(i) (a)

For the this function, we are assuming that there is no padding and that the stride size is 1, therefore the output dimensions will be $(n - k + 1) * (n - k + 1)$.

Function pseudocode:

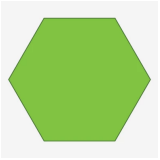
```
input: k*k kernel, n*n array
output: (n-k+1)*(n-k+1) array
size = n - k + 1;
len = kernel.width;
int result[size][size];
for i := 0 to size - 1
     for j := 0 to size - 1
```

$$result[i][j] = \sum_{k=0}^{len} \sum_{l=0}^{len} kernel[k][l] * array[i + k][j + l]$$

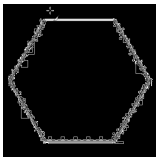
```
return result;
```

(i) (b)

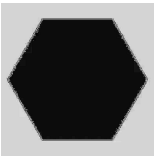
The image selected was a green hexagon:



The image is mostly green so I will only consider the green channel when applying the kernel. When **kernel1** is applied, the result is this:



It appears that the kernel has applied edge detection to the original image.
When **kernel2** is applied, the result is this:



It appears that this kernel has applied region detection to the original image by identifying the region outside of the hexagon as opposed to the inside.

(ii) (a)

32 * 32 input, 3 channels
+
3 * 3 kernel, 16 output channels, padding = same, stride = 1x1
⇒
32 * 32 input, 16 channels
+
3 * 3 kernel, 16 output channels, padding = same stride = 2x2
⇒
16 * 16 input, 16 channels
+
3 * 3 kernel, 32 output channels, padding = same, stride = 1x1
⇒
16 * 16 input, 32 channels
+
3 * 3 kernel, 32 output channels, padding = same, stride = 2x2
⇒
8 * 8 input, 32 channels
⇒
dropout layer
⇒
flatten layer
⇒
dense layer

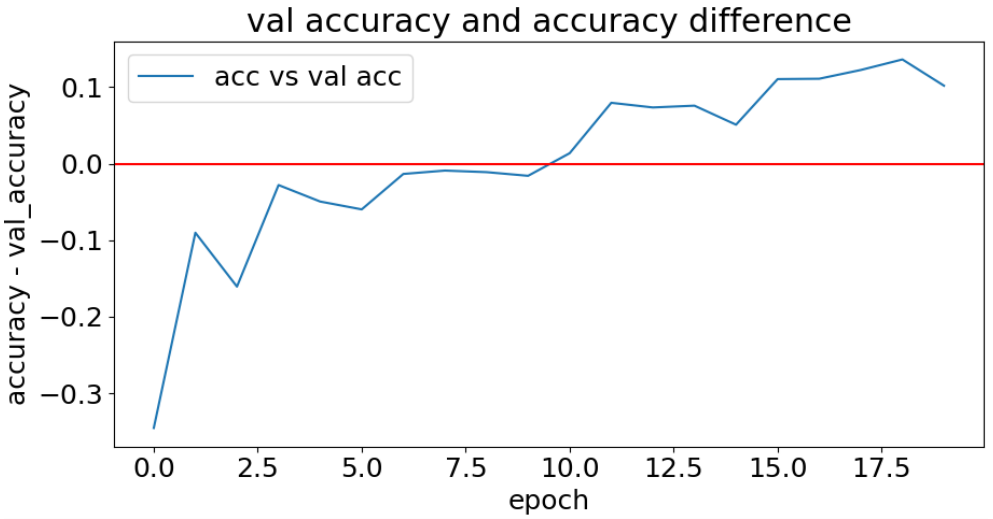
(ii) (b) (i)

The keras model has 37,146 total parameters.
The layer with the most parameters will be the deep layer because it's connected to every layer in the model so its parameters will be at least equal to the sum of all the parameters of the previous layers. But the convolution layer with the most parameters was the last convolution layer i.e 3 * 3 * 32 kernel with 32 channels. It has the highest number of parameters because when we multiply the width, height, input channels and number of output channels together, we get the highest value.

