Assignment 2
Case Study for Implementation

April 1, 2022

Anthony Horgan
ID: 17452572
MSc. Artificial Intelligence
National University of Ireland Galway
Galway City, Co. Galway
Ireland
a.horgan5@nuigalway.ie

Patrick Dorrian
ID: 17372533
MSc. Artificial Intelligence
National University of Ireland Galway
Galway City, Co. Galway
Ireland
p.dorrian2@nuigalway.ie

Darragh Minogue
ID: 21253383
MSc. Artificial Intelligence
National University of Ireland Galway
Galway City, Co. Galway
Ireland
d.minogue3@nuigalway.ie

Abstract— This paper presents a novel approach for image classification. A new fully weighted kernel layer is created within Keras Tensorflow, and the model was trained and validated on a small dataset of 298 images. An accuracy of 64.7% was obtained.

Keywords—Neural Networks, Keras, Custom Layer.

I. INTRODUCTION

This paper presents the team's findings from assignment 2 as part of the module: Research Topics for AI. The assignment requires the creation of a new neural network to classify alpacas. See Figure 1. For a detailed explanation of the proposed framework.

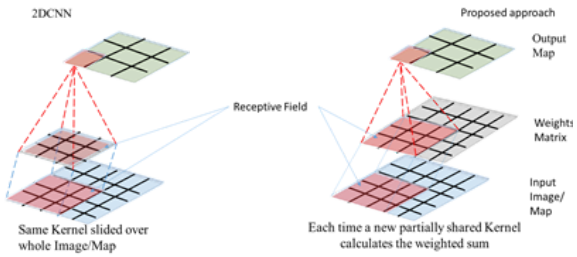

Figure 1. Proposed Approach vs 2D CNN

II. CONTRIBUTIONS FROM TEAM

Table 1. Breakdown of contributions

Task	**Responsibility**
Data Preparation and environment setup	Darragh
Model Development & simulations	Patrick and Anthony
Tensorboard setup	Anthony
Report Writing & Visualisations	Anthony & Darragh
Testing	Patrick
Debugging	Patrick

III. APPROACH

The goal of the team was to be succinct and develop a model on top of Keras which can handle the backpropagation and the convolution operations. With this objective, a custom layer was built on top of Keras architecture, based on the specific requirements of the assignment using the keras functional API.

The layer and convolutional elements are decomposed into two parts. The first contains elementwise multiplication with an initialised weight matrix of the same shape as the feature map/the RGB image: (160, 160, 3). Then once the layer receives input, an elementwise multiplication/Hadamard product is calculated from the inputs and the weight matrix. Secondly dot product pooling of the specified window side is moved over the input to create the feature map. This process is repeated until all the pixel values in the output map are calculated. Tensorflow is used to handle the pooling and these two operations combine to form the novel convolution operation. Bias is finally added before the pooling is returned.

Key input arguments such as padding and stride are read as arguments into the custom layers, but all other additional keyword arguments are passed to the parent class: *tensorflow.keras.layers.Layer*. These keyword arguments are taken in during object construction and when the inputs are provided, the weights and biases are then initialised.

IV. IMPLEMENTATION OF MODEL

This section provides a high level summary of the successful implementation of the model. It displays one table from keras and three figures from Tensorboard: 1) a summary of the network from Keras; 2) a structure graph of the overall architecture of the customised layer from Tensorboard, 2) the accuracy of the model from 1-500 epochs, and 3) the binary cross entropy loss of the model from 1-500 epochs.

Detailed analysis on the implementation of the model is discussed in section VI.

