UNIVERSITY PARTNER

UNIVERSITY OF WOLVERHAMPTON

HERALD COLLEGE KATHMANDU

Artificial Intelligence and Machine Learning.

6CS012

An Image Classification with Convolutional Neural Network

University ID: 2329231

Student Name: Anish Bahadur Karki

University Email Address: A.B.Karki@wlv.ac.uk

Submitted Date: 2025/05/15

Abstract

This project evaluates a range of deep learning approaches to classify the images of sign language, measuring the efficiency of the custom Convolutional Neural Networks (CNNs) against transfer learning with the MobileNetV2. Sign language processing is a difficult task as there are variations in gestures, lighting condition, and shape of hands of different individuals. In the case of custom CNN models, several convolutional layers were designed, which extract features, and classify images while transfer learning leveraged pre-trained MobileNetV2 models fine-tuned with the dataset of sign language. According to the revealed results, transfer learning was a significantly better option than custom CNN models. The fine-tuned MobileNetV2 model exhibit excellent generalization performance achieving validation accuracy of close to 90% while compared to the baseline and deeper CNN models that were overwhelmed by the overfitting challenge. Due to transfer learning, the model has been able to converge faster and stabilize in training, and perform well with lower computational resources. With all these advantages, the models still exhibited some overfitting, particularly when the data set wasn’t that big. Future work would be able to overcome the shortcomings by using more diverse data set, using more data augmentation techniques, as well as testing hyperparameter optimization. Other studies of advanced architectures, or self-supervised learning methodologies, may also improve performance, especially in the low data cases. In general, the project presents the effectiveness of the transfer learning in the case of the sign language image classification, and especially if computational and data resources are limited.

Table of Contents

Contents

Table of Figure

1. Introduction

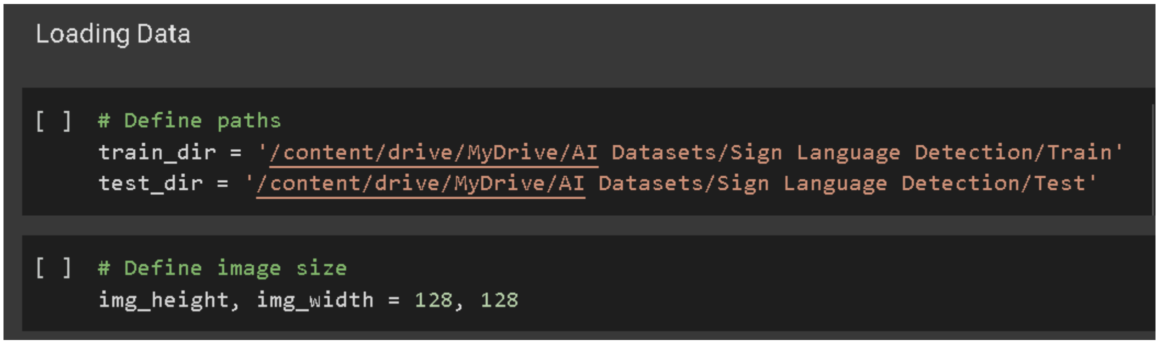
In this project, we are trying to categories a number of sign language gestures using deep learning. In their turn, the aim is to create the model that will be able to correctly recognize different hand signs when looking at their images. Such system might be helpful in a real life, for instance, in supporting technologies for the hearing and/or speech impaired people, in educational tools, or in gesture-based communication systems.

We are using Convolutional Neural Networks (CNNs) because they have provided such exceptional performance in the recognition of images. CNNs are able to learn about the information so that they can recognize different hand gestures.

Firstly, we import sign language images, then we perform the usual operations such as removing the useless data and enhancing the images for further work with them, which includes pre-processing. Then we build and train our CNN model with this data being ready. Once we are through with training, we utilize the model on new images and determine its performance. We conduct accuracy metrics and loss graphs, a confusion matrix to analyses the model performance.

2. Dataset

2.1. Source



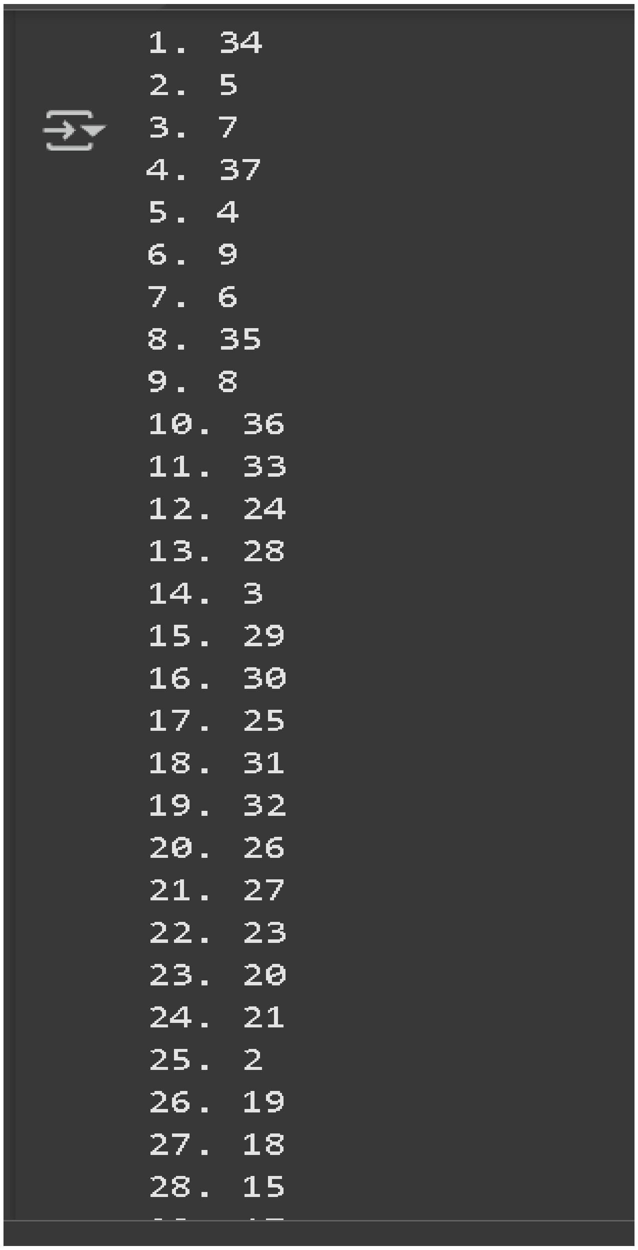
Loading Data

```
# Define paths
train_dir = '/content/drive/MyDrive/AI Datasets/Sign Language Detection/Train'
test_dir = '/content/drive/MyDrive/AI Datasets/Sign Language Detection/Test'
```

```
# Define image size
img_height, img_width = 128, 128
```

