AML – Assignment 3 – Thanasit
Convolution – Neural Network – Cats and Dogs Dataset

1. Summary

I created six models with different configurations and datasets. I found that training size is a crucial factor for achieving good performance. However, pre-trained model is less sensitive by traning size. There are several ways to prevent model overfitting, such as data augmentation, dropout, and regularization. A higher number of layers and nodes results in a more complex model, which works well with larger training datasets. A pre-trained model, VGG16, demonstrated impressive performance within only a few iterations. My own model, the Improved Model, also achieved strong performance compared to VGG16. I can further improve my model by increasing the number of layers, adding regularization, and adjusting learning rates to enhance its performance.

As shown in the table below, increasing the training set size from 1,000 to 2,000 observations results in a significant 10% accuracy improvement. Adding more layers and nodes further boosts performance by an impressive 7%. With adjustments to my custom model and a training size of 12,000 observations, I achieved an accuracy of 91.15%. While VGG16, with its refined tuning and pre-trained features, outperforms my model by 5%, I'm very proud of my model's results. Interestingly, the pre-trained model performs remarkably well even with a smaller dataset: with just 2,000 observations, VGG16 reaches an accuracy of 93.2%. Given a much larger training dataset, VGG16's accuracy climbs to an outstanding 96.2%.

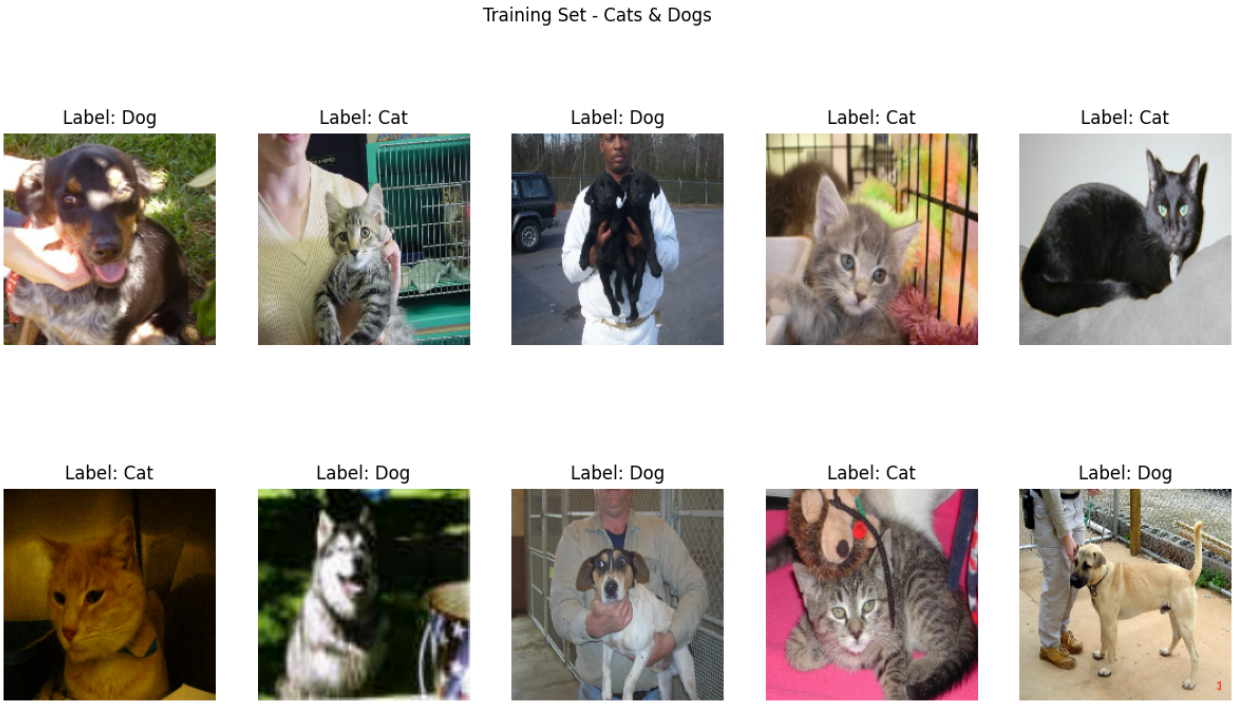
Model Comparison Table

Model	Base Model	Adjusted Base Model	Improved Adjusted Base Model	Improved Model	VGG16 Model: small	VGG16 Model: large
Dataset	Super Small	Small	Small	Large	Small	Large
Training Size	1000	2000	2000	12000	2000	12000
Validation Size	500	500	500	4000	500	4000
Testing Size	500	500	500	4000	500	4000
Data Augmentation Process	No	No	Yes	Yes	Yes	Yes
Number of Layers	5	5	6	7	16	16
Highest Number of Nodes	256	256	512	512	512	512
Optimizer	rmsprop	rmsprop	adam	adam	adam	adam
Learning Rate Adjustable	No	No	No	Yes	Yes	Yes
Validation Accuracy	64.2%	73.4%	80.2%	91.15%	93.2%	96.22%
Testing Accuracy	69.4%	72.2%	80.0%	91.93%	94.6%	96.7%
Epochs	14	11	56	38	6	2

