Owen Casey / G00383711

Abstract

This project explores hand gesture recognition utilizing convolutional neural networks (CNNs) through the HaGRID dataset. From scratch and transfer learning based (VGG-16) CNN models are used to compare performance and efficiency, providing insights into CNN architecture proficiencies and limitations.

Keywords: convolutional neural networks, hand gesture recognition, HaGRID dataset, transfer learning, VGG-16, data augmentation, RGB & grayscale.

1 Introduction

This project investigates and analyzes hand gesture recognition through the use of convolutional neural networks (CNNs). Hand gesture recognition is crucial in many areas of technology, such as virtual reality, robotics, and human-computer interactions. Gesture recognition is often challenging due to the potential for variations in environment, lighting, and orientation. The HaGRID dataset is utilized to provide a comprehensive collection of hand gesture images which are spread across 18 different consistent classes [1].

The overall objective of this project is to design and compare multiple CNN models, analyzing their proficiency in recognizing hand gestures from the HaGRID dataset. Key objectives include:

- Building and analyzing CNN models from scratch, comparing performance with a pre-trained, transfer learning based model, VGG-16.
- Investigating the effect of image size and color (RGB vs. grayscale) on model performance.
- Exploring deeper layered architecture and data augmentation and their affect on performance.
- Evaluate each model on their accuracy, loss, speed of training, and ability to accurately identify gestures.

2 Dataset and Splitting

The original Hand Gesture Recognition Image Dataset (HaGRID) is over 700GB in size, containing 554,800 images classified into 18 unique classes, with each image coming with full RGB color as standard. Due to the original dataset being so large, a reduced version of the dataset is used. The dataset utilized in the project is around 4GB in size, containing 125,912 images in total. This reduction in size is essential for efficient processing of the dataset.

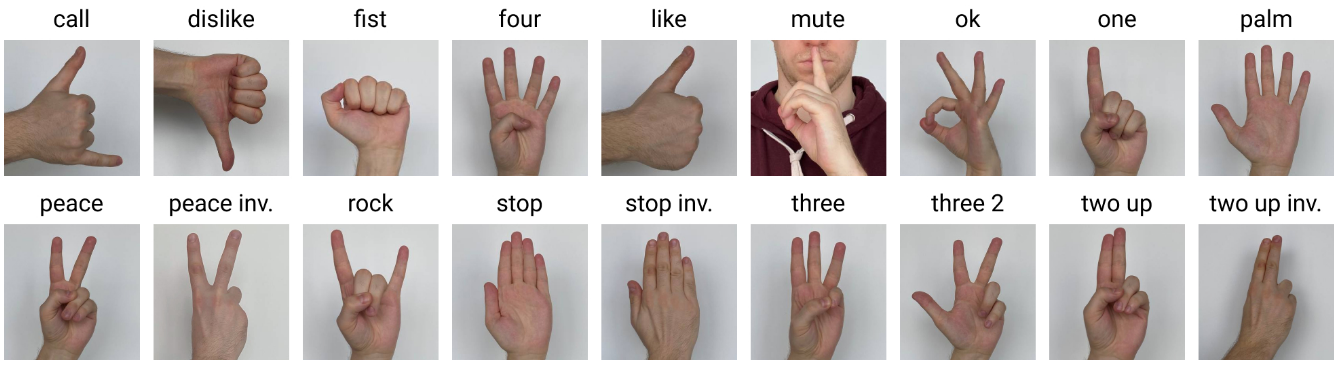

Figure 1: Gesture Classes in the HaGRID Dataset

2.1 Data Splitting Strategy

To ensure consistent model evaluation and training, the dataset was split into subsets. Training, validation and test sets were used.

- **Training Set (70%)**: This set is used to train the models themselves, recognizing patterns in the data and adjusting the model to minimize loss during the training process.
- **Validation Set (10%)**: The validation set is used to tune parameters and ensure the models do not overfit during training.

- **Test Set (20%)**: Used for the final evaluation of the model, providing an unbiased result of the models performance.