A. Structure of Model

Table 2. Keras Model Summary

Layer (type)	**Output Shape**	**# Param**
input_1 (InputLayer)	[(None, 160, 160, 3)]	0
macro_conv2d (MacroConv2D)	(None, 158, 158, 16)	1228816
activation (RELU)	(None, 158, 158, 16)	0
max_pooling2d (MaxPooling2D)	(None, 79, 79, 16)	0
macro_conv2d_1 (MacroConv2D	(None, 77, 77, 12)	1198284
activation_1 (RELU)	(None, 77, 77, 12)	0
max_pooling2d_1 (MaxPooling2D)	(None, 38, 38, 12)	0
macro_conv2d_2 (MacroConv2D	(None, 36, 36, 8)	138632
activation_2 (RELU)	(None, 36, 36, 8)	0
flatten (Flatten)	(None, 10368)	0
dense (Dense)	(None, 1)	10369

```
=================================================================
Total params: 2,576,101
Trainable params: 2,576,101
Non-trainable params: 0
```

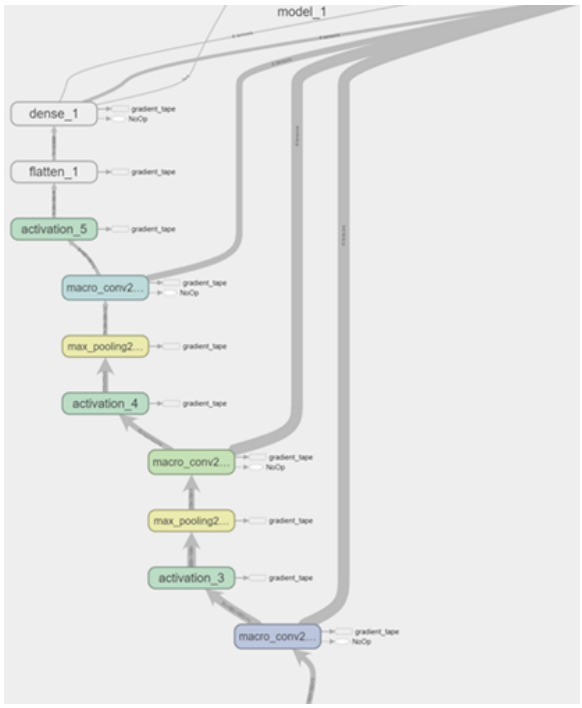

Figure 2. Structure Graph of Customised Layer from Tensorboard.

B. Accuracy and Loss

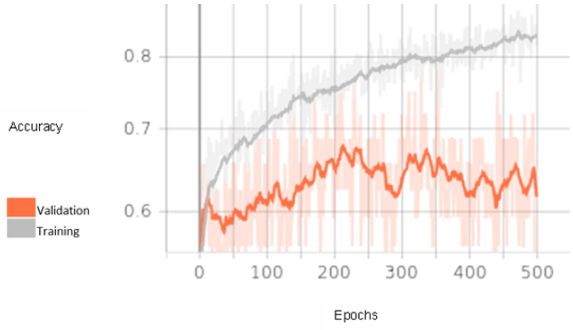

Figure 4. Epoch Accuracy of Neural Network

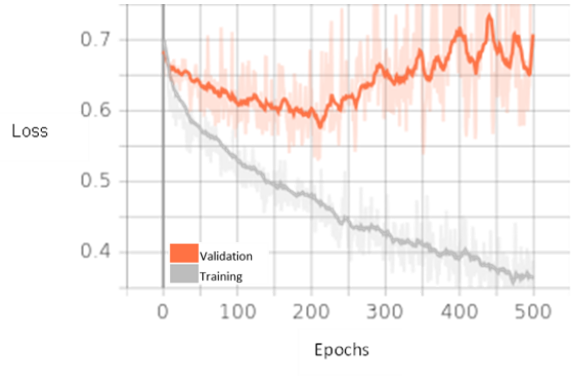

Figure 5. Binary Cross Entropy Loss of Neural Network

V. LOADING AND DISTRIBUTION OF DATA

Google Colab was used among the team as it allows for the writing and execution of python code in a shared notebook. Given the data was somewhat balanced, the two sub-class folders, one with alpacas and the other with non-alpacas were randomly split into training, validation and test sets: 80:10:10.

