

Classifying American Sign Language Alphabet

Kelechi Nnebedum, Atmika Sarukkai, Michelle Cheung

The goal of this project is to create a model that can recognize the ASL alphabet letter. This model will be trained to identify the alphabet letter when given an image of a hand signing a letter. This project will use the Sign Language MNIST dataset from Kaggle.

Problem Statement

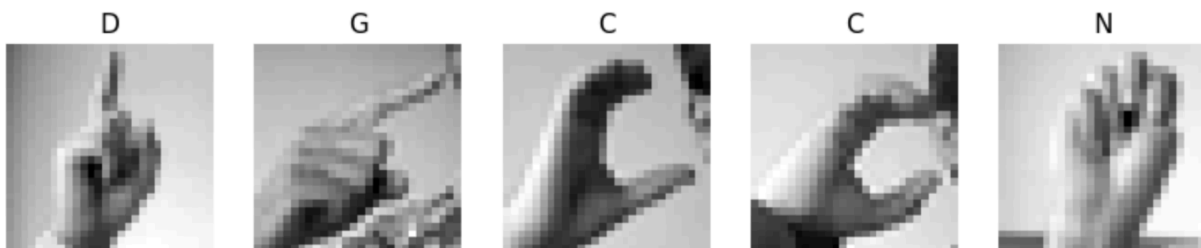
Communication and socialization is a key part of the human experience. Being able to communicate with others is needed in order to form relationships, transfer information, and connect to others. However, learning new languages is not something that is possible or accessible to everyone. While ASL is much more complicated than just stringing together letters of the alphabet, creating a model that can accurately classify the ASL alphabet is a great first step to creating a much larger model that can understand more ASL and allow for more people to communicate with those that use ASL.

Objective

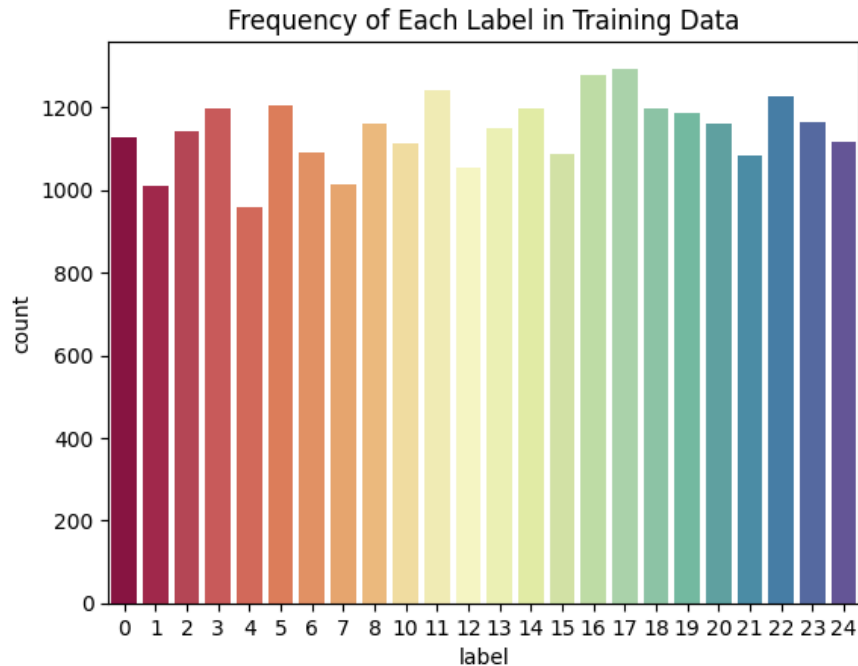
The objective of this project is to build a machine learning model that can accurately classify the 26 letters of the ASL alphabet. This model can be built on at a later point in time to recognize other words in ASL. But at this current moment, the model can be used as a starting point to make communication easier between those who use ASL and those who are not familiar with it.

Approach/ Methodology

1. **Data Collection:** Download the Sign Language MNIST dataset from Kaggle into a notebook for data preprocessing and model building.



2. **Exploratory Data Analysis:** Conduct exploratory data analysis on the dataset to better understand the images in the data set and the distribution of the data labels.

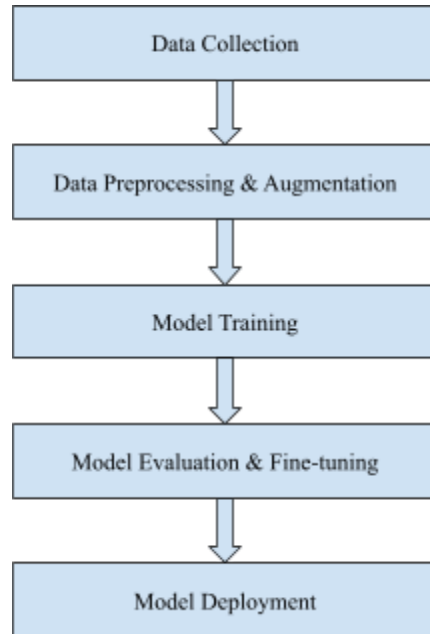


This barplot was created to see the distribution of classes within the dataset. This is integral to know before augmenting the data and training the model so that we can ensure there is balance between classes. Not all classes are balanced, although the largest difference is no more than 200 images. Given that each label has over 1000 images, we were not concerned about an imbalance of data leading to biased results.

