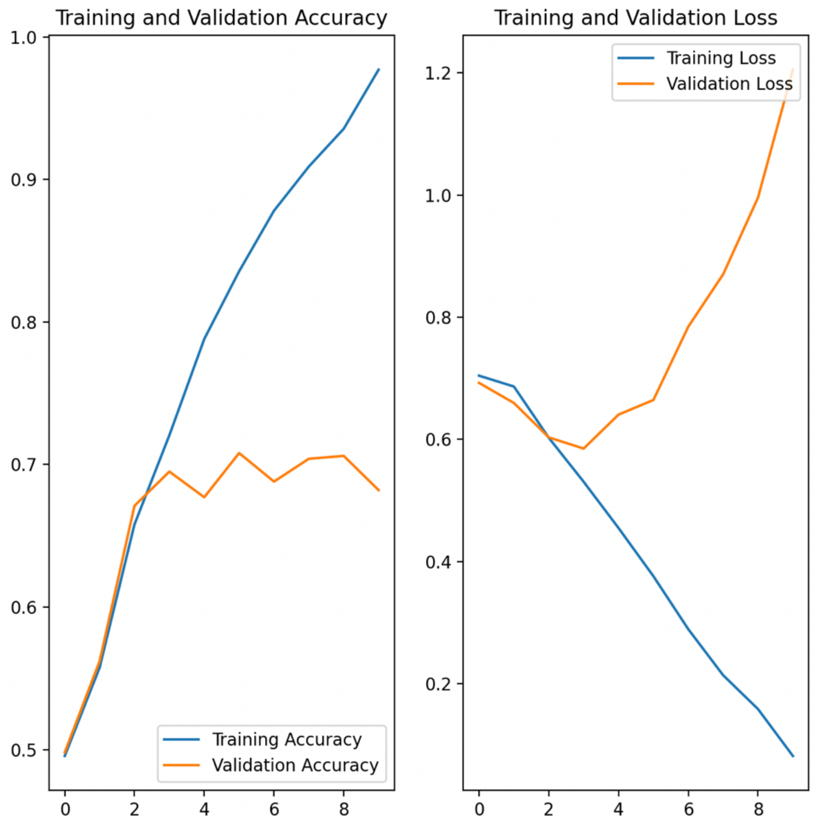
Thomas Selin
DT075 Datateknik AV, Tillämpad maskininlärning
Laboration 2
2024-10-25

Inspecting the original plots



Comment:
Both graphs show signs of overfitting as the training set has much better accuracy and its loss goes down while validation loss goes up shows that the model is learning/memorizing the training data and not gaining learnings that can be generalized on the validation set or other unseen data.

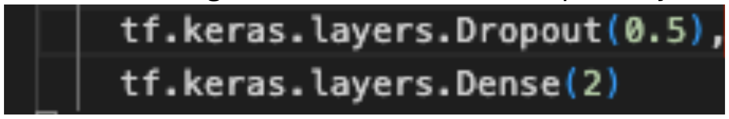
Also, reaching almost 100% accuracy for the training set increases the suspicion of overfitting.

A few possible reasons for overfitting that should be addressed:
- Not enough regularization (dropout etc.)
- Too limited dataset

Also, running the test for a cat image resulted in a wrong prediction (dog).

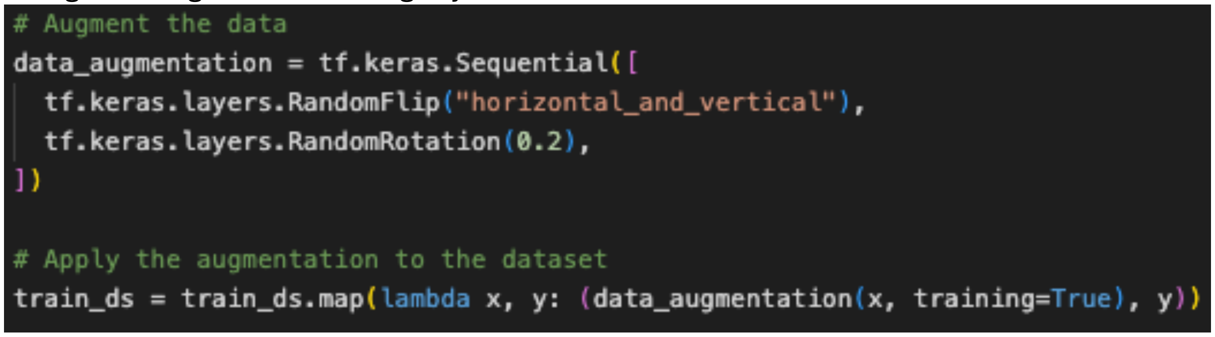
Improving the accuracy of the model

To add more regularization, I added a dropout layer as the second last layer.



