

CS577 – Deep Learning

Homework 4

Niti Wattanasirichaigoon
A20406934

Theoretical Questions

$R =$

1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

 $G =$

2	2	2	2
2	2	2	2
2	2	2	2
2	2	2	2

 $B =$

1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4

 Filter =

1	1	1
1	1	1
1	1	1

1. Without zero padding

$9 + 18 + 18$	$9 + 18 + 18$
$9 + 18 + 27$	$9 + 18 + 27$

 =

45	45
54	54

2. With zero padding

18	27	27	18
30	45	45	30
36	54	54	36
26	39	39	26

3. Dilated convolution (with zero padding)

20	20	20	20
24	24	24	24
20	20	20	20
24	24	24	24

4. Template matching in convolution can be thought as matching small parts of the image with a filter. If a portion of the picture aligns with the template, the convolution value (weighted sum) will be high, indicating that the pattern is present in that portion.

5. Multiple-scale analysis can be achieved using a series of convolutions. After each convolution, the image gets smaller and smaller while each pixel (value) represents the corresponding area in the image before. The decreasing layer size represents a pyramid structure.
6. To compensate for the spatial resolution decrease, we increase the depth (number of channels) by using more filters. This is done to keep the number of coefficients (weights) the same.
7. 128x128x32 tensor and 16 3x3x32 filters. The output tensor without zero padding will have decreased length on each side and channels equal to the number of filters:
output: 126x126x16
8. $\text{Size} = (w-k+2p)/s + 1$
size = $(128-3+0)/2 + 1 = 62 + 1 = 63$
output: 63x63x16
9. The number of channels can be reduced using a 1x1 convolution by determining the number of 1x1 filters used. For example, if we have a 64x64x32 tensor and 16 1x1x32 filters, the resulting tensor will be 64x64x16.
10. Convolution layers can be thought of feature matching between small patches of the image (windows) and filters. Early layers will deal with more basic features found among the image, while deeper layers represent a higher level convolution between several features.
11. Max pooling output of I in question 1:

1	1	2	2	2	2
1	1	2	2	4	4

12. Max pooling is done to reduce the image size while also returning information about the underlying window. The presence of a high convolution value in the window will indicate whether a feature is there or not.
13. Data augmentation is the process of modifying the original image in different ways to create a lot more variety in input (cropping, rotating, flipping etc.). Data augmentation is extremely useful when our training data set is small since we can use it to synthesize a lot more training data.
14. Transfer learning is the process of using pre-trained weights as a part of our model. The pre-trained weights are usually trained over millions of inputs, giving it robustness and making our feature selection process strong even if we have a small training data set.
15. We need to freeze the coefficients of the pre-trained network or else they will get changed (updated) during model training. We focus on training only the dense layers of our model first.

16. The coefficients of a pre-trained network can then be fine-tuned to our model by gradual training; after some period of training the dense layers, we unfreeze some layers of the pre-trained network to allow them to adapt to our problem.
17. Inception blocks are used as opposed to the densely connected architecture to reduce the number of parameters and resources used for computation (very deep networks have the risk of vanishing gradients). Most of the parameters lies within the dense layers, so inception models compensate this by growing the network sparsely. A traditional deep CNN will have much more parameters and is also more prone to overfitting. Inception networks like GoLeNet have multiple outputs to help pass gradients from different depths of the network.
18. The advantage of residual blocks is that information can pass through even if the gradients are reduced to zero. This also allows the model to learn more easily since deviation from the identity is easier to calculate compared to a function. In the case where weights are reduced to zero, the identity is passed which allows for learning in other areas in the network (as opposed to passing a zero gradient).
19. Intermediate activations of convolution layers can be visualized by giving an input to a trained modified model. The modified model will give us activation outputs from each layer. Since these outputs are the product of the image against the weights of a layer, visualizing these outputs will give us a sense of what each layer is doing (what features is it detecting) and better understand our model.
20. The filter weights of a trained convolution layer can be visualized by finding an imaginary output that maximizes the layer's activation. To do this, we apply gradient ascent to find the input that optimizes the filter activation, normalize, and then visualize it. The resulting images we get from each filter represents the patterns that each of them are working by. Lower level blocks usually detect simpler features such as edges while higher level blocks will usually detect more sophisticated patterns.
21. Class activation heatmaps can be visualized by first feeding an image to a network. Then we compute gradients of a selected output node with respect to each channel of the target layer where activation is to be computed. The activation outputs are then weighted by their average gradients, resulting in our activation map. We then superimpose the activation onto the original image to visualize the heatmap. We visualize heatmaps to understand which parts of the image contributes the most to layer activation. If the heatmap shows that our layer is activated by some unintended features of our image, then we know something in our network is off.

