Sign Language Detection Using CNN

Team members:

- **Chandra Teja Kommineni**
- **Rajesh Karumanchi**

Abstract:

Speech impairment is a disability which affects an individual's ability to communicate using speech and hearing. People who are affected by this use other media of communication such as sign language. Although sign language is ubiquitous in recent times, there remains a challenge for non-sign language speakers to communicate with sign language speakers or signers. 3 CNN models were developed that will detect & classify the hand sign images the corresponding alphabets. Keras was used for training of images. The models acquired a testing accuracy of 91.6%, 89.6% & 93.8%. To further improve the accuracy we used weighted ensemble techniques which gave a test accuracy of 95.6%. Further, To compare with state of the art CNN architectures, we implemented training set on ResNet 50, and the test accuracy is of 99%.

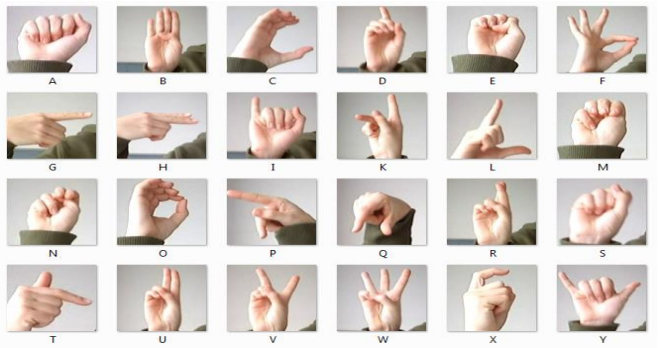
I. INTRODUCTION

Communication is essential in building a nation. Good communication leads to better understanding, and it encompasses all the members of the community, including the deaf. Sign language bridges the gap of communication with other people. However, most hearing people do not understand sign language and learning it is not an easy process. As a result, there is still an undeniable barrier between the hearing impaired and hearing majority.

Deaf and Mute people use hand gesture sign language to communicate, hence normal people face problems in recognizing their language by signs made. Hence there is a need for systems that recognize the different signs and conveys the information to normal people.

We obtained the dataset from Kaggle https://www.kaggle.com/datasets/datamunge/sign-language-mnist. Each training and test case represents a label (0-25) as a one-to-one map for each alphabetic letter A-Z (and no cases for 9=J or 25=Z because of gesture motions).

Train, test data sets are of shape (27455, 785) & (7172,785), out of which 784 are pixel intensity values & the first column is its label (28*28 pixel size image + Label).

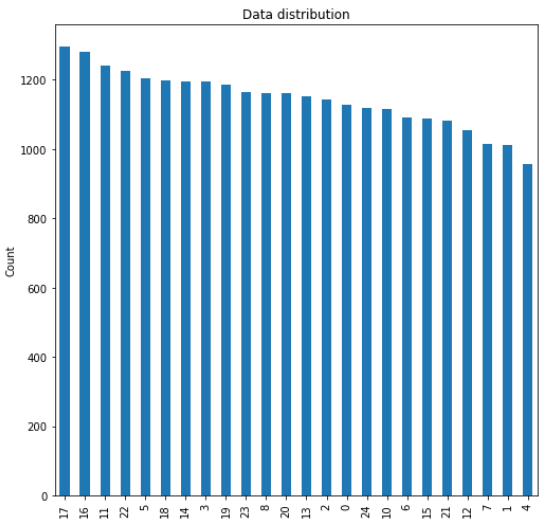


From https://www.kaggle.com/datasets/datamunge/sign-language-mnist

II. DATA ANALYSIS and MODELING

A. Data Analysis

Histogram below shows the class wise distribution of data. We can observe that the dataset is fairly balanced across all the classes.



