Programming for Data Analytics 2

POE

Kieran Glezer-Jones November 2022
ST10041889

Contents

The Construction of an American Sign Language (ASL) Classification Algorithm.

1 Introduction

For a long time now, deaf people have struggled to interact with others as their means of communication are limited. The vast majority of people don't know any form of sign language even though they may wish they did. This means that if a deaf person wishes to communicate with another person without external tools, they will need to hope that person is one of the few capable of conversing with them via the means of hand signals. However, what if the method of learning sign language was significantly easier? I wish to make it easier for people to begin learning sign language and break down the barrier to entry by making it as beginner-friendly as possible.

The core idea is quite simple, people can stare at tests all day and match images to text with their mouse during online courses when learning sign language, but as most seasoned coders will tell you, it's important to get your hands on the keyboard when learning. Similarly, it's important for someone learning sign language to be able to practice making their hand symbols. That's why I want to create an algorithm that will take in an image of your hands and tell you what symbol you are making. This will allow the person to test themselves as well as provide instant feedback on whether they are correct or not. It can be made into a game, where the learning application will give you a word to spell out and if you are capable of doing it within the time limit you win.

Therefore, this report will focus on the planning, construction, and improvement of such an algorithm that can aid in the creation of a sign language learning tool.

For the type of sign language used, American Sign Language was chosen as it is the most popular sign language in the world as well as having distinguishing symbols that are easy to understand for anyone learning sign language for the first time (Sans, 2021). The language at its base consists of 29 classes that represent the English alphabet plus a few special symbols. The algorithm will be a classification algorithm that is capable of identifying these classes.

2 Background

The following section looks at what is already known about the dataset, tools, and requirements before we begin coding. It will additionally look at any important concepts that will be relevant in understanding why certain actions were taken.

2.1 American Sign Language

ASL is a form of communication crucial for many deaf people. It's not necessarily mandatory as external tools such as phones are capable of pulling their weight, but without tools, it can be difficult and frustrating for a deaf person to communicate with someone incapable of utilizing the language.

The language consists of quite a few classes but for this report, we will focus on the Alphabet plus some special cases.

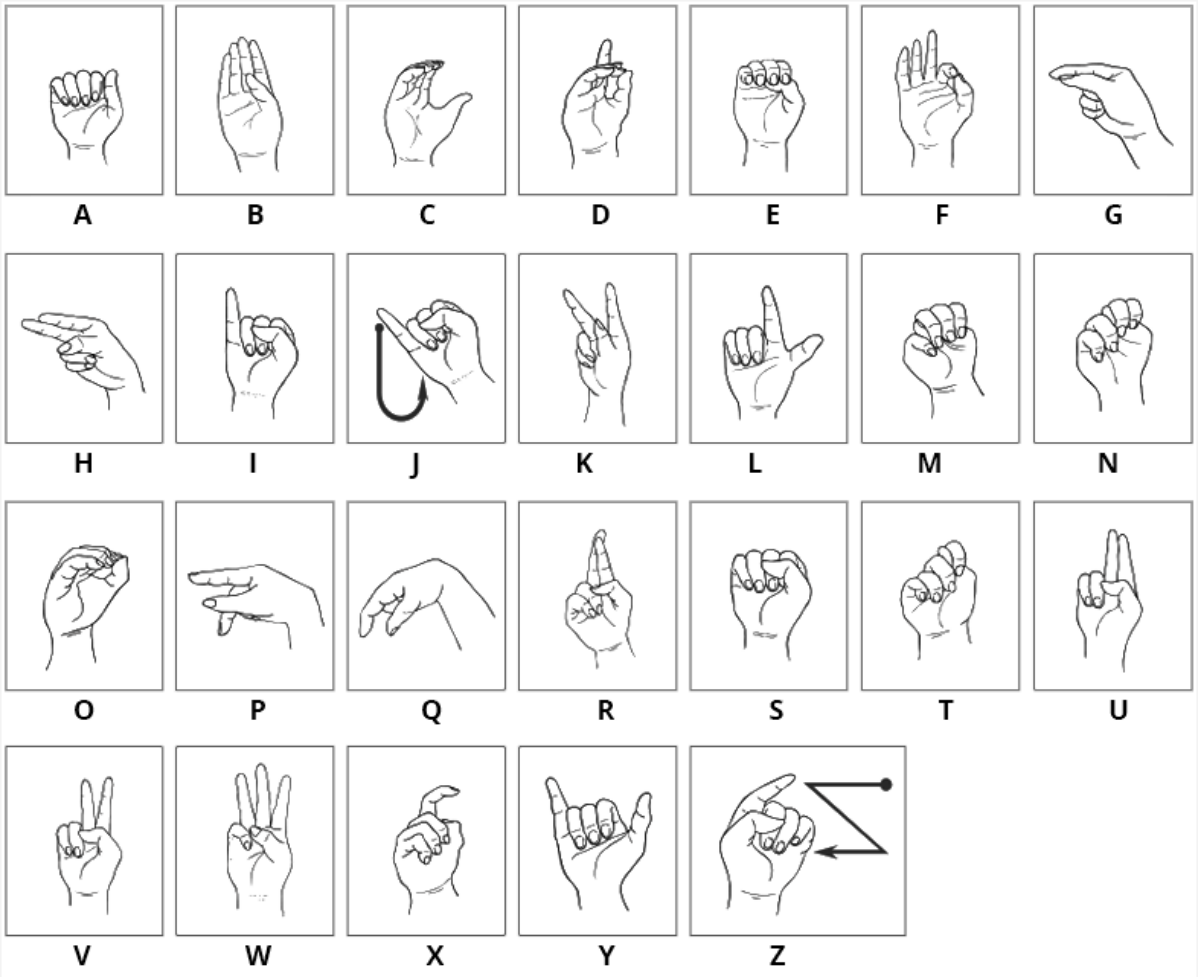

Figure 1 ASL by (APSEA, 2022)

2.2 The Dataset

The dataset consists of a few thousand images of hand signals for their respective category. There are exactly 3000 images per each of the 29 categories, which brings the total up to 87000 before we generate more by flipping, blurring, rotating, etc. The dataset is sourced from Kaggle.com and a link to it can be found in the README located in the parent folder of this report.

The dataset consists of all 26 letters in the alphabet plus 3 special characters, which are [Nothing], [Del], and [Space]. These letters will be the target for the algorithm while the images will be the inputs.

2.3 Supervised Image Classification

For this report, we will be using a supervised version of image classification as we already have access to the labels for the outputs. This means that we know exactly what it is that we want the algorithm to find.

Image classification is the process of supplying an algorithm with an image and then having that algorithm figure out what the image is (Lu & Weng, 2007). In this case, we will feed the algorithm an image of a hand symbol, and then the algorithm will decide which of the 29 classes that image is most likely.

2.4 Exploratory and Descriptive Analysis

When we take raw historical data and then explore it to try and find patterns and trends, we are performing descriptive analysis (The Independent Institute of Education, 2022).

For image classification, we can reduce the dimensions of the images and break them down to the lowest form to try and get an understanding of what parts of the image would the model find most important. Especially with hand signals, where your fingers might be obstructed by lighting or find themselves quite dark. With these discoveries, we can try and set a guideline for the user to be in the best position and lighting for the most accurate results.

2.5 Predictive Analysis

The predictive analysis aims to predict what something is based on large amounts of historical data, trends, and patterns (Cote, 2021).

The end goal for our project is to take an image and then predict whether the symbol is an A, B, C, Etc. Additionally, we can try to string lots of symbols together to create sentences or whole words. The issue, however, is the chance that the algorithm gets a single letter in the word incorrect. To compensate for this, we can attempt to add a pseudo-prediction algorithm similar to auto-correct, which will take an incorrect word and predict what the word was supposed to be. This will be done at a very small scope.

2.6 Why the Dataset is Appropriate Data for Image Classification

The following section goes over what is needed for the data to be considered appropriate for a model that needs to perform image classification.

2.6.1 Appropriate Images

The first and most important factor is that the input data for image classification needs to be images. If it isn't an image, then the algorithm is useless for image classification. Additionally, it can't just be any images, the images need to be related to what we want to classify. If we want to know what an H is in ASL, we need images of someone performing the H symbol.

Luckily the dataset satisfies these criteria as it provides 3000 images per category of someone performing the relevant symbol.

2.6.2 Dimensions

The images cannot be too small, or the model is going to struggle. The model might be able to make out what a 32 x 32-pixel image is if it was trying to classify text on a screen, but logically, the more complicated the classification is the more pixels you will need. Hand symbols aren't the most complicated things to classify as the algorithm can assume that it is a hand from the start. However, there are very small detail differences between certain letters, just as M being identical to N, except for an extra finger in a hard-to-distinguish space.

Because of this, the images must not be too small to begin with so that the algorithm has a decent amount of room to work with when analysing the pixel. It doesn't need to be massive, but at least large enough that there can be distinguishing pixels.