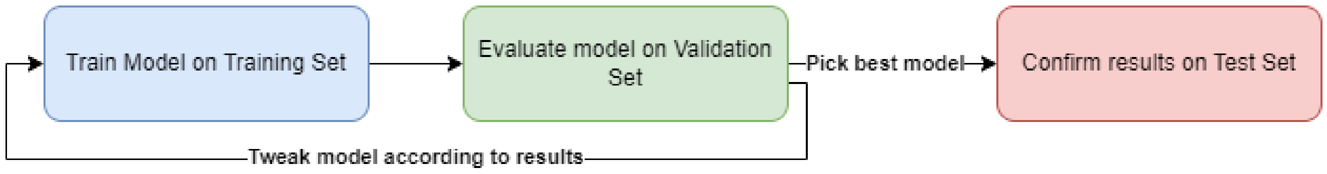
Train Model on Training Set
Evaluate model on Validation Set
Pick best model
Confirm results on Test Set
Tweak model according to results

Figure 2: Data Splitting Process

3 Data Preprocessing

3.1 Image Resizing

The original images for the dataset come in a 512x512 size. After initial testing, this was deemed far too large to train the models efficiently. Other image dimensions, such as 256x256 and 128x128 were tested, but the size of these images drastically increased the training time, and did not improve the performance of the model enough to justify this, as 128x128 images performed identically to 64x64 images. Ultimately, a uniform 64x64 image size was utilized across the dataset for every model. The models all performed relatively well on this image size and the resizing reduces the computational load and speeds the training of each model significantly.

3.2 RGB & Grayscale

The dataset was prepared for both RGB and grayscale formats, to investigate the difference in the models performance with different color information. RGB retains the original color information of the images, while grayscale converts each image to a single color channel. This means that RGB images may be able to distinguish between subtle differences in images with color, while grayscale images have less information to process so as a result are less computationally complex to process, consequently being quicker to train with less inference time.

3.3 Data Augmentation

Data augmentation techniques were used to simulate variations in gestures. These techniques include random flips, both horizontal and vertical, random rotations of up to 20 degress, and random zooms of up to 10%. This process is implemented to help the models recognize gestures from different perspectives and angles.

3.4 Utilizing the GPU

To accelerate the training process, the GPU was utilized instead of the CPU, leveraging a NVIDIA RTX 4060 graphics card. GPUs excel at performing thousands of parallel computations simultaneously, which speeds up training significantly when compared to a CPU. Parallel computation with TensorFlow and GPUs can significantly reduce image classification time, particularly for large datasets [2]. Mixed precision was enabled through TensorFlow, using the mixed_float16 setting, which enabled the models to execute computations quicker, consuming less memory and maintaining model accuracy. Data caching and prefetching was also utilized to ensure data was loaded with low latency, improving GPU utilization and training time.

4 Classifier Training

4.1 Overview of Models

Overall, there are 6 models covered in this project. Each subsequent model was based on the original from scratch CNN and base VGG-16 model, with techniques such as grayscale and data augmentation being tested for performance. The models are evaluated on the training and validation loss and accuracy, as well as the time taken per epoch to train. It's important to note that the first epoch typically takes longer to train due to the initial data loading overhead and model initialization. Each model was executed with **10 epochs**, to ensure no model has a significant advantage over another. All models incorporated overfitting prevention techniques such as dropout, early stopping, or regularization. Visualizations display accuracy and loss over epochs.

4.1.1 CNN from scratch(Color)

This model is a from scratch CNN with three convolutional layers, ReLU activation, and max pooling layers. The network ends with a fully connected layer. This model utilizes hyperparameters such as Adam for optimization, dropout to prevent overfitting, and categorical crossentropy as the loss function over the ten epoch runtime. The model achieved a **74.58%** training accuracy, a validation accuracy of **76.45%**, a training loss of **0.7703**, and a validation loss of **0.7582**. The model took on average, **72.2** seconds per epoch to complete.