Programming Questions

1. Binary Classification

Problem Statement

Download the cats and dogs dataset, choose a subset of 2000 cats and 2000 dog images to use for training, validation, and testing data. Build a CNN classifier including several convolution, pooling and normalization layers, flatten the data between convolution and dense layers. Evaluate performance and tune hyper parameters. Visualize activation of some convolution layers and filters and draw a conclusion. Replace convolution layers with VGG16's base, train with frozen weights. Unfreeze the convolution base and continue to train. Modify data generators for data augmentation and retrain with frozen weights model and evaluate the results.

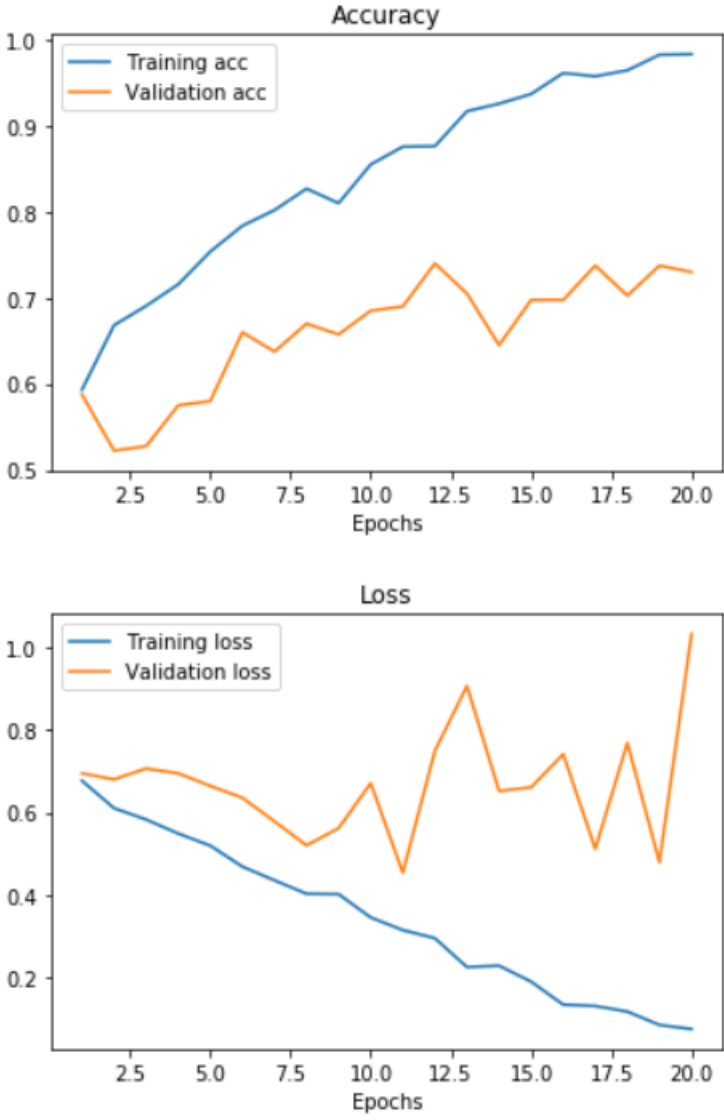
Implementation Details

I randomly selected 2000 cats and 2000 dog images from the downloaded dataset and split each of them into 1200 for training, 400 for validation, and 400 for testing. The initial model will have several convolution layers each followed by max pooling. I will include a layer of batch normalization. After flattening, there is a dense layer then an output layer with one sigmoid node (binary classification) using binary cross-entropy loss and RMSprop optimizer. The model will be evaluated by loss and accuracy.

