Machine Learning Engineer Nanodegree

Capstone Project

Pooja Lodhi

February 28th, 2019

I. Definition

Project Overview

Communication is one important requirement for living in society [1]. Similarly, for people who are deaf and mute have sign language as their mode of communication. It's tough for normal people to understand it. So, it important to have application to understanding the sign and gesture then convert them back to language normal people can understand.

Gesture recognition has much application in improving human-computer interaction. There is various research done in this domain [1,5,6,7]. According to the literature review, there are many approaches to sign language recognition. It is mainly categorized into as sensor-based and vision based. Our main focus was on vision based. Further exploring vision based we came to know there are many different techniques or algorithm that can be used in vision-based approach. For example, Feedforward ANN, Backpropagation, Convexity defect, Convex Hull, KNN, HMM, Ad boost and Haar, PCA, SVM, Kalman filter and K-Curvature [5].

In this project we are going to design a model using transfer knowledge which able to classify the ASL Alphabets correctly. The project used the ASL dataset from Kaggle.

Problem Statement

American sign language (ASL) is one of the natural languages used for communication. This language has a different gesture for various alphabets. Thus, design an application to understand the sign and gesture then convert them back to language normal people can understand is a good problem to solve.

The goal of this project would be to design a model which accurately predict the alphabet corresponding to ASL Gesture. Then to use the corresponding model to real-life data and find its accuracy for new and unknown data.

Metrics

As the dataset is a balanced dataset thus accuracy is one of the good evaluation metrics. Accuracy in classification problems is the number of correct predictions made by the model over all kind of predictions made [10].

$$Classification\ accuracy = \frac{Number\ of\ correct\ prediction}{Total\ number\ of\ predictions}$$

There are many kernels linked to the dataset on the Kaggle website. So, we can compare our model with them.

II. Analysis

Data Exploration

ASL dataset from Kaggle is used for this project [3]. The size of the data is 1GB which has 8700 images. It is well structured in different folders. The image size is 200x200 pixels. There are 29 classes, of which 26 are for the letters A-Z and 3 classes for SPACE, DELETE and NOTHING. The Dataset is well balanced with 3000 images per class.

These images well structure to train a model to classify hand gesture to appropriate alphabet. It contains a train dataset folder and test dataset folder.

Dataset Size	1GB
Dataset Type	Dataset has image of jpeg format
Total Image Count	8700
Image Size	200X200 pixels
Classes	29
Images/Class	3000 images/class

Table 1: Dataset Summary

For testing, the dataset already has a test folder. This folder has one image that belongs to each class label. Other than that, we would prefer images from real life and other sources for testing the model. For the validation set, we might use the 10% images from the test dataset.

The sample images of each alphabet are provided in figure 1.

Figure 1: List of sample images of ASL alphabets

Exploratory Visualization

There were lots of images with variation in color intensities. This might be due to the time the photo was taken and angle the photo was taken from. The images are colorful. They have 3 channels R-Red, G-Green, B-Blue. Figure 2 below have image and a histogram corresponding to each image which provides the distribution of color in the images. In the histogram Red, Green, Blue line represent respective channels.

