December 14, 2021

American Sign Language Letter Detection Data Analysis and Preparation

Cody Hawkins
CS5300
University of Missouri-St. Louis

Contents

1 Abstract

Object detection is a widely used form of computer vision with use cases such as finding defects in metal at factories, crop detection and analysis, car and pedestrian detection in self driving vehicles. Some of the notable models in the object detection domain are YOLO, VGG16, VGG19, Faster R-CNN. Convolutional neural networks are the driving force behind most notable models that are able to disseminate information rapidly and find the patterns in images to classify objects. To find the objects the models utilize localization, segmentation, anchor boxes, and single pass localization. The fastest models break the images in to grids of small, medium and large sizes and detect objects within the regions. The bounding areas in the regions are then classified by an accuracy score to only return the highest in the area. In this paper I have created a convolutional neural network that is able to detect American sign language letters and locally detect the hand position in an image. The focus of the task is to make understanding sign language easy for those that don't know the language by recording an ASL alphabet hand gesture and providing an alphabet letter. This is will allow for better communication in the non-signing communities but also foster learning throughout the process.

2 Introduction

2.1 What is American Sign Language?

Many people think of learning a new language as learning a foreign language such as Spanish, Cantonese or Chinese, but how many people think of sign language? American Sign Language (ASL) is a complete, natural language that has the same linguistic properties as spoken languages, with grammar that differs from English.[2]

2.2 Signing along with the computer

Learning sign language can be difficult, but what if you could have an application to help you? The idea behind this project is to teach a convolutional neural network to learn all 26 letters of ASL along with hand detection to help someone determine if they are signing the correct letter.

2.3 Motivation

Technology is a readily available tool that allows many people to learn subjects of interest online without having to travel to a class to learn the subject. With the current circumstances surrounding the world, providing a safe way to teach someone the American Sign Language alphabet sounds like an interesting endeavor.

3 Dataset

The dataset used for this project was gathered with the use of Opencv and my computer camera at 30 frames per second at five second intervals. Images were gathered both indoors and outdoors, standing and seated and just images of hands. More than one person was recorded in the dataset to help and some additional image features. In total there are 8,569 (240 x 240 x 1) gray scale images along with x, y, w and h bounding box points where the hand is located in the image. There are 26 classes of binary values of 0 and 1 for each image. The following inputs in the ASL CSV file are

1. Image file name (.JPEG)
2. X point for bounding box
3. Y point for bounding box
4. W point for bounding box
5. H point for bounding box
6. Letter class of image

4 Visualization of data distributions

below is a selection of images representing the range of pixels in the dataset along with a histogram for each data point used

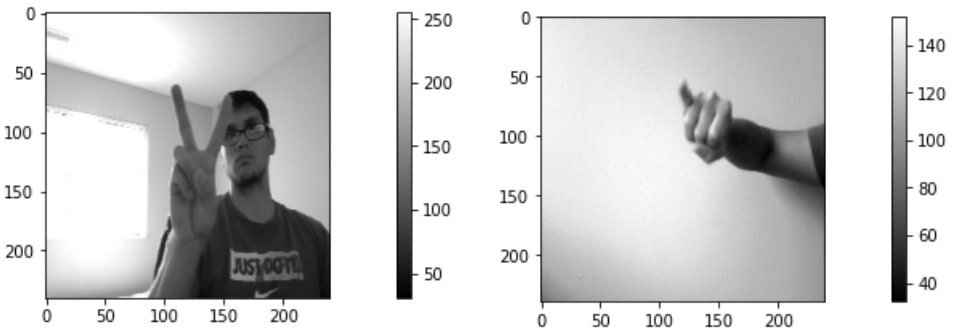

Fig 1. Colormaps of random selections of images showing uint8 images: left letter V and right letter J respectively

