Final Report of Internship Program 2021

On

"AMERICAN SIGN LANGUAGE RECOGNITION"

MEDTOUREASY

25th March 2021

Performed by:

Pravin Konasirasgi

ACKNOWLDEGMENTS

The internship opportunity that I had with MedTourEasy was a great change for learning and understanding the intricacies of the subject of Data Visualizations in Machine Learning; and also, for personal as well as professional development. I am very obliged for having a chance to interact with so many professionals who guided me throughout the internship project and made it a great learning curve for me.

Firstly, I express my sincere gratitude and special thanks to the Training & Developement Team of MedTourEasy who gave me an opportunity to carry out my internship at their esteemed organization. Also, I express my thanks to the team for making me understand the details of the Machine Learning profile and training me in the same so that I can carry out the project properly and with maximum client satisfaction and also for spearing his valuable time in spite of his busy schedule.

I would also like to thank the team of MedTourEasy and my colleagues who made the working environment productive and very conducive.

TABLE OF CONTENTS

ABSTRACT

The communication gap between the deaf and hearing population is clearly noticed. To make possible the communication between the Deaf and the hearing population and to overpass the gap in access to next generation Human Computer Interfaces, automated sign language analysis is highly crucial. Conversely, an enhanced solution is to build up a conversion system that translates a sign language gestures to text or speech. Exploration and experimentation of an efficient methodology based on facet features analysis. For a recognition system that can recognize gestures from video which can be used as a translation, A methodology has been proposed that extracts candidate hand gestures from sequence of video frames and collect hand features. The system has three separate parts namely: Hand Detection, Shape Matching and Hu moments comparison. The Hand Detection section detects hand through skin detection and by finding contours. It also includes processing of video frames. The procedure of shape matching is attained by comparing the histograms. The values of Hu moments of candidate hand region is identified using contour region analysis and compared to run matches and identify the particular sign language alphabet. Experimental analysis supports the efficiency of the proposed methodology on benchmark data.

American Sign Language (ASL) is a visual gestural language which is used by many people who are deaf or hard-of-hearing. In this paper, we design a visual recognition system based on action recognition techniques to recognize individual ASL signs. Specifically, we focus on recognition of words in videos of continuous ASL signing. The proposed framework combines multiple signal modalities because ASL includes gestures of both hands, body movements, and facial expressions. We have collected a corpus of RBG + depth videos of multi-sentence ASL performances, from both fluent signers and ASL students; this corpus has served as a source for training and testing sets for multiple evaluation experiments reported in this paper. Experimental results demonstrate that the proposed framework can automatically recognize ASL.

1.1 About the Company

MedTourEasy, a global healthcare company, provides you the informational resources needed to evaluate your global options. It helps you find the right healthcare solution based on specific health needs, affordable care while meeting the quality standards that you expect to have in healthcare. MedTourEasy improves access to healthcare for people everywhere. It is an easy to use platform and service that helps patients to get medical second opinions and to schedule affordable, high-quality medical treatment abroad.

1.2 About the Project

The National Institute on Deafness and Other Communications Disorders (NIDCD) indicates that the 200-year-old American Sign Language is a complete, complex language (of which letter gestures are only part) but is the primary language for many deaf North Americans.

American Sign Language (ASL) is the primary language used by many deaf individuals in North America, and it is also used by hard-of-hearing and hearing individuals. The language is as rich as spoken languages and employs signs made with the hand, along with facial gestures and bodily postures.

Therefore, to build a system that can recognise sign language will help the deaf and hard-of-hearing better communicate using modern-day technologies. In this article, we will go through different architectures of CNN and see how it performs on classifying the Sign Language.

Hence, this project aims at training a convolutional neural network to classify images of ASL letters. After loading, examining, and preprocessing the data, we will train the network and test its performance.