3. **Data Preprocessing & Augmentation:** We reshape, normalize, and binarize the data to optimize the performance of the model and reduce loss. We generate more data with data augmentation to prevent overfitting. We follow the steps below to preprocess and augment the data:
 - 3.1. Download the dataset
 - 3.2. Reshape the data to size (28, 28, 1)
 - 3.3. Normalize the data by dividing all pixel values by 255
 - 3.4. Augment the training dataset
 - 3.4.1. Adjusting brightness by random factor between [-1, 1]
 - 3.4.2. Adjust contrast by random factor between [0.1, 10]
 - 3.4.3. Applied a random left or right flip
 - 3.5. Concatenate original training dataset with augmented data
4. **Model Development:** We implement several machine learning and deep learning models. The models that we implement are multiclass logistic regression, random forest classification, neural network, residual neural network (ResNet), convolutional neural network (CNN), recurrent neural network (RNN), region-based convolutional neural network (RCNN), SqueezeNet, LeNet-5, and Efficient Net. Models are fit on the training data that are preprocessed and augmented in the earlier step.
5. **Model Evaluation:** We evaluate the performance of the model, specifically accuracy, on the test set by displaying a confusion matrix. For brevity, we only discuss the models that met

our success criteria in the discussion section. For more details on the results of the other models, please refer to the appendix.

Block Diagram



Dataset

<https://www.kaggle.com/datasets/datamunge/sign-language-mnist/data>

This dataset follows a CSV format containing labels and pixel values for each row instance. The labels correspond to American Sign Language letter hand gestures for 24 classes of letters. The original images, before being converted to pixels for the CSV, can be seen in [this repository](#) in the “Dataset.zip” file. This data set was extended through inputting the images into an image pipeline that created 50+ variations of each image. Each instance in the CSV represents a 28x28 pixel image with grayscale values between 0-255. The training data has a total of 27,455 instances, and the test data contains 7,172 instances. We will be utilizing these datasets for model creation, validation, and testing.

What is considered success/failure?

We choose a test accuracy of 90% that we must reach to have a successful model. We would need to ensure that the classes in the dataset are balanced, and we would want to ensure that each letter reaches the 90% accuracy threshold, as we want our model to be able to perform equally well on all letters. Since letter identification is crucial to communicate with others, it’s essential that our model is able to accurately differentiate between different letters so that nothing gets lost in translation.

Evaluation Parameters

To measure the success of our model, we will consider accuracy to understand the ratio of correctly classified letters. We will create a confusion matrix to further analyze the false positive, false negative, true positive, and true negative rates for each letter class. Utilizing the confusion matrix, we will be able to assess where problems are arising in the model by understanding the precision and recall for each class.

We plan to assess these model performance metrics on a variety of classification models with various hyperparameter tuning and finally identify the best performing model.

Experiments

Multiclass Linear Regression

The first model that we implemented was a multiclass linear regression model. This model is a sequential model that consists of a flattening layer and a dense layer. We used the stochastic gradient descent optimizer with a learning rate of 0.01. When fitting the model we ran 5 epochs with a batch size of 64 and used a validation split of 0.1.

Random Forest

The random forest classifier defines a grid of hyperparameters including the number of estimators [50,100], and the maximum tree depth [5,10]. The model utilizes grid search to find the optimal combination of hyperparameters to maximize model accuracy. The most optimal parameters are a max_depth of 10, and 100 estimators. The model was fit on the training data utilizing the best parameters obtained.

Convolutional Neural Network

The CNN model is a sequential model. The first layer is a 2D convolutional layer, then a 2D max pooling layer, and a dropout layer with a rate of 0.25. Then we added another 2D convolutional layer, another 2D max pooling layer, and another dropout layer with a rate of 0.25. Finally, we added a flattening layer, two dense layers and a final dropout layer. This model used an Adam optimizer and was trained with 5 epochs.

LeNet-5

The LeNet-5 model is also a sequential model. The first layer is a 2D convolutional layer with 6 filters and a kernel size of (5,5). This layer is followed by a 2D average pooling layer. Next, we add another 2D convolutional layer with 16 filters and a kernel size of (5,5). Then we add another 2D average pooling layer.

After the second average pooling layer, we add a flattening layer and our first Dense layer with 120 units. We then add 2 more dense layers, with 84 and 25 units, respectively. This model is then compiled using the Adam optimizer.

EfficientNet

EfficientNet is a type of convolutional neural network. The first layer is a 3x3 convolutional layer and the head layer is a 1x1 convolution layer. In between these two layers are 7 Inverted Residual Blocks, which are also called MBConv blocks.

Simple Neural Network/ Multilayer Perceptron

The simple neural network or multilayer perceptron has one hidden layer with the size of [256, 128] using the activation ReLU, optimizer Adam, and learning rate of 0.01.