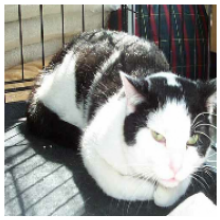
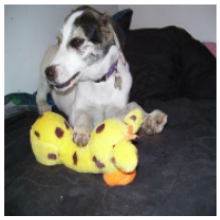
2. Data preparation

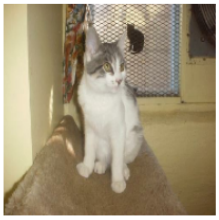
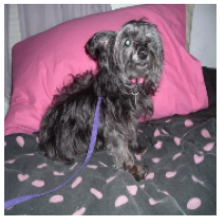
I downloaded a full cat and dog dataset from Kaggle (https://www.microsoft.com/en-us/download/details.aspx?id=54765). There are 12,500 picture of cats and 12,500 picture of dogs. I cleaned data by removing all corrupted image, around 300 pictures have been removed. Then I randomly create 3 datasets.

a. Super small dataset (total of 2,000 pictures): a 1,000-training dataset (500 for cats and 500 for dogs), a 500-validation dataset (250 for cats and 250 for dogs), and a 500-testing dataset (250 for cats and 250 for dogs).
b. Small dataset (total of 3,000 pictures): a 2,000-training dataset (500 for cats and 500 for dogs), a 500-validation dataset (250 for cats and 250 for dogs), and a 500-testing dataset (250 for cats and 250 for dogs).
c. Large dataset (total of 20,000 pictures): a 12,000-training dataset (6,000 for cats and 6,000 for dogs), a 4,000-validation dataset (2,000 for cats and 2,000 for dogs), and a 4,000-testing dataset (2,000 for cats and 2,000 for dogs).

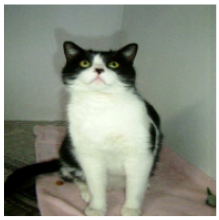


Validation Set - Cats & Dogs

Label: Cat 
Label: Cat 
Label: Cat 
Label: Dog 
Label: Dog 

Label: Cat 
Label: Cat 
Label: Dog 
Label: Cat 
Label: Dog 

Test Set - Cats & Dogs

Label: Cat 
Label: Cat 
Label: Cat 
Label: Dog 
Label: Dog 

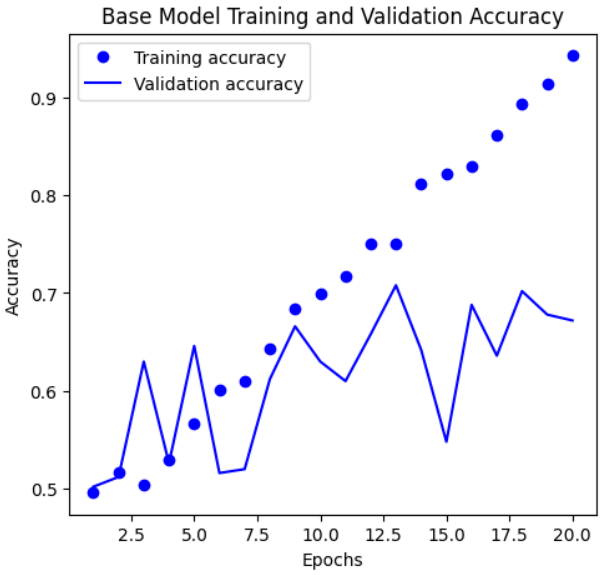
Label: Dog 
Label: Cat 
Label: Dog 
Label: Dog 
Label: Dog 

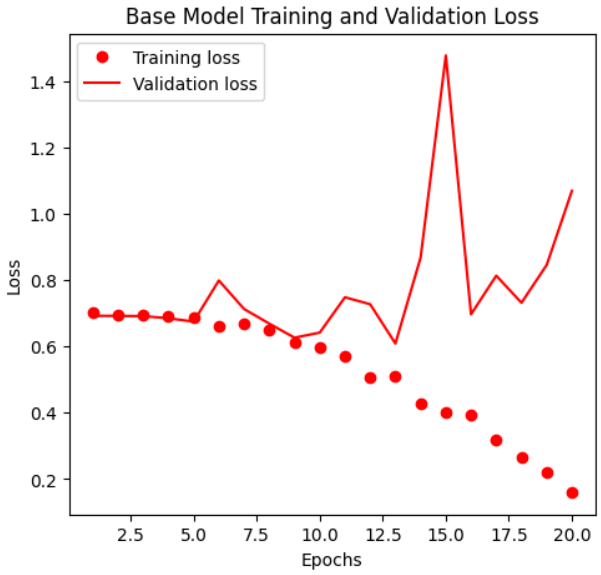
3. Base Model

This model uses the super small dataset and has five Conv2D layers with 32, 64, 128, 256, and 256 nodes, respectively. Since the output is either cat or dog, it is a binary classification problem. The output layer has one node with a 'Sigmoid' activation function. The loss function is 'binary_crossentropy,' and accuracy is used as the performance metric. I included Early Stopping to monitor validation loss ('val_loss') with a patience setting of 7.

```
inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
base_model = keras.Model(inputs=inputs, outputs=outputs)

base_model.compile(loss="binary_crossentropy",
                   optimizer="rmsprop",
                   metrics=["accuracy"])

base_model.summary()

base_model_callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="base_model.keras",
        save_best_only=True,
        monitor="val_loss"  # Tracks the validation loss
    ),
    keras.callbacks.EarlyStopping(
        monitor="val_loss",
        patience=7,  # Stop if no improvement after 5 epochs
        restore_best_weights=True
    )
]
base_model_history = base_model.fit(
 train_dataset,
    epochs=75,
    validation_data=validation_dataset,
    callbacks=base_model_callbacks
)
```