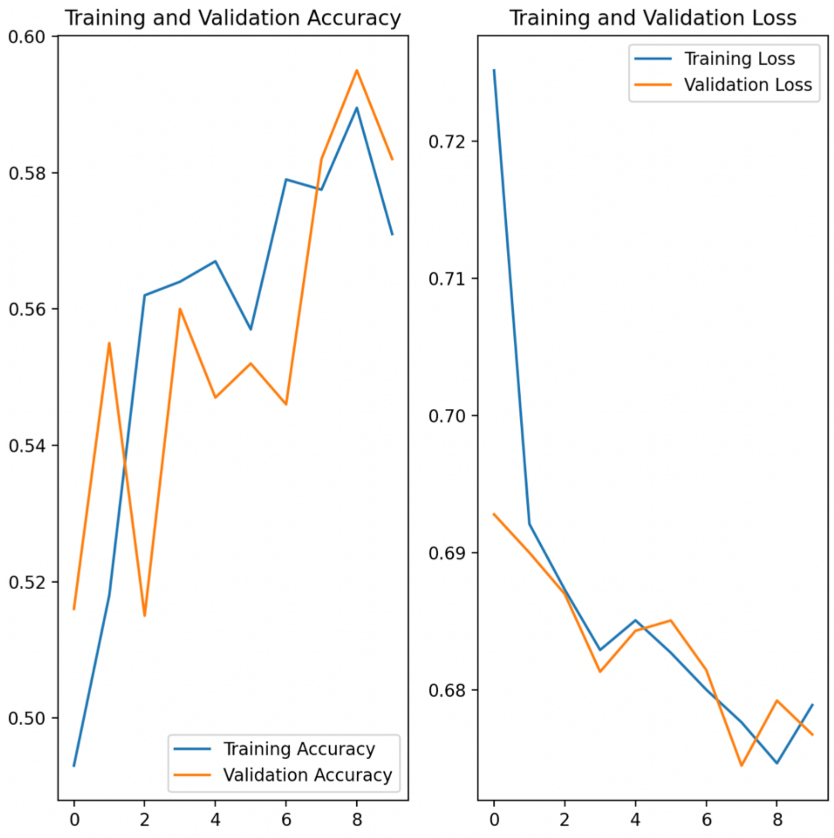
```
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(2)
```

I augmented the data to lessen the risk that the model learns the training data too exactly, and not being able to generalize to slightly different scenarios.



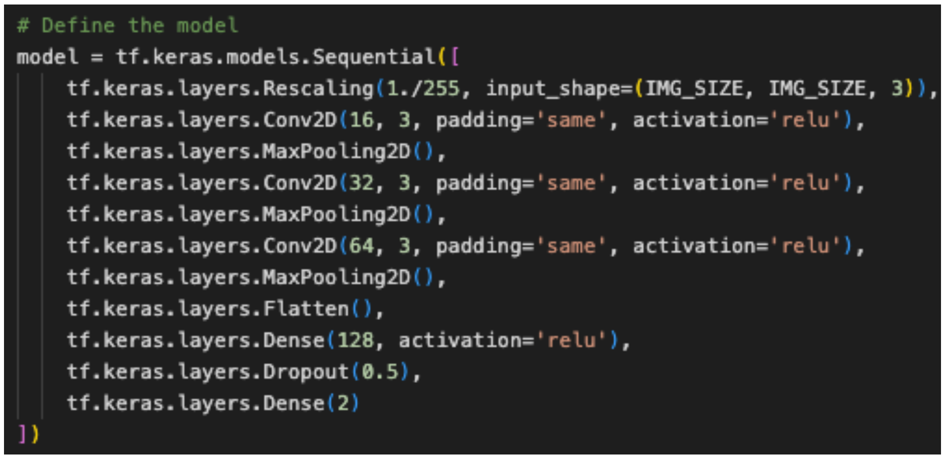
```
# Augment the data
data_augmentation = tf.keras.Sequential([
  tf.keras.layers.RandomFlip("horizontal_and_vertical"),
  tf.keras.layers.RandomRotation(0.2),
])

# Apply the augmentation to the dataset
train_ds = train_ds.map(lambda x, y: (data_augmentation(x, training=True), y))
```

