American Sign Language Alphabet Recognition / Classification

Ajay Karthick Senthil Kumar
senthilkumar.a@northeastern.edu

Firdose Shaik
shaik.fi@northeastern.edu

Master of Science in Data Science, Khoury College of Computer Sciences, Northeastern University, Boston, Massachusetts

classify-asl-alphabets

Abstract

American Sign Language (ASL) provides an effective means of communication among people who are hard of hearing, but it is less known outside, and relatively only a small group of people (about 250000 to 500000 people) in the United States understand it. Therefore, there exists a communication gap between the deaf and hearing community because many English speakers cannot understand the signs used in ASL. With the exponential advancement in technology and massive growth of deep learning, we can bridge the gap between hearing-impaired people who uses ASL and people who cannot understand ASL. We developed a deep learning model with Convolutional Neural Network (CNN) architecture to recognize (classify) the ASL alphabets which pave the way to make ASL more accessible and interpretable. This project involves developing a CNN model that perfectly classifies the ASL hand gestures in real-time with high accuracy after trying out different design architectures. And then, we developed an application that captures the real-time ASL alphabet hand gestures shown by the end user using the web camera, feed it to the CNN model, and the model will classify and gives the corresponding English Alphabet.

Introduction

American Sign Language (ASL) is a commonly used visual language in the United States by people who are hard of hearing. It uses hand gestures to convey thoughts instead of spoken words. It has different ways of signaling to express one's thoughts. It has 26 signs that can be shown by hand gestures to represent the alphabet of the English language. Being an expressive language using signs, it is crucial for those who are hard of hearing to convey their thoughts, feelings, and ideas to other people. However, the number of people who can interpret this language is comparatively very limited. Due to the limitation of the population of ASL users, the hearing-impaired person can communicate with only a limited number of people, using ASL since only a small set of people understand it. And it causes pidginization and leads to the communication gap between the deaf and hearing community. This real-time problem motivates us to look for a solution that assists in bridging the gap between these two populations. Implementing a mechanism that can aid in developing a platform to make ASL more easily interpretable outside the ASL users' community would benefit the hearing-impaired people as well as the hearing community.

The task of developing a solution to this problem is interesting as it resolves a real-time issue and benefits many people. It is very essential to address this problem as much as it is interesting. The use case of this problem that exists in the real world is when the hearing-impaired person communicates with a person through ASL who doesn't understand it.

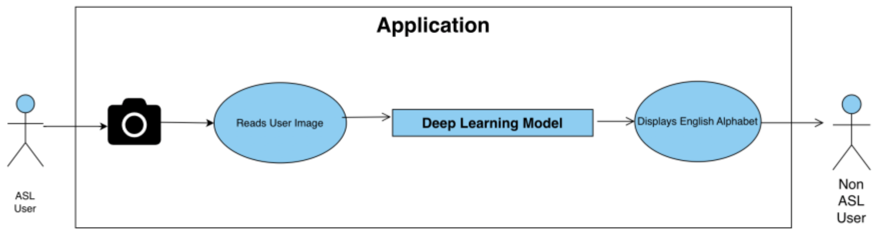


Figure 1 (Use case Diagram of the application)

To tackle this problem and provide a solution to this problem, we are proposing to develop a Convolutional Neural Network model, a type of deep learning model, which can classify the ASL hand gestures shown by the user. We are choosing Convolutional Neural Network (CNN) as this is a computer vision task and CNN has been outstanding at computer vision tasks. In the CNN model, we build a sequence of layers that gradually learns the patterns from the image by mapping the image to a feature representation. We are not using a feedforward neural network as it does not recognize the spatial structure of the image. CNN model architecture that we chose to develop can recognize the spatial structure of the image by learning the local patterns. Learning the local patterns of the image makes it capable of recognizing the pattern no matter where it is located in the image. Also due to parameter sharing of the convolutional layers, the CNN model captures the invariant properties of the input image.

While most of the competing approach uses Transfer Learning where it uses pre-training models to achieve this task, we intend to build the entire CNN model from scratch and train it with the sufficiently large training data.

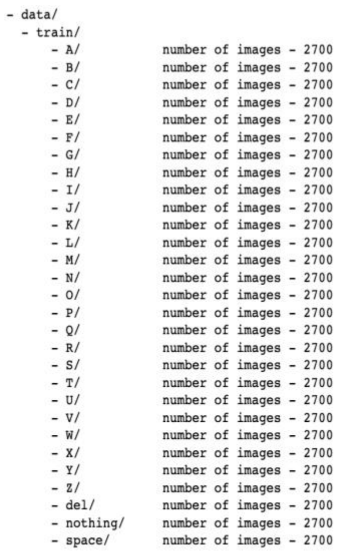
The key components of our approach are developing CNN models with different design architectures, training them with the huge images data set, evaluating each model with various metrics and find the CNN model architecture that best classifies the input ASL hand gestures. And then develop an application that captures the real-time ASL hand gestures and classifies it with our best CNN model.

Preliminaries

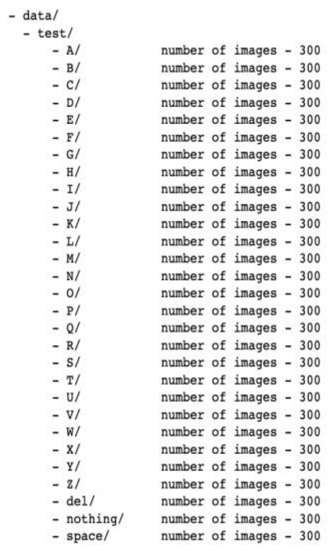
<u>Dataset</u>

As it is a computer vision task, our dataset is images. The total number of images is 87000. The number of labels in our task is 29. This includes the 26 English alphabet (A-Z), space, delete and nothing. The number of images we used to train the model is 78300 and the number of training samples per label is 2700 and the number of images we used to evaluate the model is 8700 and the size of test dataset per label is 300.

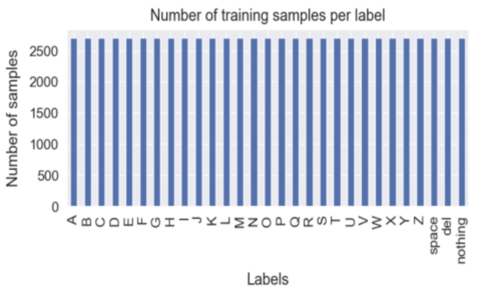
All the images are of the format JPEG. All the images are of the size (the horizontal and vertical size in pixels) is (200, 200). And all the images are in RGB mode.

