


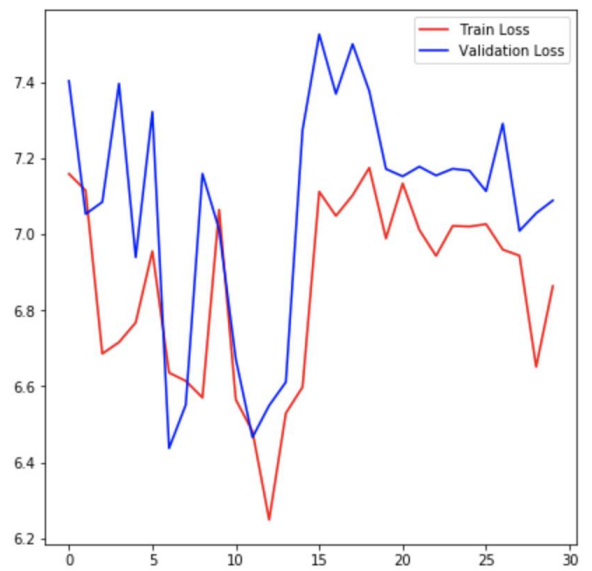
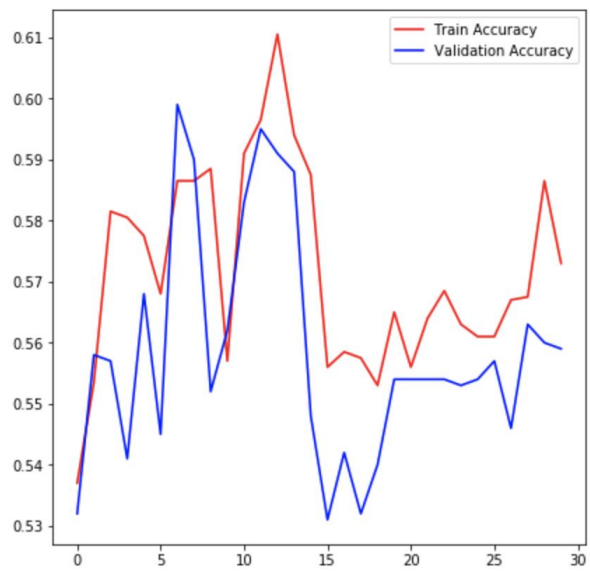
## Binary classification:

- (a) Download the Kaggle's Cats and Dogs  dataset and select a subset of 2000 dogs and 2000 cat images to be used for training, validation, and testing. The use of subsets is intended to simulate the condition of limited data.
- (b) Define a training, validation, and testing data generators.
- (c) Build a convolutional neural network using several convolution, pooling, and normalization layers, followed by one or more dense layers. Flatten the data between the convolution and dense layers.
- (d) Evaluate performance and tune hyperparameters as needed.
- (e) Visualize the activation of some of the convolution layers and draw a conclusion.
- (f) Visualize the filters learned in training by finding the input that will maximize their response and draw a conclusion.
- (g) Replace your convolution layers with the pre-trained convolution base of VGG16. Train with the convolution base frozen and evaluate the results.

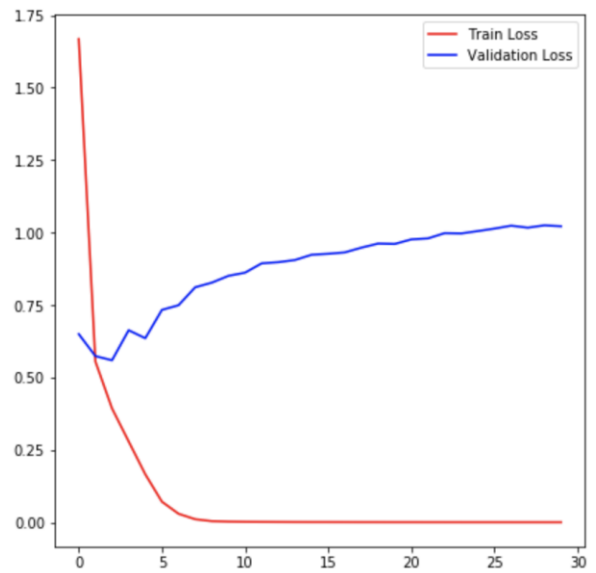
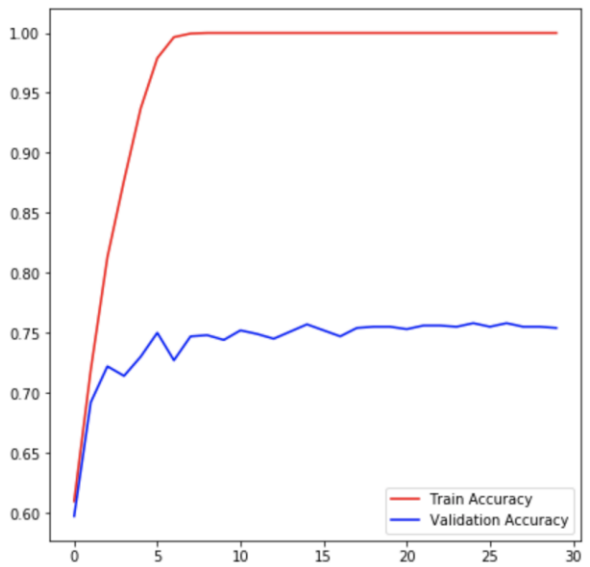
### 1. Hyperparameter tuning:

#### a. 1 Conv Layer with Max pooling

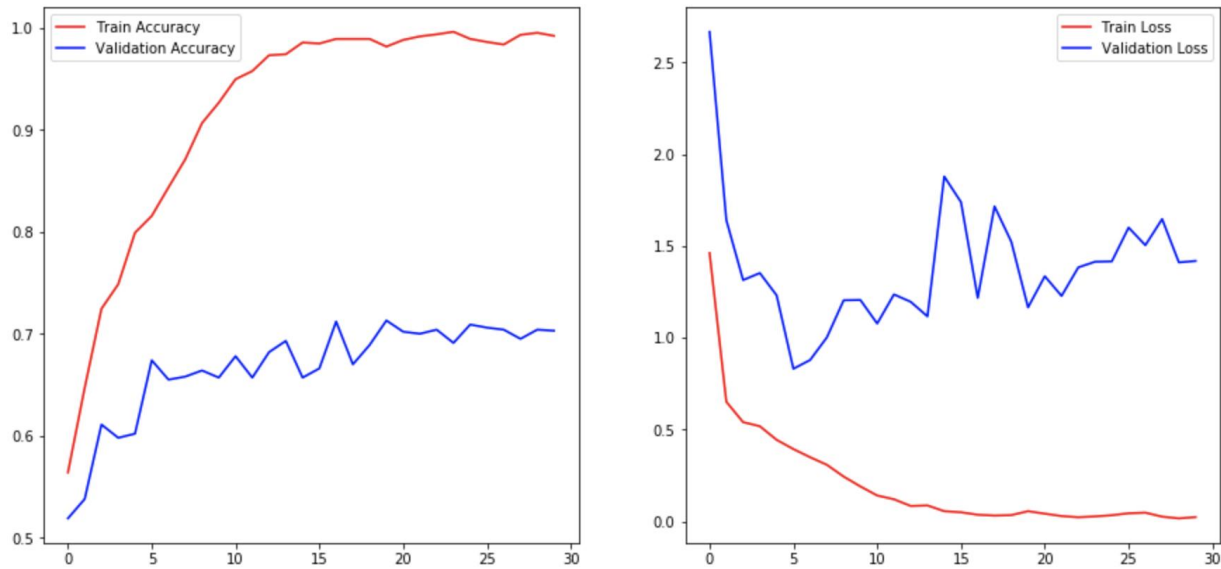
Layer (type)	Output Shape	Param #
input_3 (InputLayer)	(None, 150, 150, 3)	0
conv2d_3 (Conv2D)	(None, 150, 150, 16)	448
leaky_re_lu_3 (LeakyReLU)	(None, 150, 150, 16)	0
max_pooling2d_3 (MaxPooling2D)	(None, 75, 75, 16)	0
batch_normalization_3 (Batch Normalization)	(None, 75, 75, 16)	64
flatten_3 (Flatten)	(None, 90000)	0
dense_5 (Dense)	(None, 512)	46080512
dense_6 (Dense)	(None, 1)	513
Total params: 46,081,537		
Trainable params: 46,081,505		
Non-trainable params: 32		



## b. 4 Conv2D Layers



c. Adding Dropout 0.2



As expected with a small dataset, adding dropout has not improved the performance, although it does reduce overfitting a small extent, which is useful.

d. Adding more dense layers did not improve the performance. Final Model architecture:

**4 convolution layer:**

```
hid = Conv2D(16, kernel_size=3, strides=1, input_shape=self.img_shape, padding = 'same')(img)
hid = LeakyReLU(alpha=0.2)(hid)
hid = MaxPooling2D((2,2))(hid)
hid = Dropout(0.2)(hid)
hid = BatchNormalization(momentum = 0.8)(hid)
```