Python Packages used

- Pandas
- Numpy
- Matplotlib
- Keras models and layers API's

Pre-processing steps:

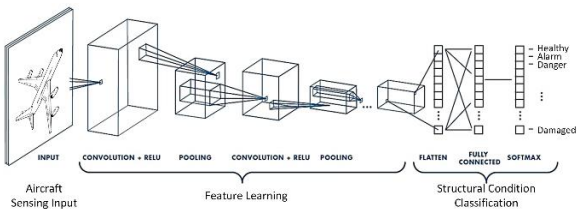
1. Normalization: To Noramlise the pixel intensity values, each pixel intensity value was divided by 255 (which is the maximum possible intensity). After normalization, all pixel intensity values have a range between 0 &1.

$$X_N = (X - X_{min})/(X_{max} - X_{min})$$

2. One hot encoding: All the categorical class variables in the form of alphabets were converted into a form that could be provided to ML algorithms to do a better job in prediction. One hot encoding converts the classes in the form of a one hot vector which contains all zero's except for the class index values which is given a value 1.

3. Mini-Batch: Since, the data set is very huge, it is divides into 215 batches with each batch having 128 training images. Total of 10 epochs were used to update weight parameters.

B. Model



We used CNN models for sign language classification. CNN models were first choice when it comes to image recognition because of various advantages such as sparse connections, parameter sharing. Etc.

Sparse connections: Single element in feature map connected to small patch of elements in the image, whereas each unit is connected to every other unit in the case feedforward networks.

Parameter sharing: Same weights were used for different patches of input images, whereas each connection has a different weight incase of feedforward networks. This allows to perform classification with less parameters.

Model 1:

Below is the summary of model 1. It has a 3 convolution layers followed by a max-pooling layer and dropout layer. The output from these layers is fed to 2 fully connected dense layers.

