

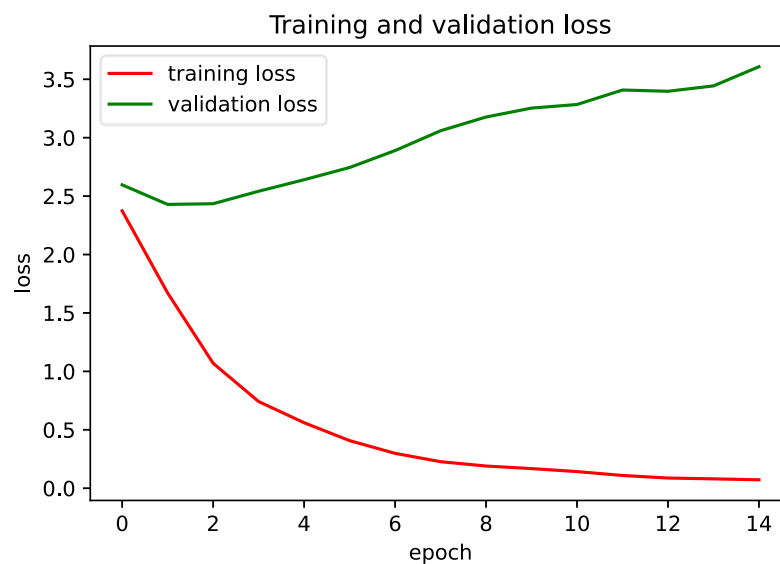
COMP5623 Coursework on Image Classification and Visualizations with Convolutional Neural Networks – ImageNet10

Name	Benjamin Jeffrey
Student username & ID	mm20bwj, 201479413

QUESTION I [55 marks]

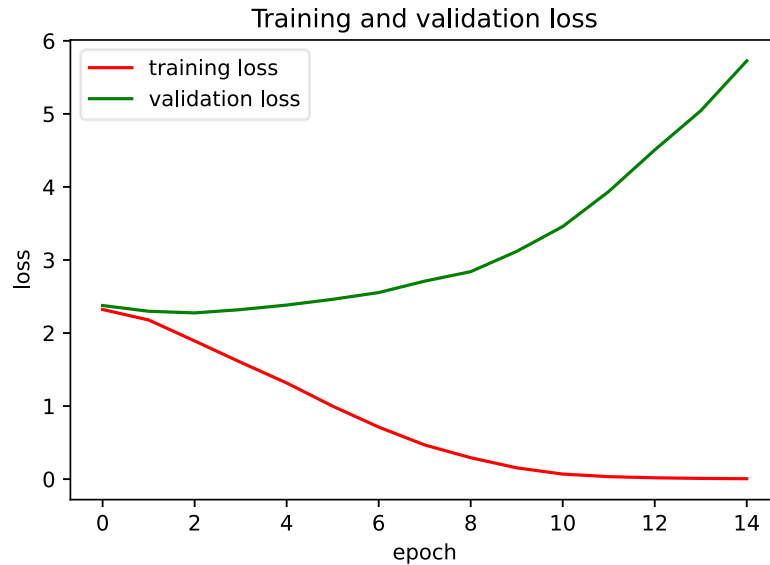
1.1 Single-batch training [16 marks]

1.1.1. Display graph 1.1.1 (training & validation loss over training epochs) and briefly explain what is happening and why. [4 marks]



The model seems to be training well on the training data, with the training loss quickly decreasing. However, after the 3rd epoch the validation loss already starts to increase. Meaning that as we keep updating the networks parameters to decrease the loss for the training data, the loss for the validation data increases. In other words, our model becomes better and better at predicting the training data but worse at predicting the validation data. So even with just one linear layer the model seems to be overfitting and does not generalise well, as it only performs well on the specific batch of data used for training.