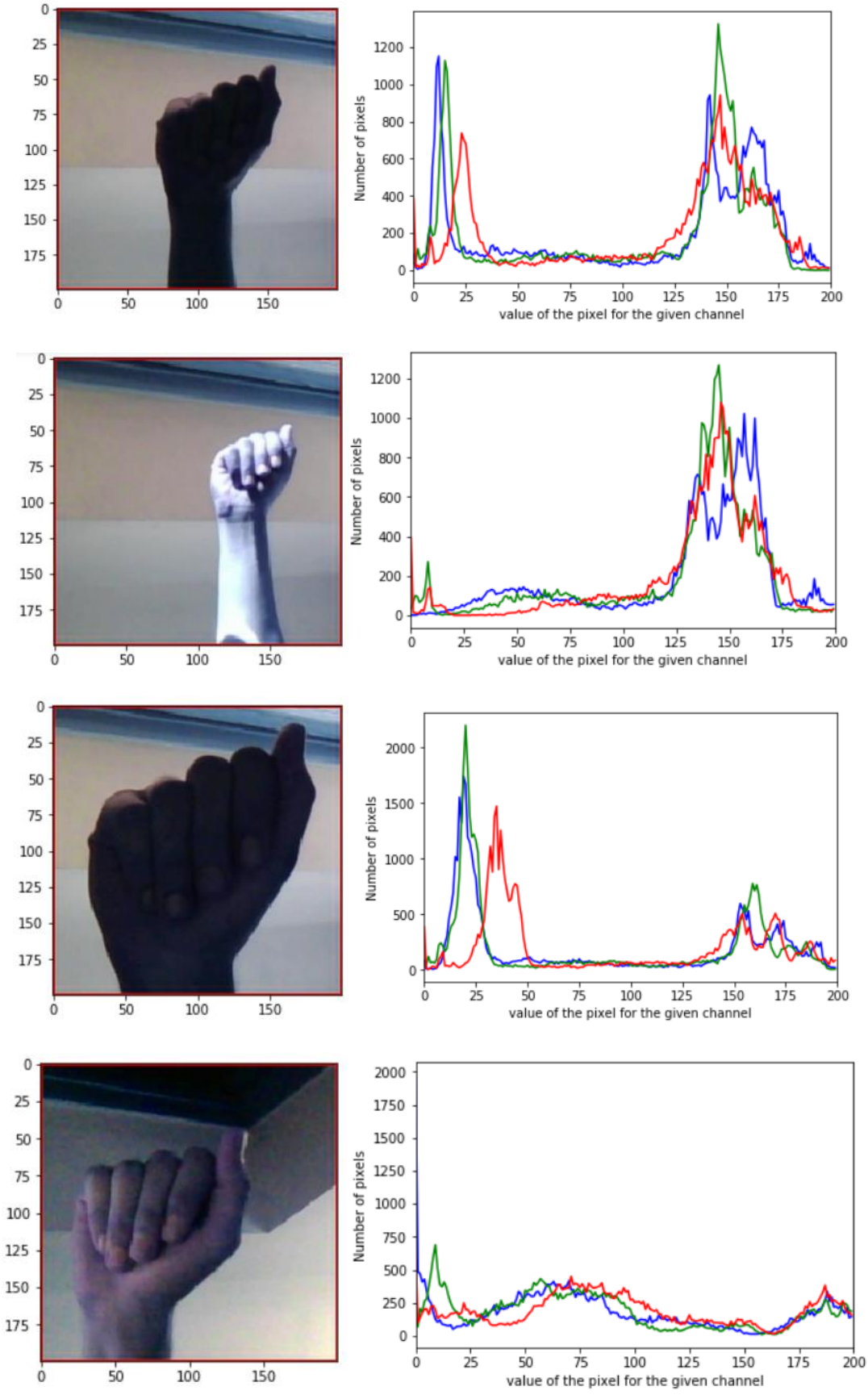


Figure 2: Histogram of colour distribution of different channels

Algorithms and Techniques

The classifier used in this project are the Convolution Neural Networks (CNN). As we are dealing with image dataset it one of the best suited techniques. A CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers can either convolutional, pooling or fully connected. When a CNN is given an input and it learns by itself that what features it has to detect. One has to not provide the initial values of features or what kind of patterns it has to detect.

CNN have various layers:

- *Convolutional* - It performs the convolutional operation over the input.
- *Pooling layers* -Pooling layers reduce the spatial dimensions of the input Volume for the next Convolutional Layer. It does not affect the depth dimension of the Volume.
- *Fully connected layer* - The fully connected is also known as Dense layer. It is fully connected with the output of the previous layer. They are typically used in the last stages of the CNN to connect with the output layer and construct the desired number of outputs.
- *Dropout layer* - The term "dropout" means to drop some units which can be both hidden and visible in a neural network. It is a regularization technique used for reducing overfitting.
- *Flatten* - Flattens the output of the convolutional layers to feed into the Dense layers.

Activation functions are used to introduce non-linear properties to our Network. Their main purpose is to convert input signal of a node in A-NN to an output signal. That output signal now is used as input in the next layer in the stack. Some activation functions that were used throughout the project are:

- *Softmax* – This function is mostly used in the final layer. It converts the input into the output of each unit to be between 0 and 1. It also divides each output such that the total sum of the outputs is equal to 1.
- *ReLu* - A ReLu (or rectified linear unit) has output 0 if the input is less than 0, and raw output otherwise. i.e, if the input is greater than 0, the output is equal to the input.