Luckily the images in the dataset are quite large at 224x224 pixels each, this is too large for our project as the training will be too intensive, but too big is better than too small as we can scale down the images.

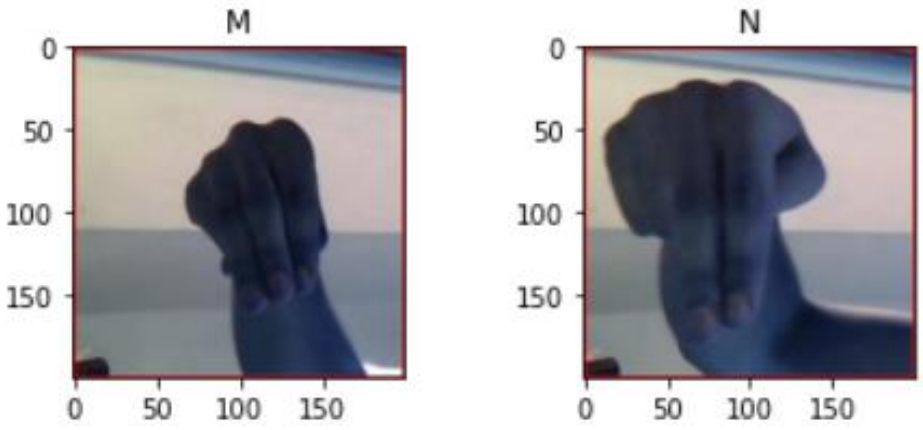

Figure 2 M and N in ASL

2.6.3 Enough Data

Without enough data, the model runs the risk of becoming severely underfit as it might not be able to grasp the trend of the data at all. This is not good as the model will be next to useless. It's quite the task to train a model to act like a human, and even more so with deep learning which requires a large amount of data (Brownlee, 2017).

Luckily, we have more than enough data to work with. 3000 images for each of the 29 classes gives us 87 000 starting images. If we start flipping and rotating images and adding those to the batch, then we start finding ourselves with a lot of images to work with.

2.6.4 Legit Source

The model needs to be fed using appropriate images. Which means the images need to be consistent and correct. In the case of building a sign language detection bot, if the images were of the incorrect sign, then the bot wouldn't be very useful. It also wouldn't be very helpful if some of the images in the folder were incorrect, so it's important to make sure that all the images in all the folders are of the correct sign.

Each image was manually viewed before selection, and the source is highly rated on the Kaggle website, so the bot did not suffer from this very much.

2.6.5 Target

For our image classification model, we know exactly what we want it to do. We want the model to predict our 29 classes, which means the model needs to have those 29 classes available for us to get. It would be a mission to try and sort the images manually but luckily the images are already sorted into their respective folders. The folders also segment a few images as testing images in a separate folder for us to try and use.

2.6.6 Clear Images

The images aren't messy and have a very clear and consistent background. With the background being somewhat consistent, the algorithm will notice that the only difference between each class is that the foreground changes. There is also no obstruction to the pictures such as blur. The foreground is also cantered which should help a decent amount.

2.6.7 Diversity

This is one of the few things that the dataset lacks as the images are of the same person's hand. Hopefully it won't impact the final model too much, it will, however, make the model slightly worse. This isn't a massive deal as this project is merely a prototype but if this algorithm was to be created for real production use, then a more diverse cast of hands would need to be collected.

This is because in reality people don't share the same hand features so having little to no diversity will lead to overfitting where it performs well on that one person's hand but will not work on other people with different features.

2.6.8 Lighting

The images should have good lighting (Huellman, 2022). But I would go so far as to contradict this slightly. I believe that the lighting should be the same as the lighting you expect the model to be utilized in. It is unlikely that an application that is used with learning intent will be run from a camera outside. It is much more likely that the camera will be inside a room with standard room lighting.

This isn't to say that all the images should be standard room lighting, but I think it would be harmful to the model to focus on studio level lighting when the real data is unlikely to be that good. So, a decent mix of both good lighting, standard lighting, and bad lighting will likely go a long way.

Thankfully the images within the dataset are all standard room lighting, which if we could select one of the three options mentioned above, would be the best one to go with.

2.7 Important Libraries

LIBRARY	FUNCTION
PANDAS	This will be used to store data in a DataFrame. This won't be used to store the input data but can be used to instead store any metrics that were collected during training.
CV2	This package will allow us to store the images in a cv2 variable. That variable has a host of useful functions as well as being able to visualise the image within a notebook.
MATPLOTLIB	This library will be used to visualise not only the images but also any other metrics we might be interested in.
PIL	Pillow is an open-source library that will allow us to manipulate images manually, such as rotating or flipping them.
TENSORFLOW	TensorFlow is a massive library that specialises in aiding people in performing deep learning tasks. The functionality of the library is huge, but we will be focusing on the construction of a model.
KERAS	Keras provides us with an interface to manipulate and work with artificial neural networks. It is hosted within TensorFlow and will be used to help us create the layers of the CNN.
GARBAGE COLLECTOR	A simple library that will allow us to call the garbage collection at will. This will help us save a lot of space as we work with large variables.
SCI-KIT LEARN	This library has a lot of useful functions for machine learning, but for our deep learning project, we will use it to split the data randomly using the train_test_split function.

3 Data Wrangling

There are six steps involved in performing data wrangling. Those steps are discovering, cleaning, enriching, validating, and publishing (Stobierski, 2021). This process describes the steps we take when getting the data ready for the model to use.

3.1 Discovering / Data Exploration/ Descriptive Analysis

This is the step where we familiarise ourselves with the data to understand what needs to be done and if there are any issues that need to be sorted out (Stobierski, 2021). It's important to know what we are working with as it will help us draw up a plan to tackle the rest of the construction process.

3.1.1 Data Collection

The first step we will take is to collect the data that we are looking for. A nice dataset that was highly rated on Kaggle was chosen and downloaded. The link to the dataset can be found within the README located in this document's parent folder.

The folder consists of a few subfolders, breaking the data into training and testing folders. Within the testing folder are some images we can play with after the model is trained. In the training folder there are folders for each class.

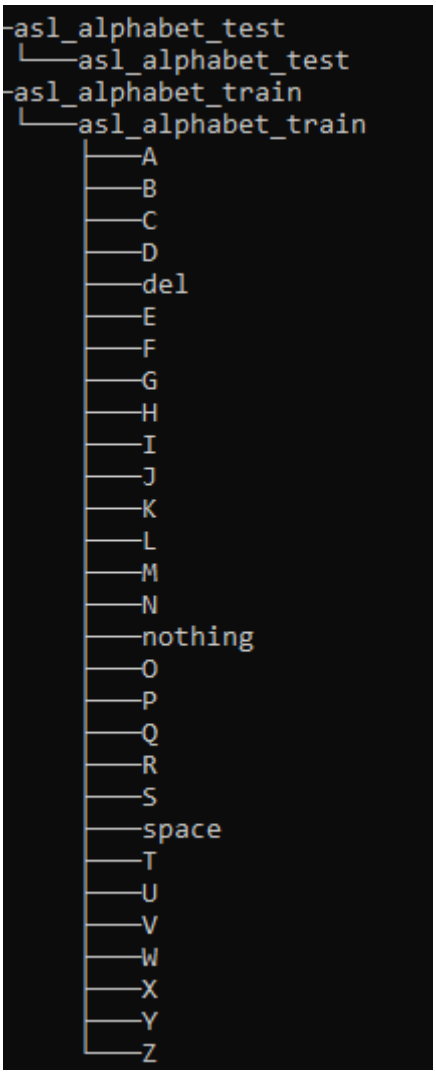

Figure 3 File directory of the images

3.1.2 Data Visualisation

Now that we have collected the data, we can take a look and see what the data looks like. In this instance, we will be looking to see what the symbols for each of the classes look like.