1.1.2 Display graph 1.1.2 (training & validation loss over training epochs, with modified architecture) and explain how and why it shows that the model is overfitting the training batch. [8 marks]



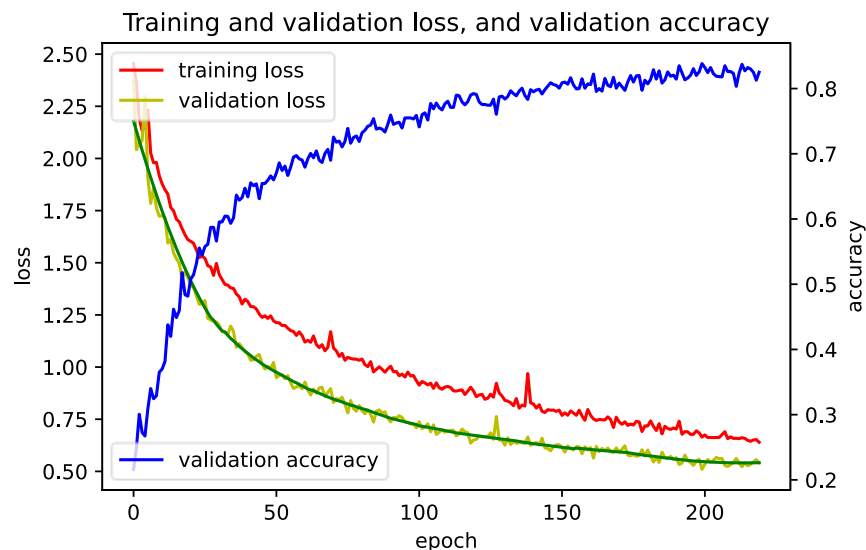
As the network was already overfitting with one fully connected layer there was technically no need to further increase the model complexity. However, by adding some convolution and pooling layers we can observe that the validation error increases much more drastically. The model still trains well and effectively decreases the training loss, which is to be expected from a network this complex and using only 64 images to train. After epoch 12 it seems that the model has essentially just memorised the training data as the training loss starts to hit 0. At this point the model would have 100% accuracy at prediction on the training data. The model achieved this by learning a lot of noise inherent to the training data however, as a lot of the little details it learned from the data are most likely only applicable to the specific batch used to train this model. As is evident again from the rapidly rising validation loss, this model is definitely overfitting as it performs rather poorly at predicting anything other than the training data. The model therefore does not generalise well due to its high variance and we should investigate methods to combat overfitting if this persists while training with the full training set. As adding more training data usually helps to reduce overfitting the model might not overfit when using the full training set.

1.1.3 Fill in table 1.1.3 (your adjusted architecture after single-batch training), adding rows and columns as necessary. [4 marks]

Input channels	Output channels	Layer type	Kernel size	Add. Info
3*128*128	8*124*124	Conv2d, ReLU	5	0 Padding
8*124*124	8*62*62	MaxPool2d	2	Stride=2
8*62*62	16*58*58	Conv2d, ReLU	5	0 Padding
16*58*58	16*29*29	MaxPool2d	2	Stride=2
16*29*29	64	Linear, ReLU	-	-
64	10	Linear	-	-

1.2 Fine-tuning on full dataset [18 marks]

1.2.1 Display graph 1.2.1 and indicate what the optimal number of training epochs is and why. [4 marks]



To determine the optimal time to stop training the network we used a smoothed version of the validation loss, shown green in the graph above. By using the smoothed line, we could determine when the validation loss started to flatten out and stop the training automatically. From previous experiments we know that continued training would not further decrease the validation loss or even increase it, making any further epochs a waste of time or detrimental. An increase in the validation loss would indicate overfitting which we want to avoid, as that would make the model likely worse at predicting on unseen data. We can also see from the graph above that the validation accuracy flattens out as well, which is expected but reaffirms that further training would not have served any purpose. Our final model stopped training at 220 epochs.

Somewhat counterintuitively, the validation loss seems to be lower than the training loss throughout the training process. This is likely due to the extensive transformations applied to the training data, for the purposes of data augmentation. Some of the transformations like crops and perspective changes likely make the training data a more difficult challenge to classify than the validation data which we did not transform beyond the necessary resize and normalisation operations.