RNN

The RNN model we implemented is a simple neural network for classification, which uses a simple RNN layer (64 units) followed by a Dense layer with softmax activation, with an Adam optimizer.

RCNN

This model leverages the image processing power of a CNN and the sequence prediction power of a RNN. This model includes two convolutional layers with ReLU activation and max pooling, a reshaping layer to input into the LSTM layer, and two recurrent (LSTM) layers. The model is then followed by a Dense layer with softmax activation and compiled with an Adam optimizer. We performed hyperparameter tuning, which involved testing different activation functions (ReLU, sigmoid, tanh) on the convolution layers and experimenting with the number of convolution layers. We found that using 'relu' on each convolution layer and using two convolution layers in the model performed well while still maintaining computational efficiency.

SqueezeNet

This model is a type of CNN that is known for keeping a low parameter count and having efficient computational performance. It first has an initial convolution layer with 64 filters of size 3x3 that helps reduce spatial dimensions and extract initial features. It then feeds this into fire modules that consist of convolution layers that "squeeze" and "expand" the features to reduce parameters. The model architecture includes dropout (0.5 rate), L2 regularization, with the occasional 3x3 max pooling layer for dimensionality reduction. The model ends with a Global Average Pooling, an output layer with a softmax activation, and is compiled with an Adam optimizer and 0.01 learning rate.

Our first iteration of this model performed fairly well, but to increase the accuracy and generalizability, we added a learning rate scheduler and performed further data augmentation on the data. A learning rate scheduler adjusts the learning rate based on the epoch number. This allows for more tailored adjustment to the weights while the model runs. Further data augmentation was also applied to the data before feeding the data into the model to further assist in combating overfitting.

ResNet

The ResNet model takes in input images with shape (28,28,1). The model starts with a convolutional layer with 64 filters followed by batch normalization and activation, and max pooling is applied to reduce the spatial dimensions. These convolutional and pooling layers are repeated three times, forming the initial feature extraction layers.

Two residual plots consist of two convolutional layers with batch normalization and activation functions, followed by an operation to combine the input with the output of the convolutional layers. The residual blocks are repeated 9 times (3 blocks with 64 filters each, 3 blocks with 128 filters each, and 3 blocks with 256 filters) to make the model more complex, which would therefore allow it to capture more nuanced features.

After the final residual block, a global average pooling layer is applied to reduce the spatial dimensions to (1, 1, 256). Then a dense layer with 25 units and a softmax activation function is applied for classifying the images into the 25 classes.

Results

The results from our experiments are summarized in the table below.

Table 1: Model Results Ordered by Test Accuracy

Model	Test Accuracy (%)	Max Validation Accuracy (%)
RCNN	95.11	99.84
SqueezeNet	94.88	95.10
Convolutional Neural Network	94.23	99.93
ResNet	91.48	97.92
LeNet-5	84.38	100
EfficientNet B0	79.43	98.78
Random Forest Classifier	73.61	97.13
Recurrent Neural Network	63.32	63.31
Multiclass Logistic Regression	49.78	50.88
Neural Network	46.26	61.72

Tests/ Graphs/ Discussions

As mentioned earlier, we will only go into depth about the results of the models that met our success criteria, obtaining a test set accuracy of 90% or above. The models that managed to meet this criteria were the convolutional neural network (CNN), ResNet, recurrent neural network (RNN), and SqueezeNet.

Accuracy Score

To identify the algorithms that met our success criteria, we examined the testing accuracy for each of the models. See Table 1 in the results section to see the accuracy of each of these models.

The figure displays four confusion matrices arranged in a 2x2 grid, comparing predicted labels (X-axis) against true labels (Y-axis) for four different models: ResNet, SqueezeNet, CNN, and RNN. Each matrix shows counts for 26 classes (A-Z) on both axes. The diagonal elements, representing correct classifications, are significantly higher than the off-diagonal elements, indicating high model accuracy. The RNN model shows slightly higher counts on the diagonal compared to the others.

Top Row: ResNet and SqueezeNet

- ResNet:** The matrix shows high counts along the diagonal, with some off-diagonal counts visible, particularly for classes like 'A' and 'B'.
- SqueezeNet:** Similar to ResNet, SqueezeNet shows high accuracy with a strong diagonal, though with slightly more off-diagonal counts in some regions.

Bottom Row: CNN and RNN

- CNN:** The matrix shows high counts along the diagonal, with some off-diagonal counts visible, particularly for classes like 'A' and 'B'.
- RNN:** The matrix shows high counts along the diagonal, with some off-diagonal counts visible, particularly for classes like 'A' and 'B'.