Results and Discussion

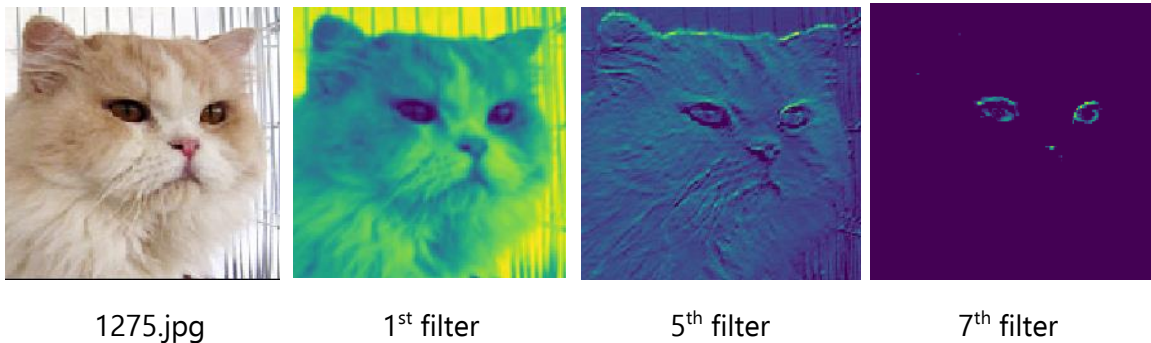
A total of 4 models were trained in the first step, tested over different hyper parameters. All plots can be found in python notebook.

Model	Description	Evaluation
1	The model contains a 32 3x3 conv layer, a 64 3x3 layer, a batch normalization layer, and a 128 3x3 layer. Each conv layer is followed by max pooling. After the conv block, the data is flattened and followed by dense layers of 512, 64 and 1 (output) nodes.	Validation loss values diverge in values from 1-6. Validation accuracy lies around 0.6-0.7.
2	Similar to model1, but moved the batch normalization layer to be after the first conv layer and add an additional 128 3x3 conv layer before flattening.	Validation loss varies between values of 0.6-1.2. Validation accuracy approximately 0.7.

3	Similar to model 2 but this time the batch normalization layer is moved to be after the last conv layer before flattening. The dense layers are reduced to two layers with 128 and 1 (output) nodes instead.	Validation loss still lies around 0.6 with occasional large spikes. Accuracy barely above 0.7
4	<p>Similar to model3 but using a SGD optimizer with learning rate of 0.001 and momentum of 0.9 instead of RMPprop.</p>  <p>The figure consists of two line graphs. The top graph, titled 'Accuracy', plots accuracy against epochs (0 to 20). The y-axis ranges from 0.5 to 1.0. The training accuracy (blue line) starts at approximately 0.6, rises to about 0.75 by epoch 5, and then continues to increase with some fluctuations, reaching nearly 1.0 by epoch 20. The validation accuracy (orange line) starts at approximately 0.6, drops to about 0.55 by epoch 2.5, and then fluctuates between 0.65 and 0.75 for the remainder of the training process. The bottom graph, titled 'Loss', plots loss against epochs (0 to 20). The y-axis ranges from 0.2 to 1.0. The training loss (blue line) starts at approximately 0.7, decreases steadily to about 0.4 by epoch 10, and then continues to decrease more slowly, reaching approximately 0.1 by epoch 20. The validation loss (orange line) starts at approximately 0.7, fluctuates between 0.5 and 0.8 for most of the training process, and then spikes sharply to over 1.0 at epoch 20.</p>	Less variation in validation loss, spreading from 0.5-1.0. Validation accuracy saturates around 0.7