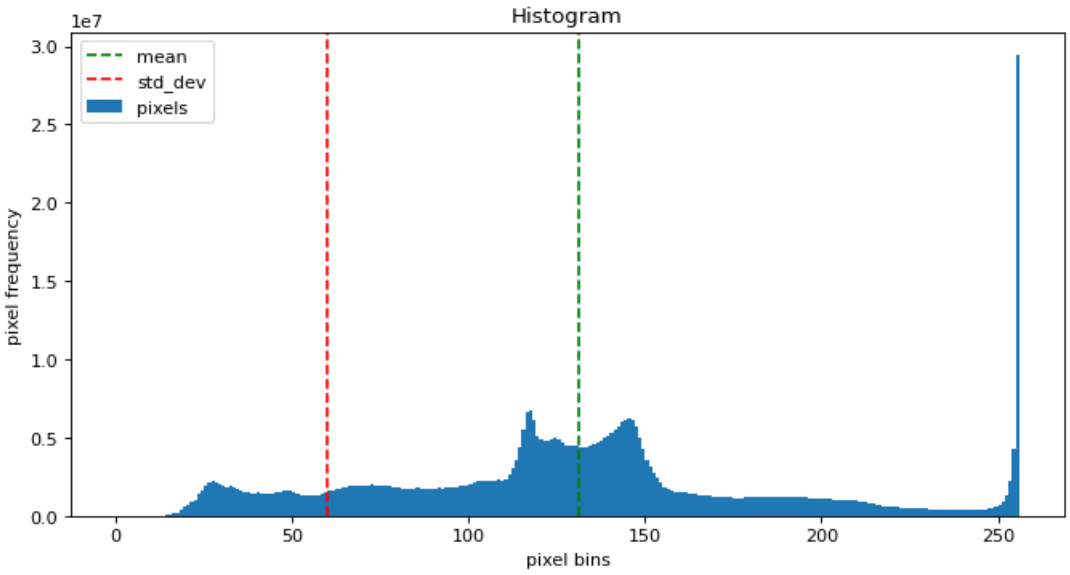

Fig 2. Histogram of full array of pixels of unit8 images

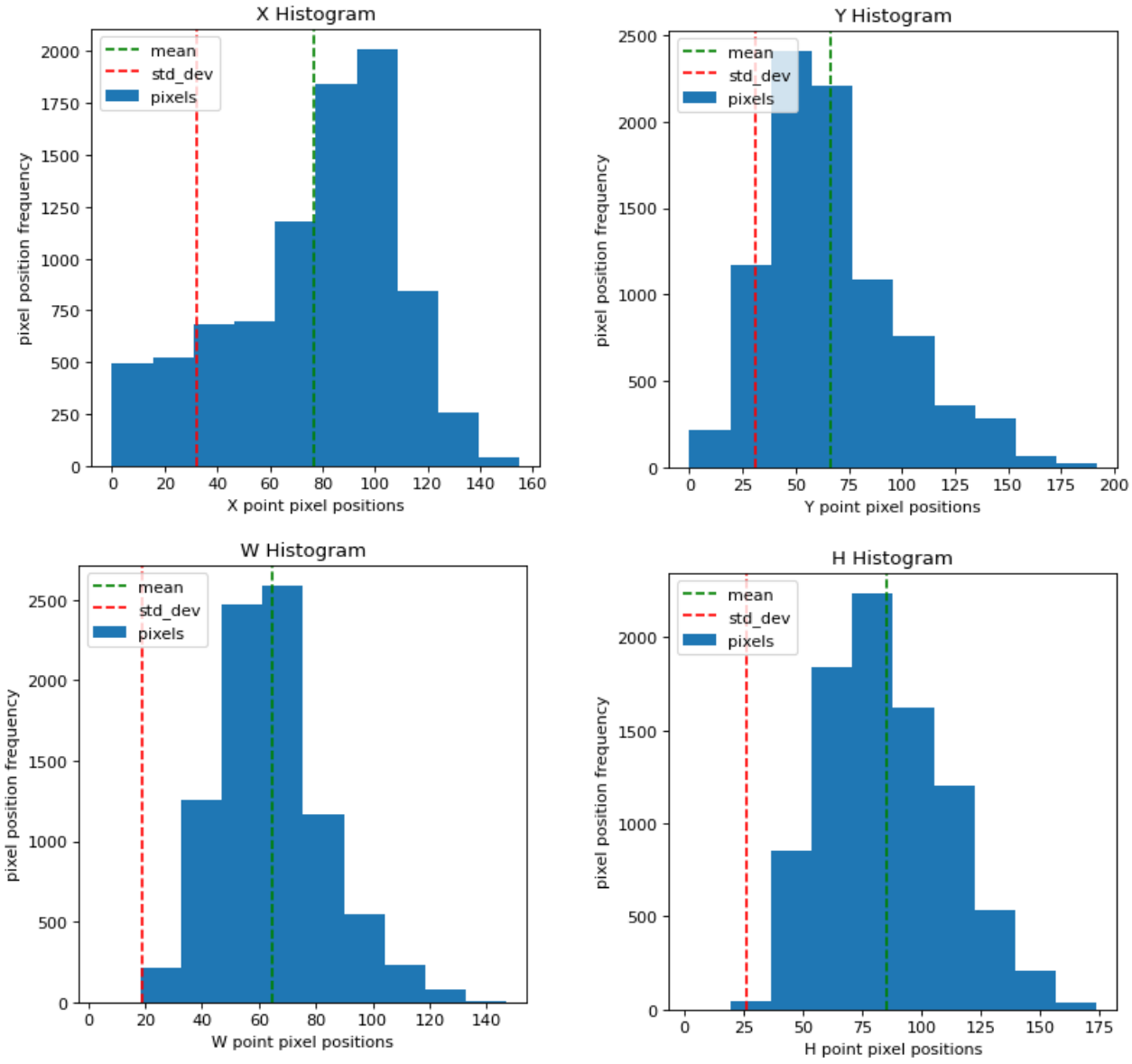

Fig 3. Histograms of bounding box points of hands

	img_name	x	y	w	h	letter
0	kl06x.jpg	72	103	76	75	A
1	xx8ga.jpg	72	99	74	76	A
2	nyiwx.jpg	73	92	78	73	A
3	ii$ks.jpg	76	81	73	74	A
4	o04n7.jpg	74	71	77	81	A

Fig 4. Input Features

4.1 Output Labels

The distribution of the output labels is slightly skewed in favor of some images due to issues with the hard ware not writing the images to disk fast enough causing fewer samples in some areas, but its still good enough to work with. A number of images had to be retaken as the camera froze several times while recording.