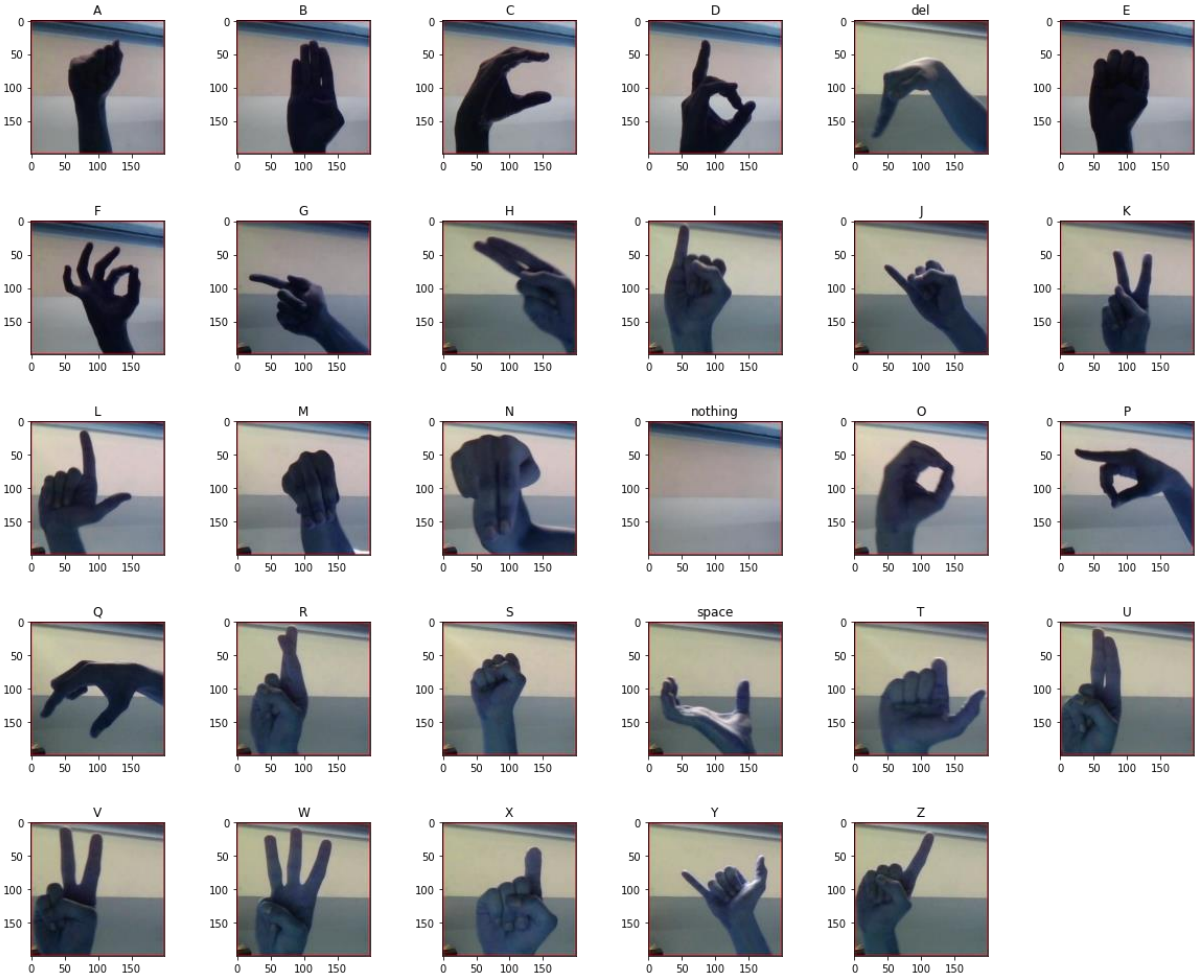

Figure 4 Twenty-nine classes

We see that each of the signs is quite distinct, but there are a few that might cause a problem, such as M and N. If we were to build an algorithm on these images, it might struggle with these small details. Luckily there seem to be pretty distinct shapes between the images. So, when we scale down it will probably still be possible to make out certain symbols.

The downside is that there doesn't seem to be much variation between the hands in terms of whom they belong to. This might prove problematic in the end, but the user was smart, they kept the room darkish and clear so the algorithm will focus on the shape more than the small details. So, it will still likely perform well in testing. However, if this was for a real application, I would have heavily suggested getting more variations.

3.2 Structuring

This is the part of the process where we take the raw data and transform it into a format that we can easily work with (Stobierski, 2021).

In our case this will involve taking the raw image, loading it into pandas using the cv2 library, and then converting it into an array so that we can work with the individual pieces. The array will host the RGB value for each of the pixels within the original image.

```
#Load the image to an array
image = cv2.imread('Images/asl_alphabet_train/asl_alphabet_train/A/A1.jpg')
```

For each image, there will be three channels, each of which represents either the blue, green, or red layer of the image.

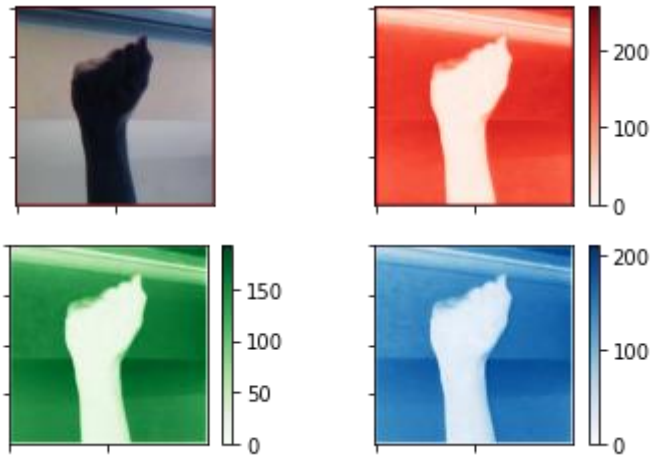

Figure 5 The RGB channels of an image

We will also scale each of the images down so that they are a bit smaller and more consistent. This will also reduce the amount of time that we will spend training the models as well as make the model focus more on shapes, rather than specific features, such as wrinkles as they won't show up on a lower resolution.

```
#Resize the image
resize_img = image_array.resize((IMAGE_DIMENTION , IMAGE_DIMENTION))
```

For now, we will resize the image to 64x64 from 224x224.

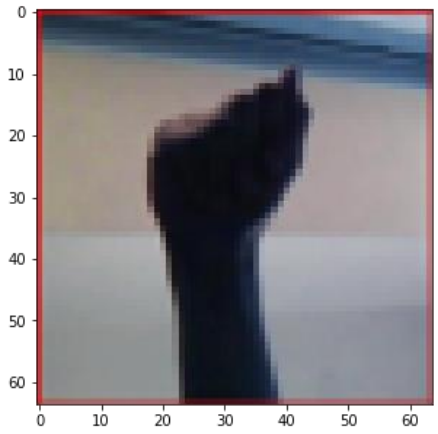

Figure 6 Resized image

Lastly, we will look at the fact that the image is currently in colour. From the beginning, we understood that the shape of the hand was going to be more important than its colour. This is because there are many different types of skin tones, and the algorithm doesn't need to identify that it is a hand. It just needs to identify the shape under the assumption that it is a hand. So we will grayscale the image.

```
#Convert to greyscale
image_array = image_array.convert('L')
```

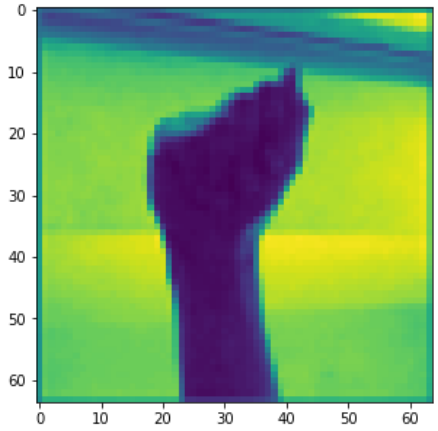

Figure 7 Grayscale image

3.3 Cleaning

This is the part of the operation where we look for any errors in the data that might impact the model's performance (Stobierski, 2021). In our case this involves making sure that all the pictures are correct. This is a tedious process where we manually check to make sure that there are no errors in the file names and that there are no images that stand out as incorrect. The dataset wasn't too large so it wasn't too difficult to scroll through the file system, but with larger datasets this issue should be kept in mind while collecting the initial data.

3.4 Data Enriching

This is the step where we consider if we have all the necessary data that we need (Stobierski, 2021). In this case we do not as we not only don't meet the minimum count for the images, but at its current state it will be incredibly overfit. The data at the moment is too perfect so we need to take precautions to account for the randomness of real life. To do this we will take each image and rotate it 45 degrees and add this additional image to the dataset. So now we have double the dataset, but we won't stop there, we will also rotate each image 75 degrees as well, and then flip each image, followed by a little bit of blur. These steps are to add a little bit of imperfection to the images as well as pad the dataset. The dataset size, however, has now been completely thrown out of scope, so we will reduce the initial amount of data we feed in from 3000 images per class to 690 images per class, so we are left with just over 100 thousand images.

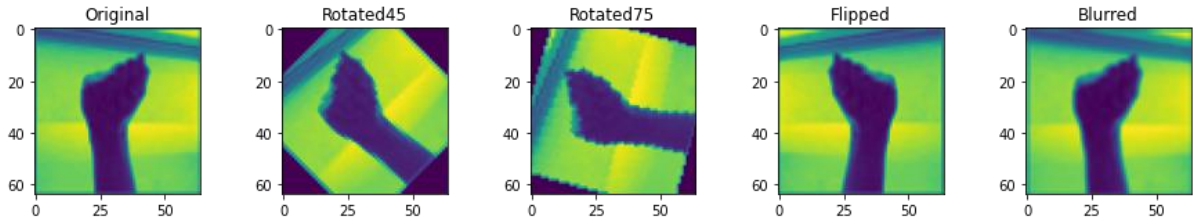

Figure 8 Augmented Data

3.5 Validating the data

This is the process where we verify that the data is consistent and of a high enough quality for our model (Stobierski, 2021).

