CNNs Report

Pratyay Suvarnapathaki, 2020111016

Task 3A

Sr No	Modification	F1-score	Training time (s)
0	Base Model	0.710	54.0
1	Decreasing CNN Layers	0.555	30.0
2	Increasing CNN Layers	0.656	56.3
3	Prev + Increasing number of CNN Filters too	0.711	58.8
4	Increasing MLP Layers	0.702	50.7
5	Prev while decreasing CNN Layers	0.587	31.0

Note: Layer-wise visualisations in notebook.

Observations:

1. Reducing the number of CNN layers may result in a simpler model that is less capable of learning complex patterns, leading to a lower F1-score. However, a simpler model can be trained faster due to fewer parameters, which explains the shorter training time.
2. Increasing the number of CNN layers allows the model to capture more complex features and patterns in the data, resulting in a higher F1-score than case 1. However, a more complex model requires more parameters and more computation, which explains the longer training time. Moreover, the fact that the performance isn't on par with the baseline may also be explained by the same reason. Perhaps given more epochs, it would have caught up and maybe even surpassed the baseline.
3. Increasing the number of output filters in the convolution probably compensates for the deficiency in information in case 2, yielding a similar f1 score as the baseline.
4. Increasing the number of MLP layers does not seem to impact the score or time much. This is probably because of he relatively small size of the overall model. Also, maybe the tweaks made to the MLP simply weren't significant enough.

5. Increasing the number of MLP layers while decreasing the number of CNN layers may result in a simpler model that is less capable of learning complex patterns, leading to a lower F1-score. The F1-score is barely higher than in case 1 (simply decreasing CNN layers) -> the MLP isn't an 'effective substitute' for CNNs. Really puts into perspective how powerful CNNs are!

Task 3B

Sr No	Modification	F1-score	Training time (s)
0	Base Model - 3x3 Filters	0.710	54.0
1	5x5 Filters	0.681	79.8
2	Staggering (7x7,5x5,3x3)	0.670	72.4

Note: Layer-wise visualisations in notebook.
Observations:

1. The model with 5x5 filters has a lower F1-score of 0.681 as compared to the base model. This could be because larger filter sizes are not as effective in capturing finer details as smaller filters. Additionally, larger filter sizes also increase the number of parameters in the model, which may have led to overfitting.
2. In an attempt to improve accuracy by drawing inspiration from the 'coarse-to-fine' approach, I tried this staggered approach, but it ended up not being very effective at all, probably because it doesn't really address the core problems discussed above.

Task 3C

Sr No	Modification	F1-score	Training time (s)
0	Base Model - No Dropout	0.710	54.0
1	Very High Dropout (0.4)	0.426	56.5
2	Decently Low Dropout (0.1) + 15 epochs	0.725	85.7

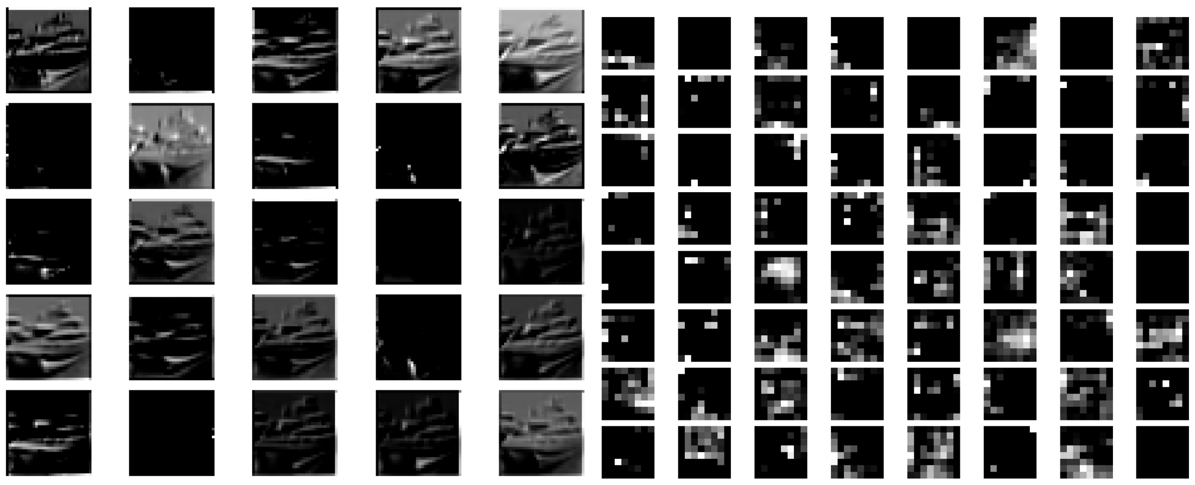
Note: Layer-wise visualisations in notebook.