Layer (type)	Output Shape	Param #
input_layer_93 (InputLayer)	(None, 180, 180, 3)	0
rescaling_48 (Rescaling)	(None, 180, 180, 3)	0
conv2d_249 (Conv2D)	(None, 178, 178, 32)	896
max_pooling2d_230 (MaxPooling2D)	(None, 89, 89, 32)	0
conv2d_250 (Conv2D)	(None, 87, 87, 64)	18,496
max_pooling2d_231 (MaxPooling2D)	(None, 43, 43, 64)	0
conv2d_251 (Conv2D)	(None, 41, 41, 128)	73,856
max_pooling2d_232 (MaxPooling2D)	(None, 20, 20, 128)	0
conv2d_252 (Conv2D)	(None, 18, 18, 256)	295,168
max_pooling2d_233 (MaxPooling2D)	(None, 9, 9, 256)	0
conv2d_253 (Conv2D)	(None, 7, 7, 256)	590,080
flatten_53 (Flatten)	(None, 12544)	0
dense_100 (Dense)	(None, 1)	12,545





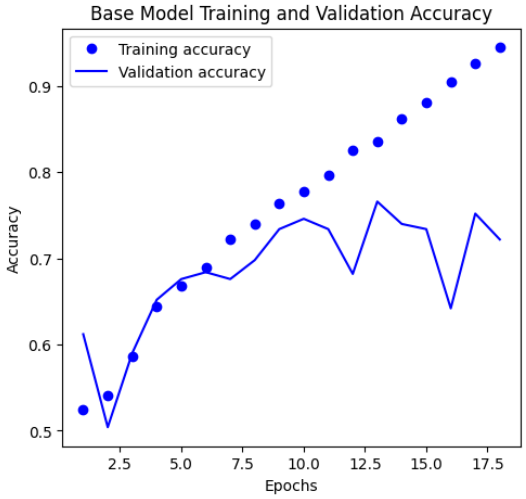

```
Epoch 14/75
Corrupt JPEG data: 2226 extraneous bytes before marker 0xd9
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 75ms/step - accuracy: 0.8114 - loss: 0.4467 - val_accuracy: 0.6420 - val_loss: 0.8676
```

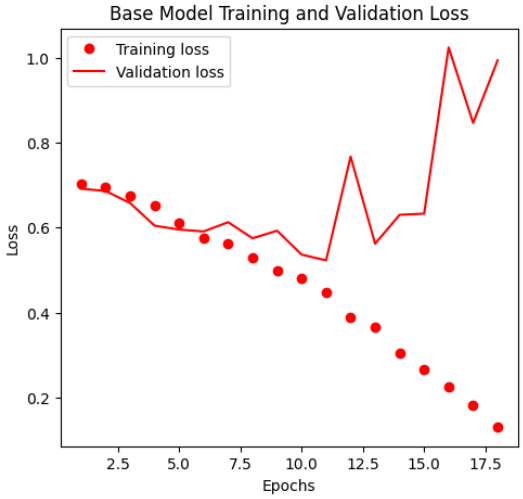
The best validation loss is 0.8676, with a validation accuracy of 64.2% at epoch 14. I evaluated the base model (at the epoch with the best validation loss, epoch 8), and the testing accuracy of the base model is 69.4%.

```
13/16 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - accuracy: 0.6518 - loss: 0.6586
Corrupt JPEG data: 65 extraneous bytes before marker 0xd9
16/16 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.6613 - loss: 0.6468
Test accuracy: 0.6940
```

4. Adjusted Base Model

This model is constructed with the same structure as the Base Model. The main difference is the dataset; this model uses the small dataset, which has a higher number of training samples. I expect to achieve better results.






```
Epoch 11/100
51/63 ━━━━━━━━━━━━━━━━━━━━ 0s 67ms/step - accuracy: 0.7778 - loss: 0.4816
Corrupt JPEG data: 214 extraneous bytes before marker 0xd9
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 73ms/step - accuracy: 0.7816 - loss: 0.4752 - val_accuracy: 0.7340 - val_loss: 0.5230
```

As expected, the best validation loss at epoch 11 is 0.5230, which is lower than that of the Base Model. The same trend applies to the validation accuracy, which is 73.4%, showing a significant 10% improvement. I evaluated the Adjusted Base Model (at the epoch with the best validation loss, epoch 11), and the testing accuracy of the adjusted base model is 72.2%.