**Flatter Layer:**

```
hid = Flatten()(hid)
```

**Dence Layer:**

```
hid = Dense(512, activation="relu")(hid)
```

## Model Summary:

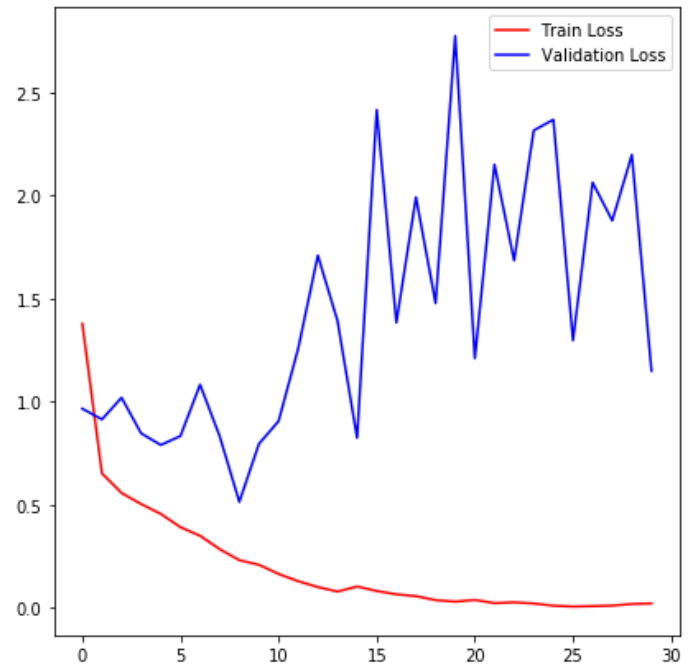
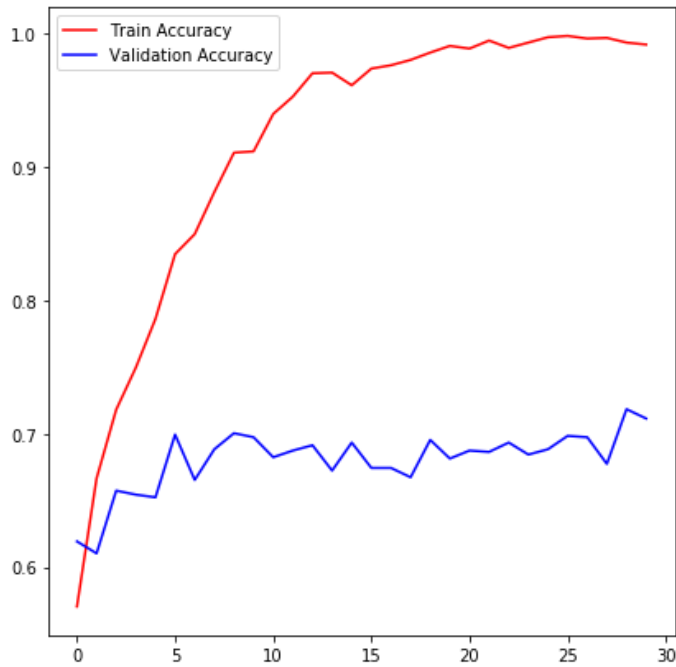
Model: "model\_3"

Layer (type)	Output Shape	Param #
=====		
input_3 (InputLayer)	(None, 150, 150, 3)	0
conv2d_9 (Conv2D)	(None, 150, 150, 16)	448
leaky_re_lu_9 (LeakyReLU)	(None, 150, 150, 16)	0
max_pooling2d_9 (MaxPooling2	(None, 75, 75, 16)	0
dropout_9 (Dropout)	(None, 75, 75, 16)	0
batch_normalization_9 (Batch	(None, 75, 75, 16)	64
conv2d_10 (Conv2D)	(None, 75, 75, 32)	4640
leaky_re_lu_10 (LeakyReLU)	(None, 75, 75, 32)	0
max_pooling2d_10 (MaxPooling	(None, 37, 37, 32)	0
dropout_10 (Dropout)	(None, 37, 37, 32)	0
batch_normalization_10 (Batc	(None, 37, 37, 32)	128
conv2d_11 (Conv2D)	(None, 37, 37, 64)	18496
leaky_re_lu_11 (LeakyReLU)	(None, 37, 37, 64)	0
max_pooling2d_11 (MaxPooling	(None, 18, 18, 64)	0
dropout_11 (Dropout)	(None, 18, 18, 64)	0
batch_normalization_11 (Batc	(None, 18, 18, 64)	256
conv2d_12 (Conv2D)	(None, 18, 18, 128)	73856
leaky_re_lu_12 (LeakyReLU)	(None, 18, 18, 128)	0
max_pooling2d_12 (MaxPooling	(None, 9, 9, 128)	0
dropout_12 (Dropout)	(None, 9, 9, 128)	0
batch_normalization_12 (Batc	(None, 9, 9, 128)	512
flatten_3 (Flatten)	(None, 10368)	0
dense_5 (Dense)	(None, 512)	5308928
dense_6 (Dense)	(None, 1)	513
=====		
Total params: 5,407,841		
Trainable params: 5,407,361		
Non-trainable params: 480		

70 % Accuracy:

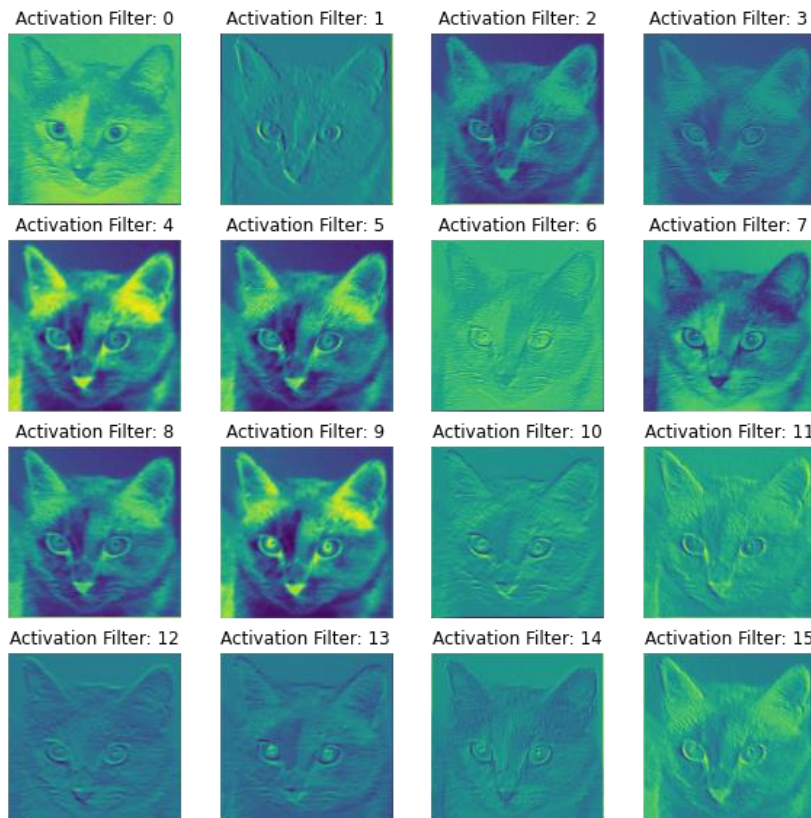
```
mod.model.evaluate_generator(mod.test_generator, steps=40)
```

[1.4481793642044067, 0.6980000138282776]

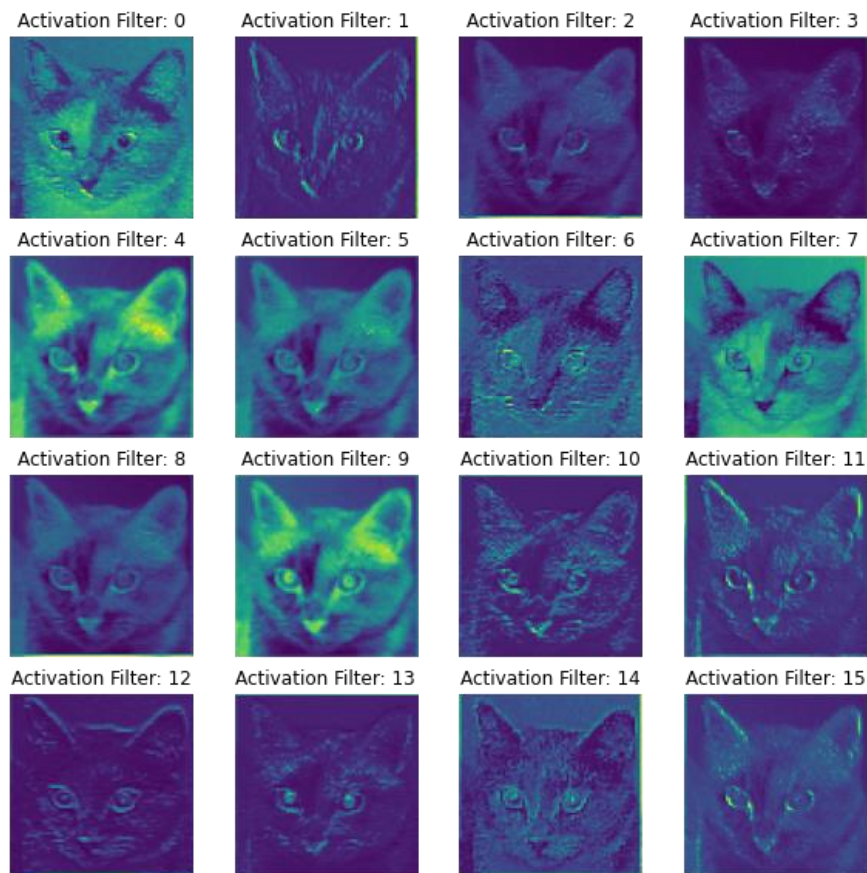


## 2. Activation Outputs:

a. First Layer Activation – Before Batch Normalization and Dropout layers:



b. Activations after Dropout layer.



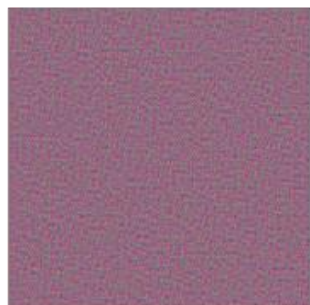
- Here, we can see the patterns that are learnt by the different filters in the convolution block. For example, filter 0 seems to highlight the edges in the image, and filter 13 is activated by the eyes of the cat.