Discussion of Model Choices

	loss float64 0.45947703719139...	accuracy float64 0.8969222307205...	val_loss float64 0.3465967178344...	val_accuracy float64 0.9340490698814...	lr float32 9.9999988378840...
0	0.5212143064	0.8974685669	0.3968010247	0.9426938295	0.000099999902
1	0.5165258646	0.8968222307	0.390583545	0.944645822	0.000099999902
2	0.5142750144	0.897031486	0.4021840096	0.9340490699	0.000099999902
3	0.5029852986	0.8997450471	0.3881708384	0.939347446	0.000099999902
4	0.4991381764	0.90183784	0.3792445064	0.9443669915	0.000099999902
5	0.4971146584	0.9020943642	0.3756307065	0.9443669915	0.000099999902
6	0.4890582561	0.9024585485	0.376829654	0.9418572187	0.000099999902
7	0.4869216383	0.9018029571	0.3610725701	0.9493864775	0.000099999902
8	0.4860278368	0.9018029571	0.35838449	0.9509202242	0.000099999902
9	0.481167419	0.9041886926	0.3720012009	0.9347462058	0.000099999902

	loss float64 0.45947703719139...	accuracy float64 0.8969222307205...	val_loss float64 0.3465967178344...	val_accuracy float64 0.9340490698814...	lr float32 9.9999988378840...
10	0.4755012095	0.9053177834	0.3555725515	0.948410511	0.000099999998
11	0.4747172296	0.9060280323	0.3515081406	0.9510596991	0.000099999998
12	0.4686306417	0.9071753621	0.3493291736	0.952732861	0.000099999998
13	0.4673527777	0.9080131054	0.3548802435	0.950083673	0.000099999998
14	0.470995307	0.904571116	0.3511996865	0.9498047829	0.000099999998
15	0.4664799571	0.9081041813	0.3538643718	0.948410511	0.000099999998
16	0.4655533135	0.9074667692	0.3558245897	0.9470161796	0.000099999998
17	0.474165231	0.9042615294	0.3475342691	0.9535694122	0.000099999998
18	0.4625905454	0.9067746997	0.3557461202	0.9460401535	0.000099999998
19	0.4703852832	0.9073210955	0.3519485295	0.9477131355	0.000099999998

	loss float64 0.45947703719139...	accuracy float64 0.8969222307205...	val_loss float64 0.3465967178344...	val_accuracy float64 0.9340490698814...	lr float32 9.9999988378840...
20	0.4660232663	0.9084320068	0.3530189097	0.9475739002	9.999998838e-7
21	0.4647898376	0.9082134366	0.3525566757	0.9482710361	9.999998838e-7
22	0.470707953	0.9046257734	0.3465967178	0.9507808089	9.999998838e-7
23	0.4628165066	0.9077399373	0.3500147462	0.9499441981	9.999998838e-7
24	0.4664349258	0.908614099	0.3502299488	0.9503625035	9.999998838e-7
25	0.4698184133	0.9066836834	0.349355534	0.950083673	9.999998838e-7
26	0.4594770372	0.909306109	0.3488986357	0.9495259523	9.999998838e-7
27	0.4606997967	0.9079220295	0.3487765789	0.9507808089	9.999998838e-7
28	0.4624199271	0.9087961912	0.3496713638	0.9502230883	9.999998838e-7
29	0.4684564471	0.907248199	0.3525931537	0.9488287568	9.999998838e-7

The architecture of a CNN is built to handle image classification really well. The SqueezeNet model is a type of convolutional neural network (CNN) that is designed to keep a low parameter count while also having efficient computational performance. It achieves this through using "fire modules" that utilize "squeeze" layers to reduce input channels and a "expand" layer that has 1x1 and 3x3 convolution filters. This architecture combined with pooling layers to reduce overfitting lead to a high accuracy model that can run on less computational power, making it a highly efficient and accurate model. Adding a learning rate scheduler and more data augmentation also further

improved the model. The SqueezeNet model performed the second best out of all of our models, with a test accuracy of 94.88%.

Convolutional Neural Network

	loss float64	accuracy float64	val_loss float64	val_accuracy float64
0	1.198838949	0.6154920459	0.2262177467	0.9471862912
1	0.3893961012	0.8667921424	0.08663012832	0.982334733
2	0.2442774326	0.9161456227	0.0308318194	0.9969040155
3	0.1820463985	0.9381816983	0.01341621764	0.9981788397
4	0.1425946355	0.9509298205	0.01021732576	0.999271512

Convolutional Neural Networks, or CNNs, are often employed in image classification and speech recognition models. This is due to the fact that its convolution layer is great at reducing the dimensionality of the input without losing any of the important information within it. The pooling layer further reduces the dimensionality of the images and retains the important features. They work by learning the spatial hierarchy of images and capture the most important features in its early layers and more complex features in the later layers. The CNN performed the third best out of all of our models, with a testing accuracy of 94.23%.

RCNN

	loss float64	accuracy float64	val_loss float64	val_accuracy float64
0	1.538775682	0.5024181008	0.3031855524	0.898743391
1	0.2978386879	0.9051579237	0.05034258217	0.9867055416
2	0.1194592118	0.9652158022	0.01265888009	0.9972682595
3	0.07531820983	0.9790971279	0.008891502395	0.9969040155
4	0.05445238203	0.985106945	0.006496924907	0.9983609319

Recurrent Convolutional Neural Networks (RCNN) integrate both convolutional layers and recurrent (LSTM) layers together to process both image and sequence prediction effectively. The convolutional layers are able to extract image features, capturing its spatial hierarchies, while still preventing overfitting by harnessing max pooling and dropout. It then utilizes LSTM layers to capture temporal dependencies and relationships in features that were extracted by the convolutional layers. The RCNN model performed the best out of all of our models, with a test accuracy of 95.11%.