- **Training**: 261 images
- **Validation**: 32 images
- **Test**: 34 images

The size of the dataset is extremely small and data augmentation was therefore conducted to produce more samples for training the network. This included rotations, zooming, shearing and horizontal flipping, width and height shifting and brightness alteration. Augmentation was only performed on the training set, and not the validation or test set.

All datasets were normalised by dividing all values by 255 to convert the output between the range of 0 and 1.

Figure 6. Training Set 3x3 plot which includes data augmentation

The Google Colab notebook is accessible via: https://colab.research.google.com/drive/1il59y1JgrdjQd4Tk_zYvPAMEz7hq11V_?usp=sharing

VI. TRAINING AND TEST PERFORMANCE

This section describes the results of the model trained only using stochastic gradient descent (SGD). The hyperparameters selected for tuning were:

1. Epochs: 50, 100, 250, 500
2. Learning rate: 0.1, 0.01, 0.001; and
3. Batch size: 8, 16, 32

The process of hyperparameter tuning is an arduous and precarious process. The different hyperparameters cannot be tuned independently. For example, for each different choice of batch size there is a unique optimal learning rate. For this reason we chose to perform a grid search over the hyperparameter values chosen above. This way, we can discover the optimal combination of hyperparameters.

The results are summarised in the table below. To measure performance, accuracy was used as the dataset is adequately balanced and not overly skewed, it's more interpretable and it's also an easy in-built feature of keras. The best accuracy on the validation set was 65.6% which was achieved at 500 epochs, with a learning rate of 0.1 and a batch size of 8. The smaller batch size was likely more successful because of the small size of the dataset, while training of 500 epochs allowed more training time on the dataset.

Overall the training took approximately 30 minutes per 500 epochs, 4 seconds per epoch run on a Nvidia GTX 1650 GPU.

Table 2. SGD Model Results

No.	Epochs	Learning Rate	Batch Size	Training	Validation
1	50	0.1	8	57.50%	46.80%
2	50	0.1	16	58.30%	56.50%
3	50	0.1	32	56.70%	56.30%
4	50	0.01	8	56.70%	56.30%
5	50	0.01	16	56.70%	56.30%
6	50	0.01	32	56.70%	56.30%
7	50	0.001	8	56.60%	56.20%
8	50	0.001	16	56.50%	56.00%
9	50	0.001	32	-	-
10	100	0.1	8	63.20%	40.60%
11	100	0.1	16	59.00%	53.10%
12	100	0.1	32	59.90%	55.10%
13	100	0.01	8	56.70%	56.30%
14	100	0.01	16	56.70%	56.30%
15	100	0.01	32	56.70%	56.30%
16	100	0.001	8	56.60%	56.20%
17	100	0.001	16	56.70%	56.30%
18	100	0.001	32	-	-
19	250	0.1	8	69.30%	50.00%
20	250	0.1	16	60.20%	50.00%
21	250	0.1	32	63.60%	55.10%
22	250	0.01	8	62.20%	55.00%
23	250	0.01	16	62.20%	55.00%
24	250	0.01	32	56.70%	56.30%
25	250	0.001	8	56.60%	56.20%
26	250	0.001	16	56.70%	56.30%
27	250	0.001	32	-	-
28	500	0.1	8	66.70%	65.60%
29	500	0.1	16	67.60%	57.60%
30	500	0.1	32	66.20%	53.90%
31	500	0.01	8	65.00%	57.30%
32	500	0.01	16	62.40%	53.20%
33	500	0.01	32	56.70%	56.30%
34	500	0.001	8	56.60%	56.20%
35	500	0.001	16	56.70%	56.30%
36	500	0.001	32	-	-

- Not available. Learning rate didn't add value at this point. Experiment was halted.

