

# Machine Learning Engineer Nanodegree

## Capstone Project

---

Mahmoud Nada  
September 18th, 2019

### I. Definition

---

#### Project Overview

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.

A lot of recent progress has been made towards developing computer vision systems that translate sign language to spoken language. This technology often relies on complex neural network architectures that can detect subtle patterns in streaming video. However, as a first step, towards understanding how to build a translation system, we can reduce the size of the problem by translating individual letters, instead of sentences.

In this project the dataset is a collection of images of alphabets from the American Sign Language, separated in 29 folders which represent the various classes. The training data set contains 87,000 images which are 200x200 pixels. There are 29 classes, of which 26 are for the letters A-Z and 3 classes for SPACE, DELETE and NOTHING, each 29 train classes has 3000 images, these 3 classes are very helpful in real time applications, and classification. The test data set contains a mere 29 images, to encourage the use of real-world test images. This dataset is publicly available at Kaggle in this link:

<https://kaggle.com/grassknotted/asl-alphabet>



## Problem Statement

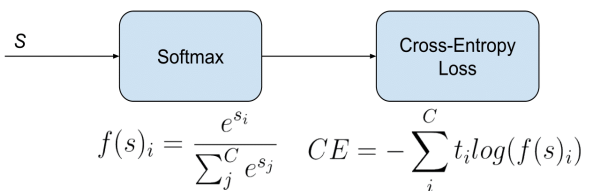
Our problem is that deaf individuals in America have always hard time describing what they want to someone who is not familiar with ASL. So, the goal is to build a model that can recognize ASL Letters in real time so that it makes it easier for them to communicate in simpler and more efficient way, the tasks involved are the following:

1. Pre-Process the ASL Dataset (resize the images, rescaling, normalization, one hot encoding, etc ...)
2. Split the images into training and testing images.
3. Create CNN model to be trained on the training images.
4. Use multiple transfer learning models to speed up the training process.
5. Save the weights of the best models' results to use them in real time Translation.
6. Use OpenCV to make a python script that tracks hand palm translates the sign language in real time.

The final application is expected to be helpful and useful and will be able to interpret ASL with any camera instantaneously.

## Metrics

This is a multi-class classification problem, according to this model's characteristics I'll use Categorical Cross-Entropy loss (which is the most popular loss function at multi-class classification) to measure performance of my model and also use Accuracy to measure the algorithm's performance in an interpretable way, the Categorical Cross-Entropy Loss is defined as below:

$$L(y, \hat{y}) = - \sum_{j=0}^M \sum_{i=0}^N (y_{ij} * \log(\hat{y}_{ij}))$$


The diagram illustrates the process of calculating the Cross-Entropy Loss. It starts with an input  $S$  entering a box labeled "Softmax". Below this box is the formula  $f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}}$ . An arrow points from the "Softmax" box to another box labeled "Cross-Entropy Loss". Below this second box is the formula  $CE = - \sum_i^C t_i \log(f(s)_i)$ .

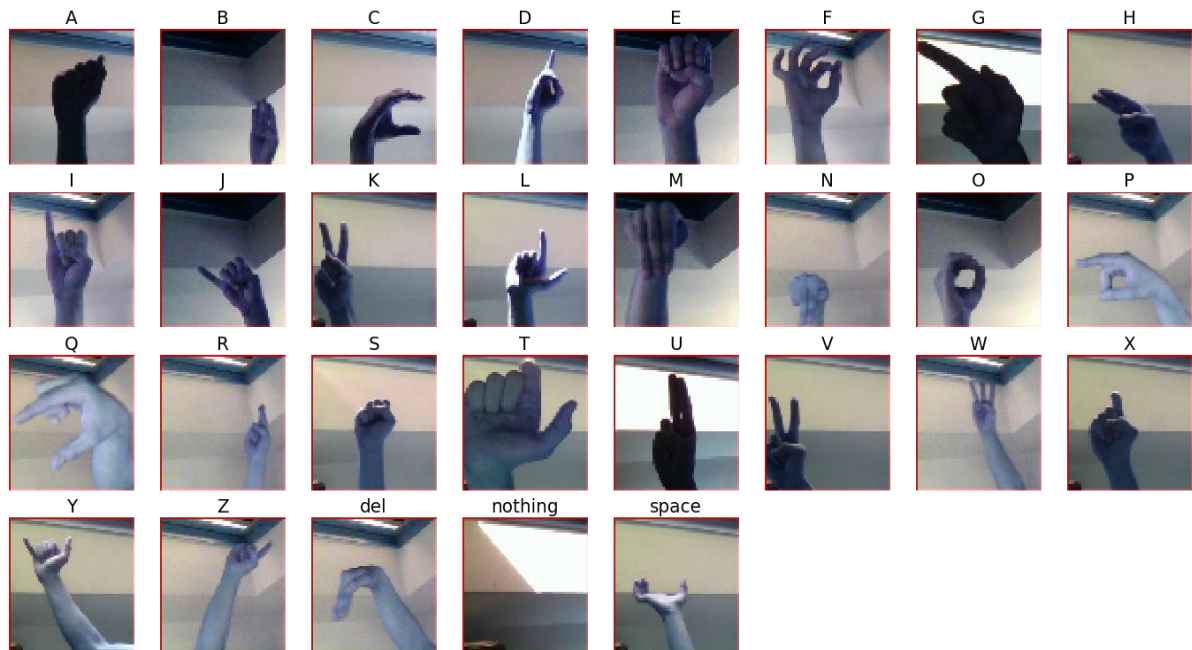
where  $\hat{y}$  is the predicted value, Categorical Cross-Entropy will compare the distribution of the predictions (the activations in the output layer, one for each class) with the true distribution, where the probability of the true class is set to 1 and 0 for the other classes. To put it in a different way, the true class is represented as a one-hot encoded vector, and the closer the model's outputs are to that vector, the lower the loss.