When comparing the performance of the training data to the testing data, we see that the accuracy (on my machine) is 61% whereas the accuracy of the training data is 49%. This is expected because the model is tuned towards the training data so when it comes to new data, it is harder for it to perform as well.
To compare the testing and training data to some baseline predictor, I have chosen one that just chooses the most common class.
When comparing the baseline predictor to the training data, the accuracy we get is 10% vs 61% and when we compare the baseline predictor to the testing data, the accuracy is 10% vs 49%. There are 10 classes and the number of images with each class are about the same so picking the most common will be correct 10% of the time which is the same as random. The fact that both the testing and training data both significantly outperform our baseline predictor indicates that the model has real predictive capabilities.

(ii) (b) (ii)

One of the side effects of overfitting is that the predictive power or in this case, the accuracy of the model is much higher when applied to training data vs predicting data. From this observation we could infer overfitting by comparing the accuracy to the validation accuracy and if the accuracy is sufficiently high enough, we can assume overfitting has occurred. If we plot the percentage difference between the accuracy and validation accuracy, we get this:



(Percentage difference between validation accuracy and accuracy for each epoch)

From the graph, we can see that the percentage difference increases to over 10% which may indicate overfitting.

For underfitting, we would get poor accuracy in both the training and validation stages which is indicative of the early epochs where the accuracy is quite low.

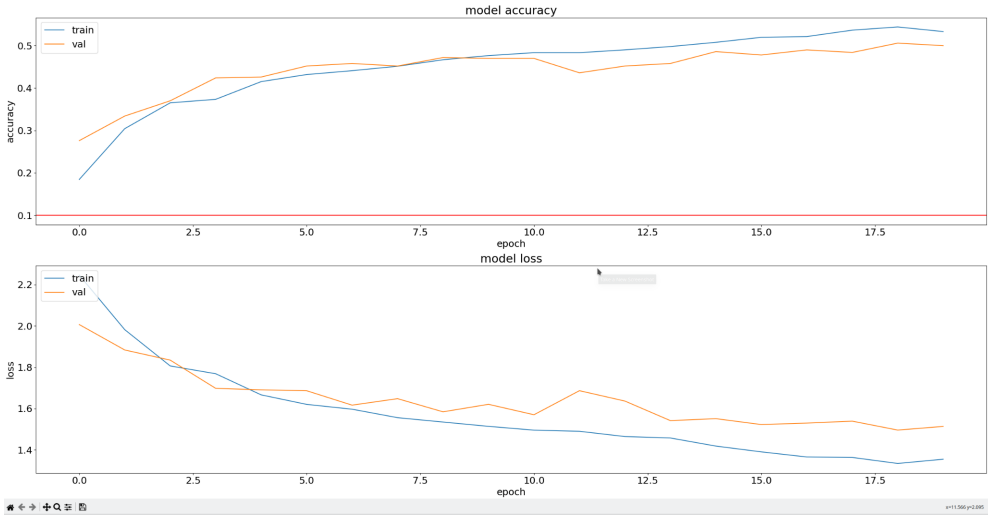
(ii) (b) (iii)

Dataset	Time Taken	Accuracy (train)	Accuracy (test)
5k	20.16s	0.6	0.48
10k	40.47s	0.66	0.56
20k	70.52s	0.71	0.63
40k	159.17s	0.71	0.67