VII. FURTHER ANALYSIS

To improve upon the results obtained using SGD, the optimisation algorithm- Adam- was chosen as it has both momentum and acceleration, allowing it to descend fast, spending less time descending when the gradient is obvious. The model was retrained using Adam across the same hyper-parameters mentioned in the previous section. The result was an accuracy of 70.9%, a 5.3% increase on SDG.

The results of the hyperparameters are displayed in the table below.

Table 3. Adam Model Results

No.	Epochs	Learning Rate	Batch Size	Training	Validation
1	50	0.1	8	56.20%	55.20%
2	50	0.1	16	55.00%	54.40%
3	50	0.1	32	55.00%	54.80%
4	50	0.01	8	62.40%	58.10%
5	50	0.01	16	60.60%	56.10%
6	50	0.01	32	61.30%	56.70%
7	50	0.001	8	67.70%	59.00%
8	50	0.001	16	67.40%	60.10%
9	50	0.001	32	65.90%	58.50%
10	100	0.1	8	56.40%	56.20%
11	100	0.1	16	56.60%	53.30%
12	100	0.1	32	56.50%	53.80%
13	100	0.01	8	62.10%	58.30%
14	100	0.01	16	60.00%	51.60%
15	100	0.01	32	63.00%	57.10%
16	100	0.001	8	71.30%	64.60%
17	100	0.001	16	71.30%	64.90%
18	100	0.001	32	69.40%	59.10%
19	250	0.1	8	56.50%	56.30%
20	250	0.1	16	56.70%	56.30%
21	250	0.1	32	56.90%	56.10%
22	250	0.01	8	57.00%	55.90%
23	250	0.01	16	62.90%	55.40%
24	250	0.01	32	62.00%	65.60%
25	250	0.001	8	77.60%	61.80%
26	250	0.001	16	77.40%	70.90%
27	250	0.001	32	75.90%	68.90%
28	500	0.1	8	56.00%	56.20%
29	500	0.1	16	56.70%	56.30%
30	500	0.1	32	56.70%	56.30%
31	500	0.01	8	57.30%	56.20%
32	500	0.01	16	56.60%	56.00%
33	500	0.01	32	60.70%	54.80%
34	500	0.001	8	82.70%	61.30%
35	500	0.001	16	81.90%	69.30%
36	500	0.001	32	81.40%	63.50%

B. Visualisations of Performance.

Graphs are presented below to visually display the effects of the changes in the hyperparameters. This helps to observe emerging trends.

