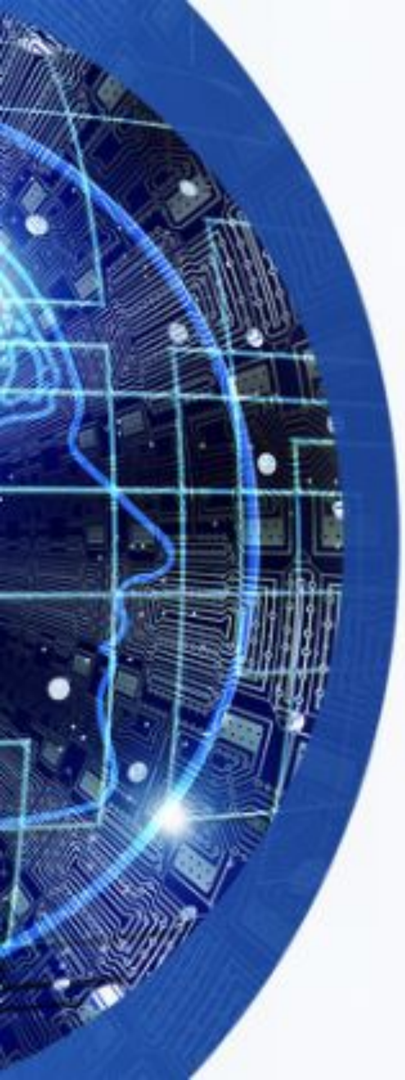
Ai-Spy

IMAGE RECOGNITION

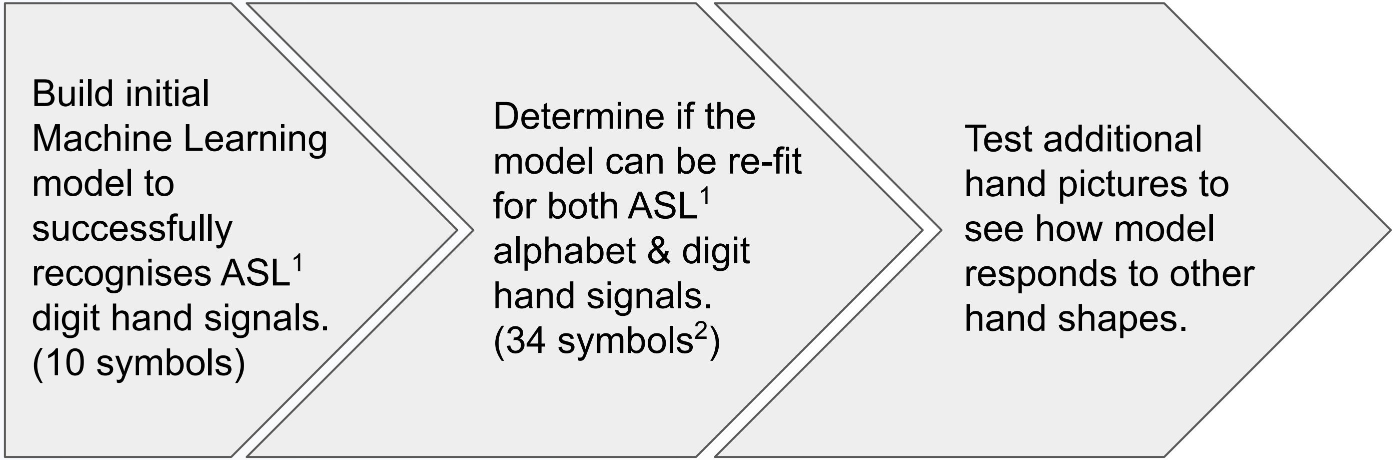
Project 2 Team:
Grace Ho
Pamela Moreno
Jay Richardson
Muhammad Hasnain Dosani

Project Goal

Train a machine learning model to successfully recognise American Sign Language digit and alphabet hand signals.



Methodology



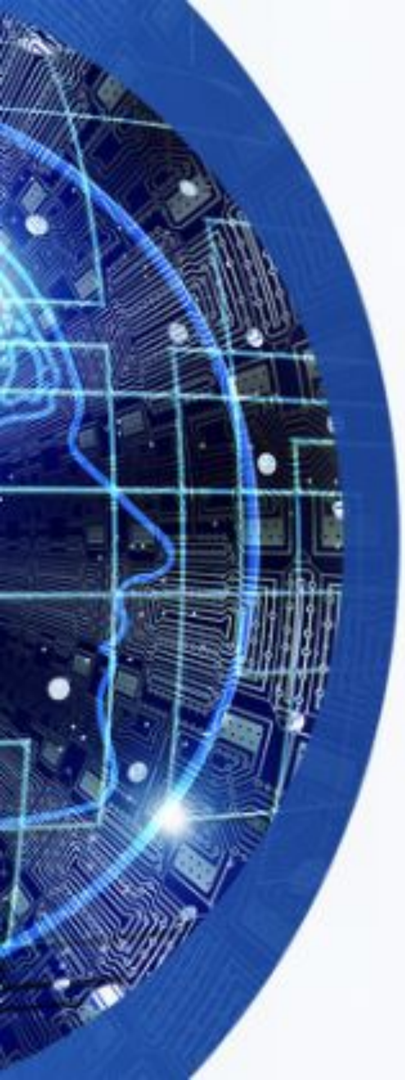
Build initial Machine Learning model to successfully recognises ASL[1] digit hand signals. (10 symbols)

Determine if the model can be re-fit for both ASL[1] alphabet & digit hand signals. (34 symbols[2])

Test additional hand pictures to see how model responds to other hand shapes.

[1] American Sign Language.

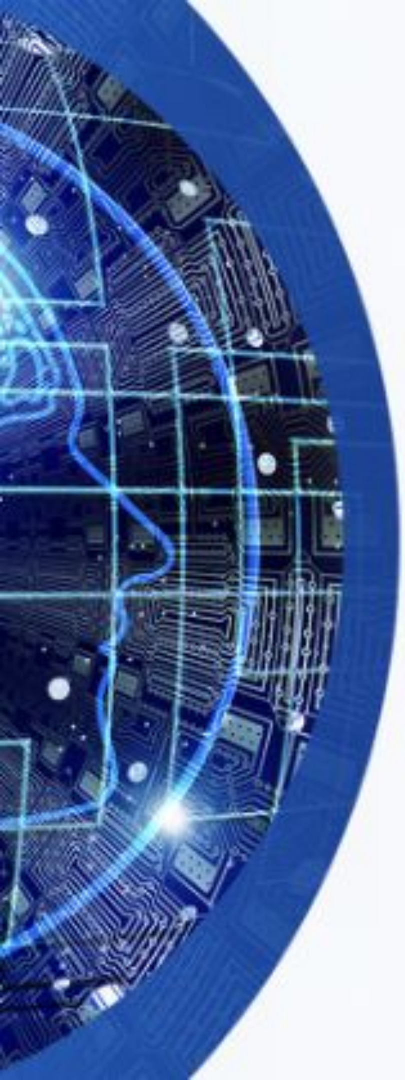
[2] Excluding characters J & Z which is movement based.