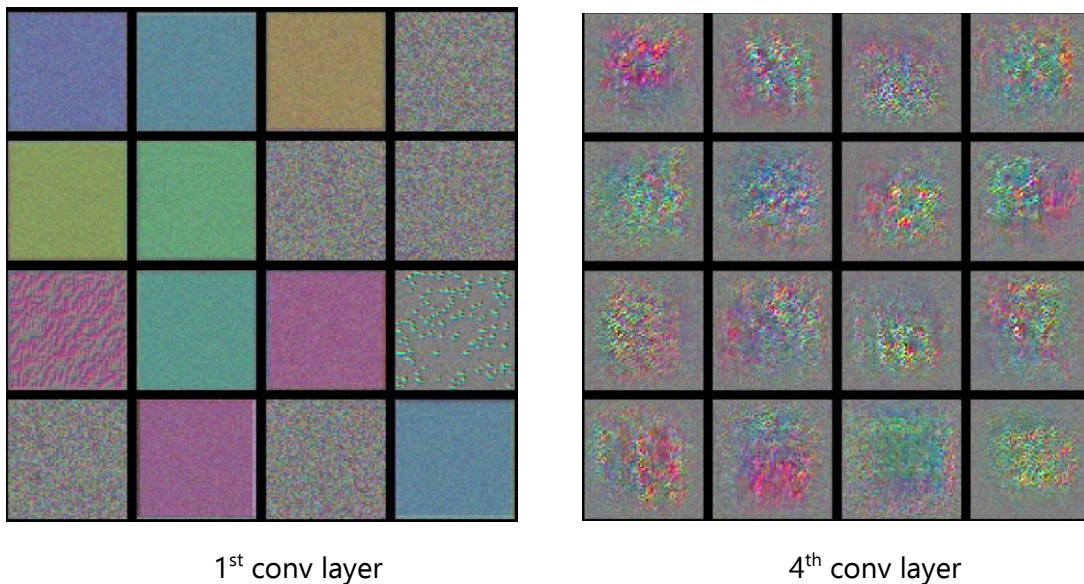
Although all models ended up having a validation accuracy of around 0.7, **model 4** was chosen to be the best model because of its less extreme swings in validation loss and a clear ability to overfit the training data. The model was retrained at its optimal number of epochs (19 epochs) and evaluated using the test data. **Loss: 0.29274, Accuracy: 0.71250.**

To visualize layer activations, I imported the cat-1275 image and use it as my input and created layer outputs to view activations. Examining the first layer activations, I plotted the activation output images to learn about the filters. Here are some filters from the first layer.



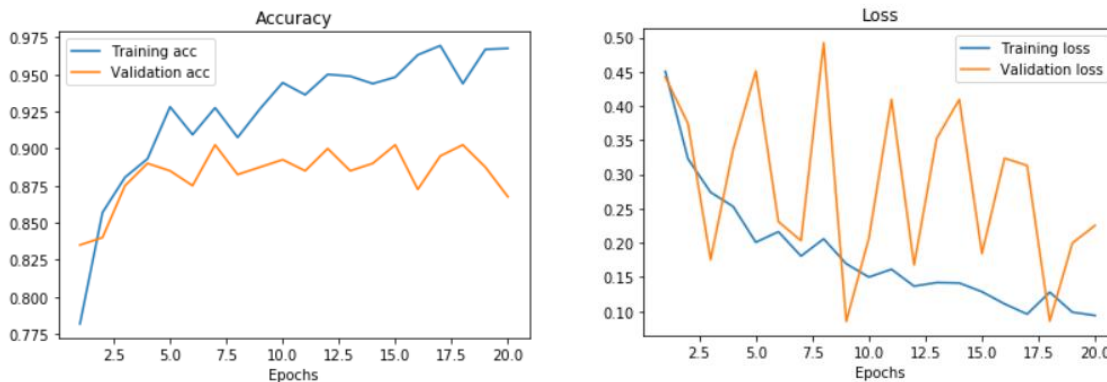
We can see how these filters in the first layer are activated differently (bright areas). The 1st filter most likely activates from 'white' regions in the picture, the 5th filter detects horizontal edges, and the 7th filter detects edges around the eyes.

