1. Definition

1.1 Project Overview

The project is based around Deep Learning of neural networks for image classification. This is an area that has been around for a long time but has seen monumental expansion and development in more recent years. This has been a result of the increase in computer processing power and in particular the introduction of Graphic Processing Unit's (GPU's). Deep learning is of particular interest to me because of its ability to adapt to a wide variety of use cases.

Social media applications have developed a use for deep learning to classify images uploaded to their platforms, for example recognising human faces. Snapchat is an excellent example of how social media applies this deep learning technology. When the application detects a human face, the feature allows the user to alter their face in real-time, for example by applying dog ears, which follow the face when the screen moves. Facebook currently uses this image recognition technology to detect faces, and suggest their names to 'tag.' Apple also use image recognition as a security mechanism to unlock their new phones.

This project will focus on how image classification can be utilized to detect and translate letters of the alphabet from American Sign Language (ASL) into written text. Solving this particular problem would be very beneficial to allowing people who use ASL to communicate with people who do not understand ASL.

A link to the dataset used in this project is included in the footnotes below. [1]

1.2 Problem Statement

The problem to be solved is two fold:

1. Allowing people who communicate using ASL, to communicate more easily with people who cannot interpret sign language. This will operate at a very simple level by taking an image of a letter in the ASL alphabet and translating it into text.
2. Providing a simple to use learning interface, which can detect and recognise someone signing and provide accurate feedback, thereby allowing people to more easily learn ASL.

This is a common problem in the real world and one that would be very beneficial to solve via classification.

[1] https://www.kaggle.com/grassknoted/asl-alphabet

1.3 Metrics

The evaluation metrics for the model developed in this project will be based on how many of the testing images are detected and classified correctly. The formula for this is given below:

$$Accuracy = 100 * \frac{No.of\ correctly\ classified\ testing\ images}{Total\ No.of\ testing\ images}$$

This accuracy will be derived from the testing dataset that will be generated out of the raw dataset used for this project. Due to the dataset having equal numbers of images for each category in both testing and training (balanced dataset), the accuracy metric will provide a truthful outcome of the developed model.

I will also input a defined list of images that spell out a particular sentence and inspect how accurately this can be translated into written text. This will provide additional testing evidence, so anyone reading the project can easily see the output of the model in written text.

2. Analysis

2.1 Data Exploration

The dataset for this project was sourced from Kaggle. This dataset contains 3,000 images for each letter of the ASL alphabet, along with the symbols for 'space', 'delete' and 'nothing'. It also contains a testing dataset with 1 image for each of the symbols mentioned above.

Below are two examples of the images from the dataset.

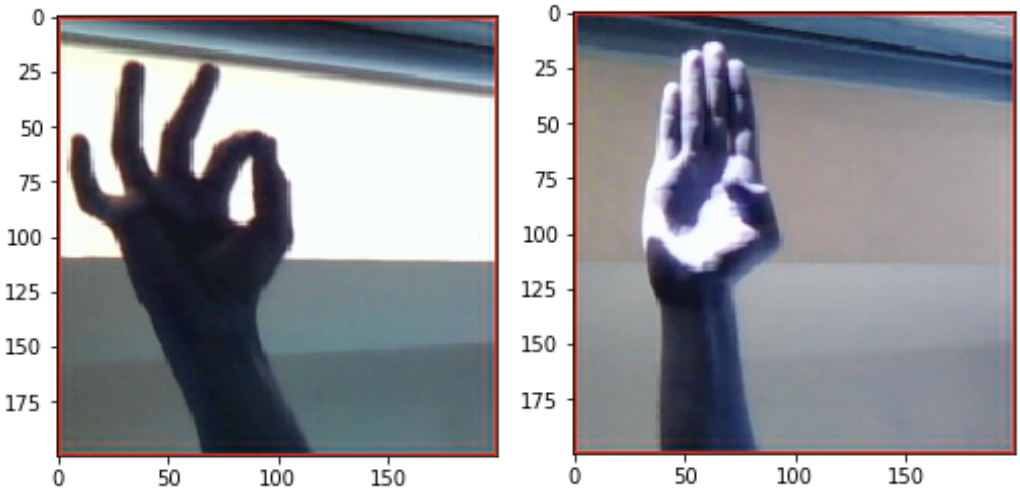

Fig. 1 – Sample images from the dataset

For a lot of image visualization projects it is necessary to resize the images before using them for training/analysis. However given that the images in the dataset used for this project are 200x200 pixels in size, it is not necessary to preform and size transformation.