1.3 Objectives and Deliverables

This project focuses on to build a neural network able to classify which letter of the American Sign Language (ASL) alphabet is being signed, given an image of a signing hand. This project is a first step towards building a possible sign language translator, which can take communications in sign language and translate them into written and oral language. Such a translator would greatly lower the barrier for many deaf and mute individuals to be able to better communicate with others in day to day interactions.

This goal is further motivated by the isolation that is felt within the deaf community. Loneliness and depression exists in higher rates among the deaf population, especially when they are immersed in a hearing world. Large barriers that profoundly affect life quality stem from the communication disconnect between the deaf and the hearing. Some examples are information deprivation, limitation of social connections, and difficulty integrating in society.

Most research implementations for this task have used depth maps generated by depth camera and high resolution images. The objective of this project was to see if neural networks are able to classify signed ASL letters using simple images of hands taken with a personal device such as a laptop webcam. This is in alignment with the motivation as this would make a future implementation of a real time ASL-to-oral/written language translator practical in an everyday situation.

I. METHODOLOGY

This project presents the implementation techniques which are utilized to build a Sign Language Identification System. It comprises of a thorough explanation of the system and provides with the general idea of the required image/video handling algorithms and functions needed to run the application.

C#.NET Framework is used because it is capable of performing in congenial combinations and comprehensible edge to course the system. The OpenCV and EmguCV libraries that have been developed for image dealing are utilized to jot down the required functions. The fundamental design of these libraries are at real time computer vision, and at managing diverse image/video processes such as gesture recognition, motion tracking, object identification, etc.

A divide and conquer approach has been pursued to build the system where the complete system is partitioned into several components. To concurrently focus on small tasks and complete them in parallel, this approach is very useful. The subsequent step would be to accumulate them and investigate their functionality on the whole. The following block diagram shows the nonrepresentational version of processes involved in sign language recognition system.

2.1 Flow of the Project

The project followed the following steps to accomplish the desired objectives and deliverables. Each step has been explained in detail in the following section

