**EE 541 – Computational Introduction to Deep Learning**

# Final Project Report - Group 9

Christopher Conroy, Sudarshana Sudheendra Rao

December 12, 2022

**Project Title:** Deep Learning based American Sign Language Recognition

## 1 Introduction

### 1.1 Motivation

Sign language is a prominent form of communication for deaf communities across the world. However, there is a limited number of people outside of these communities who understand sign language. Therefore, there exists a need for a translation mechanism to convert sign language into a form that can be readily understood by the general public. Furthermore, the nature of the sign language used is dependent upon region and, as such, no universal sign language exists. In the United States, American Sign Language (ASL) is the most widely used form and generally uses individual hand signs to convey full words. Fingerspelling is used in cases for which a sign for a word does not exist by signing individual letters.

### 1.2 Objective

This project aims to develop a Deep Learning model to translate sign language to a text-based representation where the scope of the input is limited to the ASL alphabet used in fingerspelling. Specifically, when given a single image containing a hand sign, the model should classify the image according to the ASL alphabet with a high degree of accuracy. In particular, this project aims to investigate various Deep Learning model approaches in order to select the best model for this task and evaluate its ability to generalize to unseen ASL data.

### 1.3 Theoretical Background

A Convolutional Neural Network (CNN) is used to classify input images to the respective output class. CNN usually consists of three main neural layers namely- convolutional layer, pooling layer, and fully connected neural layer. The convolutional layer is primarily used for feature extraction from the input images. The feature extraction is done by passing a kernel or filter through the input images. Pooling layer is used to reduce the dimensions of the image while retaining all features of the image. Fully connected layer convolves the input with the weight matrices and bias vectors. Neurons in the convolutional and fully connected layers are dropped at a user-specific rate to remove lazy neurons from the network. Figure 1 illustrates the generic CNN architecture from which further architectures are derived in this work.

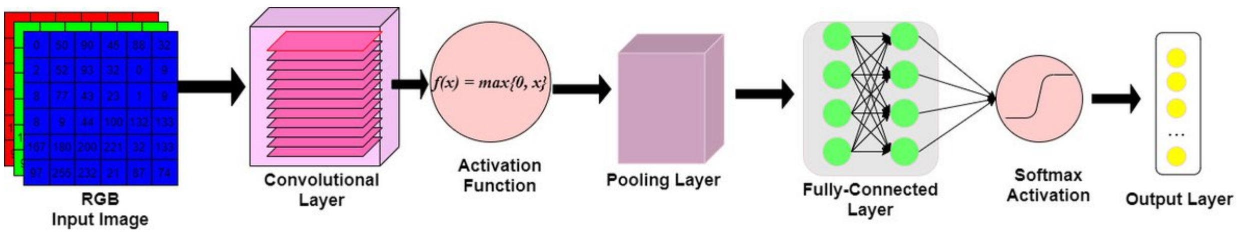


Figure 1: General Architecture of a CNN Image Classifier [3]

The loss function used by models in this work is cross-entropy loss and its equation is given by,

$$C = \sum_{i=1}^{n} y_i \ln a_i$$

The final layer of the models in this work use softmax as the activation function which is defined as

$$a = h(s) = \frac{1}{\sum_{m=1}^{M} e^{S_m}} \begin{bmatrix} e^{S_1} \\ e^{S_2} \\ . \\ . \\ . \\ e^{S_M} \end{bmatrix}$$

## 1.4 Literature Review

A number of Deep Learning models have been developed for the ASL classification task. The approaches that have seen the most success have made use of both RGB and depth data. However, there has also been work using only RGB data which has seen some degree of success. In general, the most commonly used and most successful Deep Learning model is the Convolutional Neural Network (CNN). Transfer learning was used in [1] using the GoogLeNet architecture trained on the ILSVRC2012 dataset for the same ASL alphabet as in this work, but with a different dataset. The use of transfer learning significantly reduced the time to attain convergence. On the other hand, a CNN with a simpler architecture was trained from scratch in [2] for only a subset of the ASL alphabet with some degree of success. These two papers provide a good starting point for the development of the model in this project for both a transfer learning and from scratch approach.

Convolutional Neural Networks using Deep Learning techniques are found to be robust for ASL recognition. Transfer Learning has become popular for this application where many pre-trained models like AlexNet and modified VGG16 architectures (to name a few) are employed for feature extraction and multiclass classification. Support Vector Machine (SVM) is used as a multiclass classifier in [6] to achieve better (model) recognition performance. The training dataset and validation split followed in this paper was 70-30. Also, the accuracies of different architectures were compared by plotting a bar graph. The final accuracy obtained was 99.82%.

In recent times, research has found that Adapted Deep Convolutional Neural Network (ADCNN) is suitable for ASL recognition. Furthermore, ADCNN obviates overfitting of data by using the ReLU activation function, softmax at the output layer, and L2 regularization. Data augmentation plays an important role in ASL recognition and in the paper [7], the input images are shifted horizontally and vertically by a magnitude of 20 degrees. The above data augmentation increases the number of input datasets to achieve the robustness of the ADCNN model. The ADCNN model was trained with 3750 ASL images (which include modifications such as rotating the image, changes in illumination, and deliberate addition of noise in the images) and an accuracy of 99.73% was achieved. The ADCNN's performance was better than that of a baseline CNN (which achieved an accuracy of 95.73%). It was concluded that the ADCNN's performance was 4% better than that of the baseline CNN.

An ensemble-based CNN model leverages both depth and color images of benchmarked finger-spelling datasets for accurate ASL recognition. Paper [8] used ensemble-based CNN where a pre-trained VGG-19 model with softmax activation (at the final output layer) and a multi-class SVM trained on the VGG-19 were used. The model was trained using 4096 features. The data features that had 90% of variance were fused into a single feature vector having 8192 dimensions. Principal Component Analysis (PCA) was employed to reduce the dimensionality for accurate classification. PCA reduced the dimensionality from 8192 to 820. Further, SVM was trained using the reduced features for multi-class classification. Ultimately, the model achieved an accuracy of 99.93%.

## 1.5 Dataset

The dataset (training & testing combined) is 1.03 GB. It consists of 87,209 images (training & testing combined) in jpeg format. Each image is in RGB (Red-Green-Blue) scale and its dimensions are 200 x 200 pixels. The dataset folder has two sub-directories, namely the training dataset and the testing dataset.

The training dataset contains one sub-folder for each of the 29 classes. The 29 classes are the 26 English alphabet characters ('a' through 'z'), space, delete, and nothing. There are 3000 training images for each class. The testing dataset consists of 29 images (i.e., one image for each class). Sample images from this Kaggle dataset are shown in Figure 2 and the dataset can be accessed here.