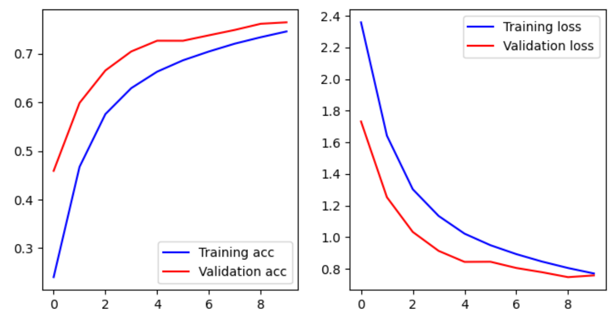

Figure 3: Base CNN accuracy & loss

4.1.2 Base VGG-16

The VGG-16 model uses transfer learning techniques, with pre-trained ImageNet weights. The last 8 layers of the model are unfrozen, meaning that they are now trainable and their weights are allowed to update during the training process. This means that the original initial layers remain frozen, based on the ImageNet weights. By fine-tuning the last 8 layers, the model is tailored more specifically to the HaGRID dataset. Several hyperparameters were used to optimize the training process. A low learning rate was used with the Adam optimizer to allow fine adjustments. Regularization and a dropout rate of 0.5 were utilized to prevent overfitting. The model performed well in terms of training times, so early stopping and plateau reductions were added, which helped avoid overfitting by stopping training early if the validation loss did not improve and reducing the learning rate when progress plateaus. The model achieved a **96.75%** training accuracy, a **87.37%** validation accuracy, a training loss of **0.4064** and a validation loss of **0.8859**. The model took approximately **175.7** seconds to complete per epoch.

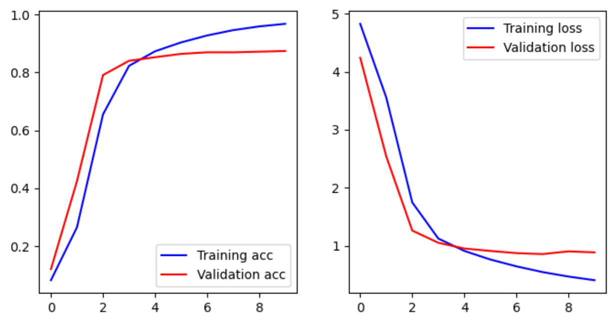

Figure 4: Base VGG-16 accuracy & loss

4.1.3 CNN from scratch(Gray)

The grayscale CNN model uses a specific grayscale-based version of the dataset, designed for a single channel input. The model is based off the original CNN from scratch, so it has the same layers with three convolutional layers with ReLU activation and max pooling, followed by fully connected layers with dropout to reduce overfitting. The hyperparameters remain the same, utilizing the Adam optimizer and categorical crossentropy. Upon completion of training, the model improved considerably, starting on an initial training accuracy of 19.32% and a validation accuracy of 37.89%, increasing up to a respectable training accuracy of **64.32%** and a validation accuracy of **65.18%** respectively. The model achieved a training loss of **1.0959** and a validation loss of **1.0787**. The model took approximately **95.3** seconds to train per epoch.

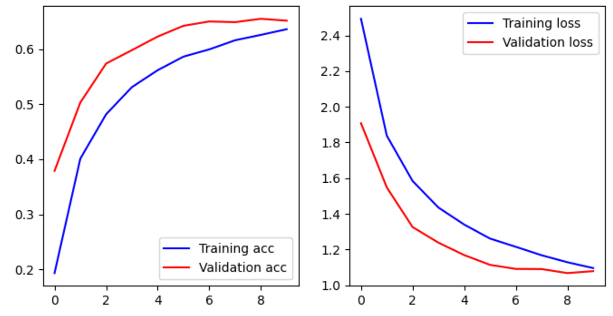

Figure 5: Grayscale CNN accuracy & loss

4.1.4 Deeper CNN

The deep CNN model was designed to be more complex, utilizing more layers overall in comparison to the base CNN model. The network starts with using a data augmentation layer that applies random flips, rotations and zooms to the images, enabling the model to better understand models at different perspectives. The model utilizes many more convolutional layers than the initial CNN, increasing in complexity as the network deepens. Each convolutional layer is followed by batch normalization, which maintains the mean output close to 0 and the standard deviation close to 1, which stabilizes the network overall. Hyperparameters such as the optimizer Adam, pooling layers and dropout layers are utilized, to prevent overfitting. The model also utilized early stopping and plateau reduction, which helped with plateaus and stopping training when required. The deeper CNN achieved a training accuracy of **71.62%**, a validation accuracy of 74.00%, a training loss of **0.8893** and a validation loss of **0.8062**. However, the model took significantly longer to train, at around **524** seconds per epoch.