The number of training images is much greater than required for my analysis but the training dataset provided was not sufficient. As a result of this, I split the training set up into three categories. I also included a selection of images, which spell out the sentence 'Learning ASL will now be easy'.

1. Training data
    - 4,988 total training images
    - 172 images for each image category (26 letters of the ASL, plus 'space', 'delete' and 'nothing').
2. Validation data
    - 406 total validation images
    - 14 images for each image category
3. Testing data
    - 406 testing images
    - 14 images from each category
    - 29 images that spell out the sentence 'Congratulations on learning ASL'.

2.2 Exploratory Visualization

From the graphs below we can see that the number of training, validation and testing samples for each image classification are equal therefore the dataset is balanced.

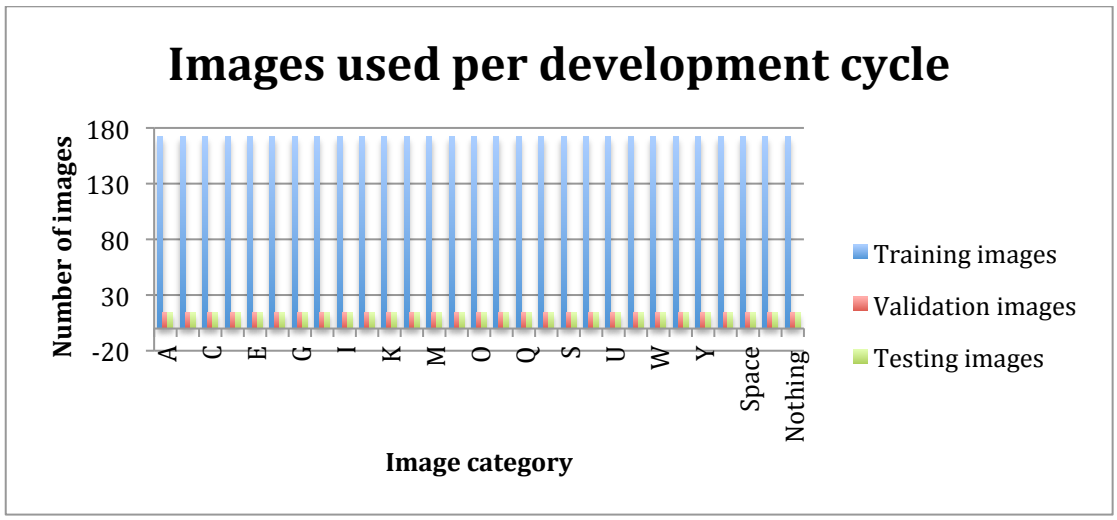

Fig. 2 – Graph showing the number of images used for each category during training, validation and testing.

2.3 Algorithms and Techniques

The classification technique used in this model is a Convolutional Neural Network (CNN) with the aide of Transfer Learning.

Transfer learning involves taking a pre-trained CNN to help develop a highly accurate model without having to train the neural network from scratch, thus saving significant amounts of training time. The model that I chose as a starter for my CNN was the VGG_16 model with the final three layers removed.

The neural network details used were as follows:

1. VGG_16 model with final three layers removed
    a. Bottleneck features are calculated from this truncated VGG16 model by passing the images to it as 4D tensors
2. Convolutional layers with the following changeable parameters:
    a. Number of filters
    b. Kernel size
    c. Padding
    d. Activation
    e. Input shape
3. Max Pooling Layers with the following parameters:
    a. Pool size
4. Global average pooling layer
5. Fully connected dense layer with:
    a. Specified number of targets (matching the number of categories of images to classify)
    b. Activation parameter

The network compiler contained the following:

1. Error loss function
2. Optimizer
3. Metrics

For training the model the training parameters were set to:

1. Number of epochs (training runs)
2. Batch size (number of images processed at any one time)

The reason for adding an extra convolutional layer was to allow the model to train on images, similar to those in the testing set. The VGG_16 model itself would not have been trained on ASL images previously and while it would be excellent at predicting certain traits in images it is important to carry out further training to allow it learn the specifics of the images we wish to classify.

The final fully connected layer contains the same number of outputs, as there are categories of image. The outputted result for each image will be a list of probabilities mapping to the confidence the model has that the tested image is each of the possible targets, with the sum of all the probabilities equaling 1.