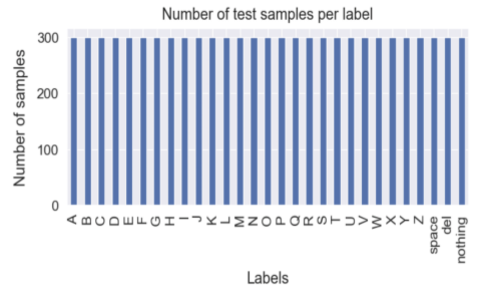


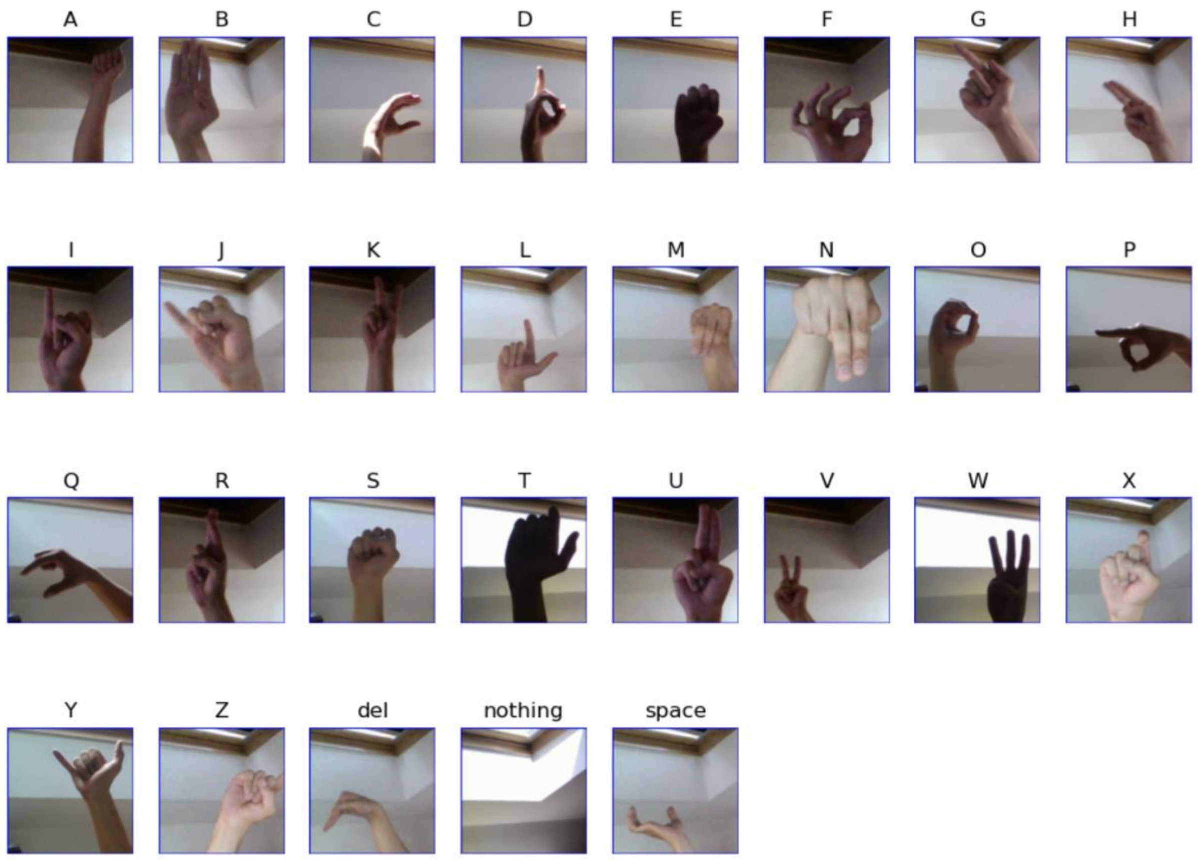
```
- data/
  - train/
      - A/          number of images - 2700
      - B/          number of images - 2700
      - C/          number of images - 2700
      - D/          number of images - 2700
      - E/          number of images - 2700
      - F/          number of images - 2700
      - G/          number of images - 2700
      - H/          number of images - 2700
      - I/          number of images - 2700
      - J/          number of images - 2700
      - K/          number of images - 2700
      - L/          number of images - 2700
      - M/          number of images - 2700
      - N/          number of images - 2700
      - O/          number of images - 2700
      - P/          number of images - 2700
      - Q/          number of images - 2700
      - R/          number of images - 2700
      - S/          number of images - 2700
      - T/          number of images - 2700
      - U/          number of images - 2700
      - V/          number of images - 2700
      - W/          number of images - 2700
      - X/          number of images - 2700
      - Y/          number of images - 2700
      - Z/          number of images - 2700
      - del/        number of images - 2700
      - nothing/    number of images - 2700
      - space/      number of images - 2700
```



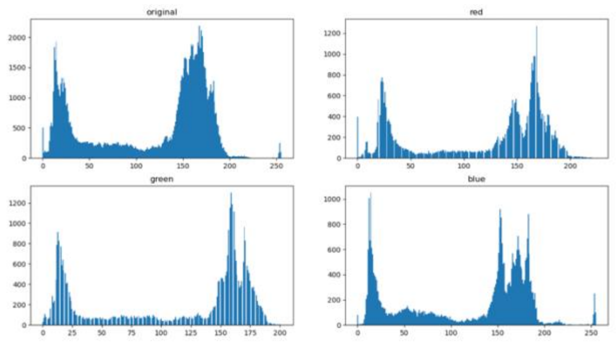
```
- data/
  - test/
      - A/          number of images - 300
      - B/          number of images - 300
      - C/          number of images - 300
      - D/          number of images - 300
      - E/          number of images - 300
      - F/          number of images - 300
      - G/          number of images - 300
      - H/          number of images - 300
      - I/          number of images - 300
      - J/          number of images - 300
      - K/          number of images - 300
      - L/          number of images - 300
      - M/          number of images - 300
      - N/          number of images - 300
      - O/          number of images - 300
      - P/          number of images - 300
      - Q/          number of images - 300
      - R/          number of images - 300
      - S/          number of images - 300
      - T/          number of images - 300
      - U/          number of images - 300
      - V/          number of images - 300
      - W/          number of images - 300
      - X/          number of images - 300
      - Y/          number of images - 300
      - Z/          number of images - 300
      - del/        number of images - 300
      - nothing/    number of images - 300
      - space/      number of images - 300
```





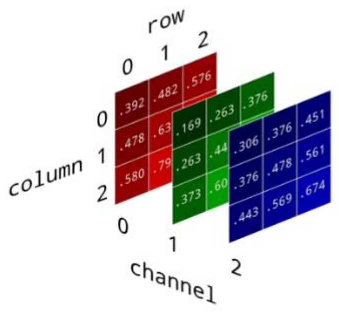


Exploratory Data Analysis



As all the images are RGB images, there are three channels for each color and each color channel contains the intensities of each pixel of all pixels of the size (width x height). Each pixel intensity ranges between 0 and 255. The 256 different intensities of the original, red color channel, green color channel and blue color channel is analyzed for randomly selected images, and it appears that the intensities are uniformly distributed across three channels for the images.

Data Preprocessing

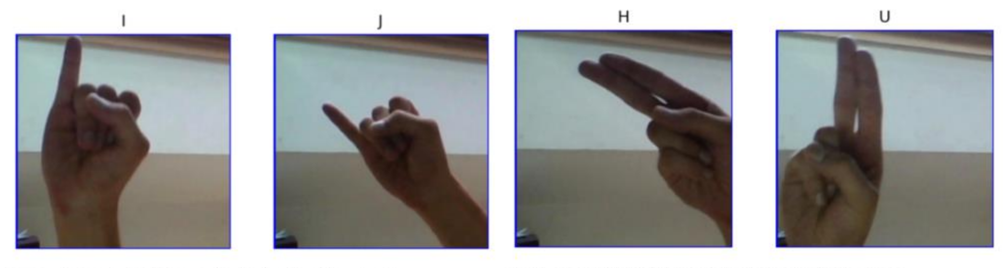


As we deal with the unstructured data in our model, it is important to preprocess the images data. The Data Preprocessing steps include decoding each image to 3 Dimensional floating-point tensors containing pixel values, resizing all images to standard size (width and height) while converting them to tensors, rescaling the pixel values from [0 to 255] range to [0 to 1] range and stack each image tensors in a sequence vertically so that the resultant tensor is of the following shape when fed to the model.

