Matilde Piccoli, mp2419, 01764158

1. CNN Introduction (02)

Task 1: The main influencing factors in the performance of the CNN were: the number and type of filters, the type of pooling, padding and activation functions, the order and number of the layers. (Figure 1) It was observed that increasing the number of Convolutional filters (32,64,128 respectively) as the output volume decreases (after each Pooling) was a good strategy to increase the performance (from model 2 to model 1) whilst avoiding an exponentially increasing complexity of the model (possible overfitting). This technique is in fact also used in the AlexNet [**1**]. A sweet-spot for the receptive field width was found at (4,4), as when this was further increased (model 3) the validation error decreased whilst training increased, leading to overfitting. Max Pooling of (2,2) was found to be the most suitable in this case as it introduces translation invariance to the model, selecting the most prominent features extracted by the Convolution. When Pooling was more heavily applied, the down-sampling lead to increasingly higher loss also of essential information, leading to a worse performance. Decreasing the number of layers to one Convolutional and one Dense led to lower error both in training and validation accuracy (underfitting, model 4). However, increasing the number of Convolutional filters (model 2) instead of Dense ones (model 5) in each respective layer helped increasing the complexity without introducing redundancy. The above analysis resulted in the best architecture (model 1) (Figure 2).

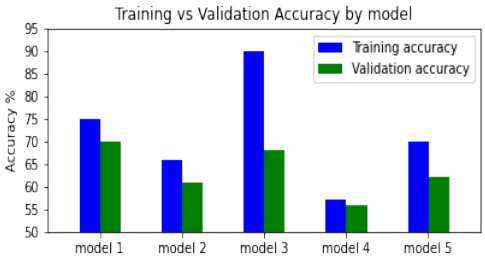

Figure 1: Performance of different models (Appendix A)

Task 2: My best CNN architecture (Conv(64), MaxPool, GlobalAvgPool, Dense)(Appendix B) resulted in a validation error of 56%. This model was chosen as it had a low training and generalisation error for all epochs (figure 3). Other architectures were tested but the main results led to either a higher error both in training and validation (figure 4), suggesting that the complexity was below the sweet-spot (underfitting) [**2**] (Appendix B), or much lower training (34%) but higher validation loss due to the high complexity introduced by multiple Convolutional and FC layers (overfitting) (figure 5, Appendix B). Using the architecture of Figure 3, similar performances were found for all the image categories of the data set (Table 1).

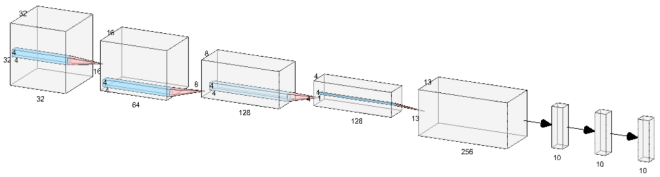

Figure 2: Best architecture for classification (Appendix A)

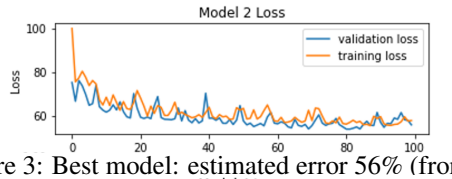

Figure 3: Best model: estimated error 56% (frontal)

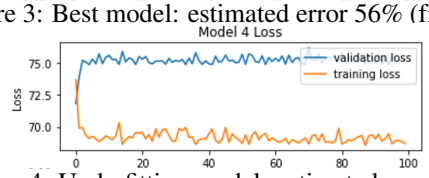

Figure 4: Underfitting model: estimated error 75%

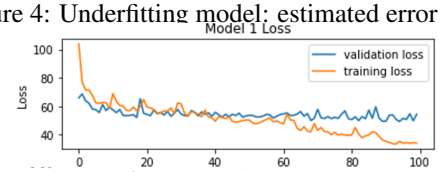

Figure 5: Overfitting model: estimate error 58%

Image type	Estimation error (%)	Training Error (%)
frontal	56.02	54.40
kitchen	60.63	53.59
bedroom	53.36	51.56
bathroom	61.50	60.21

Table 1: Performance for each image category

2. Appendix A

Note on the testing methodology of different architectures used for Task 2.1: To compare the influence of different changes in the model's architecture, various model were tested changing only one characteristic at time (e.g. number of filter, number of layers, Pooling, activation functions). Only the most relevant results were reported in Figure 1. The corresponding model can be found in Figure 7,Figure 9,Figure 10. Model 3 has the same structure as model 1 but with receptive field of width 6 instead of 4.