This process does not refer to how we validate the model, instead, it looks to make sure that the data is in the right shape for the model to work with it. This is a hard process but one easy way to somewhat verify that we can work with the data is to build a small testing model to make sure that it doesn't crash.

If we left the data as is, it would fail as the convolutional neural network from Keras requires the use of NumPy arrays, which means we need to convert all the image arrays into NumPy arrays so that the model can work with it.

```
picturesArray = np.array(data)
labelsArray = np.array(labels)
```

Once a testing model is built, we can run the data through one or two epochs to see if it is happy with the data.

```
Epoch 1/2
2175/2175 [==============================] - 65s 29ms/step - loss: 2.0696 - accuracy: 0.4372 - val_loss: 0.7942 - val_accuracy: 0.7661
Epoch 2/2
2175/2175 [==============================] - 62s 28ms/step - loss: 0.7462 - accuracy: 0.7519 - val_loss: 0.3984 - val_accuracy: 0.8932
```

Figure 9 Verifying the data

3.6 Publishing the Data

This is the part where we make the data available to others or the model (Stobierski, 2021). For this step we will save the data into .npy files so that we can call this data without having to process it again.

```
np.save('Numpy Arrays/_X_picArray' , picturesArray)
np.save('Numpy Arrays/_y_picArray' , labelsArray)
```

Afterward, we can call the data with a simple load function, which will take the .npy file and store the contents into a NumPy array.

```
picturesArray = np.load('Numpy Arrays/X_picArray.npy')
labelsArray = np.load('Numpy Arrays/y_picArray.npy')
```

4 Model Construction

The following section goes over the creation of the neural net as well as the decisions made to build the best possible model.

4.1 Convolutional Neural Net

A convolutional neural network (CNN) is a deep learning algorithm that can assign importance to various aspects of an image. The result of this is the ability to differentiate key concepts from different images. The network is designed after the neurons in the human brain and was inspired by the visual cortex. This inspiration comes from how individual neurons respond to stimuli from specific regions of the visual field called the receptive field. And the brain has a collection of these fields which cover your entire sight range (Saha, 2018).

A convolutional neural network is a feedforward neural network, which means that the data is passed from layer to layer, with slight weight adjustments to finally come to the final output layer which throws out a prediction (Baktha & Tripathy, 2017).

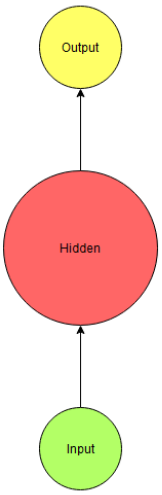

Figure 10 Feedforward neural net

When we train the convolutional neural network, we utilize backpropagation where we tune the weights of each node based on the error rate received from the previous epoch. This essentially helps us calculate the gradient of the loss function (Johnson, 2022).

The input for the CNN requires each pixel to be broken down into its RGB value for each channel (Saha, 2018). In essence, we have 3 versions of the same image, one with the values for red, another with blue, and finally one for green.

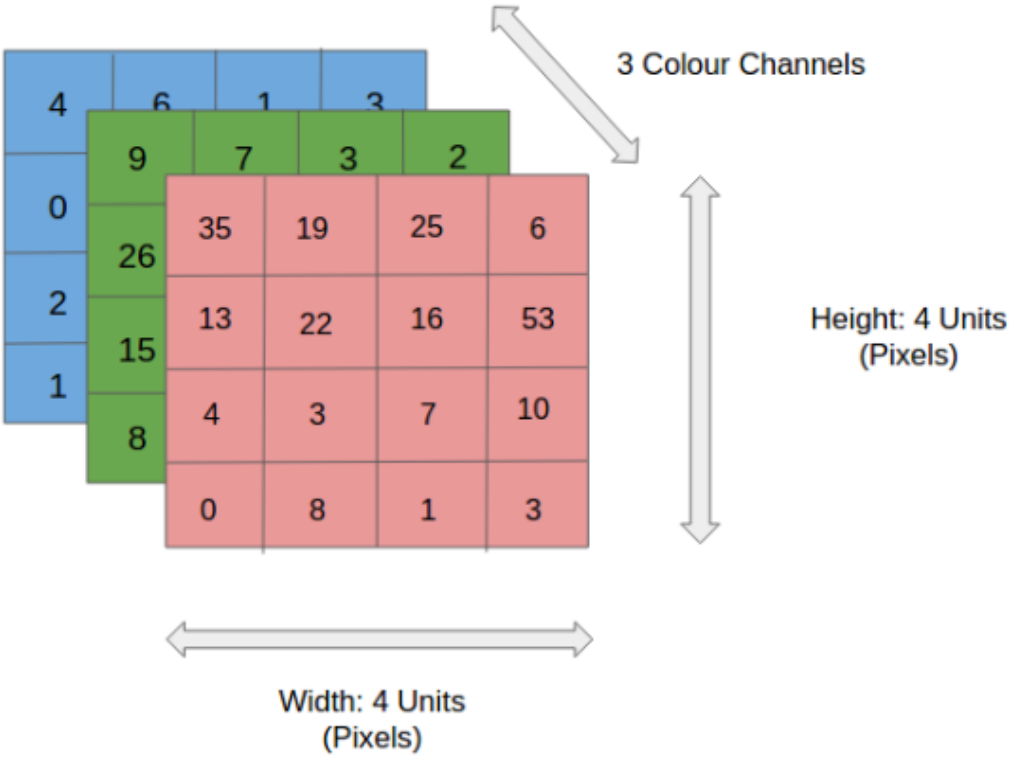

Figure 11 CNN layers by (Saha, 2018)

The CNN will then reduce the dimensions of the image down to a form that is easier to process by iterating through batches (Saha, 2018). This is done because the computational power to process very large images would be immense and unforgiving. So, we need to scale the data down.

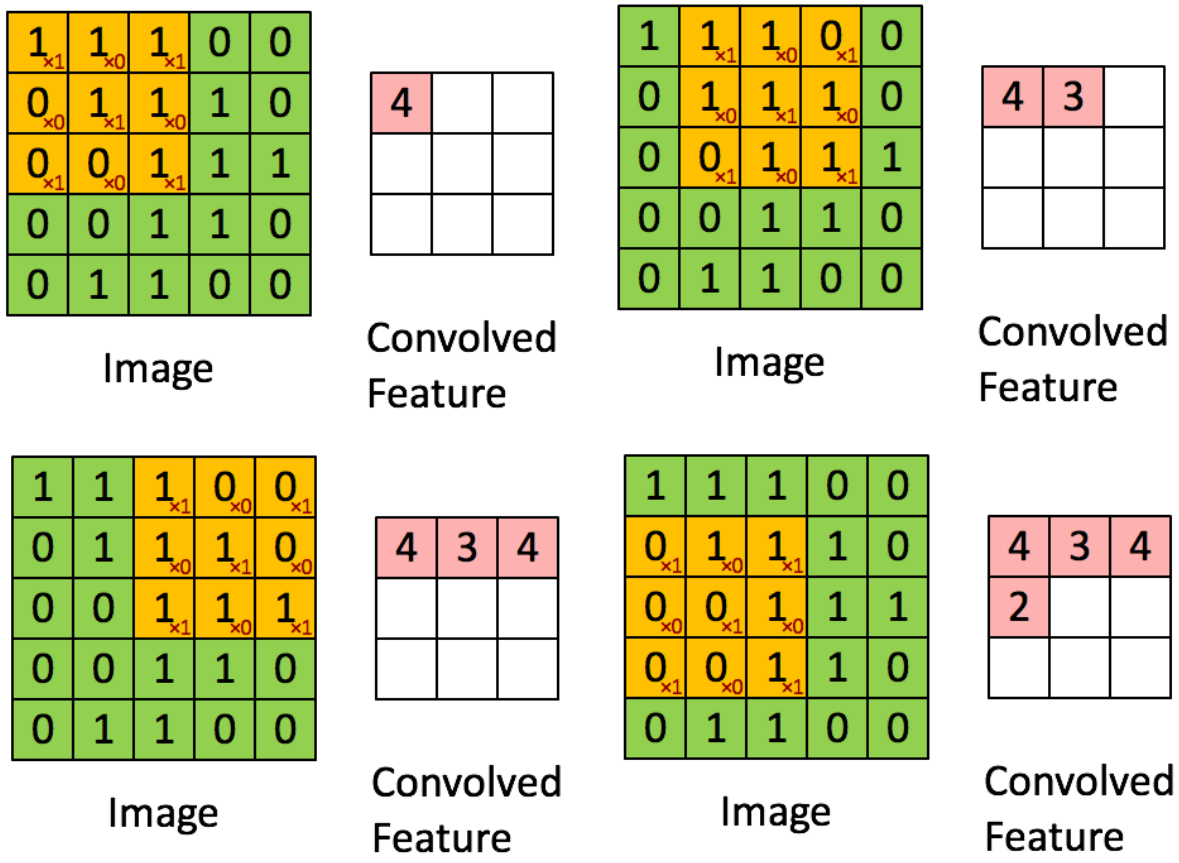

Figure 13 CNN iterations by (Saha, 2018)

After it has broken these layers down it uses them as a filter for the final image render (Saha, 2018).