[batch_size, image_height, image_width, image_channels]

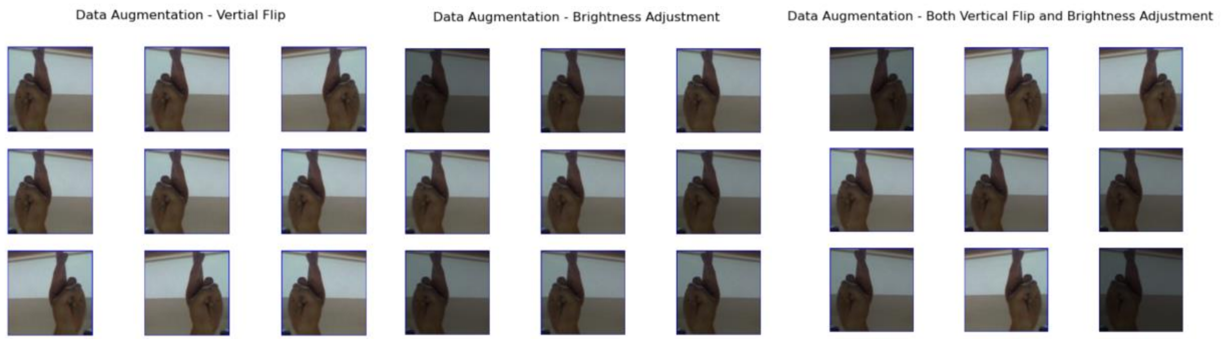
Data Augmentation

Data Augmentation is an important step in deep learning that helps to mitigate the problem of overfitting and data scarcity. Data Augmentation usually includes rotation, shearing transformation, flipping and brightness alteration. Transformation like rotation is not possible on this data because most of the alphabet signs differ by rotation and inclination angle in the hand gesture.



Signs of I and J differ only by inclination angle Signs of H and U differ only by inclination angle

While we analyze all image files, we noted that all the images that we extracted from the internet source are based on right-hand gestures, but in the real world, it is possible that the ASL user who uses our application could be left-handed. Hence it is needed to generate the left-hand gestures images data and it can be done through data augmentation by vertical flip. In addition to this, brightness adjustment is also added as a part of data augmentation.


Data Augmentation - Vertial Flip
Data Augmentation - Brightness Adjustment
Data Augmentation - Both Vertical Flip and Brightness Adjustment

Building Models

After preprocessing the data, we started building Convolutional Neural Networks by trying out different design architectures. Our focus is to find the best-performing CNN model that classifies the ASL alphabet with the best accuracy.

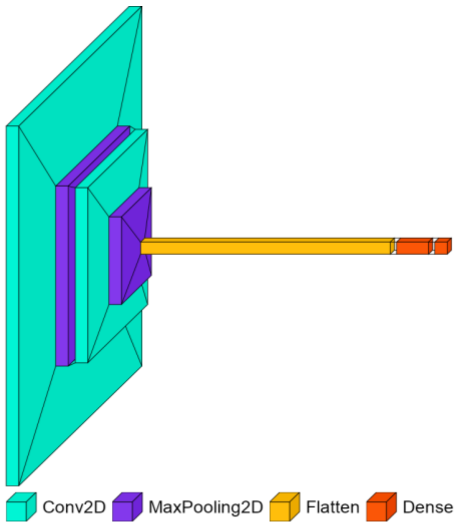
Model 1

We started with a CNN that contains 2 Convolution layers each followed by a max pooling layer and 2 Dense Layers. We used TensorFlow Keras to set up the environment, build, train and evaluate the models.

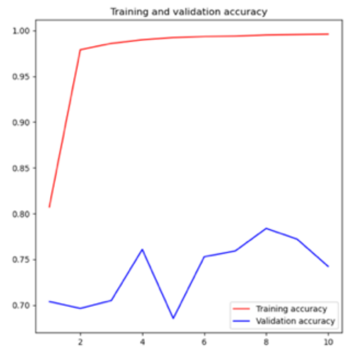
The convolution Layer is the fundamental component of the CNN, and it enables CNN to learn the local spatial patterns by connecting each neuron only to a small local region. This layer learns the local features according to the hyperparameters such as the depth and width of the filters, stride, and zero-padding values. The output of the single filter in the layer is called a feature map that is learned from the input image. The consecutive convolutional layers will learn bigger local patterns made of the features of the previous layers.

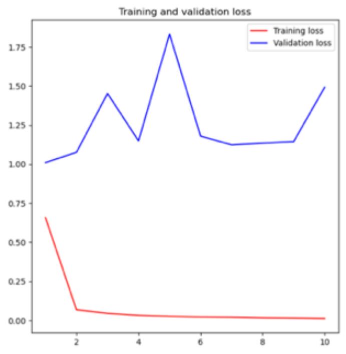
MaxPooling Layer is usually applied immediately after the convolutional layer. It takes each feature map that is generated by the previous convolutional layer and downsamples it. This layer uses extracting windows that are applied to the input feature maps, extracts the maximum values in each window, and outputs them. It is applied to all the feature maps and downsampled them.

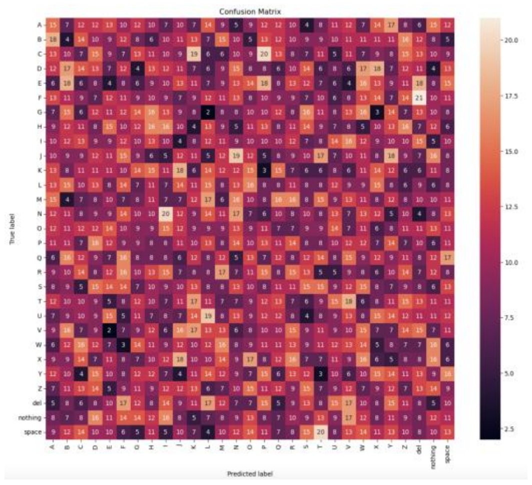
The CNN model made of 2 Convolutional layers followed by MaxPooling operation and 2 Dense Layers was built and trained with Adam Optimizer and Loss Function as Categorical Cross Entropy Loss. After the training phase, we looked at the metrics to see if the model learned well. The training accuracy improves with the number of epochs, and it reached 99% but the validation accuracy does not get better, and it flattens out at some point. This is a result of the overfitting problem. And then this model is evaluated on the test data and the test accuracy of this model is 96.45%.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 32)      896

 max_pooling2d (MaxPooling2D  (None, 74, 74, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 72, 72, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 36, 36, 64)       0
 2D)