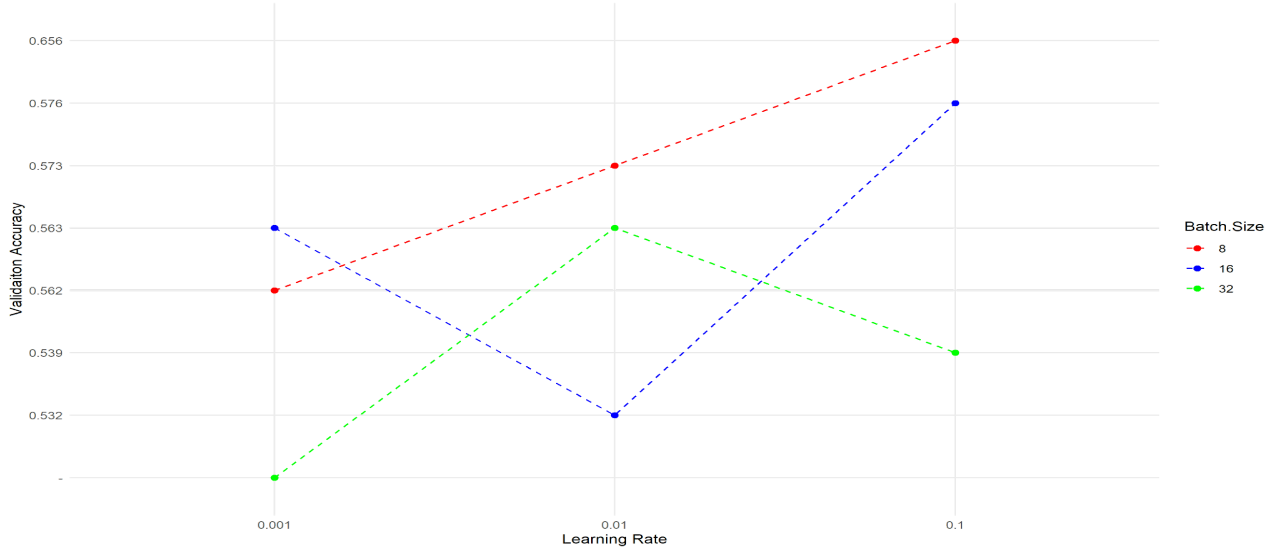

Figure 6. Accuracy over learning rate for SGD

From Figure 6 we can see that in general, a higher learning rate works better. As lr decreases from 0.1 to 0.001, perhaps the network gets stuck in local minima (lr too small to get out) or perhaps the lr is so small that it takes too long to train.

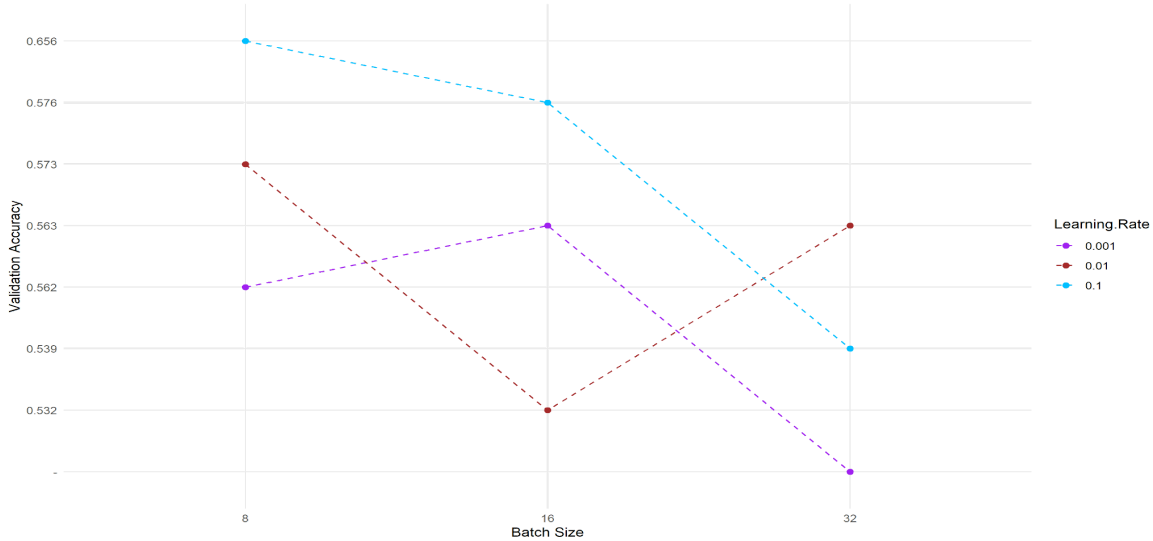

Figure 7. Accuracy over batch size for SGD

As the batch size increases there is a slight trend towards worse performance. It seems that smaller batch sizes generally perform better. This might be due to the small amount of data. With a smaller batch size, there are more gradient descent steps per epoch. Also, a smaller batch size means that the optimization process will be more stochastic/random. These random optimization jumps may allow the network to escape a local optima.