The testing dataset was further extended to contain a number of real-world images captured during the testing phase of the model. The dimensions of these real-world images were scaled down to 200 x 200 pixels to align with the Kaggle dataset. The supplementary test dataset has two directories, namely, supplementary test datasets 1 & 2 and sample images from each are shown in Figure 3 and Figure 4 respectively. The breakdown of the first supplementary test dataset (10358 images in total with a raw image folder size of 18.98 GB) is shown in Table 1. Supplementary test dataset 2 consists of 50 images in each of the 29 classes (a total of 1450 images) and is of size 2.48 GB. The supplementary dataset was captured and provided by the members of EE-541 (Fall 2022) final project group 11, namely Nicholas Adams and Gomathy Krishnan. The supplementary datasets were generated using the hands of several individuals against a blank background.

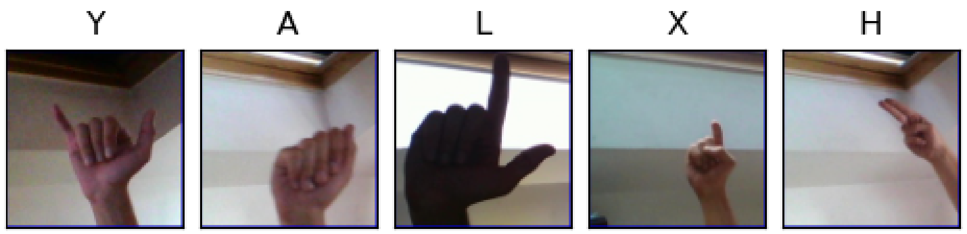


Figure 2: Sample images and labels from ASL Kaggle Dataset

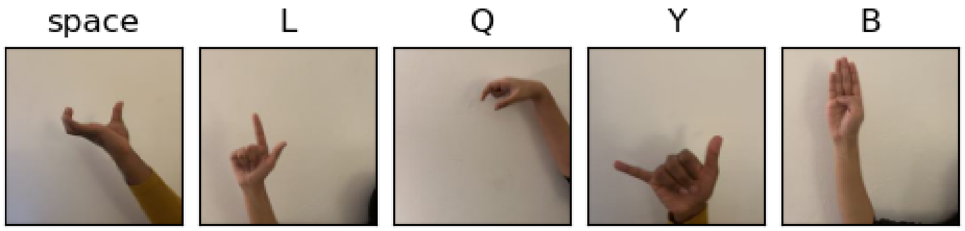


Figure 3: Sample images and labels captured in supplementary dataset 1

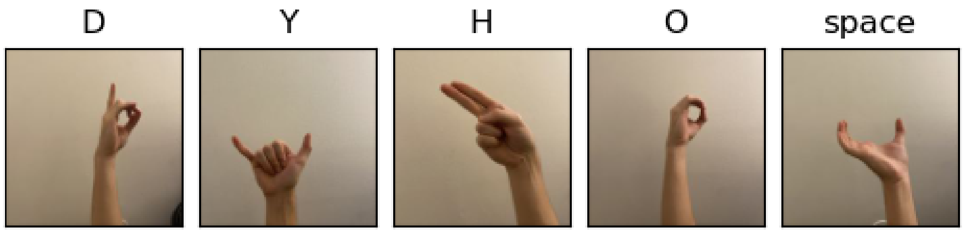


Figure 4: Sample images and labels captured in supplementary dataset 2

| **Class** | **No. images** | **Class** | **No. images** |
|---|---|---|---|
| A | 352 | NOTHING | 353 |
| B | 354 | O | 355 |
| C | 362 | P | 354 |
| D | 360 | Q | 352 |
| E | 355 | R | 355 |
| F | 351 | S | 356 |
| G | 350 | SPACE | 398 |
| H | 354 | T | 350 |
| I | 355 | U | 360 |
| J | 354 | V | w36 |
| K | 354 | W | 357 |
| L | 353 | X | 354 |
| M | 359 | Y | 354 |
| N | 353 | Z | 354 |

Table 1: Description of supplementary test dataset 1

## 2 Methodology

At a high level, this work involved the selection of 5 models, outlined in Section 2.1, which were trained and their hyperparameters tuned using only the Kaggle dataset. The performance of these models was then compared in order to identify the best model. This model was trained on the full Kaggle dataset and evaluated using the supplementary test datasets. This aimed to assess how well the best model generalizes to real-world images outside the Kaggle dataset.

### 2.1 Models

Three classes of Artificial Neural Network (ANN) architectures were selected for investigation in this work. The first class involves a single generic Multilayer Perceptron network. In contrast, the second class involves two untrained CNNs, in particular, one baseline CNN from literature and another custom CNN developed in this work. Finally, the third class involves two prominent pre-trained CNNs which are used in this work to take advantage of transfer learning. Specifically, these are the GoogLeNet and ResNet18 networks. Adam optimizer and L2 regularization were used in all of the models. This gives a total of 5 architectures that are applied and compared in this work.

#### 2.1.1 MLP

The Multi-Layer Perceptron (MLP) has three linear layers in total- two fully connected layers with ReLU activation function and one output layer. In the two fully connected layers, neurons are dropped at a 20% dropout rate. The first fully connected layer has an input size of 30,000 neurons and an output size of 1000 neurons. The second fully connected layer has 1000 input neurons and 200 output neurons. The output layer has 200 input neurons and 29 output neurons. The learning rate used was 5e-4, this MLP was trained on 50 epochs and weight decay equal to 1e-3.

#### 2.1.2 Baseline CNN

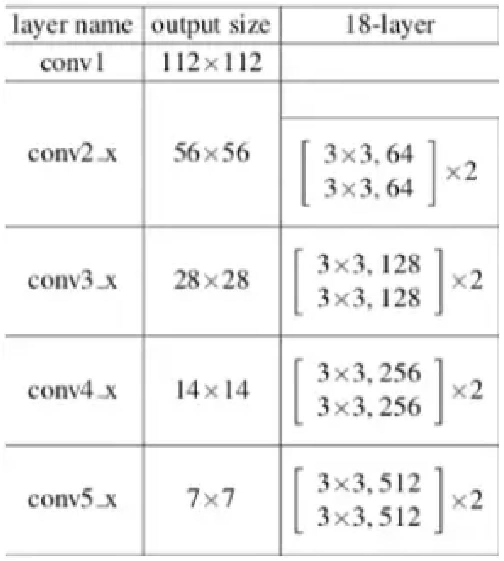
The baseline CNN architecture consists of two convolutional layers, two fully connected linear layers, and one output layer. The two convolutional layers perform max pooling operation. All four layers have ReLU activation function and 20% neuron dropout rate. The first convolutional layer has three input channels, 32 output channels, kernel size = five, and padding = two. The second convolutional layer has 32 input and output channels, kernel size = 3, and padding = 1. The first fully connected layer has 20,000 input neurons and 128 output neurons. The second fully connected layer has 128 input neurons and 64 output neurons. The final output layer has 64 input layers and 29 output neurons. The learning rate used was 5e-4, this CNN model was trained on 20 epochs, and weight decay was equal to 1e-3.