The parameters that will be used to fine tune the algorithms are as follows:

- *Training Length* - number of epochs
- *Batch Size* - number of images to train on in one step

Transfer learning is a machine learning technique where a model trained on one task is used to re-train on a second related task. Two common approaches used in transfer knowledge are *Develop Model Approach* and *Pre-trained Model Approach* [13]. In this project we have looked on various transfer knowledge models.

- *VGG-16* – It is 16-layer neural network used by the VGG team in the ILSVRC-2014 competition.
- *Xception* – It was proposed by none other than François Chollet himself, the creator and chief maintainer of the Keras library. It is an extension of the Inception architecture which replaces the standard Inception modules with depth wise separable convolutions.
- *Inception* – It is a micro-architecture was first introduced by Szegedy et al. in their 2014 paper, Going Deeper with Convolutions
- *ResNet50* – It is also called network-in-network architectures. First introduced by He et al. in their 2015 paper, Deep Residual Learning for Image Recognition.

Benchmark

In this project we ran various kernels that were already available for the ASL dataset on Kaggle's website. And also designed one benchmark model of our own. Appendix contains the list of files corresponding to each benchmark.

The kernels which were experimented were as follows:

- ***Benchmark 1***: It used 27 layers in which initial layers were Conv2D layers followed by DepthwiseConv2D and BatchNormalization layer. Then at the end it consists of GlobalAveragePooling and Dense layer. We changed the number of epochs it was trained on from 30 to 15 due to memory leaks on our local device
- ***Benchmark 2***: It used 9 layers in which first 6 layers were Conv2D layers followed by on flatten layer then at last it had 2 dense layers.
- ***Benchmark 3***: It used 9 layers in which first 6 layers were Conv2D layers followed by on flatten layer then at last it had 2 dense layers.
- ***Benchmark 4*** (Self designed benchmark): This model contains 11 sequential layers. The convolution layers were used to learn different features, where initial layers were for basic features while the layers at the end where used to understand the complex feature. All convolutional layers were followed by a max pooling layer. Then dense layers were used to get a scalar output. Figure 3 shows the architecture of the model.