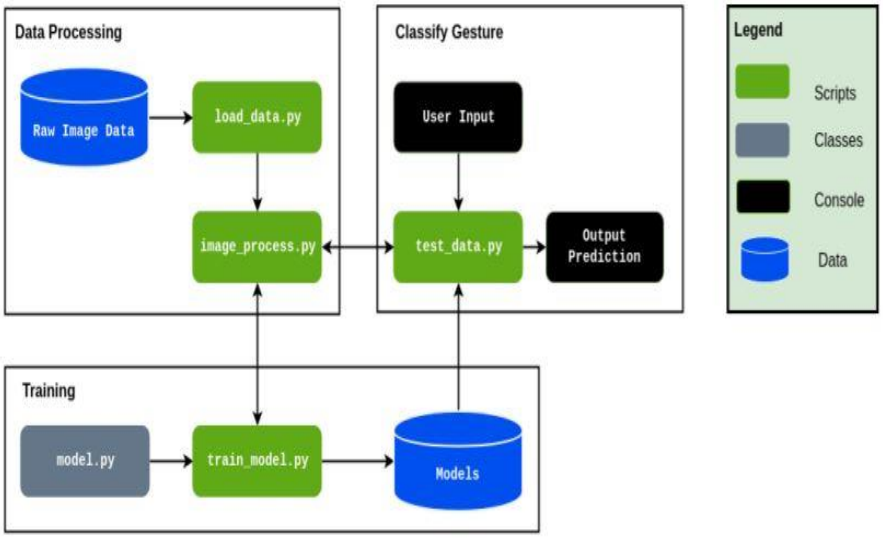
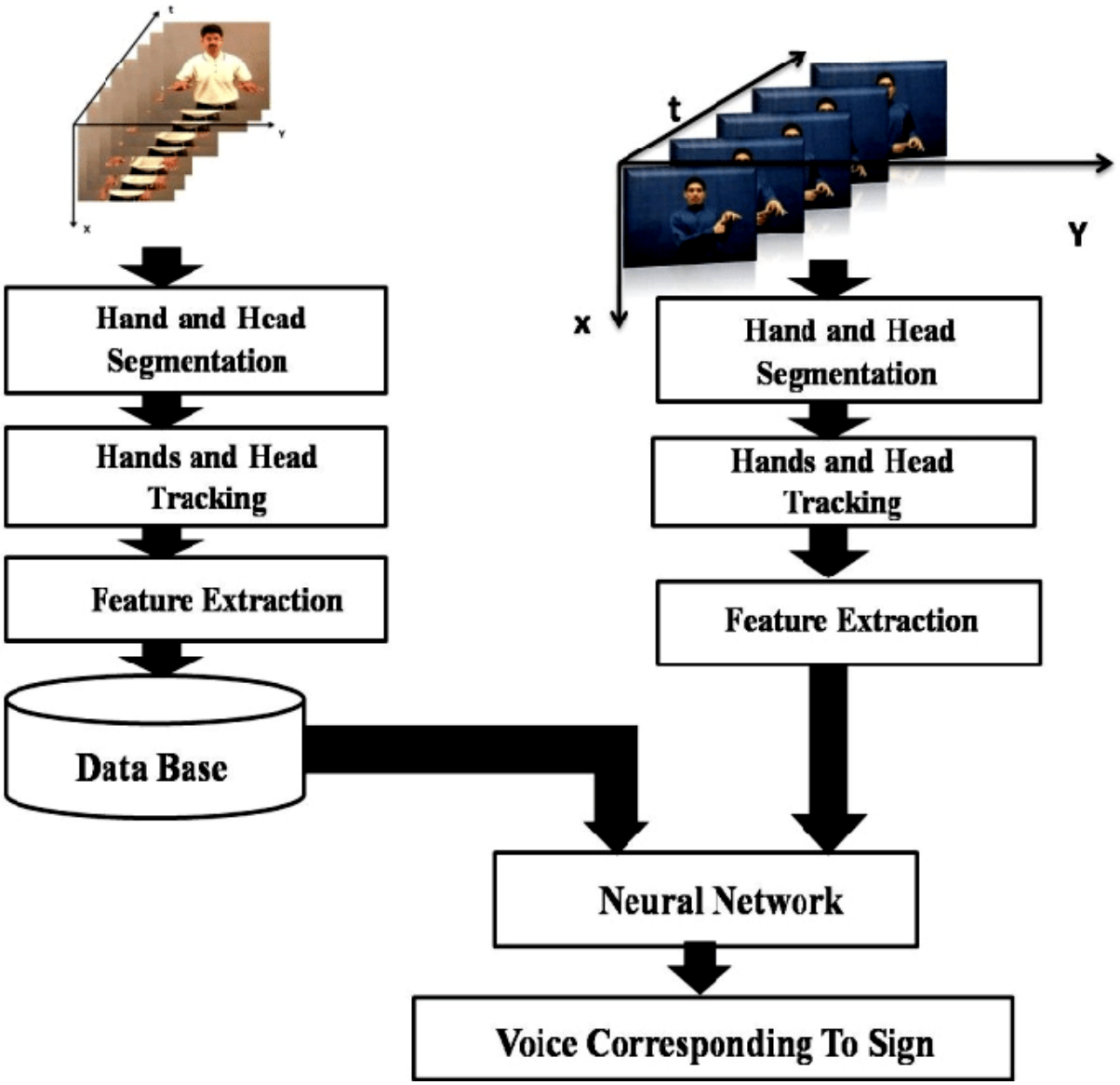


Figure 1: Block Diagram of Software

2.2 Use Case Diagram



Above figure shows the use case of the project. UML illustrates software both behaviorally (dynamic view) and structurally (static view). It has a graphical notation to create visual models of the systems and permits extension with a profile (UML). The UML include elements Such as: Actors, Activities, Use Cases, Components and so on. The Use Case diagram is a behavioral UML diagram. It expresses the functionality offered by a system in terms of actors, their objectives characterize as use cases, and any reliance amongst these use cases..