Figure 1 Datasets Source

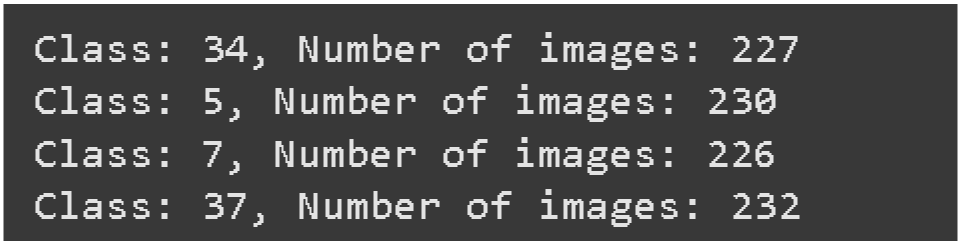
The dataset employed in the course of this project was given by our Module Leader and includes 38 classes that can be considered sign language gestures. Every class is labelled numerically from 0 to 37; each number representing a distinct sign.



```
1.  34
2.  5
3.  7
4.  37
5.  4
6.  9
7.  6
8.  35
9.  8
10.  36
11.  33
12.  24
13.  28
14.  3
15.  29
16.  30
17.  25
18.  31
19.  32
20.  26
21.  27
22.  23
23.  20
24.  21
25.  2
26.  19
27.  18
28.  15
```

Figure 2 38 Different Classes

The dataset consists of about 12,000 images comprising of 8,000 training images and 4,000 testing images. Each class has on average approximately 250 images, thus creating a balanced distribution in all categories.



```
Class: 34, Number of images: 227
Class: 5, Number of images: 230
Class: 7, Number of images: 226
Class: 37, Number of images: 232
```

Figure 3 Per Class Image Contains

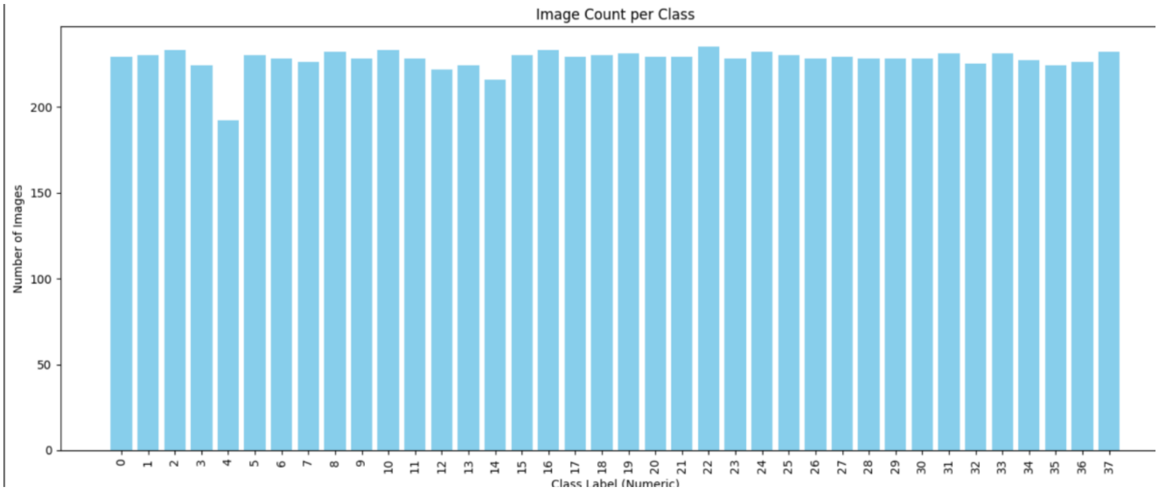


Figure 4 Visualization of Our Datasets

All pictures are RGB formatted, and have different sizes. In order to ensure consistency and the best input for training the CNN model, all the images were resized to 128 x 128 pixels. Resized to 128 x 128 pixels.

