Nima Karshenas, nk4718, 01500753

1. CNN Introduction: task 1

The first parameter that was tested was the number of Convolutional layers. The following settings were used for each layer: **model.add(Conv2D(16, (2, 2), padding='same', strides=1, activation='relu'))**. Typically the deeper the network, the more complex the features/patterns that are extracted. After 5 layers training error plateaus and the test accuracy begins decreasing, suggesting that using 5 layers best captures the complexity of the target function, after this point the model is simply overfitting to the training data.

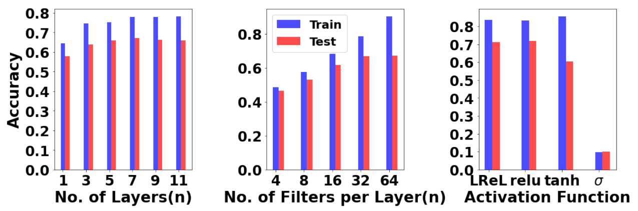


Figure 1: Plots showing the effects of the most influential parameters on the model accuracy.

In Figure 1, training accuracy increases with the number of filters per layer since more filters means more features are being extracted from each image, increasing the complexity of the model. A more complex model does not guarantee better test performance due to risk of overfitting. Although having 64 filters per layer shows better training accuracy, having 32 filters shows slightly better test accuracy indicating that having 64 filters overfits to the training data. 32 filters per layer has the best performance. As for the activation function, both ReLU and LReLU show the best results, however since ReLU is more computationally efficient, this will be used in the final model. For sigmoid activation, both test and training error fall to almost exactly $\frac{1}{N}$ where N is the number of classes. This highlights an inherent issue of using softmax and cross entropy loss coupled with sigmoid activation, in an unregularised model. The sigmoid function can saturate, forcing the prediction $\hat{y_k}$ to $\frac{1}{10}$ for every label, meaning predictions are made randomly (See Section 3.1).

The best model features 5 convolutional layers, an increasing number of filters per layer, (4x4) filter size for each filter summarised as: (8,16,32,MPool,64,MPool,128,D(16), D(10))(See section 3.2 for clarification), and shown in Figure 2. This model yielded a training accuracy of 0.83465 and a test accuracy of 0.72832 without using EarlyStopping.

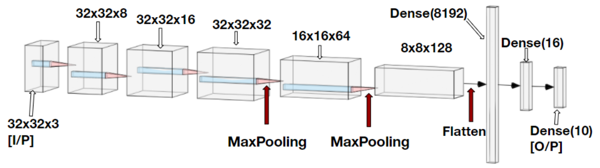


Figure 2: Model Architecture for the best model[1]

2. CNN Introduction: task 2

One very complex model: (4,4,8,16,32,64,128,256,512,512, D(1)), one very simple model: (4,4, D(1)) and the best model: (8,MPool,16, 32,64,128,MPool,256,D(32), D(1)) are trained on, producing results shown in Figure 3.

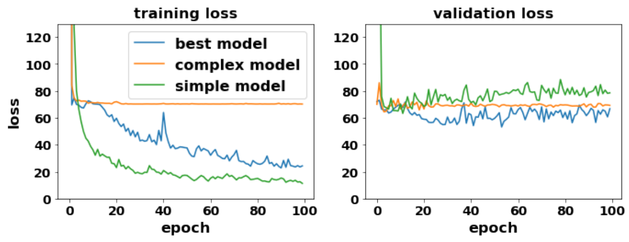


Figure 3: loss curves for 3 different architectures

For the simple model, low training and high validation losses are shown. Very few features are being extracted, thus the model is trained to recognise specific images in the training set instead of general patterns, and so the model generalises very poorly. For the complex model, it shows a completely flat curve. This suggests that the model is either getting stuck in local minima or there is a dying ReLU problem. When reducing the learning rate, the training loss falls with epochs as expected, suggesting that the issue was due to being trapped in local minima. The best model was tested on pictures from different parts of the house, using EarlyStopping. The model generalises fairly well to the other types of images, but unsurprisingly performs best with the type of image it was originally designed for.