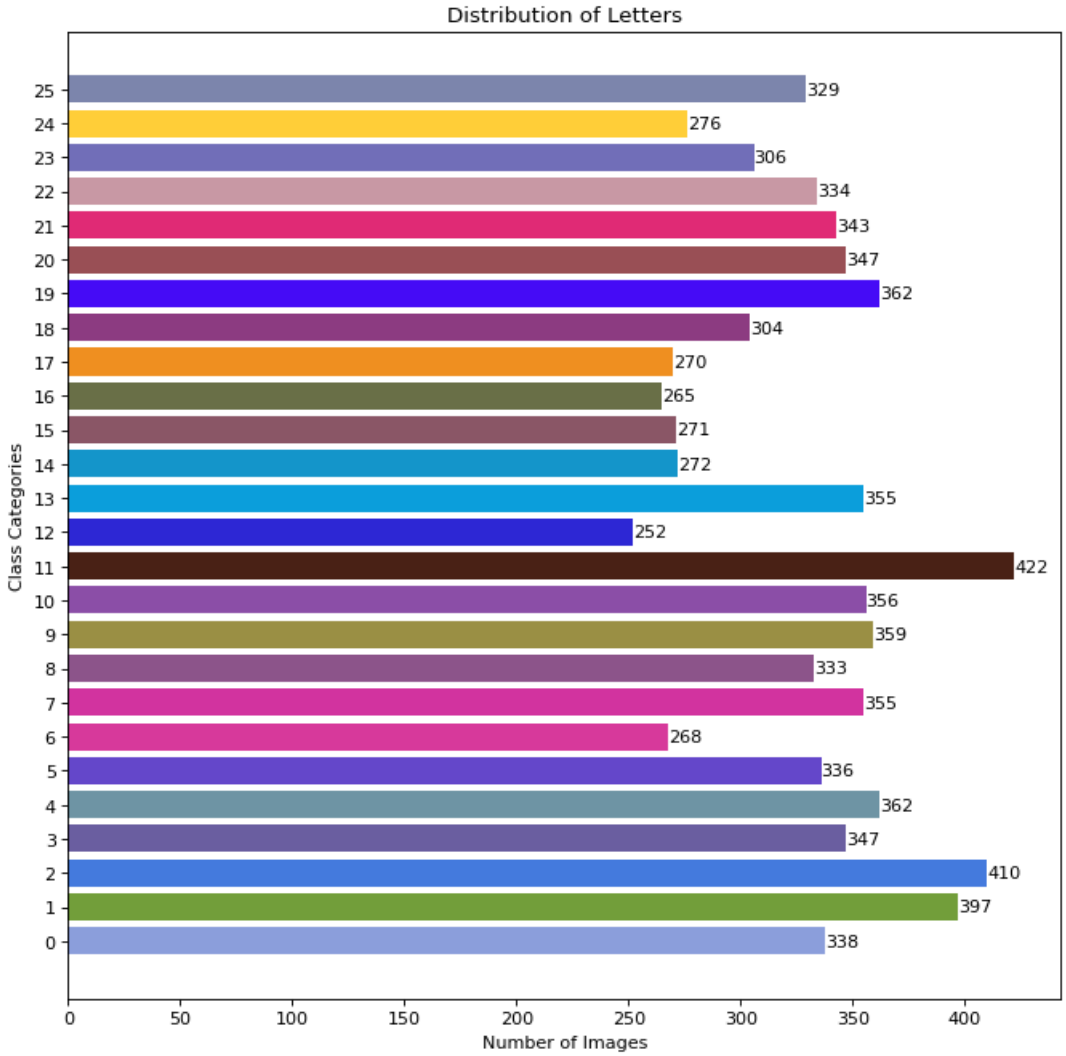

Fig 5. Label Distribution

5 Processing

5.1 Splitting the Data

The data was parsed into three sections

1. Images
2. Bounding points
3. Class labels

The images were gathered into their own list and then reshaped into a 8569 x 240 x 240 matrix. The bounding points were split into their own 8659 x 4 matrix and then stored for later use. No data normalization is performed on the bounding box points as those are the true bounding box positions of the images to be later used to pit the accuracy of the object detection portion of the model utilizing IOU or Intersection Over Union. The class labels were then parsed into their own 8659 x 1 matrix.

5.2 Data Normalization

The images were reshaped into a 8569 x 240 x 240 x 1 matrix to later feed into a convolutional neural network which requires a 4D matrix. The images were then normalized to a floating point value where all of the pixel intensity values range from 0 to 1 values. The Y labels were all normalized to integer values, one to 26 respectively and then one hot encoded to a 8569 x 26 matrix.