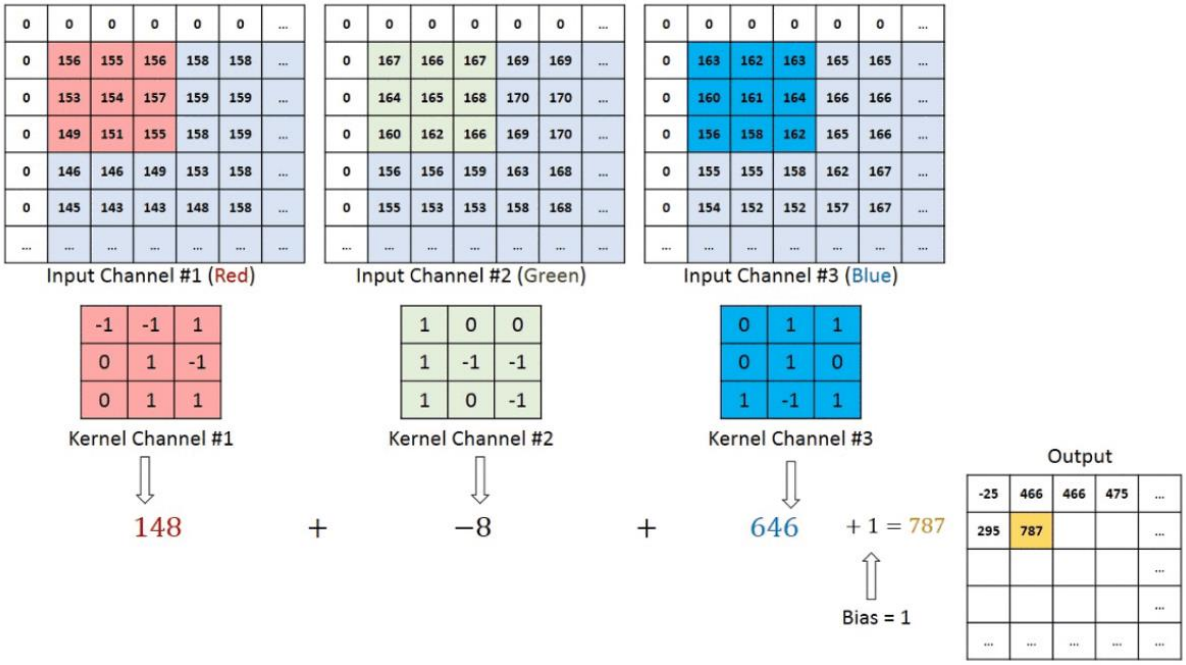

Figure 12 CNN iterating with the filter by (Saha, 2018)

Once the iterations are complete the result should be that high-level features such as edges should be extracted from the images (Saha, 2018). And this is done for every convolutional layer that is added. So, the more layers you add the more in depth the model can become.

The model can then take this new flattened version of the data and feed it into a feedforward neural network that can apply weights and perform backpropagation where the loss is propagated back into the neural net so the weights of each node can be adjusted (Misra & Jiabo, 2020).

The model is then able to take the output and analyse it to see if the classification should be X, Y, or Z. It will look for things such as grouping around certain sections of the grid, weighted averages, etc. The full model will look something like this:

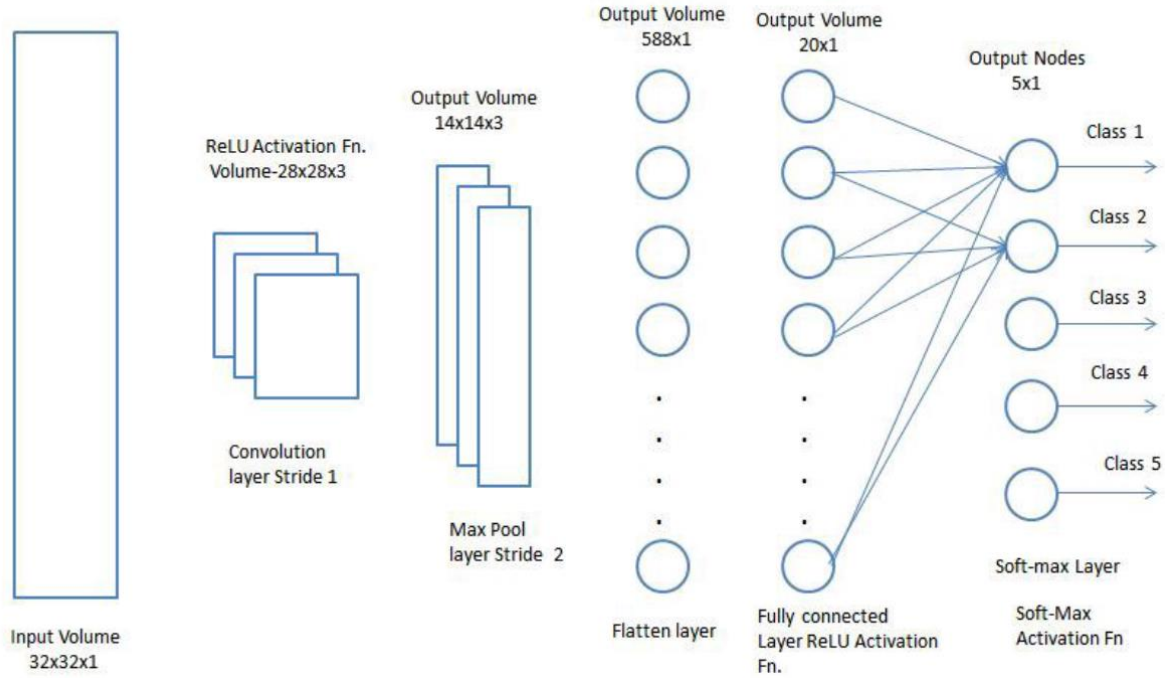

Figure 14 Full CNN by (Saha, 2018)

4.2 Preparing the Data

The following section goes over the final steps taken to prepare the data for the model.

4.2.1 Training and Testing Data

When we want to validate our data after we have built the model, it would be pointless to test the model with data that it has already seen and been trained on. To combat this, we will split the data into training and testing data and keep the testing data away from the model until we are ready to provide an unbiased simulation of the real world. For this project, we will use a split of 80% training and 20% testing.

```
X_train, X_test, y_train, y_test = train_test_split(picturesArray,
labelsArray, test_size=0.2, random_state=42)
```

A random state is set so that our data is split randomly with the seed of 42, which means that the next time we run the code the random split will be the same as the last time.

4.2.2 Reshaping the Data

The data needs to be in a format that the model can use. This means, for a convolutional neural network, the data needs to be in a NumPy array. The input arrays should have a shape of X, 64,64. X for the number of observations and 64 for the height and width. Interestingly enough there is no 3 at the end to represent the channels. This is because the method we used to grayscale the image took away the RGB layers since they are no longer relevant.

```
(80040, 64, 64)
```

Figure 15 Shape of Input Array while 3D

This is a problem, however, as the convolutional neural network is specifically looking for those RGB layers and wants a shape of (X, Y, Z, A). The important thing to keep in mind here is that the number of layers we add to the end of the shape is optional to the CNN, so it is a simple matter of stating that there is a single layer that holds the grayscale value for each pixel.

```
#Reshape X_train and X_test to 4D
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1],
X_train.shape[2], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.shape[2], 1)
```

```
(80040, 64, 64, 1)
```

Figure 16 Shape of Input Array while 4D

4.2.3 Create Dummy Values

To efficiently use categorical values within python it is important to break the data down into the easiest form for a computer to work with, namely binary. Currently, the target variable is a value from 1 to 29 as it was set that way to make identifying the number easier for a human being, as 2 would be the second letter in the alphabet which is ‘B’. However, the model won’t be able to use this as well as it would use a binary variable since they are more flexible. So, in summary, we must convert the label values into 29 columns of binary true or false values.

```
#Create dummy variables
y_cat_train = to_categorical(y_train)
y_cat_test = to_categorical(y_test)
```

We can now look at the first instance in the categorical variable.