 flatten (Flatten)           (None, 82944)             0

 dense (Dense)               (None, 512)               42467840

 dense_1 (Dense)             (None, 29)                14877

=================================================================
Total params: 42,502,109
Trainable params: 42,502,109
Non-trainable params: 0
_________________________________________________________________
```







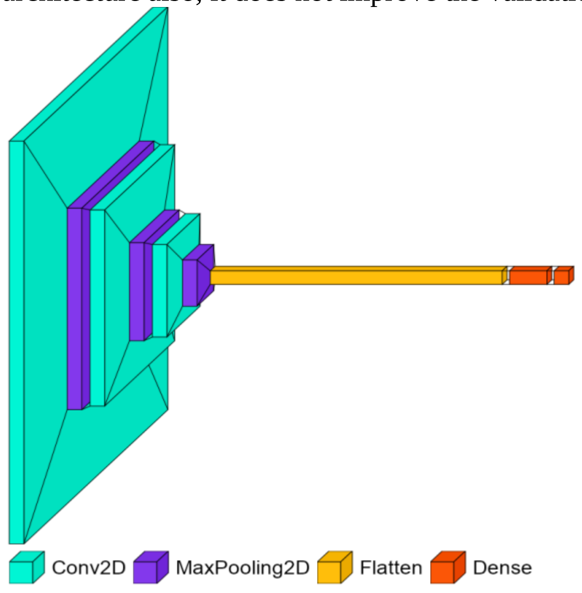
```
136/136 [==============================] - 8s 57ms/step - loss: 0.1939 - accuracy: 0.9645
Test accuracy is 96.44827842712402
Test loss is 0.19385448098182678
```

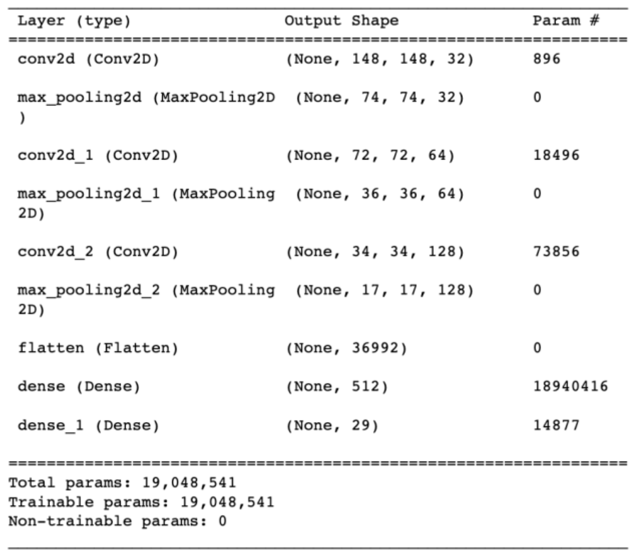
Model 1

The overfitting problem occurs when the trained model does not generalize well enough to classify the data outside the training dataset. Hence, we used a validation dataset to validate the trained model at the end of each epoch to track its performance outside the training set. However, our initial model architecture overfits the specifics of training data and does not perform better in validation data (data that the model has not seen before). We tried to capture the best parameters before the model starts to overfit to see if the current model architecture performs well with the early stopping. But it does not reduce the overfitting problem for this model architecture.

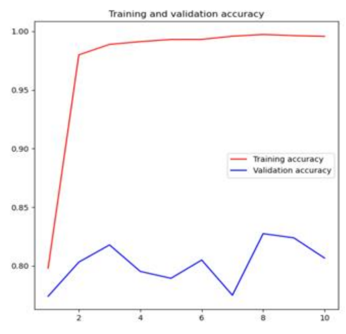
Model 2

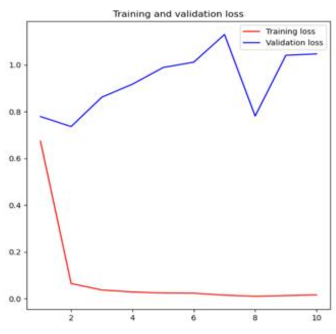
We added one more convolutional layer and a max pooling layer. Even though there is a slight increase in the validation accuracy, the overfitting problem persists. And the test accuracy of this model is 97.79%. We tried early stopping to capture the best parameters for this network as well, but in this model architecture also, it does not improve the validation accuracy a lot.

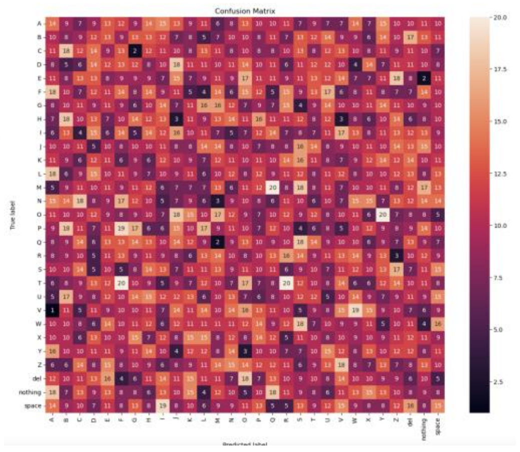


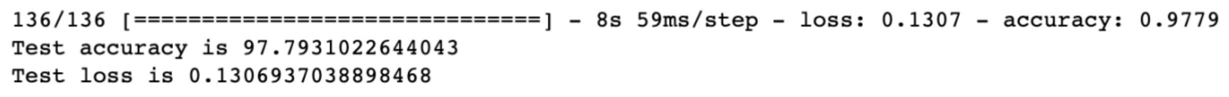


```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      896

max_pooling2d (MaxPooling2D  (None, 74, 74, 32)        0
)

conv2d_1 (Conv2D)            (None, 72, 72, 64)        18496

max_pooling2d_1 (MaxPooling  (None, 36, 36, 64)        0
2D)

conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856

max_pooling2d_2 (MaxPooling  (None, 17, 17, 128)       0
2D)

flatten (Flatten)            (None, 36992)             0

dense (Dense)                (None, 512)               18940416

dense_1 (Dense)              (None, 29)                14877

=================================================================
Total params: 19,048,541
Trainable params: 19,048,541
Non-trainable params: 0
_________________________________________________________________
```









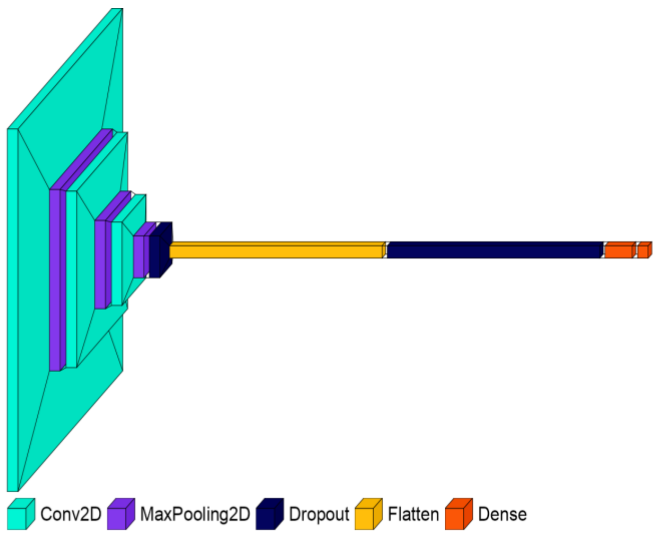
```
136/136 [==============================] - 8s 59ms/step - loss: 0.1307 - accuracy: 0.9779
Test accuracy is 97.7931022644043
Test loss is 0.1306937038898468
```

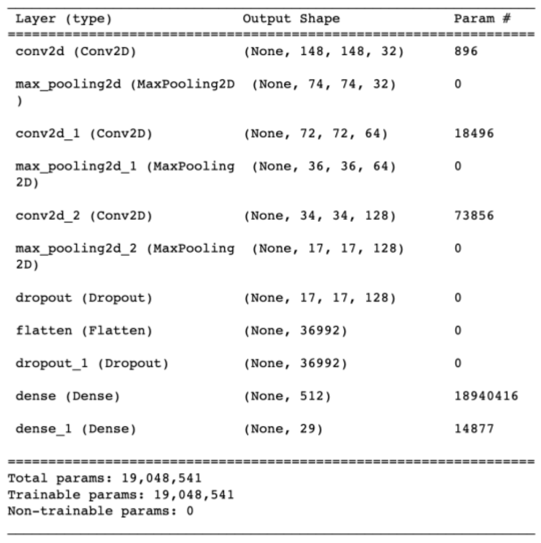
Model 2

Model 3

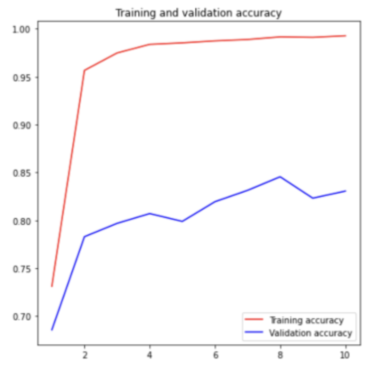
To further reduce the overfitting problem, we started adding the dropout layers to the network. At first, we added two dropout layers to the network and constructed the network (architecture is shown in the Model 3 figure). This model achieved a test accuracy of 97.85%.

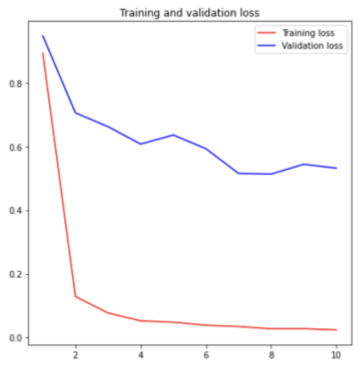
Dropout is a regularization technique used in neural networks. It is different from L1 and L2 regularization. Instead of modifying the cost function, it modifies the network architecture. Each hidden neuron has a probability p to drop out. In each mini-batch, the randomly selected hidden neurons are dropped out, thereby the training data of each mini-batch will be trained with different neural networks. This dropout regularization reduces complex co-adaptations of neurons. Because without dropout regularization, the complex co-adaptations of neurons make the network fails to generalize on the unseen dataset. Randomly dropping a few neurons makes the network generalize well and hence reduces the overfitting problem.

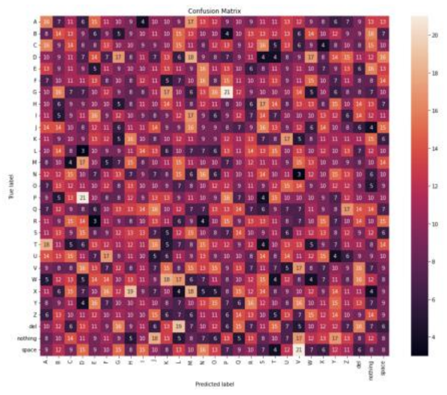


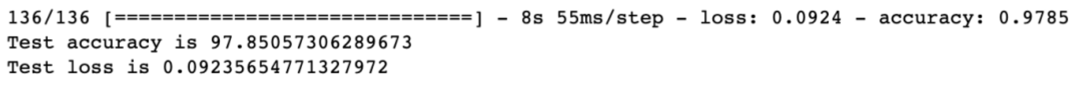