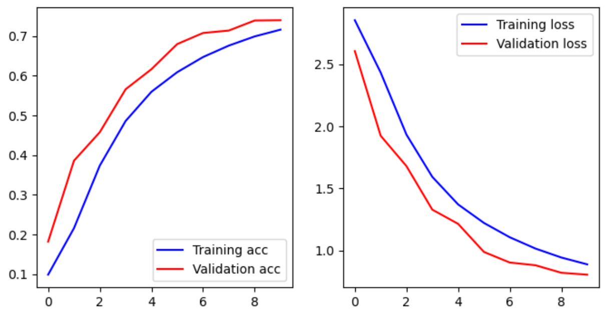

Figure 6: Deep CNN accuracy & loss

4.1.5 Data Augmented CNN

The data augmented CNN model uses random flips, rotations, and zooms to enhance its ability to generalize new images from different perspectives and angles. The network features several convolutional and max pooling layers. This model is also based on the initial CNN model, so it shares a lot of similarities. It uses the same hyperparameters such as Adam, dropout layers, and categorical crossentropy, as well as early stopping and plateau reduction. The model achieved the lowest scores out of all the models, with a training accuracy of **43.24%**, a validation accuracy of **52.64%**, a training loss of **1.7602** and a validation loss of **1.4927**. The model also took an average of **516.6** seconds per epoch for training.

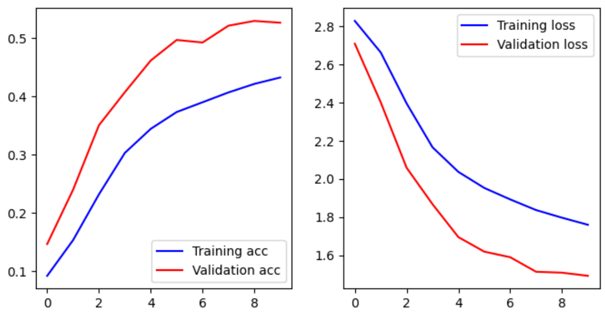

Figure 7: Data Augmentation CNN accuracy & loss

4.1.6 Data Augmented VGG-16

The model is built upon the initial VGG-16 model, implementing similar hyperparameters and layers, including the same freezing process and ImageNet weights. The data augmentation incorporates random flips, rotations and zooms to help the model understand a wider variety of images at different perspectives and angles. This model scored well, achieving a training accuracy of **81.94%**, a validation accuracy of **82.55%**, a training loss of **0.9637** and a validation loss of **0.9338**. This model took on average **619.5** seconds per epoch.

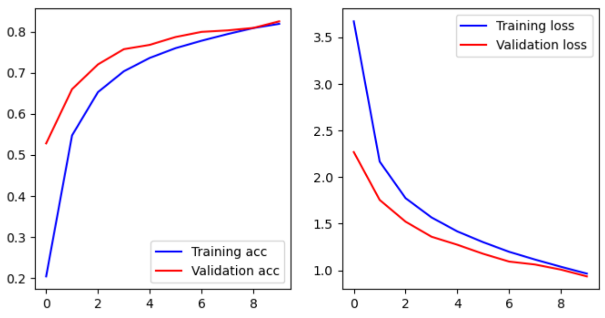

Figure 8: Data Augmented VGG-16 accuracy & loss

Table 1: Model Performance Summary

Model	Training Accuracy	Val Accuracy	Training Loss	Val Loss	Avg Time(s)
Base CNN (Color)	74.58%	76.45%	0.7703	0.7582	72.2
Base VGG-16 (Color)	96.75%	87.37%	0.4064	0.8859	175.7
CNN Scratch (Gray)	64.32%	65.18%	1.0959	1.0787	95.3
Deeper CNN	71.62%	74.00%	0.8893	0.8062	524
Augmented Scratch	43.24%	52.64%	1.7602	1.4927	516.6
Augmented VGG-16	81.94%	82.55%	0.9637	0.9338	619.5