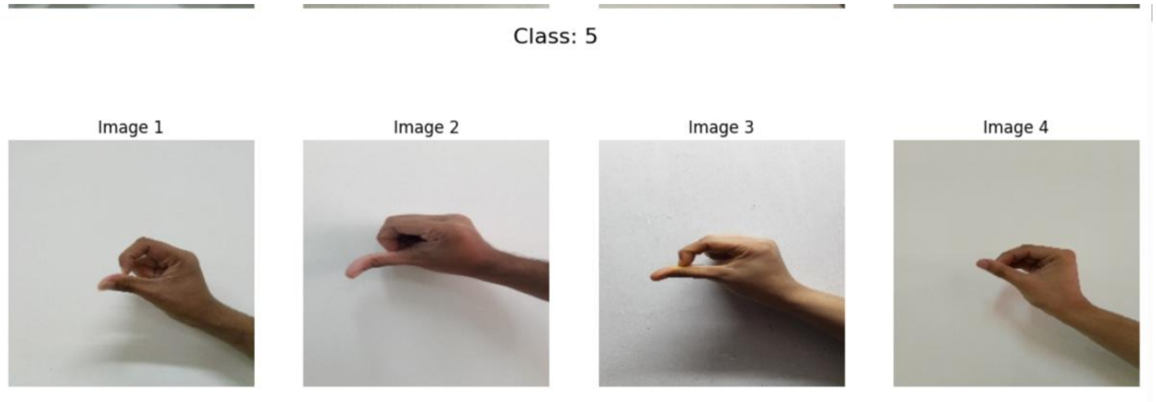


Figure 5 Class 5 Image Sample

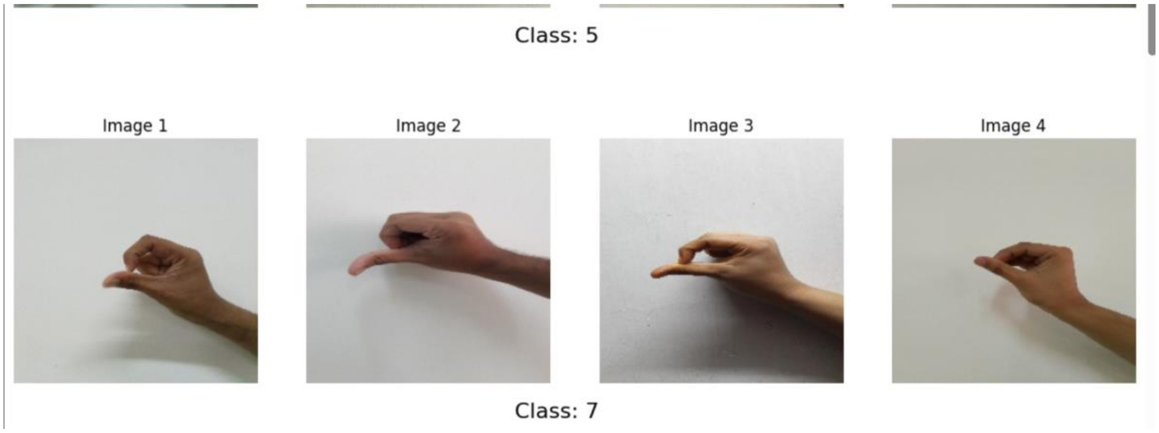


Figure 6 Class 7 Image Sample

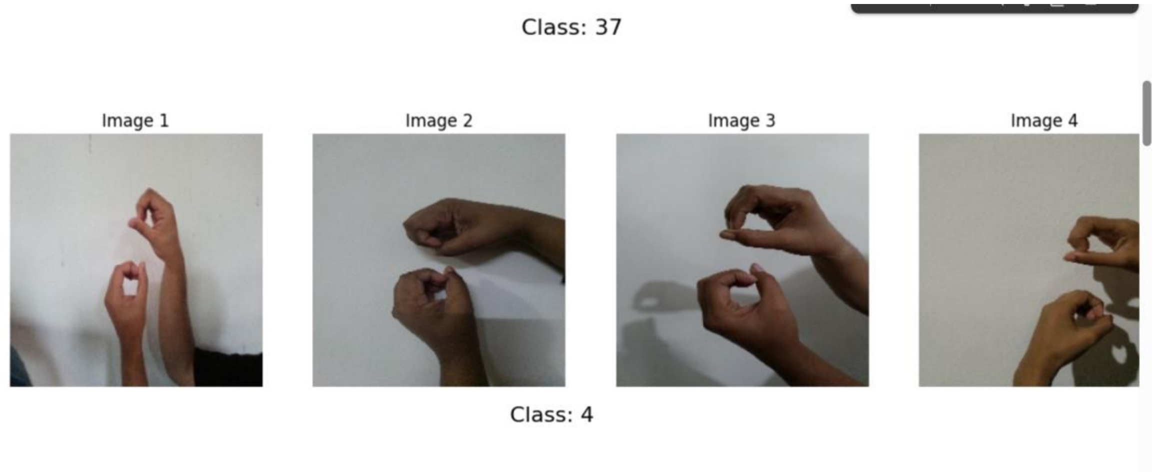
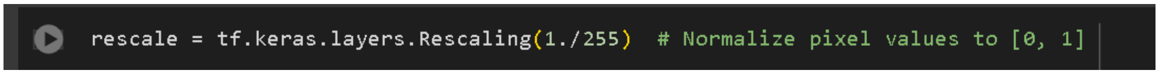


Figure 7 Class 37 Image Samples

2.2. Pre-processing

I. All sign language images' pixel values were normalised according to a scaling of its range, which was initial 0-255 to be between 0 and 1. This normalisation process assists the model to train more efficiently and it increases the rate of convergence speed and accuracy.



```
rescale = tf.keras.layers.Rescaling(1./255)  # Normalize pixel values to [0, 1]
```

Figure 8 Normalize Pixels Values

I. Data augmentation techniques were employed to amplify the diversity of the training data set and make the model generalise better. Some of these tactics were random horizontal flipping, rotation and image shift. Augmentation diminishes the overfitting threat because exposing the model to more variations for each class of signs will be implemented.

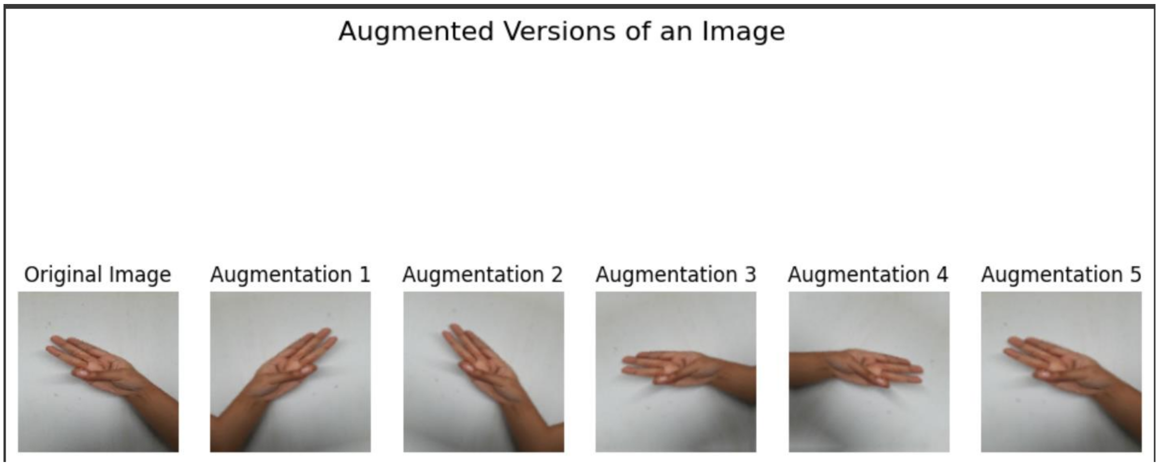


Figure 9 Augmented version of an Image

I. The dataset was structured in terms of the two primary directories. a training one and another one for testing. Furthermore, the training data was also divided into 80% for the training and 20% for validation. This validation set was used for monitoring the learning performance of the model without exposing itself to the test data while training.

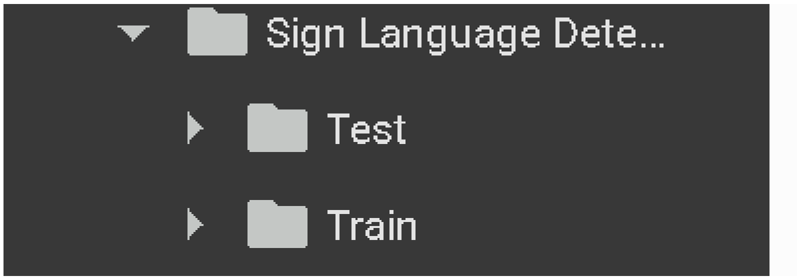
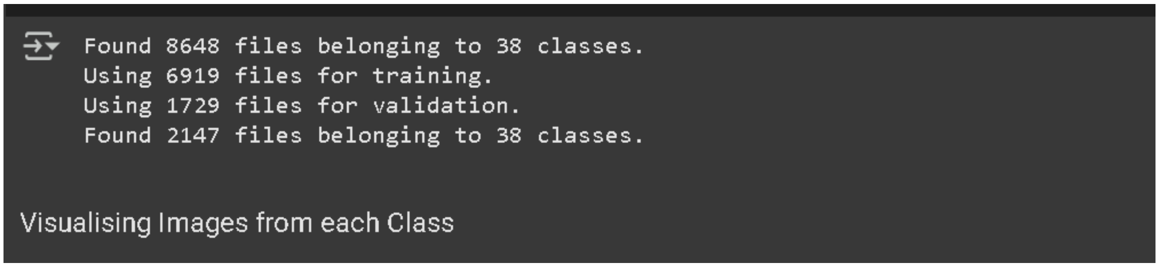


Figure 10 Datasets Split



```
Found 8648 files belonging to 38 classes.
Using 6919 files for training.
Using 1729 files for validation.
Found 2147 files belonging to 38 classes.
```