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.7150 - loss: 0.5609
Test accuracy: 0.7220
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 143ms/step
```

5. Improved Adjusted Base Model

The small dataset is used in this model. There are five Conv2D layers consisting of 32, 64, 128, 256, and 512 nodes, respectively, as I increased the number of nodes from 256 to 512 in the last layer. In the output layer, I added one Dense layer with 256 nodes and included a Global Average Pooling layer. Lastly, I used a Flatten layer before the final output layer, which is a Dense layer with one node and a 'Sigmoid' activation function. The loss function is 'binary_crossentropy,' and accuracy is used as the performance metric. I included Early Stopping to monitor validation loss ('val_loss') with a patience setting of 7. Additionally, I changed the optimizer from 'rmsprop' to 'adam'.

```
inputs = keras.Input(shape=(180, 180, 3))

x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)

x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)

x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)

x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)

x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)

x = layers.Conv2D(filters=512, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)

x = layers.Dense(256, activation='relu')(x)
x = layers.GlobalAveragePooling2D()(x)

x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)

improved_adj_base_model = keras.Model(inputs=inputs, outputs=outputs)

# Compile the model with a learning rate reduction callback
improved_adj_base_model.compile(loss="binary_crossentropy",
                optimizer="adam",
                metrics=["accuracy"])

improved_adj_base_model_callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="improved_adj_base_model.keras",
        save_best_only=True,
        monitor="val_loss"  # Tracks the validation loss
    ),
    keras.callbacks.EarlyStopping(
        monitor="val_loss",
        patience=7,  # Stop if no improvement after 5 epochs
        restore_best_weights=True
    )
]

improved_adj_base_model_history = improved_adj_base_model.fit(
    improved_adj_base_model_train_dataset,
    epochs=100,
    callbacks=improved_adj_base_model_callbacks,
    validation_data=improved_adj_base_model_validation_dataset
)
```

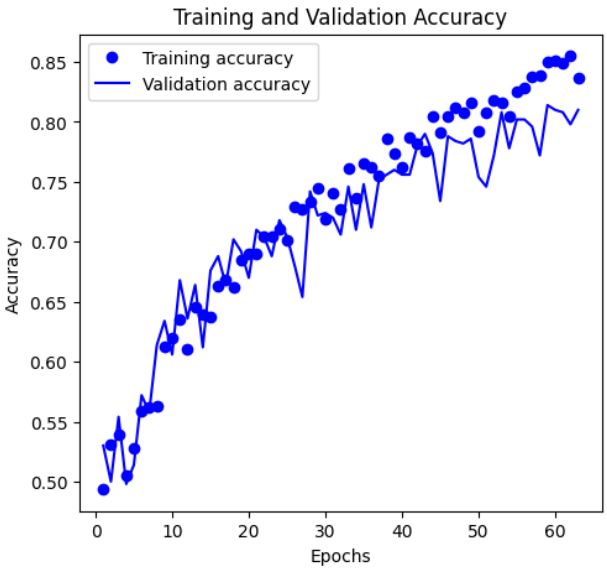
Layer (type)	Output Shape	Param #
input_layer_96 (InputLayer)	(None, 180, 180, 3)	0
sequential_35 (Sequential)	(None, 180, 180, 3)	0
rescaling_51 (Rescaling)	(None, 180, 180, 3)	0
conv2d_264 (Conv2D)	(None, 178, 178, 32)	896
max_pooling2d_242 (MaxPooling2D)	(None, 89, 89, 32)	0
conv2d_265 (Conv2D)	(None, 87, 87, 64)	18,496
max_pooling2d_243 (MaxPooling2D)	(None, 43, 43, 64)	0
conv2d_266 (Conv2D)	(None, 41, 41, 128)	73,856
max_pooling2d_244 (MaxPooling2D)	(None, 20, 20, 128)	0
conv2d_267 (Conv2D)	(None, 18, 18, 256)	295,168
max_pooling2d_245 (MaxPooling2D)	(None, 9, 9, 256)	0
conv2d_268 (Conv2D)	(None, 7, 7, 512)	1,180,160
max_pooling2d_246 (MaxPooling2D)	(None, 3, 3, 512)	0
dense_103 (Dense)	(None, 3, 3, 256)	131,328
global_average_pooling2d_10 (GlobalAveragePooling2D)	(None, 256)	0
flatten_56 (Flatten)	(None, 256)	0
dense_104 (Dense)	(None, 1)	257

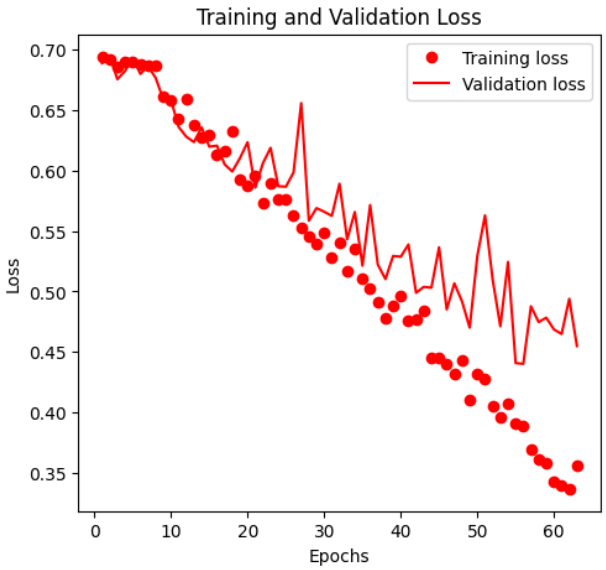
```
Epoch 56/100
25/63 ━━━━━━━━━━━━━━━━━━━━ 3s 82ms/step - accuracy: 0.8208 - loss: 0.3964
Corrupt JPEG data: 214 extraneous bytes before marker 0xd9
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 88ms/step - accuracy: 0.8216 - loss: 0.3967 - val_accuracy: 0.8020 - val_loss: 0.4402
```

After training the model, the validation accuracy increases dramatically from 74.2% to 80.2% at 56 epochs. The validation loss decreases from 0.5230 to 0.4402 compared to the

Adjusted Base Model. This improvement results from increasing the number of training samples from 1,000 to 2,000 observations. The testing accuracy is 80%, which is a significant improvement over the Adjusted Base Model.

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.7940 - loss: 0.4689
Test accuracy: 0.8000
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 143ms/step
```