Digits Model

Dataset:

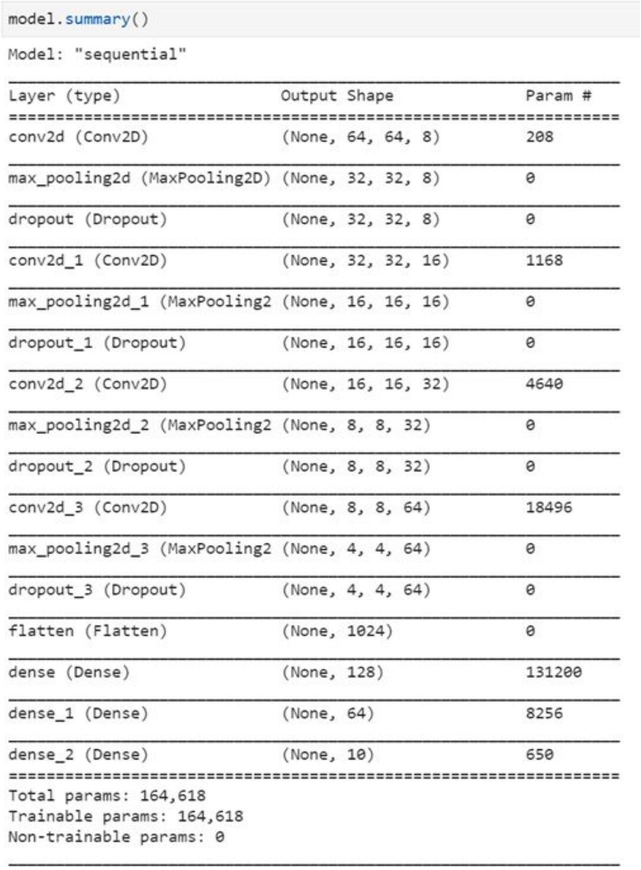
- 2062 images of ASL digit hand symbols
- 64 x 64 pixel black and white images, in numpy array (X & y files)
- 80/20 train vs test split



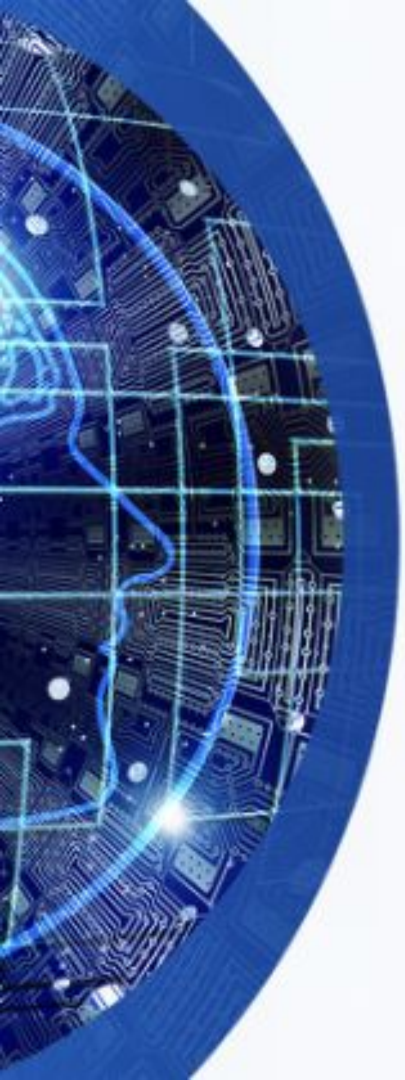
Digits Model cont.

Structure & compile used

- Keras Sequential model
- 16 layers: Convolutional 2D, MaxPooling2D, Drop out and Dense layers
- Relu activation, with Softmax final layer
- Adam optimiser, categorical_crossentropy loss



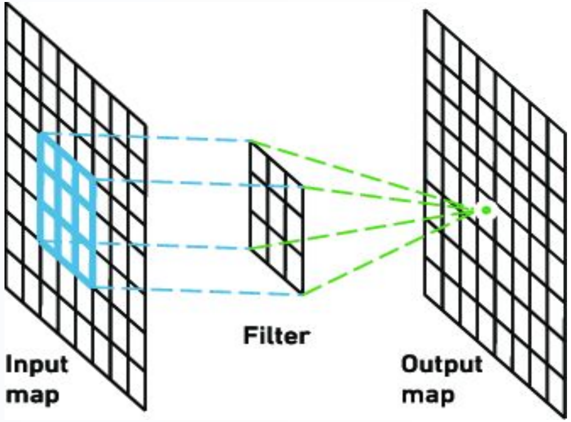
```
model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 64, 64, 8)         208       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 32, 32, 8)         0         
_________________________________________________________________
dropout (Dropout)            (None, 32, 32, 8)         0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 16)        1168      
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 16)        0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 16)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 32)        4640      
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 32)          0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 8, 8, 32)          0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 8, 64)          18496     
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 64)          0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 4, 4, 64)          0         
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0         
_________________________________________________________________
dense (Dense)                (None, 128)               131200    
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256      
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650       
=================================================================
Total params: 164,618
Trainable params: 164,618
Non-trainable params: 0
_________________________________________________________________
```

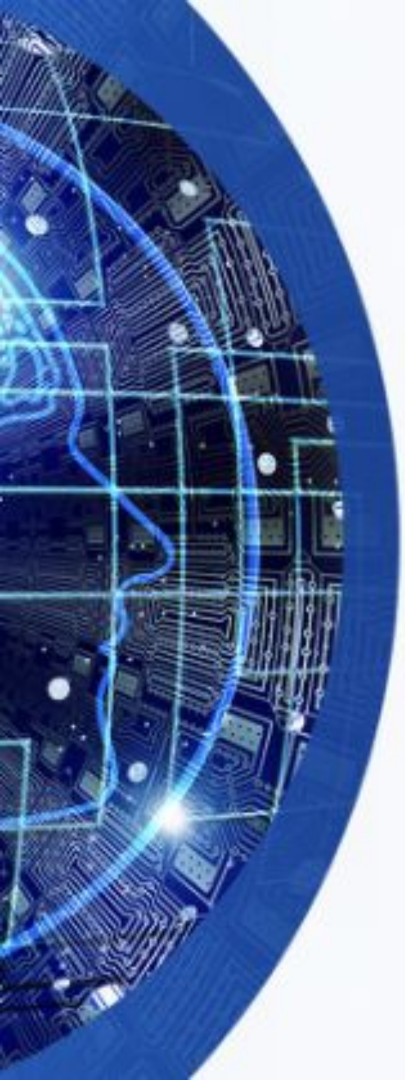


Digits Model cont. - CNN

A convolutional layer contains a set of parameters that need to be learned. Each filter is convolved with the input volume to compute an activation map.

The activation map is slid across the width and height of the input in a "pool size" set in the model architecture with the weights computed at every spatial position.