2.3 Language and Platform Used

2.3.1 Language: Python

Python is one of the most dynamic and versatile programming languages available in the industry today. Since its inception in the 1990s, Python has become hugely popular and even today there are thousands who are learning this Object-Oriented Programming language. If you are new to the world of programming, you have already heard the buzz it has created in recent times because of the features of Python and must be wondering what makes this programming language special. The important features of Python are:

- Easy to code: Python is a high-level programming language.
- Free and Open Source.
- **Object**-Oriented Language.
- GUI Programming Support.
- High-Level Language.
- Extensible feature.
- Python is **Portable** language.
- Python is Integrated language.

2.3.2 IDE: Juptyer Notebook

JupyterLab is a web-based interactive development environment for Jupyter notebooks, code, and data. JupyterLab is flexible: configure and arrange the user interface to support a wide range of workflows in data science, scientific computing, and machine learning. JupyterLab is extensible and modular: write plugins that add new components and integrate with existing ones. It was spun off from IPython in 2014 by Fernando Pérez.. Major features are:

- All in one place.
- Easy to share.
- Easy to convert.
- Language independent.
- Easy to create kernel wrappers.
- Easy to customize.
- Extensions with custom magic commands.
- Stress-free Reproducible experiments.
- Effective teaching-cum-learning tool.
- Interactive code and data exploration:

II. IMPLEMENTATION

As shown in Figure 1, the project will be structured into 3 distinct functional blocks, Data Processing, Training, Classify Gesture. The block diagram is simplified in detail to abstract some of the minutiae:

3.1 Data Collection

The primary source of data for this project was the compiled dataset of American Sign Language (ASL) called the ASL Alphabet from Kaggle user Akash [3]. The dataset is comprised of 87,000 images which are 200x200 pixels. There are 29 total classes, each with 3000 images, 26 for the letters A-Z and 3 for space, delete and nothing. This data is solely of the user Akash gesturing in ASL, with the images taken from his laptop's webcam. These photos were then cropped, rescaled, and labelled for use.

(a) ASL letter A (b) ASL letter E (c) ASL letter H (d) ASL letter Y

Figure 2: Examples of images from the Kaggle dataset used for training. Note difficulty of distinguishing fingers in the letter E.

A self-generated test set was created in order to to investigate the neural network's ability to generalize. Five different test sets of images were taken with a webcam under different lighting conditions, backgrounds, and use of dominant/non-dominant hand. These images were then cropped and preprocessed.

3.2 Data Pre-processing

The data preprocessing was done using the PILLOW library, an image processing library, and sklearn.decomposition library, which is useful for its matrix optimization and decomposition functionality.

(a) Sample from non-uniform background test set

(b) Sample from plain white background test set

(c) Sample from a darker test set

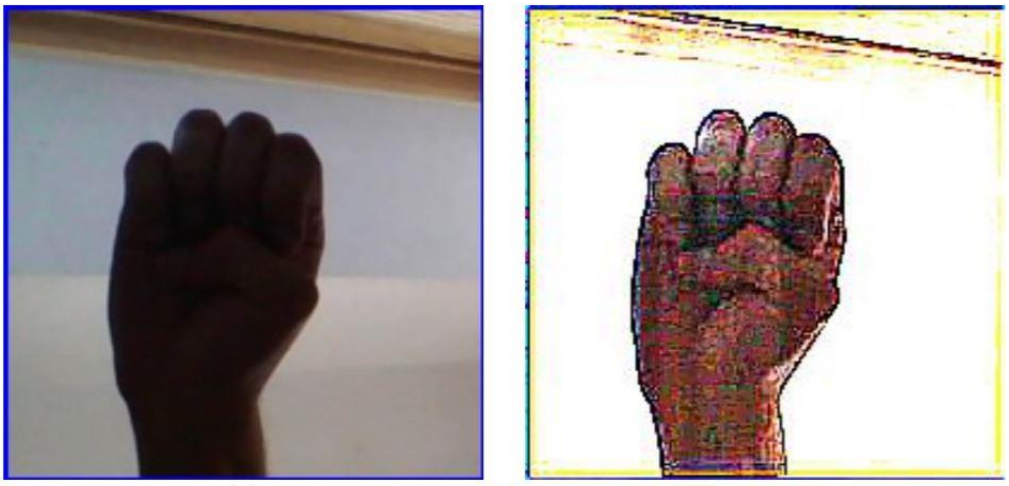
(d) Sample from plain white background test set

