of images are mostly grey.

4.5. Software

With three different models: ResNet18, Inception, and Autoencoder, we use software below in our experiment: Pytorch, NumPy, Pandas, Matplotlib, PIL packages run in Jupyter Lab

4.6. Hardware

We will use the following computer hardware and computational tool:

MacBook Pro (2.7 GHz Intel Core i5, Early 2015)

MacBook Pro (2.3 GHz Intel Core i5, 2017)

Window 10, AMD FX3600 processor with 6 cores hyperthreaded, 8GB DDR3 RAM

5. Results and Discussion

5.1. ResNet

After training ResNet18 on the random cropped pictures for 20 epochs, the accuracy reaches 86.63 percent and 85.38 percent for train and validation set in 8.5 hours which improves significantly from feeding in 200x200 original images. The accuracy and loss plots show neat smooth patterns with very slim gaps between train and validation set, which shows little overfitting (figure [8] and [9]). A 85.19 percent test accuracy also suggests the validity of this model. However, the train and validation accuracies start to show a hint of divergence, from which we decided to hold off on further training. We believe that the train and validation accuracies could still improve a little bit, but the test accuracy would more likely to remain the same.

Although, these accuracies are still far away from the 96.89 percent and 96.43 percent obtained by feeding in original images. We suspected if the current model was stuck on a local minimum that hindered a better performance due to the constant learning rate. To verify that, we set the initial learning rate to 0.0015 instead of 0.001 and applied exponential decay with gamma = 0.85 and dropout with 20 percent probability. The accuracy and loss plots show obvious fluctuation within 20 epochs(figure [10] and [11]). We suspected that the high dropout probability made the model unstable, meaning that what the model learned in each batch was more specific to the leftover network in that batch, rather than the whole network. Learning rate too big could also cause the fluctuation.

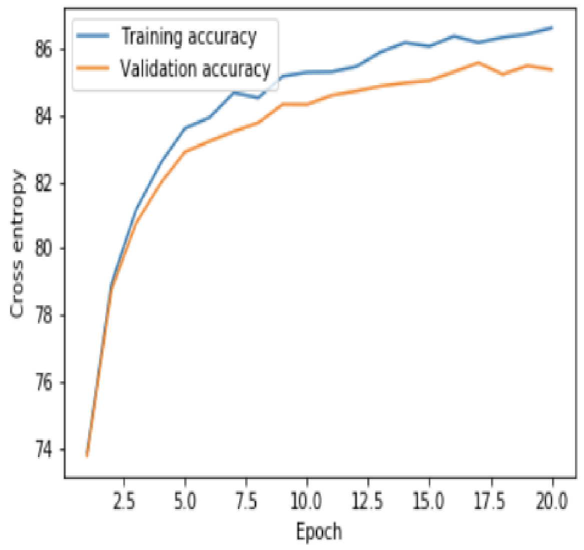


Figure 8.

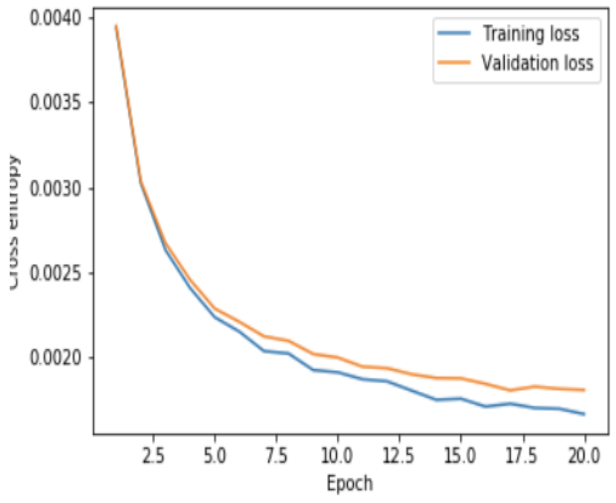


Figure 9.