Below are a normalized images and the normalized histogram of the full image array of pixel values

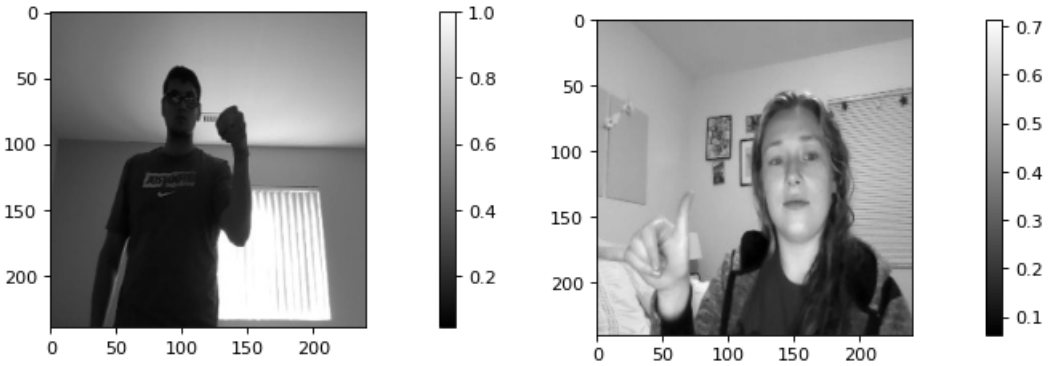

Fig 6. normalized images with pixel intensity values from 0 to 1

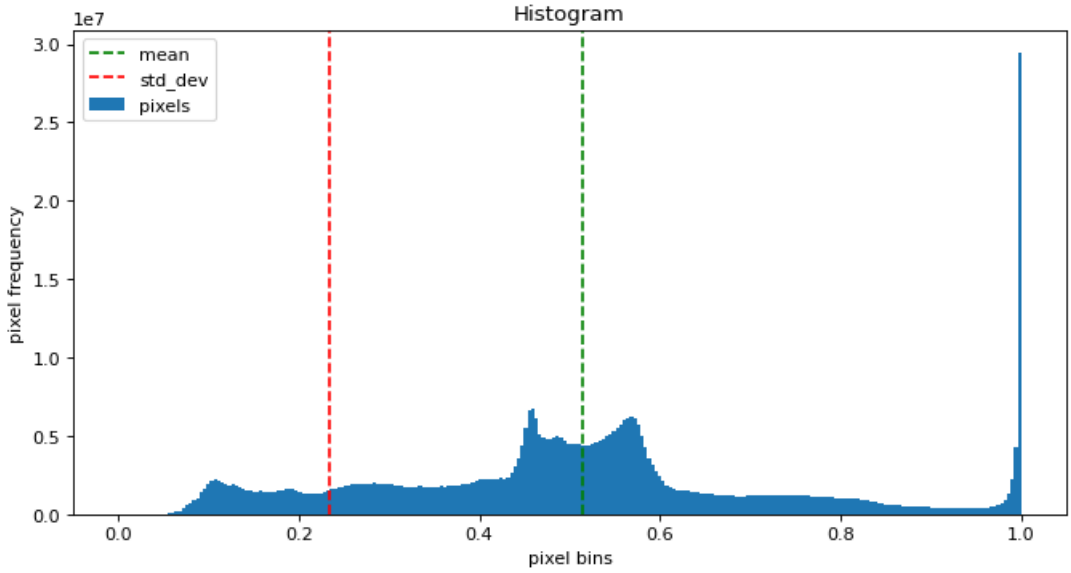

Fig 7. The same distribution as above only in floating point values now

5.3 Data Split

I opted for a traditional 70/30 split on the training and validation set to start off with. The training set consists of 5998 images and the validation split consists of 2571 images. The test set will consist of an entirely separate set of images not from the original set to see how well the model can predict against brand new data. I will have to process the new data in the coming weeks and will probably pull images from the internet and record new people signing.

6 Models

I tested four different models with varying layers and attributes to test out over fitting the model with the training data split to get a baseline. It was during the initial model i realized my machine had memory constraints that prompted me to resize my data from a 240 x 240 x 1 image array down to a 120 x 120 x 1 image array. After the data reshape i no longer had issues running the data through a small to medium sized model.

6.1 Tests

The first model was my baseline model. This was a simple 2 layer convolutional neural network with a maxpooling layer that feeds to a flattening layer before the final dense output layer. I started off the model with very few neurons, around eight in the first and double that in the second layer. This overfit extremely fast so i kept doubling neurons until i got to 64 neurons on the first layer and 128 neurons on the second layer. This model still overfit at an extremely quick pace.