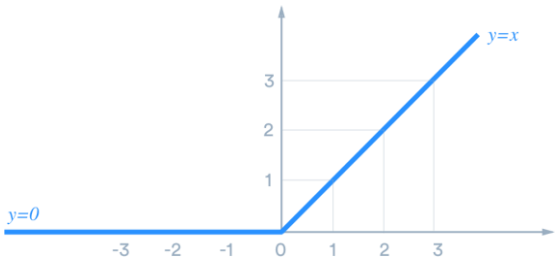
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 128)         73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 1, 1, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 1, 1, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 128)               16512
_________________________________________________________________
dense_1 (Dense)              (None, 25)                3225
=================================================================
Total params: 112,409
Trainable params: 112,409
Non-trainable params: 0
```

Train acc = 97.5%, val_acc = 93.5%

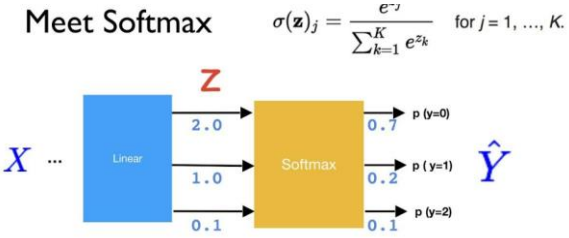
Activation function: Activation function mimics the firing of neurons in the brain, We used ReLu as the activation function after convolution operation for each convolution layer. ReLu is chosen due to it simplicity

compared to other choices such as tanh, sigmoid which has exponential terms.

ReLu



SoftMax Classifier: Output of final layer is fed to softmax activation function to obtain the probabilities of hand sign belonging to a certain class. Image shown below provides the formula of softmax and illustrate it with an example.

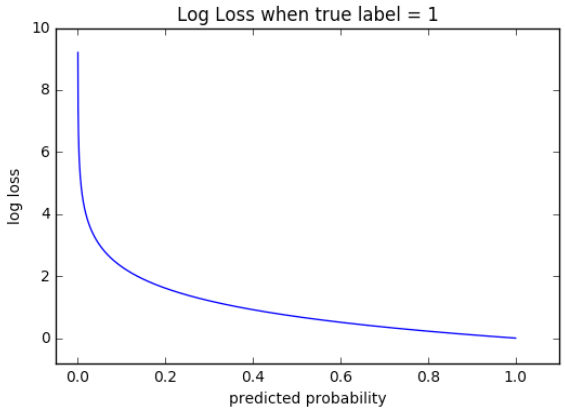


Loss function: Cross-Entropy loss functions is used to optimize the model during training. Cross-Entropy takes the output probabilities (P) and measure the distance from the truth values.

It is given by the formula shown below. T represents the target value, and S represents the predicted value.

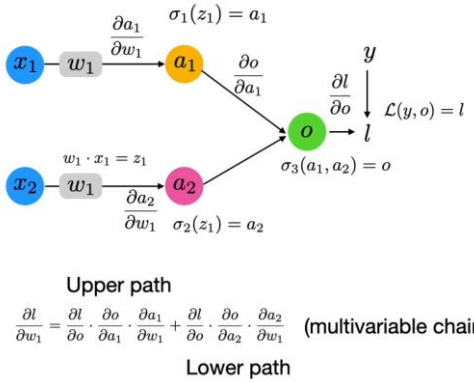
$$L_{CE} = -\sum_{i=1} T_i \log(S_i)$$

Image below shows the plot for a typical cross-entropy loss function. We can clearly observe the loss to be very high in case of prediction probability is less.



Back Propagation for CNN:

Backpropagation was shown below for a simple model. It is similar to feedforward network with minor changes such as same weights used for both inputs & sparse networks between layers.



Adam Optimizer:

Adam optimizer involves a combination of two gradient descent methodologies:

a. Momentum b. Root Mean Square Propagation (RMSP)

Momentum:
This algorithm is used to accelerate the gradient descent algorithm by taking into consideration the 'exponentially weighted average' of the gradients. Using averages makes the algorithm converge towards the minima in a faster pace.

momentum:-

$$V_{dw} = \beta_1 V_{dw_{prev}} + (1-\beta_1)\, dw$$

$$V_{dB} = \beta_1 V_{dB_{prev}} + (1-\beta_1)\, dB$$

- dw = derivative of Loss wrst weight W

- dB = derivative of Loss wrst to Bias B

-V_{dw} = aggregate of gradients (initialise V as '0' and update iteratively)

β1 = Moving average parameter (β1 = 0.9)

Root Mean Square Propagation (RMSP):

The Root Mean Square Propagation RMS is a technique to dampen out the motion in the y-axis and speed up gradient descent**.** In the loss graph et us denote the Y-axis as the bias b and the X-axis as the weight W. The idea is to slow down the learning on the y-axis direction and speed up the learning on the x-axis direction.

RMSE propagation

$$S_{dw} = \beta_2 \cdot S_{dw_{prev}} + (1-\beta_2)(dw)^2$$

$$S_{dB} = \beta_2 \cdot S_{dB_{prev}} + (1-\beta_2)(dB)^2$$

- S_{dw} = sum of squares of past gradients

- β2 = Moving average parameters (β2 = 0.999)

Adam Optimizer uses the squared gradients to scale the learning rate like RMSprop and it takes advantage of momentum by using moving average of the gradient instead of gradient itself like SGD with momentum**.**

Adam:-

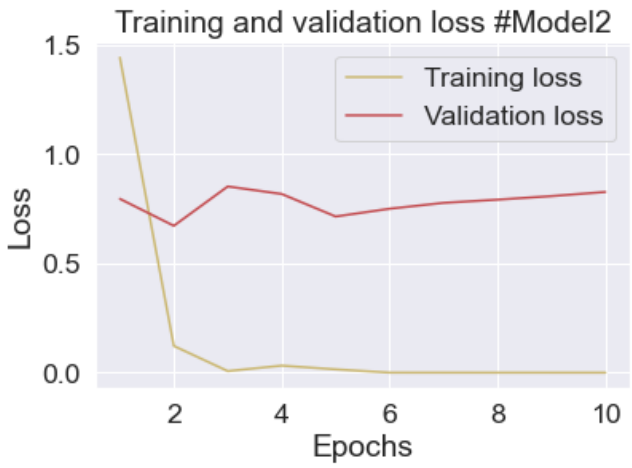
$$w = w - \alpha \cdot \frac{V_{dw}}{\sqrt{S_{dw}+\varepsilon}} \qquad B = B - \alpha \frac{V_{dB}}{\sqrt{S_{dB}+\varepsilon}}$$

Model-2:

We made this model little overfit by removing dropout layers and adding more convolution layers. We have three 32 convoultion filters followed by three 64 filters and two 128 layer filters with maxpooling layer in middle. Because of this, out model will give high accuracy on training data, but little less accuracy on validation accuracy as you can observe in the graph.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              
=================================================================
conv2d_3 (Conv2D)            (None, 26, 26, 32)        
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 24, 24, 32)        
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 12, 32)        
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 10, 10, 64)        
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 8, 8, 64)          
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 6, 6, 64)          
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 3, 3, 64)          
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 1, 1, 128)         
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 1, 1, 25)          
_________________________________________________________________
flatten_1 (Flatten)          (None, 25)                
_________________________________________________________________
dense_2 (Dense)              (None, 25)                
=================================================================
Total params: 179,651
Trainable params: 179,651
Non-trainable params: 0
```