6. Improved Model

In this model, I used a large dataset. I also added one more Dense layer with 64 nodes in the output block. In total, there are two Dense layers in the output block with 256 and 64 nodes, respectively. I left the rest of the model configuration the same as that of the Improved Adjusted Base Model.

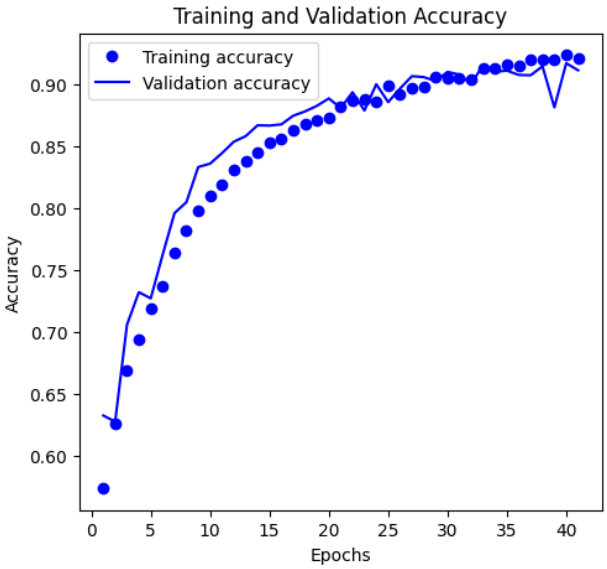
```python
inputs = keras.Input(shape=(180, 180, 3))

x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)

x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)

x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)

x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)

x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)

x = layers.Conv2D(filters=512, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)

x = layers.Dense(256, activation='relu')(x)
x = layers.Dense(64, activation='relu')(x)
x = layers.GlobalAveragePooling2D()(x)

x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)

improved_model = keras.Model(inputs=inputs, outputs=outputs)

improved_model.summary()
```

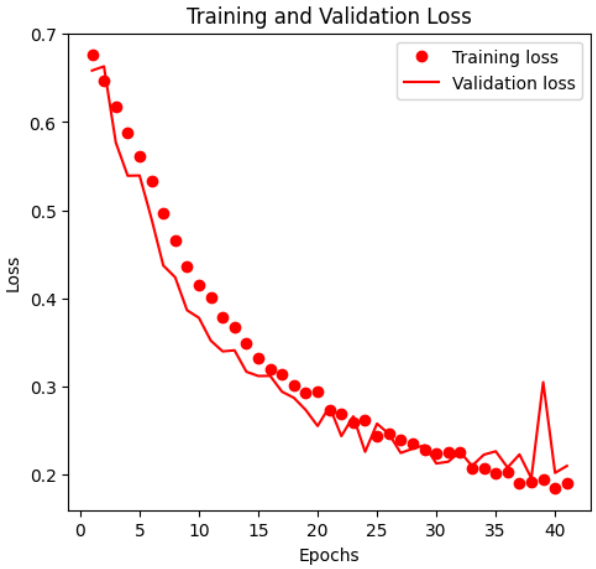
Layer (type)	Output Shape	Param #
input_layer_98 (InputLayer)	(None, 180, 180, 3)	0
sequential_36 (Sequential)	(None, 180, 180, 3)	0
rescaling_52 (Rescaling)	(None, 180, 180, 3)	0
conv2d_269 (Conv2D)	(None, 178, 178, 32)	896
max_pooling2d_247 (MaxPooling2D)	(None, 89, 89, 32)	0
conv2d_270 (Conv2D)	(None, 87, 87, 64)	18,496
max_pooling2d_248 (MaxPooling2D)	(None, 43, 43, 64)	0
conv2d_271 (Conv2D)	(None, 41, 41, 128)	73,856
max_pooling2d_249 (MaxPooling2D)	(None, 20, 20, 128)	0
conv2d_272 (Conv2D)	(None, 18, 18, 256)	295,168
max_pooling2d_250 (MaxPooling2D)	(None, 9, 9, 256)	0
conv2d_273 (Conv2D)	(None, 7, 7, 512)	1,180,160
max_pooling2d_251 (MaxPooling2D)	(None, 3, 3, 512)	0
dense_105 (Dense)	(None, 3, 3, 256)	131,328
dense_106 (Dense)	(None, 3, 3, 64)	16,448
global_average_pooling2d_11 (GlobalAveragePooling2D)	(None, 64)	0
flatten_57 (Flatten)	(None, 64)	0
dense_107 (Dense)	(None, 1)	65

```python
# Compile the model with a learning rate reduction callback
improved_model.compile(loss="binary_crossentropy",
                       optimizer="adam",
                       metrics=["accuracy"])

improved_model_callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="improved_model.keras",
        save_best_only=True,
        monitor="val_loss"  # Tracks the validation loss
    ),
    keras.callbacks.EarlyStopping(
        monitor="val_loss",
        patience=7,  # Stop if no improvement after 5 epochs
        restore_best_weights=True
    )
]

improved_model_history = improved_model.fit(
    improved_model_train_dataset,
    epochs=100,
    callbacks=improved_model_callbacks,
    validation_data=improved_model_validation_dataset
)
```