We then visualize our trained filters by defining a function that implements gradient ascent to find the input that best activates a filter. It will loop over filters of a selected layer and output the filter images. I plotted the first 16 filters of the 1st and 4th conv layers in the best model.



We can see how the filters of the first conv layer detects lower level pattern such as texture of fur and possibly spots. The 4th conv layer filters appear to activate around the middle of the image. This is because it focuses more on the center of the image to detect higher level features like facial structures of cats and dogs.

In the next part I loaded pre-trained conv layers of VGG16 and used them as a base for a new model. The dense layers of VGG16 were excluded and the conv layer weights were frozen before I started the training this model (model5) for 20 epochs.



The model performed even better than model4 which was expected. It has a validation loss oscillating around 0.1 and 0.5 and a validation accuracy in between 0.85 and 0.90. Evaluating this model with the test set, we get a **Loss of 0.03216** and an **Accuracy of 0.85500**.

The model's 5th conv block (conv_block5) was then unfrozen to allow for some final tuning. I then continued to train the model for an additional 15 epochs and evaluated the results again. This model was able to reach a **Loss of 0.10004** and an **Accuracy of 0.88999**.

Finally I redefined the data generators to include data augmentation. The new training data generator is then used to fit our best model (model5 with conv weights frozen) and evaluated by the usual test set. The loss and accuracy plots are similar to those of the previous model5's. The final evaluation however, gave us a **Loss of 0.24643** and an **Accuracy of 0.87000**. We would expect this version of the model to perform better than the previous one because it implements data augmentation. This difference is not that significant and may be caused by overfitting of our small validation set or an imbalanced selection of the training set.

2. Multi-class Classification

Problem Statement

Download the CIFAR-10 dataset into Keras and build a CNN multi-class classifier. Test the model performance and tune hyper parameters. Add one or two inception blocks and test performance. Remove the inception blocks and add one or two residual blocks instead. Evaluate and compare the results.

Implementation Details

The initial model will consist mainly of three blocks. The first block will have two 32 3x3 Conv2D layers followed by max pooling and a dropout (0.2). The second block is the same as the first, but with 64 3x3 Conv2D layers instead. The layers will be flattened before entering the fully connected layer block which contains 512 nodes followed by an output layer of 10 nodes. The layers use relu activation except for the output (softmax). The model uses an RMSprop optimizer and loss/accuracy for the evaluation metric. The model hyper parameters will then be tuned further.

For the inception model, I added an inception block in between the second block and the dense layer block. The inception model consists of three branches: a 64 3x3 conv layer, a 64 3x3 conv layer followed by a 64 3x3 conv layer with 'same' padding, and a 64 3x3 conv layer followed by max pooling of stride 1 and 'same' padding. The branches are concatenated before flattening.

For the residual model, I replaced the inception block with a residual block. The residual block consists of two 128 3x3 Conv2D layers and a 64 1x1 Conv2D layer to match the size with the input. The input and output of the residual block are concatenated before a final max pooling.