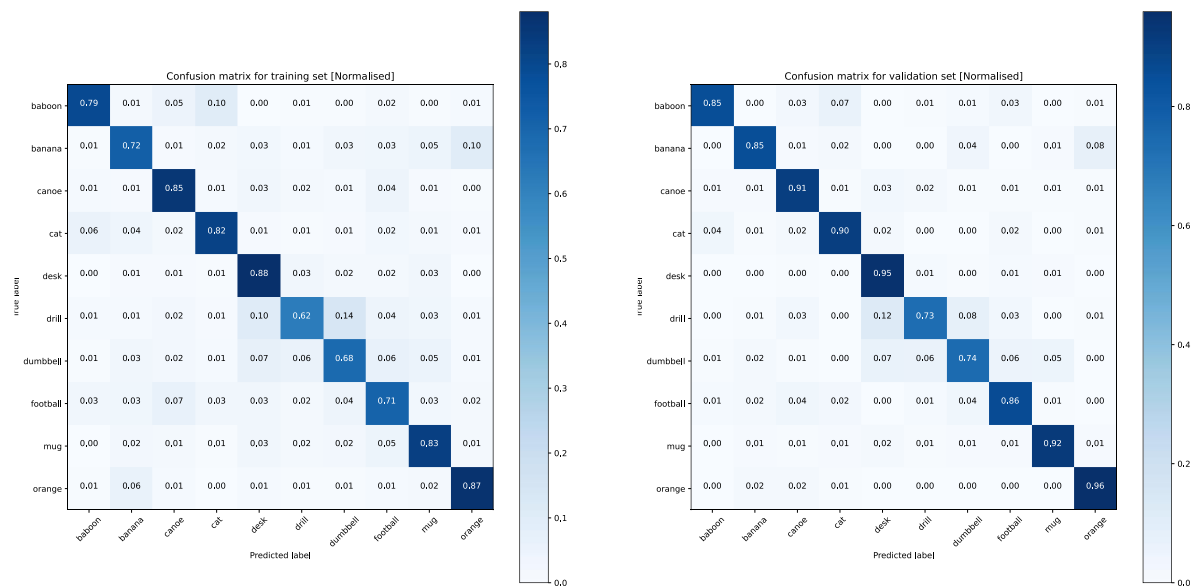
1.2.2 Describe in detail your fine-tuning process on the complete dataset, including any adjustments you made to the network or training process to increase prediction accuracy. Explain why these adjustments increased accuracy. [10 marks]

Model	Batch size	Image size	Transforms	Model layers	Model parameters	Epochs	Train loss low	Val loss low	Val acc high	Train time (min.)
v1	64	128*128	0	10	865,722	20	0.022	1.439	51.3%	5.63
v2	64	128*128	1	10	865,722	50	1.103	1.144	61.9%	13.85
v3	64	128*128	1	22	39,936,954	50	0.843	1.076	64.3%	20.53
v4	64	128*128	2	27	25,517,754	100	0.951	1.027	66.3%	38.59
v5	64	128*128	5	38	34,831,210	132	0.581	0.782	75.1%	90.40
v6	64	128*128	7	38	34,831,210	128	0.685	0.594	80.4%	86.72
v7	32	192*192	7	44	36,012,394	220	0.640	0.510	83.8%	412.35