Residual Neural Network (ResNet)

	loss float64	accuracy float64	val_loss float64	val_accuracy float64
0	0.3093271554	0.8996944427	0.1194068938	0.9601165652
1	0.04344162345	0.9867257476	0.1184969693	0.9602986574
2	0.03491003066	0.9899836183	0.05542080104	0.9821526408
3	0.02570730075	0.9921892285	0.008979537524	0.9972682595
4	0.01917227358	0.9947793484	0.07197802514	0.9792387486

The next algorithm that passed our success criteria was the ResNet model. ResNet, short for Residual Network, is a specialized convolutional neural network (CNN) designed for image data, capable of accommodating deep architectures with hundreds to thousands of layers. A key feature of ResNet is its use of residual connections, allowing information to bypass layers directly, preserving insights from earlier stages and addressing issues like vanishing gradients in deep networks. This model also implements residual learning, where it predicts the discrepancy between desired and actual outputs, simplifying training, particularly in deep networks. The ResNet architecture includes one input layer followed by a convolutional layer with 64 filters, batch normalization, and activation. Then three residual blocks with 64 filters are stacked. Then a residual block with 128 filters and stride 2 is added, followed by two more residual blocks with 128 filters each. Another set of residual blocks follows with 256 filters and stride 2, followed by two more blocks with 256 filters each. The model ends with global average pooling and a dense layer with softmax activation for classification. The model is compiled with the Adam optimizer and sparse categorical cross-entropy loss function. The model is fit on the training data for 5 epochs with a batch size of 32. Our model results in a training accuracy of 99.48%, a validation accuracy of 97.92%, and a test accuracy of 91.48%, suggesting minimal overfitting. From the confusion matrix, we observe the model is good at classifying most letters, but struggles distinguishing certain letter pairs such as U and G, and E and M.

Constraints

There were several constraints that we faced that could have affected the results of our experiments, such as:

Time Constraints

Given the fact that this project needed to be completed within the semester, one of the main constraints that we faced was time. As the project began a few weeks into the semester, the amount of time to work on the project was limited, meaning we did not have the opportunity to dive deeper into hyperparameter tuning for many of the models, and did not have time to further optimize accuracy of the models that we did implement.

Budget Constraints

This project was done using free resources that we compiled, such as the tensorflow and keras libraries to build our models, and the deepnote platform to host and run our experiments. When using free resources, we are limited by computational power. So although we were able to

complete the project using free resources, we were unable to utilize other libraries or larger datasets due to a lack of power.

Machine Constraints

Training deep learning models can sometimes be computationally heavy work and require lots of computing power. Given the budget constraints that we faced, we were also limited to what types of machine and computing power we were able to access as well, meaning that we may not have always had the power necessary to experiment further with the models.

Data Constraints

MNIST datasets are usually regarded quite highly due to cleanliness of data and ease of use when training models. While this dataset does not differ from other MNIST datasets in that regard, a few things that we noticed was the fact that the labels are not completely balanced in the training dataset. The difference in labels wasn't enough to cause much concern, but this does bring up possible concerns about bias within the model. Not only that, but the images were also all containing fair-skinned hands, which leads to possible concerns about the accuracy of the model on hands that are not fair skinned. However, this was not an issue we were able to address given the amount of time we had to work with the data.

Standards

This experiment was done in the coding environment within the Deepnote platform using Python 3.9. This platform gave us access to one of their basic machines for free which contained 2 vCPUs with 5 GB of memory. In order to manipulate and augment our data, we utilized functions within the Numpy and Pandas libraries. For model creation, training, and deployment, we used Tensorflow and Keras.

Comparison

When comparing the different models and their performance, we see that the RCNN model performed the best. Although the CNN had the highest Validation Accuracy (0.9987), the RCNN had a higher training accuracy (0.9851) and had the highest test accuracy (0.9511), while still having a high validation accuracy as well (0.9984). This shows that the RCNN has strong learning power on the training data but is still able to generalize well in unseen data. Overall, the SqueezeNet, CNN, RCNN, ResNet, LeNet, EfficientNet models all performed well because of their architectures, which are well built to handle image classification. The models that performed poorly include the Simple Neural Network, Multiclass Logistic Regression, and the Random Forest model. This poor performance is due to the lack of depth in these models and lack of feature extraction capabilities. Factors that influenced the performance of the models include data augmentation and how advanced a model's architecture was.