More analysis on the architectures performance: Other observations on the architecture performance are explained below.

Using 'same' padding, instead of 'valid', led to a better performance since the former type of padding maintained the information at the borders of the image. Additionally, if too many filters (128) were applied in Convolutional and Dense layers the generalisation error increased due to overfitting and the computation complexity increased accordingly, making the simulation longer and more memory-consuming. Finally, different Convolutional layers with smaller number of filters led to much higher computational power but relatively low benefit in terms of accuracy compared to a single Convolutional layer with higher number of filters (higher volume).

Task 2.1 best model: The Best architecture for given classification problem consisted in three sequences of Convolution and Pooling with respectively receptive field of size (4,4) and (2,2), followed by three Dense layers. The activation functions used were Relu for Convolutional layers and Softmax after the Dense ones as appropriate fro a classification problem (Figure 6). Although much more simpler, this architecture recalls the one of AlexNet and share the same ratio in maximising the efficiency of the training computation[**1**].

3. Appendix B

Task 2.2 best model analysis: The best architecture for the regression task is shown in Figure 11.

Alternative architectures: Multiple models were tested to evaluate the best architecture. The most relevant models are reported in Figure 12 and Figure 13 as examples of high and low complexities. In particular, it was observed that using Global Average Pooling instead of Flatten with FC layers improved the performance whilst reducing the number of parameters to tune; Global Average Pooling also makes the model less dependent on spatial translation as it sums all the spatial information of the image. Furthermore, it was noticed that multiple Convolution layers led to a worse generalisation as the models started overfitting on the training data due to its high number of parameters (gap between validation and training loss in Figure 5). This could be fixed using different regularisation techniques, such as Dropout or Batch Normalisation.

Layer (type)	Output Shape	Param #
conv2d_12 (Conv2D)	(None, 32, 32, 32)	1568
activation_16 (Activation)	(None, 32, 32, 32)	0
max_pooling2d_12 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_13 (Conv2D)	(None, 16, 16, 64)	32832
activation_17 (Activation)	(None, 16, 16, 64)	0
max_pooling2d_13 (MaxPooling2D)	(None, 8, 8, 64)	0
conv2d_14 (Conv2D)	(None, 8, 8, 128)	131200
activation_18 (Activation)	(None, 8, 8, 128)	0
max_pooling2d_14 (MaxPooling2D)	(None, 4, 4, 128)	0
flatten_4 (Flatten)	(None, 2048)	0
dense_14 (Dense)	(None, 10)	20490
dense_15 (Dense)	(None, 10)	110
dense_16 (Dense)	(None, 10)	110
activation_19 (Activation)	(None, 10)	0

Total params: 186,310
Trainable params: 186,310
Non-trainable params: 0

Figure 6: Best architecture for classification (model 1 in Figure 1, architecture of Figure 2)

Layer (type)	Output Shape	Param #
conv2d_15 (Conv2D)	(None, 32, 32, 32)	1568
activation_20 (Activation)	(None, 32, 32, 32)	0
max_pooling2d_15 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_16 (Conv2D)	(None, 16, 16, 32)	16416
activation_21 (Activation)	(None, 16, 16, 32)	0
max_pooling2d_16 (MaxPooling2D)	(None, 8, 8, 32)	0
conv2d_17 (Conv2D)	(None, 8, 8, 32)	16416
activation_22 (Activation)	(None, 8, 8, 32)	0
max_pooling2d_17 (MaxPooling2D)	(None, 4, 4, 32)	0
flatten_5 (Flatten)	(None, 512)	0
dense_17 (Dense)	(None, 10)	5130
dense_18 (Dense)	(None, 10)	110
dense_19 (Dense)	(None, 10)	110
activation_23 (Activation)	(None, 10)	0