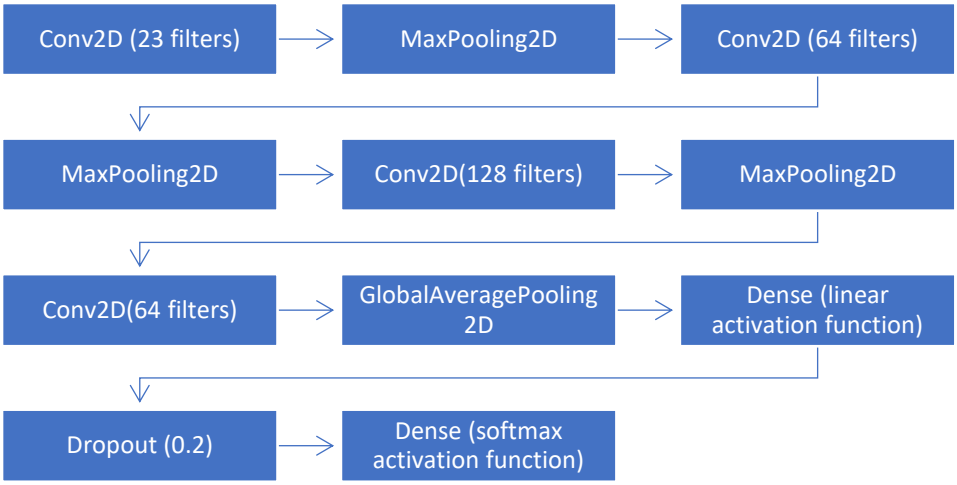


Figure 3: Architecture of benchmark 4

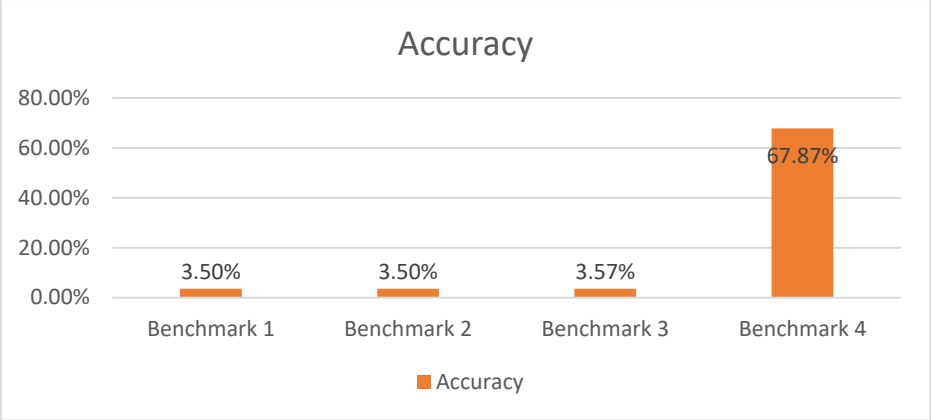


Figure 4: Accuracy of different benchmark models on test dataset

Figure 4 clearly shows that the accuracy of benchmark 4 is good for comparisons than all the other benchmark models. Thus, we are going to compare our final model with our self-designed Benchmark.

III. Methodology

Data Pre-processing

In Data Pre-processing we took two major steps that is image resizing and data augmentation. As the data is balanced and all the images are of same size thus no abnormalities need to be handled.

The images are originally of 200x200 pixels for our model we reduce it to 64x64 pixels. Thus, the final dimensions of the image were (1,64,64,3).

For data augmentation we had rotation range of 20, width shift range of 0.2 and height shift range of 0.2.

Implementation

The device used for implementation was Inspiron 15 7000, with intel core i7 8th gen and 8GB RAM.

First, we experimented and tested the benchmark models. For this we selected some already existing models for that dataset on Kaggle website. We ran them locally and tired to gather the results. As these models were not performing up to mark then we decided to design one benchmark model on our own. The results of benchmark models are already discussed in analysis section above.

Then, as per our proposed workflow, we looked into different pretrained models. They all were trained keeping all the parameters same. Which helped us in identifying the model which performs best for the given dataset.

Batch Size	64
Validation Split	0.1 (10%)
Epochs (model without weight)	20
Epochs (model with pre-trained weight)	39
Steps per epoch	39

Table 2: List of constant parameters

Using the above parameters, we trained VGG-16, Xception, ResNet50. For all the models we added a top layer which take input in (64,64,3) dimension and a bottom layer which gives output in 29 different classes. While working on inception it gave some errors which we were not able to resolve.

The results of the following models are provided in the graph below.

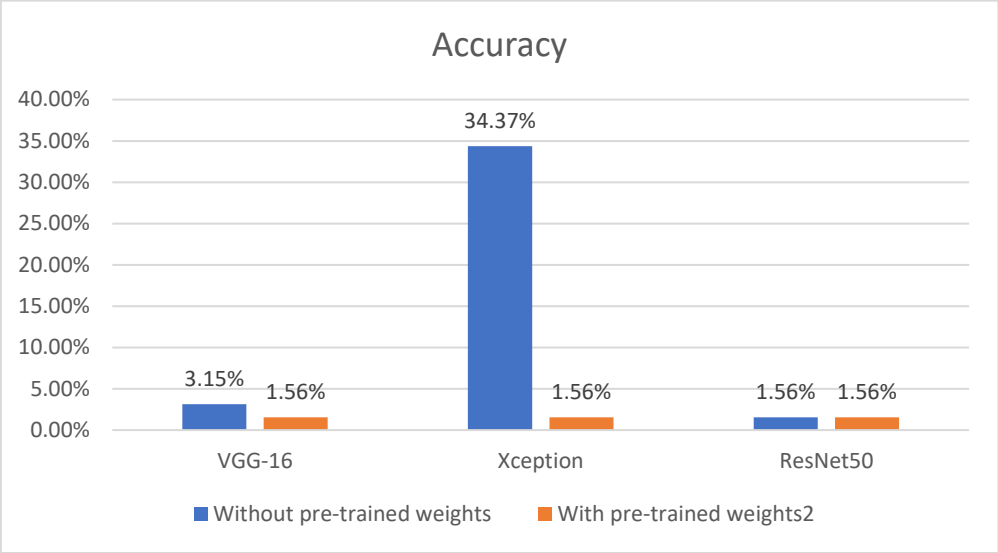


Figure 5: Accuracy of different transfer learning models on validation dataset

After analysing the results, from figure 5 we can conclude the xception model is best for this scenario.