Visualising Images from each Class

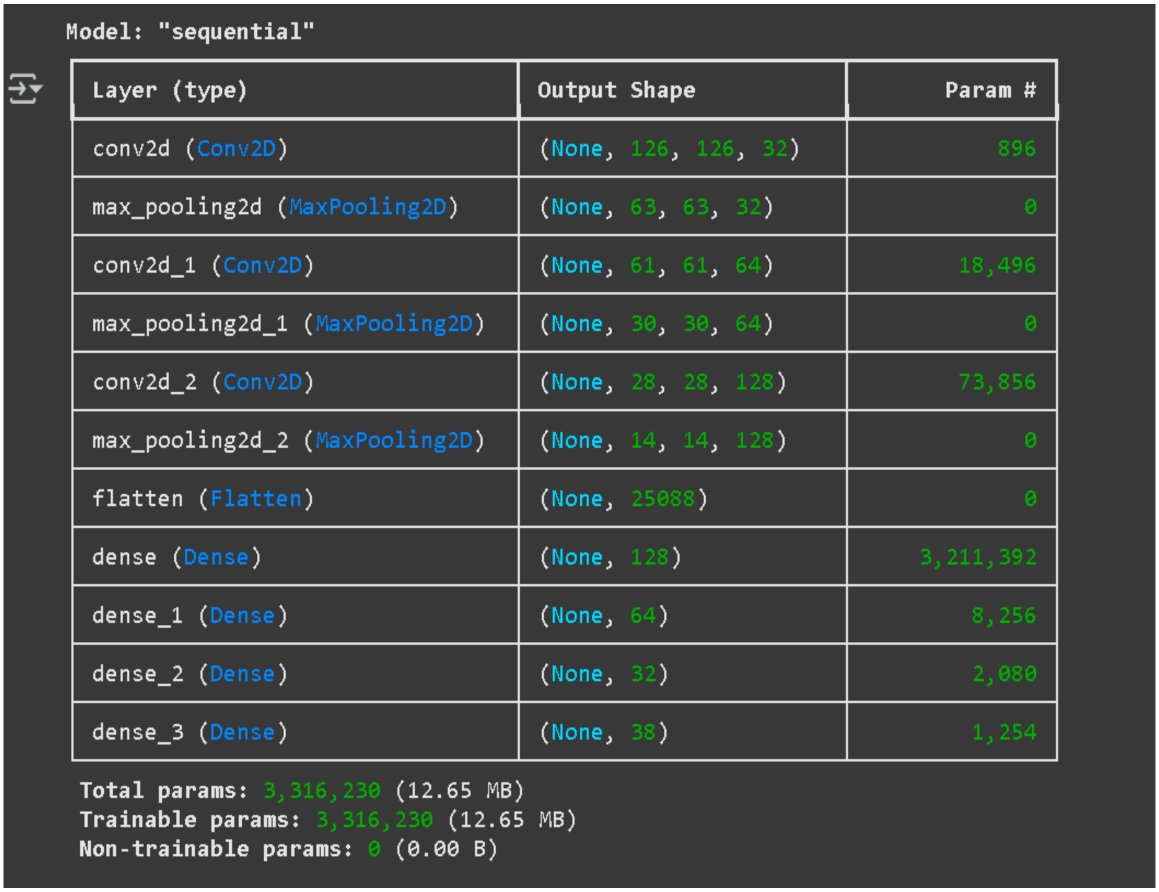
Figure 11 Split of Images for Train and Validation

3. Methodology

This project used two models of the Convolutional Neural Network (CNN) which were designed and trained in order to recognise the sign language gestures. The first is a baseline model with no pretentions, whereas the second is a deeper model that is characterised by more layers and complexity. Both of the models were designed to process the input images in the RGB format of 128 × 128 × 3 (width, height, colour channels). The aim of the both models was the adequate Building of all the images and their classification into one of the 38 set of the sign language labelled with a number from 0 to 37.

3.1. Baseline Model

The **baseline model** was a simple Convolutional Neural Network (CNN) consisting of the following components:



```
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                     ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ conv2d (Conv2D)                  │ (None, 126, 126, 32)   │           896 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d (MaxPooling2D)     │ (None, 63, 63, 32)     │             0 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_1 (Conv2D)                │ (None, 61, 61, 64)     │        18,496 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_1 (MaxPooling2D)   │ (None, 30, 30, 64)     │             0 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_2 (Conv2D)                │ (None, 28, 28, 128)    │        73,856 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_2 (MaxPooling2D)   │ (None, 14, 14, 128)    │             0 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten (Flatten)                │ (None, 25088)          │             0 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                    │ (None, 128)            │     3,211,392 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                  │ (None, 64)             │         8,256 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_2 (Dense)                  │ (None, 32)             │         2,080 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_3 (Dense)                  │ (None, 38)             │         1,254 │
└──────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 3,316,230 (12.65 MB)
 Trainable params: 3,316,230 (12.65 MB)
 Non-trainable params: 0 (0.00 B)
```

Figure 12 Baseline CNN Model

1. **Convolutional Layers:**

- The model had **three convolutional layers**.
- The number of filters increased in each layer: **32**, **64**, and **128**.
- Each layer used a **3 × 3 kernel size**.
- **ReLU** was applied as the activation function after each convolution.
- A **2 × 2 max pooling** layers followed each convolution to reduce the spatial dimensions of the feature maps.
- **Stride** was set to 1 for the convolution layers and 2 for the pooling layers.

2. **Fully Connected (Dense) Layers:**

- After flattening the output of the convolutional layers, two dense layers were added.

- The first dense layer had **128 units** with **ReLU** activation.
- The second dense layer had **64 units**, also with **ReLU** activation.

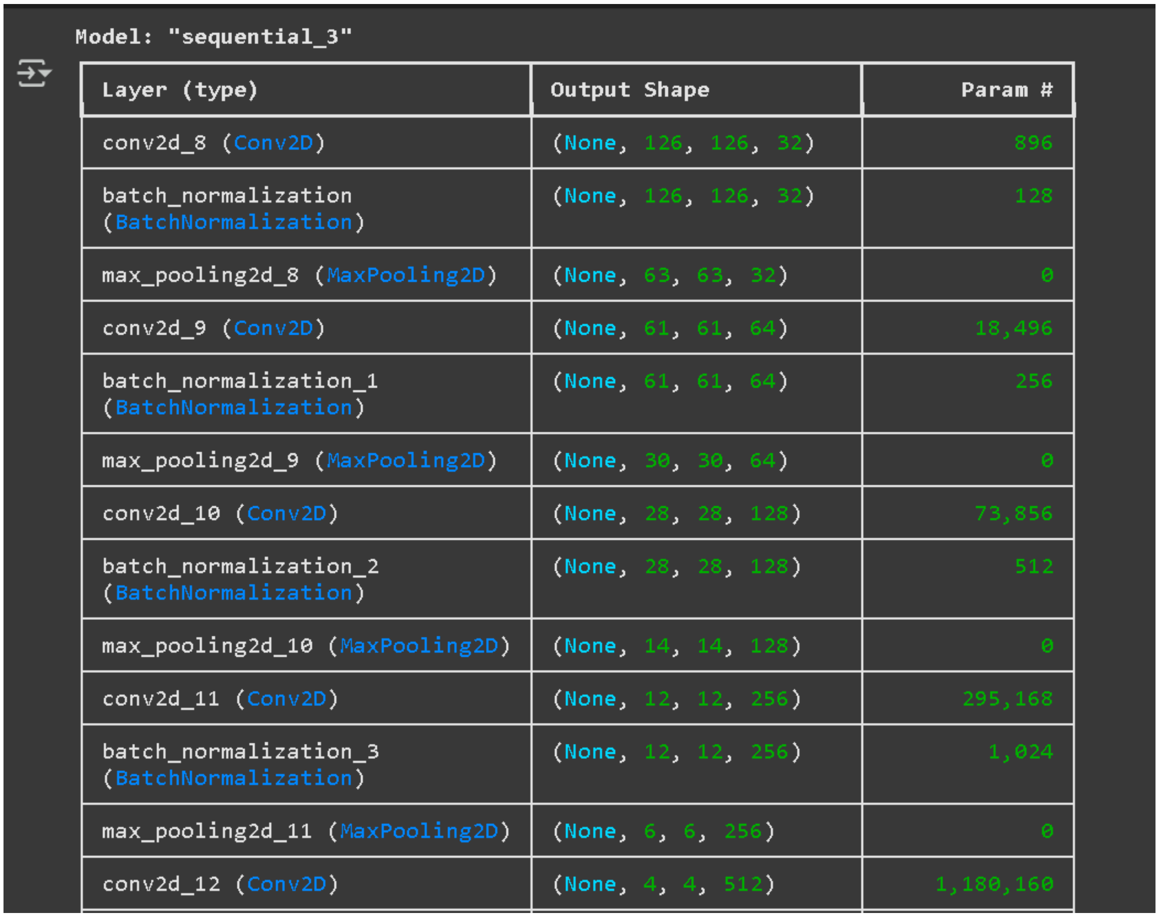
3. **Output Layer:**

- The final output layer consisted of **38 units** (corresponding to the 38 sign language classes).
- A **softmax** activation function was used to generate a probability distribution across the 38 classes.