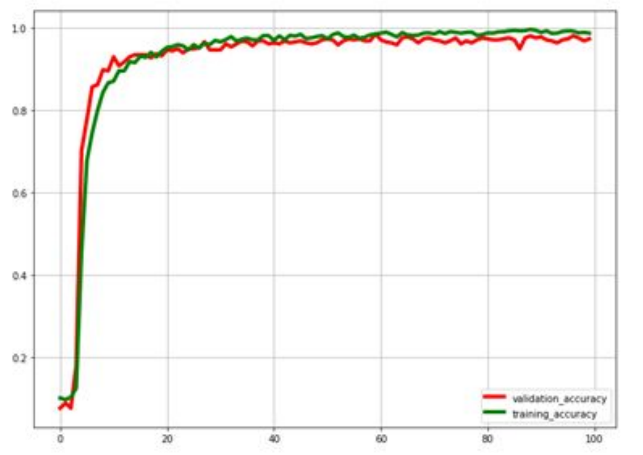




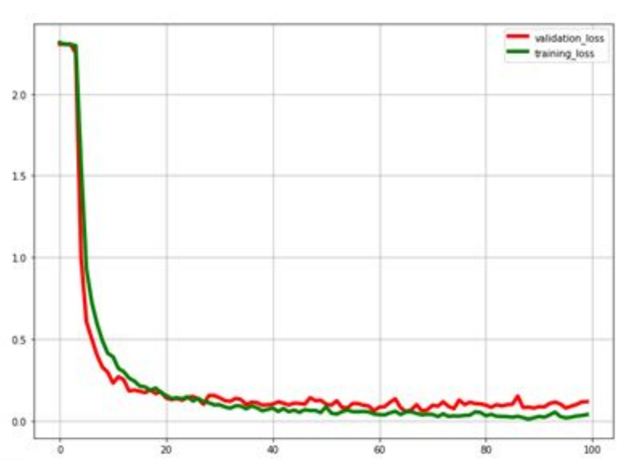
Digits Model cont.

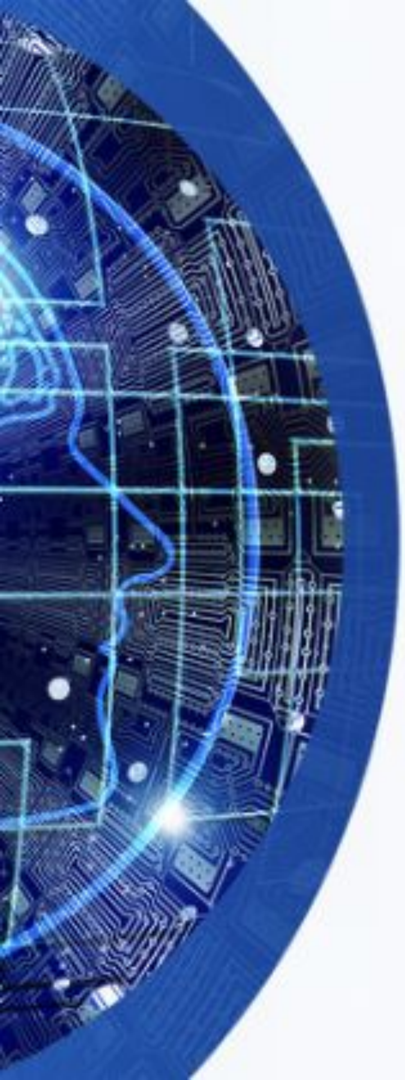
After 100 epochs, validation accuracy at 97%+ and validation loss at 11%.

Model Accuracy

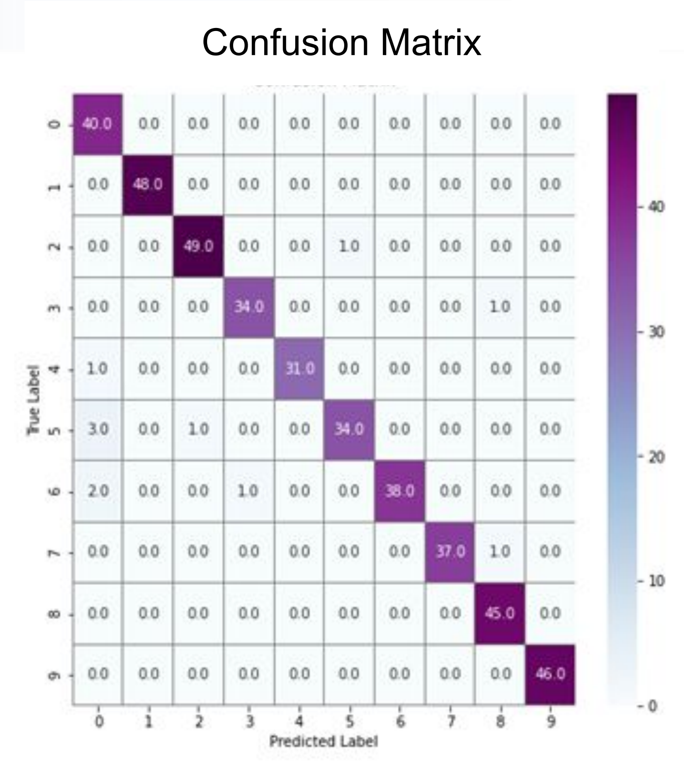


Model Loss



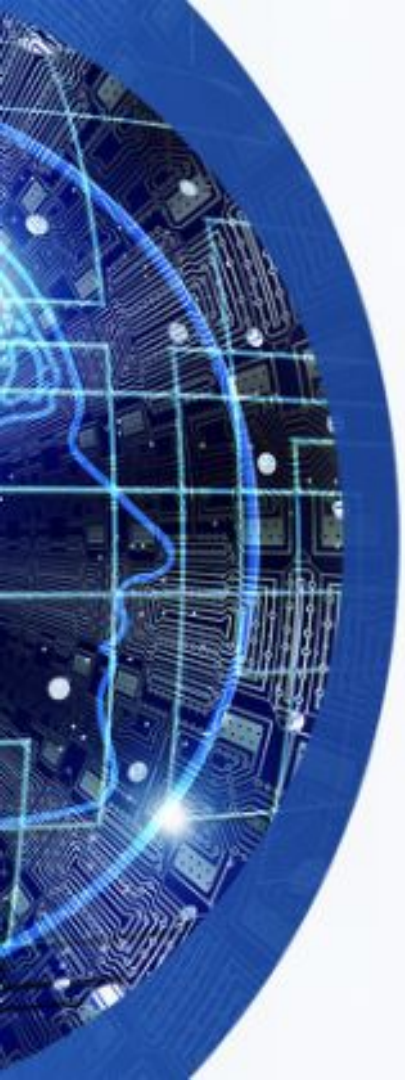


Digits Model cont.



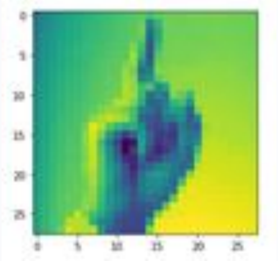
Confusion Matrix

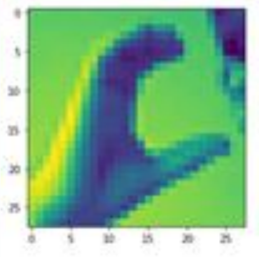
	0	1	2	3	4	5	6	7	8	9
0	40.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	48.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	49.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	34.0	0.0	0.0	0.0	0.0	1.0	0.0
4	1.0	0.0	0.0	0.0	31.0	0.0	0.0	0.0	0.0	0.0
5	3.0	0.0	1.0	0.0	0.0	34.0	0.0	0.0	0.0	0.0
6	2.0	0.0	0.0	1.0	0.0	0.0	38.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	37.0	1.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	45.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	46.0



Alphabet Dataset

- Dataset:
  - 34,627 images of ASL alphabet hand symbols (~1,400 per letter)
  - 28 x 28 pixel black and white images, in csv format
  - ~80/20 train vs test split
- Data transformation required:
  - Pixel values adjusted from 0-255 to between 0-1
  - One hot encoding of y class
  - Resize to 64 x 64 using Pillow function
  - Image augmentation





