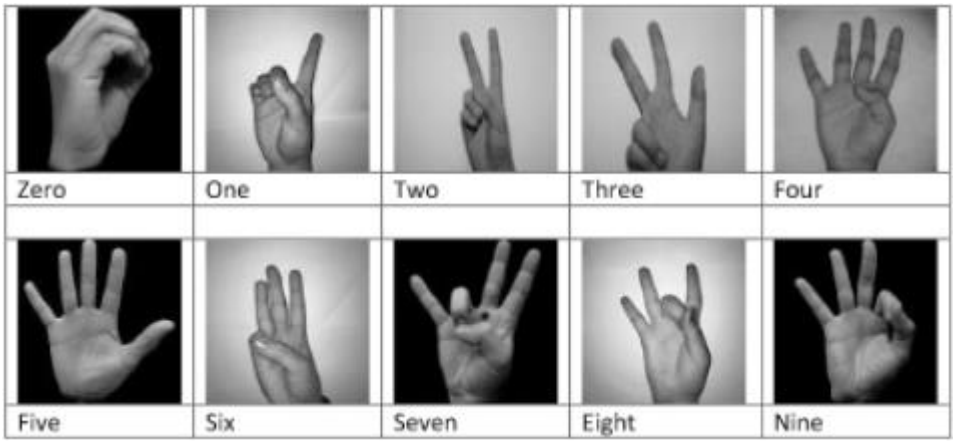
Model Developed & Experimentation

The dataset provided filled with categorical training, test and validation folders ranging from 1-10 digits include a total of 2764 images and are all noticeably grey scaled. An example subset of the dataset displayed *[Fig. 01]* are images to target throughout the development of both CNN models.



[Figure 1: Example images from dataset]

Loading up images from this dataset into our designed models are done through Image Data Generators. The purpose of this is primarily for simple use for both storing and loading data sets within the model, and being able to apply methods for the 2nd model with ease.

A Convolutional neural network (CNN) is statistically the most commonly interpreted deep learning methods to handle visual images (static or dynamic) *[Ref. 01*]. It involves features with input data, 2D convolutional layers resulting in this architecture being perfect for processing 2D media. The architecture contains filters that are usually manually implemented in other systems. CNN works by extracting features directly from images during its training on a dataset resulting in a highly accurate models for image classification. The operations of a CNN model involve *[Ref. 02]*:

- **Convolutional Operation**: The first and most important step in functioning a CNN. Focuses on extracting important features from an input. When an image is provided as an input, its placed in a form of matrices of pixels. As our dataset is greyscale, each imaged matrix ranges from 0-255 which can be normalised to values of 0-1 (0 as white, 1 as black).

- **Pooling**: To avoid incorrect output due to conditions that may affect the accuracy of predictions, pooling reduces the object sizes, preserves features and handles distortions.

- **Flattening**: Converts the pooled feature map into a vectorized format*[Ref. 03]* that can then be passed through the neural network to have it processed later on.

- **Activation**: After training, with the use of SoftMax classification, prediction accuracy of objects in the image can be provided.

- **Fully connected layers**: All operations regarding recognition and classification are performed on this layer. This connects all neurons to every other neuron in the previous layer.

Baseline CNN Model Overview (Model 1)

The structure of the baseline CNN model involves*[Fig. 02]*:

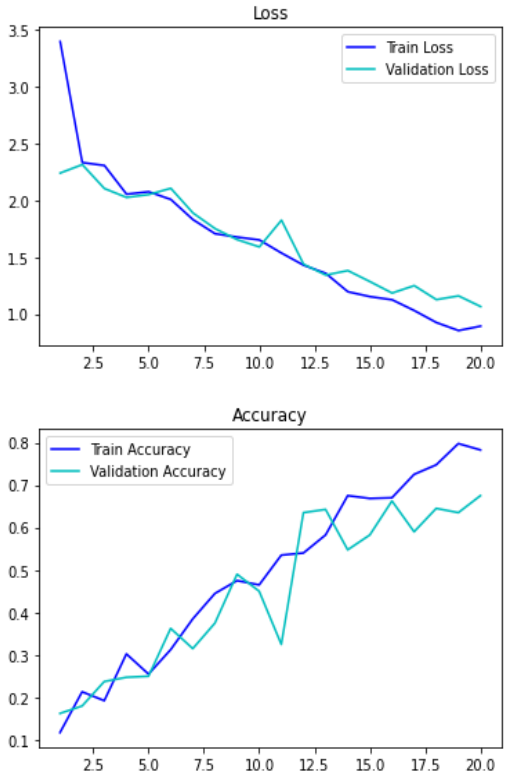
- Layers consisting of Conv2D, MaxPooling2D, Flatten and Dense layers for classification.
- It's input layers follows a 150x150x3 in respect to the dimensions of the input images (size of frames).
- Contains a single convolutional layer altogether (Conv1).
- Convolutional Filter dimension for each layer is 3x3.
- Filters for each layer: Conv1 – 32.
- Activation Operation for each convolutional layer is Relu (Rectified linear units).
- Max pooling applied after each filter of 2x2.
- After all convolutional layers have been applied, flattening is initiated for classification.
- After flattening, Output Dense layers are applied. Kernels for each dense layer: 128 (Relu activation), 10 (Softmax Activation – The best optimal gradient descent activation for categorical data).
- For compiling the model:
    - Loss function set as 'Categorical_Crossentropy' as there are multiple classifications of digits.
    - Optimizer function set as 'RMSprop' as a default method.
    - Metrics set for accuracy.
- Contains a total number of 1,088,042 parameters for training.

```
#Model

from keras import models, layers
from tensorflow.keras import optimizers

network = models.Sequential()
network.add(layers.Conv2D(32, (3, 3), activation="relu", input_shape=(150,150,3)))
network.add(layers.MaxPooling2D(2, 2))

network.add(layers.Flatten())
network.add(layers.Dense(128, activation="relu"))
network.add(layers.Dense(10, activation="softmax"))

network.compile(loss="categorical_crossentropy", optimizer=optimizers.RMSprop(lr=1e-4), metrics=["accuracy"])

network.summary()
```