Figure 3: Examples of the signed letter A T from two test sets with differing lighting and background

Image Enhancement: A combination of brightness, contrast, sharpness, and color enhancement was used on the images. For example, the contrast and brightness were changed such that fingers could be distinguished when the image was very dark.

Edge Enhancement: Edge enhancement is an image filtering techniques that makes edges more defined. This is achieved by the increase of contrast in a local region of the image that is detected as an edge. This has the effect of making the border of the hand and fingers, versus the background, much more clear and distinct. This can potentially help the neural network identify the hand and its boundaries.

Image Whitening: ZCA, or image whitening, is a technique that uses the singular value decomposition of a matrix. This algorithm decorrelates the data, and removes the redundant, or obvious, information out of the data. This allows for the neural network to look for more complex and sophisticated relationships, and to uncover the underlying structure of the patterns it is being trained on. The covariance matrix of the image is set to identity, and the mean to zero.



(a)

(b) Preprocessed Image

Figure 4: Examples of image preprocessing.

3.3 Data Processing

The load data.py script contains functions to load the Raw Image Data and save the image data as numpy arrays into file storage. The process data.py script will load the image data from data.npy and preprocess the image by resizing/rescaling the image, and applying filters and ZCA whitening to enhance features. During training the processed image data was split into training, validation, and testing data and written to storage. Training also involves a load dataset.py script that loads the relevant data split into a Dataset class. For use of the trained model in classifying gestures, an individual image is loaded and processed from the filesystem.

3.4 Training

The training loop for the model is contained in train model.py. The model is trained with hyperparameters obtained from a config file that lists the learning rate, batch size, image filtering, and number of epochs. The configuration used to train the model is saved along with the model architecture for future evaluation and tweaking for improved results. Within the training loop, the training and validation datasets are loaded as Dataloaders and the model is trained using Adam Optimizer with Cross Entropy Loss. The model is evaluated every epoch on the validation set and the model with best validation accuracy is saved to storage for further evaluation and use. Upon finishing training, the training and validation error and loss is saved to the disk, along with a plot of error and loss over training.

3.5 Classify Gesture

After a model has been trained, it can be used to classify a new ASL gesture that is available as a file on the filesystem. The user inputs the filepath of the gesture image and the test data.py script will pass the filepath to process data.py to load and preprocess the file the same way as the model has been trained

3.6 Contour Extraction

A contour is a sequence of points that represent, in one manner or another, a curve in an image [13]. This demonstration can be different depending on the given situation. The function FindContours() computes contours of the skin detected region from binary

frames of the video. It takes frames created by Canny (), which have boundary pixels in them and frames created by function AdaptiveThreshold(), in which the edges are hidden as boundaries between positive and negative regions. After this, the biggest contours are extracted and filtered from the found contours to get the boundary of the bright pixels. The contours points can be seen by using DrawContours().

3.7 Gesture Recgonition Technique

For the purpose of sign language identification, following steps have been implemented:
- Feature Collection
- Shape Matching
- Hu Moments Comparison
- Recognition Results

3.7.1 Feature Collection

The xy coordinates of each contour point for every frame of the video is saved in an array. The convex hull of the hand contour is computed to understand the shape of hand and then convexity defects is computed. The shapes of many complex objects are well characterized by such defects. ConvexityDefects() calculates the defects and in result gives a series of the defects. For evaluating two contours of different images, contour moments of hand area are computed. A moment is a crass feature of the shape figured out by summing or integrating over all of the pixels of the hand form. After this, the Hu moments of the contour moments are calculated. Linear groupings of the moments are called the Hu Invariant Moments. By merge the various hand contour moments, it is likely to generate invariant methods illustrating distinct aspects of the frame in a way that does not vary to rotation, scale and (for all but the one called h1) reflection. The cvGetHuMoments() function computes the Hu moments from the central moments. The following is the actual definition of the 7 Hu invariant moments .