Then we diminished the initial learning rate and dropout probability to 0.0012 and 5 percent. Note that the learning rate decay was also applied every 4 epochs since the fourth to the last epoch this time. The result patterns are much more stable now with less overfitting (figure [12] and [13]). However, neither the train or the validation accuracy passed 81 percent and we doubted that they would significantly increase with further training. Until now we are able

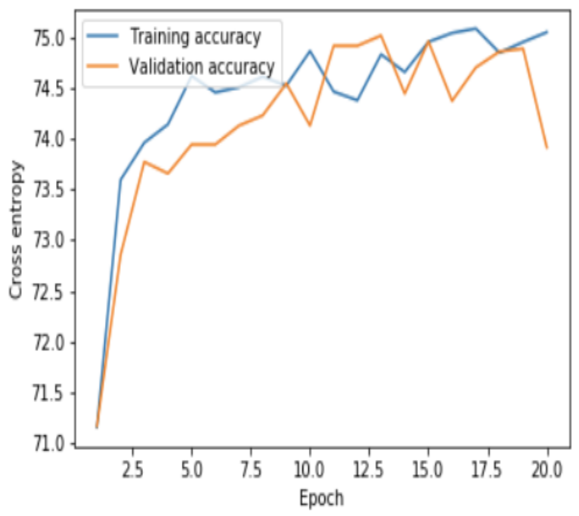


Figure 10.

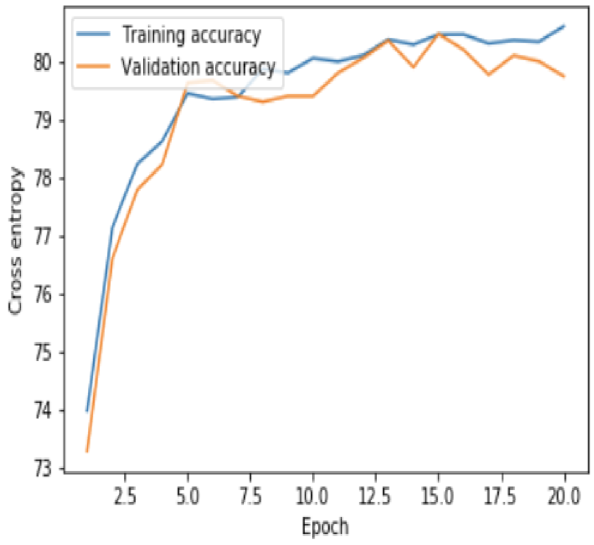


Figure 12.

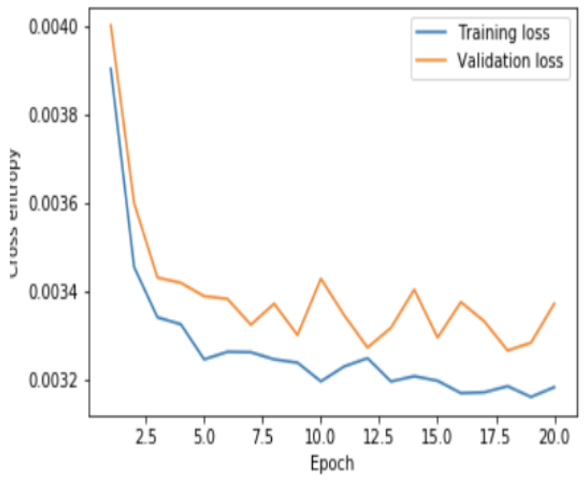


Figure 11.