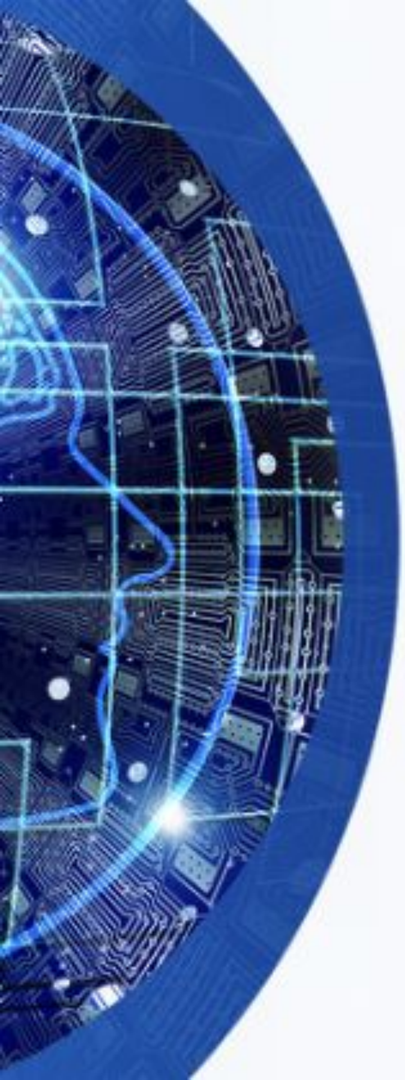
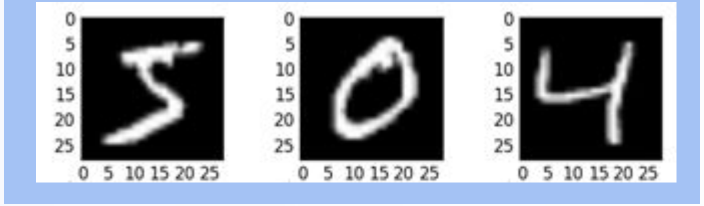
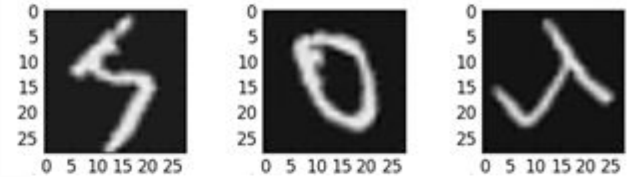