## II. Analysis

---

### Data Exploration

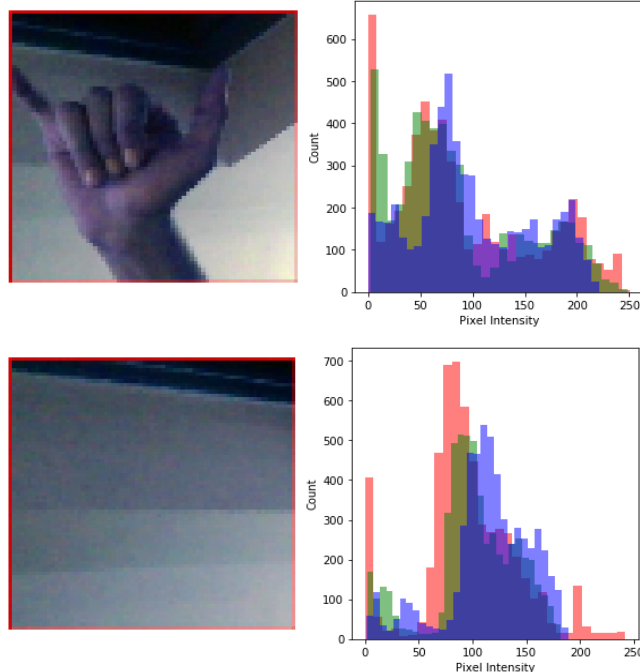
The ASL dataset from Kaggle has 87,000 coloured images which are 200x200 pixels. There are 29 classes, of which 26 are for the letters A-Z and 3 classes for SPACE, DELETE and NOTHING, each 29 train classes has 3000 images.



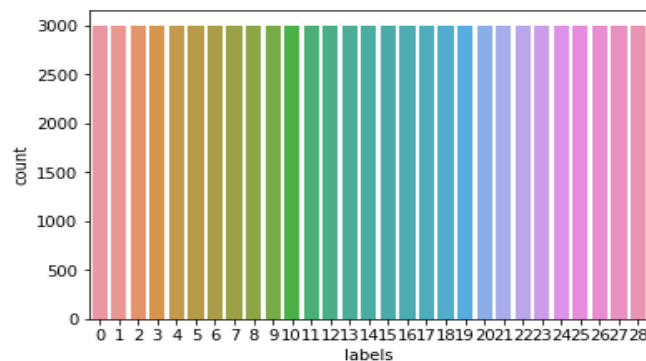
This dataset has a huge amount of images with various illumination condition some of them in the light and others in shadow, this will help the model to accurately predict testing data from the same dataset but it's predicted to perform poorly on images from other backgrounds are hand colour ,etc. thus, I will try to include more varied data in our dataset later in order to get better results with real-world images.

## Exploratory Visualization

1. There lots of data images but as mentioned before they are all taken from the same place so they don't have lots of variation in pixels colour intensity as we can see below most of the images has higher blue pixels intensity.



2. The training data provided for this competition was distributed uniformly among the different classes of the sign letters with 3000 images at each class.



## Algorithms and Techniques

1. **Deep Learning** - Deep learning (also known as deep structured learning or hierarchical learning) is part of a broader family of machine learning methods based on learning data representations, as opposed to task-specific algorithms.
2. **Convolutional neural network** - In machine learning, a convolutional neural network (CNN or ConvNet) is a class of deep, feed-forward artificial neural networks that has successfully been applied to analysing visual imagery. A

CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers are either convolutional, pooling or fully connected. We give CNN an input and it learns by itself that what features it must detect. We won't specify the initial values of features or what kind of patterns it must detect.