When having done these two steps I got these plots:

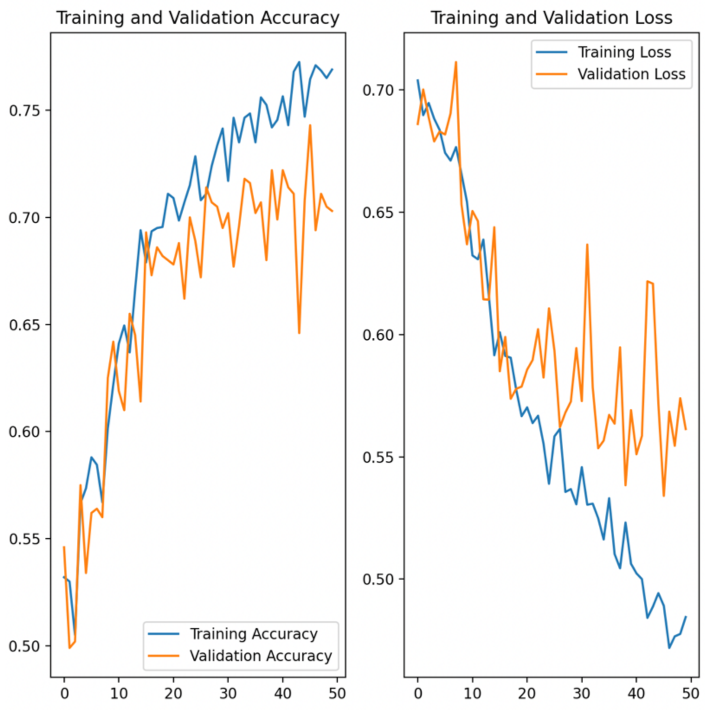


These plots strongly suggest that overfitting has lessened.
Still the test predicts **dog** instead of the correct prediction which is **cat**.

I then tried to increase nr. of neurons per layer and added 1 more layer plus a MaxPooling2D() layer:

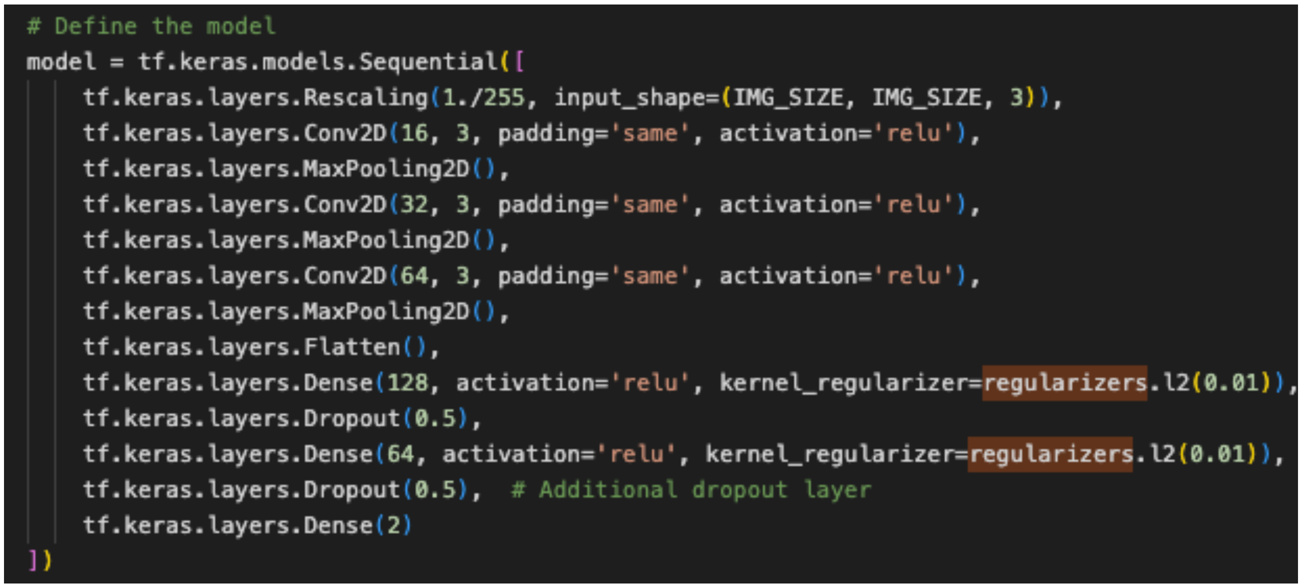

```
# Define the model
model = tf.keras.models.Sequential([
    tf.keras.layers.Rescaling(1./255, input_shape=(IMG_SIZE, IMG_SIZE, 3)),
    tf.keras.layers.Conv2D(16, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(32, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(2)
])
```