So, we used Xception model without pre-trained weights as our **Final Model.** Then we further refined that model by tuning different parameters.

Refinement

The parameters that were used to refine the models were the steps per epochs and the epochs.

Initially we had number of epochs as 20 with the steps per epochs as 39. This led to an accuracy of 34.37% on validation set. Let's us name it as **refinement 1**.

Then we tired by changing the number of epochs to 10 with steps per epochs as 39 it led to an accuracy of 17.18 % on the validation. Let's us name it as **refinement 2**.

Then we thought that may be increasing the steps per epochs might increase the accuracy so we changed it to 77 but lead to an accuracy of 15.62% on the validation set. Let's us name it as **refinement 3**.

Then later we thought to keep the steps per epochs as 39 and changed the number of epochs to 10, 20, 50, 100, 150. On training our model with 50 epochs the final validation accuracy that we got was 78.12%. Let's us name it as **refinement 4**. Then we changed the epochs to 100 which lead to system error so reduced it to 90 epochs. The 90 epochs model showed 89.06% accuracy. Let's us name it as **refinement 5**.

Later we also tried keeping half of the images in a batch by making the steps per epochs equal to the length of train generator divided by 2. That result into the accuracy of 96.87%. Let's us name it as **refinement 6**.

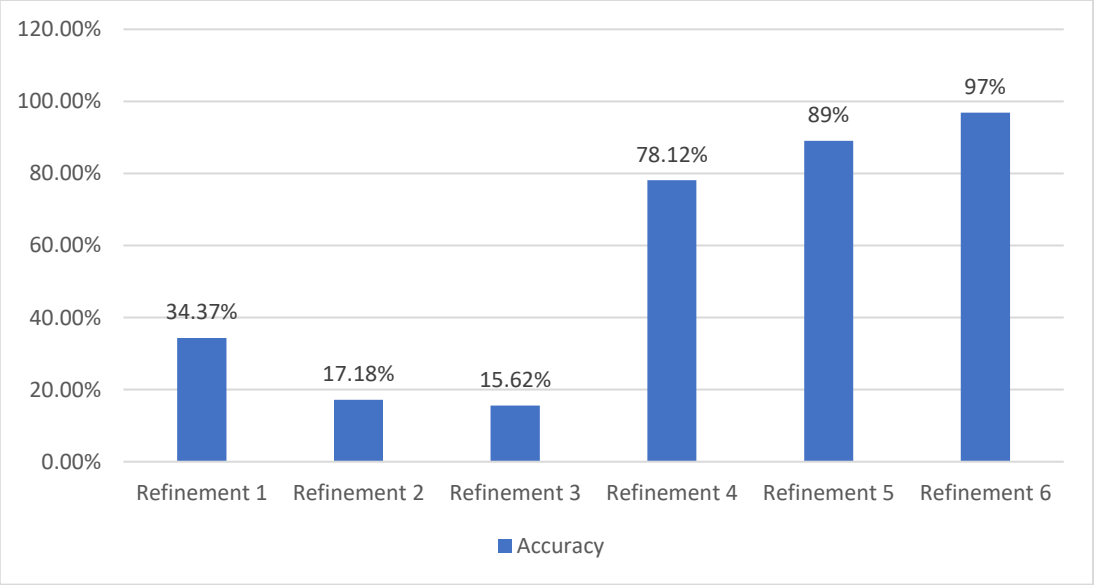


Figure 5: Accuracy of different refinement models on validation dataset

IV. Results

Model Evaluation and Validation

Validation set was used during the development of model to evaluate performance of the given model. Thus, the final model and hyperparameters were selected because they performed better than others.

The final model which out performs all the others. It was a Xception model with following hyperparameters, number of epochs were 10, the steps per epoch were 612 and rest of the fields were as discussed in the previous sections.

Figure 6.2 below plots the training and validation loss for the final model. While figure 6.1 shows the accuracy of the model on the training and validation set.