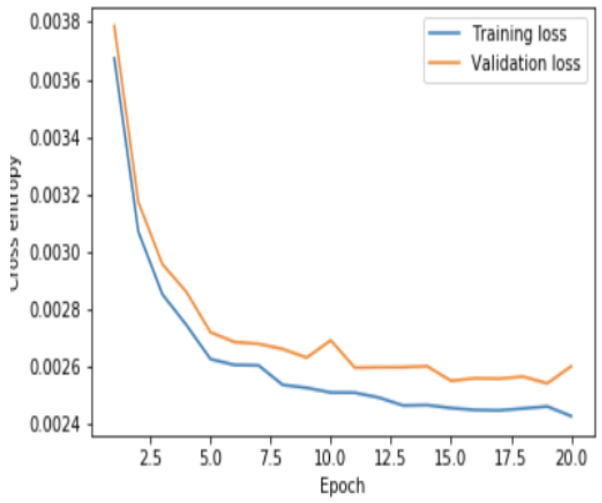
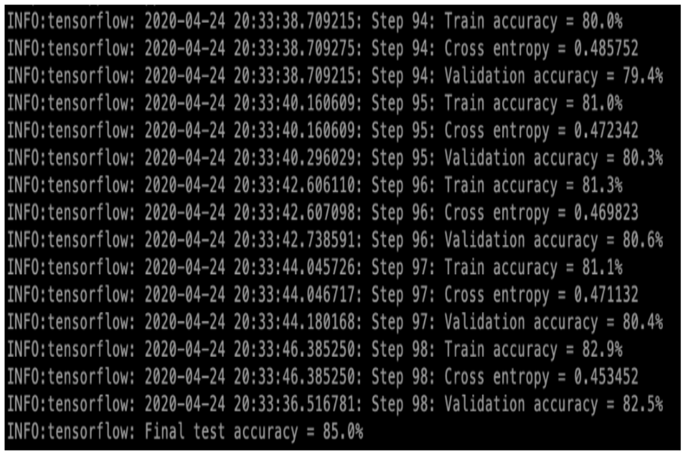


Figure 13.

to conclude that among these, the parameters in the pre-trained ResNet18 performs the best.

5.2. Inception v-3

Figure [14] show the variation of accuracy and cross entropy based on American Sign Language's dataset. The blue line represents the training set. As we can see from Figure [14], the validation accuracy and train accuracy are very close. The training accuracy shows the percentage of the images used in the current training batch that were labeled with the correct class. Validation accuracy: The validation accuracy is the precision (percentage of correctly labelled images) on a randomly selected group of images from a different set. Cross entropy is a loss function that gives a glimpse into how well the learning process is progressing (lower numbers are better here). Finally, we get a test accuracy about 85 percents, which is similar to ResNet model, but it still has some advantages such as saving parameters and thus saving training time.



```
INFO:tensorflow: 2020-04-24 20:33:38.709215: Step 94: Train accuracy = 80.0%
INFO:tensorflow: 2020-04-24 20:33:38.709275: Step 94: Cross entropy = 0.485752
INFO:tensorflow: 2020-04-24 20:33:38.709215: Step 94: Validation accuracy = 79.4%
INFO:tensorflow: 2020-04-24 20:33:40.160609: Step 95: Train accuracy = 81.0%
INFO:tensorflow: 2020-04-24 20:33:40.160609: Step 95: Cross entropy = 0.472342
INFO:tensorflow: 2020-04-24 20:33:40.296029: Step 95: Validation accuracy = 80.3%
INFO:tensorflow: 2020-04-24 20:33:42.606110: Step 96: Train accuracy = 81.3%
INFO:tensorflow: 2020-04-24 20:33:42.607098: Step 96: Cross entropy = 0.469823
INFO:tensorflow: 2020-04-24 20:33:42.738591: Step 96: Validation accuracy = 80.6%
INFO:tensorflow: 2020-04-24 20:33:44.045726: Step 97: Train accuracy = 81.1%
INFO:tensorflow: 2020-04-24 20:33:44.046717: Step 97: Cross entropy = 0.471132
INFO:tensorflow: 2020-04-24 20:33:44.180168: Step 97: Validation accuracy = 80.4%
INFO:tensorflow: 2020-04-24 20:33:46.385250: Step 98: Train accuracy = 82.9%
INFO:tensorflow: 2020-04-24 20:33:46.385250: Step 98: Cross entropy = 0.453452
INFO:tensorflow: 2020-04-24 20:33:36.516781: Step 98: Validation accuracy = 82.5%
INFO:tensorflow: Final test accuracy = 85.0%
```

Figure 14.