Various Layers of CNN: -

- **Convolutional** - Also referred to as Conv. layer, it forms the basis of the CNN and performs the core operations of training and consequently firing the of the network. It performs the convolutional operation over the input.
  - **Pooling layers** - Pooling layers reduce the spatial dimensions (Width x Height) 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 or **Dense** layer is configured exactly the way its name implies. It is fully connected with the output of the previous layer. Fully connected layers are typically used in the last stages of the CNN to be connected to the output layer and construct the desired number of outputs.
  - **Dropout layer** - Dropout is a regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data. It is a very efficient way of performing model averaging with neural networks. The term "dropout" refers to dropping out units (both hidden and visible) in a neural network.
  - **Flatten** - Flattens the output of the convolutional layers to feed into the Dense layers.
  - **Batch Normalization** - enables the use of higher learning rates, greatly accelerating the learning process. It also enabled the training of deep neural networks with sigmoid activations that were previously deemed too difficult to train due to the vanishing gradient problem
  - **Global Average Pooling (GAP)** - layers to minimize overfitting by reducing the total number of parameters in the model. Similar to max pooling layers, GAP layers are used to reduce the spatial dimensions of a three-dimensional tensor. However, GAP layers perform a more extreme type of dimensionality reduction.
3. **Activation Functions** - In CNN, the activation function of a node defines the output of that node given an input or set of inputs. Some activation functions are:

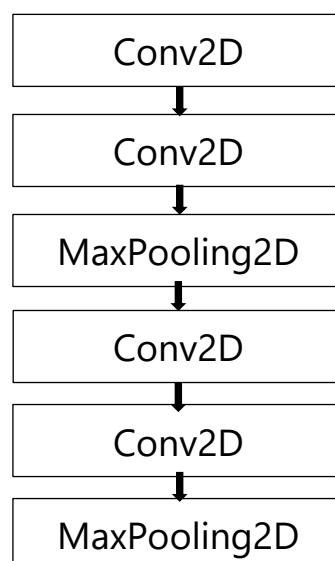
- **Softmax** - The softmax function squashes the output of each unit to be between 0 and 1, just like a sigmoid function. It also divides each output such that the total sum of the outputs is equal to 1.
  - **ReLU** - A ReLU (or rectified linear unit) has output 0 if the input is less than 0, and raw output otherwise. i.e., if the input is greater than 0, the output is equal to the input.
4. **Transfer Learning** - In transfer learning, we take the learned understanding and pass it to a new deep learning model. We take a pre-trained neural network and adapt it to a new neural network with different dataset. For this problem we will use multiple transfer learning models to see which solves our problem the best.
- **VGG16**
  - **VGG-19**
  - **ResNet**
  - **MobileNet**
  - **MobileNetV2**

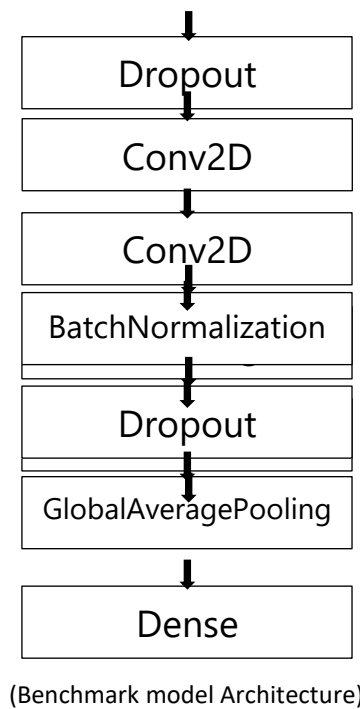
Our pre-processed images will go straight to the CNN model that will be trained on them.

## Benchmark

At this project I built a CNN model From scratch and considered the metrics of accuracy at testing data as benchmark result, this result is obtained by training a model then evaluating its performance on data that it was not trained on before to see how will it performs.

Model architecture for Benchmark: -





This bench mark is not yet trained so I will assume that it has Zero accuracy.

