Cera Oh
CS510 CV and DL
Summer 2023
8/01/2023

Programming 3 Report

Introduction

I'm turning in one python code named "CNN.py". We were given a folder of dataset with images of cats and dogs. I deleted the unnecessary second folder from the dataset. The dataset was loaded using *tf.keras.utils.image_dataset_from_directory()* call with batch size 10 and *shuffle* set to *True*. I also resized the image with the above call by setting the image size to (150, 150) and *crop_to_aspect_ratio* set to *True*. I have hard coded the folder paths to be the same location as the code file. The data label is encoded as binary so 0 = cat and 1 = dog. I have all *<model name>.summary()* for models commented out in the code. I also have commented out the filter visualization portion of the code as mine outputs one image per channel of per filter. These can be "turned on" by uncommenting them. Each model was compiled and trained using the format shown below:

```
model.compile(
    optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy',
tf.keras.metrics.TruePositives(), tf.keras.metrics.FalsePositives(),
tf.keras.metrics.TrueNegatives(), tf.keras.metrics.FalseNegatives()])

model.fit(training_set_normalized, epochs=5, batch_size=10,
          callbacks=[tf.keras.callbacks.EarlyStopping(patience=2)])
```

Epoch was set to 5 and batch size was set to 10 for the model used in 4(ii) and the epoch was set to 15 and batch size was set to 10 for 4 (iii) subnet model. I collected True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) values from the runs to compute the confusion matrix. Running the code runs predictions on untrained model and untrained subnet model and outputs accuracy and confusion matrix per model. After training the models, the code outputs loss, accuracy, and confusion matrix per model. For part 4(iii) sub-network, I grabbed the first 300 layers of the pretrained model with:

```
# Subnet pre_model:
c = 0
subnet = models.Sequential()
for c in range(300):
    layer = pre_model.get_layer(index=c)
    subnet.add(layer)
```

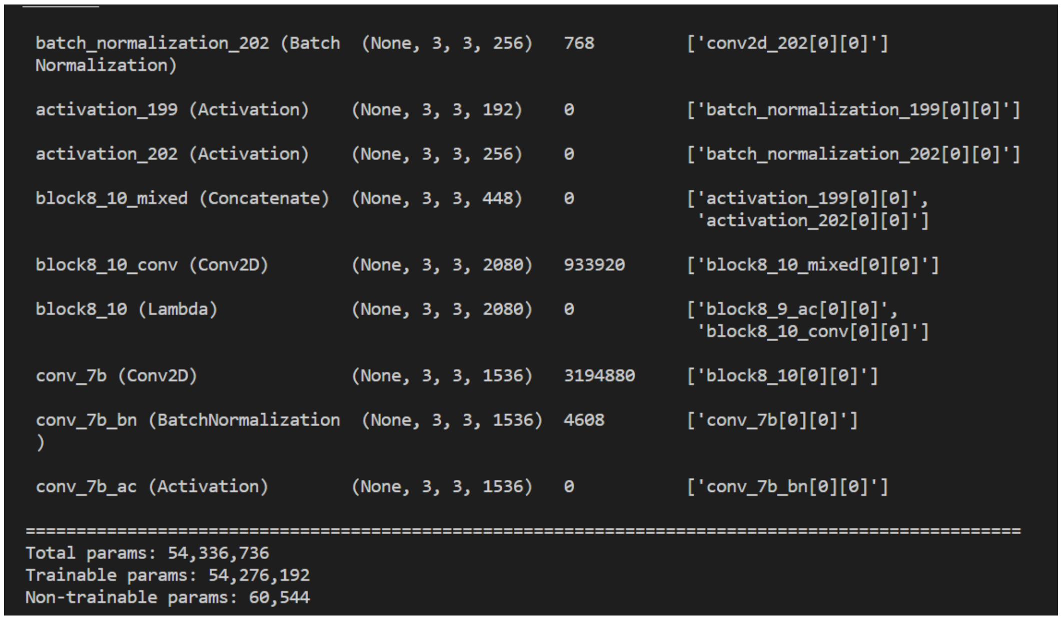
Cera Oh
CS510 CV and DL
Summer 2023
8/01/2023

The transfer heads were added following the instructions in the assignment sheet.

Results/Discussion

Step 1:

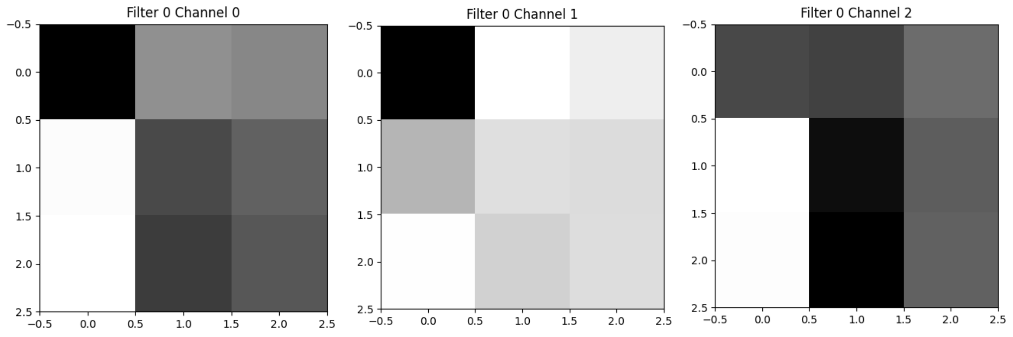
Running *pre_model.summary()* gives below output**:**