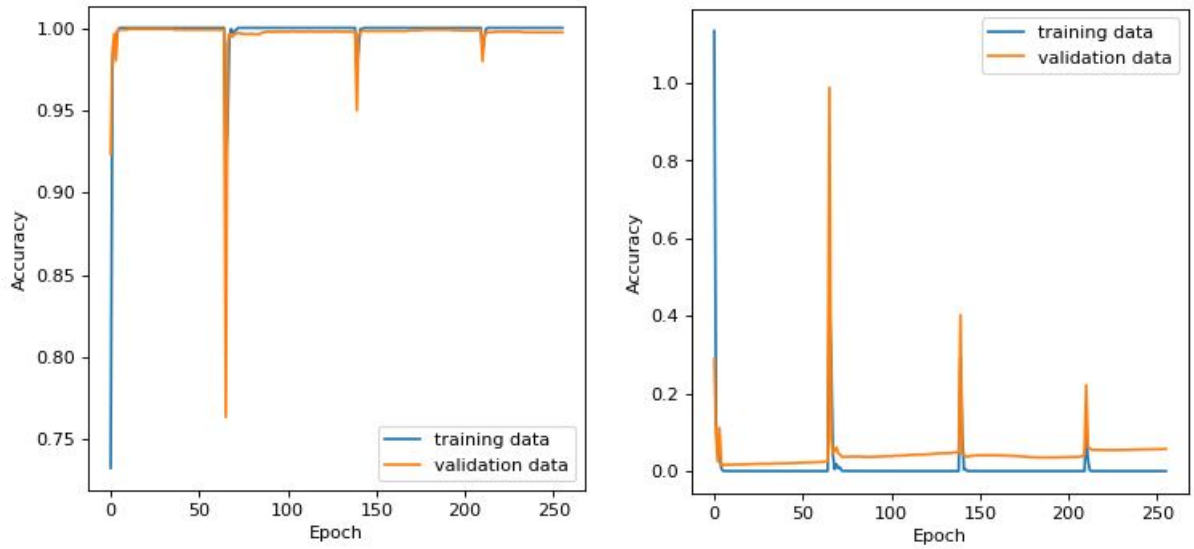

Fig 8. Notice the model overfits within the first few epochs

The second model i decided to try adding additional layers just as a test before adopting other methods to mitigate overfitting the model right away.

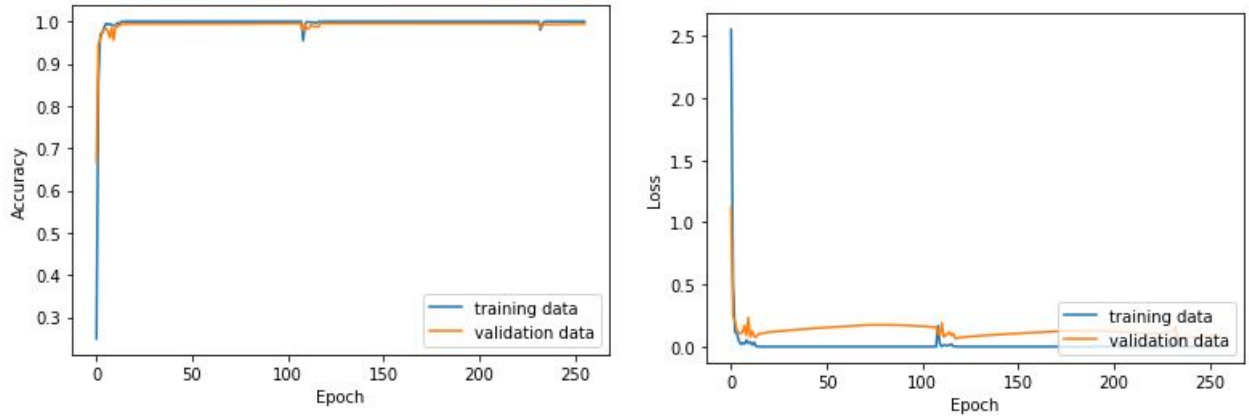

Fig 9. This model overfit just as quickly as the first

The third model dropout, L1 and L2 weight regularizers were utilized. The number of convolutional layers were reduced to two and an additional dense layer was added before the output dense layer.

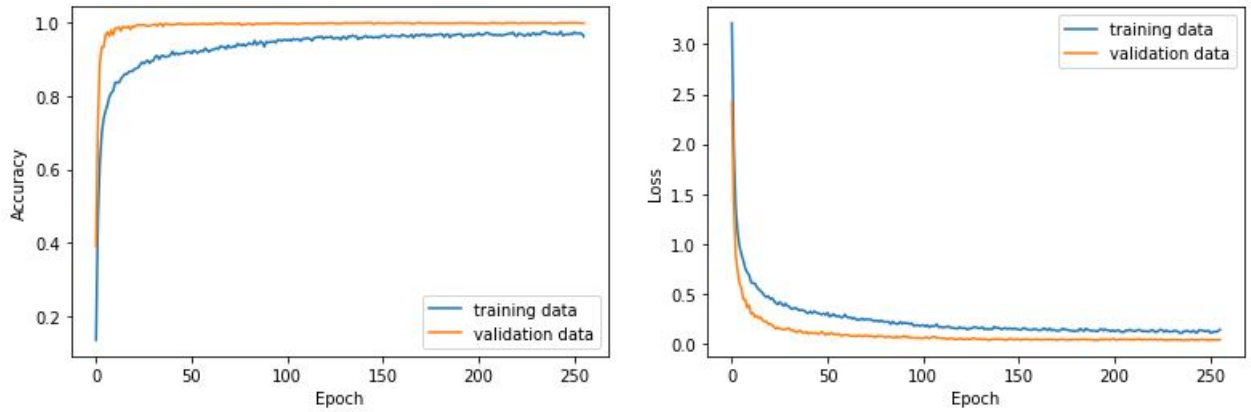

Fig 10. The dropout and weight regularizers helped, and can be adjusted to help with the overfitting

The final model i decided to use data augmentation to test out how that affected the performance of the model. I made the model simple with two convolutional layers and a maxpooling layer.