Also, I increased nr of Epochs to 50 to enable longer training now when the model is more complex. See plots below. The accuracy has increased. The test still gave the wrong prediction.



I then added one more dropout layer after the first dense layer to help prevent overfitting.

I added one more dense layer and I also added L2 regularization to the dense layers, except the output layer, to penalize large weights. That can also prevent overfitting.

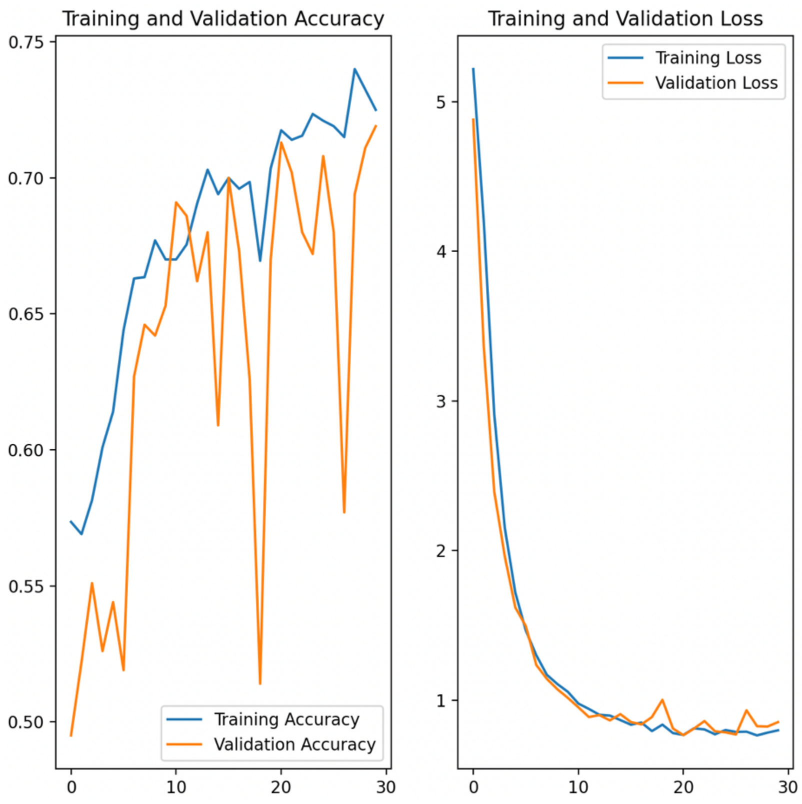


```
# Define the model
model = tf.keras.models.Sequential([
    tf.keras.layers.Rescaling(1./255, input_shape=(IMG_SIZE, IMG_SIZE, 3)),
    tf.keras.layers.Conv2D(16, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(32, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(64, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    tf.keras.layers.Dropout(0.5),  # Additional dropout layer
    tf.keras.layers.Dense(2)
])
```

This model still gave wrong prediction in the test (and the plots were strange)