3.7.2. Shape Matching

The Shape matching algorithm is based on 2D pair-wise geometrical histogram for the hand contours. The function cvCalcPGH() determines minimum/ maximum distances of the contour edges and angle connecting the contour points. The 2D pair-wise geometrical histogram of both template and target video frames are calculated. After this, both the histograms are normalized and cvCompareHist() is used to compare two dense histograms. The resultant value is stored in a variable, the lower the value of comparison the higher the matching.

3.7.3. HU Invariant Moments Comparison

The comparison of hand contour area of both the template and the target video frames is done. To compare two objects Hu moments is used and determine whether they are similar. The Similarity depends on the criterion which is provided. The type of CONTOURS MATCH returns a value, the lower the value the higher the possibility of getting the right answer.

3.7.4. Recognition Results

If the values of both the above variables are less than the provided threshold while matching/comparing target video frames and reference templates, the corresponding sign is displayed on the screen.

American Sign Language Hand Gestures (https://www.kaggle.com/datamunge/sign-language-mnist#amer_sign2.png)

III. SAMPLE SCREENSHOTS AND OBSERVATIONS

4.1 Define the model

Now it's time to define a convolutional neural network to classify the data.
This network accepts an image of an American Sign Language letter as input. The output layer returns the network's predicted probabilities that the image belongs in each category.

Input:

In [ ]:
```
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Flatten, Dense
from keras.models import Sequential

model = Sequential()
# First convolutional layer accepts image input
model.add(Conv2D(filters=5, kernel_size=5, padding='same', activation='relu',
                 input_shape=(50, 50, 3)))
# Add a max pooling layer
model.add(MaxPooling2D(pool_size=4))
# Add a convolutional layer
model.add(Conv2D(filters=15, kernel_size=5, padding='same', activation='relu'))
# Add another max pooling layer
model.add(MaxPooling2D(pool_size=4))
# Flatten and feed to output layer
model.add(Flatten())
model.add(Dense(3, activation='softmax'))

# Summarize the model
model.summary()
```

Output:

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 50, 50, 5)         380
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 12, 5)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 12, 12, 15)        1890
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 3, 3, 15)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 135)               0
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 408
=================================================================
Total params: 2,678
Trainable params: 2,678
Non-trainable params: 0
_________________________________________________________________
```

4.2 Training and Validation

Our models were trained using Adam optimizer and Cross Entropy Loss. Adam optimizer is known for converging quickly in comparison with Stochastic Gradient Descent (SGD), even while using momentum. However, initially Adam would not decrease our loss thus we abandoned it to use SGD. Debugging Adam optimizer after our final presentation taught us that lowering learning rate significantly can help Adam to converge during training. Thus allowing us to train more models towards the end of our project.

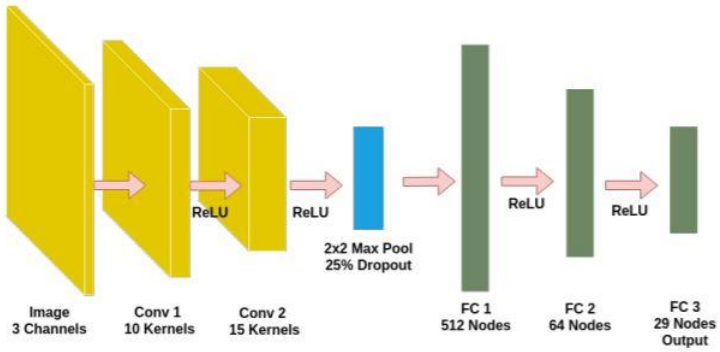


Figure 6: Model Architecture designed for this task for improved training time and establishing a baseline.

Of our two models, the one based on the paper was shown not to be viable, as it took much longer to train without showing any significant decrease in accuracy or loss. We believe this is likely due to the more difficult nature of our classification with the inclusion of background in the images and the lower resolution, causing training to be more difficult. Thus, we decided to focus on improving our smaller model which initially trained to 40% validation accuracy.

Although we had a very large dataset to work with; 3,000 samples for each of 29 classes, after processing the images into numpy arrays, we found our personal computers could load a maximum of 50-100 samples/class and our Google Cloud server could load 200 samples/class. The need to load small datasets actually led us to test the effect of increasing the data available to our models. On our preliminary model, which used strides of 2 on both layers (instead of the strides of 1 and then 2, on our final model) we found the following relation between samples/class and model accuracy:

Samples/Class	Avg. Validation Accuracy
25	27.3%
50	34.8%
100	46.1%
200	58.2%

Table 1: Effect of Increasing Dataset Size on Validation Accuracy

This is likely due to the small amount of samples to train on leading to bad generalization and learning of the sample space. Increasing the size of our dataset to 200 samples/class led to better model results, with peak validation accuracy of 60.3% in epoch 17. However, taking a look at our loss function, we see that the validation loss is increasing, indicating overfitting of the model. After we applied filtering, enhancement, and ZCA whitening to our dataset, the model performance increased drastically as shown in. The peak validation accuracy achieved is 77.25% in epoch 24. As shown by the plot of loss, the validation loss is still decreasing, albeit at a slower rate than the training loss, indicating that the model is not drastically overfitting. This shows that preprocessing our images by applying filters and ZCA whitening helps to enhance relevant features for the model to learn.

7. Train the model

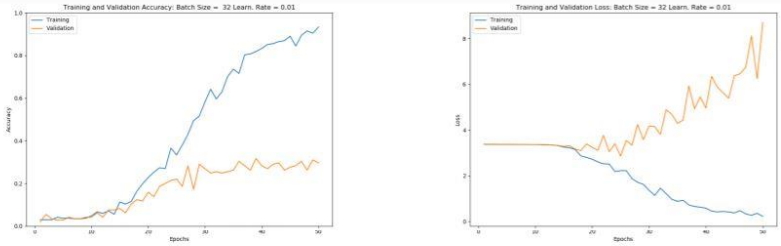
Once we have compiled the model, we're ready to fit it to the training data.

In [23]:
```
# Train the model
hist = model.fit(x_train, y_train_OH,
                 validation_split=0.2,
                 epochs=2,
                 batch_size=32)
```