Limitations of the Study

- **Limited Dataset:** The image data consisted of grayscale images of ASL from 10 people's hands which were all of a lighter skin tone. The original images can be seen in [this repository](#) in the "Dataset.zip" file. Although there was data augmentation to create thousands of variations, this data may not be able to capture real-life variations such as signing styles, skin tones, and hand sizes. Therefore, the model might not generalize well to all users.
- **Missing Letters:** The data set is missing the letters J and Z since these signs require motion. Our model currently only predicts still images. Therefore, we cannot conclusively state that this classification study classifies the entirety of the alphabet.
- **Limited Real-life Application:** If this classifier were to be applied to real-life tools, we would also have to take into account facial expressions and body movements, as this provides crucial context to people signing and communicating in ASL. Furthermore, the model would be most useful if it expanded to the rest of the ASL signs and wasn't just limited to the alphabet. These would be important factors to take into consideration if we were to scale this model to an applicable tool.

AI Fairness

In building machine learning models to classify ASL alphabet letters, it is important that questions of AI fairness are considered to ensure the most equitable outcomes and to avoid perpetuating biases.

One such bias that we must be aware of is the present of diverse and representative data of the ASL-speaking population. The over-presence of certain demographics in the dataset could lead to inaccurate predictions for underrepresented groups with specific physical characteristics such as skin tone or hand shape, which could lead to unequal access to communication tools and resources.

Another potential AI fairness issue we may observe within the data is related to different dialects within ASL speakers. Certain signs, handshapes, and gestures may differ depending on geographic region, cultural influences, and community norms. Issues might arise if the model is biased towards certain select dialects that are represented within the dataset, and may not be representative of some minority dialects. In order to address this issue, it is important that diverse dialectal representations are present in the training data, that include non-standard signs and promote cultural inclusivity.

It is also important to note that the labels assigned to hand signals are in English for the purposes of this project. English-centric labeling may reflect linguistic bias and unintentionally promote English-speaking perspectives over other languages and communication forms, as well as marginalize ASL users of other languages by prioritizing English norms and conventions. Additionally, assigning English labels may not capture the semantic and cultural nuances of ASL signs, which could lead to mismatched between labeled categories and underlying meanings of the signs– this could result in reduced accuracy of the classification model and could create possible miscommunications between users. ASL is its own language with its own cultural heritage, and assigning English labels without considering context might undermine efforts to promote linguistic diversity and inclusion, which directly goes against the intent of this project.

Future Work

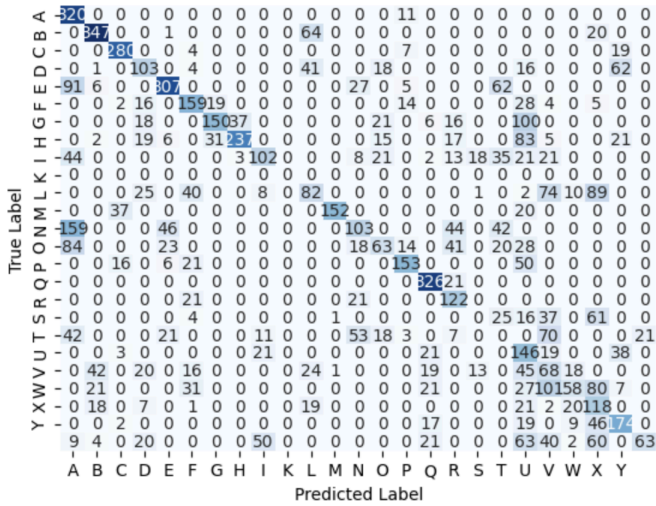
Diversifying our data is crucial to enhance the robustness of our ASL classification model. By incorporating data from different racial and ethnic groups, skin tones, and other physical features, as well as including variations in pixelation, image quality, and background scenery, we can move towards generalizing more effectively to real-world ASL communication environments and use cases. In addition to English labels, we can also consider multilingual labeling to accommodate users with diverse linguistic backgrounds.

We would also like to explore transfer learning techniques to leverage pre-trained models from related tasks (such as CNNs for image recognition) to improve the performance of this ASL classification model. Transfer learning may allow us to extract higher-level features from ASL images, like hand shapes, movements, and orientations, which would be very beneficial for image recognition.

A natural follow-up step to this project would be to delve into video data of ASL movements to extract deeper meaning beyond alphabet recognition. Being able to parse ASL sentences and phrases, investigate emotion and expression, and incorporate contextual information (such as body language and environmental cues) would significantly facilitate communication accessibility for deaf individuals in both face-to-face and digital settings.