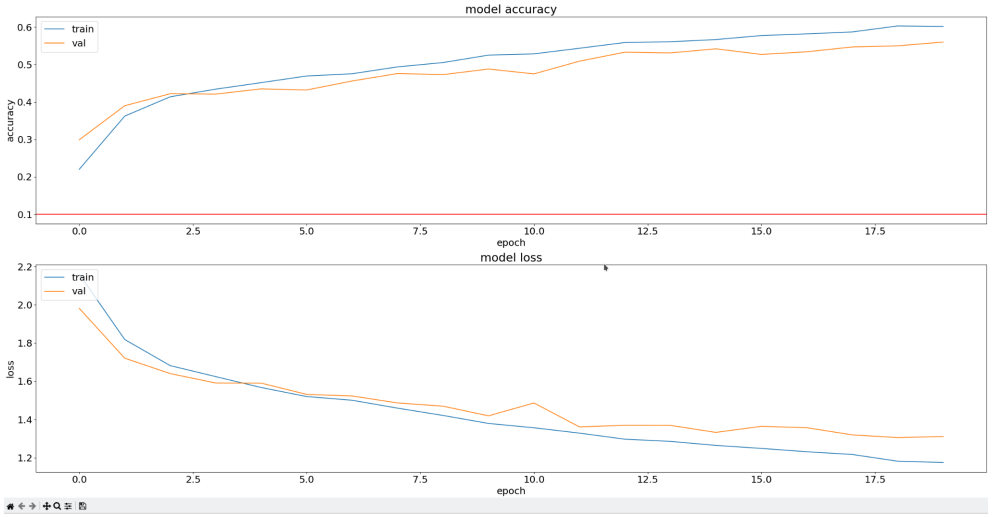
Table detailing the time taken, the accuracy of the training and test data for each dataset size.

Q. How does the prediction accuracy on the training and test data vary with the amount of training data?

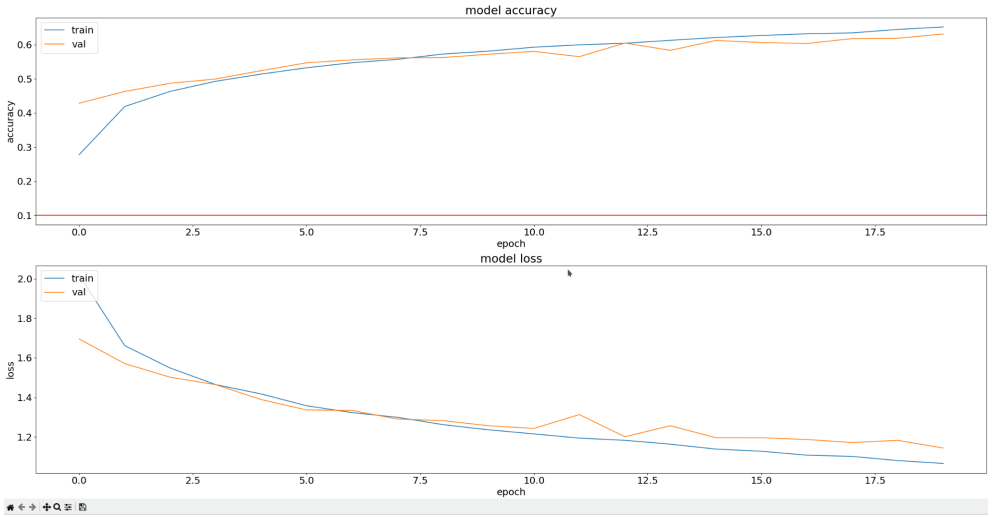
A. From the table, we see that as the dataset size increases, so does the accuracy for both the training data and the testing data.