2.4 Benchmark

Due to the complexity of comparing results from generated from two different datasets, I created my own benchmark model to compare my final model against. The benchmark model was a CNN, which I built from scratch and generated some statistics on how accurate the model was.

This benchmark model was then used as a reference when I developed my final model, which used transfer learning to build more a complex and more accurate model.

The neural network used for my benchmark model contained the following:

1. Four convolutional layers containing:
    a. 16, 32, 64 and 64 filters respectively.
    b. Kernel size = 2
    c. Padding = ‘sane’
    d. Activation = ‘relu’
2. Four max pooling layers, one following each of the convolutional layers from step 1.
3. One global average pooling layer.
4. One fully connected dense layer.

The compiler and training method used was the same as in section 2.3 above. When the model was trained it was then tested with the testing data and produced an accuracy of 38%. While this is reasonably accurate given that random guessing would theoretically produce an accuracy of 3.45%, it is clearly not sufficient enough. This model is also quite slow to train as each epoch has a training time of 15 seconds. This is why I chose to use transfer learning to aide in creating a more precise model.

3. Methodology

3.1. Data Preprocessing

To process the images for training and testing I needed to convert them from 2D images to 4D tensors. The steps to carry out this data pre-processing were as follows:

1. As previously stated, the dataset used in this project was sourced from Kaggle. It was stated that all images were of the dimensions 200x200 pixels however just as a precaution I resized the image to these dimensions again.
2. The 200x200 pixel image was then transformed into a 3D tensor of the form (200, 200, 3). The 3 in this section, represents the three colour channels (RGB).

3. The images were then transformed to a 4D tensor with shape (1, 200, 200, 3) where the one simply implies one image.
4. The 4D tensors are then divided by 255 to standardize all values to have a value between 0 and 1. This is due to RGB values being exactly 8 bits in size, ranging from 0 to 255 or (00000000 to 11111111 in binary). Thus dividing all values by 255 ensures the outputs are all between 0 and 1.

3.2 Implementation

The implementation can be sub-divided into two main phases. These are:

1. The development of the CNN model.
2. Implementing the model (compiling and training).

For phase one of this process, I previously detailed in section 2.3 that the starting step would be to take the VGG_16 model and remove the last 3 layers. This then allows us to pass the 4D tensors of the training data through this truncated model to calculate the bottleneck features.

The diagram below shows the lifecycle of the final model and the parameters that were tuned at each step.

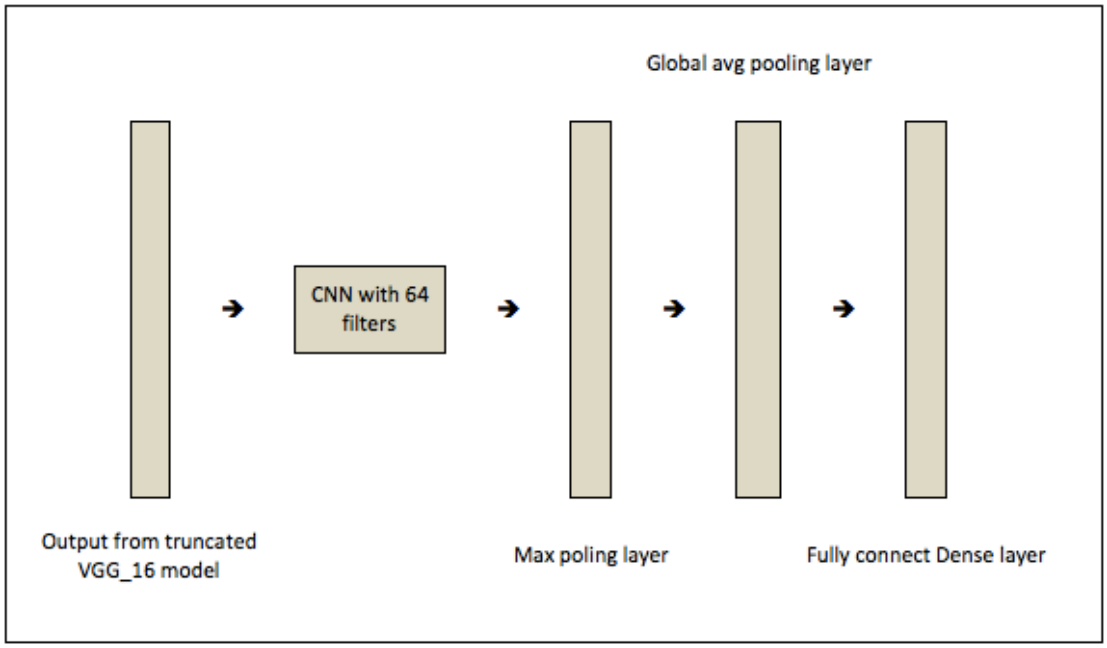

Fig. 3 – High-level architecture diagram of the model used in this project.