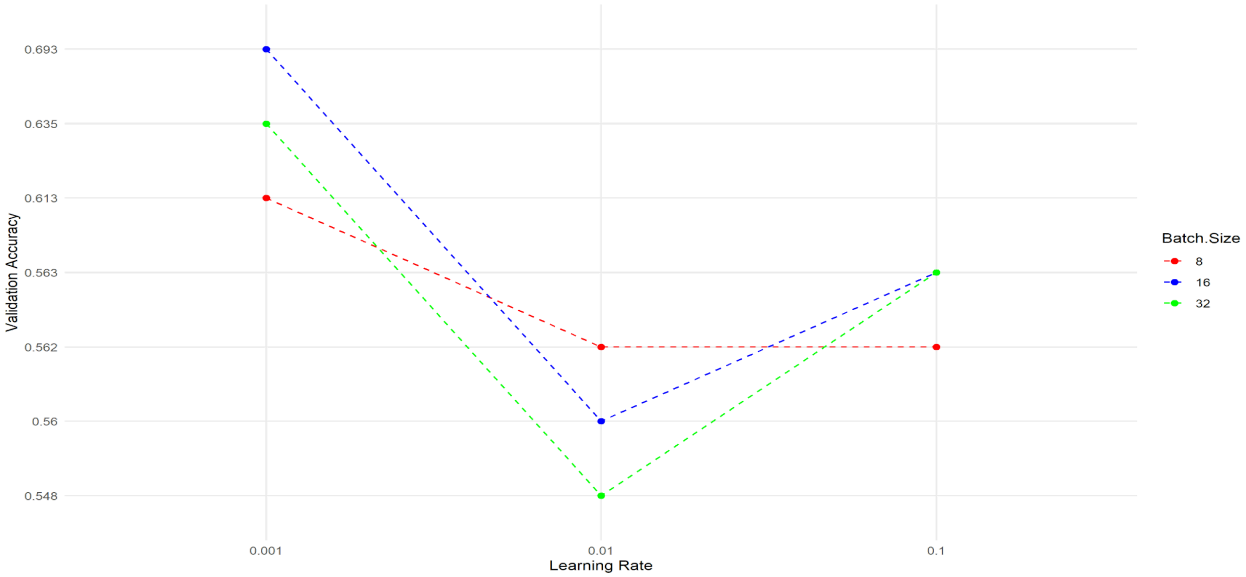

Figure 8. Accuracy over learning rate for Adam

In contrast to SGD, a smaller learning rate seems to work better for Adam. The best performing networks were trained with a learning rate of 0.001. A good starting place learning rate for SGD is around 0.1 and a good starting place for learning rate with Adam is around 0.001-0.0001.