Image Augmentation

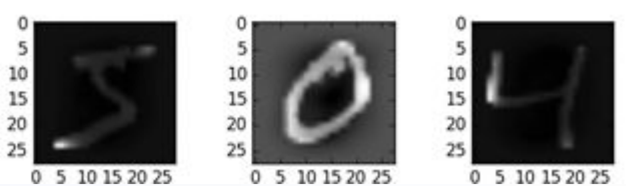
Original



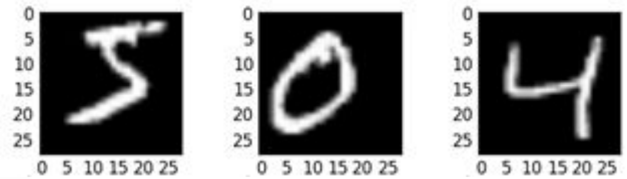
Random Rotations



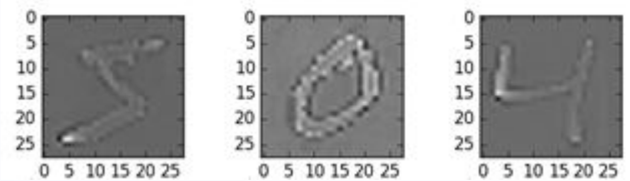
Feature Standardisation



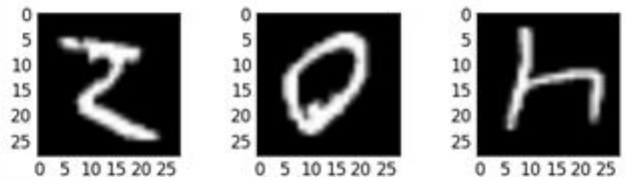
Random Shifts

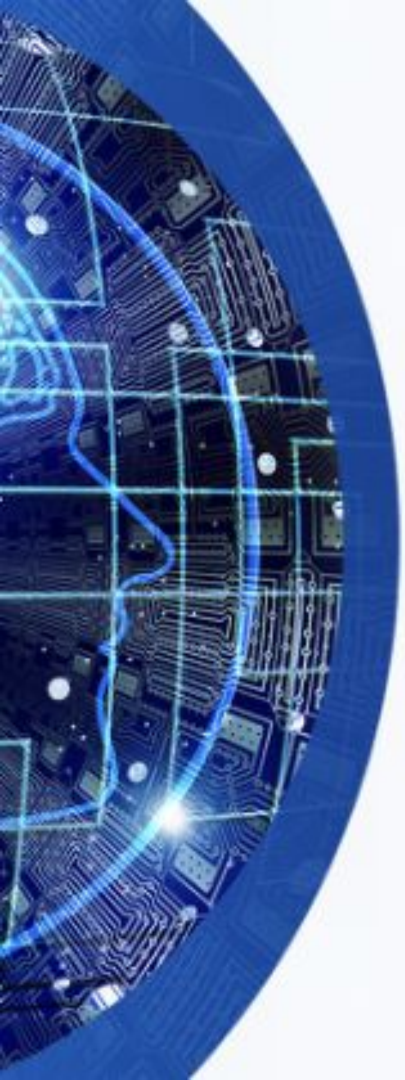


ZCA Whitening



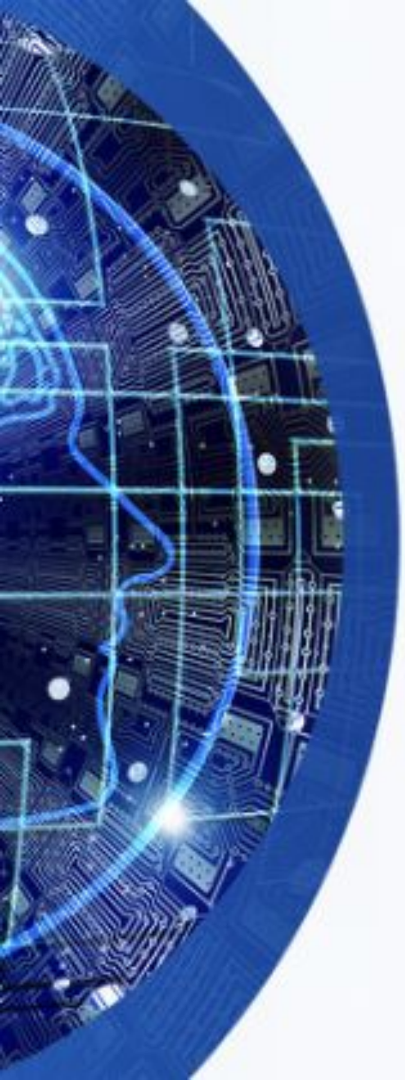
Random Flips





Combined Alphabet & Digits Model

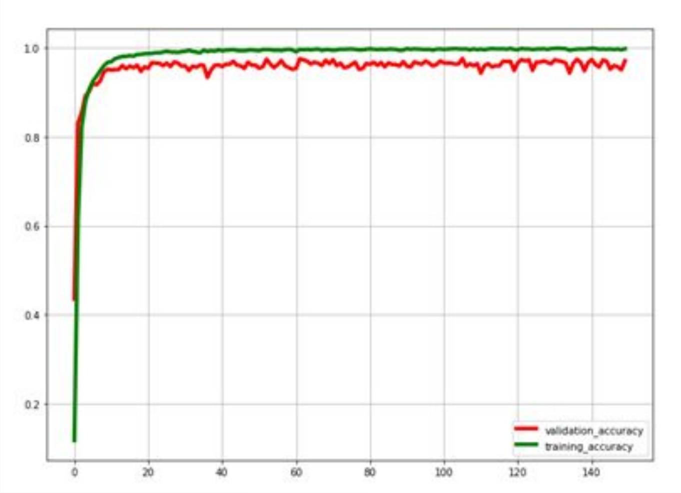
- Dataset:
  - 36,684 images of ASL digit and alphabet hand symbols
  - Data amended to numpy array, using block_diag to join the 2 datasets
  - ~80/20 train vs test split
- Structure & compile used
  - Similar structure and compile used as Digits model.



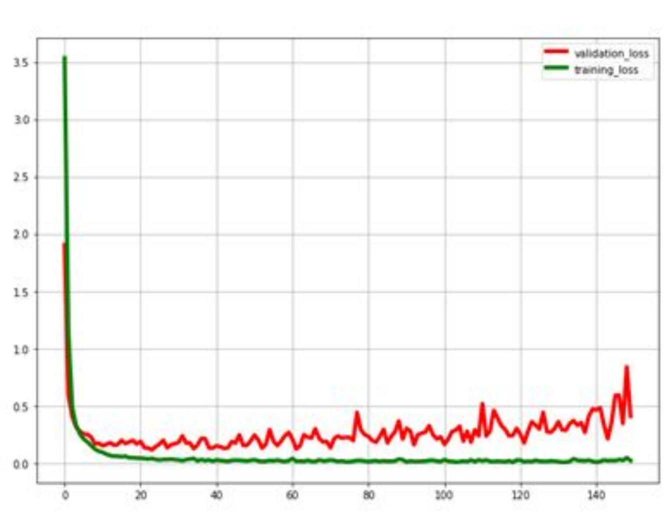
Combined Alphabet & Digits Model cont.

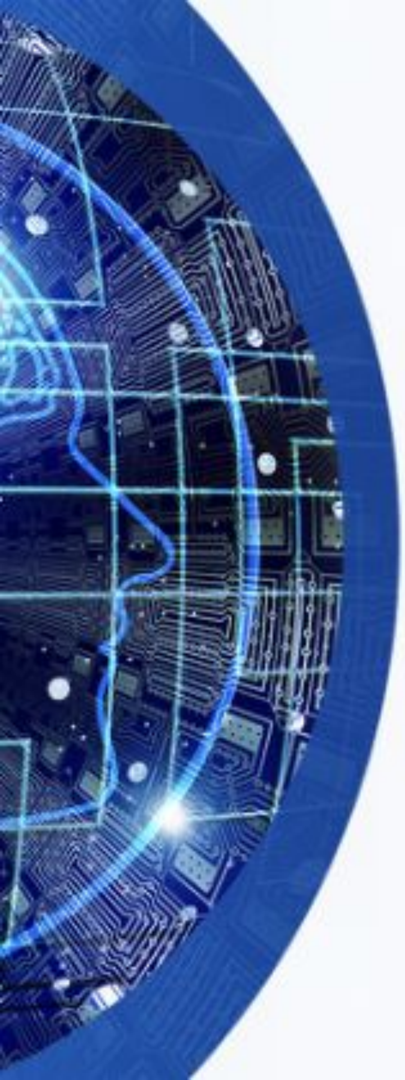
After 150 epochs, validation accuracy at 97%+ and validation loss at 40%.

Model Accuracy



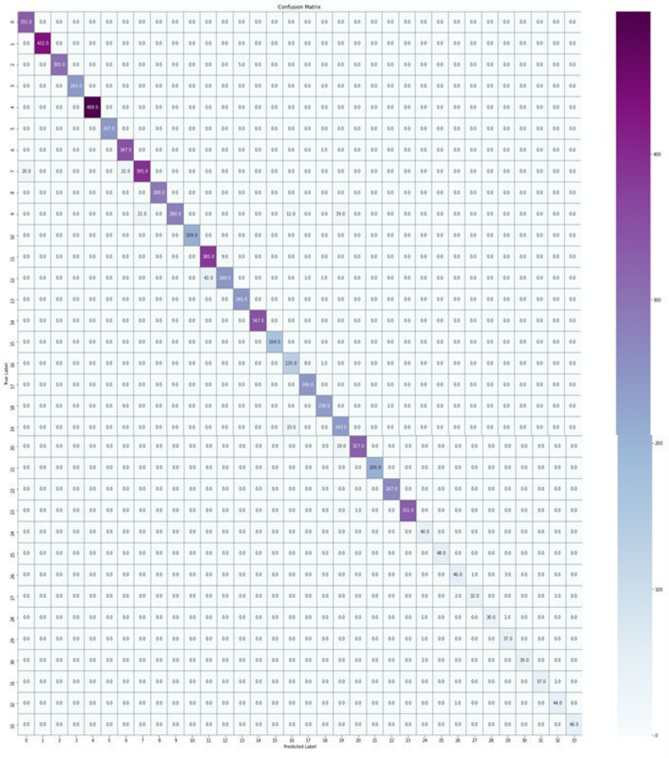
Model Loss

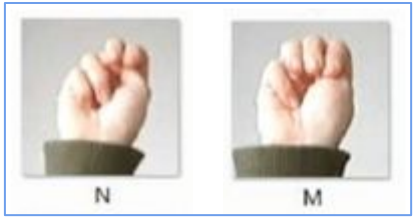


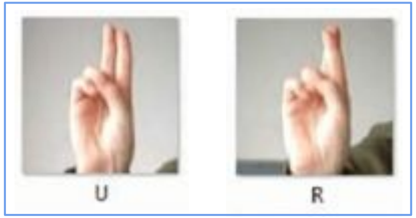


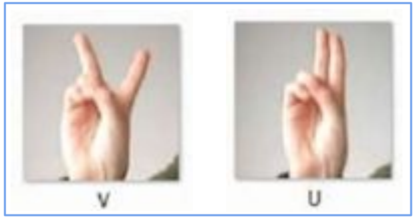
Combined Alphabet & Digits Model cont.