[Figure 2: Implementation of Baseline CNN model]

Results from baseline model*[Fig. 03]*:

- **Training Loss** – 0.892, **Training Accuracy**: 78.3%. (Incline in accuracy, Decline of loss)
- **Test Loss** – 1.3078, **Test Accuracy**: 69% (Incline in accuracy, Decline of loss)




```
test_loss, test_accuracy = network.evaluate(test_generator, verbose=2)
print ("Overall accuracy: {0}".format(test_accuracy))
```

```
15/15 - 1s - loss: 1.0378 - accuracy: 0.6899
Overall accuracy: 0.6898954510688782
```

[Figure 3: Baseline Model Training / Validation / Test Loss & Accuracy]

Reflection of Baseline Model

Now that a baseline CNN model has been developed outcoming in increasing accuracies along with decreasing losses, there are various methods to introduce into the model that can substantially improve the model overall. The adjustment of parameters for CNN training may be necessary throughout the experimentation phase as within the baseline model, it cycled across 20 epochs with 20 steps per epoch vastly improving per cycle. *[Ref. 04]*Throughout the experimentation phase, the main consideration will be hyper tuning parameters to what will optimise and improve the accuracy overall for the model. *[Sathwick]*Hyper tuning parameters consist of modifying the general parameters of the model to appropriate values that result in improvement which are; number of neurons, convolutional layers, activation function, optimizer, learning rate, batch size, and epochs. An unexpected result from the baseline model, was the non-existent use of under or overfitting but in fact resulting sensibly. With the addition of methods and hyper tuning parameters, I do expect overfitting to occur within any of the models throughout experimentation but I also intend to overcome overfitting and achieve the best outcome.

Methods for improvement

Data Augmentation

A popular technique largely used to enhance the operations of a CNN training model *[Alex Hernandex-Garcia]*. By synthetically expanding the data sets by applying transformations and modifying properties of the dataset, resulting in improvement for generalisation and regularisation for models. By recognising samples in ways, the model has never seen before, augmentation is inevitable to improve prediction accuracy. Since augmentation increases the size of the training dataset, it's considered one of the best techniques for reducing overfitting. The use the image data generator that I decided to implement when loading the datasets, ImageDataGenerator can also be easily used for data augmentation. The transformations I'll be performing for augmentations on my datasets consist of; Rotation range, rescaling, width shift range, height shift range, shear range, horizontal flip, and fill mode.

Batch Normalisation

A normalization technique performed between layers that normalises activations in batches. It eliminates the problem of internal covariate shift *[Martin Riva]* which is the change in the input distribution of an internal layer of a CNN. This is expected to;reduce regularisation effects, improve the gradient flow, improves learning speed, able to handle large learning rates and overall allows CNN models to work efficiently.*[Kurtis Pykes]* There are two types of batch normalisation:

- L1: Eradicates the sum of absolute values of the weights
- L2: Penalises the sum of squares of the weights (also tends to shrink coefficients evenly)

Dropout

Dropout is considered as a regularisation technique and prevents overfitting within the model. *[Sathwick]* Dropout ultimately nullifies the contribution of some neurons towards the next layer and leaves all other neurons unmodified. Dropout does contain parameters when applying a dropout layer within the model which focuses on the rates of preventing. It's optimal to set these parameters at either 0.4 or 0.5.

Hyper-tuning Parameters

- Number of Convolutional layers and Max pooling pairs: represents the pairs of Conv2D and MaxPooling2D that we stack together to build the CNN. It's logically expected that the more pairs stacked, the greater the mode's ability to identify complex image patterns.
- Filter Maps: Research suggests that filter maps are optimally tweaked within the sets of 8-16-32, 24-48-92, 32-64-128.
- Filter Size: Adjusting filter sizes between 5x5 to 9x9 is experimental and may certainly improve the model's performance as deeper networks can detect complex patterns.
- Optimisers: Modifying the default optimiser and adjusting its learning rate may also take into effect of improve the model's performance. Studies display that the best overall default optimiser to use in comparison to RMSprop is Adam *[Lili Jiang]* as Adam accelerates the usage of the gradient descent algorithm by taking into consideration the 'exponentially weighted average' of the gradients.