4. **Model Parameters:**

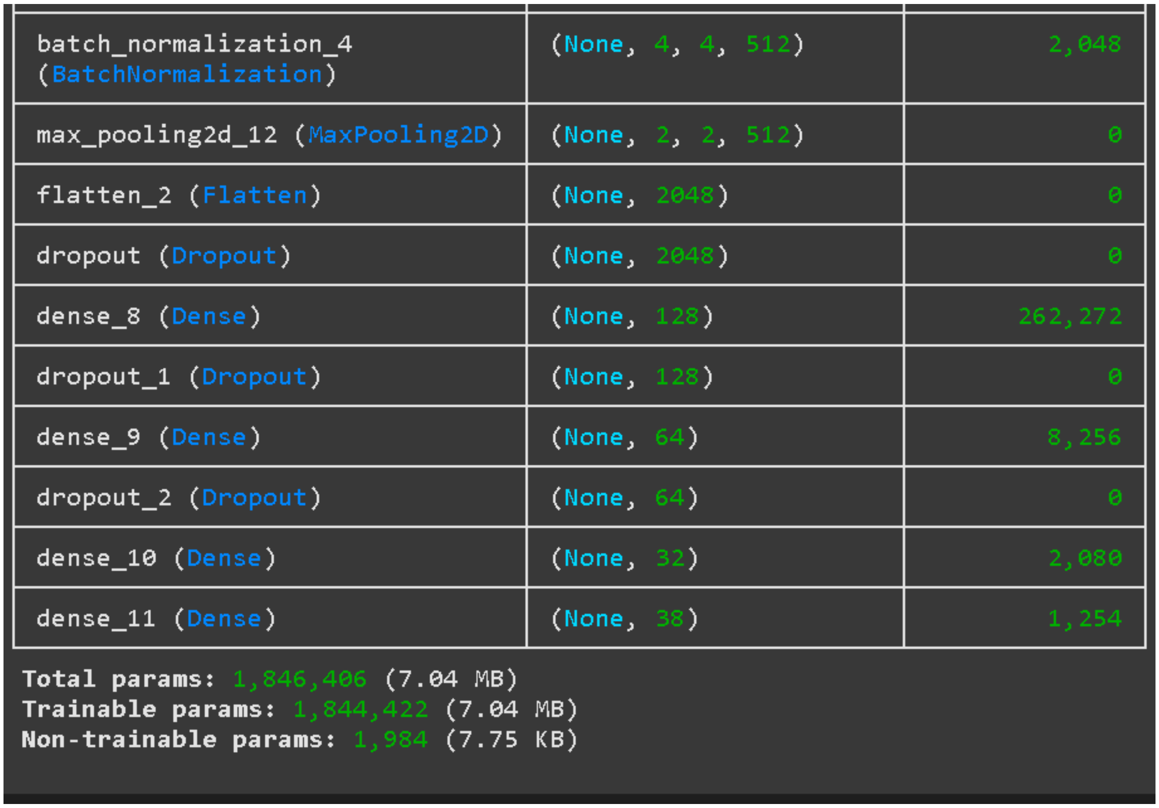
- The total number of trainable parameters in this model was approximately **3,316,230**.

3.2. Deeper Model



```
Model: "sequential_3"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ conv2d_8 (Conv2D)               │ (None, 126, 126, 32)   │           896 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization             │ (None, 126, 126, 32)   │           128 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_8 (MaxPooling2D)  │ (None, 63, 63, 32)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_9 (Conv2D)               │ (None, 61, 61, 64)     │        18,496 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_1           │ (None, 61, 61, 64)     │           256 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_9 (MaxPooling2D)  │ (None, 30, 30, 64)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_10 (Conv2D)              │ (None, 28, 28, 128)    │        73,856 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_2           │ (None, 28, 28, 128)    │           512 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_10 (MaxPooling2D) │ (None, 14, 14, 128)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_11 (Conv2D)              │ (None, 12, 12, 256)    │       295,168 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_3           │ (None, 12, 12, 256)    │         1,024 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_11 (MaxPooling2D) │ (None, 6, 6, 256)      │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_12 (Conv2D)              │ (None, 4, 4, 512)      │     1,180,160 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
```

Figure 13 Deeper CNN Model



```
│ batch_normalization_4            │ (None, 4, 4, 512)         │          2,048 │
│ (BatchNormalization)             │                           │                │
├──────────────────────────────────┼───────────────────────────┼────────────────┤
│ max_pooling2d_12 (MaxPooling2D)  │ (None, 2, 2, 512)         │              0 │
├──────────────────────────────────┼───────────────────────────┼────────────────┤
│ flatten_2 (Flatten)              │ (None, 2048)              │              0 │
├──────────────────────────────────┼───────────────────────────┼────────────────┤
│ dropout (Dropout)                │ (None, 2048)              │              0 │
├──────────────────────────────────┼───────────────────────────┼────────────────┤
│ dense_8 (Dense)                  │ (None, 128)               │        262,272 │
├──────────────────────────────────┼───────────────────────────┼────────────────┤
│ dropout_1 (Dropout)              │ (None, 128)               │              0 │
├──────────────────────────────────┼───────────────────────────┼────────────────┤
│ dense_9 (Dense)                  │ (None, 64)                │          8,256 │
├──────────────────────────────────┼───────────────────────────┼────────────────┤
│ dropout_2 (Dropout)              │ (None, 64)                │              0 │
├──────────────────────────────────┼───────────────────────────┼────────────────┤
│ dense_10 (Dense)                 │ (None, 32)                │          2,080 │
├──────────────────────────────────┼───────────────────────────┼────────────────┤
│ dense_11 (Dense)                 │ (None, 38)                │          1,254 │
└──────────────────────────────────┴───────────────────────────┴────────────────┘
 Total params: 1,846,406 (7.04 MB)
 Trainable params: 1,844,422 (7.04 MB)
 Non-trainable params: 1,984 (7.75 KB)
```

Figure 14 Deeper CNN Model 2

1. Block 1: Initial Convolutional Block

- **Conv2D Layer**: The first layer is a 2D convolutional layer with 32 filters, a kernel size of (3,3), and the ReLU activation function. Convolutional layers are used to extract features from the input images.
- **Batch Normalization**: This layer normalizes the activations in the network, speeding up training and making the network more stable.
- **MaxPooling2D**: A max-pooling layer with a pool size of (2,2) is used to reduce the spatial dimensions (height and width) of the feature maps, which helps in reducing the computational load and preventing overfitting.