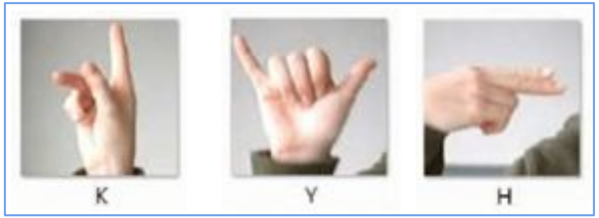
Confusion Matrix

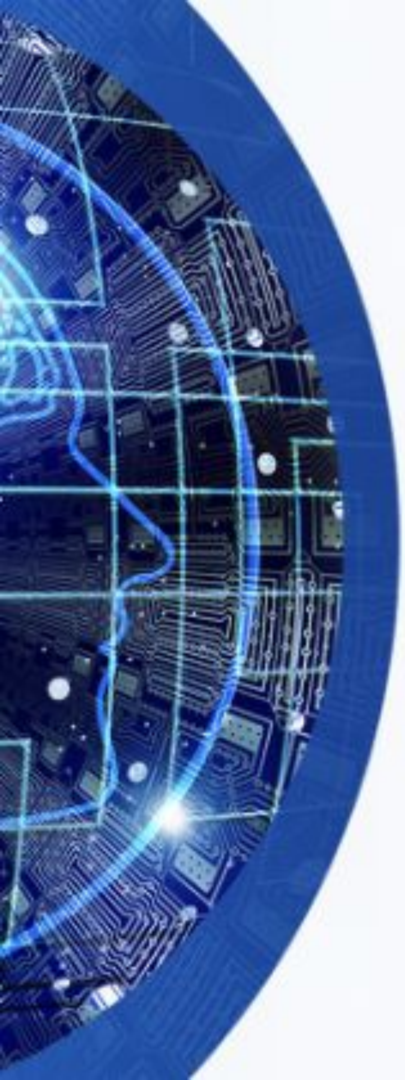






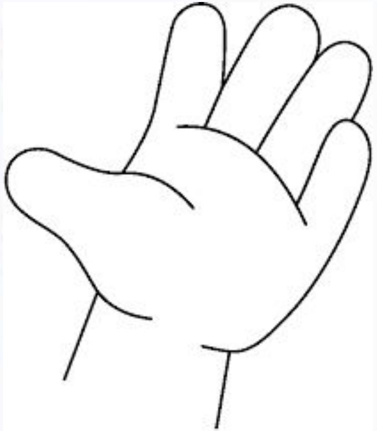




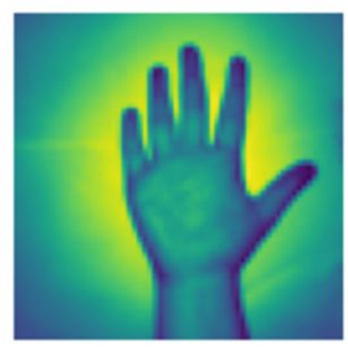


Additional Pictures - 1

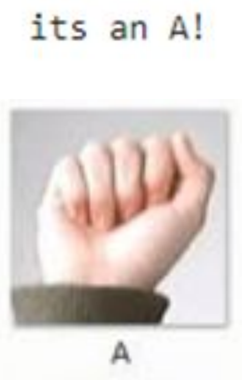
New Input



Our Expectation = 5

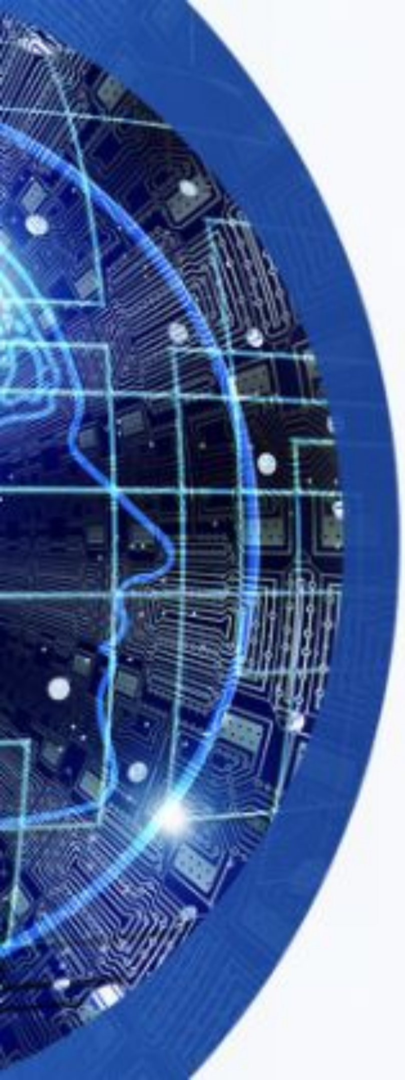


Model Output



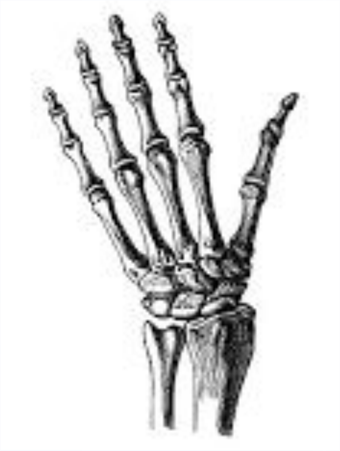
```
its an A!
```

A

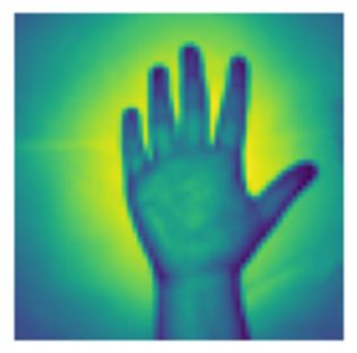


Additional Pictures - 2

New Input

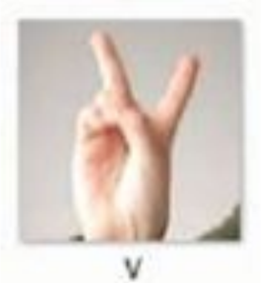


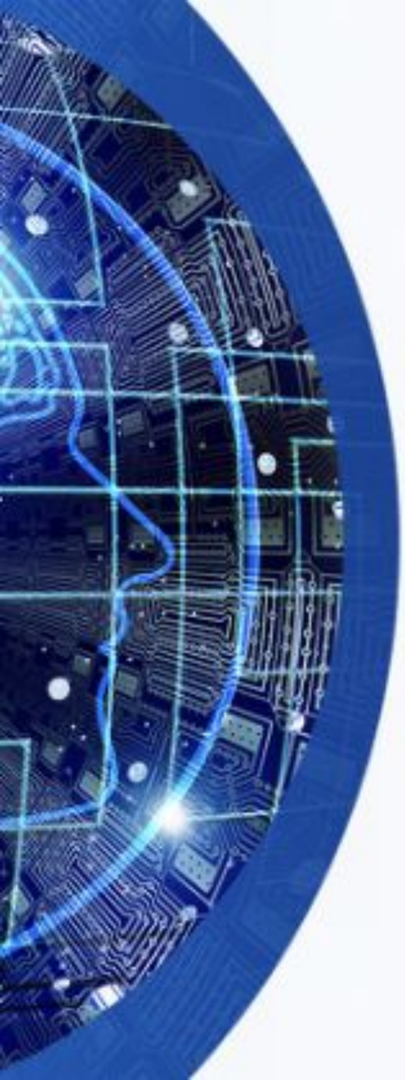
Our Expectation = 5



Model Output

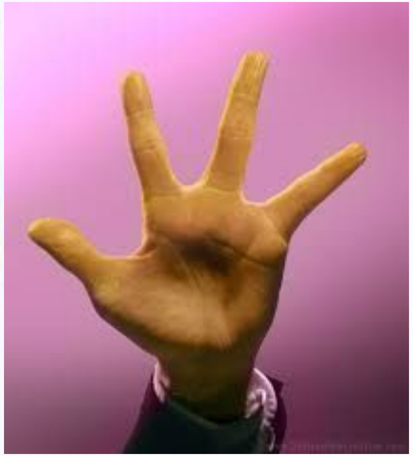
```
its a V!
```



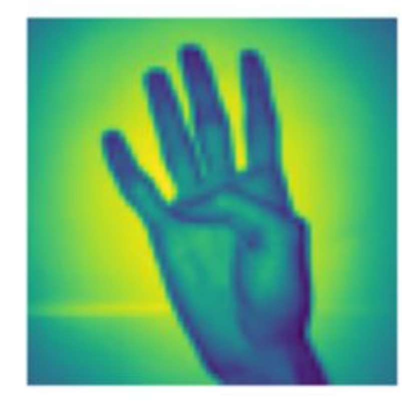


Additional Pictures - 3

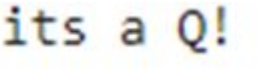
New Input



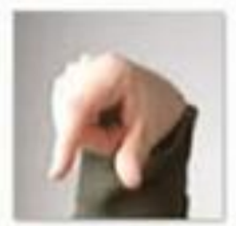
Our Expectation = 4



Model Output



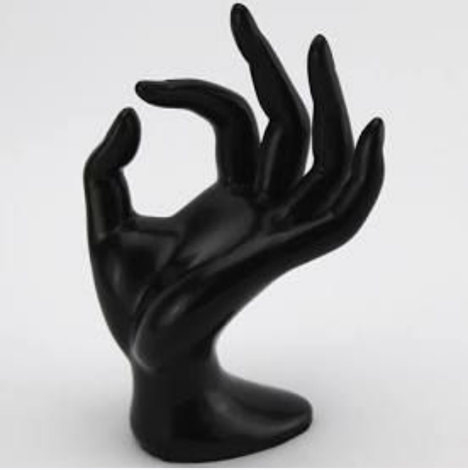
```
its a Q!
```



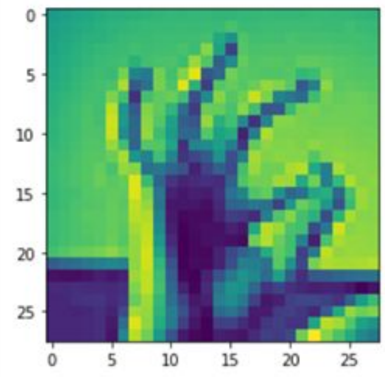
Q

Additional Pictures - 4

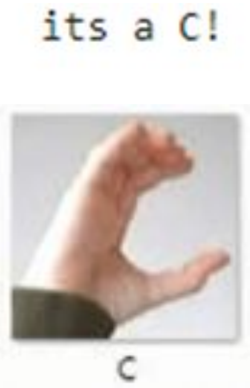
New Input



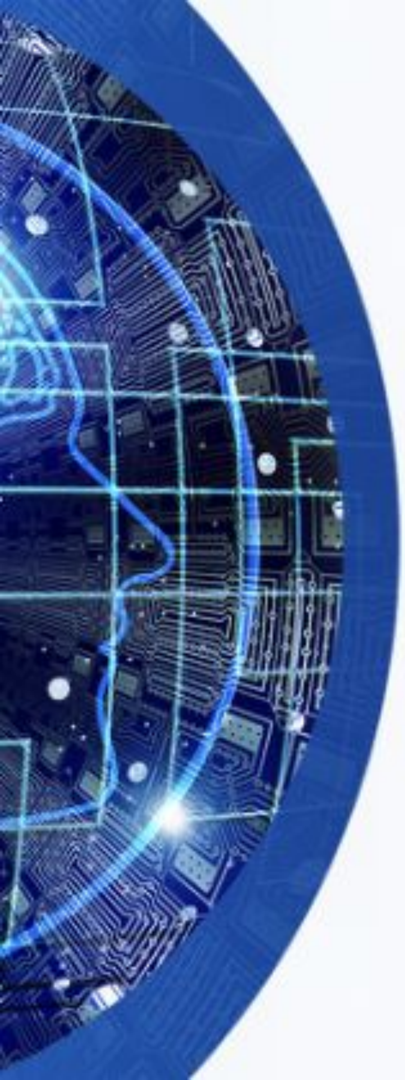
Our Expectation = F



Model Output

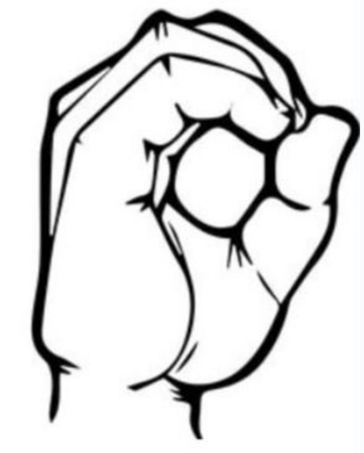


```
its a C!
```

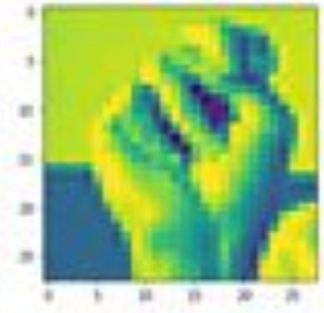


Additional Pictures - 5

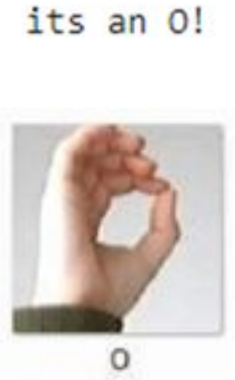
New Input



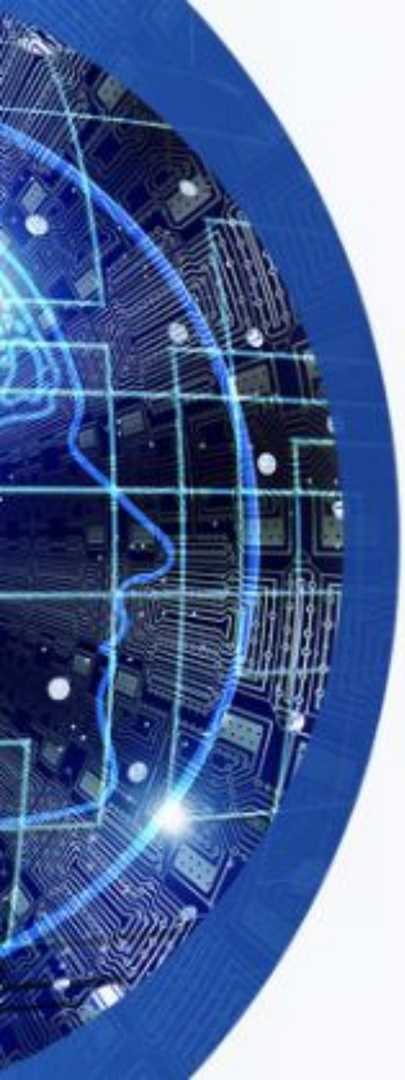
Our Expectation = O