Experimentation

For experimentation, I import various methods discussed earlier in this study onto my baseline model in order to construct the second model that recreates the most accurate model in comparison to the baseline model. A total of 15 models were produced, each resulting in vital intelligence for the next model that eventually leads to the final model *[Fig. 04]*.

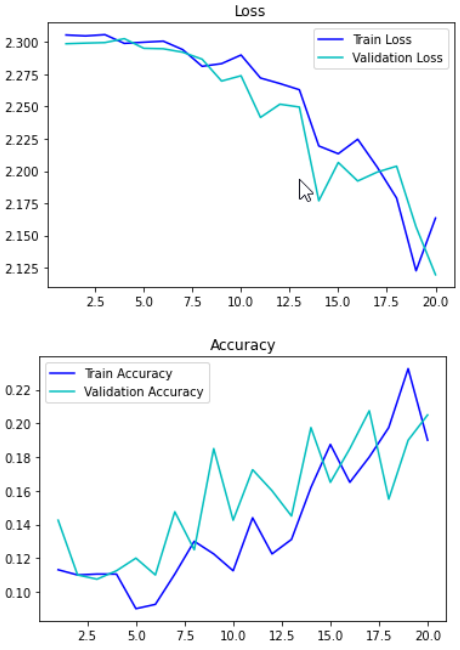
Note: All experiments ran across 20 epochs with 20 steps per epoch.

Model Num	Num Of ConvLayers	Augmentation	Filter Maps	Filter Size	Dropout	Batch Normalization	Learning Rate	Training Acc (1 d.p)	Test Acc (1 d.p
Baseline Model (1)	1	No	32	3x3	No	No	RMSprop	78%	69%
Model 2	4	Yes	32-64-128	3x3	No	No	RMSprop	23%	22%
Model 3	4	No	32-64-128	3x3	0.5	No	RMSprop	80%	85%
Model 4	3	No	32-64-128	3x3	No	No	Adam	97%	87%
Model 5	3	No	24-48-96	3x3	No	No	Adam	95%	87%
Model 6	3	No	8-16-32	3x3	No	No	Adam	95%	87%
Model 7	3	Yes	8-16-32	3x3	0.4	l1(0.0001)	Adam	24%	22%
Model 8	3	No	8-16-32	3x3	No	l1(0.0001)	Adam	90%	84%
Model 9	3	No	8-16-32	3x3	0.4	l2(0.0001)	Adam	94%	84%
Model 10	3	No	8-16-32	3x3	No	l2(0.0001)	Adam	100%	91%
Model 11	3	No	8-16-32	9x9	0.5	l2(0.0001)	Adam	97%	93%
Model 12	3	No	8-16-32	8x8	0.5	l2(0.0001)	Adam	98%	94%
Final Model	3	No	8-16-32	7x7	0.5	l2(0.0001)	Adam	100%	99%

[Figure 4: Information of experiments Table]

Model Number	Modifications (from baseline model)	Conclusion
2 [Fig. 05]	Data Augmentation	The first major adjustment of the model was the increase of convolutional layers spiking up to four. The surprising outcome displayed in its results [Fig. 05] presents an unexpected heavily overfitted model with both training and testing accuracies outcoming to 23% and 22% respectively. Performing augmentation on the model of ASL was futile but to ensure its uselessness, I intend to implement it once again under different model conditions (Model 7).
3 [Fig. 06]	Drop out (0.5)	Its outcome was underwhelming as there was a decreasing in both training and testing accuracies but by a minority amount (decreasing in 6% training, 2% testing). The reason for this result could be due to the irrational parameters set within the dropout call of 0.5, or is ineffective to cause a positive impact due to the conditions and input values of the model itself. Therefore, I will implement dropout under different model conditions to see if it once again portrays a negative outcome.
4 [Fig. 07]	3 Conv Layers Optimiser = Adam	This greatly impacted the results of the training data but remained static for the test data (Training at 97%, testing at 87%). My idea for this occurring was to do with the model overfitting which I hope will be solved when tuning the number of layers, filter maps, and filter sizes. The outcome of this experiment displayed that both the reduction of convolutional layers and the change of optimisers have positively affected the results. Therefore, these parameters will remain within the upcoming experiments.
5 [Fig. 08]	Filter map = 24-48-92	The idea of adjusting filter maps is inevitable to modify results in a positive or negative outcome *[Sathwick]*. The observations of models 5 & 6 both focuses on the modification of filter maps of 24-48-92 and 8-16-32 respectively. In comparison the filter map set at Model 4 (32-64-128), this experiment was about which model is expected to outcome the best which led to 32-64-128 performing the best in accuracies. The graph of model 6 presents minor overfitting but resulting greater in accuracies than model 5. It's suggested that having the lower filter map improves efficiency in performance for training. Therefore, in later experiments, I intend to continue the usage of 8-16-32 as its similar to outcome of 32-64-128 and more efficient.
6 [Fig. 09]	Filter map = 8-16-32	
7 [Fig. 10]	3 Conv Layers Data Augmentation Filter map = 8-16-32 Dropout Batch Normalisation l1 Optimiser = Adam	This model introduces a wide variety of differential conditions in comparison to what was originally set within the baseline model. Batch normalisation is introduced with the epsilon factor of 0.0001. I deemed it necessary to try augmentation and dropout as the model is different to what it originally was. The overfitting and underwhelming results portrayed in the results demonstrates that it's the outcome caused by augmentation as its similar to the disastrous results from model 2. Therefore, experiments from now on will exclude data augmentation.