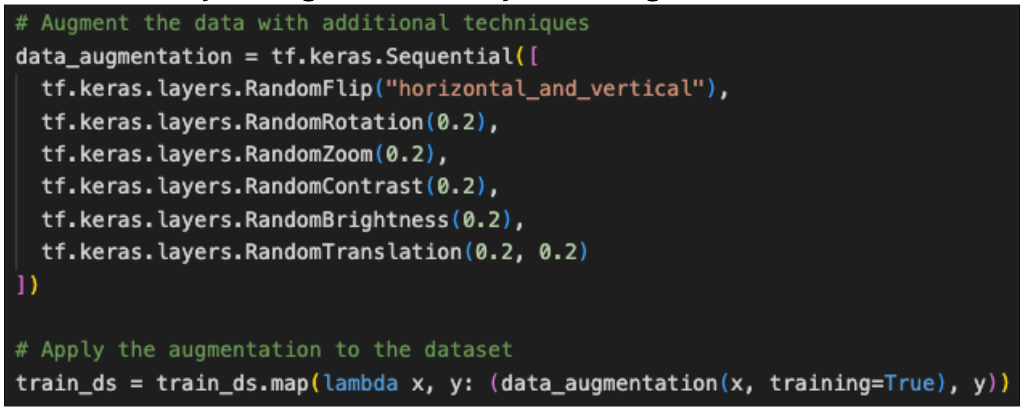
Now I tried the following:

1. Added BatchNormalization layers which can help with regularization.
2. Use double Conv2D before each MaxPooling to be able to learn more complex input features.
3. Increased number of filters to improve ability to learn complex/detailed patterns.
4. Added a softmax activation for final layer to make model output more interpretable. Gives a numerical value for the probability of an input image belonging being a cat and dog.



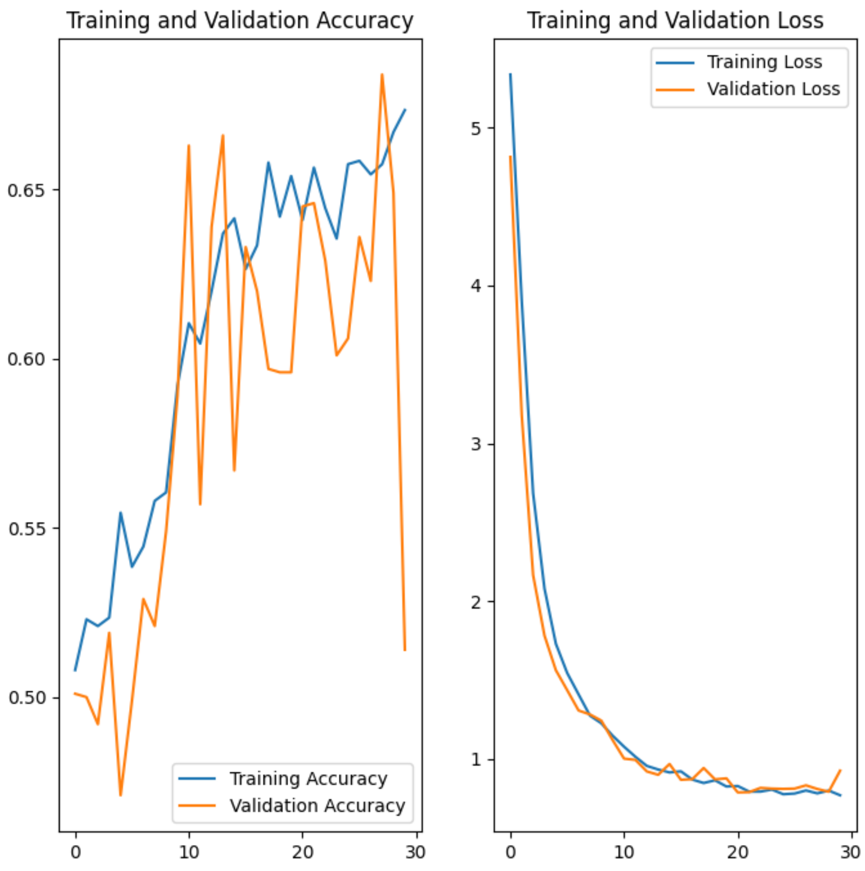
The left graph shows the highest accuracy for the validation so far, but this model still gave wrong prediction in the test.