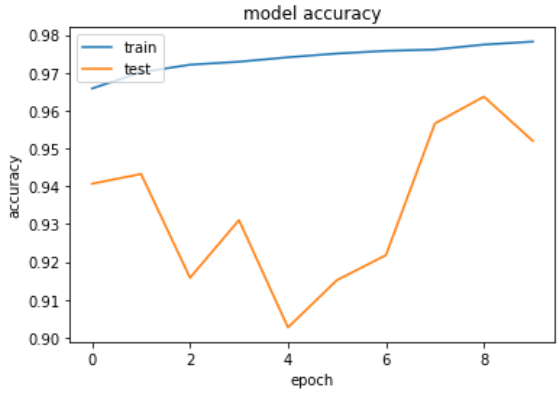


Figure 6.1: Model Accuracy

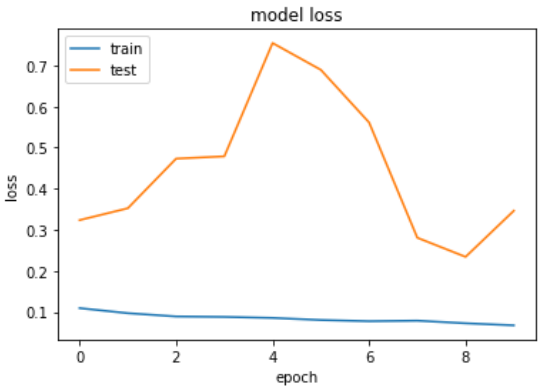


Figure 6.2: Model loss

Justification

The final model performed far better than benchmark model. The statistics can be clearly seen in figure7 below.

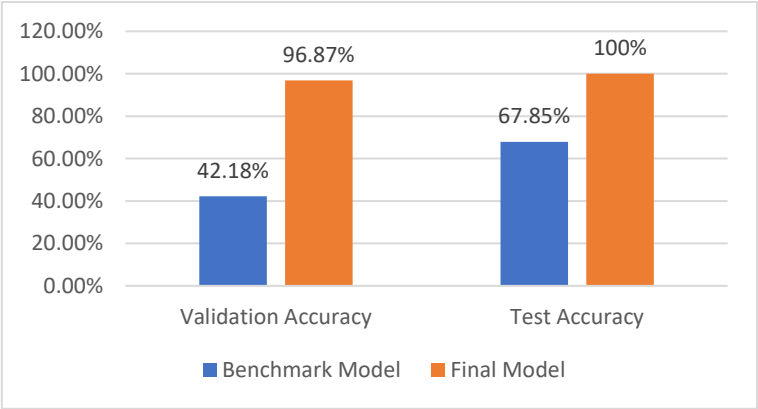


Figure 7: compares the validation and test accuracy of final and benchmark model

It is believed the final solution will surely contribute towards solving the given problem significantly. Further training it with more realistic data might lead to improved solution.

V. Conclusion

Free-Form Visualization

The model was tested on some randomly selected real-life images. And some of the samples are given below. The final model was not able to detect the hand gesture correctly if there is something else is present in the image other than hand gesture. Secondly if image quality is very poor then also model did incorrect classification.

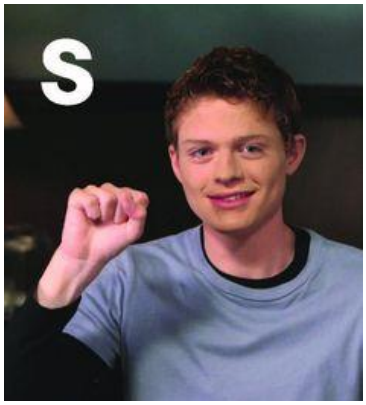
B

O

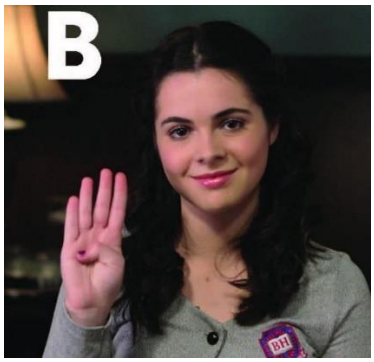
L

W

S



Q



M

P

J

All the results of real dataset is provided in final_code.ipynb.

Reflection

This section would discuss the interesting or difficult things encountered in the project. And would summarize the flow followed in the project.