Train acc = 100% val_acc = 89%

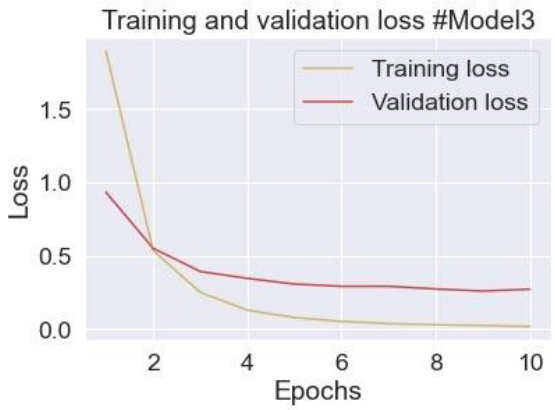


Model-3:

Model3 is a simple model with two convolution layers with maxpooling and dropout in middle followed by a fully connected dense layer.

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              
=================================================================
conv2d_10 (Conv2D)           (None, 26, 26, 32)        
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 13, 13, 32)        
_________________________________________________________________
dropout_3 (Dropout)          (None, 13, 13, 32)        
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 11, 11, 64)        
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 5, 5, 64)          
_________________________________________________________________
dropout_4 (Dropout)          (None, 5, 5, 64)          
_________________________________________________________________
flatten_2 (Flatten)          (None, 1600)              
_________________________________________________________________
dense_3 (Dense)              (None, 25)                
=================================================================
Total params: 58,841
Trainable params: 58,841
Non-trainable params: 0
```

Train acc= 99.6% val_acc = 92.8%



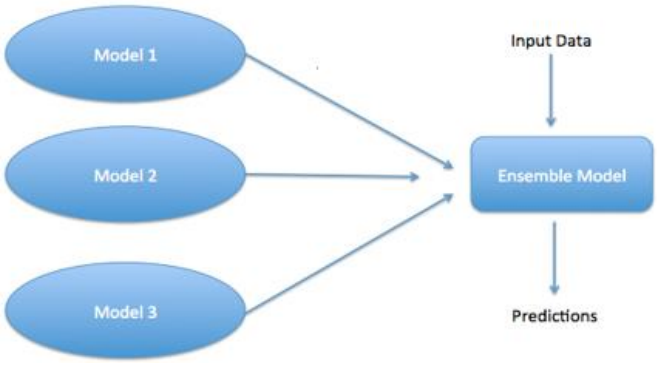
Why Ensemble Techniques:

Robustness:
- It also limits the spread or dispersion of model forecasts and performance.

Performance:
-Ensemble can make better forecasts and deliver better results than any single component model.

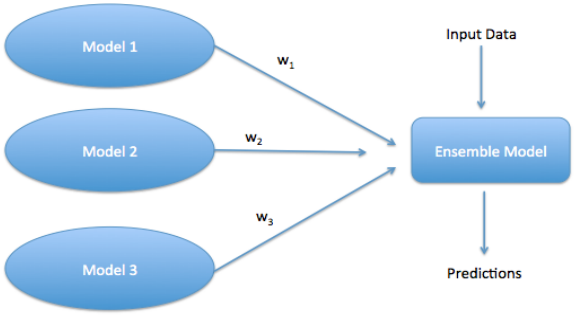
An ensemble can create lower variance and lower bias

Average Ensemble:



E1 = argmax(sum across classes)

Weighted Ensemble:



E2 = argmax(weighted sum across classes)

Hyper-parameter tuning:

Grid search:

```
for w1 in range(0, 5):
    for w2 in range(0,5):
        for w3 in range(0,5):
            wts = [w1/10.,w2/10.,w3/10.]
            wted_preds1 = np.tensordot(preds1, wts, axes=((0),(0)))
            wted_ensemble_pred = np.argmax(wted_preds1, axis=1)
            weighted_accuracy = accuracy_score(y_test, wted_ensemble_pred)
            df = df.append(pd.DataFrame({'wt1':wts[0],'wt2':wts[1],
                                         'wt3':wts[2], 'acc':weighted_accuracy*100}, index=[0]), ignore_index=True)
```

The basic idea of grid search is, it runs through all the different parameters that is fed into the parameter grid and produces the best combination of parameters, based on a scoring metric of our choice. Here inorder to achieve best accuracy with respect to weights we

applied grid search concept on all 3 weights with values in range 0 to 0.5. While for the CNN based models we took default approach like starting with 32 layers and then increasing it in powers of 2. Applied popular metrics for activation ('Relu') and optimizer ('adam').

Loss Metrics:

Categorical crossentropy(Log loss) is a loss function that is used in multi-class classification tasks. During model training, the model weights are iteratively adjusted accordingly with the aim of minimizing the Cross-Entropy loss. Entropy of a random variable X is the level of uncertainty inherent in the variables possible outcome. The greater the value of entropy,H(x) , the greater the uncertainty for probability distribution and the smaller the value the less the uncertainty.

$$L_{\mathrm{CE}} = -\sum_{i=1}^{n} t_i \log(p_i),$$

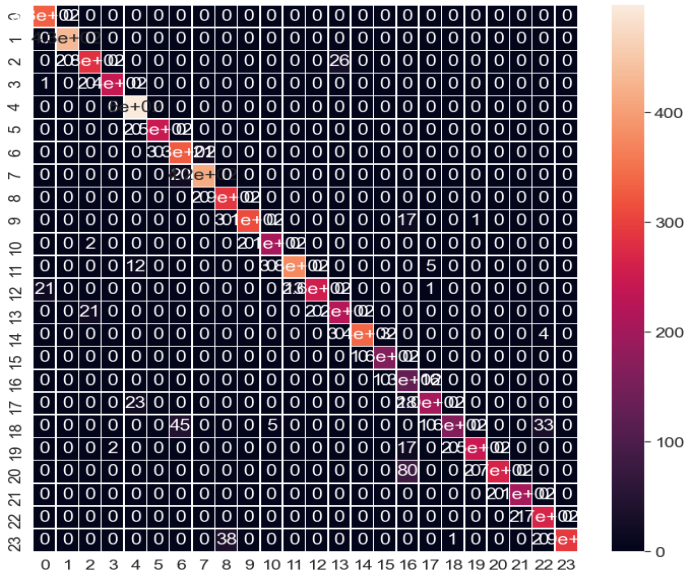
t_i = truth or actual label for ith class

p_i = predicted probability for ith class

Confusion Matrix:

Classification is the process of categorizing a given set of data into classes. Each row in a confusion matrix represents an actual class, while each column represents a predicted class or vice versa.

Accuracy = (ratio of correctly predicted examples) / (the total examples)



Comparison of accuracies:

Model1,2,3 corresponds to our CNN based deep learning models, average ensemble corresponds to ensemble with equal weights, whereas weighted average ensemble takes the best possible weights obtained from grid search approach.

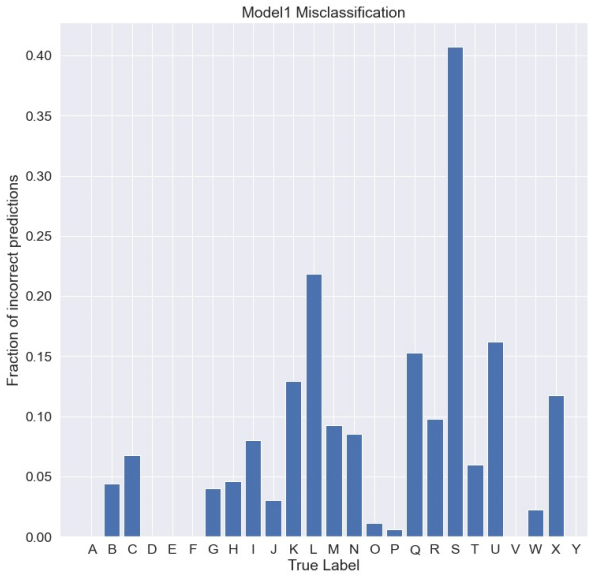
```
Accuracy Score for model1 =  0.916201896263246
Accuracy Score for model2 =  0.8961238148354713
Accuracy Score for model3 =  0.9382320133853876
Accuracy Score for average ensemble =  0.9525934188510876
Accuracy Score for weighted average ensemble =  0.9500836586726157
Max accuracy of  0.2  obained with w1= 0.3  w2= 0.4  and w3=
95.6079196876743
```

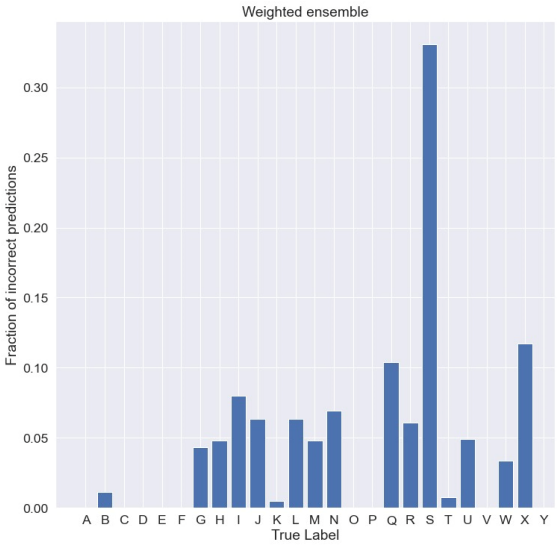
As we can see ensemble approach works better when compared to individual models because it lets the individual models to 'vote'. As the individual models won't repeat the same mistake we can reduce the error to good extent. For example if there are 25 classifiers and each classifier has an error rate of 0.35, then the combined error can be reduced to 0.006.

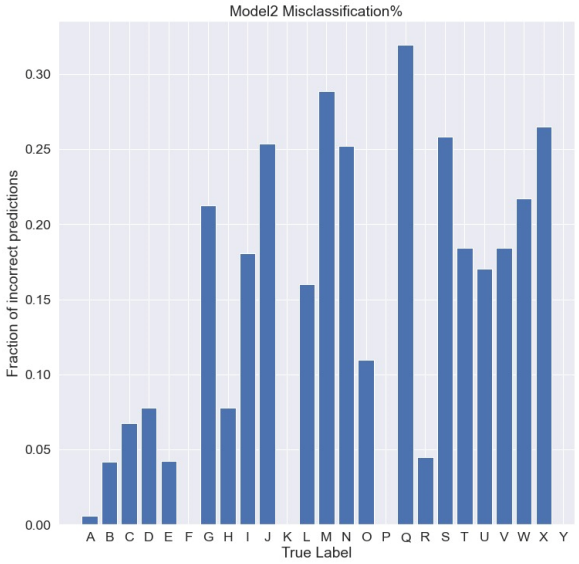
$$\sum_{i=1}^{25} \binom{25}{i} \varepsilon^{\mathrm{i}} (1-\varepsilon)^{25-i} = 0.06$$

We anticipated that the ensemble model with Model1,2,3 as base learners gives better accuracy and it reached our expectation.

Misclassification analysis:

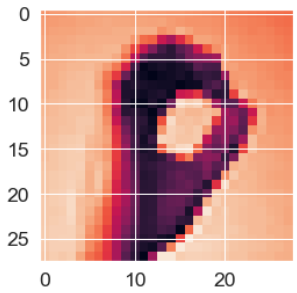


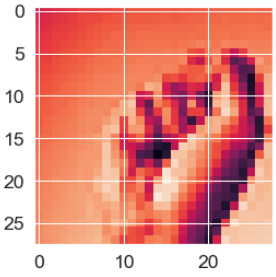


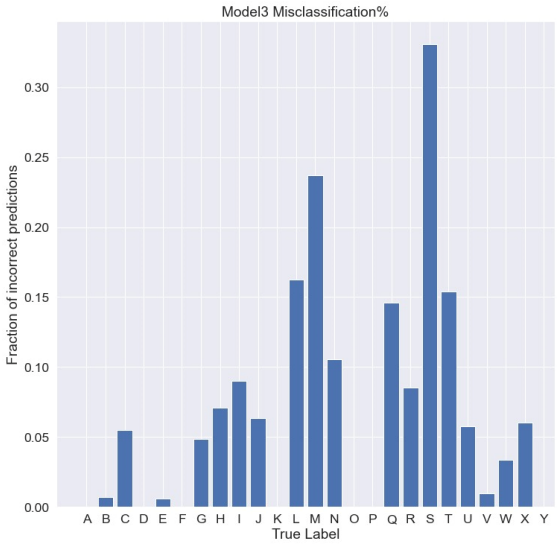


Weighted ensemble was able to reduce misclassifications for classes like A,B,C,D,E,K,M,O,T,V e.t.c

Results:








```
...: print("True Label
Predicted Label:  0
True Label:  0
```

```
...: print("True Labe
Predicted Label:  S
True Label:  S
```

Conclusion:

In this project we developed multiple models to classify mnist data. We have used three CNN based deep learning models for initial classification and achieved test accuracies of 91.6%, 89.6% & 93.8%. We then implemented average and weighted ensemble models with our CNN based models as base

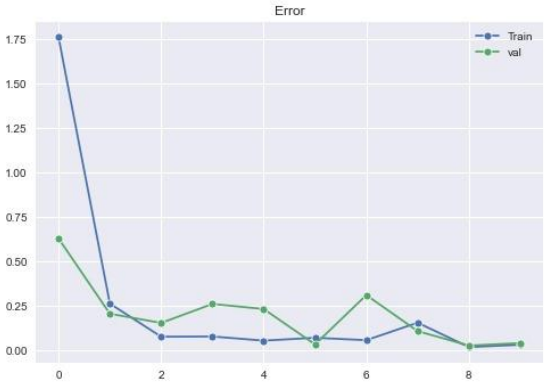
learners. Using grid search to decide on optimal weights, we acheived 95.6% accuracy.

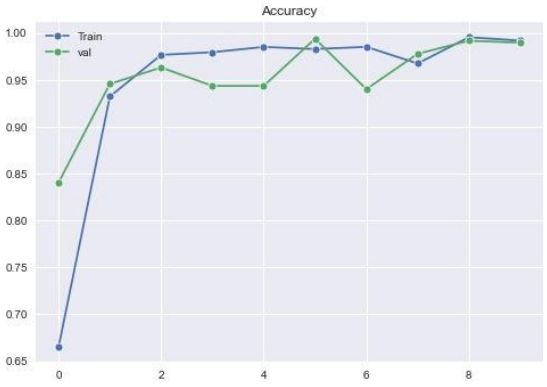
Future work (RESNET)

ResNet makes it possible to train up to hundreds or even thousands of layers and still achieves compelling performance. We want to understand the architecture& how the deep neural network handles the problem of vanishing gradients.

layer name	output size	18-layer	34-layer	50-layer
conv1	112×112	7×7, 64, stride 2		
conv2_x	56×56	3×3 max pool, stric		
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc,		

However, To compare the accuracy of our ensemble model with RESNET 50, we implemented the dataset on it & have achieved training accuracy of 99.17% , and validation accuracy of 98.95%.





References:

[1] https://www.kaggle.com/datasets/datamunge/sign-language-mnist

[2] https://www.mdpi.com/1424-8220/19/22/4933

[3] https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

[4] https://towardsdatascience.com/convolution-neural-networks-a-beginners-guide-implementing-a-mnist-hand-written-digit-8aa60330d022

[5] https://www.researchgate.net/figure/Overview-of-CNN-hyperparameters_tbl2_320723116

[6] https://pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/

[7] https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e

[8] https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33

[9] https://iq.opengenus.org/resnet50-architecture/

[10] https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00444-8

CODE:

https://github.com/Rajesh-26/DS5220