#### 2.1.3 Custom CNN

This custom-built architecture consists of three convolutional blocks, two fully connected linear layers, and one output layer. Each convolutional block performs convolution followed by batch normalization and max pooling. The two fully connected layers also perform batch normalization. All the layers utilize ReLU as the activation function and 20% neuron dropout rate. The final output layer has 128 input neurons and 29 output neurons. The learning rate used was 5e-4, this CNN model was trained on 20 epochs, and weight decay was equal to 1e-3.

#### 2.1.4 ResNet18

The custom CNN was benchmarked against two pre-trained models namely- ResNet18 and GoogLeNet. The ResNet18 is a CNN that has 18 deep layers. ResNet performs 3x3 convolution with a fixed kernel dimension (F = [64, 128, 256, 512] in the respective layers), bypassing the input after every two convolutions. Figure 5 depicts the architecture of the ResNet18 model, according to which ResNet consists of several blocks which go deeper forming a deep neural network. In every block, the number of layers (4) remains the same. conv 1 (first) layer is where convolution, batch normalization, and max pooling operations take place. In the subsequent layers, ReLU activation operation on the input takes place except in the final output layer. ResNet solves the vanishing gradients problem as the weights propagate backward to the initial filters. The ResNet18 model used in this project was trained on 10 epochs using a learning rate of 5e-4, with a weight decay of 1e-4. [4]



| layer name | output size | 18-layer |
|---|---|---|
| conv1 | 112×112 | |
| conv2_x | 56×56 | $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$ |
| conv3_x | 28×28 | $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$ |
| conv4_x | 14×14 | $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$ |
| conv5_x | 7×7 | $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$ |

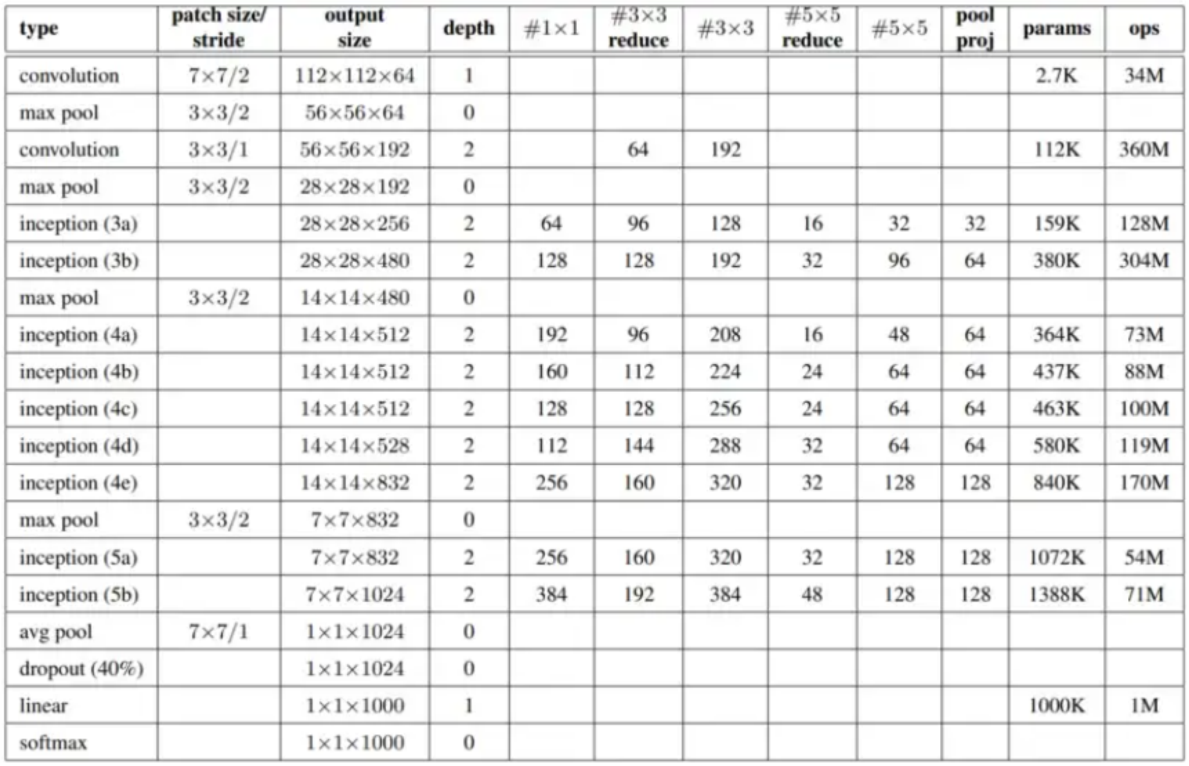
Figure 5: Architecture of ResNet [4]

#### 2.1.5 GoogLeNet

GoogLeNet is a CNN model with 27 deep layers (including 3 pooling layers and 9 inception modules). The GoogLeNet model used in this project was trained on 10 epochs using a learning rate of 5e-4, with a weight decay of 1e-4. The characteristic features of GoogLeNet are (refer to the headings of the table in Figure 6):

- Column "type" is the different layers of GoogLeNet.
- Column "patch size" is the size of the kernel (filter) used in the convolutional and pooling layers. In GoogLeNet, the height and width of the kernels have to be the same. "stride" is used to define the padding value of the kernel.
- Column "depth" refers to the number of neurons present in the respective layer.
- Columns "#1x1, #3x3, and #5x5" are the number of convolution filters used within the inception module.
- Column "pool proj" refers to the number of 1x1 filters used (within the inception module) after pooling.

- Column "params" is the total number of weights present in the respective layer.
- Column "ops" refers to the number of mathematical operations carried out in the respective layer. [5]



| type | patch size/ stride | output size | depth | #1×1 | #3×3 reduce | #3×3 | #5×5 reduce | #5×5 | pool proj | params | ops |
|---|---|---|---|---|---|---|---|---|---|---|---|
| convolution | 7×7/2 | 112×112×64 | 1 | | | | | | | 2.7K | 34M |
| max pool | 3×3/2 | 56×56×64 | 0 | | | | | | | | |
| convolution | 3×3/1 | 56×56×192 | 2 | | 64 | 192 | | | | 112K | 360M |
| max pool | 3×3/2 | 28×28×192 | 0 | | | | | | | | |
| inception (3a) | | 28×28×256 | 2 | 64 | 96 | 128 | 16 | 32 | 32 | 159K | 128M |
| inception (3b) | | 28×28×480 | 2 | 128 | 128 | 192 | 32 | 96 | 64 | 380K | 304M |
| max pool | 3×3/2 | 14×14×480 | 0 | | | | | | | | |
| inception (4a) | | 14×14×512 | 2 | 192 | 96 | 208 | 16 | 48 | 64 | 364K | 73M |
| inception (4b) | | 14×14×512 | 2 | 160 | 112 | 224 | 24 | 64 | 64 | 437K | 88M |
| inception (4c) | | 14×14×512 | 2 | 128 | 128 | 256 | 24 | 64 | 64 | 463K | 100M |
| inception (4d) | | 14×14×528 | 2 | 112 | 144 | 288 | 32 | 64 | 64 | 580K | 119M |
| inception (4e) | | 14×14×832 | 2 | 256 | 160 | 320 | 32 | 128 | 128 | 840K | 170M |
| max pool | 3×3/2 | 7×7×832 | 0 | | | | | | | | |
| inception (5a) | | 7×7×832 | 2 | 256 | 160 | 320 | 32 | 128 | 128 | 1072K | 54M |
| inception (5b) | | 7×7×1024 | 2 | 384 | 192 | 384 | 48 | 128 | 128 | 1388K | 71M |
| avg pool | 7×7/1 | 1×1×1024 | 0 | | | | | | | | |
| dropout (40%) | | 1×1×1024 | 0 | | | | | | | | |
| linear | | 1×1×1000 | 1 | | | | | | | 1000K | 1M |
| softmax | | 1×1×1000 | 0 | | | | | | | | |