Observations:

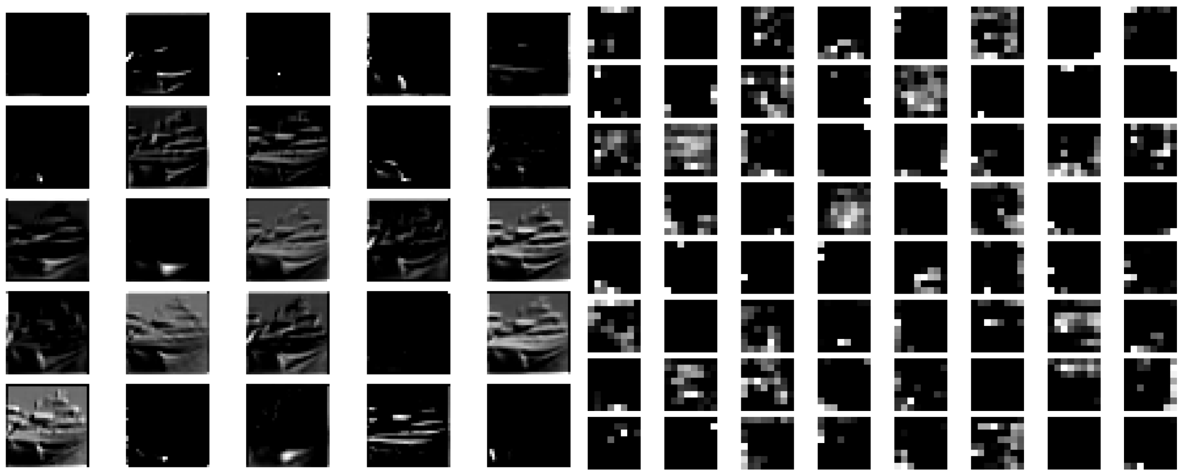
1. The significantly lower f1-score could be because such a high dropout rate is leading to significant dropout of important features during training, which leads to a less effective model.
2. The ‘decently low dropout’ leads to an f1-score about on-par with the base model. (Not explicitly shown in notebook, as I modified this in-place to 15 epochs). However, training this for 15 epochs shows a good improvement. For comparison, I have explored using 15 epochs for the baseline in Q3-F. Here, we can demonstrably see the improvement caused using dropout.

Task 3D

Baseline model, first vs final layer



Best Model (‘decent_dropout’), first vs final layer



In both cases, we see that the initial layers of the CNN capture features that us humans can relate with in a better way, this is simply because the initial features resemble the source image to a great extent. On the other hand, going deeper into the network, the features in the latent space are capture higher level details that aren't very 'relatable' to us humans. They are simply activations in the relevant areas of the encoded image.

Task 3E

The observation for Task 3D remains consistent across the three class-wise images explored. All image+filter outputs included in notebook.

Task 3F

Learning rate

Modification	Modification - Numerically	F1-score	Training time (s)
LR=0.0001	0.1x	0.444	49.3
Base Model - 0.001	x	0.710	54.0
LR=0.01	10x	0.753	48.8

A low learning rate simply does not result in the model converging in the available number of epochs. On the other hand, a rate of 0.01 gives a much higher score here. This is probably due to the fact that it is able to move closer to true convergence within the given number of epochs. However, in the long run, this might prove to be an excessive rate, because in the verbose fit() logs, I can already see the accuracy fluctuating up *and down,* indicating that the gradient descent process is already 'jumping over' local minimas.

Batch Size

Modification	Modification - Numerically	F1-score	Training time (s)
BS = 32	0.5x	0.702	79.4
Base Model - 64	x	0.710	54.0
BS = 128	2x	0.647	35.9

The f1-score for BS=32 is almost the same as for the base model (BS=64). This could be because a smaller batch size allows for more frequent weight updates, which can improve the model's performance. However, the smaller batch size has also resulted in a longer training time.
On the other hand, BS=128 has a lower f1-score, probably because a larger batch size reduces the variance in each batch, resulting in a model that is less sensitive to the individual examples in each batch.