I then tried to improve the data augmentation to try to improve the robustness and generalization of the model by adding more diversity to training data.



```python
# Augment the data with additional techniques
data_augmentation = tf.keras.Sequential([
  tf.keras.layers.RandomFlip("horizontal_and_vertical"),
  tf.keras.layers.RandomRotation(0.2),
  tf.keras.layers.RandomZoom(0.2),
  tf.keras.layers.RandomContrast(0.2),
  tf.keras.layers.RandomBrightness(0.2),
  tf.keras.layers.RandomTranslation(0.2, 0.2)
])

# Apply the augmentation to the dataset
train_ds = train_ds.map(lambda x, y: (data_augmentation(x, training=True), y))
```

The plots looked like this:



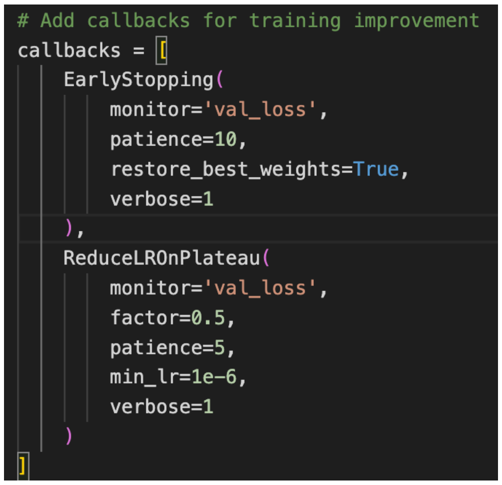
It didn’t help so I reversed the latest changes to data augmentation. Instead, I added images of about 11000 cats and 11000 dogs more from Kaggle.com to the training data. I changed randomflip() I the data augmentation to only do horizontal flip, as vertical flip doesn’t make as much sense for normal animal pictures and because is saw no pictures with cats/dogs upside down. I also added a random zoom to try to improve the data augmentation (tf.keras.layers.RandomZoom(0.2)).

I added a dropout layer after each convolutional block to lessen overfitting.
I specified the default pool_size value for clarity. I increased nr. of neurons in the second last dense layer to hopefully be able to model this quite complex data better.

I decreased the regularization strength (0.01 -> 0.001) as strong regularization can make learning unstable, which we saw signs of in earlier plot.

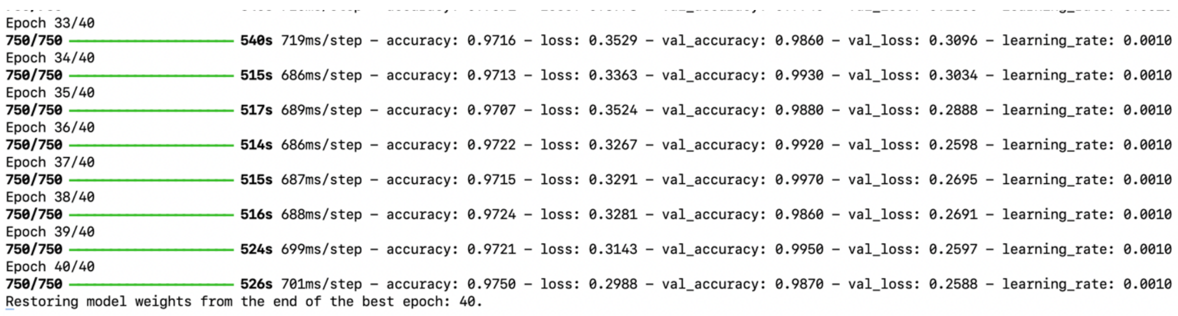
I added a callback to stop training early when enough convergence has happened, which can also prevent overfitting. See image below.

I added a callback to reduce the learning rate on plateau for validation loss, which can improve efficiency of training/convergence and potentially lead to better accuracy.