Image Type	Estimation error(%)	Training error(%)
Frontal	50.37	28.58
Kitchen	56.15	23.82
Bedroom	54.14	25.88
Bathroom	53.67	30.4816

3. Appendix

3.1. Collapse of accuracy for Sigmoid Activation

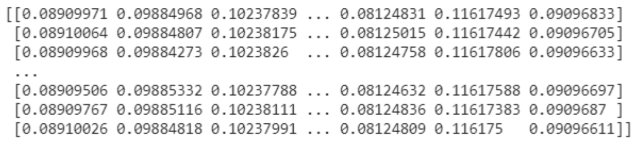
The model label ground truth for the kth label is denoted by y_k where in $k \in 1, 2, .. \ , 9, 10$ in this example. The prediction for that label is denoted by $\hat{y_k} \in [0, 1]$, which acts as a certainty of it being a certain label. $\mathbf{W_k}$ denotes the weight matrix of the final layers and $\mathbf{x_k}$ denotes the ouputs from the sigmoid activation function of previous layer. The softmax activation function is used for the final layer neurons of classification tasks, and in this task. Such that:

$$\hat{y_k} = \frac{e^{\mathbf{W}_k \mathbf{x}_k + \mathbf{b_k}}}{\sum_i e^{\mathbf{W}_i \mathbf{x}_i + \mathbf{b_i}}}$$

If $\mathbf{x_k}$ has saturated to a value very close to 0, $\mathbf{W}_k \mathbf{x}_k \approx \mathbf{0}$. In keras biases are initialised as 0, if gradient vanishes biases will stay around 0, in this case:

$$\hat{y_k} \approx \frac{e^0}{\sum_i e^0} \approx \frac{e^0}{10 \cdot e^0} \approx \frac{1}{10} \quad \text{for all k.}$$

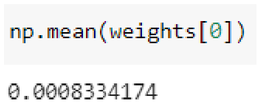
Since each label has the same probability of being true, guesses are essentially being made randomly, hence why there is an accuracy of approximately $\frac{1}{10}$. To confirm this, we inspect the values of parameters in the network where this issue is occuring:



```
[[0.08909971 0.09884968 0.10237839 ... 0.08124831 0.11617493 0.09096833]
 [0.08910064 0.09884807 0.10238175 ... 0.08125015 0.11617442 0.09096705]
 [0.08909968 0.09884273 0.1023826  ... 0.08124758 0.11617806 0.09096633]
 ...
 [0.08909506 0.09885332 0.10237788 ... 0.08124632 0.11617588 0.09096697]
 [0.08909767 0.09885116 0.10238111 ... 0.08124836 0.11617383 0.0909687 ]
 [0.08910026 0.09884818 0.10237991 ... 0.08124809 0.116175   0.09096611]]
```

Figure 4: output matrix

Here we see that indeed $\hat{y_k} \approx \frac{1}{10}$ for each class.

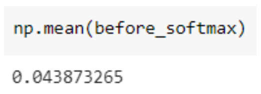


```
np.mean(weights[0])

0.0008334174
```

Figure 5: average value of weights, $\mathbf{W}_i$

The mean value of the weights is very close to 0.

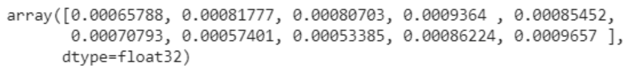


```
np.mean(before_softmax)

0.043873265
```

Figure 6: mean value og the outputs from the sigmoid activation function in the penultimate layer

Both $\mathbf{W}_i$ and $\mathbf{x_i}$ are very close to 0, thus indeed $\mathbf{W}_k \mathbf{x}_k \approx \mathbf{0}$. Finally we inspect biases, which confirms the result:



```
array([0.00065788, 0.00081777, 0.00080703, 0.0009364 , 0.00085452,
       0.00070793, 0.00057401, 0.00053385, 0.00086224, 0.0009657 ],
      dtype=float32)
```

Figure 7: biases of final layer

3.2. Architecture notation clarification

If we look at the example (8,16,32,MPool,64,MPool,128,D(16),D(10)) , this represents 5 convolutional layers, the first with 8 filters, the second with 16, third with 32, fourth with 64 and the final with 128. There are two MaxPooling layers denoted by MPool, one in between the third and fourth layers and one in between the fourth and fifth layers. After the convolutional layers there are 2 Dense layers, one with 16 neurons and the final layer with 10 neurons, denoted as D(16) and D(10).

References

1. https://alexlenail.me/NN-SVG/AlexNet.html