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)
```

Figure 17 Binary Categories

We see that the 11th variable is set to true, this means that this is a 'K' we are working with.

4.3 Overfitting and Underfitting

The following heading looks at what we need to keep in mind when dealing with overfitting and underfitting as well as how to test it and how to fix it if it appears to be problematic.

4.3.1 Overfitting

Overfitting occurs when the model is fit too closely to the training data which means that the model will perform very well in training but poorly in testing when it is introduced to data it has never seen before (Muller & Guido, 2017).

This dataset proved quite challenging when it came to dealing with overfitting as the pictures were of a single person's hand, so it struggled when introduced to hands that were not that specific person. However, with some clever techniques, the model eventually managed to accurately predict the testing data as well as an external person's hands.

The first step that was taken was to grayscale the images, so the skin colour of the person's hand didn't play a massive role. It would still play somewhat of a role, but it proved to not be a problem in the end.

Secondly, the images were distorted slightly to add a bit of error to the data. This included rotating the images slightly at random or adding a bit of blur. If the input data is too perfect, then the model will struggle when it comes to the unreliability of humans.

Thirdly, the images themselves were done quite well when you consider that the person only had his hands in the frame with no external objects on the walls and no curtains. It's not perfect as the hand is in front of a window but it worked alright in the end.

The fourth measure that was taken was the addition of dropout to each of the convolutional layers. This involves randomly and temporarily turning off certain nodes in a neural network as it is training to force the other nodes to adopt a portion of their weights (Brownlee, 2018). Conceptually it makes a lot of sense. For the sake of simplicity let's assume each node looks after each pixel. If that pixel never goes higher than 100 then that node won't know what to do when a higher value is present, there. However, if a node that often gets those higher values is turned off for a cycle, then the ignorant node suddenly needs to realise that weights above 100 exist and what it needs to do when they are present.

The fifth measure was the introduction of early stopping, which would stop the training if the accuracy was going down or not improving. This prevents the model from training for too long.

The final step that was taken right at the end was the addition of more data at a higher resolution. Instead of breaking the images down into 64x64 chunks, the images were instead left at their full capacity of 224x244. This allowed certain features such as finger placement in front of the palm to be picked up easier. This was only done at the end of the assignment and transfer learning was utilized as someone else had already taken the liberty of sitting through the training time, using the same data, which likely took an enormous amount of time.

4.3.2 Underfitting

Underfitting is when the model struggles to grasp the trends and patterns of the data, so it performs poorly in both training and testing (Muller & Guido, 2017).

The biggest contributor to underfitting is bad data to begin with. Taking our dataset as an example, if the images in the folders were inconsistent where the sign for K was mixed into the folder for L, or if the dimensions boast an extremely low resolution, or if the lighting is really bad, etc. The data must be pre-processed correctly without error.

Another contributor is if there is not enough data to begin with which can easily be fixed with a little bit of data augmentation such as duplicating images at a slight angle.

Another issue could be the use of bad parameters, such as having the batch size too large or using high values for the dropout.

Luckily this model shows no issues that might be caused by underfitting as it struggles much more with overfitting.

4.3.3 Testing for Overfitting and Underfitting

Normally it is quite easy to test for overfitting and underfitting as you can introduce the testing data that the model has never seen before. This is still a very good indicator, but it's important to consider that the testing data is still the same person's hand that was used to train the data. This is of course a big problem, so to validate the testing data, external images supplied from a willing model were used to see if the model could pickup what she was trying to say.

In the end, the model is still slightly overfit, but it isn't a massive deal as it won't be used in real production. If the model were to be used in production I would motivate for more variety between the hand models.

4.4 Creating the Convolutional Neural Network

The first step in the creation of the convolutional neural network is the addition of a convolutional input layer. The dimensions of which need to be [The height of the image, The Width of the image, and the number of channels]. Since we have grayscale images, we will use a single channel.

```
model.add(Conv2D(32,(5,5),input_shape=(IMAGE_DIMENTION,IMAGE_DIMENTION, 1)))
```

Next, we will add a dropout layer to randomly turn off 20% of the nodes during training.

```
model.add(Dropout(0.2))
```

Next, we add a layer to take the feature map produced by the Convolutional layer and create an activation map as its output.

```
model.add(Activation('relu'))
```

Next, we need a layer that will take the max feature from a selection.

```
model.add(MaxPooling2D((2, 2)))
```

We repeat the above steps a few times until we are happy with the results. At the end of the model we need to flatten the data and then condense the categories down into 128 possibilities which are then converted into the final 30 possible outputs.

```
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(30, activation='softmax'))
```

If we put it all together it will look like this:

```
model = Sequential()

model.add(Conv2D(32,(5,5),input_shape=(IMAGE_DIMENTION,IMAGE_DIMENTION, 1)))
model.add(Dropout(0.2))
model.add(Activation('relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Dropout(0.2))
model.add(Activation('relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Dropout(0.2))
model.add(Activation('relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(30, activation='softmax'))
```

4.5 Interpreting the Results

After running the model for the first time after selecting the best parameters we notice that the accuracy of the model is shockingly high.