### III. Methodology

---

#### Data Pre-processing

Pre-processing the data to be ready for the training process went as below:

1. Loading the images from its original directories.
  - I have 29 classes each in a separate directory containing their own images
2. Resizing the images to 64x64 pixels to make the training less computationally heavy
  - Training a model with 200x200 pixel images will be computationally heavy and requires more RAM and GPU units.
3. The images are divided into a training and testing sets using validation split of 0.1.
4. Sort the training and testing sets to be able to properly plot the samples of images in organized form.
5. One hot encoding the labels of training and testing sets

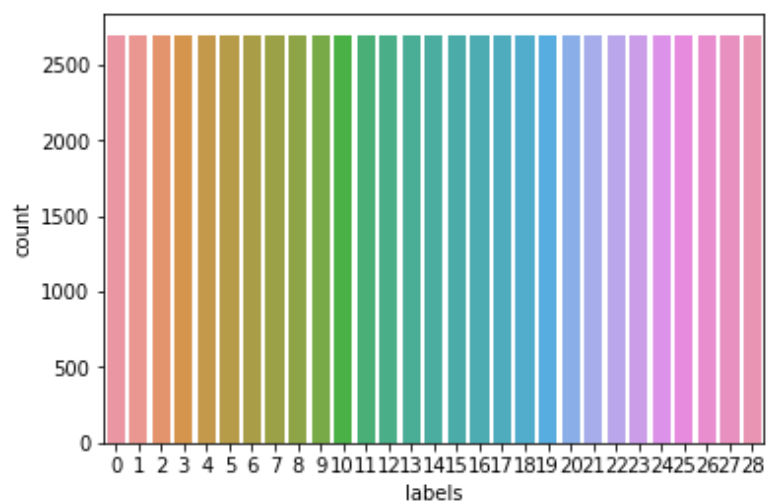
- This conversion will turn the one-dimensional array of labels into a two-dimensional array. Each row in the two-dimensional array of one-hot encoded labels corresponds to a different label. The row has a 1 in the column that corresponds to the correct label, and 0 elsewhere.
6. Normalize RGB values of our sets
- we divide by 255 the range can be described with a 0.0-1.0 where 0.0 means 0 (0x00 in hex) and 1.0 means 255 (0xFF in hex). Normalization will help us remove distortions caused by lights and shadows in an image. This is a good idea for our dataset as we have seen that our images have a lot of different light and shadow areas.

After these steps of pre-processing our data now it's ready to be trained.

## Implementation

After pre-processing the data what left was:

1. more visualising of our data nature as below:
  - visualising samples of the training and testing data with the helper function `plot_images` its output was provided in Data Exploration part.
  - the distribution of images in each training class which was uniform with same number of images in each class.



2. Creating a CNN that will be trained on the training data
  - First, an initial benchmark CNN model architecture was defined and then trained on the pre-processed training data using validation split of 0.1
  - All our convolutional layers have a Relu activation function and kernel size of 5 and padding parameter as 'same' and various number of filters each.



- After each 2 Convolutional layers I added a MaxPooling2D layer with pool size of 4x4.
  - I added two more layers which are BatchNormalization and GlobalAveragePooling2D.
  - I used one final Dense layer with 'Softmax' activation function.
  - This model was at last compiled with Categorical Cross-Entropy loss function and Adam optimizer with accuracy metrics.
  - A check pointer that keeps track with the training procedure and saves the best models weights constantly within each epoch in an H5 file that can be restored and used again any time later for example the real time interpreting was added.
  - An early stopping technique also was added to the model, it's simply a monitor that tracks accuracy or validation loss of the model at each epoch, this is added to save time in case the model is training with no progress in the accuracy it halts the training operation after a pre-specified duration.
  - At this point everything is ready for the training and this is where I start the training or fitting the model with batch size of 64 and 10 epochs.
  - The accuracy of the model will vary a lot in training, so I changed multiple hyper-parameters to get to the best accuracy.
  - At last the best model that I figured was the one described above.
3. Using Transfer learning methods to try to improve the performance of our model:
- First, I imported a model from keras application library and removed its final fully connected layer and replaced it with custom Dense of mine, I also added a Dropout and GlobalAveragePooling2D to this imported model
  - The models I imported and tried where
    - i. **VGG16**
    - ii. **VGG-19**
    - iii. **ResNet**
    - iv. **MobileNet**
    - v. **MobileNetV2**
  - I found that the model with best validation and training accuracy was VGG16
  - The image size in our sets at this phase is 64x64 this prevented me from using other pre-trained model like **Inception** as it requires minimum input shape of 75x75, this was not easy to obtain because I use google colab and its really slow at file handling and reading

images also when I tried to do so the RAM was entirely consumed and the session crashed.