2. **Block 2: Second Convolutional Block**

- **Conv2D Layer**: The second convolutional layer has 64 filters and uses the same kernel size of (3,3). The activation function is still ReLU.
- **Batch Normalization**: Another normalization step to stabilize learning.

- **MaxPooling2D**: This layer further reduces the spatial dimensions of the feature map.

3. **Block 3: Third Convolutional Block**

- **Conv2D Layer**: This layer has 128 filters, further increasing the model's capacity to capture complex features.
- **Batch Normalisation**: Normalization continues to optimize the learning process.
- **MaxPooling2D**: Again, pooling is applied to down sample the feature map.

4. **Block 4: Fourth Convolutional Block**

- **Conv2D Layer**: The fourth block uses 256 filters, indicating an even deeper level of abstraction in feature extraction.
- **Batch Normalization**: As in previous blocks, this normalizes the activations.
- **MaxPooling2D**: A pooling operation further reduces the spatial dimensions of the feature maps.

5. **Block 5: Fifth Convolutional Block**

- **Conv2D Layer**: The fifth block has 512 filters, making it the deepest layer in the network. The number of filters increases as we go deeper to capture more complex patterns in the data.
- **Batch Normalization**: Normalization continues for stability.
- **MaxPooling2D**: Pooling is applied to further down sample the feature maps.

6. **Flattening and Dense Layers:**

- **Flatten**: After the convolutional blocks, the 2D feature maps are flattened into a 1D vector to prepare for the fully connected (dense) layers. This step converts the output of the convolutional layers into a format that can be fed into a dense layer.

- **Dropout (0.5)**: A dropout layer with a rate of 50% is applied to prevent overfitting. Dropout randomly drops units from the network during training, which helps the model generalize better.

7. Fully Connected Layers:

- **Dense Layer (128 units)**: The first dense layer has 128 neurons and uses the ReLU activation function.
- **Dropout (0.4)**: A second dropout layer is applied with a rate of 40% to further reduce overfitting.
- **Dense Layer (64 units)**: The second dense layer contains 64 neurons and uses ReLU as the activation function.
- **Dropout (0.3)**: Another dropout layer with a rate of 30% is applied to help prevent overfitting.
- **Dense Layer (32 units)**: This layer contains 32 neurons, also using ReLU activation.

8. **Output Layer:**

- **Dense Layer (38 units)**: The final layer has 38 neurons, corresponding to the number of classes in the classification task. This layer uses the softmax activation function, which is commonly used for multi-class classification problems as it outputs a probability distribution over the 38 classes.

3.3. Loss Function

Sparse Categorical Crossentropy loss function was chosen because it works very well for multi-class classification problems, which have class labels represented as integers and not as one-hot encoded vectors. This function calculates cross-entropy between true labels and predicted probabilities without manual one-hot encoding, which ensures streamlining data pre-processing and enhancing performance efficiency. It is especially suitable for the last output layer of the model (that uses a softmax activation to output a probability distribution over several categories (in the case at hand, 38 categories).

3.4. Optimizer

For training the model, it was reasoned that the Adam optimizer was to be used, owing to its efficiency and effectiveness with deep learning tasks. It automatically regulates the learning rate during training with the basis on the first and second moments of the gradients, leading to faster convergence and fewer oscillations as opposed to the conventional stochastic gradient descent (SGD). This makes Adam ideally suitable for complicated models such as deep convolutional neural networks, wherein optimization has to be adaptive and efficient.

3.5. Hyperparameters

We used the CNN model that contains a learning rate of 0.001, a batch size of 32, and 20 training epochs. These are generic values which strike a nice balance between training speed and performance of a model. The learning rate determines how fast the model adjusts its weights, and 0.001 is suitable for stable convergence. An adequate batch size is 32 – it will help to train the model efficiently with no high amounts of memory usage. 20 epochs of training the model allows it to become knowledgeable from the data without overfitting. Compared comprehensively, these settings ensure efficient training of the model and good accuracy of multi-class classification tasks.

4. Experiments and Results

To measure the performance of both models, an experiment utilising the prepared fruit dataset has been conducted. Training was carried out for 20 epochs, and the training and validation performance were observed at every epoch. This was useful in evaluating how well the models learned from the data and how good they were in generalising to unseen validation samples.

4.1. Baseline vs Deeper Architecture

a. Accuracy

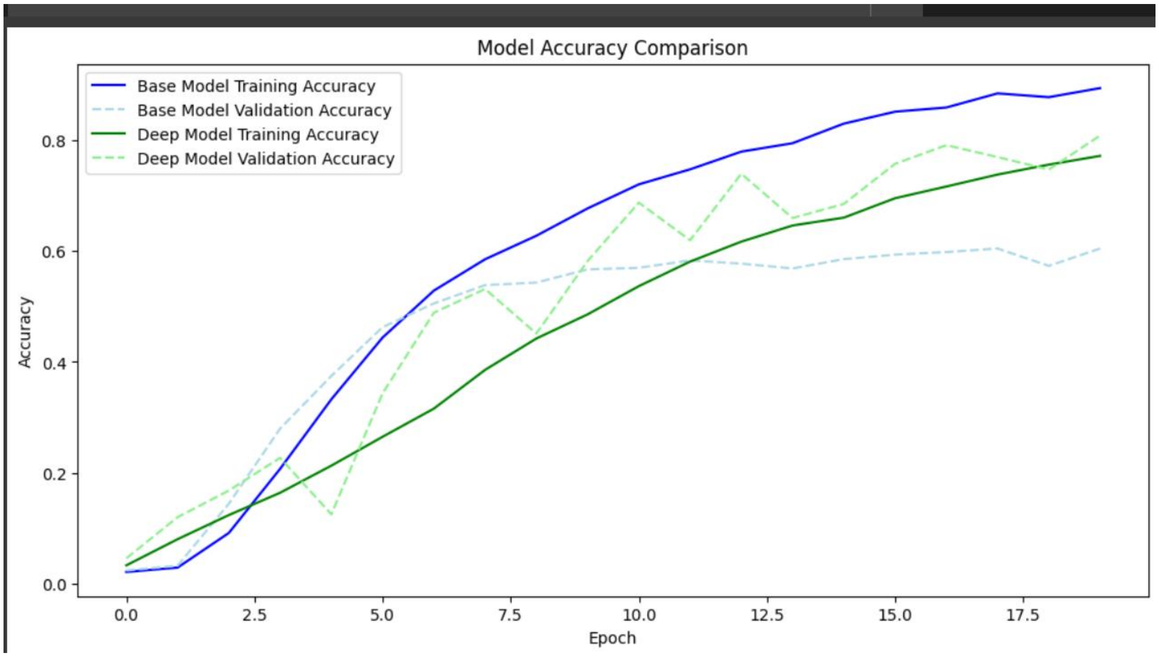


Figure 15 Model Accuracy Comparison

Performance Comparison

1. Baseline Model:

- **Training Accuracy:** ~52.99%
- **Training Loss:** 1.7737
- **Test Accuracy:** 55.43%
- **Test Loss:** 1.6420
- The baseline model showed limited learning capability. Both training and test accuracies were low, indicating that the model struggled to extract useful features from the data.