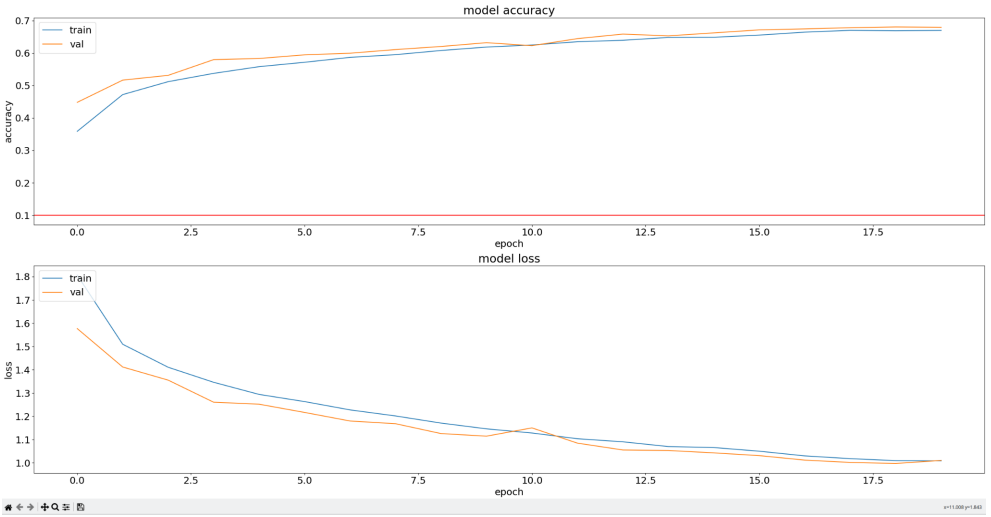
Plot of model accuracy and model loss on 5k dataset



Plot of model accuracy and model loss on 10k dataset



Plot of model accuracy and model loss on 20k dataset



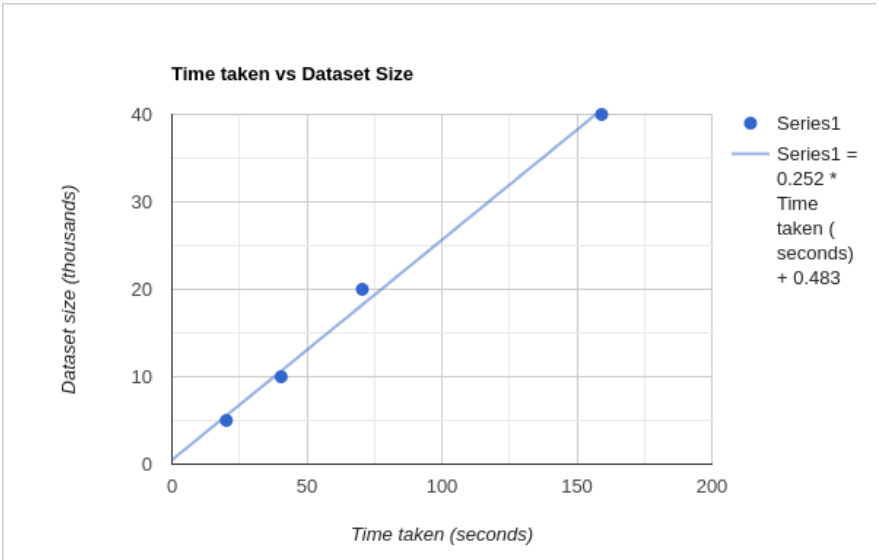
Plot of model accuracy and model loss on 40k dataset

Q. How does the history vary with the amount of training data used and what does this indicate about over/under-fitting?

A. From the plots, we can see that as we increase the size of the dataset, the overlap of the plot of the model accuracy for the testing dataset and training dataset increases which indicates that less overfitting has occurred.
With regards to underfitting, that can only be recognised if the accuracy of the model is poor which is not the case.