Figure 6: Architecture of GoogLeNet [5]

## 2.2 Data Pre-Processing

The Kaggle dataset and the two supplementary datasets were captured from different sources and therefore available in different forms. Specifically, the Kaggle images have 200 x 200-pixel dimensions while the supplementary dataset images have 3024 x 4032-pixel dimensions. A number of image input sizes were trialed with the baseline CNN model and images with 100 x 100-pixel dimensions were found to be a good trade-off between training time and accuracy. Specifically, 100 x 100-pixel images resulted in approximately 50% reduction in training time with similar accuracy compared to 200 x 200-pixel images. Therefore, the first step in the data pre-processing pipeline is to reduce the images to 100 x 100 pixels. This corresponds to step (a) in Figure 7 below. The Kaggle images are scaled down directly while the supplementary images are square center cropped first before being scaled down. It was checked that this did not crop out any important features from the image.

All of the Kaggle dataset images are generated using a single person's hand in a single environment. Therefore, there is little variation in brightness, skin color, and contrast besides that generated from the person's hand moving between shaded and well-lit regions of the environment. In order to introduce such variation to aid with generalization, a random color jitter is applied to each image. Specifically, the brightness, contrast, saturation, and hue of each image are altered slightly by a random magnitude. This corresponds to step (b) in Figure 7.

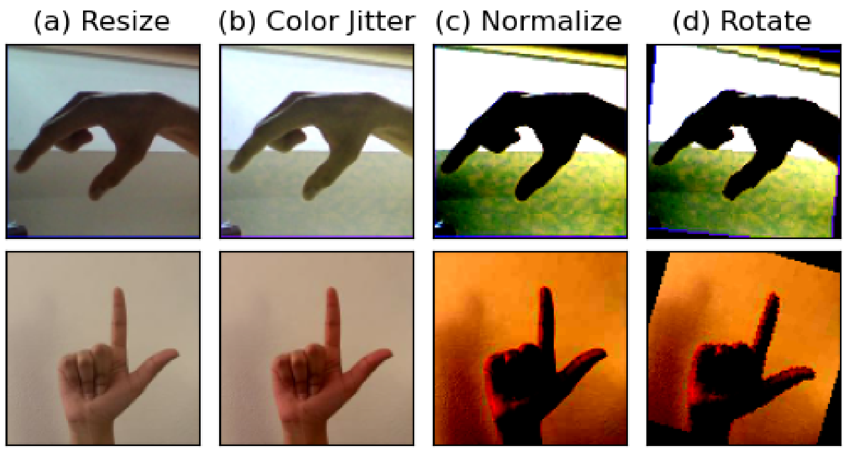


Figure 7: Data pre-processing steps applied to an image from the Kaggle ASL dataset (row 1) and an image from supplementary dataset 1 (row 2)

In order to ensure each input parameter to the network has a similar distribution, the images are normalized such that each input channel of the images has zero mean and unit variance. In order to do this, the initial distribution of the dataset was computed and used. In general, normalization of the input images was seen to decrease convergence time and improve accuracy. The result of normalization can be seen in step (c) in Figure 7. Since pixels usually have positive values, the resulting negative values from normalization are cut off at zero and contribute to greater areas of black when displayed.

The final pre-processing step involves the random rotation of the image in either direction up to a maximum angle of 20 degrees. This is performed to ensure the model learns the hand signs in a rotationally invariant manner as real-world signs are likely not to be as consistently oriented as in the Kaggle dataset. This corresponds to step (d) in Figure 7.

### 2.3 Data Handling

Since the primary objective in the first stage was to compare the performance of the 5 ANN architectures using only the Kaggle dataset, the dataset was split in order to facilitate this comparison. Due to the high degree of similarity between many images in the Kaggle dataset and a large number of images, several of the models of concern in this work were capable of attaining a high degree of accuracy with the dataset. This would make it difficult to compare models. Therefore, a training, validation, and test split of 40/30/30 was used to highlight the models with the best generalization capabilities. A random stratified split was used to ensure a balanced sample of classes was captured in each set.

Only the training and validation sets were used to train and tune the hyperparameters of each model. The test set was not used during this phase. Only once the hyperparameters had been finalized for each model and each model completely trained using the training data was the testing set used. It was used to train the final model for each architecture and select the best model for the next stage.

Once the best model was selected, the entire Kaggle dataset was combined again and used to train the model until convergence. The supplementary datasets were used as two separate test sets to evaluate the model on real-world data outside the Kaggle dataset once it had been fully trained on the entire Kaggle dataset. In addition, the final model was evaluated again (without making any changes to the model), but without the color jitter and random rotation pre-processing steps being applied to assess the contribution these steps have to the model's ability to generalize.

### 2.4 Evaluation Metrics

The following four evaluation metrics were used as the primary means of assessing the performance of each model for comparison purposes during the first experimental stage. In addition, they were also used to evaluate the performance of the final model.

1. Precision: Precision is the ratio of the number of true positives (predictions) over the total number of true positives and false positives detected by the model.

$$\frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

2. Recall: Recall is the ratio of true positives over the sum of true positives and false negatives.

$$\frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

3. F1- Score: F1-Score is the harmonic mean of the precision and recall.

$$\frac{2 * \text{Recall} * \text{Precision}}{\text{Recall} + \text{Precision}}$$

4. Accuracy: The accuracy of the model is computed by the formula given below,

$$\frac{\text{True Positive} + \text{True Negative}}{\text{True Positive} + \text{False Negative} + \text{True Negative} + \text{False Positive}}$$

In addition to these metrics, the average execution time of each epoch was also recorded to assess the time complexity of the training process for each model. Finally, a confusion matrix was used to visualize the classification behavior of the best model on both supplementary testing datasets.