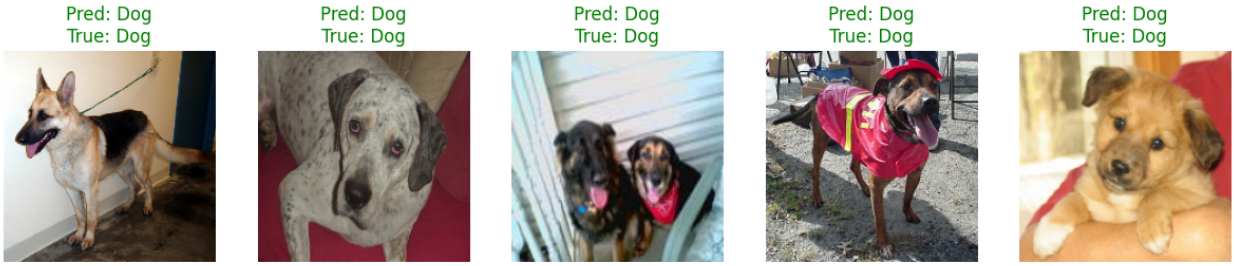
```
Epoch 38/100
156/375 ━━━━━━━━━━━━━━━━━━━━ 18s 85ms/step - accuracy: 0.9179 - loss: 0.1924
Corrupt JPEG data: 128 extraneous bytes before marker 0xd9
Corrupt JPEG data: 99 extraneous bytes before marker 0xd9
212/375 ━━━━━━━━━━━━━━━━━━━━ 13s 85ms/step - accuracy: 0.9193 - loss: 0.1911
Corrupt JPEG data: 396 extraneous bytes before marker 0xd9
237/375 ━━━━━━━━━━━━━━━━━━━━ 11s 85ms/step - accuracy: 0.9196 - loss: 0.1910
Corrupt JPEG data: 228 extraneous bytes before marker 0xd9
298/375 ━━━━━━━━━━━━━━━━━━━━ 6s 85ms/step - accuracy: 0.9200 - loss: 0.1904
Corrupt JPEG data: 252 extraneous bytes before marker 0xd9
375/375 ━━━━━━━━━━━━━━━━━━━━ 0s 84ms/step - accuracy: 0.9203 - loss: 0.1900
Corrupt JPEG data: 1403 extraneous bytes before marker 0xd9
375/375 ━━━━━━━━━━━━━━━━━━━━ 34s 91ms/step - accuracy: 0.9203 - loss: 0.1900 - val_accuracy: 0.9150 - val_loss: 0.1957
```

With a larger training dataset, the performance of the model noticeably increases, as expected. At the best epoch, epoch 38, the validation accuracy is 91.5%, and the validation loss is 0.1957. This represents a 10% increase in validation accuracy compared to the Improved Adjusted Base Model. The test accuracy is 91.93%, which is much higher than that of the Improved Adjusted Base Model.






```
 40/125 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9244 - loss: 0.1999
Corrupt JPEG data: 239 extraneous bytes before marker 0xd9
 75/125 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.9239 - loss: 0.1967
Corrupt JPEG data: 162 extraneous bytes before marker 0xd9
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 0.9233 - loss: 0.1985
Test accuracy: 0.9193
```



7. VGG16 Model: Small dataset

In the last model, I will use VGG16, which is a pre-trained model. Since this model is very powerful, I set the patience parameter to 3 instead of 7. I am using a small dataset to see how the pre-trained model performs when facing a small dataset. I am using a small dataset to maximize the benefits of training with a superior model.