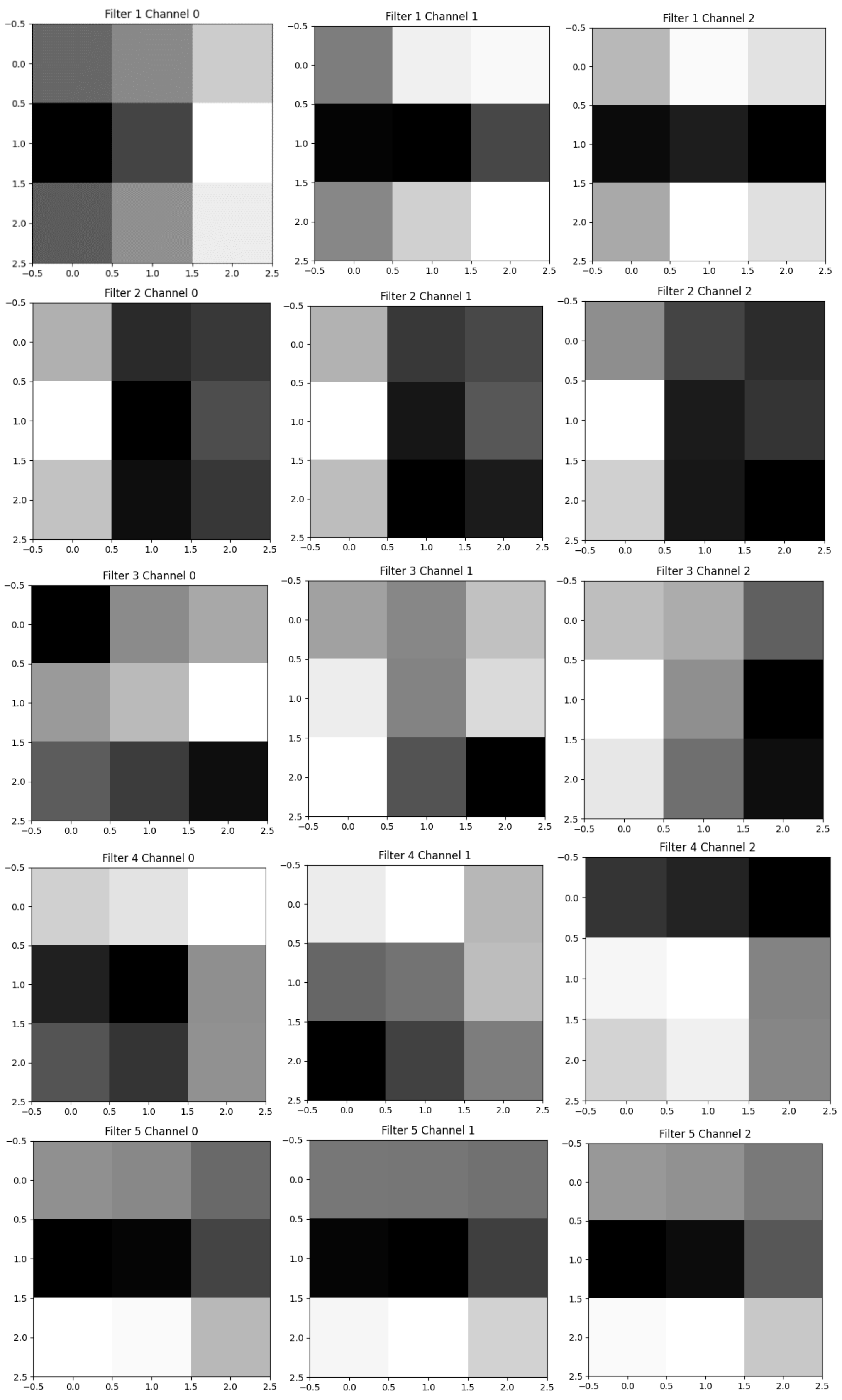
```
 batch_normalization_202 (Batch  (None, 3, 3, 256)   768        ['conv2d_202[0][0]']
 Normalization)

 activation_199 (Activation)    (None, 3, 3, 192)    0          ['batch_normalization_199[0][0]']

 activation_202 (Activation)    (None, 3, 3, 256)    0          ['batch_normalization_202[0][0]']

 block8_10_mixed (Concatenate)  (None, 3, 3, 448)    0          ['activation_199[0][0]',
                                                                 'activation_202[0][0]']

 block8_10_conv (Conv2D)        (None, 3, 3, 2080)   933920     ['block8_10_mixed[0][0]']

 block8_10 (Lambda)             (None, 3, 3, 2080)   0          ['block8_9_ac[0][0]',
                                                                 'block8_10_conv[0][0]']

 conv_7b (Conv2D)               (None, 3, 3, 1536)   3194880    ['block8_10[0][0]']

 conv_7b_bn (BatchNormalization  (None, 3, 3, 1536)  4608       ['conv_7b[0][0]']
 )

 conv_7b_ac (Activation)        (None, 3, 3, 1536)   0          ['conv_7b_bn[0][0]']

==================================================================================================
Total params: 54,336,736
Trainable params: 54,276,192
Non-trainable params: 60,544
```