## 3 Results

### 3.1 Kaggle ASL Dataset

All 5 models were trained iteratively and their hyperparameters were tuned individually using the Kaggle dataset split as described in Section 2.3. The optimal training and evaluation metrics for each model are presented and compared in this section. In general, the training for each model was halted when the training converged and before the model was over-fitted. The training loss and accuracy for the MLP model are shown in Figure 8. The model required a comparatively large number of 50 epochs to reach convergence. Additionally, the model had a large generalization error of approximately 20% to the validation set.

The training loss and accuracy for the baseline CNN model are shown in Figure 9. This model attained a better training accuracy with a lower generalization error of approximately 10% in a far shorter training cycle of only 20 epochs. The training was also more stable in general with a smoother training curve. The training loss and accuracy for the custom CNN model are shown in Figure 10. The primary difference between this model and the baseline CNN model is the depth of the network. This configuration change results in better training and validation accuracy than the baseline CNN model with a slightly lower generalization error. The custom CNN attains similar performance as the baseline CNN in only 5 epochs but had a comparatively more unstable training curve.

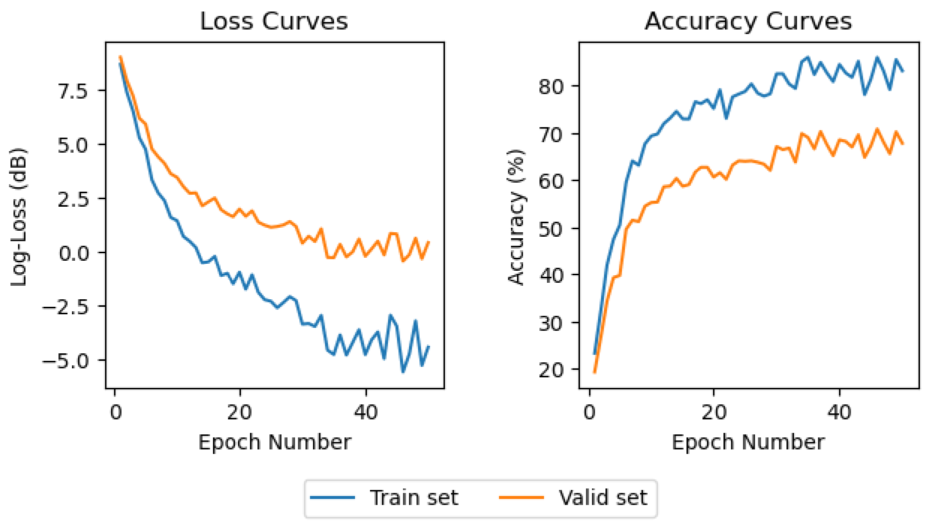


Figure 8: Training loss and accuracy curves for MLP model

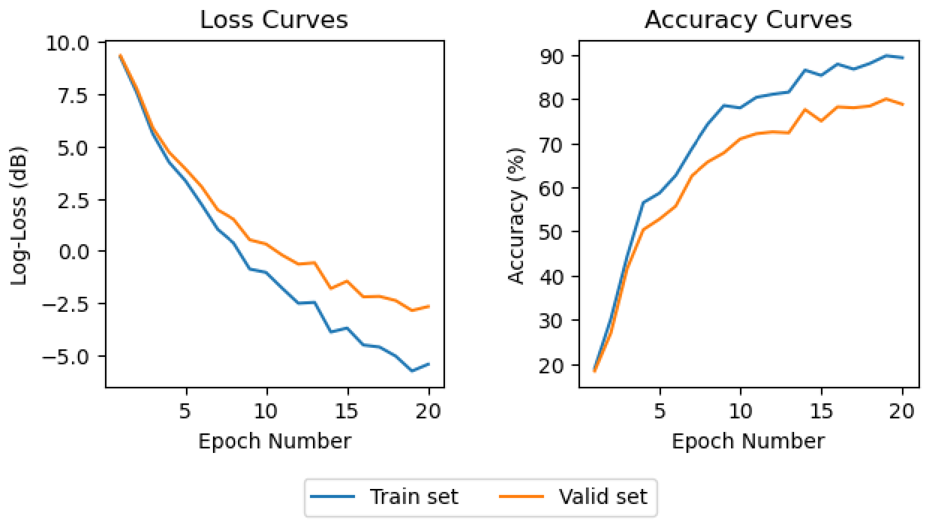


Figure 9: Training loss and accuracy curves for baseline CNN model

The training loss and accuracy attained when using transfer learning in conjunction with the GoogLeNet model is shown in Figure 11 while Figure 12 shows the same results when used in conjunction with the ResNet18 architecture. In comparison to the MLP model as well as the CNN models, the transfer learning training process exhibited by far the fastest convergence times with the GoogLeNet model already attaining over 90% validation accuracy after only a single epoch and attaining 100% training accuracy after only 3 epochs. The ResNet model training results exhibited similar rates of convergence. In addition, both transfer learning models had far lower generalization errors on the order of 1% to 2%. This is far superior to the MLP and CNN models. Despite the GoogLeNet and ResNet architectures attaining such similar training metrics, it is noted that the GoogLeNet training was far more stable than the ResNet18 training. This can be seen in Figure 12 where the training and test loss drop substantially around epoch 8 while the GoogLeNet training curve in Figure 11 is notably smoother.