#### 4. Real time interpretation with webcam

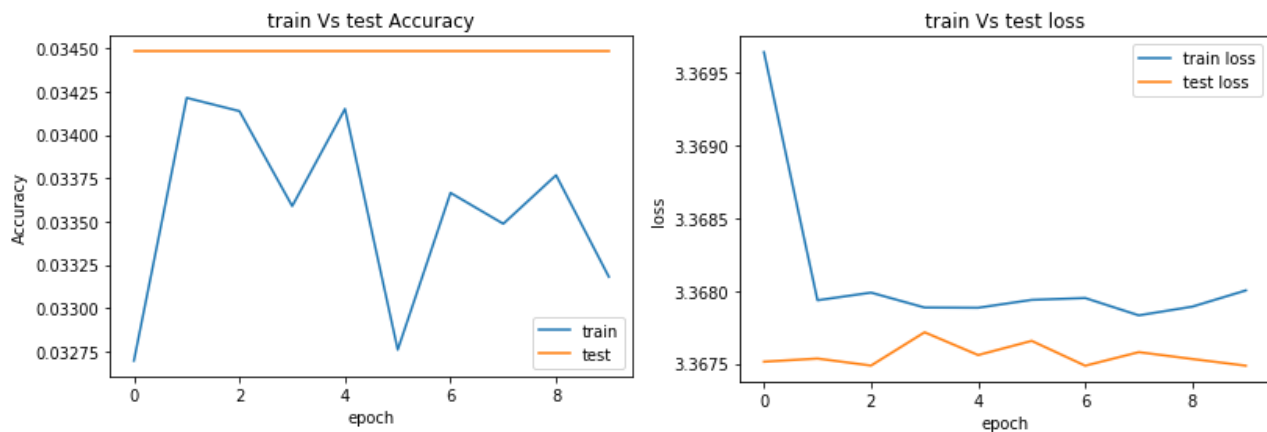
- After training a model with satisfying accuracy I saved this model so I can use in this task.
- I used OpenCV to capture live feed from webcam.
- From this live feed I marked a certain region for me to position my palm at then cropped this region, resized it to 64x64 pixels, normalized these resized frames and sent it to our model.
- The model returns the prediction of the received frames it gets then we put the prediction on the screen.

Many difficulties occurred at implementation, as I mentioned earlier I used Google colab to train my model at, as it provides free 12 GB of RAM and a great Nvidia Tesla GPU unit, that helped a lot in training, but since I'm now importing data from google drive this process took too much time (around 7 hours) to get all 87000 images imported, that was fixed by importing the images on my local computer (it took around 20 minutes) then pickled this data and uploaded it to google drive this was also difficult the data was around 1 GB and it was just a 64x64 images, I tried to increase the size so I can use other transfer learning models, 200x200 pixel data was pickled by a library called joblib the data was 10 GB and it was impossible to upload such a file to google drive, so I tried the size of 75x75 which was the minimum input shape of almost all other transfer learning models, the pickled data was 2 GB I managed to upload it to google drive after Zipping it, another problem raised at this point, after importing the pickle file to colab half of the RAM was consumed , after normalizing the images and dividing them by 255, almost all RAM was consumed and the training process was not able to complete and the session was crashing every time, so eventually I settled with the 64x64 images data.

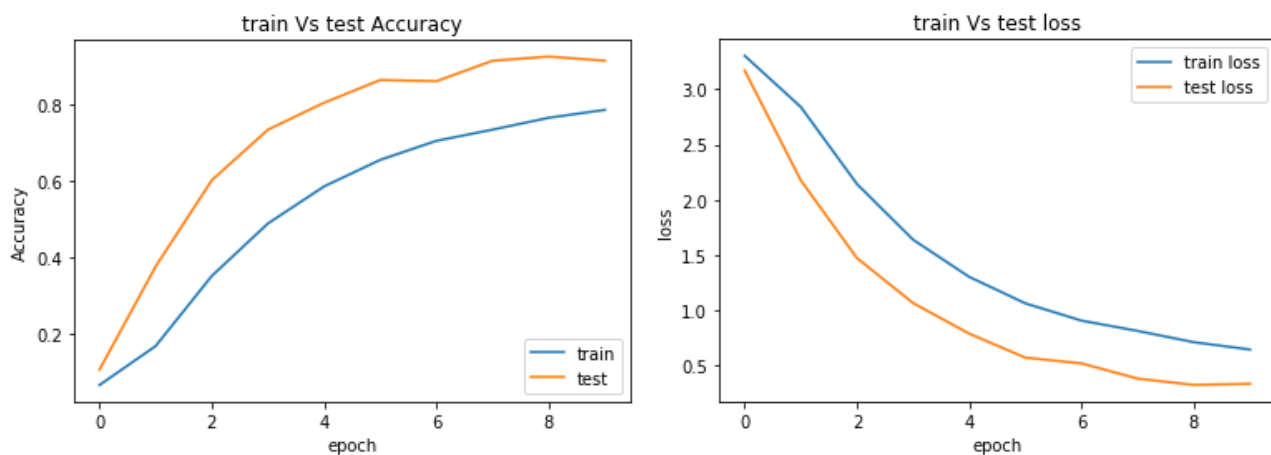
## Refinement

My model had so many phases from very low accuracy at training and testing set let us discuss in detail these phases:

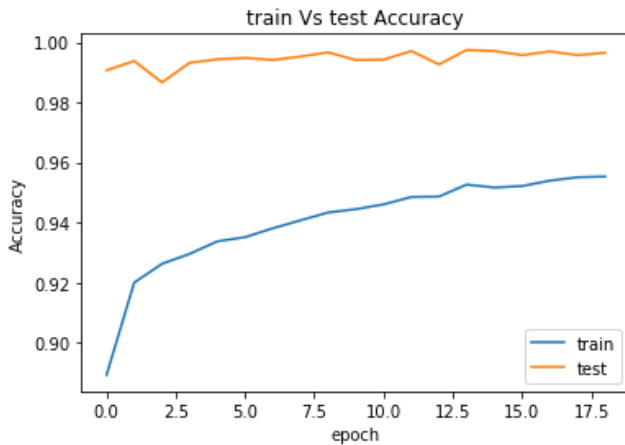
- First as mentioned in implementation I used Convolutional layers with kernel or window size of 5x5 and MaxPooling2D with pool size of 4x4 this model was trained with Adam optimizer, Categorical Cross-Entropy and accuracy as a metric it also has a patch size of 64 and 10 epochs this model was incredibly bad as the accuracy did not even get to 5 %



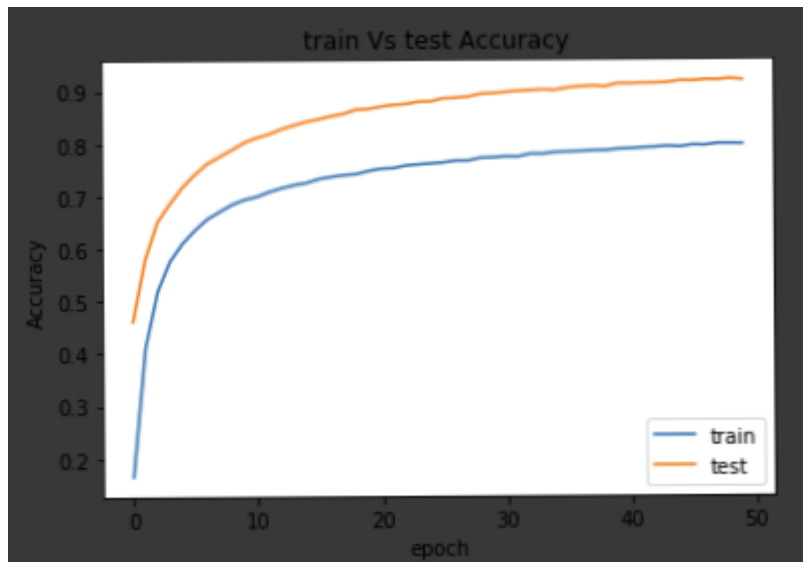
- I had many attempts to increase the performance with hyper-parameter tuning for example, I tried multiple optimizers like (sgd and rmsprop) also varied the convolutional layers filters, I added and removed dense layers, also added Dropout layers with different configurations of dropout value, I also varied the kernel size and pool size.
- After a while of tuning there were hope, the model with kernel size of 3x3 and pool size of 3x3 made a huge success with accuracy of 95% at testing set.



- At last the best model had a slight change of the one above the only change is I made the batch size equals 128 and the accuracy was great, it has an accuracy of 99.965 % at the testing set.



- I also used transfer learning to have more accuracy that can help in real world frame by frame data, so I tried all the pre-trained models as mentioned above and the one with the highest accuracy was the VGG16 model with accuracy of 95.6%.