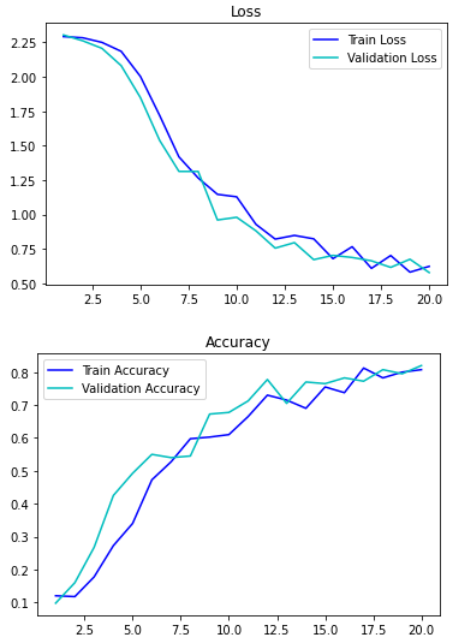
8 [Fig. 11]	3 Conv Layers Filter map = 8-16-32 Batch Normalisation l1 Optimiser = Adam	This experiment proves the optimisation and improvement from the use of batch normalisation. I will continue to test batch normalisation between l1 and l2 factors and expect to implement it within the final model.
9 [Fig. 12]	3 Conv Layers Filter map = 8-16-32 Batch Normalisation l2 Dropout 0.4 Optimiser = Adam	From this model, we can start to comprehend the factors that are beneficial to the model, and factors that are preventing improvement. With the use of l2 batch normalisation, I'm to assume that this was the cause that brought the training and test accuracies to as high as 94% and 84% and can possibly result in higher. Dropout may be the factor that holds back improvement of results which is what I'll be taking into account for the next few experiments.
10 [Fig. 13]	3 Conv Layers Filter map = 8-16-32 Batch Normalisation l2 Optimiser = Adam	Similar to model 9, with the removal of dropout – The results display substantial improvement with a training accuracy of 100% and testing accuracy of 91%. My understanding of testing accuracy being lower than training is due to model overfitting. To overcome this, input layers are to be tweaked.
11 [Fig. 14]	3 Conv Layers Filter map = 8-16-32 Batch Normalisation l2 Filter Size = 9x9 Optimiser = Adam	The appropriate modification for input layers would be to adjust the filter sizes. For this experiment, I decided to set it to an extreme amount (to 9x9) to see if it makes any noticeable improvements. The results indicate an improvement towards test accuracy meaning overfitting was occurring regularly throughout the experiments. As this filter size is unreasonable, the upcoming experiments are about adjusting its size to the right amount.
12 [Fig. 15]	3 Conv Layers Filter map = 8-16-32 Batch Normalisation l2 Filter Size = 8x8 Dropout = 0.5 Optimiser = Adam	With the adjustment of the filter size to 8x8. I believed it to be appropriate to reintroduce drop out at a 0.5 probability as its results in prior experimentation models could've been due to model overfitting. Since overfitting has been resolved, dropout along with 8x8 filter size portrays great results in most specifically, the testing accuracy. It's progress of loss and accuracy is corelating towards the training results as demonstrated in the graph.
Final Model [Fig. 16]	3 Conv Layers Filter map = 8-16-32 Batch Normalisation l2 Filter Size = 7x7 Dropout = 0.5 Optimiser = Adam Epochs = 80	Reducing the filter size from model 12 to 7x7 portrayed a perfect training accuracy of 100% and 98.6% for test accuracy. A test loss of 0.1 is arguably great in comparison to the baseline loss of 0.4. Both the loss of training and test correlate towards each other. Dropout of 0.5 deemed reliable to use in the environment of a non-overfitting model whereas data augmentation caused a great risk of overfitting. As this was expected to be the final model, to get the most accurate results from this model were to run a large amount of cycles leading to 80 epochs.




```
: test_loss, test_accuracy = network.evaluate(test_generator, verbose=2)
  print ("Overall accuracy: {0}".format(test_accuracy))
```

```
15/15 - 3s - loss: 2.1261 - accuracy: 0.2230
Overall accuracy: 0.2229965180158615
```