Number of Epochs

Epoch 1/25

782/782 - 6s - loss: 1.7595 - accuracy: 0.3598 - val_loss: 1.4575 - val_accuracy: 0.4698 - 6s/epoch - 8ms/step

Epoch 5/25

782/782 - 5s - loss: 0.9474 - accuracy: 0.6680 - val_loss: 1.0078 - val_accuracy: 0.6515 - 5s/epoch - 6ms/step

Epoch 10/25

782/782 - 5s - loss: 0.6386 - accuracy: 0.7781 - val_loss: 0.8174 - val_accuracy: 0.7181 - 5s/epoch - 6ms/step

Epoch 15/25

782/782 - 5s - loss: 0.4074 - accuracy: 0.8575 - val_loss: 0.9478 - val_accuracy: 0.7151 - 5s/epoch - 6ms/step

Epoch 20/25

782/782 - 5s - loss: 0.2249 - accuracy: 0.9217 - val_loss: 1.1049 - val_accuracy: 0.7180 - 5s/epoch - 6ms/step

Epoch 25/25

782/782 - 5s - loss: 0.1120 - accuracy: 0.9607 - val_loss: 1.3349 - val_accuracy: 0.7303 - 5s/epoch - 6ms/step

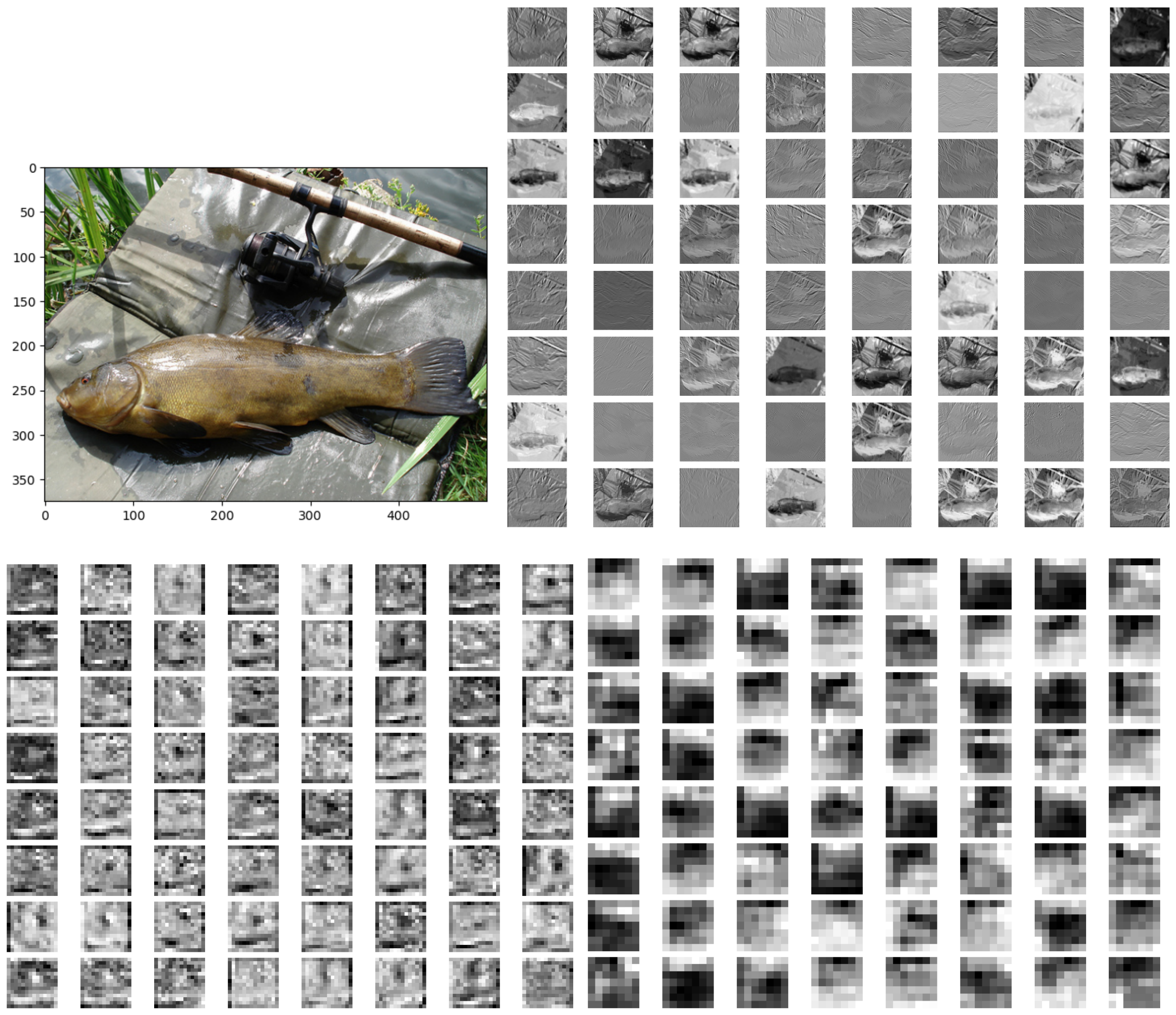
At the end,
Accuracy: 73.030
F1 score: 0.731
Time taken: 120.9 seconds

We see the typical ‘diminshing returns’ pattern emerge as the model converges. Most of the learning is complete by epoch 10.