Results and Analysis

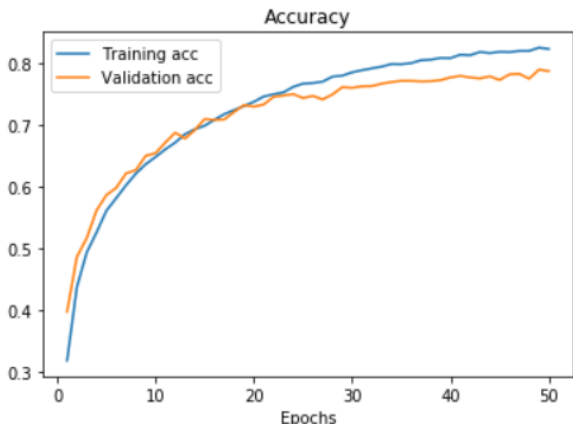
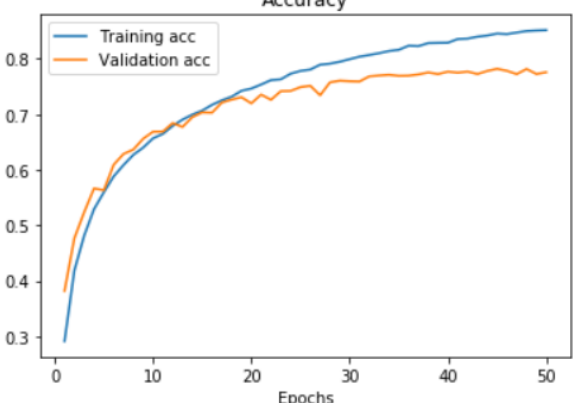
The initial model was trained over 20 epochs and found that the validation loss and accuracy is still improving. The second model was trained over 50 epochs, and an additional dropout (0.25) layer was added between the dense layer and the output layer.

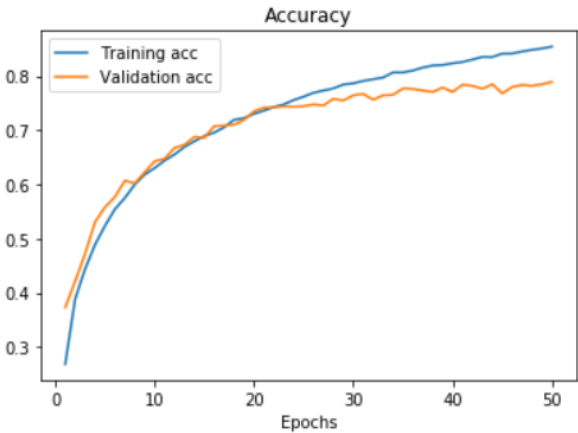
Model	Description	Evaluation
1	The model contains two 32 3x3 conv layers, max pooling, dropout (0.25), followed by two 64 3x3 conv layers, max pooling, dropout (0.25), flattening, dense (512) then dense (10) as the output. Trained over 20 epochs	Validation loss and accuracy continuing to improve.

2	Similar to model1, but has an additional dropout (0.25) between dense (512) and output layer and trained over 50 epochs	Validation loss saturates around 0.7. Validation accuracy around 0.78.
---	---	--

Model2 was then evaluated using the test set, yielding a **Loss of 0.63801**, and an **Accuracy of 0.73830**. This model will serve as a control for the next model experiments.

The inception block was constructed using the functional API, but to have the exact same structure as model2, then an additional inception block is added in between the second conv block and the fully connected block. The residual block was constructed in similar fashion. The results are as follows. (All models trained for 50 epochs.)

Model	Description	Evaluation
2	Model2 (control) 	Best validation acc: 0.7890 Test accuracy: 0.7838 Training time: 1050 sec
3	Model2 with inception block 	Best validation acc: 0.7819 Test accuracy: 0.7748 Training time: 1311 sec

4	<p>Model2 with residual block</p>  <p>Best validation acc: 0.7891 Test accuracy: 0.7807 Training time: 1147 sec</p>	
---	--	--

The performance of all three models are very similar. This is most likely because of the sheer amount of image data we have this time (40000 for training, 10000 for validation, and 10000 for testing), that slight changes in the model would not have significant effects. All the models have an evaluated **accuracy** of around **0.78**.

We can see from the time used to train, however, that the residual model was able to train faster than the inception model. Although the number of parameters might be different, it is expected that the residual model should be easier to train. The residual model took longer than the control because the residual block is an extra addition.

In order to see more distinct differences between these three models, we can either 1) lower the amount of data used or 2) add more complicated structure to the inception and residual models.