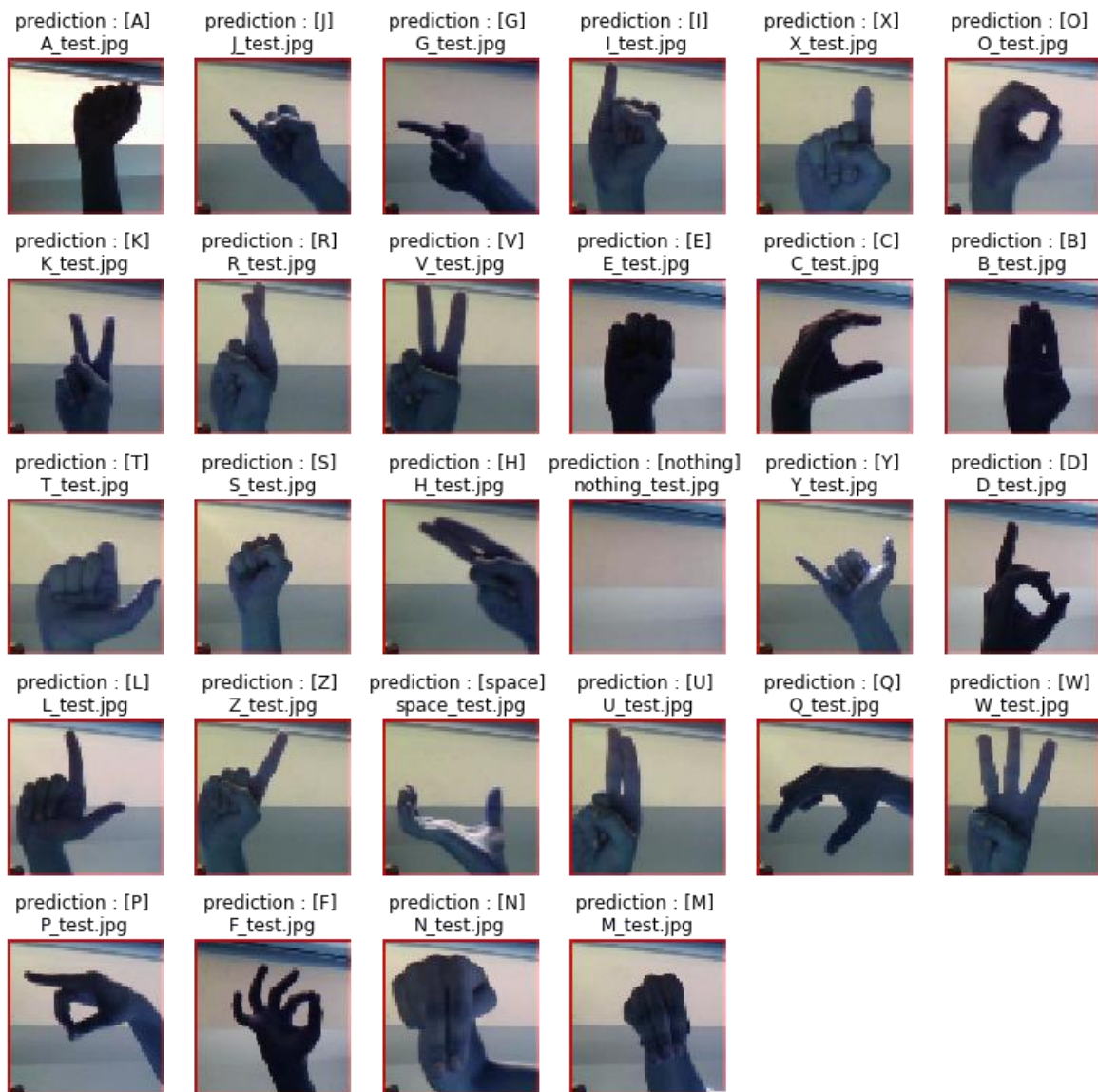


## IV. Results

### Model Evaluation and Validation

During development, a validation set was used to evaluate the model. The final architecture and hyperparameters were chosen because they performed the best among the tried combinations.

This validation set images were not in the original images from the far beginning and yet our model managed to predict all of them correctly as we can see below.



This is also a complete description of the final model and training process:

- All my convolutional layer's filters have the shape of 3x3, padding set to same, Relu activation function, default stride length of 1 and input shape equals (64,64,3)
- All MaxPooling2D layers have 3x3 pool size
- First two convolutional layers learn 8 and 16 filters followed by a MaxPooling2D layer.
- second two convolutional layers learn 16 and 32 filters followed by a MaxPooling2D layer and a Dropout layer with 0.5 probability.
- third two convolutional layers learn 32 and 64 filters followed by a MaxPooling2D layer and a Dropout layer with 0.5 probability.
- I added a BatchNormalization layer followed by a Dropout layer with 0.5 probability and GlobalAveragePooling2D.

- At the end there is a fully connected layer or dense layer that has 29 output and a Softmax activation function.

This final model with these final parameters is appropriate and been tested on various inputs with good generalization of the unseen data.

Also, when I tested this model on real images from webcam it did well so this model can be trusted as we can see in the Free-Form Visualization below.

## Justification

The final model performs better than the benchmark and other models as well. Where benchmark model gets an accuracy below 5% the final model made a significant improvement with accuracy of 99.96%.

This final solution provides a real time ASL interpreter with acceptable accuracy at real world and amazing accuracy in the training and validation sets, this solution is significant enough to solve the problem as we can see below in real world data.