```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 32)      896

 max_pooling2d (MaxPooling2D  (None, 74, 74, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 72, 72, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 36, 36, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 34, 34, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 17, 17, 128)      0
 2D)

 dropout (Dropout)           (None, 17, 17, 128)       0

 flatten (Flatten)           (None, 36992)             0

 dropout_1 (Dropout)         (None, 36992)             0

 dense (Dense)               (None, 512)               18940416

 dense_1 (Dense)             (None, 29)                14877

=================================================================
Total params: 19,048,541
Trainable params: 19,048,541
Non-trainable params: 0
```







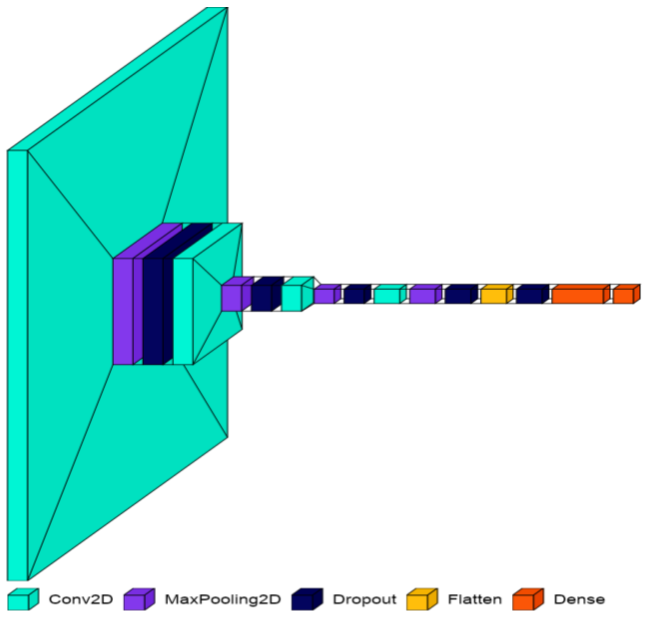


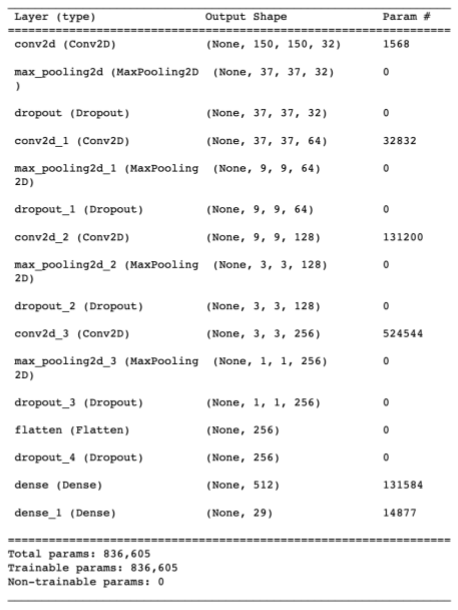
```
136/136 [==============================] - 8s 55ms/step - loss: 0.0924 - accuracy: 0.9785
Test accuracy is 97.85057306289673
Test loss is 0.09235654771327972
```

Model 3

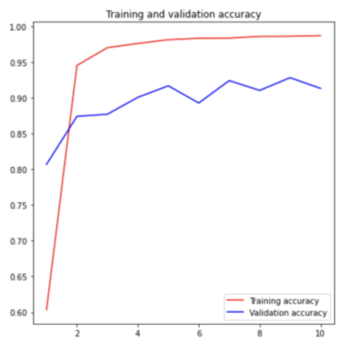
Model 4

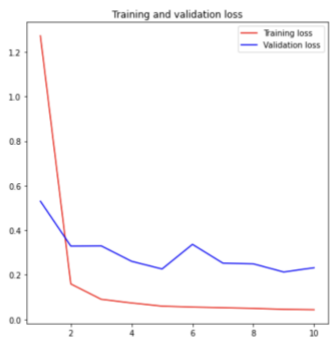
After introducing two dropout layers, the validation accuracy improved, validation loss started decreasing and it is less compared to the previous models without dropout layers but still, there was a notable difference between the training accuracy and validation accuracy. Hence to reduce the overfitting further and enable the network to generalize better, we added a few more dropout layers (model architecture is shown in the model 4 figure). In this model design architecture, we added 4 convolutional layers, and three dropout layers to handle the overfitting problem and enhance the overall model performance.

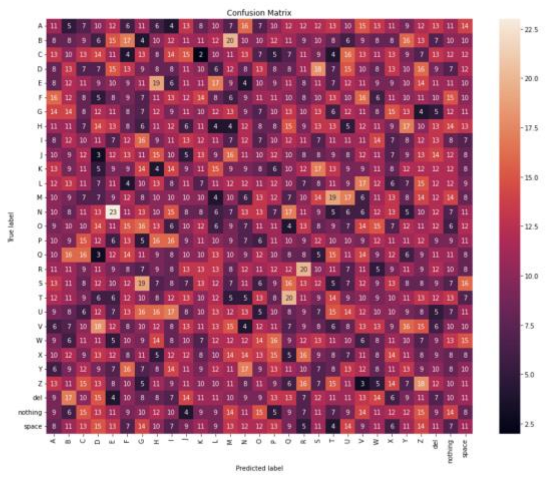


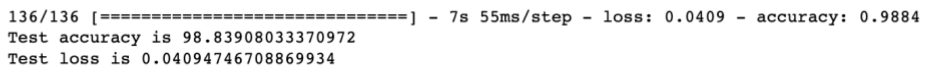