```
Train on 1280 samples, validate on 320 samples
Epoch 1/2
1280/1280 [==============================] - 2s 2ms/step - loss: 0.9573 - acc: 0.6141 - val_loss: 0.7661 - val_acc: 0.8594
Epoch 2/2
1280/1280 [==============================] - 2s 2ms/step - loss: 0.6147 - acc: 0.8820 - val_loss: 0.4707 - val_acc: 0.9000
```

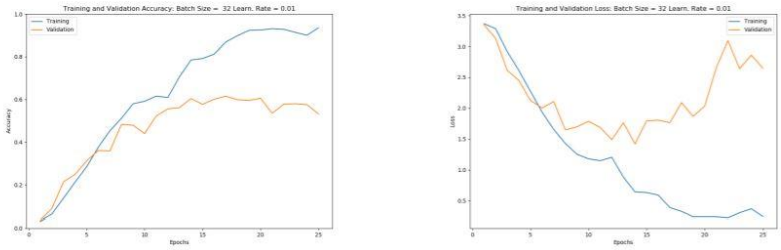
4.3 Testing the Model

To determine whether our preprocessing of images actually results in a more robust model, we verified on a test set comprised of images from the original dataset, and our own collected image data. The performance of the models on the test set is shown in Table 2. We see that the model trained on preprocessed images performs much better than the model trained on the original images, likely due to the former's lack of overfitting. Taking a look at the confusion matrix for the Filtered Model on the Kaggle test set in Figure 10, we can see that the model is fairly good at predicting the correct letter. Taking a look at some of confused letters, like V and W in Figure 11, we see that the two letters are similar in shape. This provides us with some intuition about why the model might confuse these two letters, as a W looks like a repeated V



(a) Training and Validation Accuracy (b) Training and Validation Loss

Figure 7: Training and Validation Performance of Initial Model Trained on Small Dataset of 50 Samples/Class Demonstrating Overfitting



(a) Training and Validation Accuracy (b) Training and Validation Loss

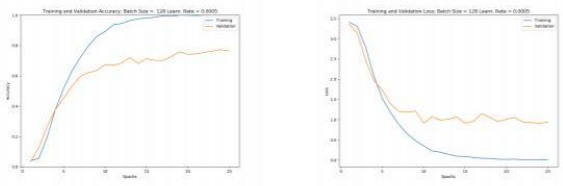
Figure 8: Training and Validation Performance of Model Trained on Original Images

8. Test the model

To evaluate the model, we'll use the test dataset. This will tell us how the network performs when classifying images it has never seen before!

If the classification accuracy on the test dataset is similar to the training dataset, this is a good sign that the model did not overfit to the training data.

In [ ]:
```
# Obtain accuracy on test set
score = model.evaluate(x=x_test,
                       y=y_test_OH,
                       verbose=0)
print('Test accuracy:', score[1])
```



(a) Training and Validation Accuracy (b) Training and Validation Loss

Figure 9: Training and Validation Performance of Model Trained on Filtered Images

Category	Unfiltered Model	Filtered Model
Kaggle Test Accuracy	76.8%	62.1%
Collected Test Accuracy	27.1%	8.03%

Table 2: Effect of Increasing Dataset Size on Validation Accuracy

IV. CONCLUSION AND FUTURE SCOPE

While this analysis sets a solid baseline for American Sign Language recognition, more work needs to be done to apply this concept in real-time. This would require further work with the LeapMotion API to enable real-time generation of data, feeding through the model, and identification of the word and/or numbers. This would also require the model to be able to handle more than the 60 class labels it currently deals with.

In conclusion, we see this application having real potential in improving the lives of the hearing-impaired and as such it would be a worthy goal to continue development.

Future Steps:

• Use Dynamic Loading for Dataset: Our original dataset was quite large and is impossible to use without a server with a lot of RAM and disk space. A possible solution is to split the file names into training, validation, and test sets and dynamically loading images in the Dataset class. Using such a loading technique would allow us to train the model on more samples in the dataset.

V. REFERENCES

1) https://www.grin.com/document/276571

2) https://www.eecg.utoronto.ca/~jayar/mie324/asl.pdf Akash. ASL Alphabet. url: https://www.kaggle.com/grassknoted/asl-alphabet. (accessed: 24.10.2018).

3) https://towardsdatascience.com/american-sign-language-recognition-using-cnn-36910b86d651

4) D. Metaxas. Sign language and human activity recognition, June 2011. CVPR Workshop on Gesture Recognition.

5) M. Ranzato. Efcient learning of sparse representations with an energy-based model, 2006. Courant Institute of Mathematical Sciences.

6) S. Sarkar. Segmentation-robust representations, matching, and modeling for sign language recognition, June 2011. CVPR Workshop on Gesture Recognition, Co-authors: Barbara Loeding, Ruiduo Yang, Sunita Nayak, Ayush Parashar.

7) P. Y. Simard. Best practices for convolutional neural networks applied to visual document analysis, August 2003. Seventh International Conference on Document Analysis and Recognition.

8) X. Teng. A hand gesture recognition system based on local linear embedding, April 2005. Journal of Visual Languages and Computing.

9) https://www.kaggle.com/rushikesh0203/mnist-sign-language-recognition-cnn-99-94-accuracy

10) https://github.com/Heisenberg0203/AmericanSignLanguage-Recognizer