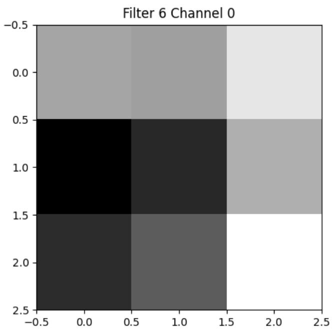
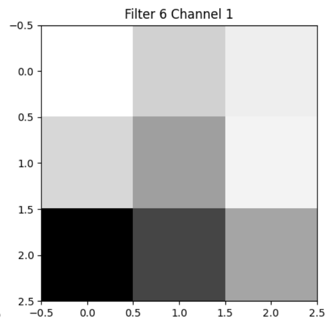
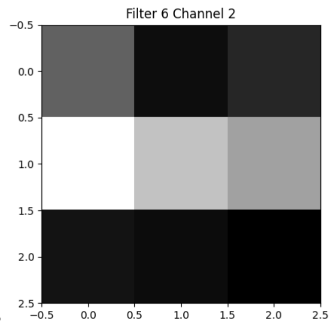
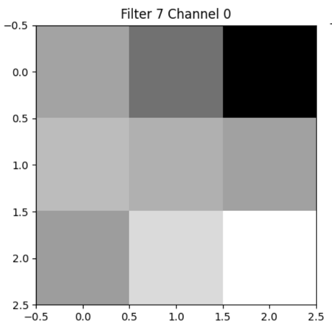
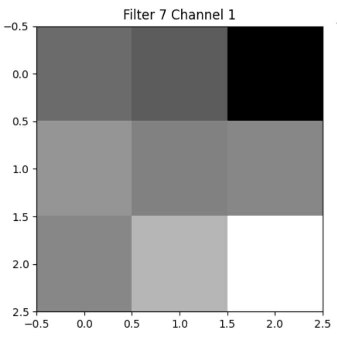
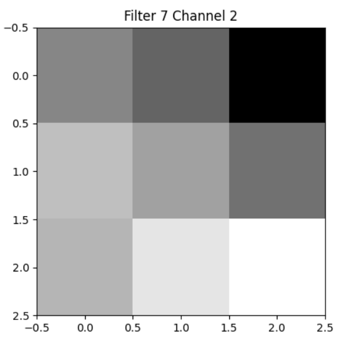
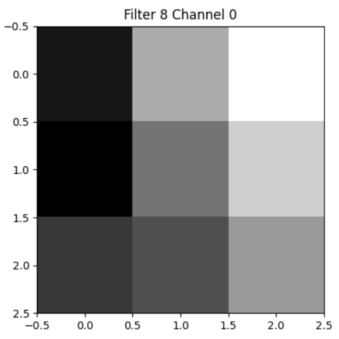
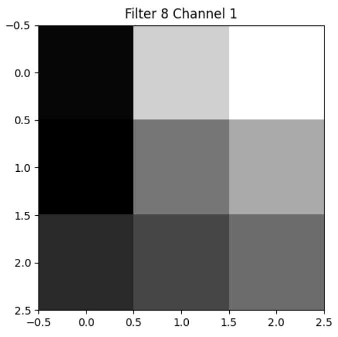
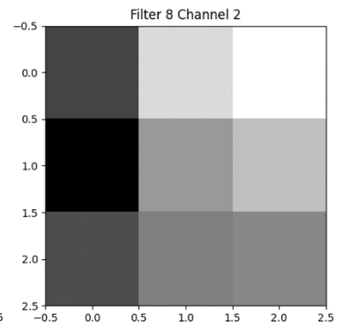
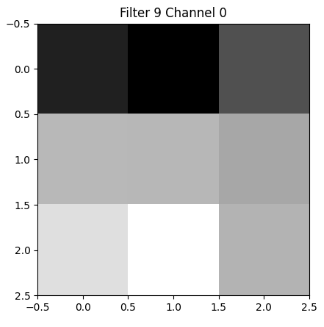
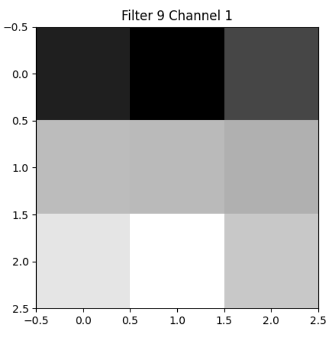
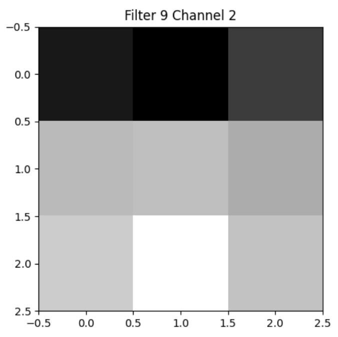
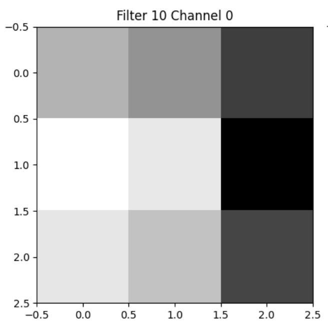
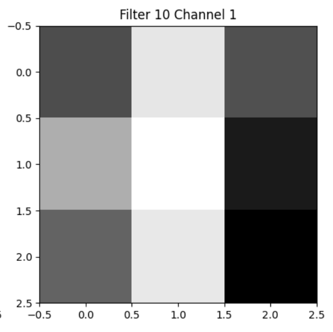
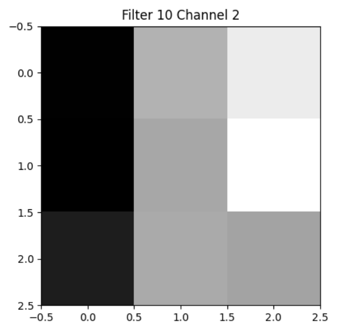
The number of trainable parameters were 54,276,192.

First convolutional layer filters were then visualized for each channel of each filter:

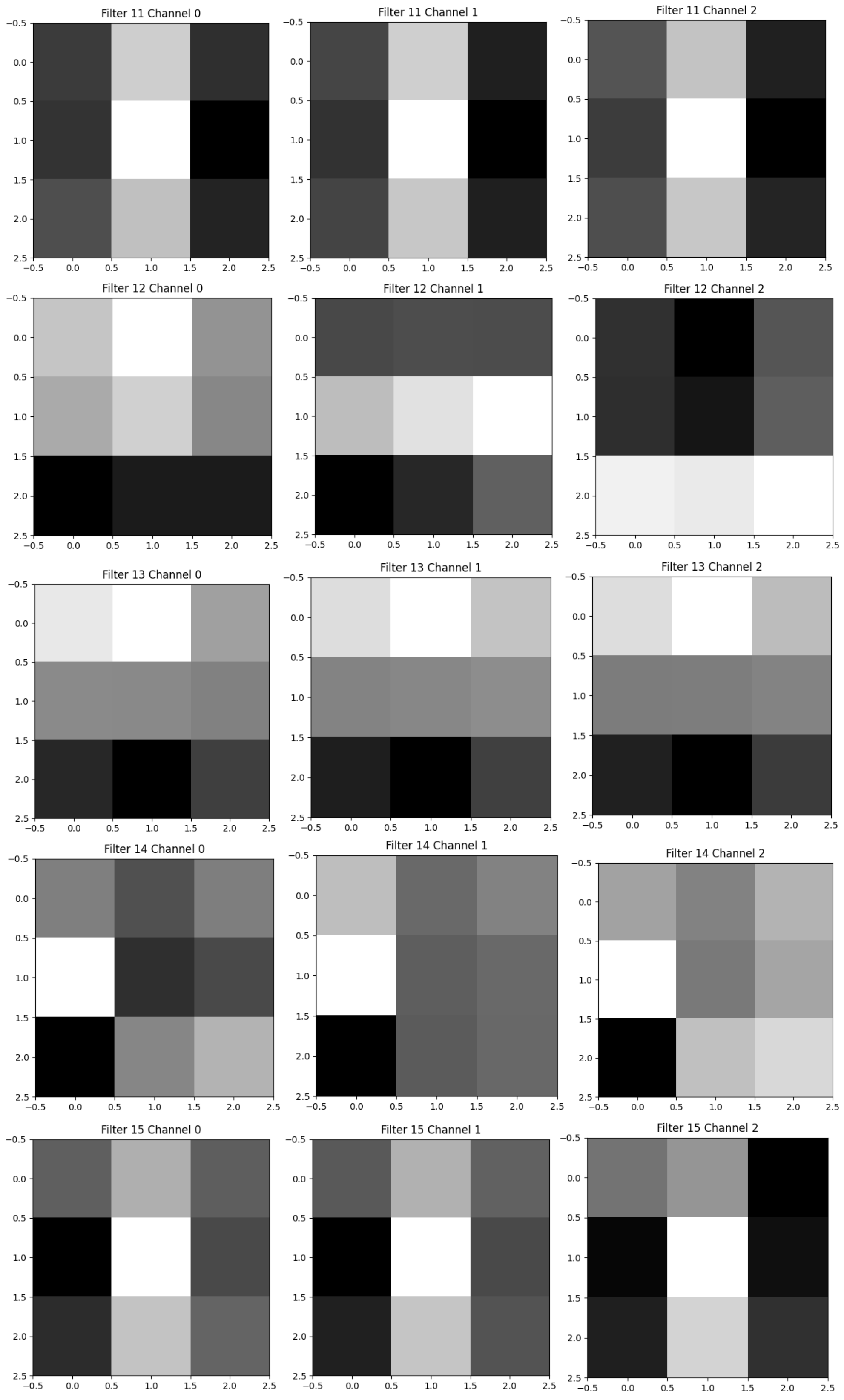


Cera Oh
CS510 CV and DL
Summer 2023
8/01/2023



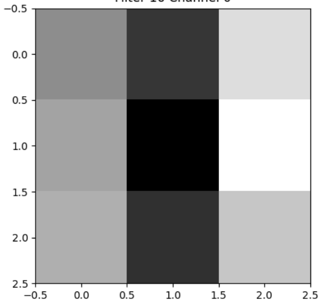

Filter 6 Channel 0

Filter 6 Channel 1

Filter 6 Channel 2

Filter 7 Channel 0

Filter 7 Channel 1

Filter 7 Channel 2

Filter 8 Channel 0

Filter 8 Channel 1

Filter 8 Channel 2

Filter 9 Channel 0

Filter 9 Channel 1

Filter 9 Channel 2

Filter 10 Channel 0

Filter 10 Channel 1

Filter 10 Channel 2

Cera Oh
CS510 CV and DL
Summer 2023
8/01/2023

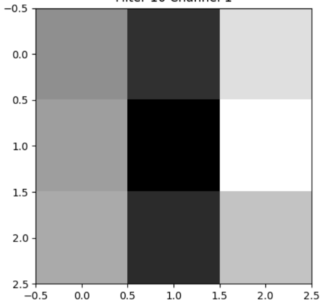


Cera Oh
CS510 CV and DL
Summer 2023
8/01/2023

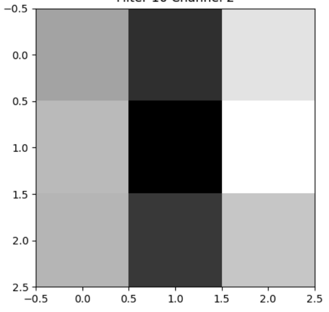
Filter 16 Channel 0



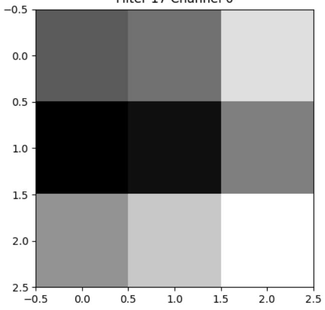
Filter 16 Channel 1



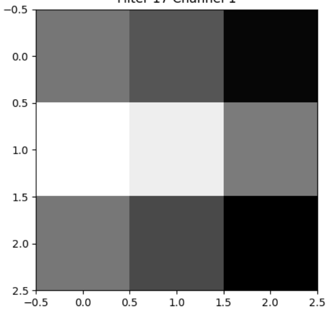
Filter 16 Channel 2



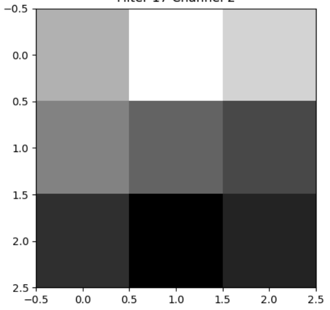
Filter 17 Channel 0



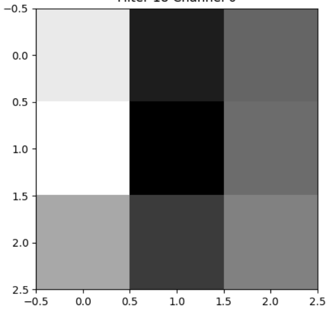
Filter 17 Channel 1



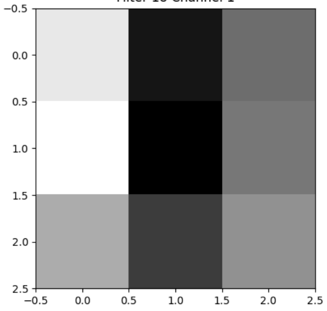
Filter 17 Channel 2



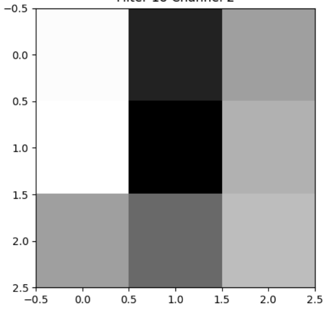
Filter 18 Channel 0