Model Output

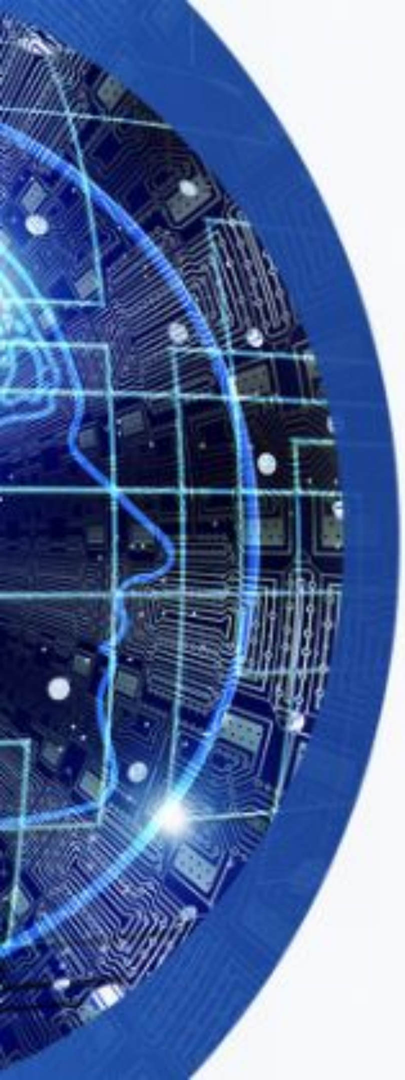


```
its an O!
```

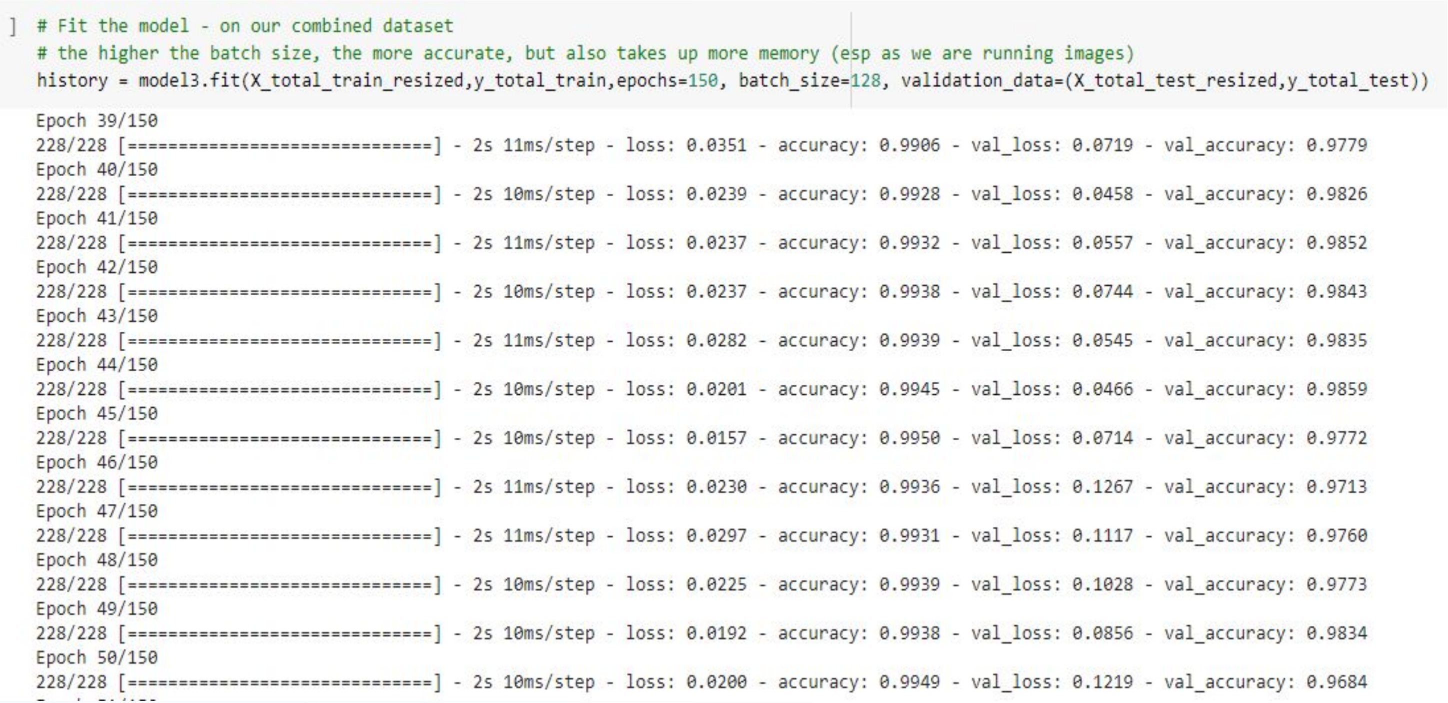


Learnings

- Integrity of data - encountered problems with labelling / resolution of data and resulting model output
- Significant difference in model accuracy output between running on our own computer vs running on Google Colab.
- Colab run models may not be compatible to your computer. Also encountered issues with saving datasets and GPU usage.
- Reconsider flip and rotate image augmentation, these may have confused the model and created some erroneous classifications.
- Adjustment of epochs when running the model



Learnings cont.



```
] # Fit the model - on our combined dataset
  # the higher the batch size, the more accurate, but also takes up more memory (esp as we are running images)
  history = model3.fit(X_total_train_resized,y_total_train,epochs=150, batch_size=128, validation_data=(X_total_test_resized,y_total_test))

  Epoch 39/150
  228/228 [==============================] - 2s 11ms/step - loss: 0.0351 - accuracy: 0.9906 - val_loss: 0.0719 - val_accuracy: 0.9779
  Epoch 40/150
  228/228 [==============================] - 2s 10ms/step - loss: 0.0239 - accuracy: 0.9928 - val_loss: 0.0458 - val_accuracy: 0.9826
  Epoch 41/150
  228/228 [==============================] - 2s 11ms/step - loss: 0.0237 - accuracy: 0.9932 - val_loss: 0.0557 - val_accuracy: 0.9852