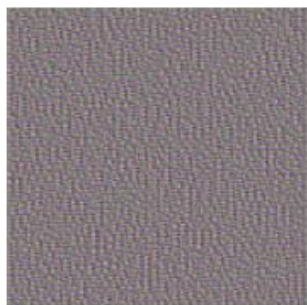
### 3. Visualizing filters:

#### 1. Filters from the first convolution block

Filter: 0



Filter: 1



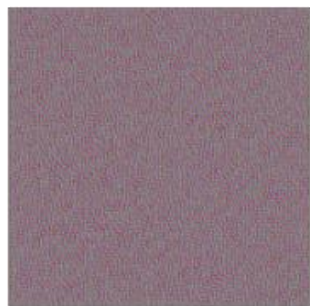
Filter: 2



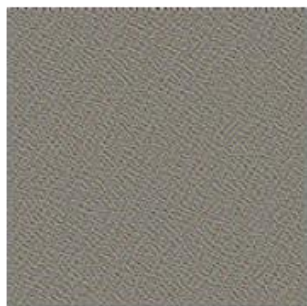
Filter: 3



Filter: 4



Filter: 5



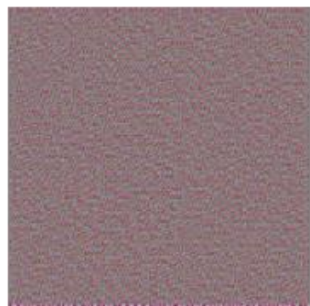
Filter: 6



Filter: 7



Filter: 8



Filter: 9



Filter: 10



Filter: 11



Filter: 12



Filter: 13



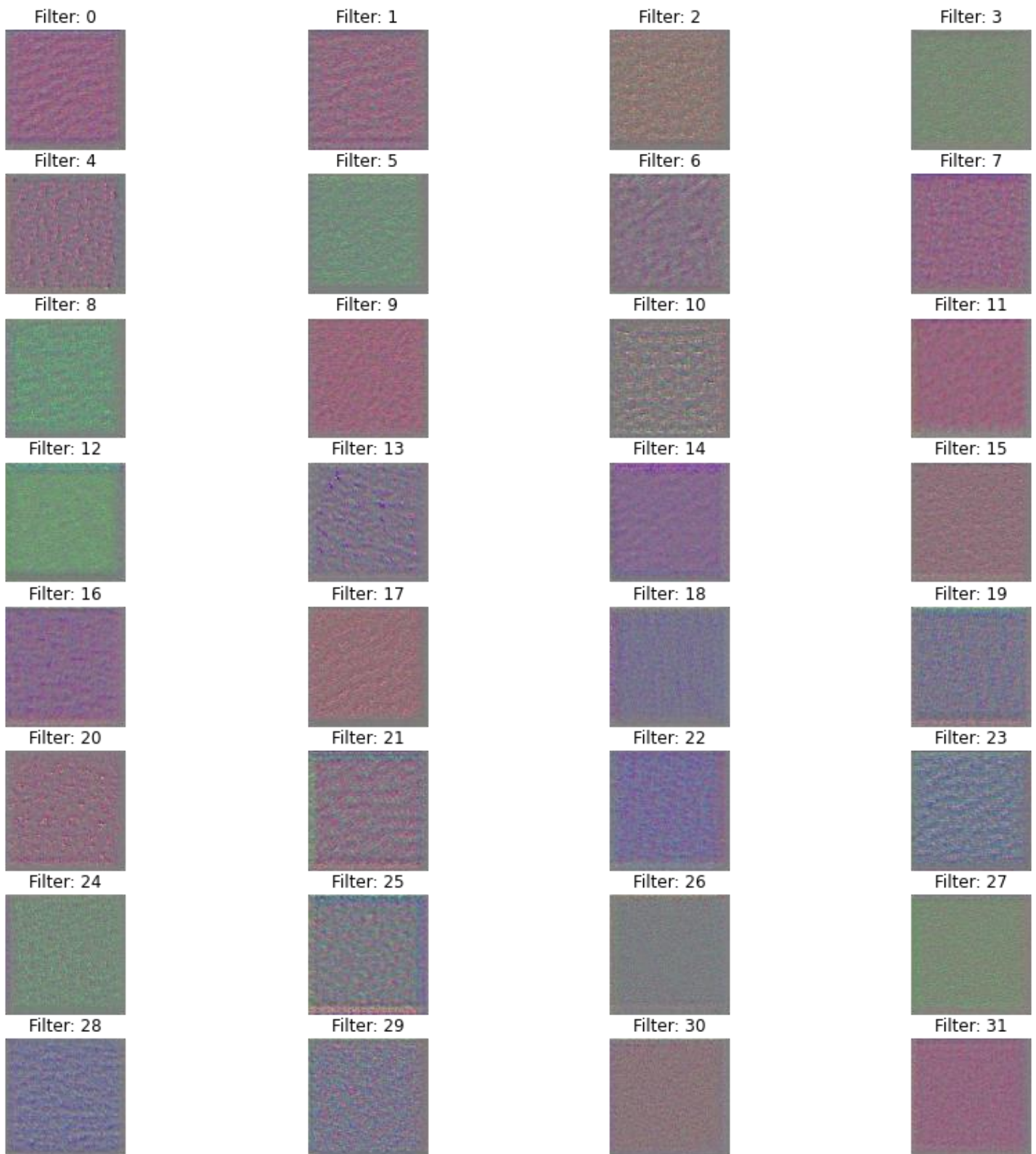
Filter: 14



Filter: 15



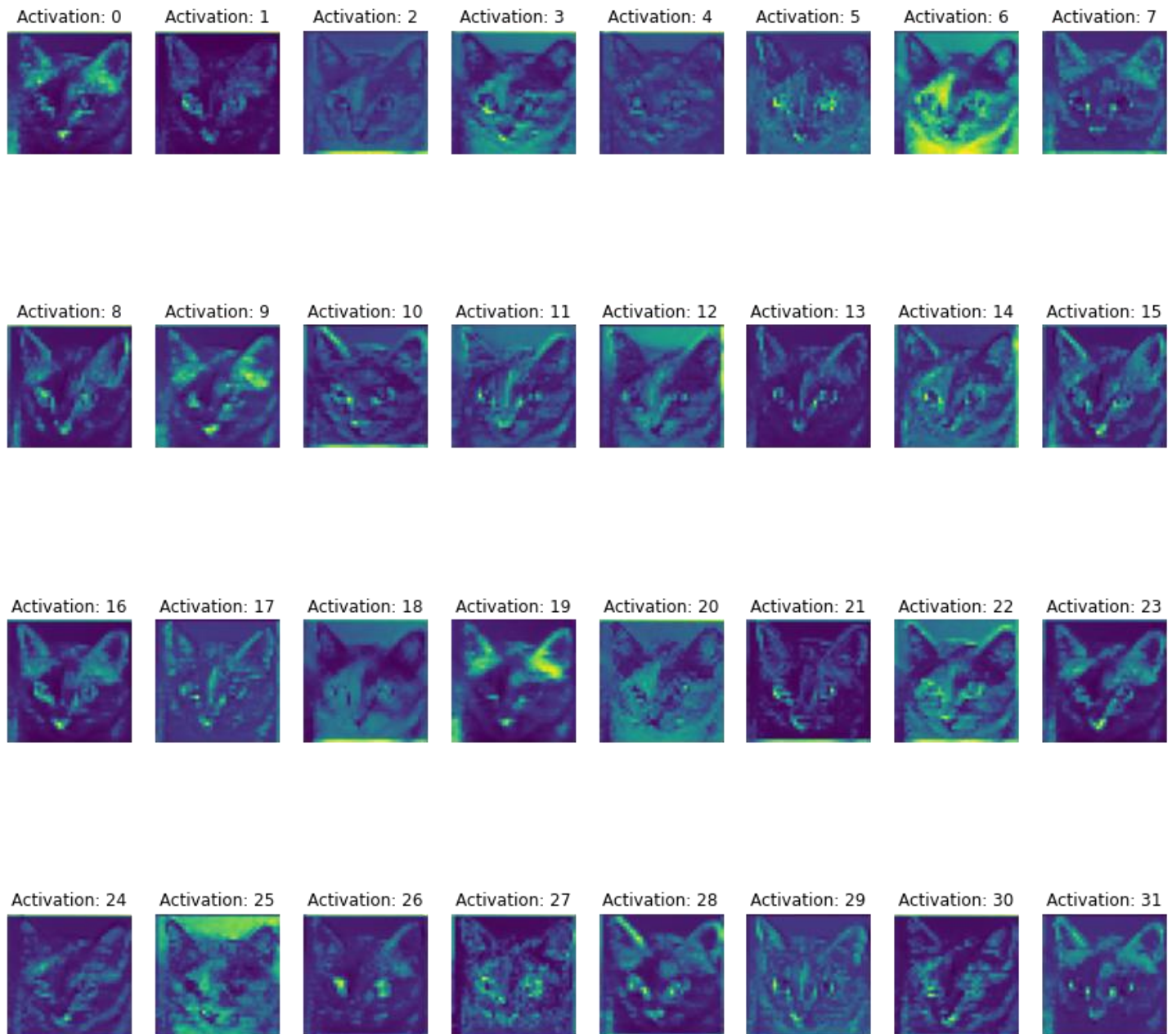
## 2. Filters from the Second convolution block