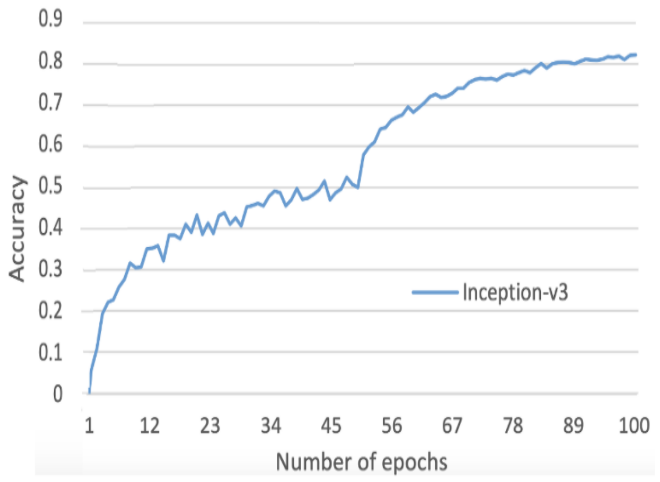


Figure 15.

5.3. Autoencoder

The model took about 2 hours to train and was able to reconstruct images fairly close to the original ones for most of the classes (figure [14] and figure [15]).Then we constructed a random forest classifier with original resized 64x64x3 images for reference. Surprisingly, this simple classifier was already able to achieve an accuracy of 99.26 percent within 12 minutes. We attribute this to the simple structure of hand gestures and clean, consistent backgrounds. However, with the encoded vectors obtained from Autoencoder, we were able to improve the accuracy by another 0.4 percent in a slightly shorter amount of time. The number of parameters was also reduced by half (64x64x3 -¿ 16x16x8) which saved computational cost. The better accuracy and higher efficiency approved the capacity of this Autoencoder model.

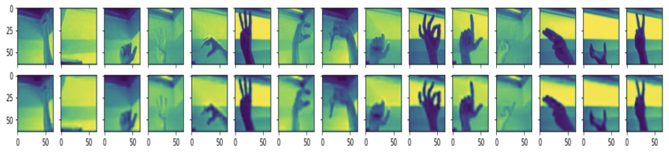


Figure 16.

6. Conclusions

We build three image classification networks implementing ResNet, Inception-v3, and Autoencoder. The classification accuracy of the model achieved approximately 85 percent on test dataset with ResNet which performed the best in our experiments. We found that shrinking the size of the training images resulted in considerably less accuracy, which sets up the goal for further experiments on improving accuracy on small images or improving training speed on large images. The consistent background and shooting angles in our images make this classification task less challenging. In the future we will collect more training images with various background noises to enhance the reliability of our models. The surprisingly good performance of Random Forest classifier also suggests that, besides neural networks, traditional machine learning approaches are also worth of attention on this type of images.

7. Acknowledgements

Part of the codes and graphs referred to Professor Raschka's lecture codes and notes on CNN, transfer learning, and Autoencoder.

8. Contributions

The experiment and writing tasks were assigned to group members evenly. All of the group members will be involved with experiment and writing tasks. For Proposal writing, Jing Wen is focusing on writing reports, finding resources for models and importing data set. Meanwhile, Hao Wang and Tianchang Li take charge to develop models and calculate accuracy. Tianchang Li implemented and tuned two models: ResNet and Autoencoder while Hao Wang implemented and tuned Inception v-3 model to help translate American Sign language.

References

[1] Akash. Asl alphabet, Apr 2018. https://www.kaggle.com/grassknoted/asl-alphabet.

[2] V. Bheda and D. Radpour. Using deep convolutional networks for gesture recognition in american sign language, 2017. [3] S.

Gattupalli, A. Ghaderi, and V. Athitsos. Evaluation of deep learning based pose estimation for sign language recognition. In Proceedings of the 9th ACM International Conference on PErvasive Technologies Related to Assistive Environments, pages 1–7, 2016.
[4] O. K. Oyedotun and A. Khashman. Deep learning in vision-based static hand gesture recognition. Neural Computing and Applications, 28(12):3941–3951, 2017.