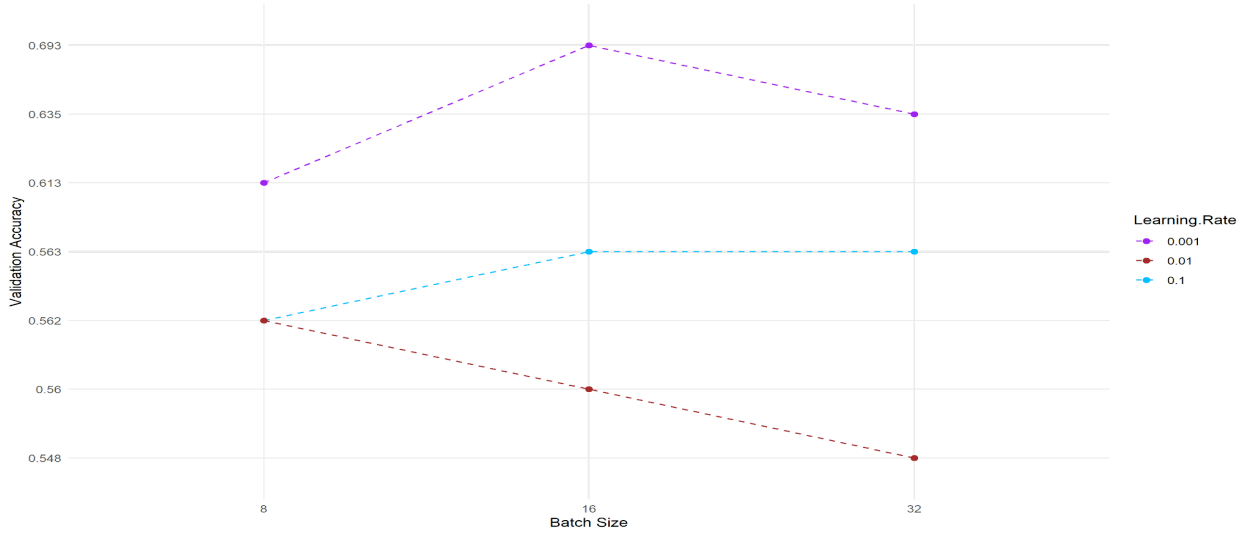

Figure 9. Accuracy over batch size for Adam

There is no strong correlation between batch size and performance for Adam. It seems that the adaptive learning rate can deal more easily with different batch sizes.

VIII. WIDTH AND DEPTH ANALYSIS AND MODEL COMPARISON

In this section, the depth and width of the model was increased and decreased to investigate the effects on accuracy. Furthermore, a comparison was also made with a 2d Convolutional Neural Network (2D-CNN). Each of these efforts were compared against the highest performing model Adam model. The following methods were applied:

1. For deeper networks, another custom layer was added with four filters, another RELU activation layer was added and a max pooling layer. To make the network shallower, the first custom layer, RELU layer and max pooling layer were removed.
2. To adjust the width, the number of filters is doubled for each custom layer. To reduce the width, the number of filters were halved per custom layer.
3. 2d Convolutional Neural Network.

Table 4. Model Comparisons

Adjustment	***Epoch***	***Training***	***Validation***
Depth Increased	*50*	*67.30%*	*57.80%*
	100	*72.70%*	*64.80%*
	150	*78.50%*	*63.10%*
	250	*80.70%*	*63.20%*
Depth Decreased	*50*	*67.70%*	*59.70%*
	100	*72.00%*	*66.50%*
	150	*77.90%*	*66.40%*
	250	*82.20%*	*59.40%*
Width Increased	*50*	*65.90%*	*59.60%*
	100	*70.60%*	*68.60%*
	150	*78.60%*	*68.00%*
	250	*83.70%*	*62.50%*
Width Decreased	*50*	*72.40%*	*56.20%*
	100	***67.40%***	***71.80%***
	150	*75.90%*	*67.00%*
	250	*82.00%*	*70.20%*
2D CNN	*50*	*70.00%*	*65.80%*
	100	*75.30%*	*62.60%*
	150	*83.70%*	*49.10%*
	250	*88.50%*	*65.60%*

The model with the number of filters halved within the custom layer, provided better results than the model trained in the previous section by 0.9%.

IX. TEST RESULTS & CONCLUSION

Once all the comparisons were made and the final model was selected, the model was trained on the test set to represent the final evaluation of the model. This avoids peeking which is present in the validation set.

See below a summary of the final result.

Table 5. Test Set Results on Final Algorithm

Algorithm	Adam
Epochs	500
Learning Rate	0.001
Batch Size	16
Training Accuracy	64.7%
Validation Accuracy	71.8%
Test Accuracy	64.7%

In the end, the mean difference between accuracy across the training and validation sets, across all the models trained was 5%. The difference in the validation and the test set in the final model selected was 7.1%.

The model performed adequately. Improvements on the accuracy score would likely take place if more training data became available. Moreover, the type of model employed in this assignment is likely to be more suitable to an embedded image processing task whereby the features of the image don't change too much e.g. anomaly detection of passport images or defect detection of products on a conveyor belt.

BIBLIOGRAPHY

1. StackOverflow, 2017, How to add a trainable hadamard product layer in keras?, https://stackoverflow.com/questions/46821845/how-to-add-a-trainable-hadamard-product-layer-in-keras