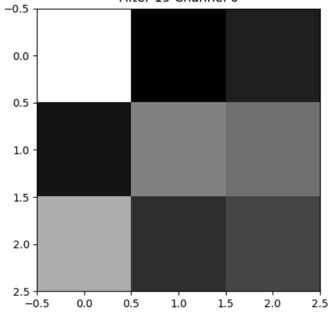
Filter 18 Channel 1



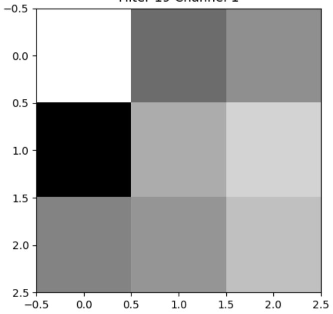
Filter 18 Channel 2



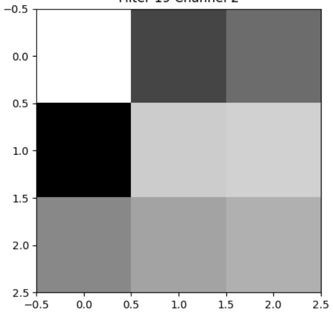
Filter 19 Channel 0



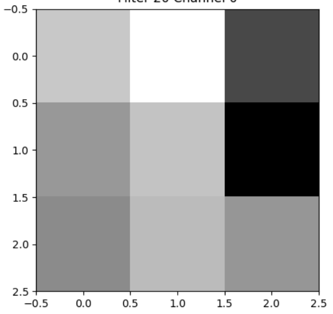
Filter 19 Channel 1



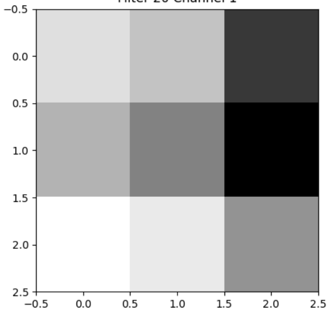
Filter 19 Channel 2



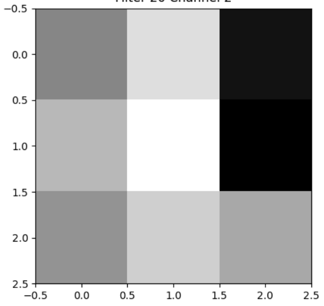
Filter 20 Channel 0



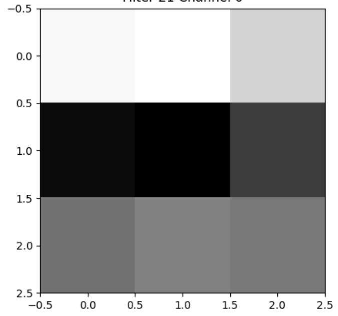
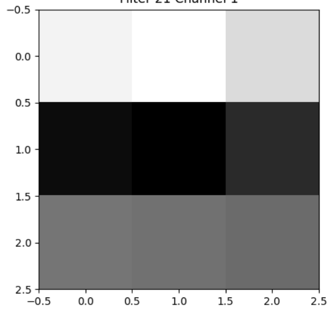
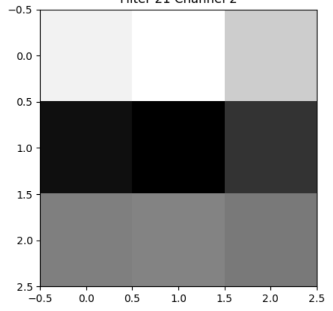
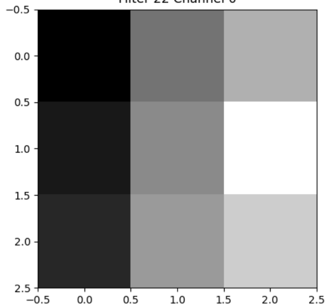
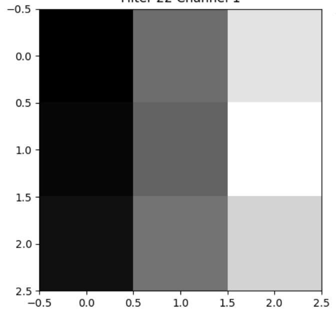
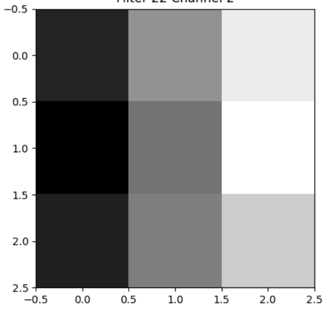
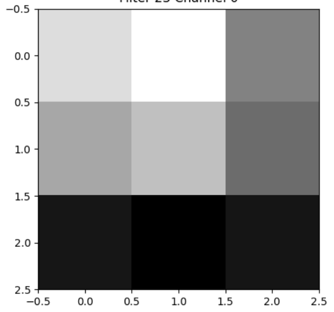
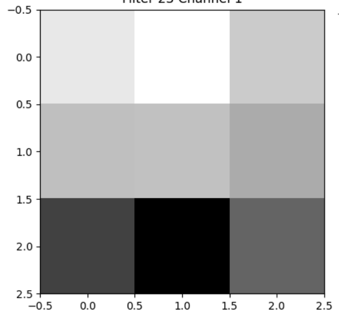
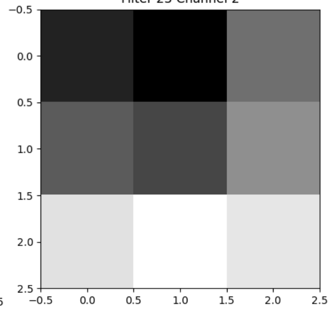
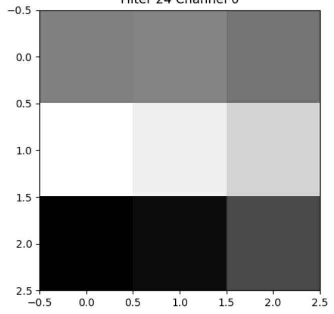
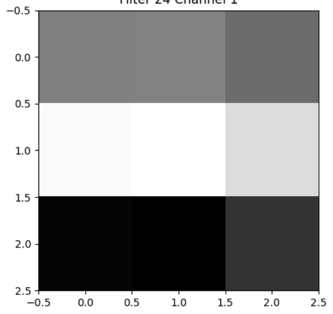
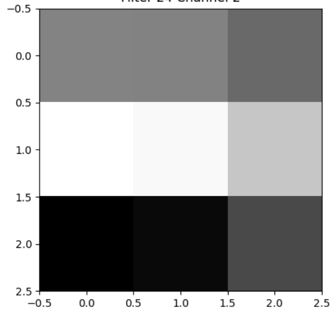
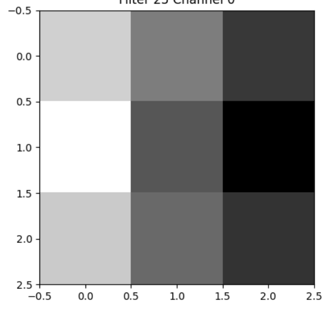
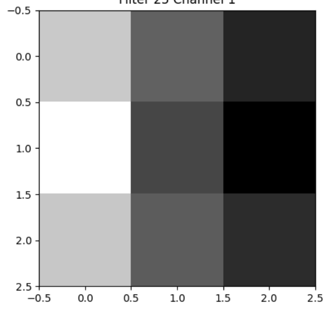
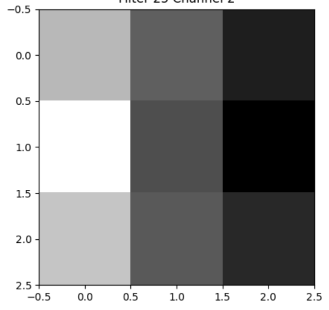
Filter 20 Channel 1