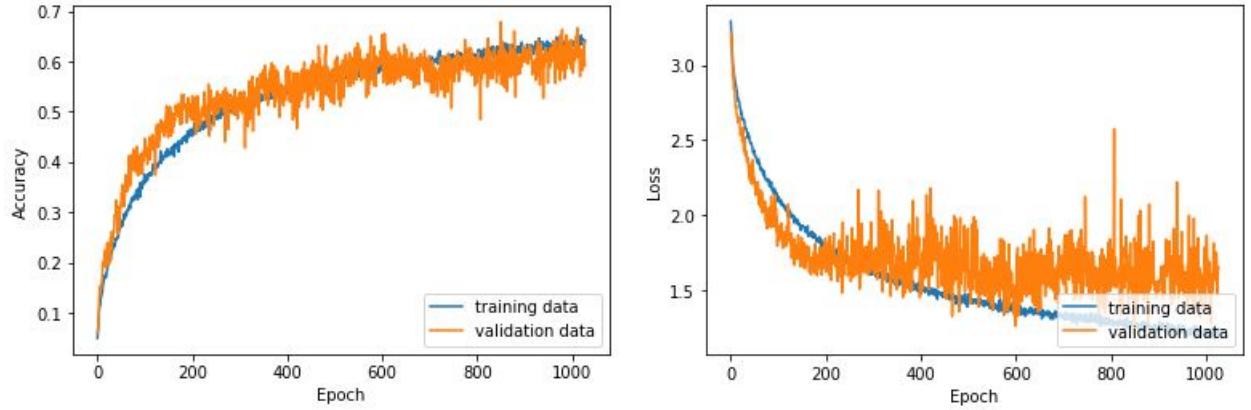

Fig 11. The augmentation kept the model from overfitting.

for the final model i will want something in between model 3 and 4 to give me the output i am looking for. i may need to utilize a combination of data augmentation and dropout with weight regularization strategies

6.2 Selecting a Model

In this next phase i decided to ditch the regularization techniques and dropout and opted for a deeper network with augmentation. I started with a very large network with multiple nodes for each layer and then either halved the nodes or quartered them to see the performance differences. I ran early stopping at 100 epochs and checked for changes in the validation loss. My final model stopped after 348 epochs out of 512.

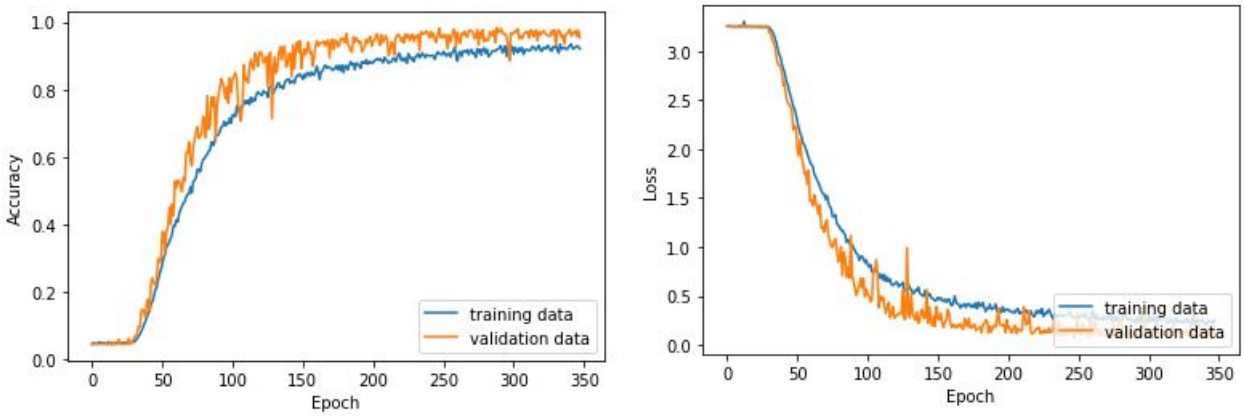

Fig 12. Final model learning curves

Out of the five models tested, the following metrics were used to test how the models ultimately performed against the training and validation data split.

- Precision: finding the ratio of correct predictions over the total number of positive predictions
- Recall: finding the ratio of correct predictions over the total number of true positives and false negative predictions
- F1-Score: weighted average of the precision and recall

Model	Precision	Recall	F1 - Score
Model 1	99.61	99.57	1.00
Model 2	98.87	98.79	0.99
Model 3	99.37	99.35	0.99
Model 4	98.93	98.80	0.99
Final Model	98.41	98.17	0.98

Table 1. metrics of all models trained over 512 epochs

Although the final model chosen does not have the highest accuracy, the model chose did not over fit. The previous models quickly reached the near 100 percent accuracy mark within 50 epochs.