Q. How does the time taken to train the network vary with the amount of training data used?

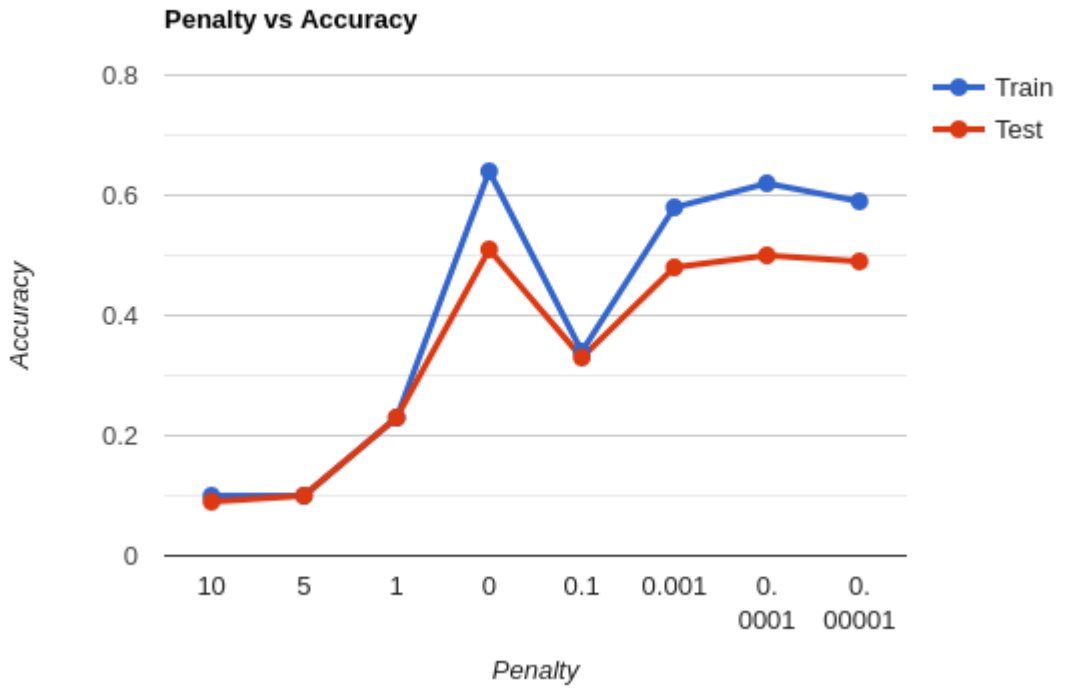
A. To try and find a relationship between the dataset size and time taken to train, I plotted the graph of the time taken to complete the training against the size of the dataset and added a best fitting curve to the dataset. From the curve, it appears that there is a linear relationship between the dataset size and the time taken to train.



(ii) (b) (iv)

Regularisation penalty	Accuracy (training)	Accuracy (testing)
10	0.10	0.09
5	0.10	0.10
1	0.23	0.23
0	0.64	0.51
0.1	0.34	0.33
0.001	0.58	0.48
0.0001	0.62	0.5
0.00001	0.59	0.49

Table of the accuracy of the model when applied to both testing and training data at different penalties for the 5k dataset. The original penalty is highlighted in blue.



Graph plotting the accuracy for each regularisation penalty for the training and testing dataset.

From the graph, we see that the accuracy increases as the penalty decreases, drops back down and again and then increases again but its performance plateaus at around 0.6. At the final point, the performance begins to decline. As a method of procuring more accurate results, increasing the dataset size would be a better method because it procures higher overall accuracies. When it comes to overfitting, the approach of increasing the dataset size is a lot better because the difference between the training and testing accuracy is smaller which implies less overfitting has occurred.

(ii) (c) (i)

	Parameter Count	Time taken	Accuracy (Train)	Accuracy (Test)
Using Strides	37,146	20.17s	.64	.51
Using max pool	37, 146	39.35s	.63	.5

Q. How many parameters does Keras say this ConvNet has?

A. 37, 146 parameters. The lack of change in the number of parameters is because using a stride of size 2x2 has the same effect of using a max pool of 2x2 when it comes to the output dimensions.

Q. How does the time taken to train the network and the prediction accuracy on the training and test data compare with that of the original network?

A. The time taken to train the model is about twice as long when using max pooling and the performance is about the same for both the accuracy and training model.

Q. If the training time has changed, why do you think that's happened?

A. When using a stride size of 2x2, values within the input layer are skipped, thus saving on computation time whereas when using a max pooling layer, the entire input layer is computed and then additional computing is done to pick the max value.