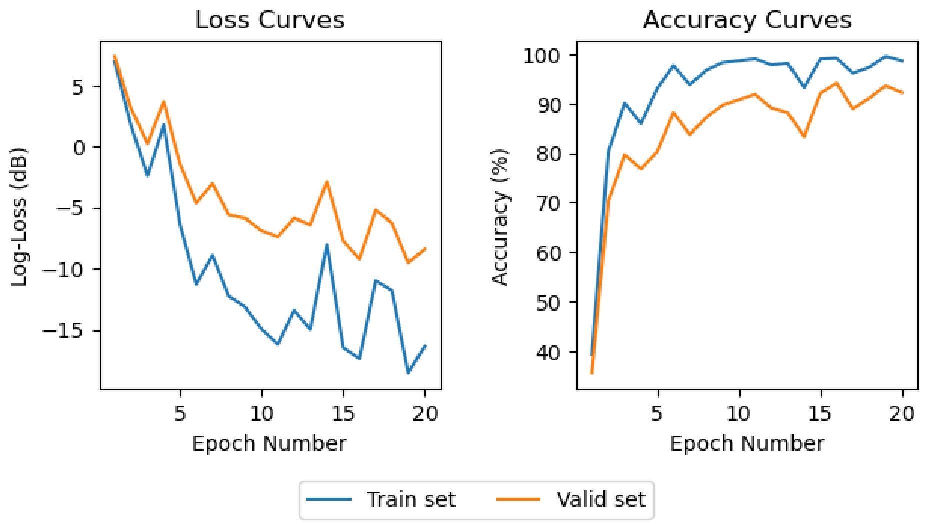


Figure 10: Training loss and accuracy curves for custom CNN model

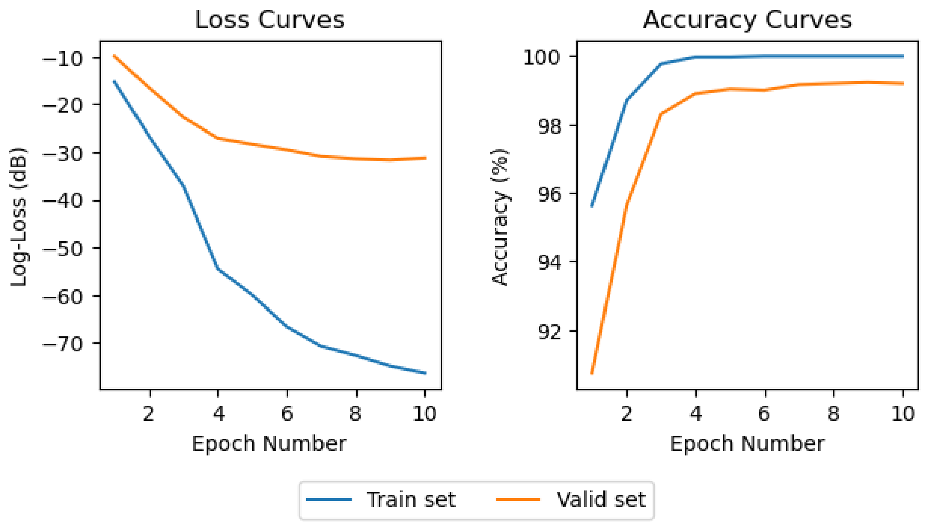


Figure 11: Training loss and accuracy curves for GoogLeNet model

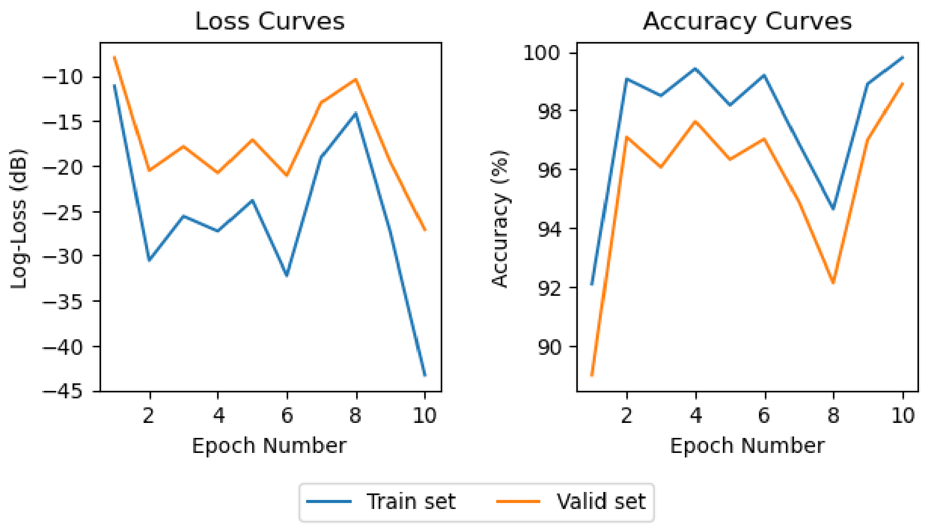


Figure 12: Training loss and accuracy curves for ResNet18 model

The suggestions regarding the performance of the models noted during the training process were confirmed when the models were evaluated using the test set. The performance metrics, including the precision, recall, F1 score, and accuracy, for each of the models on the test set, are shown in Table 2. In addition, the table also shows the number of epochs each of the models was trained for until convergence was attained as well as the average time each epoch ran for. When considering accuracy, MLP attained the lowest accuracy which indicates its architecture is not suited to this task.

The baseline CNN and custom CNN demonstrated improved accuracies in comparison of 77.93% and 85.53% respectively. However, both of these are significantly worse than the near-perfect accuracies of the transfer learning models. The GoogLeNet model attained a final test accuracy of 98.97% and the ResNet18 model attained 98.47%. In addition, these results were obtained with far fewer epochs required during training. In terms of epoch training times, all models exhibited average epoch times on the same order of magnitude. The additional metrics correspond closely to the accuracy metric and therefore do not require further discussion.

| **Model** | **Epochs** | **Epoch Time (s)** | **Precision** | **Recall** | **F1 Score** | **Accuracy (%)** |
|---|---|---|---|---|---|---|
| MLP | 50 | 24.87 | 0.6937 | 0.6618 | 0.6655 | 66.17 |
| Baseline CNN | 20 | 38.71 | 0.7934 | 0.7795 | 0.7774 | 77.93 |
| Custom CNN | 20 | 28.30 | 0.8917 | 0.8553 | 0.8657 | 85.53 |
| GoogLeNet | 10 | 33.45 | 0.9898 | 0.9897 | 0.9897 | 98.97 |
| ResNet18 | 10 | 29.27 | 0.9850 | 0.9847 | 0.9847 | 98.47 |

Table 2: Evaluation metrics for each model on ASL Kaggle test dataset

## 3.2 Extension to Supplementary Datasets

Based on the training analysis and Kaggle test set results outlined in Section 3.1, the GoogLeNet model was selected as the best-performing model. This was based on the fact that the GoogLeNet model exhibited the highest accuracy, fastest convergence, and most stable training curve. Table 3 shows the evaluation performance of the GoogLeNet model on both supplementary test datasets after being trained until convergence on the entire Kaggle dataset. Due to the far greater number of samples in the full Kaggle dataset, the model reached convergence after a single epoch, but with a substantially longer epoch period. In addition, to being evaluated on both datasets, the model was also trained with the data pre-processing step being included and excluded from the pipeline to assess the effect of the data pre-processing step on the model's ability to generalize to unseen data. This corresponds to the minimal and complete rows in Table 3 under the data pre-processing column respectively.

| Dataset | Pre-Processing | Epoch Time (s) | Precision | Recall | F1 Score | Accuracy (%) |
|---|---|---|---|---|---|---|
| Dataset 1 | Minimal | 476.14 | 0.7632 | 0.6570 | 0.6302 | 65.83 |
| | Complete | 1333.21 | 0.7895 | 0.7311 | 0.7183 | 73.25 |
| Dataset 2 | Minimal | 476.14 | 0.8466 | 0.8455 | 0.8083 | 84.28 |
| | Complete | 1333.21 | 0.9074 | 0.9090 | 0.8871 | 90.74 |

Table 3: Evaluation metrics for GoogLeNet model on supplementary test datasets 1 & 2