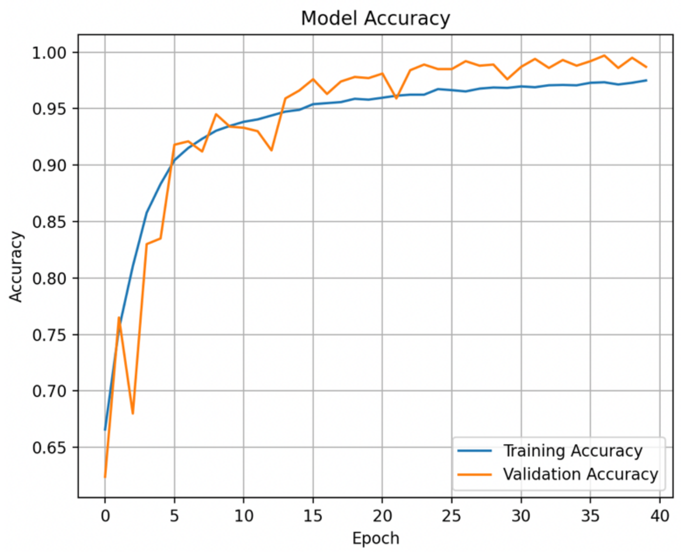


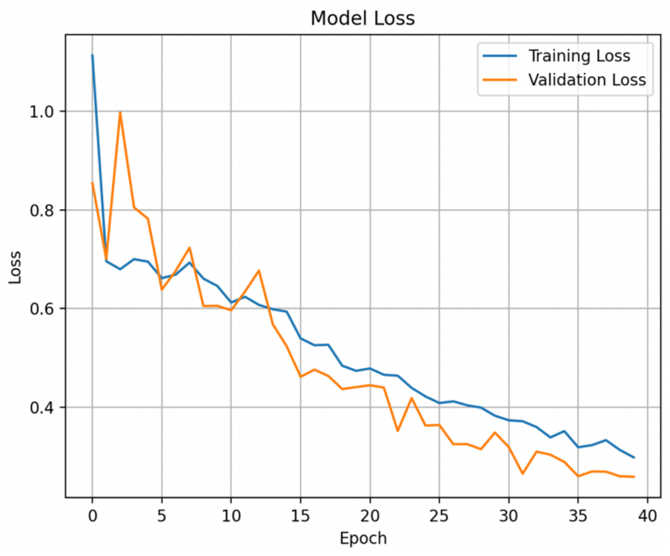
```python
# Add callbacks for training improvement
callbacks = [
    EarlyStopping(
        monitor='val_loss',
        patience=10,
        restore_best_weights=True,
        verbose=1
    ),
    ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.5,
        patience=5,
        min_lr=1e-6,
        verbose=1
    )
]
```

After training with this setup I got good results, see plot.



```
Epoch 33/40
750/750 ━━━━━━━━━━━━━━━━━━━━ 540s 719ms/step - accuracy: 0.9716 - loss: 0.3529 - val_accuracy: 0.9860 - val_loss: 0.3096 - learning_rate: 0.0010
Epoch 34/40
750/750 ━━━━━━━━━━━━━━━━━━━━ 515s 686ms/step - accuracy: 0.9713 - loss: 0.3363 - val_accuracy: 0.9930 - val_loss: 0.3034 - learning_rate: 0.0010
Epoch 35/40
750/750 ━━━━━━━━━━━━━━━━━━━━ 517s 689ms/step - accuracy: 0.9707 - loss: 0.3524 - val_accuracy: 0.9880 - val_loss: 0.2888 - learning_rate: 0.0010
Epoch 36/40
750/750 ━━━━━━━━━━━━━━━━━━━━ 514s 686ms/step - accuracy: 0.9722 - loss: 0.3267 - val_accuracy: 0.9920 - val_loss: 0.2598 - learning_rate: 0.0010
Epoch 37/40
750/750 ━━━━━━━━━━━━━━━━━━━━ 515s 687ms/step - accuracy: 0.9715 - loss: 0.3291 - val_accuracy: 0.9970 - val_loss: 0.2695 - learning_rate: 0.0010
Epoch 38/40
750/750 ━━━━━━━━━━━━━━━━━━━━ 516s 688ms/step - accuracy: 0.9724 - loss: 0.3281 - val_accuracy: 0.9860 - val_loss: 0.2691 - learning_rate: 0.0010
Epoch 39/40
750/750 ━━━━━━━━━━━━━━━━━━━━ 524s 699ms/step - accuracy: 0.9721 - loss: 0.3143 - val_accuracy: 0.9950 - val_loss: 0.2597 - learning_rate: 0.0010
Epoch 40/40
750/750 ━━━━━━━━━━━━━━━━━━━━ 526s 701ms/step - accuracy: 0.9750 - loss: 0.2988 - val_accuracy: 0.9870 - val_loss: 0.2588 - learning_rate: 0.0010
Restoring model weights from the end of the best epoch: 40.
```