The above table shows the various versions our model went through and the parameters that were used. We started with the relatively simple network from table 1.1.3. It was quickly apparent that the model was still overfitting using the entire training set. We therefore changed the resize and centre crop transforms to a random crop of 128 * 128 to get more variation in the training set. This reduced the overfitting, but the network did not perform that well, so we opted to increase the model complexity. We increased the number of convolution layers, the number of feature maps, the number of max pool layers and added an additional linear layer as well. We generally used ReLU layers after every other layer, excluding the pooling layers. These measures increased the number of trainable parameters from ~850,000 to ~40,000,000, a drastically more complex model. The performance benefit from this were not as high as expected however, with the suspected culprit being overfitting as the validation loss levelled out much earlier than the training loss. So, for v4 of our network we dialled down the complexity slightly and added a random horizontal flip to the dataset transforms. This was not enough to combat the overfitting however, so we decided to get serious with regularization methods for the next version. We added batch normalisation layers before each ReLU operation and also added random perspective, random rotation and colour jitter to our dataset transforms. We saw a significant performance improvement with this version. This is where it gets slightly complicated. For v6 we realised that applying the transforms to the validation set does not really make much sense. We did not apply them anymore therefore but did not contemplate the effect this would have on the performance metrics immediately. We therefore assumed another jump in performance with v6, which had two more transform operations applied to the training set. After re-running v5 without the transforms on the validation data later we discovered that it actually performed slightly better than v6 with a shorter required training time, making it overall the most sensible architecture. We also trained a more complex network for v7 with several convolution layers added. However, we could not train it with a batch size of 64 due

to memory constraints. The training with smaller batches on this network took about 5 times longer than the previous version at only a marginal increase in performance. We also tried different learning rates and batch sizes throughout this process but neither had any performance benefits. Since we went through the effort of training v7 and it did have the highest validation accuracy overall we still used this version for the next steps, but overall v5 seems to have been the most efficient model. Please see the code for an overview of the architecture, as it is rather large.

1.2.3 Display two confusion matrices 1.2.3 (one each for complete validation set and complete training set) for your final trained model and interpret what is shown. [4 marks]



From the two confusion matrices we can see that they look mostly the same for the training and validation set. We normalised the values in order to better compare the matrices. The validation set overall shows a higher accuracy of the model, but this was to be expected after seeing the validation loss was lower than the training loss (discussed in section 1.2.1). We can also deduce that drills and dumbbells were the most difficult classes to get right. Drills sometimes got misclassified as desks or dumbbells and dumbbells were occasionally misclassified as desks, drills, footballs or mugs. Furthermore, bananas were sometimes wrongly identified as oranges and baboons and cats were at times mixed up. Overall, though, most instances were correctly classified and lie on the diagonal of the confusion matrix which indicates quite strong performance.

1.3 Evaluation and code [21 marks]

1.3.1 Please include `[my_student_username]_test_preds.csv` with your final submission. [8 marks]

1.3.2 Please submit all relevant code you wrote for Question I in Python file `[my_student_username]_q1.py`. No need to include the config or ImageNet10 files. [13 marks]

No response needed here.

QUESTION II [45 marks]

2.1 Preparing the pre-trained network [20 marks]

2.1.1 Read through the provided template code for the AlexNet model *alexnet.py*. What exactly is being loaded in line 59? [2 marks]

Pytorch Hub is a pre-trained model repository. Using the `torch.hub.load()` function we can load a model from a GitHub repo or a local directory. In this case we load the pre-trained 'alexnet' model from the 'vision:v0.6.0' GitHub repository provided by pytorch themselves and then assign it to our local variable 'model'. This gives us access to use the model like if we created it ourselves, but it comes with its parameters already pre-trained.

2.1.2 Write the code in *explore.py* after line 50 to read in the image specified in the variable `args.image_path` and pass it through a single forward pass of the pre-trained AlexNet model. [5 marks]

2.1.3 Fill in function `extract_filter()` after line 84 extracting the filters from a given layer of the pre-trained AlexNet. [4 marks]

2.1.4 Fill in function `extract_feature_maps()` after line 105 extracting the feature maps from the convolutional layers of the pre-trained AlexNet. [6 marks]

Please submit all your Question II code in a Python file *[my_student_username]_explore.py*.

No response needed here.