  Epoch 42/150
  228/228 [==============================] - 2s 10ms/step - loss: 0.0237 - accuracy: 0.9938 - val_loss: 0.0744 - val_accuracy: 0.9843
  Epoch 43/150
  228/228 [==============================] - 2s 11ms/step - loss: 0.0282 - accuracy: 0.9939 - val_loss: 0.0545 - val_accuracy: 0.9835
  Epoch 44/150
  228/228 [==============================] - 2s 10ms/step - loss: 0.0201 - accuracy: 0.9945 - val_loss: 0.0466 - val_accuracy: 0.9859
  Epoch 45/150
  228/228 [==============================] - 2s 10ms/step - loss: 0.0157 - accuracy: 0.9950 - val_loss: 0.0714 - val_accuracy: 0.9772
  Epoch 46/150
  228/228 [==============================] - 2s 11ms/step - loss: 0.0230 - accuracy: 0.9936 - val_loss: 0.1267 - val_accuracy: 0.9713
  Epoch 47/150
  228/228 [==============================] - 2s 11ms/step - loss: 0.0297 - accuracy: 0.9931 - val_loss: 0.1117 - val_accuracy: 0.9760
  Epoch 48/150
  228/228 [==============================] - 2s 10ms/step - loss: 0.0225 - accuracy: 0.9939 - val_loss: 0.1028 - val_accuracy: 0.9773
  Epoch 49/150
  228/228 [==============================] - 2s 10ms/step - loss: 0.0192 - accuracy: 0.9938 - val_loss: 0.0856 - val_accuracy: 0.9834
  Epoch 50/150
  228/228 [==============================] - 2s 10ms/step - loss: 0.0200 - accuracy: 0.9949 - val_loss: 0.1219 - val_accuracy: 0.9684
```

Gracias!! 😏