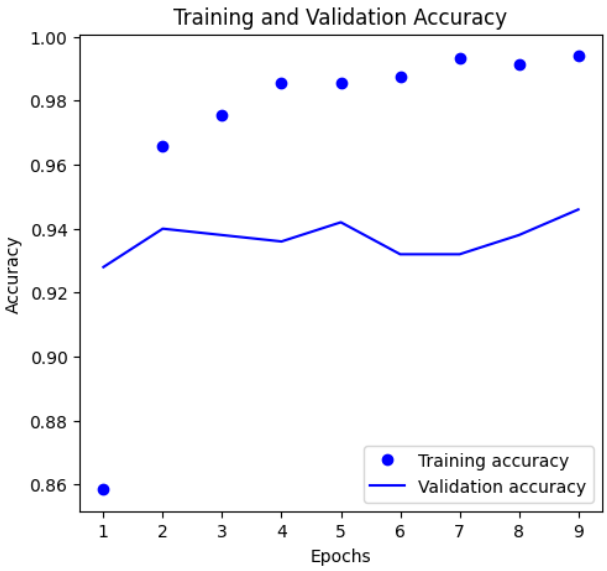
```python
# Load the VGG16 model with pre-trained ImageNet weights, without the top (classifier) layers
vgg16 = VGG16(weights='imagenet', include_top=False, input_shape=(180, 180, 3))
vgg16.trainable = False  # Freeze the base model

# Build the model
inputs = tf.keras.Input(shape=(180, 180, 3))
x = vgg16(inputs, training=False)  # Pass the inputs through the base model
x = layers.Flatten()(x)
x = layers.Dense(256, activation='relu')(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation='sigmoid')(x)  # Binary classification (cats vs. dogs)

# Create the final model
vgg16_small_model = models.Model(inputs, outputs)

vgg16_small_model.summary()

# Compile the model
vgg16_small_model.compile(optimizer=Adam(learning_rate=1e-4),
                          loss='binary_crossentropy',
                          metrics=['accuracy'])

vgg16_small_model_callbacks = [
    tf.keras.callbacks.ModelCheckpoint(
        filepath="vgg16_small_model.keras",
        save_best_only=True,
        monitor="val_loss"  # Tracks the validation loss),
    tf.keras.callbacks.EarlyStopping(
        monitor="val_loss",
        patience=3,  # Stop if no improvement after 7 epochs
        restore_best_weights=True)
]

# Train the model
vgg16_small_model_history = vgg16_small_model.fit(
    vgg16_small_train_dataset,
    validation_data=vgg16_small_validation_dataset,
    epochs=50,  # Corrected from epoch=50 to epochs=50
    callbacks=vgg16_small_model_callbacks)
```

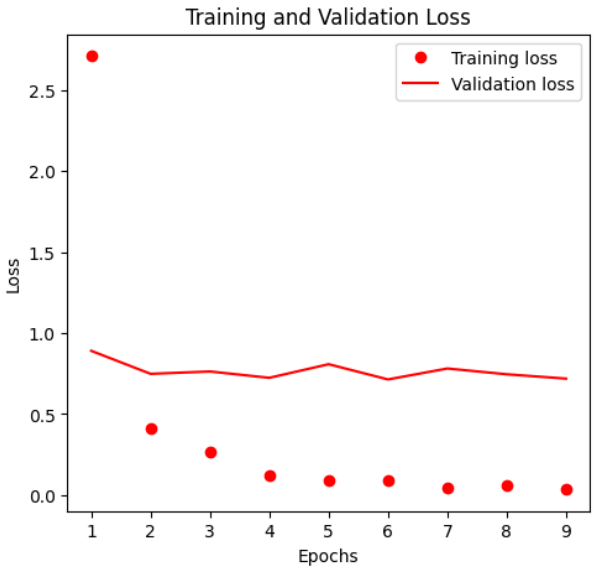
Layer (type)	Output Shape	Param #
input_layer_103 (InputLayer)	(None, 180, 180, 3)	0
vgg16 (Functional)	(None, 5, 5, 512)	14,714,688
flatten_59 (Flatten)	(None, 12800)	0
dense_110 (Dense)	(None, 256)	3,277,056
dropout_27 (Dropout)	(None, 256)	0
dense_111 (Dense)	(None, 1)	257

```
Epoch 6/50
46/63 ━━━━━━━━━━━━━━━━━━━━ 2s 171ms/step - accuracy: 0.9869 - loss: 0.0880
Corrupt JPEG data: 214 extraneous bytes before marker 0xd9
63/63 ━━━━━━━━━━━━━━━━━━━━ 14s 220ms/step - accuracy: 0.9869 - loss: 0.0911 - val_accuracy: 0.9320 - val_loss: 0.7140
```

With a small training dataset, the performance of VGG16 is slightly better than that of my own model, the Improved Model. The validation accuracy increased from 91.5% to 93.2%, demonstrating the effectiveness of the pre-trained model. The features from the pre-trained model are particularly helpful in mitigating the issues associated with a very limited training dataset. Finally, the testing accuracy reached 94.6%.