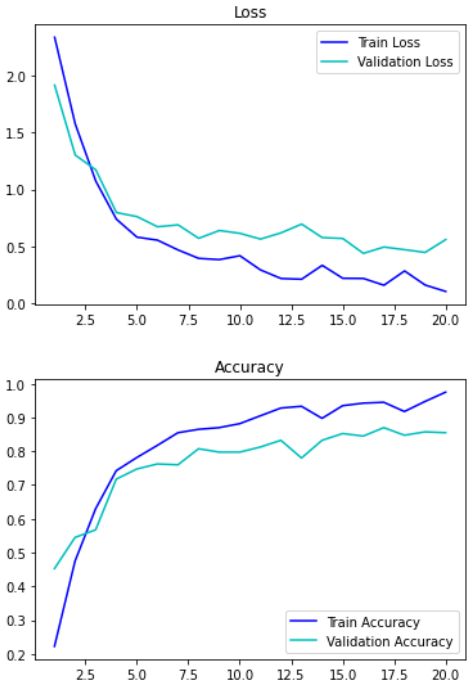
[Figure 5: Model 2 Results]




```
test_loss, test_accuracy = network.evaluate(test_generator, verbose=2)
print ("Overall accuracy: {0}".format(test_accuracy))
```

```
15/15 - 3s - loss: 0.5011 - accuracy: 0.8502
Overall accuracy: 0.8501741886138916
```

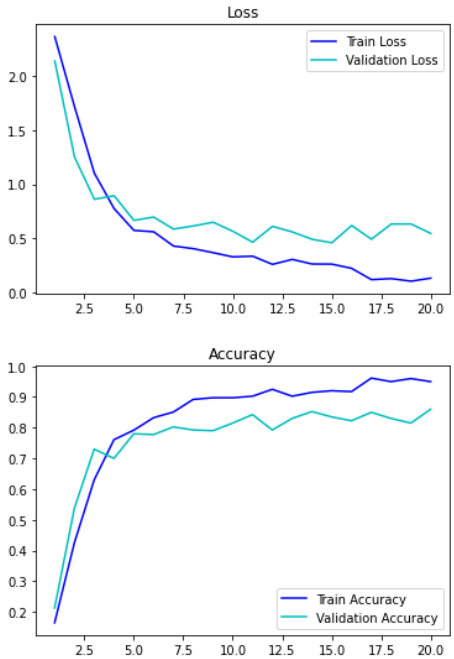
[Figure 6: Model 3 Results]




```
test_loss, test_accuracy = network.evaluate(test_generator, verbose=2)
print ("Overall accuracy: {0}".format(test_accuracy))
```

```
15/15 - 2s - loss: 0.5146 - accuracy: 0.8711
Overall accuracy: 0.8710801601409912
```

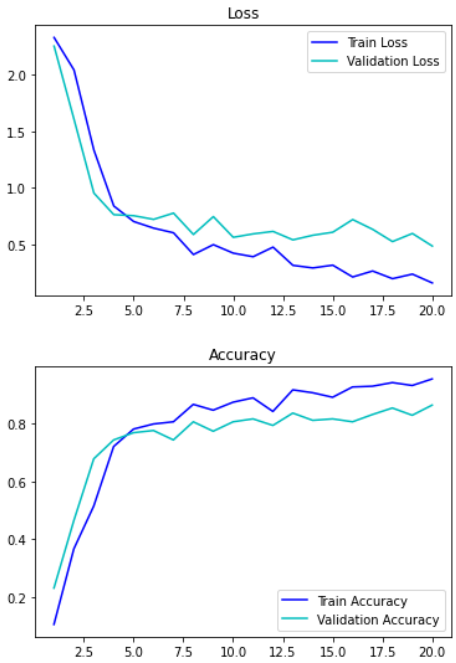
[Figure 7: Model 4 Results]




```
test_loss, test_accuracy = network.evaluate(test_generator, verbose=2)
print ("Overall accuracy: {0}".format(test_accuracy))
```

```
15/15 - 2s - loss: 0.4749 - accuracy: 0.8850
Overall accuracy: 0.8850173950195312
```

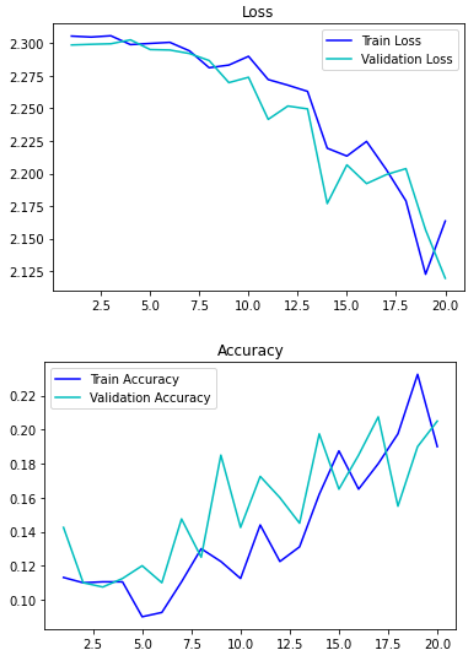
[Figure 8: Model 5 Results]




```
test_loss, test_accuracy = network.evaluate(test_generator, verbose=2)
print ("Overall accuracy: {0}".format(test_accuracy))
```

```
15/15 - 1s - loss: 0.4504 - accuracy: 0.8780
Overall accuracy: 0.8780487775802612
```