```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 150, 150, 32)      1568

 max_pooling2d (MaxPooling2D  (None, 37, 37, 32)       0
 )

 dropout (Dropout)           (None, 37, 37, 32)        0

 conv2d_1 (Conv2D)           (None, 37, 37, 64)        32832

 max_pooling2d_1 (MaxPooling  (None, 9, 9, 64)         0
 2D)

 dropout_1 (Dropout)         (None, 9, 9, 64)          0

 conv2d_2 (Conv2D)           (None, 9, 9, 128)         131200

 max_pooling2d_2 (MaxPooling  (None, 3, 3, 128)        0
 2D)

 dropout_2 (Dropout)         (None, 3, 3, 128)         0

 conv2d_3 (Conv2D)           (None, 3, 3, 256)         524544

 max_pooling2d_3 (MaxPooling  (None, 1, 1, 256)        0
 2D)

 dropout_3 (Dropout)         (None, 1, 1, 256)         0

 flatten (Flatten)           (None, 256)               0

 dropout_4 (Dropout)         (None, 256)               0

 dense (Dense)               (None, 512)               131584

 dense_1 (Dense)             (None, 29)                14877

=================================================================
Total params: 836,605
Trainable params: 836,605
Non-trainable params: 0
```









```
136/136 [==============================] - 7s 55ms/step - loss: 0.0409 - accuracy: 0.9884
Test accuracy is 98.83908033370972
Test loss is 0.04094746708869934
```

Model 4

<u>Model 5</u>

As we started to increase the number of layers, the network becomes deeper and the deeper layers are trained based on the previous layers' output but when the previous layers update their weights via gradient descent, their outputs are also updated. Hence the inputs to the deep layers don't preserve the normalization. We introduce batch normalization layers to tackle this issue. Batch Normalization is a regularization technique that normalizes (standardizes) the inputs to the deeper layers independent of the changes in their weights. It is useful for training deep nets as it helps to keep the input of all layers normalized and limit the internal covariate shift.

We introduced the Batch Normalization layer to the network along with multiple blocks of Convolutional, MaxPooling layers, and dropout layers, trained the network and the validation accuracy improved a lot. This indicated that the current network architecture learned well enough from the training data to generalize to new data that it has never seen before. This model is evaluated on the test data, and it performed well with **99.23%** accuracy.

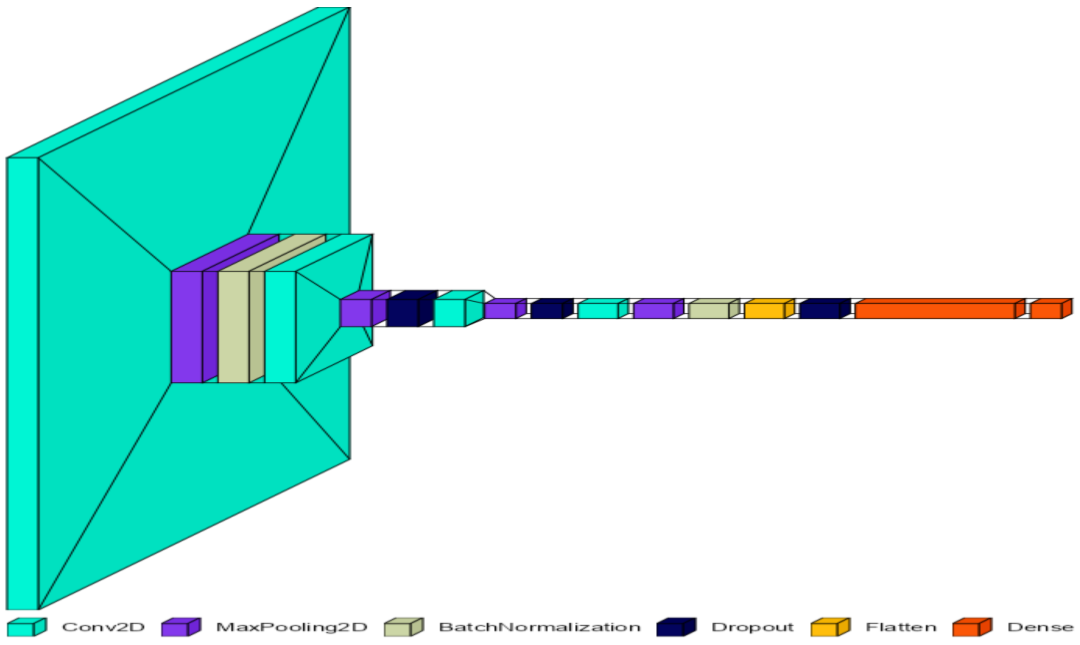