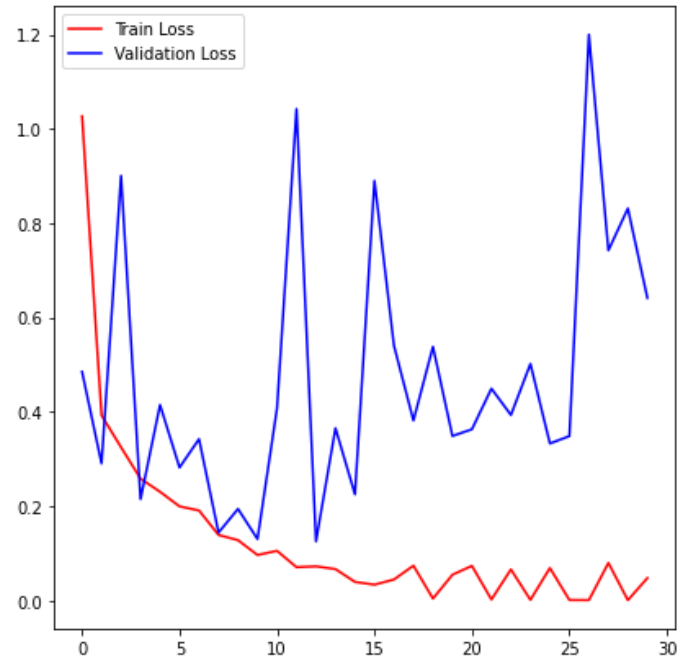
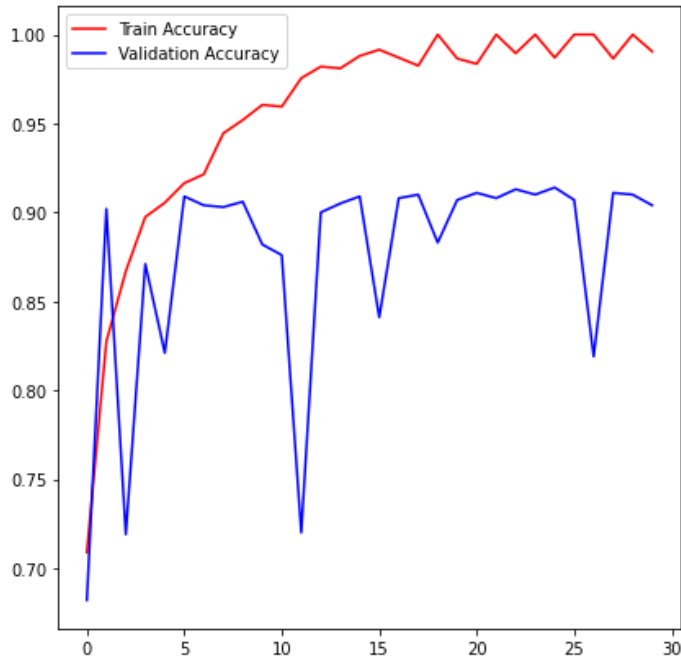
- The filters in the first convolution layer learn low level patterns such as horizontal, diagonal and vertical lines. As we go deeper into the layer, the model learns higher level features. The filter activations for the second convolution block are activated by more specific textures than the first layer filters.



c. Second Convolution Block:

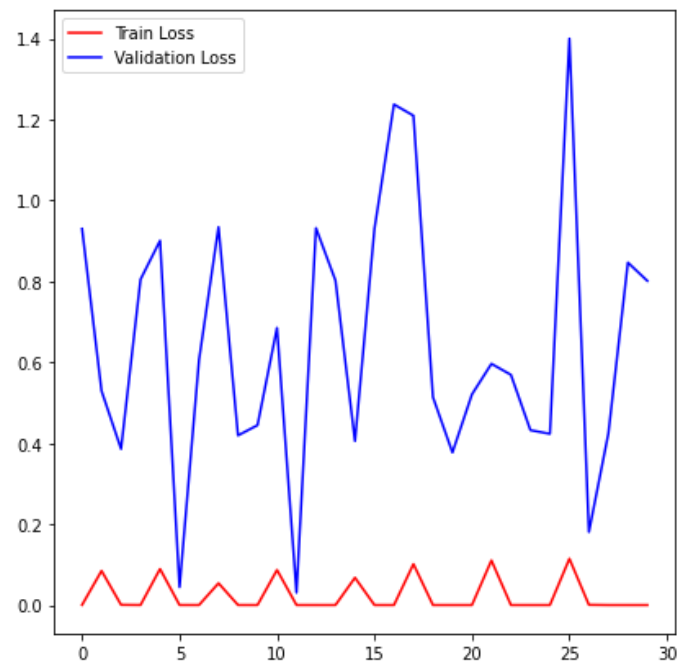
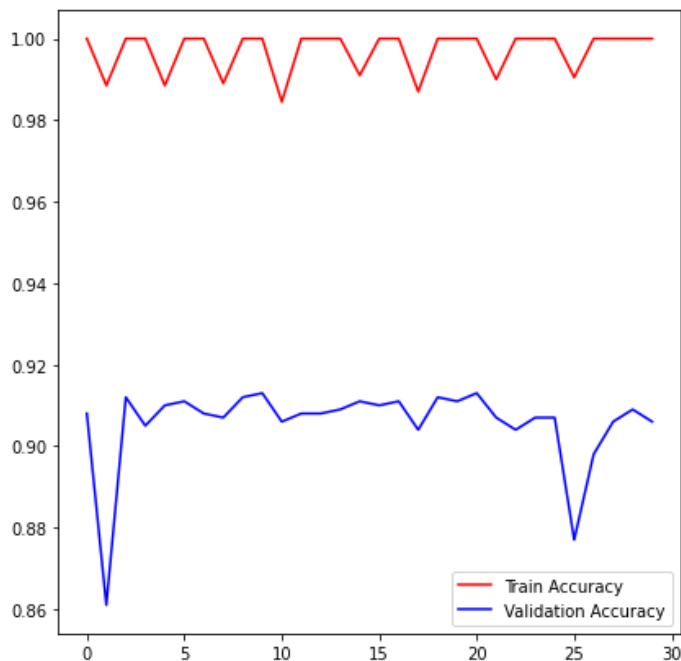