2. Deeper Model:

- **Training Accuracy:** ~78.90%
- **Training Loss:** 0.7273
- **Test Accuracy:** 80.48%
- **Test Loss:** 0.6832
- The deeper model performed significantly better, with much higher training and test accuracies. This suggests that the added depth allowed the model to learn more complex features and generalize better to unseen data.

b. Loss

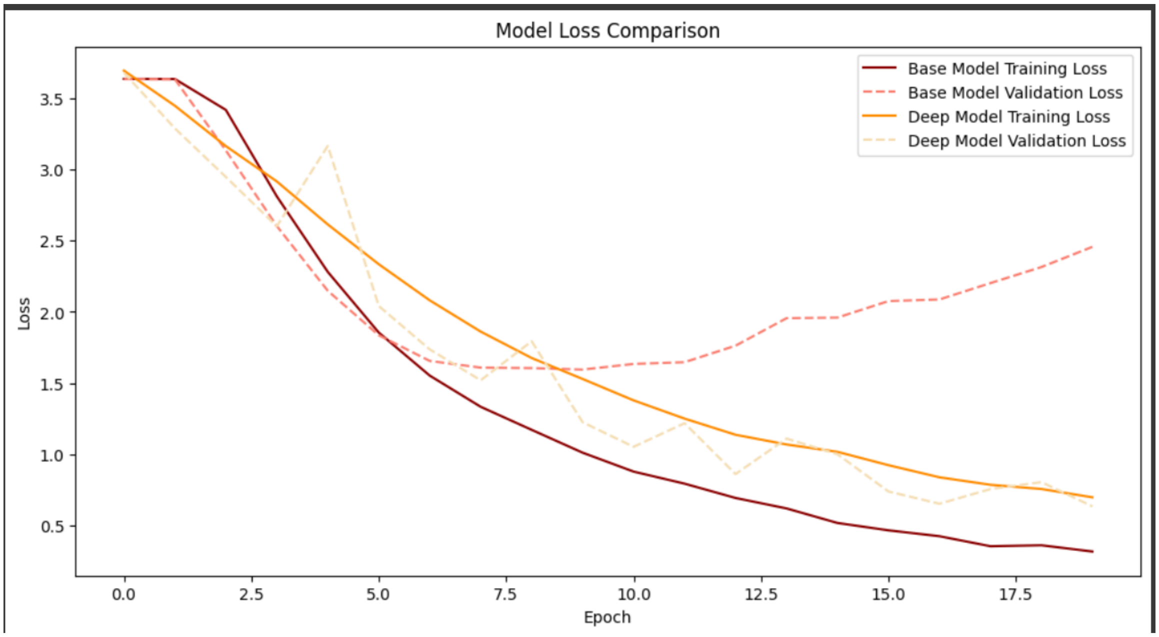


Figure 16 Model Loss Comparison

1. Baseline Model:

- Training loss dropped sharply and approached zero.

- However, validation loss increased significantly after epoch 5, rising above **2.5**, indicating **clear overfitting**. The model memorized the training data but failed to generalize.

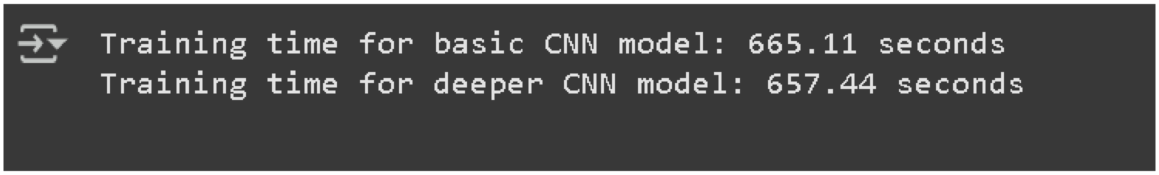
2. Deeper Model:

- Training loss decreased gradually to around **0.5**.
- Validation loss remained more stable, fluctuating between **1.3 and 1.8** throughout training, showing better learning behaviour.

3. Analysis:

- The deeper model’s more stable validation loss suggests it **learned meaningful and generalizable features**, rather than overfitting the training data. This led to improved performance on unseen examples.

c. Training Time



```
Training time for basic CNN model: 665.11 seconds
Training time for deeper CNN model: 657.44 seconds
```

Figure 17 Training Time CNN Model and Deeper CNN Model

Adam Vs SGD

a) Accuracy

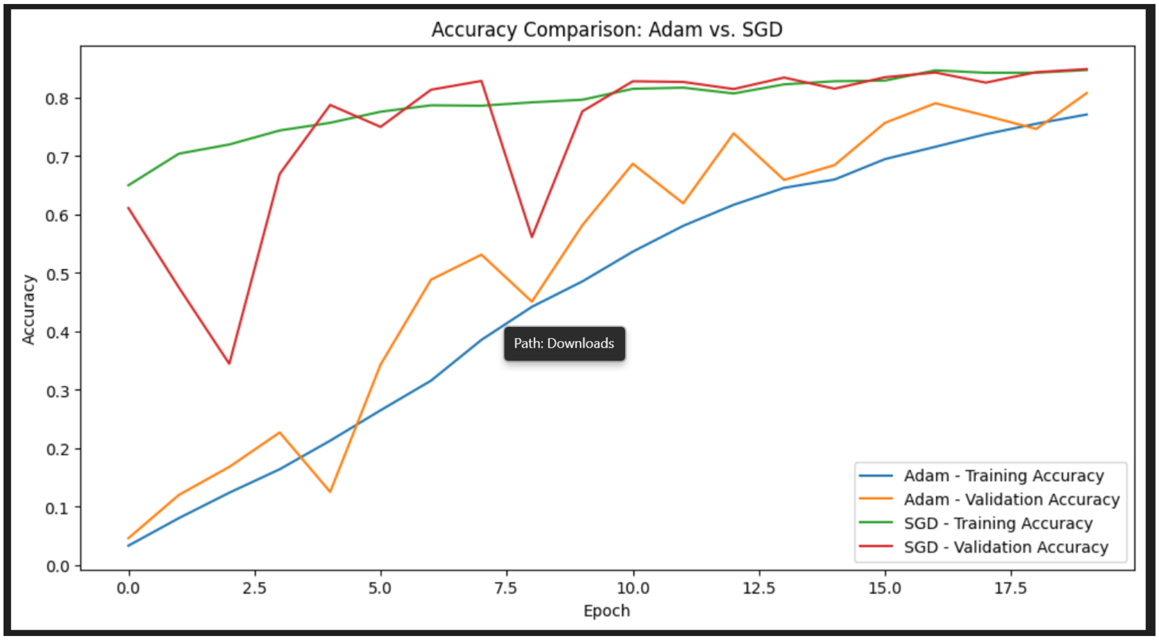


Figure 18: Accuracy Comparison Adam vs SGD

This Figure Shows the Comparison of the training and Validation Accuracy over 20 epochs for two differ optimization and they are Adam and SGD. This shows the number of train epochs and accuracy. The above lines show the performance two for Adam and respective SGD.

The Blue line shows the training accuracy of Adam. It starts at a low and increases to reach 0.75 by the end of an epoch. The orange line shows representing Adam validation accuracy.