We tried a few other design architectures as well, but it doesn't make a solid improvement in the performance of the model. Hence, we selected this model as our final model. The architecture of this model includes 4 convolutional layers. each convolutional layer followed by a max pooling operation, batch normalization layers, dropout layers, and then two dense layers. We used zero padding to preserve the spatial dimensions. We used Rectified Linear Unit (ReLU) activation for all the convolutional layers and one dense layer. We use the SoftMax activation function for the output layer.

Rectified Linear Unit (ReLU) is the activation function that makes it easier for the network to learn and perform better. This function returns the input as it is if it is greater than zero or simply returns zero. This activation function makes the model converge quickly, and it helps to check if the model is affected by the overfitting problem.

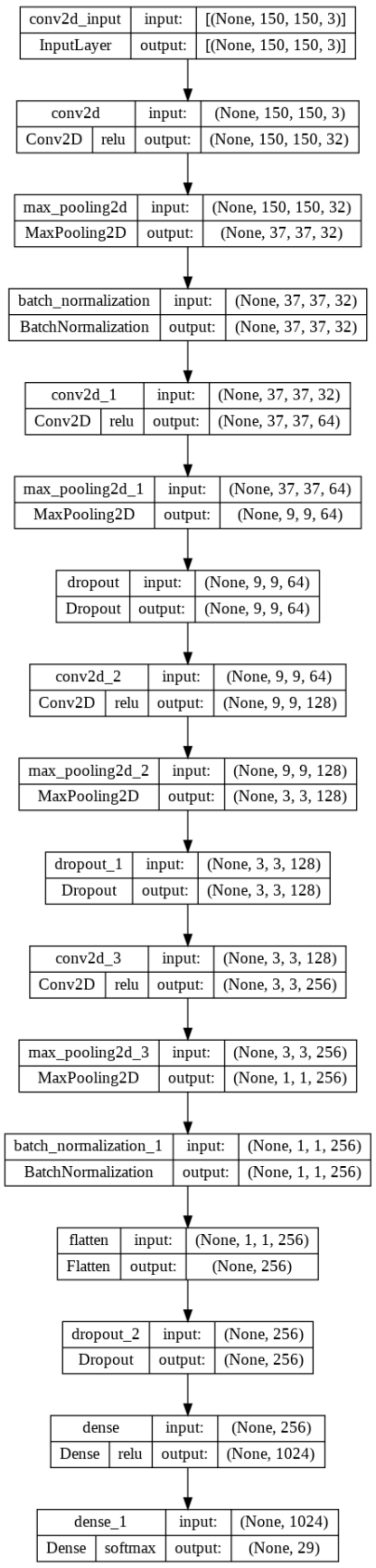
The softMax activation function is applied to the output layer which converts the outputs to represent the values of the probability distribution of certainty that the given input image is classified under each target label. This activation is also helpful in making the model converge quickly and helps to improve the model performance.

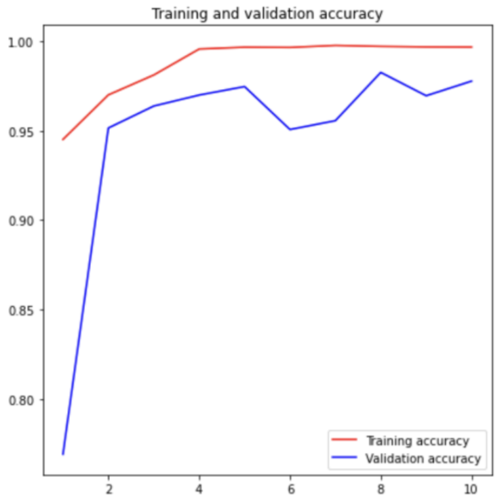
The architecture of the final model that we selected based on the performance is portrayed in the Model 5 figure.

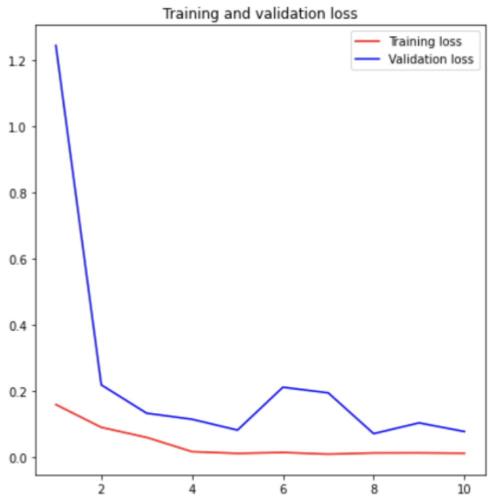
Final Model Architecture

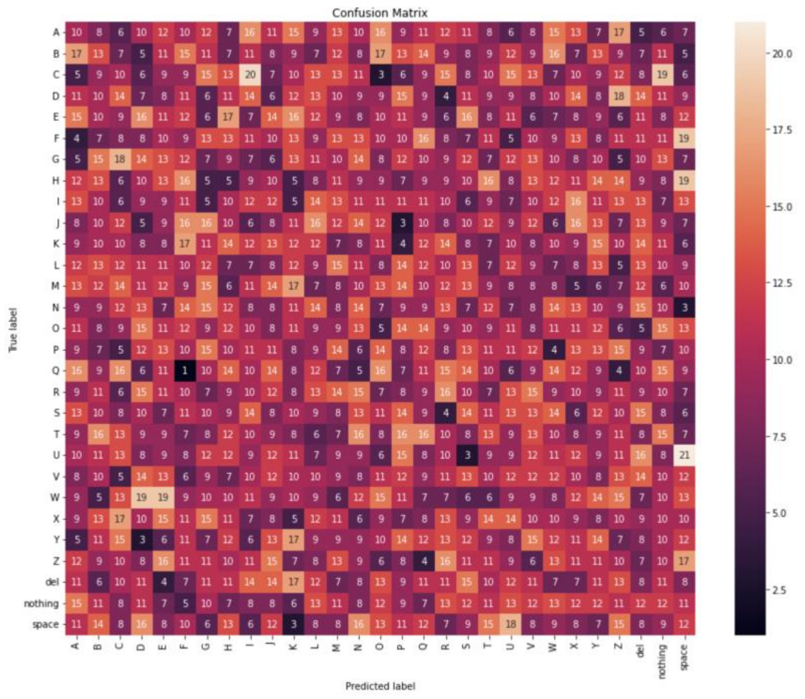


Final Model Layers


conv2d_input | InputLayer | input: [(None, 150, 150, 3)] | output: [(None, 150, 150, 3)]
conv2d | Conv2D | relu | input: (None, 150, 150, 3) | output: (None, 150, 150, 32)
max_pooling2d | MaxPooling2D | input: (None, 150, 150, 32) | output: (None, 37, 37, 32)
batch_normalization | BatchNormalization | input: (None, 37, 37, 32) | output: (None, 37, 37, 32)
conv2d_1 | Conv2D | relu | input: (None, 37, 37, 32) | output: (None, 37, 37, 64)
max_pooling2d_1 | MaxPooling2D | input: (None, 37, 37, 64) | output: (None, 9, 9, 64)
dropout | Dropout | input: (None, 9, 9, 64) | output: (None, 9, 9, 64)
conv2d_2 | Conv2D | relu | input: (None, 9, 9, 64) | output: (None, 9, 9, 128)
max_pooling2d_2 | MaxPooling2D | input: (None, 9, 9, 128) | output: (None, 3, 3, 128)
dropout_1 | Dropout | input: (None, 3, 3, 128) | output: (None, 3, 3, 128)
conv2d_3 | Conv2D | relu | input: (None, 3, 3, 128) | output: (None, 3, 3, 256)
max_pooling2d_3 | MaxPooling2D | input: (None, 3, 3, 256) | output: (None, 1, 1, 256)
batch_normalization_1 | BatchNormalization | input: (None, 1, 1, 256) | output: (None, 1, 1, 256)
flatten | Flatten | input: (None, 1, 1, 256) | output: (None, 256)
dropout_2 | Dropout | input: (None, 256) | output: (None, 256)
dense | Dense | relu | input: (None, 256) | output: (None, 1024)
dense_1 | Dense | softmax | input: (None, 1024) | output: (None, 29)