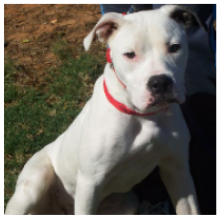
```
16/16 ━━━━━━━━━━━━━━━━━━━━ 3s 174ms/step - accuracy: 0.9466 - loss: 0.5522
Test accuracy: 0.9460
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
```





Pred: Cat
True: Cat

Pred: Dog
True: Dog



Pred: Dog
True: Dog

Pred: Cat
True: Cat

Pred: Dog
True: Dog

Pred: Dog
True: Dog

Pred: Cat
True: Cat

Pred: Dog
True: Dog

Pred: Dog
True: Cat

Pred: Dog
True: Dog

8. VGG16 Model: Large dataset

In the last model, I will use VGG16, which is a pre-trained model. Since this model is very powerful, I set the patience parameter to 3 instead of 7. I am using a small dataset to maximize the benefits of training with a superior model.

```python
# Load the VGG16 model with pre-trained ImageNet weights, without the top (classifier) layers
vgg16 = VGG16(weights='imagenet', include_top=False, input_shape=(180, 180, 3))
vgg16.trainable = False  # Freeze the base model

# Build the model
inputs = tf.keras.Input(shape=(180, 180, 3))
x = vgg16(inputs, training=False)  # Pass the inputs through the base model
x = layers.Flatten()(x)
x = layers.Dense(256, activation='relu')(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation='sigmoid')(x)  # Binary classification (cats vs. dogs)

# Create the final model
vgg16_large_model = models.Model(inputs, outputs)

vgg16_large_model.summary()

# Compile the model
vgg16_large_model.compile(optimizer=Adam(learning_rate=1e-4),
                          loss='binary_crossentropy',
                          metrics=['accuracy'])

vgg16_large_model_callbacks = [
    tf.keras.callbacks.ModelCheckpoint(
        filepath="vgg16_large_model.keras",
        save_best_only=True,
        monitor="val_loss"  # Tracks the validation loss),
    tf.keras.callbacks.EarlyStopping(
        monitor="val_loss",
        patience=3,  # Stop if no improvement after 7 epochs
        restore_best_weights=True)
]

# Train the model
vgg16_large_model_history = vgg16_large_model.fit(
    vgg16_large_train_dataset,
    validation_data=vgg16_large_validation_dataset,
    epochs=50,  # Corrected from epoch=50 to epochs=50
    callbacks=vgg16_large_model_callbacks
)
```

Layer (type)	Output Shape	Param #
input_layer_103 (InputLayer)	(None, 180, 180, 3)	0
vgg16 (Functional)	(None, 5, 5, 512)	14,714,688
flatten_59 (Flatten)	(None, 12800)	0
dense_110 (Dense)	(None, 256)	3,277,056
dropout_27 (Dropout)	(None, 256)	0
dense_111 (Dense)	(None, 1)	257

```
Epoch 2/50
 30/375 ━━━━━━━━━━━━━━━━━━━━ 59s 172ms/step - accuracy: 0.9492 - loss: 0.2574
Corrupt JPEG data: 128 extraneous bytes before marker 0xd9
 93/375 ━━━━━━━━━━━━━━━━━━━━ 48s 173ms/step - accuracy: 0.9578 - loss: 0.2212
Corrupt JPEG data: 396 extraneous bytes before marker 0xd9
118/375 ━━━━━━━━━━━━━━━━━━━━ 44s 173ms/step - accuracy: 0.9594 - loss: 0.2130
Corrupt JPEG data: 228 extraneous bytes before marker 0xd9
165/375 ━━━━━━━━━━━━━━━━━━━━ 36s 172ms/step - accuracy: 0.9613 - loss: 0.1997
Corrupt JPEG data: 252 extraneous bytes before marker 0xd9
199/375 ━━━━━━━━━━━━━━━━━━━━ 30s 172ms/step - accuracy: 0.9622 - loss: 0.1929
Corrupt JPEG data: 99 extraneous bytes before marker 0xd9
375/375 ━━━━━━━━━━━━━━━━━━━━ 0s 172ms/step - accuracy: 0.9646 - loss: 0.1725
Corrupt JPEG data: 1403 extraneous bytes before marker 0xd9
375/375 ━━━━━━━━━━━━━━━━━━━━ 85s 227ms/step - accuracy: 0.9646 - loss: 0.1724 - val_accuracy: 0.9622 - val_loss: 0.1770
```

The performance of VGG16 is much better than that of my own model, the Improved Model. The validation accuracy increases from 90.7% to 96.22%. And around 3% better than VGG16 with small dataset. Moreover, with a larger training dataset, fewer iterations are needed. Lastly, the testing accuracy is 96.7%.

```
 52/125 ━━━━━━━━━━━━━━━━━━━━ 12s 165ms/step - accuracy: 0.9675 - loss: 0.1468
Corrupt JPEG data: 162 extraneous bytes before marker 0xd9
102/125 ━━━━━━━━━━━━━━━━━━━━ 3s 166ms/step - accuracy: 0.9664 - loss: 0.1484
Corrupt JPEG data: 239 extraneous bytes before marker 0xd9
125/125 ━━━━━━━━━━━━━━━━━━━━ 21s 165ms/step - accuracy: 0.9656 - loss: 0.1527
Test accuracy: 0.9622
```