#### 4. Pre-trained VGG Model with a Frozen base



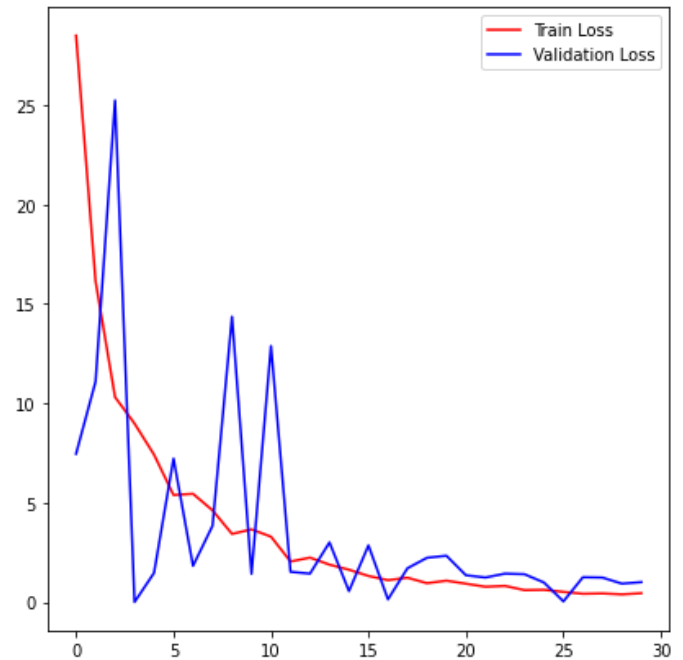
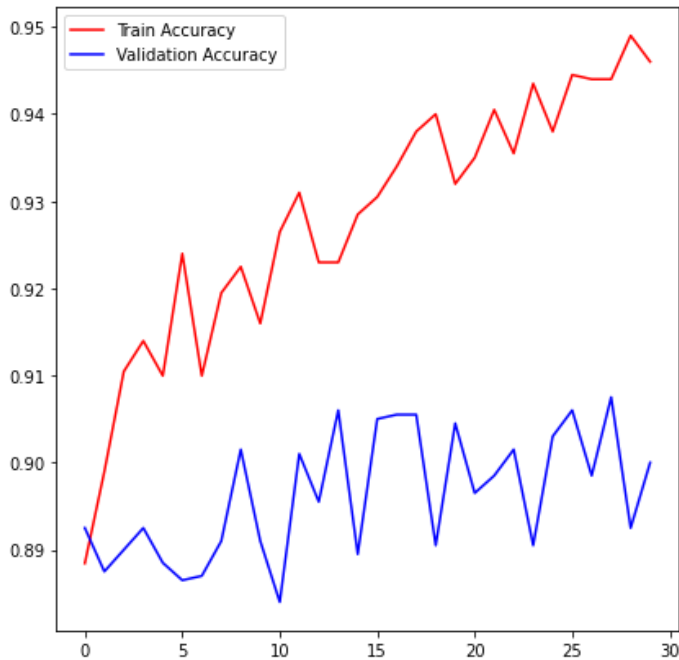
- As can be seen, the pre-trained model immediately improves the performance of the model from the 70% that I was getting using my own model.
- Evaluation on Test set: [0.642599880695343, 0.8809999823570251]
- The VGG model gives an accuracy of 88%.

#### 5. Fine tuning the model by Un-freezing the base.



- After fine-tuning, performance has improved slightly. It can be seen that the initial epochs have a high accuracy since the weights are inherited from the previous frozen base model.
- After fine tuning, the model gives an accuracy of 88.80% on the test set.

## 6. Data Augmentation



- Data Augmentation has helped reduce the overfitting drastically.
- However, the accuracy of the model on the test set has gone up to **89.15%** on the test set.
- [0.5403764247894287, 0.8914999961853027]