```
Total params: 984,189
Trainable params: 983,613
Non-trainable params: 576
```

```
136/136 [==============================] - 7s 54ms/step - loss: 0.0309 - accuracy: 0.9923
Test accuracy is 99.22988414764404
Test loss is 0.030878271907567978
```

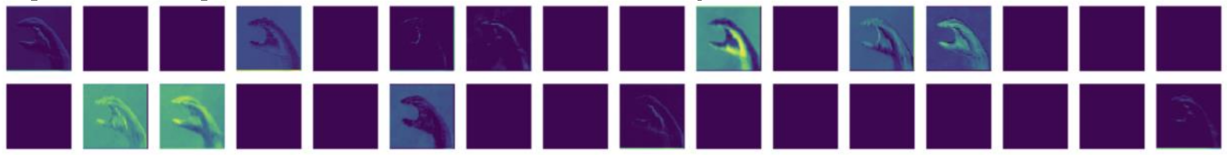
Model 5

Examining the learnings of the Model:

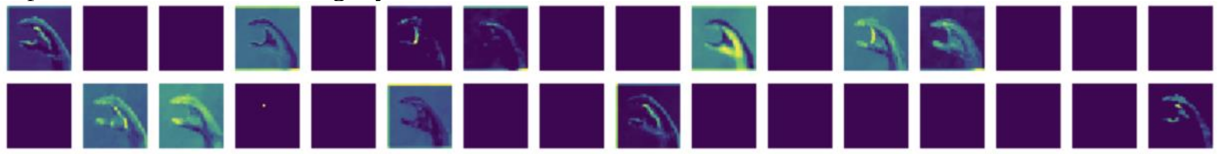
Let us examine each layer in our model that best classifies the ASL alphabet. The first convolutional layer in our model contains 32 filters, each filter of size (4, 4), zero-padding, the stride of one, and activation function as ReLU. It takes the input image and produces an output of 32 feature maps of input image size. Let us examine what features are learned by the first convolution layer and what the feature maps produced by the filters of the first layer look like.

For the input image(shown below), we looked at the output of each layer in the model that we trained.

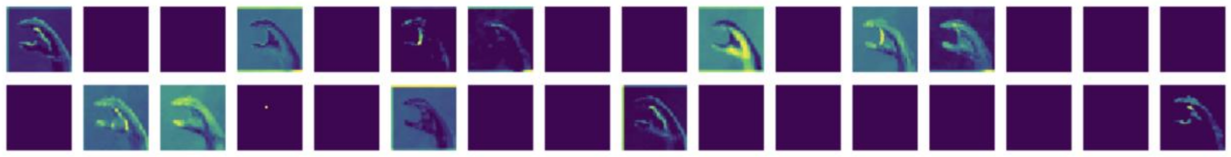
Output feature maps of the 32 filters of first convolutional layer



Output of the first MaxPooling layer

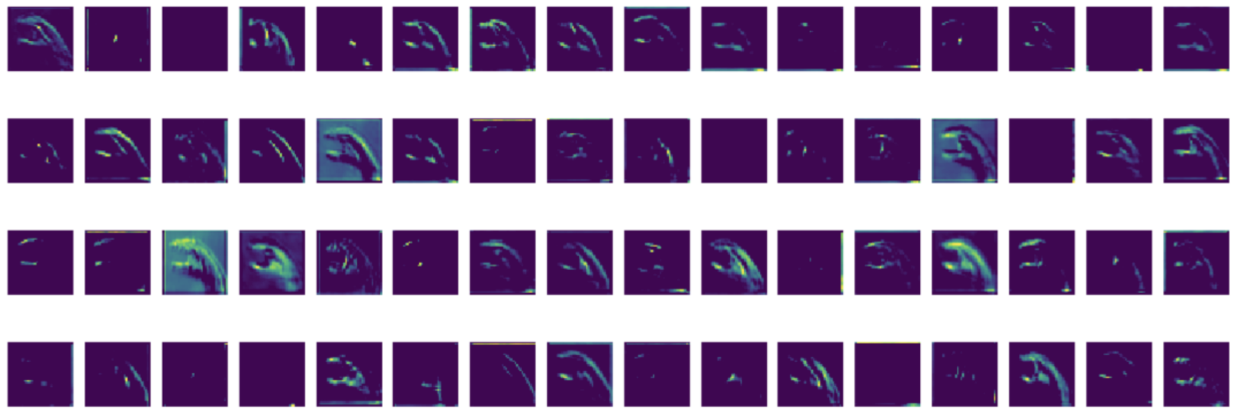


Output of the first Batch Normalization Layer

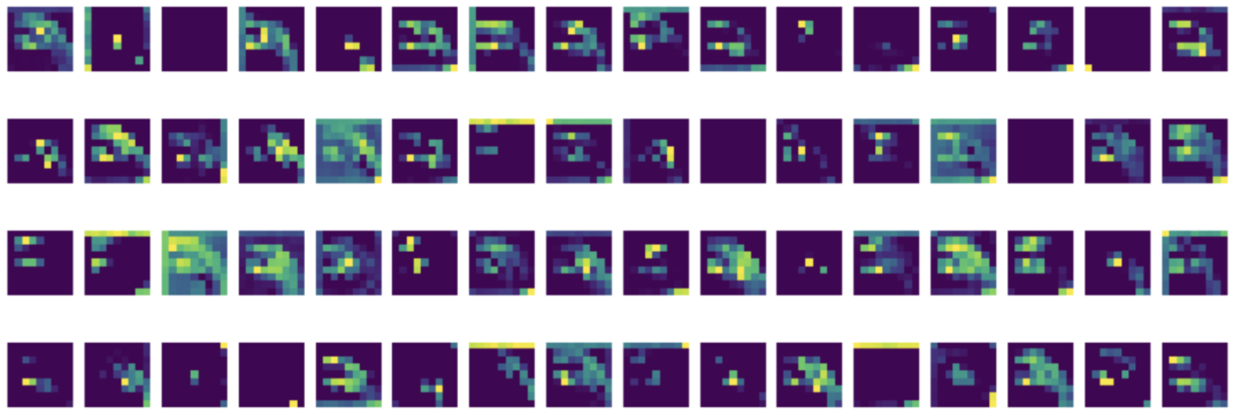


Each filter of the first convolutional layer learned different features from the input image. Almost all the filters of the first convolutional layer detected the edges of the hand in the given image. All the features it learns in the first layer will be fed to the deeper layers. The second convolutional layer contains 64 filters, each filter of size (4, 4), zero-padding, stride of one, activation function as ReLU. It takes the output of the previous layer which is 32 feature maps, and it learns higher level information from them and then outputs 64 feature maps. Let us look at the output of the next convolutional layer.

Output of the Second Convolutional Layer



Output of the Second MaxPooling Layer



From the visual representation of the outputs of the second convolution layer, we can see that the model starts to learn the deeper patterns of the images in the specific regions of the image. As we go deeper, the layers learn the higher-level representation of the input image that makes it different from other classes and useful in classifying it. This is how each layer of our model keeps on learning features from the previous layer's learning.

To make use of this model in real time, we had to develop a UI interface where users can interact with the model and take advantage of the model's classification capability. For this use case, we had to save the entire model and then load it later. Using Keras API, we saved the entire model into an artifact. We then developed a simple application using Python to capture the real-time images of the user and classify the ASL sign shown by the user using the saved model.

To capture and extract the hand gesture shown by the ASL user, we used OpenCV, CVZone, and mediapipe package which are powerful computer vision python packages. Using OpenCV, we access the web camera and capture the real-time expressions of the user, and using the HandDetector module of CVZone, we detect the hand and extract the hand gesture alone, then preprocess it by resizing it as per the image size used to train the model, rescale the input pixels from [0 to 255] range to [0 to 1] range and then feed it to our model. The model will classify the given hand gesture input to one of 29 labels. It will be displayed to the user.

```
Capturing...
The predicted alphabet is O
```

```
Capturing...
The predicted alphabet is B
```

```
Capturing...
The predicted alphabet is I
```

```
Capturing...
The predicted alphabet is C
```

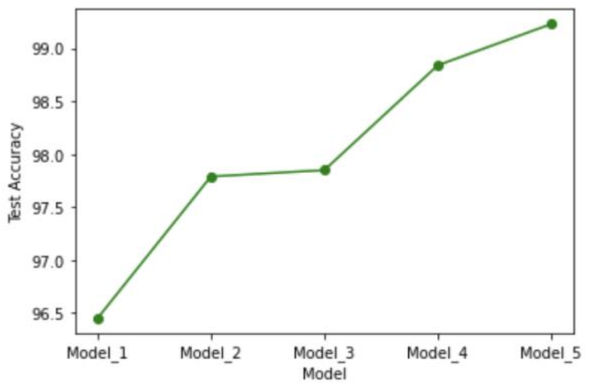
```
Capturing...
The predicted alphabet is E
```

```
Capturing...
The predicted alphabet is L
```

The model performs well in classifying real-time captures of the ASL hand gestures and classifies the alphabet signs with good accuracy.

Discussion

We noticed that the performance improves gradually with the addition of convolutional layers. It is due to that the deeper layers learn the higher-level local spatial patterns of the images which are specific to the class. Adding dropout layers helped in mitigating the overfitting problem and enables the model to generalize well outside the training data distribution. Batch Normalization helped in improving the overall performance of the model. The final model achieves the best accuracy of 99.23%.



Conclusion

In this project, we identified the problem of the communication gap between ASL users and people who don't understand it. To address this issue, we leveraged the power of deep learning technology to build a CNN model that classifies the ASL hand gestures to interpret them. We tried out different architectures and found the best one that produces good results. We then developed an application using Python that provides a user interface, captures the real-time hand gestures shown by the ASL user, feeds it to the best model that we built, and displays the corresponding English alphabet that was predicted by the model.

References

[1] Prangon Das, Tanvir Ahmed, Md. Firoj Ali "Static Hand Gesture Recognition for American Sign Language using Deep Convolutional Neural Network" DOI**:** 10.1109/TENSYMP50017.2020.9230772
[2] V. Bheda and D. Radpour, "Using deep convolutional networks for gesture recognition in American sign language," https://arxiv.org/pdf/1710.06836.pdf
[3] Garcia, Brandon and Viesca, Sigberto. "Real-time American Sign Language Recognition with Convolutional Neural Networks." In Convolutional Neural Networks for Visual Recognition at Stanford University, 2016. http://cs231n.stanford.edu/reports/2016/pdfs/214_Report.pdf
[4] Rastgoo, R.; Kiani, K.; Escalera, S. Sign Language Recognition: A Deep Survey. https://www.sciencedirect.com/science/article/abs/pii/S095741742030614X
[5] Kothadiya, D.; Bhatt, C.; Sapariya, K.; Patel, K.; Gil-González, A.-B.; Corchado, J.M. Deepsign: Sign Language Detection and Recognition Using Deep Learning. Electronics 2022, https://www.mdpi.com/2079-9292/11/11/1780