Task 4

Experiments D and E

Image 1, Layers1, 100, and 186 of ResNet

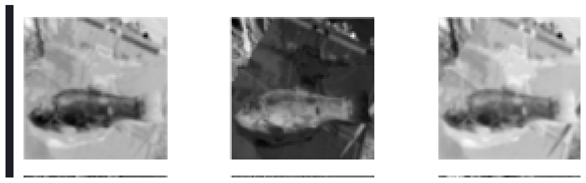


Corresponding outputs for two other images representative of their respective classes are included in the notebook.

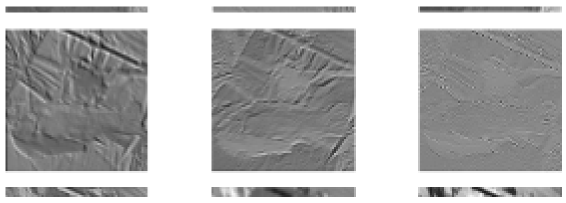
Observations:

- As with Task 3D+E, we observe a similar trend where feature maps for initial layers are more human-relatable, while they get more abstract as higher level details are captured in the latent space upon going deeper.

- In contrast to Task 3, however, we observe that the features are captured in much higher fidelity overall. This is probably due to resnet's intricate architecture and the inherent higher quality of input images.
- Moreover, upon examining the features after the first CNN layer, we can clearly see how some particular filters spot some particular visual trends, as explained in the CNN tutorial. For example, these channels here:

  Seem to detect the fish more, while these ones here:

  Focus more on the background wrinkles, ignoring the fish entirely. Such a stark contrast was missing / not entirely obvious in Task 3 (probably again due to the inherent quality of the images).
- Interestingly, some semblance of human-relatability is retained all the way till layer 100.

Experiment F

Note: Accuracy is being used as the metric here instead of f1-score.
Also, due to a personal oversight, the time taken for training in Q4 has been reported manually, instead of using callbacks as in Q3. I feel this is okay because the 'time_taken' metric is somewhat unreliable as-is (varies considerably between executions).

Learning rate

Modification	Modification - Numerically	Accuracy	Training time (s)
LR=0.0001	0.1x	0.553	340
Base Model - 0.001	x	0.552	423
LR=0.01	10x	0.471	352

A marked depature from task 3, the low learning rate here is enough to arrive at a similar accuracy as the base model. This probably has to do with simply how good resnet is, with the MLP on top not playing much of a role. However, with a higher rate, the model is unable to converge at a minima at all.

Batch Size

Modification	Modification - Numerically	Accuracy	Training time (s)
BS = 16	0.5x	0.530	401
Base Model - 32	x	0.552	423
BS = 64	2x	0.554	364

Keeping aside the base model's anomalously high training time, the results make sense. The best batch size seems to be 64 here, with a lower training time anf higher accuracy than the others.

Epochs

Epoch 1/9: 0.5292	Epoch 2/9: 0.5401	Epoch 3/9: 0.5521
Epoch 4/9: 0.5649	Epoch 5/9: 0.5651	Epoch 6/9: 0.5628
Epoch 7/9: 0.5284	Epoch 8/9: 0.5483	Epoch 9/9: 0.5534

Clearly, the default rate (chosen because it also happens to be the default rate for keras' Adam optimiser) overshoots past the minima here. However, it manages to recover back just in time. The trend is clear when observing accuracies at 3-epoch-intervals. Epoch 3 yields an accuracy of 0.55, epoch 6 being a slight improvement, and Epoch 9 being almost the same as 3 due to the overshoot in Epochs 7 and 8 that the model recovered from.

Training times:
Epochs 1 to 3: 342s
Epochs 3 to 6: 340s
Epochs 6 to 9: 338s
Training times are pretty much constantper epoch, so the relationship of num_epochs vs total training time will be almost linearly proportional.