5 Results Comparison

In this section, each model compared against eachother with the most important performance metrics. These metrics are:

- **Validation Accuracy:** Measures the model's ability to predict for unseen data. High validation accuracy suggests good generalization and reduced overfitting.
- **Validation Loss:** Indicates the model's error on the validation dataset. Low validation loss signifies efficient model predictions and better convergence during training.
- **Speed** The time required for the model to make predictions. While less important than accuracy and loss, models that perform well and don't take a significant amount of time to train are more efficient, so this should be taken under consideration.

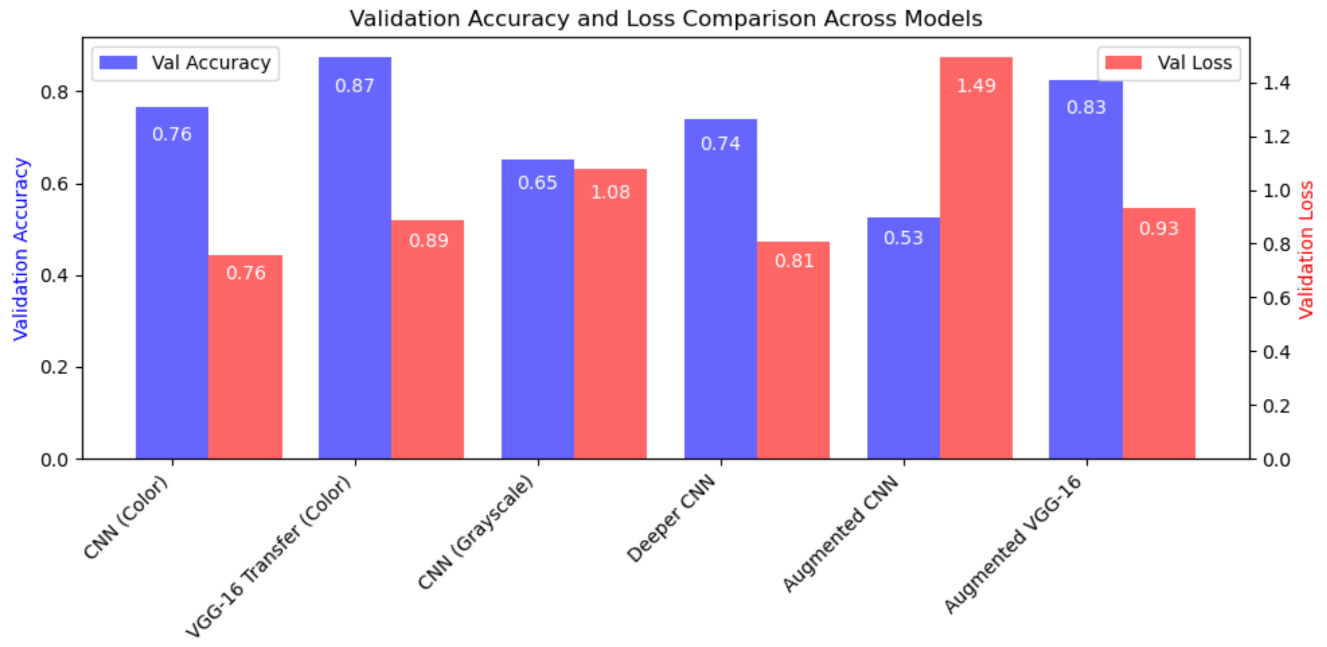

Figure 9: Validation Accuracy and Loss Comparison Across Models

5.1 Best Performing Models

The model with the highest validation accuracy was the Base VGG-16 model, scoring 87.37% overall, while the model with the lowest validation loss was the Base CNN model, scoring **0.7582**. This model also boasted the lowest average training time per epoch of **72.2** seconds, making this model the best performing from scratch model.