In Conclusion, Adam provides more steady and reliable performance on the validation side, whereas SGD is doing great in training accuracy, but feels difficulties in consistent validation accuracy, like due to sensitivity to hyperparameters.

b) Loss

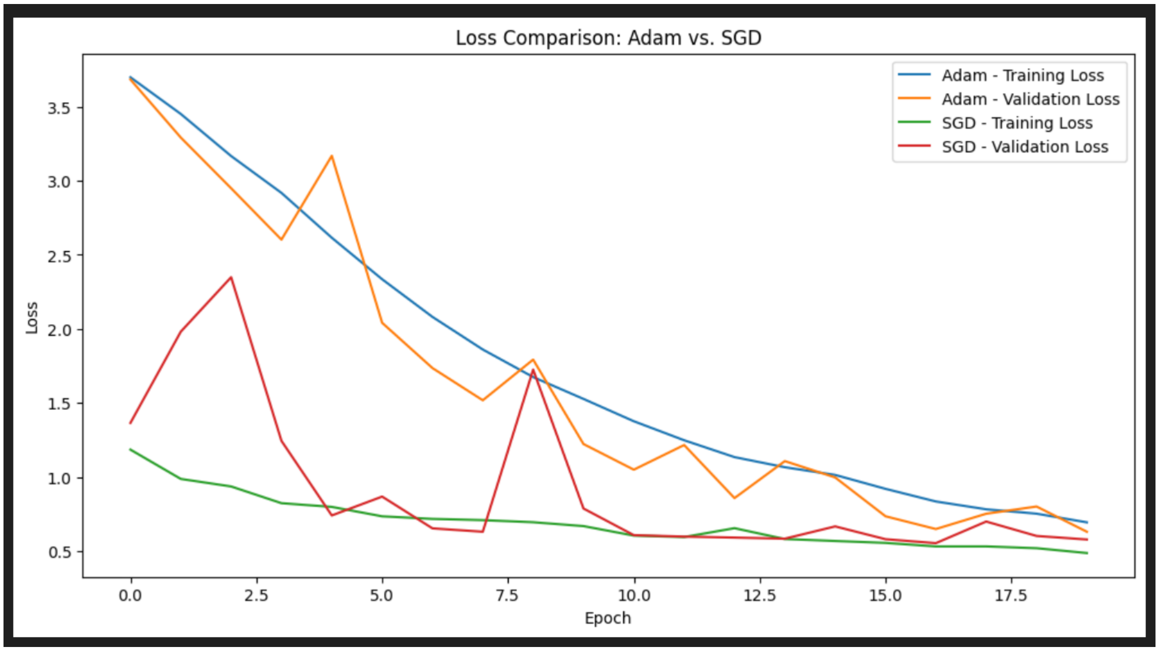


Figure 19: Loss Comparison Adam vs SGD

The figure on the above shows the comparison between training and validation loss of two optimization one is Adam and SGD by the over 20 epochs. Adam offers Smoother and more steady performance in both but the SGD is low training loss, effected by unstable validation loss maybe due to overfitting or incorrect generalization.

4.2. Comparison of all Models

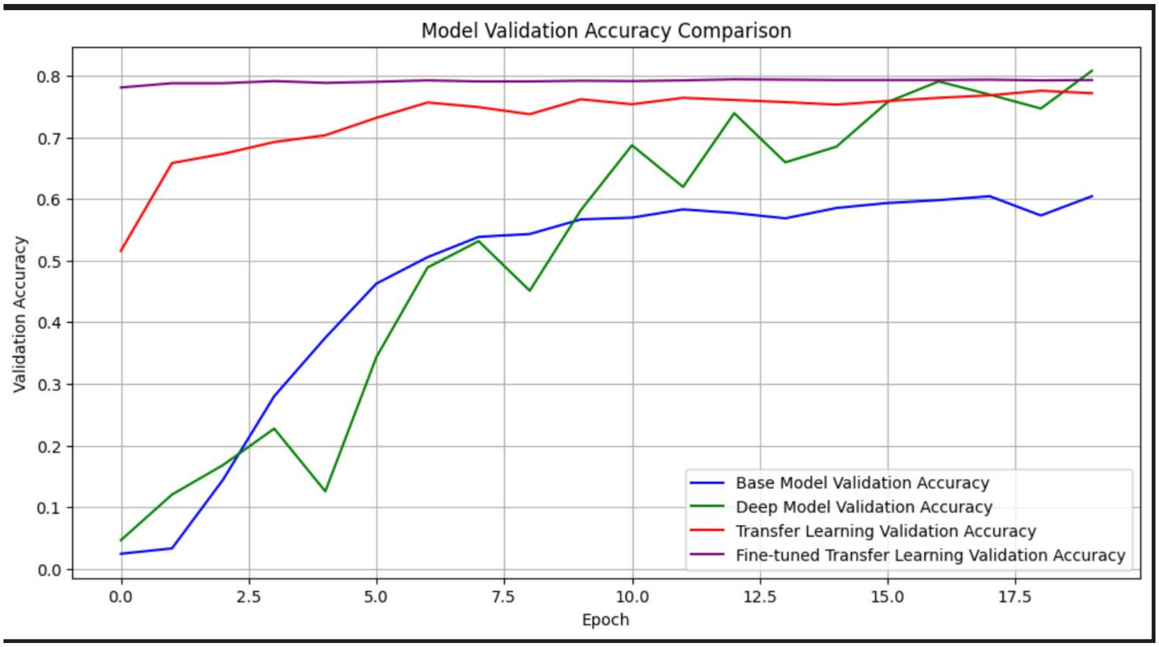


Figure 20 Comparison of all Models

This figure shows the validation accuracy of different models during 20 training epochs. On the x-axis, we see the number of epochs, and on the y-axis, we see the validation accuracy (i.e., how well our model is performing on data it has never seen). Four models is considered: base model, deep model, transfer learning and fine-tuned transfer learning models.

The blue line representing the base model starts off with super low accuracy and just gets better over time, and hovers around 0.6 at the end of training. The deep model, shown in green, also begins with a low accuracy but is more volatile and it goes above the base model, reaching around 0.8. It also indicates that deeper networks might perform better but are harder to train and more unstable in the early stages.

The red line is the transfer learning model, which begins much higher in accuracy (~0.52) due to pre-trained knowledge. It rises extremely quickly and is well above both base and deep models, confirming emphatically the value of pre-trained features. The best performing model is fine-tuned transfer learning (purple), sorry.

The model begins high and stays relatively flat across, wobbling around 0.8 accuracy.

5. Conclusion and Future Work

This project investigated various methods of utilizing deep learning to identify sign language images, where custom CNN models stood in comparison with transfer learning through MobileNetV2. The results showed that the transfer learning achieved better results compared dot the custom models. The best validation accuracy in the fine-tuned MobileNetV2 model was achieved, which is roughly 80% to 90%; while far exceeding the models developed from scratch. Although there was some level of learning by the baseline and the deeper CNN models, they tended to overfit and had poor generalization. The capability to use pre-trained features enabled the model to converge in a swifter manner, keep it more stable, and reduce the use of computational capacity. Nevertheless, the usage of a relatively small and homogenous dataset was one of the limitations, and even the carefully tuned models demonstrated some overfitting. One of the possible directions for further research would be the use of a more diverse dataset, the use of other data augmentation techniques, and the utilization of hyperparameter tuning to improve the outcome. Subsequent research may focus on other more sophisticated architectures or autonomy in learning to enhance performance in the case of limited data. After all is said and done, this project establishes the fact that transfer learning is indeed a very powerful and effective approach to address the image classification challenges, especially if the same is constrained by limited data as well as computational resources.