Details of what occurred at each step in the diagram above are as follows:

a. Sequential model defined
b. Convolutional layer added with:
    i. 64 filters

    ii. Padding = same
    iii. Activation = relu
    iv. Input shape = (6, 6, 3) -> Shape outputted by passing the train tensors through the truncated VGG_16 model.
c. Max pooling layer with:
    i. Pool size = 2
d. Global average pooling layer
e. Fully connected dense layer with:
    i. 29 outputs
    ii. Activation = softmax

For phase 2 of this process the parameters set for the compiling and fitting of the model were:

a. Error loss function = categorical_crossentropy
b. Optimizer = rmsprop
c. Metrics = Accuracy
d. Epochs = 30
e. Batch size = 10

3.3 Refinement

While trying to find the optimal solution for this project it was necessary to fine-tune certain parameters of the CNN model throughout. The 'ModelCheckpoint' function from keras help with fine-tuning itself when compiling the model, as it's possible to save only the best weights that are calculated.

The model that I developed has 133,021 trainable parameters but each training run (epoch) only took approximately 3 seconds to complete. Adding more convolutional layers to the model would increase the time for each training run but not adding enough will not produce an accurate model so it's extremely import to strike the right balancer between ensuring the model learns enough but doesn't spend time learning more than required.

The other parameters, which were altered throughout the development process were:

a. Number of filters in the CNN layer added to the model. The final design has 64.
b. Activation, the final design used 'relu'.
c. Number of training runs (epochs), the final design has 30.
d. Batch size, the final design has 10.

Below we can see the parameters for some of my earlier designs with some statistics to accompany each test.

	Test 1	Test 2	Test 3	Final
Number of filters	32	32	64	64
Epochs	20	20	50	30
Batch size	5	20	20	10
Time	3	0.5	2	3
Accuracy	96%	96.5%	98	98.3%

Fig. 4 – Table showing parameters and outputs of several iterations of the model developed in this project.

4. Results

4.1 Model Evaluation and Validation

The model developed in the project was validated during the training process with the use of a data validation set, which was 8% the size of the training set. This validation was used during the fitting of the CNN model used in this project. During each training epoch, validation data is used to when calculating the validation loss. With this validation loss determined, the model is able to choose/save these weights (if the validation loss improved) or reject the weights (if the validation loss didn't improve).

From running the CNN model a larger number of times using different parameters it was clear that the validation loss didn't improve significantly when using more then 30 epochs so the final parameters used in the model are appropriate and justified in my opinion.

For the training data used in the project I used a large number of testing images from each classification in the dataset. I also took and uploaded images of people preforming the letters of the ASL alphabet to see how the model would perform to different sized data and the results were still extremely accurate. I used these uploaded images to translate a sentence from ASL into English text and the accuracy was 100%.

Due to the larger number of training images used, along side the uploaded images it is clear that the model is quite robust and will adapt quite well to the images it is provided with. This is largely due to the benefits of using transfer learning as the model is already well capable of detecting a lot of features even before I added some extra convolutional layers on top to train further on the training dataset used in the project.

The dataset used to test the model developed in this project was balanced meaning there were an equal number of testing images from each category used in the testing stage. As a result of this it's safe to say the results produced from the model can be trusted.

4.2 Justification

The results generated by the final model show significant improvement from those on the benchmark model, rising from 44% to 98%. This is an extremely positive result and clearly shows that the model was a good fit for the dataset.

Along with the testing dataset I choose a subset of images, which would spell out the work 'Congratulations on learning ASL'. These images were then passed into the model that was developed in this project and the output displayed. This test was 100% accurate which is a first step in creating and developing an interactive model to aide with the learning on letters of the ASL alphabet.

Using transfer learning is essential to achieving a high degree of accuracy in image recognition as it greatly reduces the training time that would be required to develop a CNN model from scratch (as carries out for the benchmark model).