6.3 Architecture

An attempt was made to add the outputs as inputs to try and see how large of an architecture was needed for over fitting. The Keras Functional API was used with a simple single layer convolutional 2d layer to a max pooling layer to a flattening layer and a final dense layer. The next model was a single dense layer with the input shape set as the outputs. I was able to successfully concatenate the models and compile them into a merged model but was only able to run one epoch before running into errors. I believe its due to a misshapen tensor for the second input needed, but i will have to do some additional digging to figure it out.

The final model comprised a convolutional layer with 32 nodes and a 7x7 kernel with the strides set to two. This was fed to a max pooling layer with a pool size of 2 and strides of 2. The next layer was a convolutional layer with 96 nodes and a 3x3 kernel with a single stride. This then fed to another max pooling layer with a pool size of 2 and stride of 2. The next four layers were convolutional layers with alternating nodes of 64, 128, 128 and 256 respectively. The nodes alternated between 1 and 3 and the first three layers all had strides of one and the final layer with a stride of 2. The next layer was an average pooling layer with a pooling size of 2 and stride of 2. I then flattened everything before the next dense layer with 512 nodes. The final output layer was a densely connected layer with 26 nodes.

7 Feature Reduction

During this phase for feature reduction i trained the model two ways. The first training session i started training with the images at their full size of 240 x 240. The next round of training was a two time size reduction in the image using bicubic interpolation to reduced the number of pixels. The third round of training i reduced the full size image by three down to a 80 x 80 image. The models all models had pretty good fits to the data. The best overall seemed to be the 240 x 240 model, it had the least inconsistencies in the loss and accuracy throughout the training process.

Image Size	Precision	Recall	F1 - Score
240	98.99	98.99	98.99
120	98.40	98.29	98.35
80	98.17	98.02	98.09

Table 2. Metrics of all models trained over 512 epochs with full features and reduced features.

The next phase of training i randomly added salt and pepper noise where i would randomly change five percent of the image pixels to either a zero or 255. I wanted to see if removing random bits of information would increase the training performance or drastically reduce the models accuracy.

Image Size	Precision	Recall	F1 - Score
240	97.00	96.85	96.92
120	98.64	98.64	98.64
80	97.82	97.67	97.74

Table 3. Metrics of all models with randomly added salt and pepper noise trained over 512 epochs with full features and reduced features.

The best model of this training exercise was the 120 x 120 model. It had the least variations in loss and accuracy overall. Although the 240 x 240 model had the best overall fit on the loss metric, with one major spike around the 180th epoch.