2.1.5 Describe in words, not code, how you ensure that your filters and feature maps are pairs; that the feature maps you extract correspond to the given filter. [3 marks]

The extracted filters have 4 dimensions, e.g. (64, 3, 11, 11), where 64 is the number of output channels, or feature maps, 3 is the number of input channels and 11*11 is the filter size. The extracted feature maps have 3 dimensions, e.g. (64, 63, 63), where each feature map is of the size 63*63 and there are 64 of them. The numbers are from the first convolution layer and can change from layer to layer. We can see that filters have an additional dimension corresponding to the number of input channels, in this example 3. This is because every output feature map is influenced by all input channels. There are therefore x (= number of input channels) times more filters than there are feature maps. The first dimension of the extracted filters corresponds to the output channels of the convolution layer, so the specific feature map that was influenced by the filters in that channel. As long as we make sure to match the feature map channel with the output channel of the filters, we ensure that the selected feature map was influenced by the filters with the same output channel. In practice this really just means using the same index for the first dimension of both ndarrays. So for the previous example, selecting `filters[0]` returns an ndarray of shape (3, 11,

11) corresponding to the 3 filters of size 11*11 that influenced the 0th feature map, which in turn could be accessed by using feature_maps[0] and would have the shape (63, 63).

2.2 Visualizations [25 marks]

2.2.1 For three input images of different classes, show three pairs of filters and corresponding feature maps, each from a different layer in AlexNet. Indicate which layers you chose. For each pair, briefly explain what the filter is doing (for example: horizontal edge detection) which should be confirmed by the corresponding feature map. [15 marks]

Image #1, class: giant panda

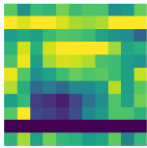


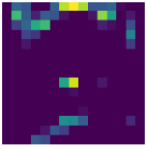

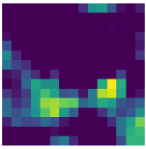
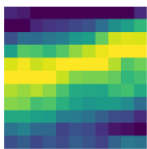


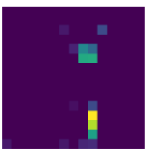
	Filter	Feature map	Brief explanation
Early layer: 1 st convolution layer			Seems to be detecting brightness, with all the dark parts of the image being activated, so the black parts of the panda are visible.
Intermediate layer: 3 rd convolution layer			Some edge detection seems to be happening on the curve of the panda's head.
Deep layer: 5 th convolution layer			Hard to tell at this point, maybe some brightness detection again. This seems to be separating the panda's arms.

Image #2, class: beer glass

	Filter	Feature map	Brief explanation
Early layer 1 st convolution layer			Looks like horizontal edge detection. The glass rim, fingers and lines in wooden table are visible.
Intermediate layer: 3 rd convolution layer			Maybe vertical edge detection. Seems to be picking up the edge of the glass.

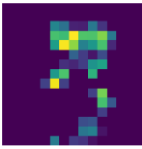
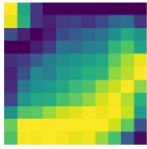
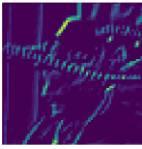

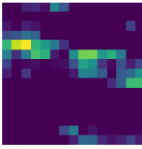

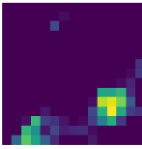
Deep layer: 5 th convolution layer			Outlines rough shape of glass, maybe based on contrast.
--	---	--	---

Image #3, class: sombrero

	Filter	Feature map	Brief explanation
Early layer 1 st convolution layer			Seems to be picking up all edges well. Clearly outlines all objects.
Intermediate layer: 3 rd convolution layer			Probably detecting horizontal edges like the brim of the sombrero.
Deep layer: 5 th convolution layer			Seems to be picking up brightness in the image. White shirt in bottom of image is highlighted.

2.2.2 Comment on how the filters and feature maps change with depth into the network. [5 marks]

The feature maps reduce drastically in size deeper into the network due to the pooling layers and some size reduction from the convolutions, making it hard to interpret them. Due to the repeated convolutions, pixels from the original image have very widespread influence as they keep getting included in convolutions by the kernels. The number of input channels to the convolution layers also increases dramatically meaning each feature map is influenced by many more filters. The filters also reduce in size making it basically impossible to guess what effect a filter has on the output. The deeper into the network the more general the effect of the filters will be, as they influence a larger proportion of the output. This also makes it harder to tell what is actually going on and most of the guesses above might very well be completely off.

Marks reserved for overall quality of report. [5 marks]

No response needed here.