```
Epoch 29/50
1601/1601 [==============================] - 9s 6ms/step - loss: 0.0239 - accuracy: 0.9936 - val_loss: 0.0187 - val_accuracy: 0.9959
```

Figure 18 Accuracy after 29 epochs

	loss	accuracy	val_loss	val_accuracy
0	1.486658	0.535345	0.781926	0.787156
1	0.415634	0.862631	0.344491	0.917241
2	0.209332	0.931047	0.272246	0.920640
3	0.143538	0.952649	0.146300	0.969065
4	0.116544	0.961257	0.141983	0.964568

Figure 19 Metrics head

	loss	accuracy	val_loss	val_accuracy
27	0.024815	0.992741	0.073545	0.976862
28	0.021231	0.993978	0.039202	0.989205
29	0.019864	0.994328	0.042745	0.989655
30	0.023620	0.993303	0.038327	0.989455
31	0.019357	0.994590	0.093814	0.968516

Figure 20 Metrics Tail

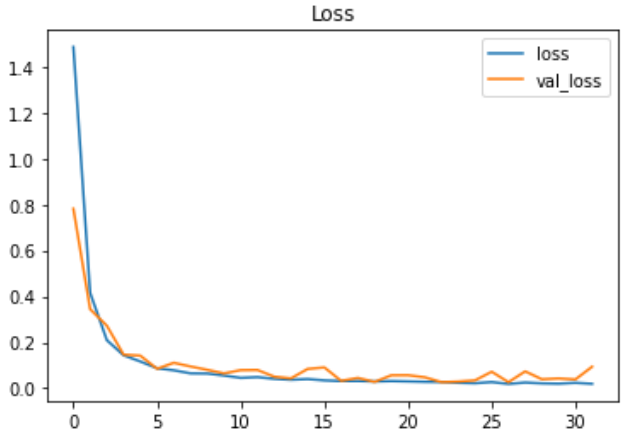

Figure 21 Val/Loss graph

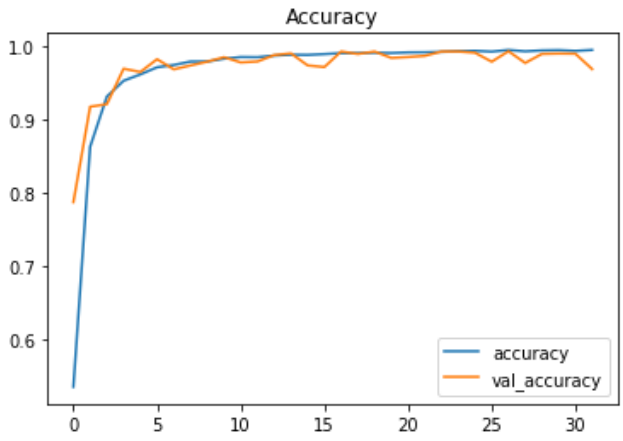

Figure 22 validation accuracy / accuracy

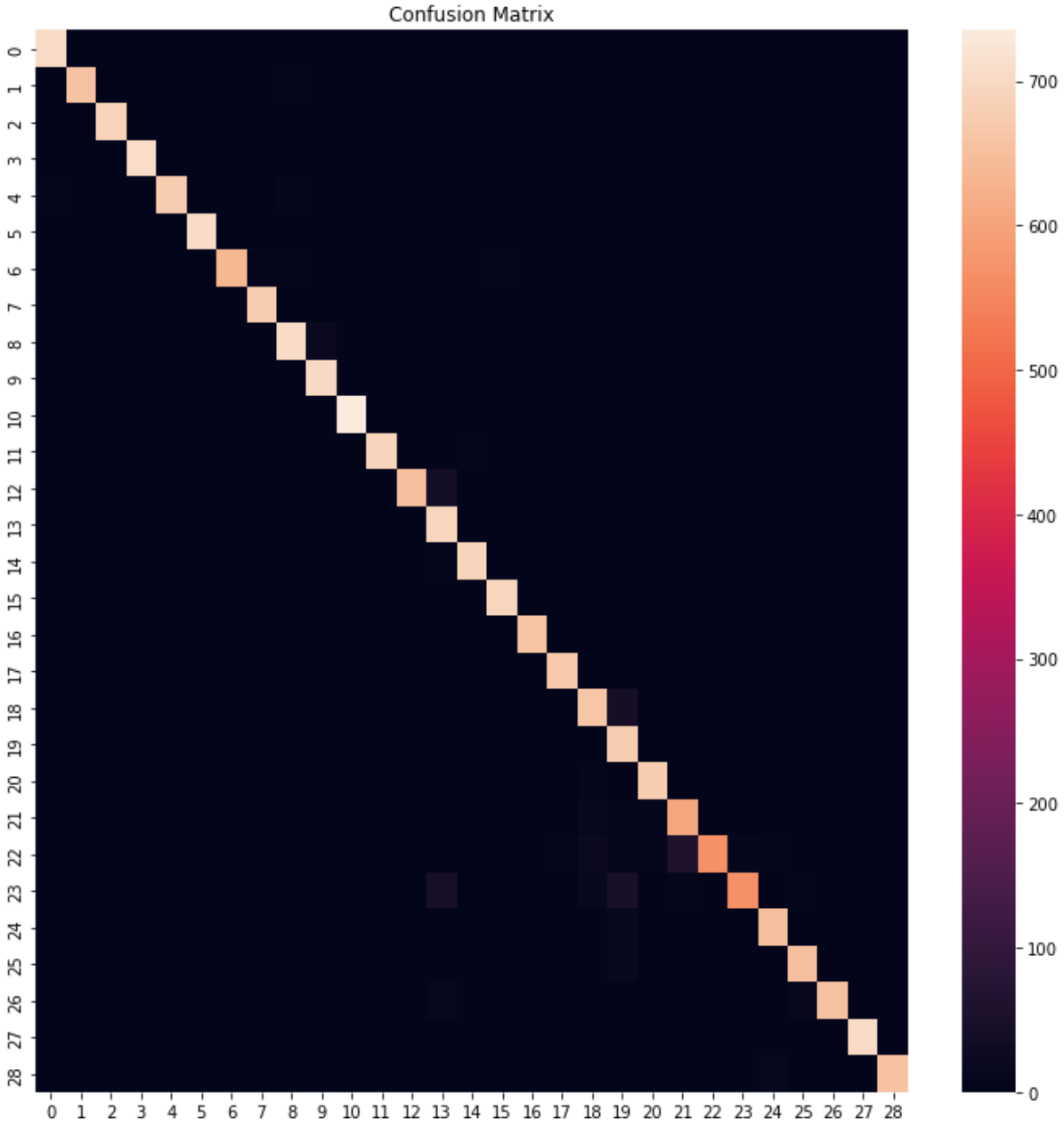

Figure 23 Confusion Matrix

	precision	recall	f1-score	support
1.0	0.98	1.00	0.99	708
2.0	1.00	0.99	0.99	664
3.0	1.00	0.98	0.99	704
4.0	1.00	1.00	1.00	711
5.0	1.00	0.96	0.98	697
6.0	1.00	1.00	1.00	705
7.0	0.99	0.97	0.98	654
8.0	0.98	1.00	0.99	674
9.0	0.96	0.97	0.97	723
10.0	0.96	1.00	0.98	699
11.0	1.00	1.00	1.00	738
12.0	1.00	0.98	0.99	707
13.0	0.98	0.95	0.96	688
14.0	0.86	1.00	0.92	693
15.0	0.98	0.98	0.98	702
16.0	0.98	0.99	0.99	699
17.0	0.99	0.99	0.99	666
18.0	0.98	1.00	0.99	671
19.0	0.92	0.93	0.92	709
20.0	0.81	1.00	0.90	677
21.0	0.98	0.96	0.97	697
22.0	0.90	0.94	0.92	642
23.0	1.00	0.82	0.90	689
24.0	0.97	0.82	0.89	692
25.0	0.96	0.98	0.97	665
26.0	0.97	0.97	0.97	669
27.0	1.00	0.95	0.97	688
28.0	1.00	1.00	1.00	702
29.0	1.00	0.97	0.99	677
accuracy			0.97	20010
macro avg	0.97	0.97	0.97	20010
weighted avg	0.97	0.97	0.97	20010

Figure 24 Classification Score

We see from the loss and accuracy graphs that the model is performing exceptionally well alongside the validation accuracy. We see from the accuracy graph that the validation accuracy is also looking exceptionally good, and almost on par with the training accuracy.

The confusion matrix shows near-perfect predictions with very few cases where the classes predicted outside of their true values. The light colours being the density of predictions, where the lighter the cell the more dense it is.

Finally, the classification report also shows a very high accuracy on many of the classes, except for T apparently which is the 20th class and has a precision of 81%. It would probably be wise to add a few more instances of T if this model was to be taken further.

These scores look fantastic but there is always the possibility that is it severely overfit due to it using the same person's hands to both train and test the model. More external testing needs to be done to validate the model completely.

4.6 Hyper Parameters

There are quite a few numbers we can tune when it comes to our model. Such as the input shape which determines the dimensions of the image we feed into the network. The higher the resolution the better the accuracy is likely to be TO AN EXTENT. A decent resolution will allow the model to pickup smaller details such as precise finger placement while being too high resolution will result in the algorithm picking up very small details such as wrinkles. So, the trick is to try and find the balance depending on how specific you need the model to be. After lots of testing, I found that 64 pixels were the lowest I could go before I started heavily sacrificing accuracy. Larger images would be great, but they require immense processing power.

Other variables such as pooling rates and dropout rates were considered and tested. The number of layers was originally too little, so an extra layer was added to help the model learn. Ultimately the parameters chosen were the ones documented under heading 4.4.

4.7 Memory Management

The following section demonstrates steps taken to build a model using proper memory management pipelines.

4.7.1 TensorFlow Data Input Pipeline

Originally when you fit a model, it will fit all the data at the same time, however, with a TensorFlow data input pipeline we can load the data in batches when we need it (TensorFlow, 2022).

The module allows us to load our data into a DataFrame type variable that works similarly to a panda DataFrame. The advantage of this is that we have access to the. map function which allows users to iterate through all the rows in the data and manipulate the information inside. This is also done in batches so that we don't use too much memory and can work with much larger images.

With the .map function, we can chain multiple custom methods together to create a pipeline for new data.

```
train_ds = train_ds.map(process_image)
.map(scale).map(expandDims).map(convertLabelToCategorical)

model.fit(train_ds)
```

4.7.2 Data Generator

The TensorFlow input pipeline is fantastic if you want to be completely customizable, but it lacks the efficiency of a normal pipeline which can be loaded into a single variable. For this functionality we can turn to the Data generator function which functions similarly to the TensorFlow input pipeline where it sends the data in batches, however, it uses pre-set parameters to manipulate the data or generate more before it is fed to the model.

```
dataGenerator = ImageDataGenerator(featurewise_center=False,
                                   samplewise_center=False,
                                   featurewise_std_normalization=False,
                                   samplewise_std_normalization=False,
                                   zca_whitening=False,
                                   rotation_range=10,
                                   zoom_range=0.1,
                                   height_shift_range=0.1,
                                   width_shift_range=0.1,
                                   horizontal_flip=True,
                                   vertical_flip=False)

dataGenerator.fit(X_train)

model.fit(dataGenerator.flow(X_train, y_cat_train, batch_size=32))
```

We fit the X_train data into the data generator and then 'flow' the data into the model, piece by piece after setting how we want to manipulate the data. For more information on the parameters please refer to the .ipynb file submitted.

4.8 Standard Pipeline

The obvious disadvantage of the data generator is that it wouldn't work very well on new data when you want an actual prediction. For this, a standard pipeline can be used to change multiple custom methods together to take new input and change it easily by fitting to a variable containing the pipeline.

```
imageConvertionPipeline = make_pipeline(ResizeImage((IMAGE_DIMENTION,
IMAGE_DIMENTION)), ConvertImageToArray(), ConvertImageToFloat(),
NormalizeImage(), ReshapeImage((1, IMAGE_DIMENTION, IMAGE_DIMENTION, 1)))