Filter 20 Channel 2

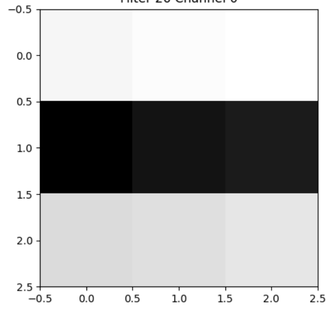


Cera Oh
CS510 CV and DL
Summer 2023
8/01/2023

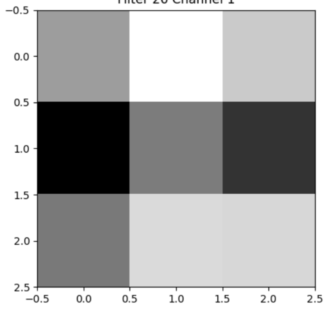
Filter 21 Channel 0

Filter 21 Channel 1

Filter 21 Channel 2

Filter 22 Channel 0

Filter 22 Channel 1

Filter 22 Channel 2

Filter 23 Channel 0

Filter 23 Channel 1

Filter 23 Channel 2

Filter 24 Channel 0

Filter 24 Channel 1

Filter 24 Channel 2

Filter 25 Channel 0

Filter 25 Channel 1

Filter 25 Channel 2


Cera Oh
CS510 CV and DL
Summer 2023
8/01/2023

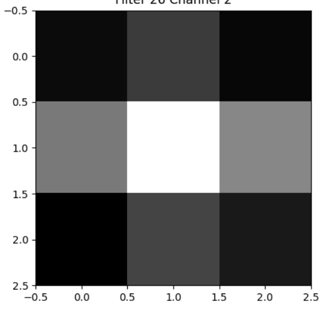
Filter 26 Channel 0



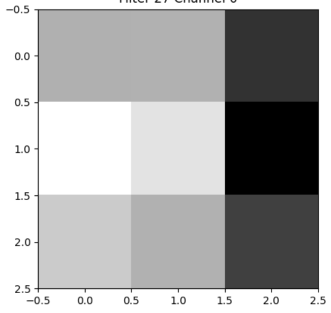
Filter 26 Channel 1



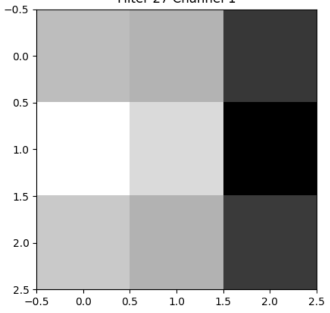
Filter 26 Channel 2



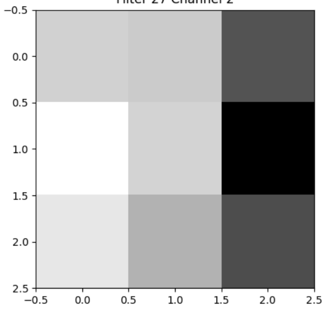
Filter 27 Channel 0



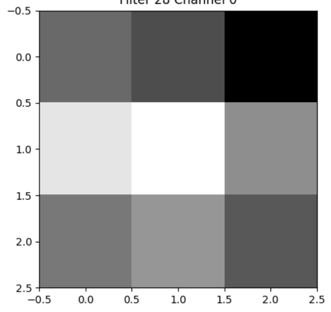
Filter 27 Channel 1



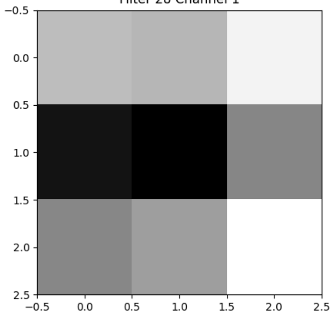
Filter 27 Channel 2



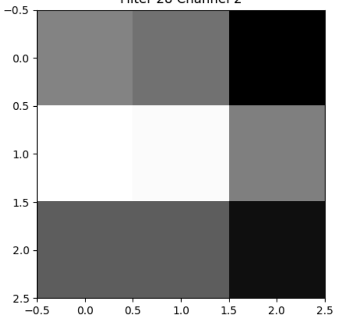
Filter 28 Channel 0



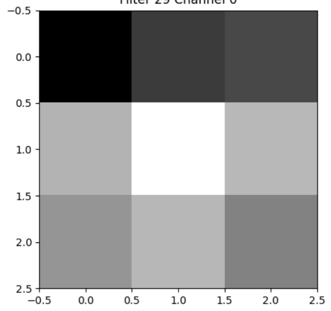
Filter 28 Channel 1



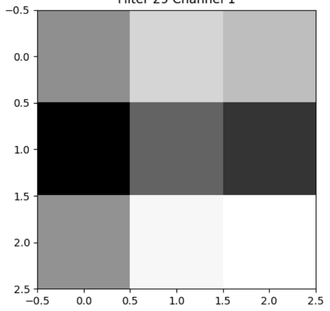
Filter 28 Channel 2



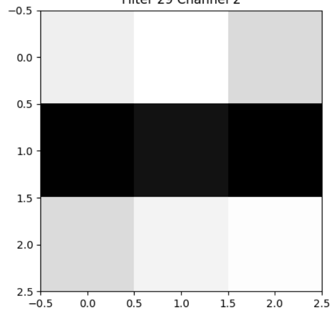
Filter 29 Channel 0



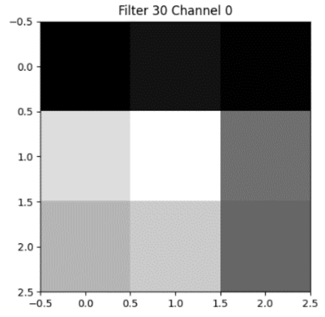
Filter 29 Channel 1



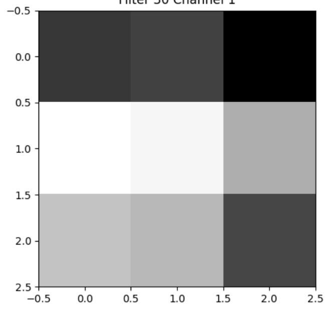
Filter 29 Channel 2



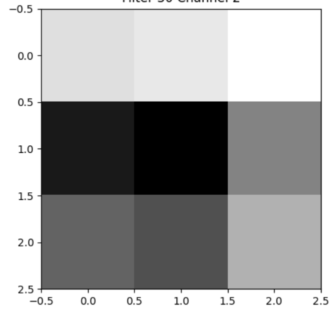
Filter 30 Channel 0

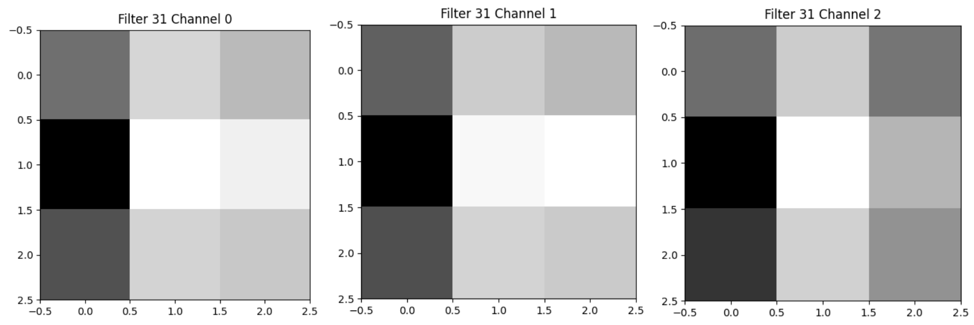


Filter 30 Channel 1



Filter 30 Channel 2

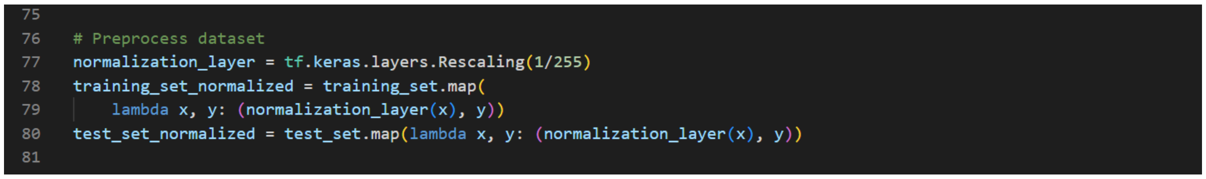




Each filter seems to focus on features (edge) in the image. According to the website Professor Rhodes linked me (https://machinelearningmastery.com/how-to-visualize-filters-and-feature-maps-in-convolutional-neural-networks/), dark square means inhibition and light