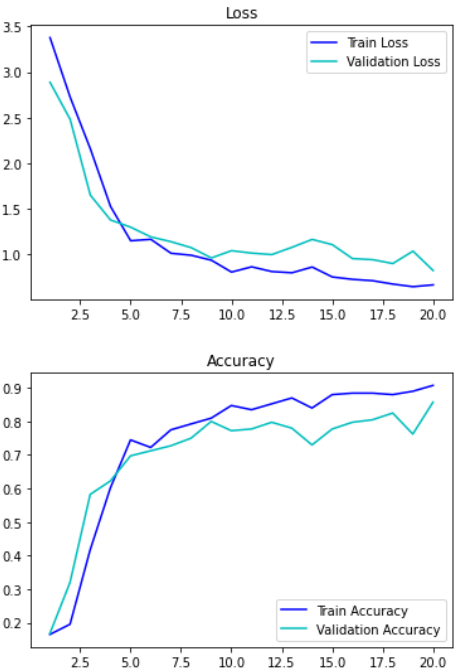
[Figure 9: Model 6 Results]




```
test_loss, test_accuracy = network.evaluate(test_generator, verbose=2)
print ("Overall accuracy: {0}".format(test_accuracy))
```

```
15/15 - 3s - loss: 2.1261 - accuracy: 0.2230
Overall accuracy: 0.2229965180158615
```

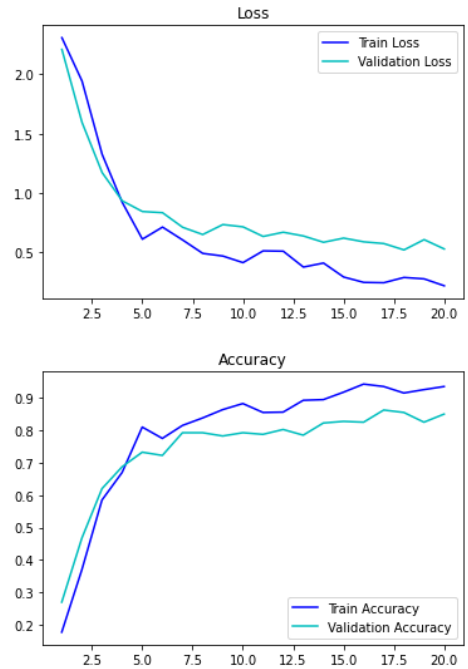
[Figure 10: Model 7 Results]




```
test_loss, test_accuracy = network.evaluate(test_generator, verbose=2)
print ("Overall accuracy: {0}".format(test_accuracy))
```

```
15/15 - 1s - loss: 0.8102 - accuracy: 0.8467
Overall accuracy: 0.8466898798942566
```

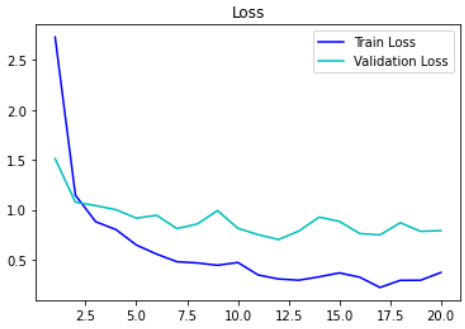
[Figure 11: Model 8 Results]

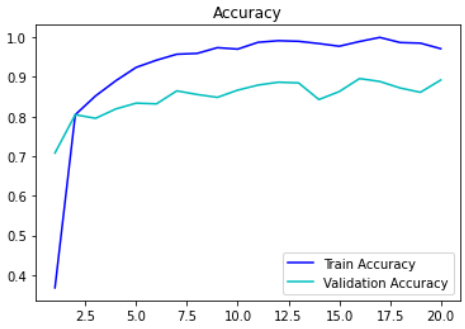



```
test_loss, test_accuracy = network.evaluate(test_generator, verbose=2)
print ("Overall accuracy: {0}".format(test_accuracy))
```

```
15/15 - 1s - loss: 0.5441 - accuracy: 0.8432
Overall accuracy: 0.8432055711746216
```

[Figure 12: Model 9 Results]