Total params: 39,750
Trainable params: 39,750
Non-trainable params: 0

Figure 7: model 2 in Figure 1

Layer (type)	Output Shape	Param #
conv2d_18 (Conv2D)	(None, 32, 32, 32)	3488
activation_24 (Activation)	(None, 32, 32, 32)	0
max_pooling2d_18 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_19 (Conv2D)	(None, 16, 16, 64)	73792
activation_25 (Activation)	(None, 16, 16, 64)	0
max_pooling2d_19 (MaxPooling2D)	(None, 8, 8, 64)	0
conv2d_20 (Conv2D)	(None, 8, 8, 128)	295040
activation_26 (Activation)	(None, 8, 8, 128)	0
max_pooling2d_20 (MaxPooling2D)	(None, 4, 4, 128)	0
flatten_6 (Flatten)	(None, 2048)	0
dense_20 (Dense)	(None, 10)	20490
dense_21 (Dense)	(None, 10)	110
dense_22 (Dense)	(None, 10)	110
activation_27 (Activation)	(None, 10)	0

Total params: 393,030
Trainable params: 393,030
Non-trainable params: 0

Figure 8: model 3 in Figure 1

Layer (type)	Output Shape	Param #
conv2d_25 (Conv2D)	(None, 32, 32, 16)	1744
activation_36 (Activation)	(None, 32, 32, 16)	0
max_pooling2d_26 (MaxPooling2D)	(None, 16, 16, 16)	0
flatten_11 (Flatten)	(None, 4096)	0
dense_33 (Dense)	(None, 10)	40970
activation_37 (Activation)	(None, 10)	0

Total params: 42,714
Trainable params: 42,714
Non-trainable params: 0

Figure 9: model 4 in Figure 1

Layer (type)	Output Shape	Param #
conv2d_24 (Conv2D)	(None, 32, 32, 16)	1744
activation_34 (Activation)	(None, 32, 32, 16)	0
max_pooling2d_25 (MaxPooling2D)	(None, 16, 16, 16)	0
flatten_10 (Flatten)	(None, 4096)	0
dense_29 (Dense)	(None, 64)	262208
dense_30 (Dense)	(None, 64)	4160
dense_31 (Dense)	(None, 64)	4160
dense_32 (Dense)	(None, 10)	650
activation_35 (Activation)	(None, 10)	0

Total params: 272,922
Trainable params: 272,922
Non-trainable params: 0

Figure 10: model 5 in Figure 1

Layer (type)	Output Shape	Param #
conv2d_9 (Conv2D)	(None, 64, 64, 16)	448
max_pooling2d_9 (MaxPooling2D)	(None, 32, 32, 16)	0
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 16)	0
dense_3 (Dense)	(None, 1)	17

Total params: 465
Trainable params: 465
Non-trainable params: 0

Figure 11: Best architecture performance: estimated error 56%

Layer (type)	Output Shape	Param #
conv2d_20 (Conv2D)	(None, 64, 64, 16)	448
max_pooling2d_20 (MaxPooling2D)	(None, 32, 32, 16)	0
flatten_6 (Flatten)	(None, 16384)	0
dense_15 (Dense)	(None, 64)	1048640
dense_16 (Dense)	(None, 1)	65

Total params: 1,049,153
Trainable params: 1,049,153
Non-trainable params: 0

Figure 12: Lower complexity model, underfitting: estimated error 75%

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 64, 64, 16)	448
max_pooling2d_6 (MaxPooling2D)	(None, 32, 32, 16)	0
conv2d_7 (Conv2D)	(None, 32, 32, 32)	4640
max_pooling2d_7 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_8 (Conv2D)	(None, 16, 16, 64)	18496
max_pooling2d_8 (MaxPooling2D)	(None, 8, 8, 64)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 64)	0
dense_2 (Dense)	(None, 1)	65

Total params: 23,649
Trainable params: 23,649
Non-trainable params: 0

Figure 13: complex model, overfitting: estimated error 58%

4. References

[**1**] K. Mikolajczyk (2022). *ICL EEE Deep Learning - Part 4, "AlexNet"* [Online] Available: https://bb.imperial.ac.uk/bbcswebdav/pid-2402003-dt-content-rid-10930897$_1$

[**2**] K. Mikolajczyk (2021). *ICL EEE Machine Learning - Part 2.2, "Performance VS Complexity curve"* [Online] Available: https://bb.imperial.ac.uk/bbcswebdav/pid-2306668-dt-content-rid-10029008$_1$/$xid - 10029008_1$