Despite this, after considering the results from the graphs and classifier training sections, it's clear that the **Base VGG-16 model** performs the best in comparison to all the other models. This model achieved an impressive **96.75%** training accuracy and **87.37%** validation accuracy, while also maintaining a low training and validation loss of **0.4064** and **0.8859**. The model took on average a respectable **175.7** seconds per epoch to train.

6 Final Evaluation

Now that the most efficient model has been chosen, a final evaluation can be carried out on the **Base VGG-16** model using the test set. After evaluating against the test set, the Base VGG-16 model achieved a test accuracy of **87.00%** and a test loss of **0.8876**, as well as a **16.68s** inference time showing that the model not only showcases strong performance but also handles unseen data well and operates efficiently. Now a detailed classification matrix can be constructed, and the model can be utilized to predict four custom images.

Table 2: Classification Report of the Base VGG-16 Model on the Test Set

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Call	0.95	0.91	0.93	1401
Dislike	0.97	0.96	0.97	1434
Fist	0.97	0.83	0.90	1363
Four	0.83	0.82	0.83	1397
Like	0.89	0.93	0.91	1324
Mute	0.97	0.97	0.97	1498
Ok	0.90	0.88	0.89	1365
One	0.71	0.81	0.76	1410
Palm	0.89	0.90	0.89	1391
Peace	0.76	0.79	0.77	1442
Peace Inverted	0.88	0.89	0.89	1332
Rock	0.87	0.81	0.84	1352
Stop	0.77	0.88	0.82	1395
Stop Inverted	0.91	0.90	0.90	1418
Three	0.81	0.79	0.80	1375
Three2	0.88	0.93	0.90	1381
Two Up	0.84	0.82	0.83	1473
Two Up Inverted	0.92	0.83	0.87	1433
			Accuracy	87.00%
			Macro Avg	87.00%
			Weighted Avg	87.00%

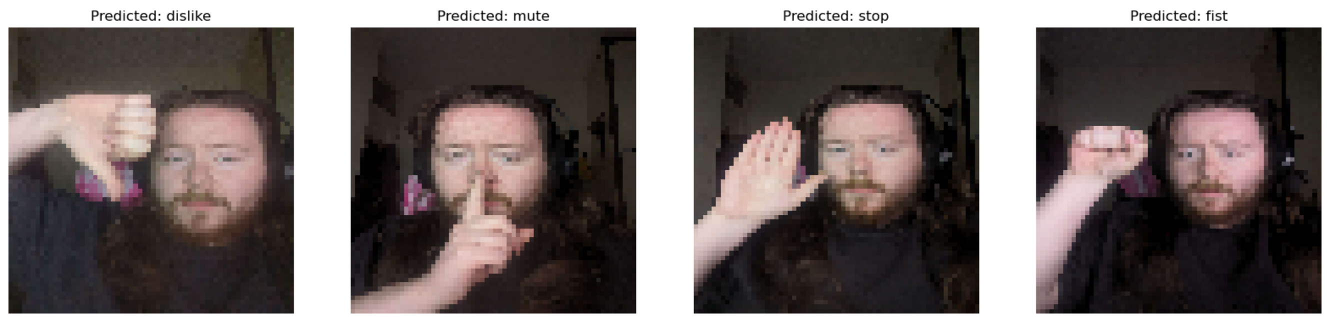

Figure 10: Example predictions by the Base VGG-16 model

7 Conclusion

The chosen model, Base VGG-16, demonstrated strong performance across various metrics, indicating its suitability for practical applications in gesture recognition. Other models performed relatively well but fell behind on training time, validation accuracy, validation loss, or a combination of all three. In conclusion, this project investigates and evaluates from scratch and transfer based learning CNN models, showcasing their efficiency and limitations in recognizing gestures.

8 References

1. Hagrid Dataset. `https://github.com/hukenovs/hagrid`
2. GPU & CPU NN Comparison. `https://digitalcommons.library.umaine.edu/etd/3524`