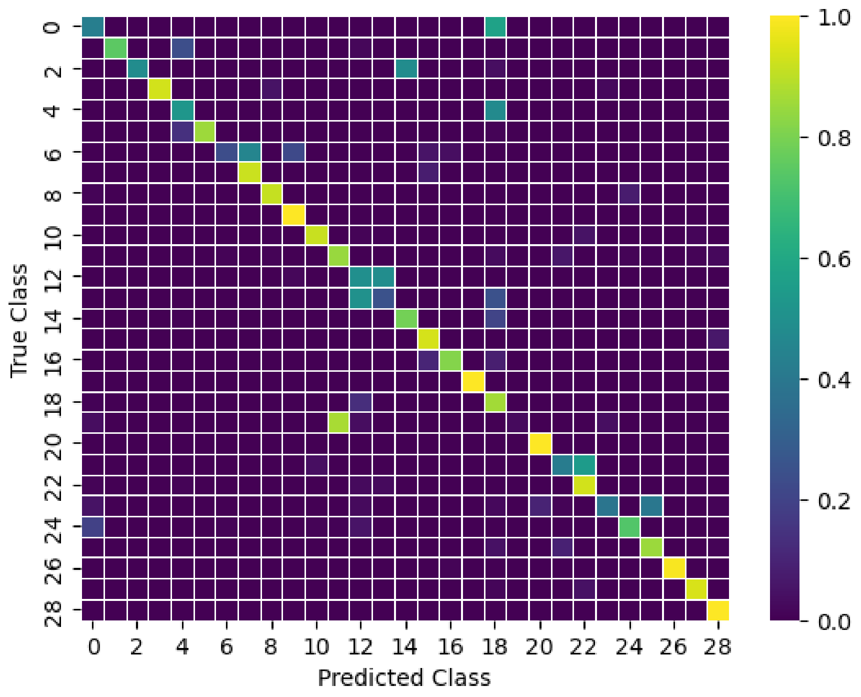


Figure 13: Confusion matrix heat map for the GoogLeNet model with pre-processing tested on supplementary dataset 1

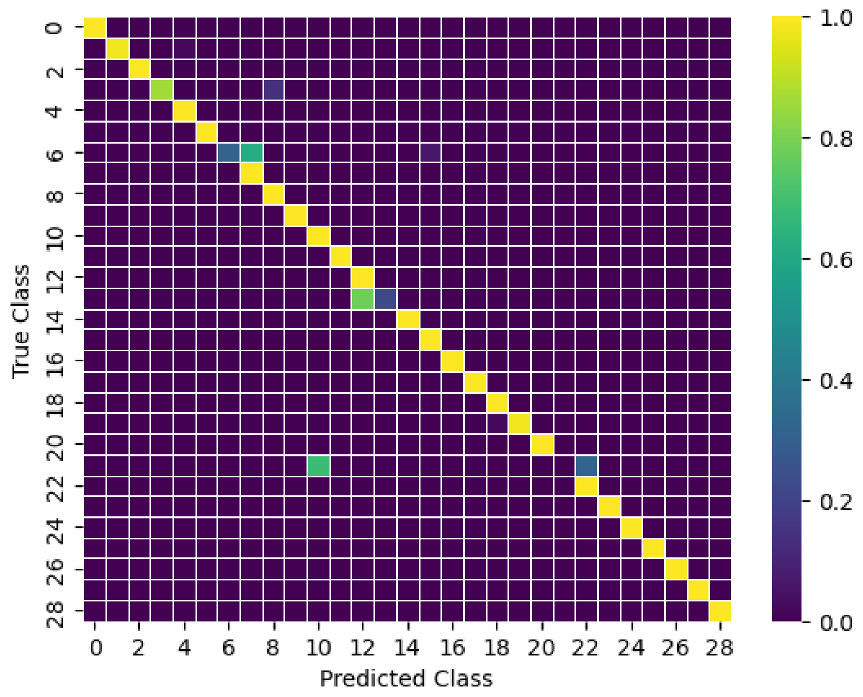


Figure 14: Confusion matrix heat map for the GoogLeNet model with pre-processing tested on supplementary dataset 2

Table 3 indicates that there was a distinct performance difference between the performance of the model on both test datasets. The model attained 73.25% accuracy on dataset 1 and 90.74% accuracy on dataset 2. In addition, the addition of the data pre-processing step is seen to improve the accuracy by approximately 7% on both datasets. This performance discrepancy between the datasets is also seen in the confusion matrix visualizations for the model classification performance. Figure 13 shows the confusion matrix normalized by true class when evaluated on supplementary dataset 1 while Figure 14 shows the equivalent for supplementary dataset 2. Figure 14 exhibits a far stronger diagonal with only a single notable confusion issue where class 21 is frequently confused for class 10. In contrast, the diagonal is far less distinct in Figure 13 with many more blocks indicating significant class confusion.

## 4 Discussion

### 4.1 General

In the first phase of the experimental phase of the project, the MLP architecture was found to have the worst performance in terms of convergence time, generalization error, and accuracy. This implies that the MLP architecture is not suited to this task. The baseline and custom CNN architecture exhibited notably better performance and generalization capability. This is expected as the convolutional structure is able to extract locally clustered features within an image far more effectively than the MLP structure. The main variable that explains the custom CNN's improved performance over the baseline CNN is the increased depth of the model which gives it a better capacity to extract useful features from the image. In addition, the CNN's also made use of regularization techniques such as L2 regularization and dropout which were both found to reduce the generalization error. Furthermore, batch normalization was also implemented and, during the hyper-parameter tuning phase, was found to increase the rate at which the training converged for the models as well as slightly improve the final validation accuracy.

Both models used with the transfer learning approaches were far superior to any of these three models in terms of the same performance metrics. This is expected as both the GoogLeNet and ResNet18 models make use of far deeper network architectures than either of the CNNs trained from scratch in this work. In addition, these models also take advantage of more complex network structures. However, the fact that the networks were pre-trained on the ImageNet dataset facilitated far faster convergence, higher accuracy, and better generalization. This is due to the fact that many objects share similar features such as edges and curves. The pre-trained kernels in these networks likely pick up these same features in the ASL dataset and therefore only the classification MLP structure of the networks need to be retrained.

As discussed in Section 3, the GoogLeNet model was selected as the best-performing model on the Kaggle dataset and was evaluated on the supplementary test datasets. The distinct performance difference between the supplementary test datasets shows the difficulty that arises when trying to assess the model's capability to generalize. The poorer performance on test dataset 1 is likely due to the greater number of people's hands being included in the dataset in comparison to only one hand being included in dataset 2. Furthermore, the skin tone of some of the hands in the first supplementary dataset differed from the Kaggle training dataset more so than the second supplementary dataset. The greater performance boost that was seen from applying the data pre-processing step with the first dataset than the second dataset is likely due to the variation in skin tones the pre-processing step introduced to the training data. In general, the use of pre-processing to diversify the training set in terms of contrast, color, saturation, and rotation significantly improved the model's ability to generalize. In addition, the use of normalization improved the training convergence time of all of the models.