## V. Conclusion

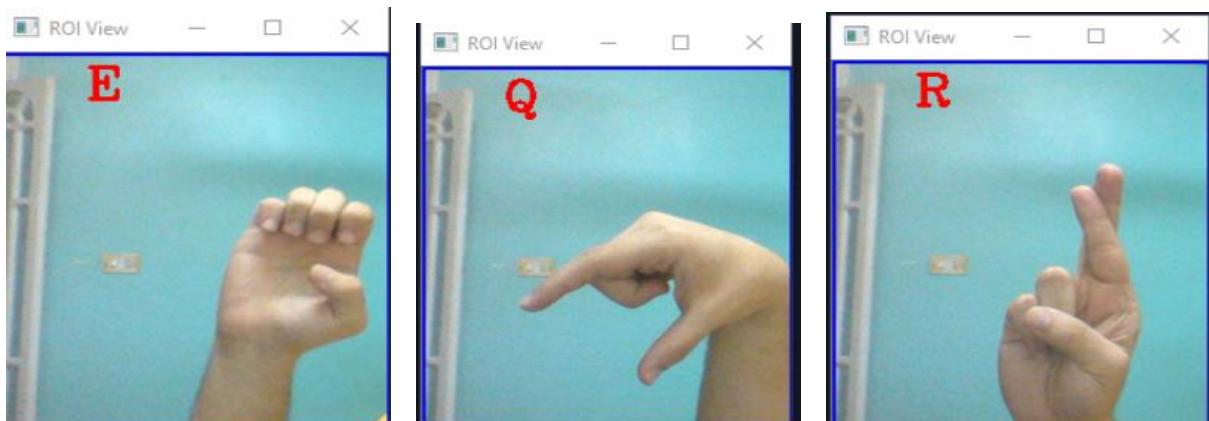
---

### Free-Form Visualization

These are examples of the final project sign letters translated in real time and interpreted at the top left of the screen.

The translation process is fast and there are no lags observed on the live feed.

The real world data may have some letters that is not well translated or needs a specific gesture just like the provider of the dataset have captured and the model trained on.





## Reflection

The process used for this project can be summarized in the following steps:

1. An initial problem and relevant, public datasets were found
2. The data was downloaded and pre-processed
3. A benchmark was created for the model
4. The model was trained using the data (multiple times, until a good set of parameters were found)
5. an OpenCV script was built to take live feed from webcam
6. The application was extended so that it can translate sign language in real time with real world data.

I found steps 4 and 5 the most difficult, as I had to familiarize myself with OpenCV and the best ways to save my trained models weights, also faced lots of difficulties in importing the huge amount of data we have in our dataset.

As for the most interesting aspects of the project, I'm very glad that I combined my knowledge of machine learning and computer vision to make a useful project, I'm also happy to use OpenCV as I believe that computer vision will be more and more interesting and super useful in the future.

I believe that the final model and this solution has met my expectations for the problem, and it should be used in a general setting to solve these types of problems.

## Improvement

There are so many improvements that can be done to this project:

- first we can add more data from different environments to have more generalized model that can efficiently work anywhere, I tried to make a custom dataset from many sources but I had a problem, the dataset was huge

and needs more computation power than the one provided by google colab, so we will need more RAM and GPU for training this model

- we also can add hand tracking models to track the position of the user's hand at every moment then send to our model to predict the sign letter, this can be done easily by TensorFlow object detection API but it also needs high computation power, and datasets of hands which are available and easy to get.
- This was step one to help people with hearing problems the next step would be translating sentences in real world for these people.
- The last step is to take these translated sentences from the last step and feed them to a TTS model to generate sound, this would be so helpful to millions of people around the world, and not just we can be limited to ASL we can make a model for other languages like Arabic sign language(ArSL)
- An addition can be made is that we can deploy this model to a smart phone so it can be at the reach of everyone
- We can also make the video calls translate the sign language this would make communicating through phones not just limited to texting

After all there can be tons of improvements that would be significantly helpful to humanity.

I would gladly proceed after this step to make this model more useful and helpful to the real world.

---

## References

- <https://www.kaggle.com/grassknotted/asl-alphabet>
- <https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy>
- [https://gombru.github.io/2018/05/23/cross\\_entropy\\_loss/](https://gombru.github.io/2018/05/23/cross_entropy_loss/)
- <http://www.jussihuotari.com/2018/01/17/why-loss-and-accuracy-metrics-conflict/>
- <https://arxiv.org/abs/1409.1556>
- <https://keras.io/applications/>
- <https://mlexplained.com/2018/01/10/an-intuitive-explanation-of-why-batch-normalization-really-works-normalization-in-deep-learning-part-1/>
- Udacity's Deep Learning Lessons - MLND