square means excitation of weights for the filter. Some filters, like filter #24, seem to have all three channels uniformly activated in certain areas (middle three squares for #24). Others, like filter #26, have different activation for different channels (colors).

Step 2:

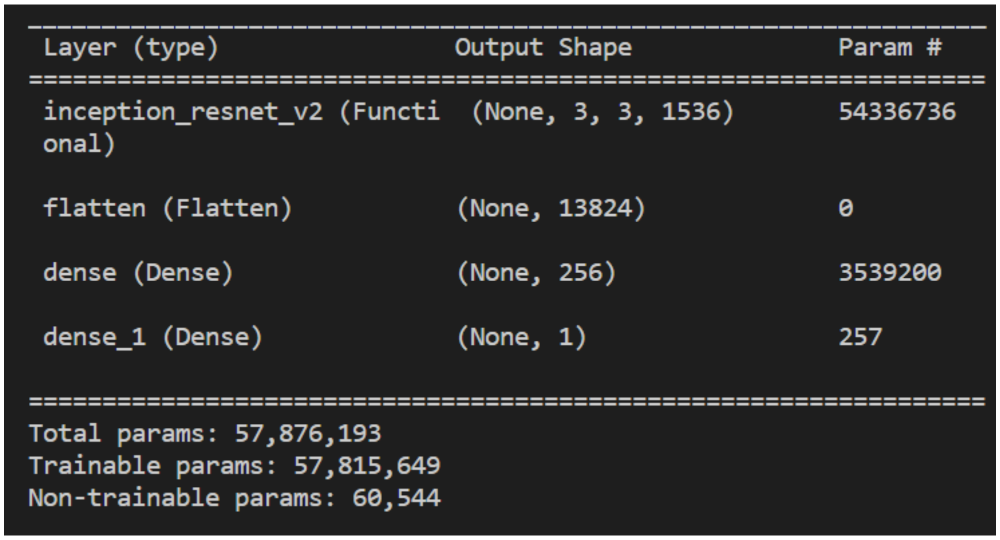
I used normalization for preprocessing data:



```

# Preprocess dataset
normalization_layer = tf.keras.layers.Rescaling(1/255)
training_set_normalized = training_set.map(
    lambda x, y: (normalization_layer(x), y))
test_set_normalized = test_set.map(lambda x, y: (normalization_layer(x), y))

```

The image was automatically resized to (150, 150, 3) using the *tf.keras.utils.image_dataset_from_directory()* call.

Step 3:
The model summary after adding the "transfer head" is shown below:



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 inception_resnet_v2 (Functi  (None, 3, 3, 1536)       54336736
 onal)

 flatten (Flatten)           (None, 13824)             0

 dense (Dense)               (None, 256)               3539200

 dense_1 (Dense)             (None, 1)                 257

=================================================================
Total params: 57,876,193
Trainable params: 57,815,649
Non-trainable params: 60,544
```

The number of total parameters was 57,876,193 (57,815,649 trainable and 60,544 untrainable).

Step 4:
(i) Transfer Model Evaluation

Confusion Matrix	
233	227
767	773

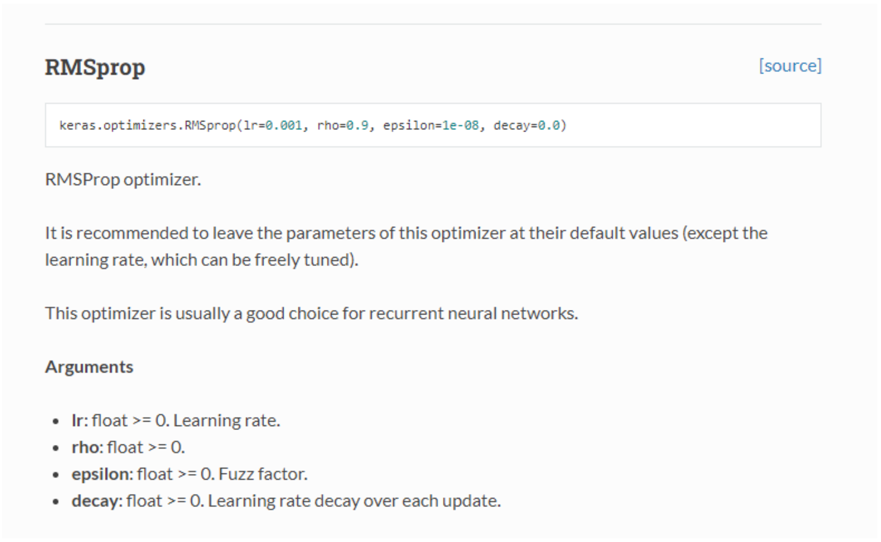
The model had an overall test accuracy of 50.03%.

(ii) Transfer Model Training

I used binary cross entropy loss by setting *loss* as '*binary_crossentropy*' and I used RMSprop as the optimizer by setting *optimizer* to *'rmsprop'* for the *compile()* call.