currentImage = cv2.imread(FilePath)
prediction = model.predict(imageConvertionPipeline.transform(currentImage))
```

4.9 Transfer Learning

Transfer learning allows you to merge two similar models together to combine their abilities into one better model (Seldon, 2021). The obvious advantage here is that you don't need to waste time and resources training a model when you can simply download one that is close to what you want, and then just add your own smaller dataset to slightly augment the model to work with what you want.

For example, you might want to recognize a specific car like a supra. So, you download a model to recognize cars, and then add your own data to see if that car is a supra or not. The advantage here is your part of the model can assume that it is a car from the beginning.

Another advantage was utilized in this project as an extra. A model was found which was identical to mine and used the same dataset to train, however, it utilized all the images and the entire 224x224 resolution of the images. This means that someone else did the processing which I couldn't do. There would be no point, however, in combining our models since it's the same data.

4.10 Model Validation

The model performed extremely well on the testing set whereas via testing it got everything right. However, we still need to test the model with external data. Luckily it proved quite good at that too. As a volunteer signed out the word chalk, which the model was able to accurately predict.

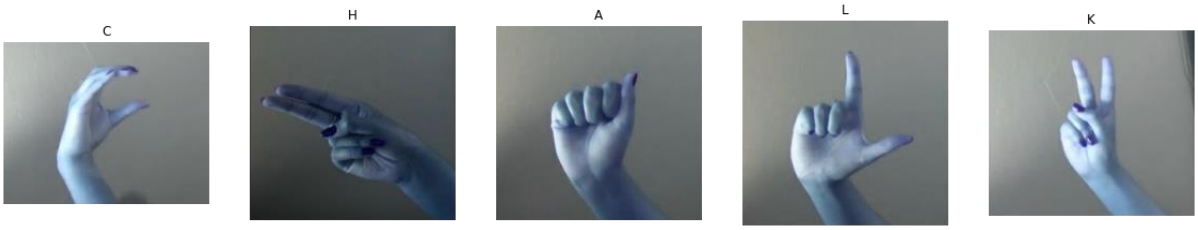

Figure 25 Model Testing

A small application was also built to allow people to upload their attempts of signing a letter to play with the model and test its capabilities.

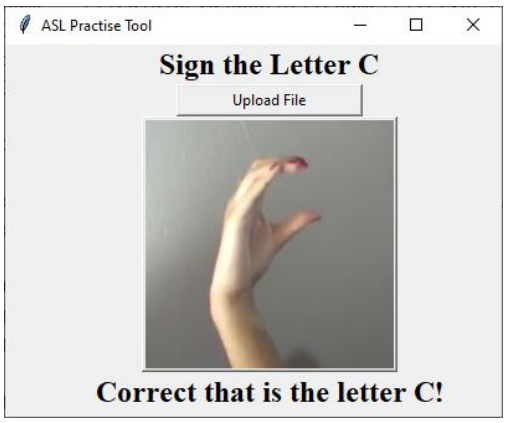

Figure 26 Application Test One

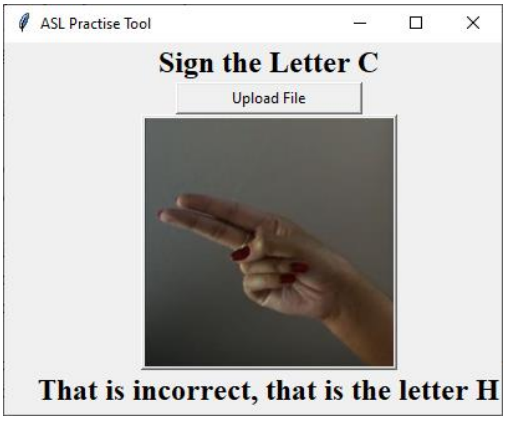

Figure 27 Application Test Two

4.11 Auto Correct

Finally, if the model finds itself trying to predict a word, it might get a single letter wrong, possibly due to human error. To combat this a very basic form of autocorrect was developed which compares the word predicted with every word in the English language to find the best fit.

```
def findClosestWord(word):
    closestWord = ''
    highestMatchPercentage = 0
    closestWordPercentage = 0
    for englishWord in english_words_set:
        if len(englishWord) == len(word):
            matchPercentage = Levenshtein.ratio(word, englishWord)
            if matchPercentage > closestWordPercentage:
                highestMatchPercentage = matchPercentage
                closestWord = englishWord
                closestWordPercentage = matchPercentage
    return closestWord, highestMatchPercentage
```

```
The original sentence is | wy treck brokr |
The corrected sentence is | my truck broke |
```

Figure 28 Auto Correct Prediction

5 Conclusion

The project began with the goal of creating a model that could accurately take a picture of a user's hand, who attempting to sign an ASL symbol, and predict what letter or concept they are attempting to convey. A dataset of over 87 000 images was selected which contained images of the entire English alphabet signed out over multiple distinguishing folders.

Image classification was conducted to try and achieve this goal using the dataset mentioned. The 87 000 images were augmented to over 100 000 with slight noise introduced to some of them. Cleaning and exploratory research was conducted and eventually, it was decided that discarding the colour of the image would be more beneficial. Following the data wrangling, a convolutional neural net was constructed to take the inputted picture and then classify the sign. Multiple convolutional layers were used with hyperparameters and dropout to perform this task. Pipelines were constructed to not only allow for better memory management but also for the automatic augmentation of input data.

The result was a model that could accurately predict what the user was trying to convey with nothing more than images supplied.

6 Recommendations

The model is currently not as good as I would like it to be, as the dataset turned out to not be perfect for the situation. The model still works quite well but it would be an embarrassment to any company in its current state. My recommendation here to any company trying to take on this project would be to get a more diverse cast of hands for the training data. This will help it perform better in real life as the users of the application will be quite diverse as well. I would also recommend more data in general and if it wasn't for an assignment, I would recommend a slightly higher resolution than 64x64.

I also recommend justifying this model by saying it is a learning tool for anyone wanting to sell the algorithm. It could be argued that the model would work quite well as a tool for video messaging applications such as Zoom, which could allow the user to sign what they are saying, and the application will type it out for them. This isn't a terrible idea as it might save them time if they are very good at sign language, but in most cases simply typing on a keyboard would be faster and more reliable. However, when you say that the tool is used to teach people how to perform sign language then suddenly it becomes more appealing. It's hard to argue that an application that spells out letters and makes you sign them on the spot wouldn't be extremely helpful to anyone trying to learn it.

Another recommendation comes under the assumption that the full application is built and working well. I recommend that any company utilizing the application try and target primary or high schools when trying to sell the app. Many American schools offer ASL as a subject already, and I'm sure that many of them would be willing to buy a license for such a helpful product. It's also very likely that those children will go to their parents and nag them for their own copy of the software.

Lastly, when utilizing the algorithm in its current form, I would recommend trying to get only your hand in the picture. With further development a hand recognition algorithm could be utilized to find the hand initially but for now it will have to be done manually.

7 References

APSEA, 2022. *American Sign Language ABC and Number Stories.* [Online]
Available at: https://apsea.ca/home-learning/dhh/self-determination-advocacy/asl-abc-and-number-stories.html
[Accessed 19 November 2022].

Baktha, K. & Tripathy, B., 2017. Investigation of recurrent neural networks in the field of sentiment analysis. *International Conference on Communication and Signal Processing (ICCSP),* pp. 2047-2050.

Brownlee, J., 2017. *A Gentle Introduction to Transfer Learning for Deep Learning.* [Online]
Available at: https://machinelearningmastery.com/transfer-learning-for-deep-learning/
[Accessed 17 November 2022].

Brownlee, J., 2018. *A Gentle Introduction to Dropout for Regularizing Deep Neural Networks.* [Online]
Available at: https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/
[Accessed 18 November 2022].

Cote, C., 2021. *Predictive analytics.* [Online]
Available at: https://online.hbs.edu/blog/post/predictive-analytics
[Accessed 22 April 2022].

Huellman, T., 2022. *How to Build an Image Classification Dataset.* [Online]
Available at: https://levity.ai/blog/create-image-classification-dataset
[Accessed 22 November 2022].

Johnson, D., 2022. *Back Propagation in Neural Network: Machine Learning Algorithm.* [Online]
Available at: https://www.guru99.com/backpropogation-neural-network.html
[Accessed 22 November 2022].

Lu, D. & Weng, Q., 2007. A survey of image classification methods and techniques for improving classification performance. *International journal of Remote sensing,* 28(5), pp. 823-870.

Misra, S. & Jiabo, H., 2020. *Artificial Neural Network Model.* [Online]
Available at: https://www.sciencedirect.com/topics/engineering/artificial-neural-network-model#:~:text=ANN%20models%20are%20the%20extreme,input%2C%20hidden%2C%20and%20output.
[Accessed 17 May 2022].

Muller, A. & Guido, S., 2017. *Introduction to Machine Learning with Python.* 1st ed. United States of America: O'Reilly.

Saha, S., 2018. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way.* [Online]
Available at: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
[Accessed 21 November 2022].

Sans, S., 2021. *Sign languages around the world.* [Online]
Available at: https://www.hearinglikeme.com/sign-languages-around-the-world/#:~:text=American%20Sign%20Language%20(ASL)%20is,Thomas%20Gallaudet%20and%20Lau

rent%20Clerc.
[Accessed 19 November 2022].

Seldon, 2021. *Transfer Learning for Machine Learning.* [Online]
Available at: https://www.seldon.io/transfer-learning#:~:text=Transfer%20learning%20helps%20developers%20take,models%20in%20an%20iterative%20way.
[Accessed 21 November 2022].

Stobierski, T., 2021. *DATA WRANGLING: WHAT IT IS & WHY IT'S IMPORTANT.* [Online]
Available at: https://online.hbs.edu/blog/post/data-wrangling
[Accessed 20 November 2022].

TensorFlow, 2022. *data.* [Online]
Available at: https://www.tensorflow.org/guide/data
[Accessed 20 November 2022].

The Independent Institute of Education, 2022. *Data Analytics Module Manual ,* s.l.: s.n.