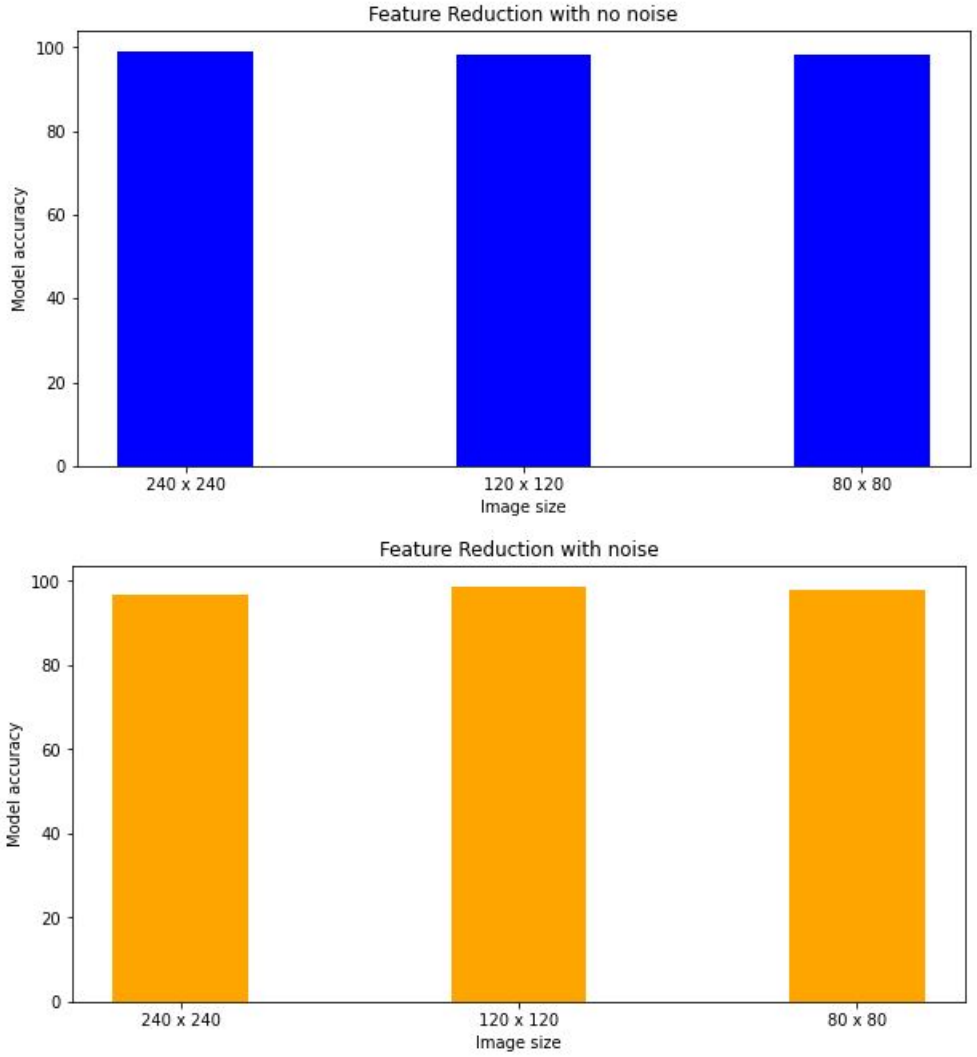

Fig 12. The accuracy's of the images as the features are reduced

8 Creating A New Model

During the process of working on phase four I came up with the idea to create a new model. In this new model i am feeding the model the training images and also the known boundary boxes of where the hand is in the image.

8.1 Multiple Inputs

I started off with just the 8569 images and known points. I divided each one of the points by the shape of the image to get the points down to a number between zero and one. I used the Keras API so that i could connect to the flatten layer for the Densely connected outputs for the class labels and also a densely connect model for the boundary points to train. I used a sigmoid output for each of the four points so that we get a value between zero and one again. I multiply those values by the image shape and get returned back a point value that is in range of the image size.

8.2 Gathering More Data

During the training portion i noticed that the accuracy was very high but when evaluating the model on images not within the training set or validation set the accuracy of the class labels was very low. So to mitigate this issue i pulled in additional images from an American Sign Language Kaggle dataset.[1] I brought in enough extra images to even out all classes to 500 images per class. After that i used data augmentation and then concatenated those results with the original images in order to bring my total images up 26,000.

8.3 Object Detection Accuracy

One of the main things you have to determine with an object detection model is the accuracy of the predicted bounding boxes from the ground truth boxes. In order to accomplish that i created an intersection over union algorithm that allows me to measure how closely the predicted boxes are to the ground truth boxes. After i am able to get those values i gather the accuracies over multiple images and get the mean accuracy precision of the bounding boxes for the model.

The intersection over union algorithm finds the maximum and minimum x and y points of both bounding boxes to determine the higher and lower boxes. Once those points are found we can determine the intersection box area between the two. below are the formulas i used

INTERSECTION

$$I = (X2 - X1) * (Y2 - Y1)$$

BOX AREA

$$A1 = |(box1_x2 - box1_x1) * (box1_y2 - box1_y1)|$$

$$A2 = |(box2_x2 - box2_x1) * (box2_y2 - box2_y1)|$$

INTERSECTION OVER UNION

$$IOU = I/A1 + A2 - I + 0.000001$$

Overall the IOU score was relatively low at around 50 percent on average. To increase this by a large factor i would need additional training data. The other options i have which i will most likely implement in the future is localization, segmentation and potentially anchor boxes. With the these additions i can create an invisible grid and take those regions and increase their size and run them through the model to see if any sign language is detected in the localized zone. If something is found in the region i can add a bounding box to the hand and then add the newly changed image back to its specified region.

9 Findings

In the previous model i was able to determine that a reduction in the number of pixels didn't affect the model accuracy to a noticeable degree.The most noticeable feature reduction was with the addition of noise. I also found that training on a deeper model with higher node counts increased the accuracy and conversely a shallower model with less nodes usually led to under fitting on the dataset. This especially was brought to light while training on the new model. decreasing the layers and nodes led to under fitting and the model was never able to fully classify or detect correctly.

After finding the correct amount of layers to train on the hyper parameter tuning was the most critical factor in increasing the learning rate and mitigating over fitting or under fitting. With the addition of the early stopping, model saving and tensorboard call backs i was not only able to save a good model but i was able to track current and past performances of the models and see the trends of the hyper parameter tuning.

10 Conclusion

Over the phases I have found that a deeper model with higher node counts worked best for the dataset. I also found that while training with the new model it helps to have a more evenly distributed dataset. In phase four I can conclude that removing pixels didn't really affect the accuracy too much but adding noise into the mixture made some noticeable changes to the accuracies and loss during training. Some weight regulation and changing the learning rate may have helped smooth out the results a bit more. The crucial component to this model learning well on such a small sized dataset was utilizing data augmentation. This was able to provide more training data and increased the models learning by a substantial margin. Overall, the model has done better than I anticipated it would finding the correct bounding area and classifying validation images. I have a lot of room to improve the model on testing images, but I now have ideas of how I can accomplish those tasks. I started to implement classification and object detection into a video platform but there is a lot of work that still needs to be done with the model before it can accurately predict. The model is not perfect, but it is accomplishing the tasks I set out for it to do.

References

[1] Akash. *ASL Alphabet*. Dec. 11, 2021. URL: https://www.kaggle.com/grassknoted/asl-alphabet.

[2] NIBCD Information Clearinghouse. *American Sign Language*. May 8, 2019. URL: https://www.nidcd.nih.gov/health/american-sign-language (visited on 09/15/2021).