```
model.compile(
    optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy',
tf.keras.metrics.TruePositives(), tf.keras.metrics.FalsePositives(),
tf.keras.metrics.TrueNegatives(), tf.keras.metrics.FalseNegatives()])

model.fit(training_set_normalized, epochs=5, batch_size=10,
          callbacks=[tf.keras.callbacks.EarlyStopping(patience=2)])
```

The RMSprop has parameters learning rate, rho, epsilon, and decay. It was recommended by the Keras reference (https://faroit.com/keras-docs/2.0.8/optimizers/) to leave the parameters as default values, which were 0.001 for the learning rate, 0.9 for rho, 1e-08 for epsilon, and 0.0 for decay.


RMSprop [source]

```
keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
```

RMSProp optimizer.

It is recommended to leave the parameters of this optimizer at their default values (except the learning rate, which can be freely tuned).

This optimizer is usually a good choice for recurrent neural networks.

Arguments

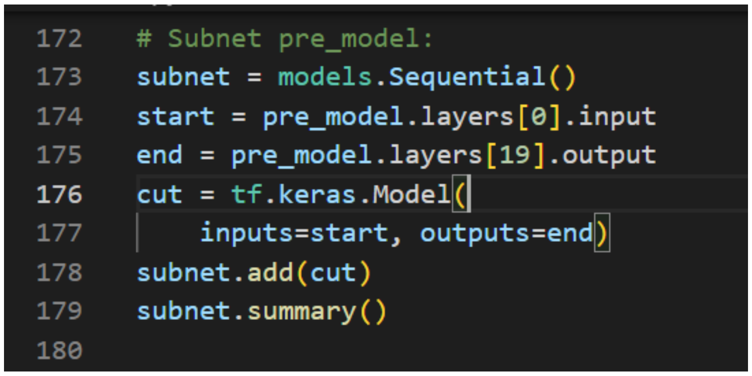
- **lr**: float >= 0. Learning rate.
- **rho**: float >= 0.
- **epsilon**: float >= 0. Fuzz factor.
- **decay**: float >= 0. Learning rate decay over each update.

The model was trained for 5 epochs and achieved 0.1050 loss and 98.37 % accuracy during training. When tested on the test set, it achieved 0.31637120246887207 per-epoch test loss and 97.1% overall test accuracy.

Confusion Matrix	
993	51
7	949

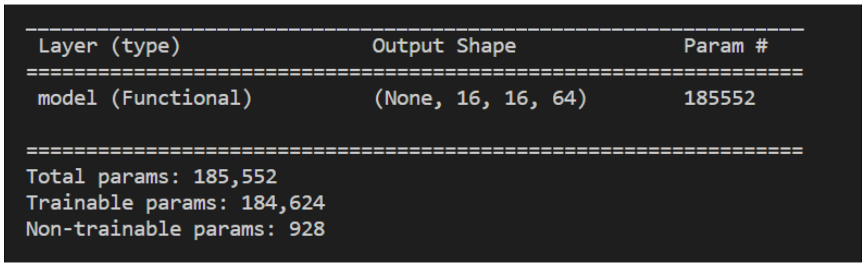
(iii) Subnet Model Evaluation

I unfroze the weights of the pretrained network and took the first 20 layers from the pretrained network to build a sub-network.



```
# Subnet pre_model:
subnet = models.Sequential()
start = pre_model.layers[0].input
end = pre_model.layers[19].output
cut = tf.keras.Model(
    inputs=start, outputs=end)
subnet.add(cut)
subnet.summary()
```

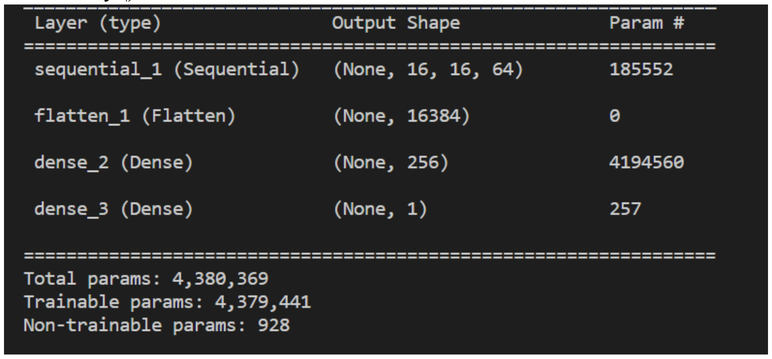
The output from the *summary()* call is below:



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 model (Functional)          (None, 16, 16, 64)        185552

=================================================================
Total params: 185,552
Trainable params: 184,624
Non-trainable params: 928
```

It had 185,552 total parameters (184,624 trainable and 928 un-trainable).

I then did the same as before to add the transfer head to the model. The output from the *summary()* call is below:



```
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_1 (Sequential)   (None, 16, 16, 64)        185552

 flatten_1 (Flatten)         (None, 16384)             0

 dense_2 (Dense)             (None, 256)               4194560

 dense_3 (Dense)             (None, 1)                 257

=================================================================
Total params: 4,380,369
Trainable params: 4,379,441
Non-trainable params: 928
```

It had 4,380,369 total parameters (4,379,441 trainable and 928 un-trainable). I then froze the weights of the subnetwork and pretrained network. Running evaluation on the network gave 49.75% overall test accuracy.

Confusion Matrix	
719	724
281	276

Then I trained the model for 15 epochs with batch size 10. At the end of the training, it achieved 0.0342 epoch loss and 99.26% accuracy.

The per-epoch test loss was 3.1100873947143555 and the overall test accuracy was 80.55%.

Confusion Matrix	
816	205
284	795

The pre-trained model by itself without training only achieved 50.03% accuracy, and it correctly categorized 233 dog images and 773 cat images, but miscategorized 227 cat images as dogs and 767 dog images as cats. It seems to think the images are more likely to contain cats. After training, the model performed much better, correctly categorizing 993 dog images and 949 cat images, and wrongly predicted 7 images of dogs as cats and 51 images of cats as dogs. Taking a subnetwork of the pretrained model decreased accuracy for both the untrained subnetwork and trained subnetwork with transfer head, but for subnet of pre-trained model, the accuracies were not far off from that of pre-trained network (50.03% vs 49.75%). The subnetwork without training only achieved 49.75% accuracy, and it correctly categorized 719 dog images and 276 cat images, but miscategorized 724 cat images as dogs and 281 dog images as cats. It seems to predict the images are more likely a dog. After training, the model performed much better, getting 80.55% test accuracy, and correctly categorizing 816 dog images and 795 cat images and wrongly predicted 205 images of dogs as cats and 284 images of cats as dogs.

Conclusion

Each filter in the first layer looks for different features for activation. The pre-trained model by itself without training has lower accuracy than fully trained model with transfer head. Taking subnetwork of pre-trained model lowered accuracy for both untrained model and trained model. The subnetwork took more epochs to train to similar training accuracy but still got lower accuracy on the test dataset. In both cases, trained models performed better than untrained ones.

References:
https://machinelearningmastery.com/how-to-visualize-filters-and-feature-maps-in-convolutional-neural-networks/
https://www.reddit.com/r/learnmachinelearning/comments/zi3zqb/how_to_remove_layers_of_keras_functional_model/
https://faroit.com/keras-docs/2.0.8/optimizers/
https://keras.io/api/