I believe the final solution/model is significant enough to have solved the problem of classifying letters of the ASL alphabet, however further training would likely be required to go to the next step and begin to classify words of the ASL. This is a much more complex problem to solve as certain words involve movement which cannot be captured in a single image.

5. Conclusion

5.1 Free-Form Visualization

When looking deeper into the results there are some interesting patterns, which would be explored further. From the results section above we know that the model is 98% accurate when evaluated on the testing dataset, however I thought it would be interesting to look at the accuracy of the individual categories. The graph below plots this:

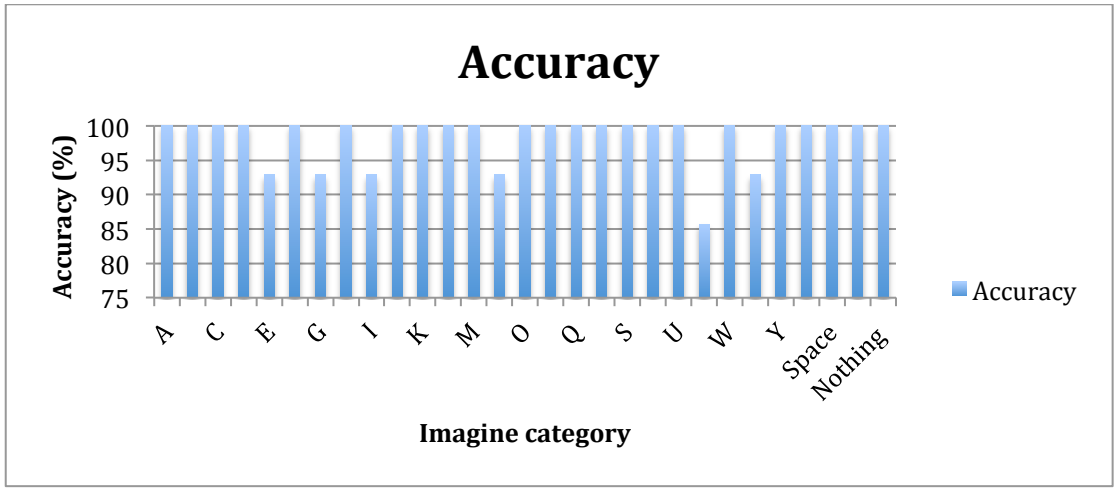

Fig. 5 – Graph plotting the accuracy of each image category.

In the graph above, there is one image category, which is 85.71% accurate, five, which are 92.86% accurate, and the rest are 100% accurate.

If we look at two of the letters (G and H), these had accuracies of 92.86% and 100% respectively. An example of these images can be seen below.

Fig. 6 – Image on the left is ASL for G and image on the right is ASL for H.

Despite the similarity between the images the accuracy is still very high, although we can see how it one might be mistaken for the other or visa versa.

5.2 Reflection

This project was overall very interesting as I was able to take a real-world problem and apply a machine-learning algorithm to it and see the visual results produced by my work. This project could be very beneficial to people by providing an interactive way to learn ASL, which is an exciting possibility.

Some aspects of the project were difficult by times yet enjoyable and a great learning experience. For example finding a suitable dataset, researching different Machine Learning and in particular researching different CNN's were all very beneficial for furthering my overall Machine Learning knowledge. Some aspects of the project were less interesting which is the case with any project. An example would be the adjusting of certain parameters over and over again to ensure the optimum solution was achieved. While this is not the most exciting part of the project, seeing the highly accurate end result makes it a very worthwhile exercise nonetheless.

The final model turned out to be extremely accurate for the given problem, which I was expecting although the accuracy achieved was slightly higher than what my initial expectations were when starting out. This high accuracy shows that the model is able to detect differences even in an image dataset, which is overall very similar as the images are all hand positions.

5.3 Improvement

Improvements can always be made to every machine-learning model and of course the model used in this project is no exception. Making the model more efficient might improve the training time required which will be increasingly beneficial when the training dataset grows significantly in size.

In order to see improvement in the model developed in this project it would likely be necessary to increase the number of target classifications. One option would be to add number targets alongside the letters of the ASL alphabet. With a larger dataset there may be a drop in the accuracy level, so further work could be carried out to improve the model again further.

The other are which would be interesting to investigate would be given the 100% accuracy in 23 out of the 29 categories, it might be useful to train less on these images and more on the less accurate images. This might increase the overall accuracy of the model without impacting on the training time.