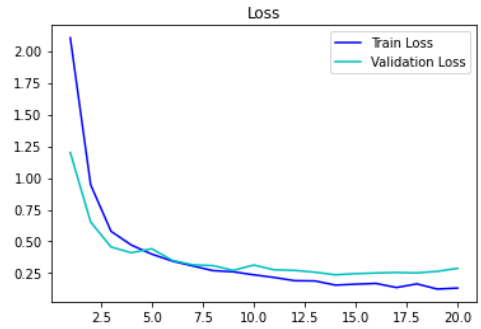


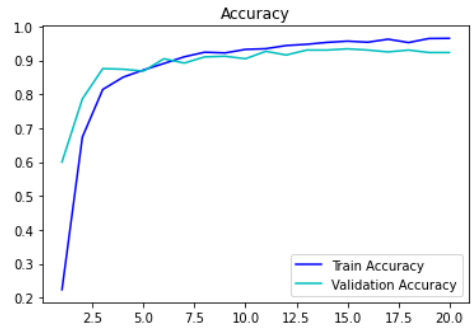



```
: test_loss, test_accuracy = network.evaluate(test_generator, verbose=2)
  print ("Overall accuracy: {0}".format(test_accuracy))
```

```
15/15 - 1s - loss: 0.7164 - accuracy: 0.9059
Overall accuracy: 0.9059233665466309
```

[Figure 13: Model 10 Results]

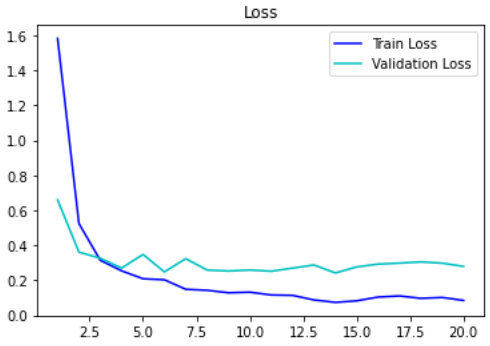


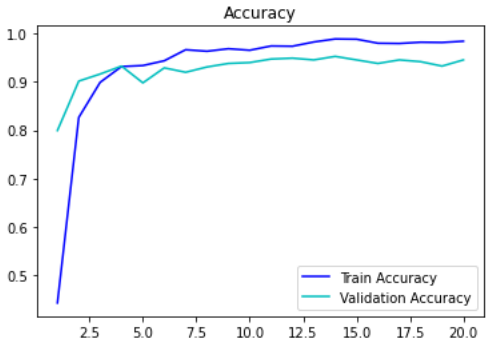



```
test_loss, test_accuracy = network.evaluate(test_generator, verbose=2)
print ("Overall accuracy: {0}".format(test_accuracy))
```

```
15/15 - 1s - loss: 0.2469 - accuracy: 0.9303
Overall accuracy: 0.9303135871887207
```

[Figure 14: Model 11 Results]






```
test_loss, test_accuracy = network.evaluate(test_generator, verbose=2)
print ("Overall accuracy: {0}".format(test_accuracy))
```

```
15/15 - 1s - loss: 0.1937 - accuracy: 0.9443
Overall accuracy: 0.9442508816719055
```

[Figure 15: Model 12 Results]

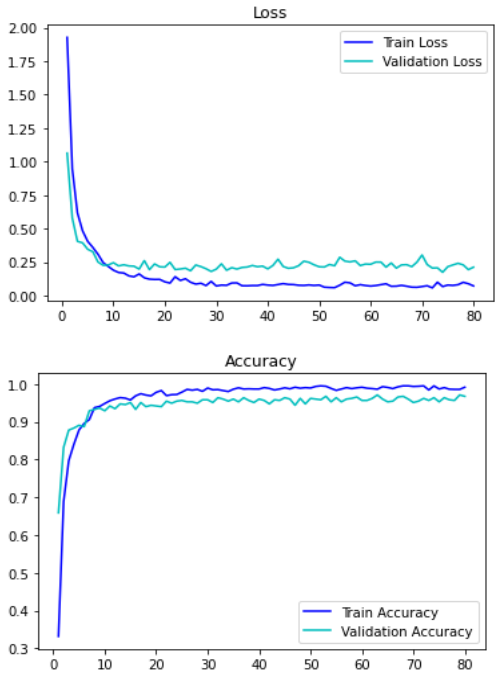
Final Model Overview

Final model results *[Fig. 16]*:

- **Training Loss**: 0.101, **Training Accuracy**: 100%
- **Test Loss**: 0.101, **Test Accuracy**: 98.6%

In comparison to the baseline model *[Fig. 03]*, The overall test losses are drastically lower by a difference of 0.9368 along with an increase in accuracy difference of 29.6%. Training accuracy is at a perfected 100% which is a substantial upgrade from the baseline accuracy of 78.3%. This alone establishes how superior the final model is compared to the baseline model.