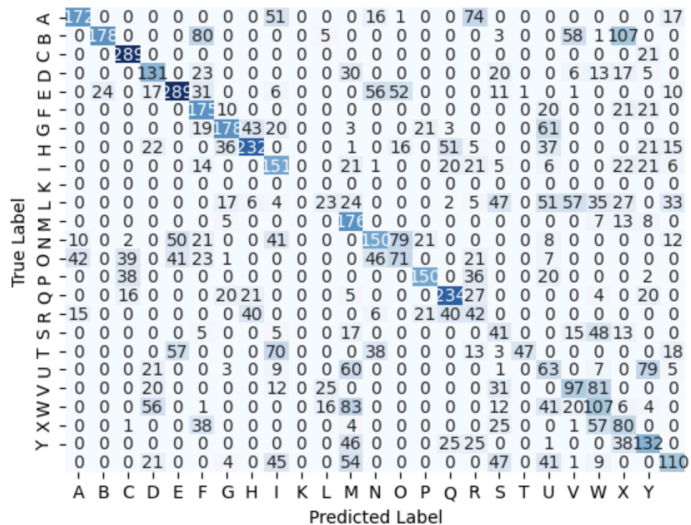
Appendix 1. Results From Multiclass Linear Regression Model

	loss float64	accuracy float64	val_loss float64	val_accuracy float64
0	2.946982622	0.1730508506	2.719741821	0.2423966527
1	2.550702572	0.3603067696	2.440551043	0.3562192619
2	2.317315578	0.4329711199	2.255244017	0.4507375658
3	2.155164957	0.4729962051	2.122723818	0.4305226803
4	2.03328681	0.5006374121	2.008379936	0.4973593056



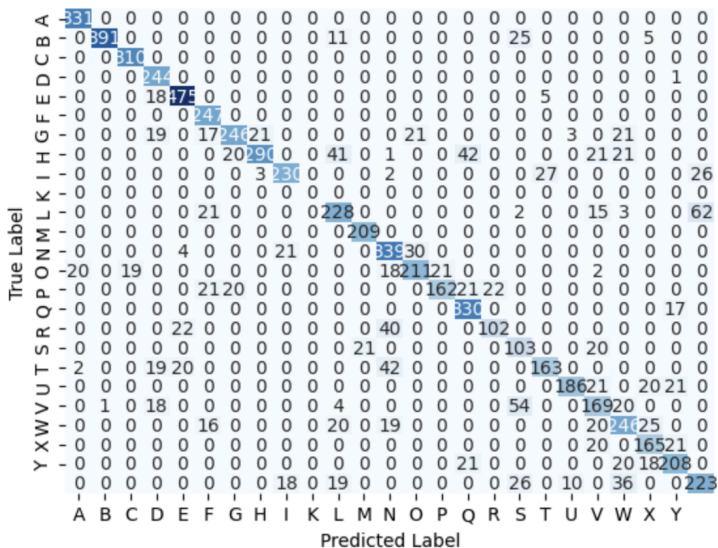
Appendix 2. Results From Neural Network

	loss float64	accuracy float64	val_loss float64	val_accuracy float64
0	2.38376832	0.2324207276	2.121890306	0.3043161631
1	1.446652174	0.4938383996	1.280969143	0.5412493348
2	1.088040709	0.6173132062	1.227478981	0.5678382516
3	0.9298267961	0.6756510735	1.008665562	0.6665452719
4	0.810726285	0.7169712186	1.138550282	0.6171917915



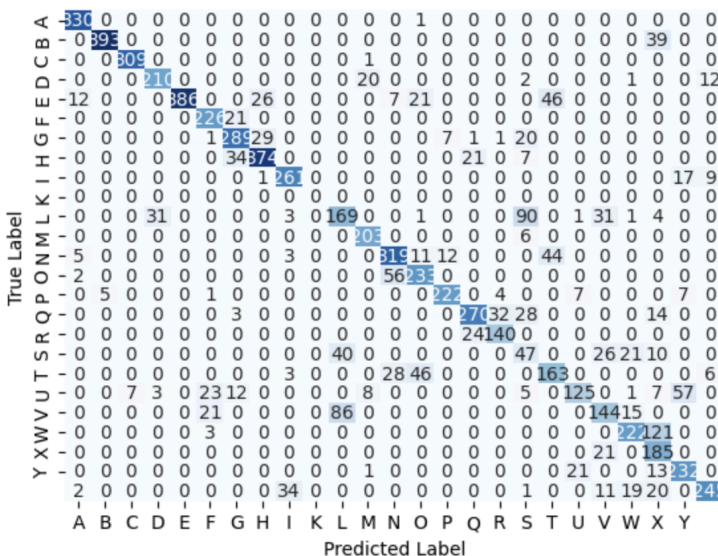
Appendix 3. Results From LeNet-5

	loss float64	accuracy float64	val_loss float64	val_accuracy float64
0	1.002983451	0.7171735764	0.2972971797	0.93990165
1	0.1195730194	0.983528614	0.04362556338	0.9969040155
2	0.02356160618	0.9985026121	0.01084137056	0.9998179078
3	0.005496049765	0.9998785853	0.003148798132	1
4	0.02688568272	0.9928367734	0.002883891342	1











Appendix 4. Results From EfficientNet Model

	loss float64	accuracy float64	val_loss float64	val_accuracy float64
0	2.381538391	0.3853983283	1.829865217	0.4170460701
1	0.7773170471	0.8040834665	1.136731267	0.8240757585
2	0.5969566703	0.8593051434	0.5280750394	0.8657803535
3	0.5220549107	0.8846597672	0.1738059223	0.9635767341
4	0.2585118115	0.9439486861	0.06014273688	0.987798214