### 4.2 Challenges

There were a number of engineering challenges that were encountered in this project. The first challenge was to evaluate the five models investigated in this work in a manner that facilitated a clear comparison between the models. Due to the great degree of similarity between the images in the Kaggle dataset, preliminary testing showed that the use of a large number of samples yielded a very high validation accuracy for both the CNN models and the pre-trained models. This made it difficult to compare the models as the performance metrics were so similar. In addition, the large training dataset size greatly increased the computational time complexity to train and tune the models to an infeasible degree given the timeline of this project. The solution to both of these constraints involved using only 10 000 samples of the 87 000 samples. In addition, a large validation and test set split was employed to highlight the model differences in generalization capability. This strategy worked well and a clear distinction was able to be drawn between the models within a reasonable time frame.

The use of a single environment and a single person's hand in the Kaggle dataset presented a substantial challenge in ensuring that the model would generalize to a real-world dataset with hands of different people in different environments and with different lighting conditions. The use of data pre-processing and augmentation helped to address these issues. In particular, the use of color jitter improved the ability of the model to generalize to different skin tones and different lighting conditions while the rotation augmentation made the model more robust to pose differences.

Another challenge encountered was the differences that arose between the training implementations for each model as each model was fine-tuned. Specifically, when a general change to the training for all models needed to be made, all the training implementations for each model needed to be updated separately, introducing the potential for inconsistencies in comparison to being introduced. This, combined with multiple individuals contributing to the code base presented logistical issues. This was solved through the appropriate use of version control and refactoring the code to be highly modular such that only a minimal amount of additional code was required to train and test each model.

## 5 Conclusion

This work was successfully able to identify the best model to use given a set of models with different architectures for the given ASL dataset as the GoogLeNet model. The use of transfer learning with existing complex architectures significantly improved the rate of convergence and accuracy of the model. When trained using the entire dataset, the model was reasonably successfully able to generalize to the real-world test datasets captured in addition to the Kaggle ASL dataset. This is impressive considering the Kaggle ASL dataset only consisted of a single hand in a single environment. The use of data pre-processing had a notable impact on the model's ability to generalize to unseen test datasets.

### 5.1 Future Work

This work should be extended to include the use of the person's left hand in both the training and testing dataset since all images in this work only included the person's right hand. People will usually sign using their preferred hand so this extension is necessary for real-world applicability. In addition, this dataset made use of only static images mapped to a single label while in reality, some of the signs require the use of movement to fully determine their meaning. The removal of this simplification should be explored. In the same vein, signs are rarely used in isolation, but rather in sequences used to create words. The use of models with some form of memory, such as Recurrent Neural Networks (RNNs) should be explored for this purpose and for application to real-time video feeds. Another possible extension is the use of segmentation in order to augment the training images with different backgrounds to improve the generalization of the model.

### 5.2 Questions Answered

This work answered a number of questions for the authors, both anticipated and unanticipated. The first pertains to what is the best model to use for ASL sign recognition generalization when given a very limited training set in terms of diversity. This work showed that the use of transfer learning models was best for this application due to the applicability of their convolutional feature detectors to general items of interest in images. Secondly, the work answered the question regarding the effect of various pre-processing when applied to the ASL dataset for classification. It showed the benefit of color jitter and rotation augmentation for generalization for a low-diversity dataset. Finally, answered the question of how well a model would be able to perform on an unseen real-world ASL dataset. This work showed the GoogLeNet model has the capability to perform relatively well by attaining 90% accuracy on the second test dataset.

### 5.3 Post-Project Curiosities

Finally, in completing this project, a number of curiosities arose for the authors. Firstly, all the future work suggestions discussed under Section 5.1 are avenues that the authors are interested in understanding more deeply. In addition, the authors would like to understand how freezing the convolutional layers during training would affect the pre-trained models' performance during extended training. This is due to the fact that the layers were not frozen in this work and extended training occasionally reduced the model's generalization capability. Finally, the authors are curious as to how the addition of a far greater range of classes would affect the performance of the model since there are numerous additional ASL signs in addition to the letter signs explored in this work.

## References

[1] Brandon Garcia and Sigberto Alarcon Viesca, "Real-time American Sign Language Recognition with Convolutional Neural Networks," Convolutional Neural Networks for Visual Recognition, 2, pp. 225-232.

[2] Vivek Bheda and Dianna Radpour, "Using Deep Convolutional Networks for Gesture Recognition in American Sign Language," 2017 arXiv, doi: 10.48550/ARXIV.1710.06836.

[3] James McDermott, "Convolutional Neural Networks — Image Classification w. Keras," Learning Data Science webpage- CNN (accessed December 2, 2022).

[4] Pablo Ruiz, "Understanding and visualizing ResNets," Towards Data Science webpage- ResNet (accessed December 2, 2022).

[5] Richmond Alake, "Deep Learning: GoogLeNet Explained," Towards Data Science webpage- GoogLeNet (accessed December 3, 2022).

[6] Abul Abbas Barbhuiya, Ram Kumar Karsh and Rahul Jain, "CNN based feature extraction and classification for sign language," Multimed Tools Appl 80, 2021, pp. 3051–3069, doi: 10.1007/s11042-020-09829-y.

[7] A. A. Alani, G. Cosma, A. Taherkhani and T. M. McGinnity, "Hand gesture recognition using an adapted convolutional neural network with data augmentation," 4th International Conference on Information Management (ICIM), 2018, pp. 5-12, doi: 10.1109/INFOMAN.2018.8392660.

[8] R. G. Rajan and P. S. Rajendran, "Gesture recognition of RGB-D and RGB static images using ensemble-based CNN architecture," 5th International Conference on Intelligent Computing and Control Systems (ICICCS), 2021, pp. 1579-1584, doi: 10.1109/ICICCS51141.2021.9432163.

## Appendix A - Timeline, Milestones and Contributions

The following list is a timeline showing the project milestones and author contributions. All dates are with regard to the year 2022 and the authors are denoted by their surname.

- Nov 23 - Start of work on the project - (Conroy, Rao).
- Nov 29 - Completion of data loader implementation for Kaggle Dataset - (Conroy).
- Nov 29 - Completion of supplementary dataset acquisition and processing - (Conroy).
- Dec 02 - Completion of model implementations - (Conroy, Rao).
- Dec 03 - Research and analysis of different pre-trained models - (Rao).
- Dec 04 - Model comparisons on Kaggle dataset and selection of best model - (Conroy).
- Dec 04 - Performed Transfer Learning using GoogLeNet and ResNet models - (Rao).
- Dec 05 - Evaluation of GoogLeNet model on test datasets - (Conroy).
- Dec 10 - Completion of the initial draft of the final report - (Conroy, Rao).
- Dec 11 - Final review of report and finalization of code repository - (Conroy, Rao).

## Appendix B - Project Repository

The code repository is on GitHub and can be accessed at https://github.com/coderconroy/ee541-final-project.