I always wanted to select a problem that would provide me with better understanding, clarity and exposure on how to deal with image classification problem. And the selected problem statement was a justified choice.

We can summarize the process used in this project was as follows:

- Firstly, a relevant dataset and problem statement was decided.
- Then the dataset was downloaded and the project virtual environment was setup using anaconda navigator.
- Then the benchmark models were found and tested. As the already existing models did not gave the desired results, leads to shifting on creating our own model.
- Then we experimented on transfer knowledge models and chosen the best one.
- At last we fine-tuned the final model to get the better result.
- Later, the final model was used to do prediction on real life data.

The issue I faced during training various models were system issues like memory overflow or page fault. Other than that, during the initial setup of the virtual there were conflict in libraries and their dependencies.

Also understood that, it is computationally very expensive to train big models on CPUs and requires GPUs which can run several processes parallelly.

The final model performs better than my benchmark. Now I have a better understanding of some concepts about CNN, transfer learning and deep learning and definitely, I will look forward doing some more projects to learn more about them.

Improvement

The few was by which solution can be improved are as follows:

- Training the model on real life data which contains noise.
- Data argumentation parameters can be changed to get more diverse dataset.
- As training a CNN it was found that it requires a lot more computing power than any traditional approaches and we really need to take care of our memory consumptions, because there were moments when we ran out of memory.
- Make the model efficient enough, so that it can take image of any size and able to detect a hand and then the hand gesture from the given image.

Appendix

Benchmark 1	Benchmark Code/99-9-asl-alphabet-classification-with-slimcnn.ipynb
Benchmark 2	Benchmark Code/classifying-images-of-the-asl-alphabet-using-keras.ipynb
Benchmark 3	Benchmark Code/running-kaggle-kernels-with-a-gpu.ipynb
Benchmark 4	Code/Main.ipynb
VGG-16	Code/Main1-VGG16.ipynb
Inception	Code/Main1-inception.ipynb
Xception	Code/Main1-xception.ipynb
ResNet	Code/Main1-resnet.ipynb
Final Model	Final Model/Final_Code.ipynb

References

[1] Sanil Jain, K.V.Sameer Raja “Indian Sign Language Character Recognition”

[2]https://www.analyticsvidhya.com/blog/2018/07/top-10-pretrained-models-get-started-deep-learning-part-1-computer-vision/

[3] https://www.kaggle.com/grassknoted/asl-alphabet

[4] https://keras.io/applications/

[5] Shivashankara S, Srinath S “A comparative Study of Various Techniques and Outcomes of Recognizing American Sign Language: A Review”, International Journal of Scientific Research Engineering & Technology, September 2017

[6] Nachamai. M, “Alphabet recognition of American Sign Language: Hand gesture recognition using SIFT Algorithm”, International Journal of Artificial Intelligence & Applications (IJAIA), Vol.4, No.1, January 2013

[7] Brandon Garcia, Sigberto Alarcon Viesca, “Real-time American Sign Language Recognition with Convolutional Neural Networks”

[8] Cao D., Ming C.L., Zhaozheng Y. American Sign Language alphabet recognition using microsoft kinect; Proceedings of the 2015 IEEE Conference on CVPRW; Boston, MA, USA. 7–12 June 2015

[9] N. Pugeault, and R. Bowden. Spelling It Out: Real-Time ASL Fingerspelling Recognition. 2011 IEEE Workshop on Consumer Depth Cameras for Computer Vision, pp. 1114-1119, 2011

[10]https://medium.com/greyatom/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d432082b

[11] http://wiki.fast.ai/index.php/Log_Loss

[12] Benchmark Models Experimented on:
https://www.kaggle.com/danrasband/classifying-images-of-the-asl-alphabet-using-keras
https://www.kaggle.com/dansbecker/running-kaggle-kernels-with-a-gpu
https://www.kaggle.com/kairess/99-9-asl-alphabet-classification-with-slimcnn

[13] https://machinelearningmastery.com/transfer-learning-for-deep-learning/

[14]https://medium.com/@arindambaidya168/https-medium-com-arindambaidya168-using-keras-imagedatagenerator-b94a87cdefad

[15]https://medium.com/@vijayabhaskar96/tutorial-image-classification-with-keras-flow-from-directory-and-generators-95f75ebe5720