Appendix 5. Results From Recurrent Neural Network

	<div>loss float64 0.28554797172546...</div> 	<div>accuracy float64 0.3051538765430...</div> 	<div>val_loss float64 1.3642778396606...</div> 	<div>val_accuracy float64 0.40184047818183...</div> 		<div>loss float64 0.28554797172546...</div> 	<div>accuracy float64 0.3051538765430...</div> 	<div>val_loss float64 1.3642778396606...</div> 	<div>val_accuracy float64 0.40184047818183...</div> 
0	2.204225302	0.3051538765	1.881172538	0.4018404782	10	0.518843174	0.8327627182	1.610295057	0.5708310008
1	1.570219994	0.4866508842	1.693029642	0.4545454681	11	0.4820474982	0.8449645042	1.543605804	0.6093140244
2	1.279777527	0.5783463717	1.585332751	0.5085052848	12	0.4379216433	0.8608632088	1.431074739	0.6367819309
3	1.084288478	0.6433436275	1.507423043	0.5333240628	13	0.4284664392	0.8652886748	1.545774579	0.6257668734
4	0.9495595694	0.6863230467	1.441354871	0.5474065542	14	0.3643342853	0.8845019341	1.526710153	0.6081985235
5	0.8396298885	0.7246038914	1.36427784	0.5900725126	15	0.3641566038	0.8860863447	1.456827641	0.6331567168
6	0.7514741421	0.753360033	1.37530148	0.5985777974	16	0.3395892382	0.8958477378	1.534705877	0.6440323591
7	0.6742414236	0.7813513279	1.405787587	0.6008086801	17	0.3118443191	0.904225111	1.75644505	0.595510304
8	0.6084111929	0.8026406765	1.376930475	0.6088957191	18	0.2855479717	0.9125114083	1.684548259	0.6289737821
9	0.5682661533	0.8151156306	1.370934248	0.6097322702	19	0.2923069	0.911655426	1.701410294	0.6331567168

True Label	A B C D E F G H I K L M N O P Q R S T U V W X Y																									
	310	0	0	0	0	0	0	0	0	0	0	0	4	17	0	0	0	0	0	0	0	0	0	0	0	0
A	0	336	0	0	10	1	0	1	0	0	5	0	0	0	0	0	6	0	0	0	0	0	71	0	0	0
B	0	0	309	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
C	0	0	0	152	0	0	0	0	0	0	26	14	0	0	0	0	2	0	2	3	25	2	19	0	0	0
D	-29	2	0	0	370	0	0	0	0	0	0	0	33	21	5	0	4	0	34	0	0	0	0	0	0	0
E	0	0	0	3	0	216	0	0	0	0	2	0	0	0	17	0	0	1	0	4	1	0	0	1	0	0
F	0	1	0	19	0	331	584	2	0	0	0	0	0	1	0	0	8	4	0	70	1	11	0	0	0	0
G	0	16	0	1	0	0	33	321	5	0	1	1	21	1	0	0	4	20	11	1	0	0	0	0	0	0
H	0	2	0	0	0	0	0	13	182	0	0	0	0	0	21	0	13	4	2	2	0	16	6	21	0	6
I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
K	0	0	0	1	0	34	7	22	0	0	61	0	0	0	0	20	0	101	0	2	18	26	21	0	18	0
L	0	1	3	0	0	0	0	0	0	0	0	177	0	0	0	1	0	0	0	2	0	0	0	24	1	0
M	9	1	0	0	30	0	0	0	2	0	1	0	22	57	0	18	0	52	0	0	0	0	0	0	0	0
N	-56	0	0	8	5	1	2	0	1	0	0	0	13	171	6	0	23	0	5	0	0	0	0	0	0	0
O	3	0	0	0	0	26	22	0	0	0	0	16	221	35	0	17	0	1	0	0	0	0	1	3	0	0
P	0	0	0	0	0	0	6	0	23	0	0	0	0	0	0	0	27	16	2	11	0	0	2	2	0	8
Q	0	4	0	0	12	0	0	0	0	0	0	0	0	1	0	0	0	116	0	31	0	0	0	0	0	0
R	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	120	0	0	18	0	0	0	0	0
S	5	0	0	0	41	0	0	0	21	0	0	36	1	0	0	0	0	1	139	0	0	0	0	0	2	0
T	0	0	0	0	0	4	21	5	24	0	0	0	0	0	0	8	11	8	0	85	0	5	26	47	4	0
U	0	0	0	42	0	0	0	0	0	0	18	0	0	0	0	0	0	98	14	0	67	27	0	0	0	0
V	0	0	0	0	28	0	2	1	0	26	12	0	0	0	0	0	0	26	0	0	2	169	60	0	20	0
W	0	3	0	0	0	0	0	0	0	0	4	0	0	0	0	4	0	26	0	0	19	14	136	0	0	0
X	0	0	0	20	0	0	0	0	0	0	30	0	0	0	0	33	7	0	0	21	0	1	2	142	11	0
Y	0	0	0	21	0	0	13	0	2	0	13	0	0	0	0	0	1	62	21	6	0	14	15	0	164	0