Noticeable graph results establish the steep decline of losses in both the training and validation datasets as well as the steep incline in accuracy and then maintaining the overall accuracy of 98%. Whereas in the baseline model, there is a gradual decline in data loss and noticeable overfitting – eventually leading to its lowest data loss value of below 50%. The accuracy in the baseline model is also a gradual incline that eventually leads to an accuracy above 60%. The final model results are far more rational and preferred.




```
test_loss, test_accuracy = network.evaluate(test_generator, verbose=2)
print ("Overall accuracy: {0}".format(test_accuracy))
```

```
15/15 - 1s - loss: 0.1014 - accuracy: 0.9861
Overall accuracy: 0.9860627055168152
```

[Figure 16: Final ModelTraining / Validation / Test Loss & Accuracy]

Study Comparison

CNN Model	Accuracy	Loss
My 1st Model (Baseline)	69%	1.3078
My 2nd Model (Final)	98.6%	0.101
Vanita Jay's Model *[Ref. 10]*	98.58%	0.11

The inconsideration of inputs and intention of upholding a high-end accuracy result for the 1st model is the reason for all external models to outperform it. The primary difference between my models and the chosen comparison model of Vanita Jay is the implementation of datasets used. Vanita Jay's model does also consider the introduction of a 'real time performance, independence, scalability and robustness'*[Ref. 10]* for the algorithm due to the uncertain conditions the dataset may provide. My model was performed under a dataset containing 2764 images unlike Vanita Jay's model who most likely to have used a much larger volume for a dataset.

Our works presents a CNN that performs with high accuracy with minimal loss that outperforms existing methods such as Vanita Jay's. Despite the difference in quality and volume of datasets, the model outperforming is due to the difference in modification throughout experimentation. Vanita Jay discusses how their model can be 'improved by implementing a CNN to learn the filter size for each channel' as well as tuning hyper parameters which is what my model peruses through and evidently outcomes with improving accuracies with per experimented model.

Conclusion

Classifying the American Sign Language digits from 1-10 with the use of a CNN proposed to be an interesting study due to understanding the possible usefulness of this model that can be used in a variety of different ways to guide individuals with hearing impairment. Specifically, within the models developed, with the implementation of the variety of methods introduced and learnt to me - such as; Data augmentation, hyper tuning parameters, transfer learning, gradient descent, embeddings, Dropout, normalisation, etc – this was able to uplift and improve my final model's results to 98.6% accuracy with a loss of 0.101. These results are considered to be successful outcomes when compared to the inconsiderate development of the baseline model in which case resulted with 69% accuracy with 1.3078 loss which is unacceptable to be used in a real-world situation.

The final model was established to be the most accurate it could be from the baseline model with its extensive experimentation and maximisation of parameters. Throughout experimentation, the models did undergo overfitting regularly and I was able to distinguish the issue of improving the network design and modifying the input layers appropriately. Aside from this, a vital improvement the model can implement to improve overall results is to increase the volume usage of all training, validation and test datasets to allow the model to train more extensively and predict classification at a higher standard.

References

- [Ref. 01] MathWorks (2019). *What Is Deep Learning? | How It Works, Techniques & Applications*. [online] Mathworks.com. Available at: https://www.mathworks.com/discovery/deep-learning.html.

- [Ref. 02] OpenGenus IQ: Computing Expertise & Legacy. (2021). *Different Basic Operations in CNN*. [online] Available at: https://iq.opengenus.org/operations-in-cnn/.

- [Ref. 03] www.superdatascience.com. (n.d.). *SuperDataScience*. [online] Available at: https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-4-full-connection.

- [Ref. 04] Yeh, W.-C., Lin, Y.-P., Liang, Y.-C. and Lai, C.-M. (n.d.). *Convolution Neural Network Hyperparameter Optimization Using Simplified Swarm Optimization*. [online] Available at: https://arxiv.org/ftp/arxiv/papers/2103/2103.03995.pdf.

- [Ref. 05] [Alex Hernandez-Garcia] Hernández-García, A. and König, P. (2018). Further advantages of data augmentation on convolutional neural networks. *arXiv:1906.11052 [cs]*, [online] 11139, pp.95–103. doi:10.1007/978-3-030-01418-6_10.

- [Ref. 06] [Martin Riva] Riva, M. (2020). *Batch Normalization in Convolutional Neural Networks | Baeldung on Computer Science*. [online] www.baeldung.com. Available at: https://www.baeldung.com/cs/batch-normalization-cnn.

- [Ref. 07] [Kurtis Pykes]neptune.ai. (2021). *Fighting Overfitting With L1 or L2 Regularization: Which One Is Better?* [online] Available at: https://neptune.ai/blog/fighting-overfitting-with-l1-or-l2-regularization.

- [Ref. 08] [Sathwick] Sathwick (2022). *Sign Language Recognition using Deep Learning*. [online] Medium. Available at: https://towardsdatascience.com/sign-language-to-text-using-deep-learning-7f9c8018c593 [Accessed 13 Jan. 2023].

- [Ref. 09] [Lili Jiang] Jiang, L. (2020). *A Visual Explanation of Gradient Descent Methods (Momentum, AdaGrad, RMSProp, Adam)*. [online] Medium. Available at: https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c.

- [Ref. 10] [Vanita Jay] Jain, V., Jain, A., Chauhan, A., Kotla, S.S. and Gautam, A. (2021). American Sign Language recognition using Support Vector Machine and Convolutional Neural Network. *International Journal of Information Technology*. doi:10.1007/s41870-021-00617-x.