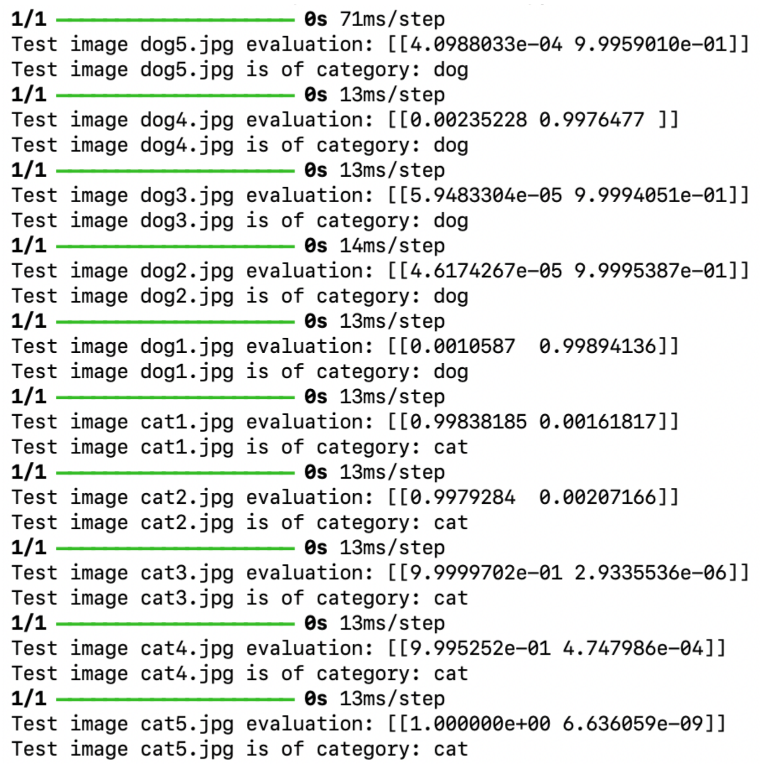




No early stoppage occurred, and the learning rate was not adjusted. Further epochs could perhaps improve the results.

I then ran the test, and it successfully predicted **cat**.
I then downloaded 5 more unseen images of cats and 5 of dogs and did the test on the (see restructured test code in Appendix 2).

I ran the test for those 10 images and all predictions were correct, suggesting a rather accurate model.



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
Test image dog5.jpg evaluation: [[4.0988033e-04 9.9959010e-01]]
Test image dog5.jpg is of category: dog
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
Test image dog4.jpg evaluation: [[0.00235228 0.9976477 ]]
Test image dog4.jpg is of category: dog
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
Test image dog3.jpg evaluation: [[5.9483304e-05 9.9994051e-01]]
Test image dog3.jpg is of category: dog
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
Test image dog2.jpg evaluation: [[4.6174267e-05 9.9995387e-01]]
Test image dog2.jpg is of category: dog
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
Test image dog1.jpg evaluation: [[0.0010587  0.99894136]]
Test image dog1.jpg is of category: dog
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
Test image cat1.jpg evaluation: [[0.99838185 0.00161817]]
Test image cat1.jpg is of category: cat
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
Test image cat2.jpg evaluation: [[0.9979284  0.00207166]]
Test image cat2.jpg is of category: cat
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
Test image cat3.jpg evaluation: [[9.9999702e-01 2.9335536e-06]]
Test image cat3.jpg is of category: cat
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
Test image cat4.jpg evaluation: [[9.995252e-01 4.747986e-04]]
Test image cat4.jpg is of category: cat
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
Test image cat5.jpg evaluation: [[1.000000e+00 6.636059e-09]]
Test image cat5.jpg is of category: cat
```

APPENDIX 1. Main code

```
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras import regularizers
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

# Variables
IMG_SIZE = 150
batch_size = 32
epochs = 40  # Increased epochs since we'll use early stopping

# Create train dataset
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    'images/train',
    image_size=(IMG_SIZE, IMG_SIZE),
    batch_size=batch_size
)

# Augment the data
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomFlip("horizontal"),  # Removed vertical flip as it's not natural for cats/dogs
    tf.keras.layers.RandomRotation(0.2),
    tf.keras.layers.RandomZoom(0.2),  # Randomly zooms images by up to 20%
])

# Create validation dataset
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
  'images/validation',
  image_size=(IMG_SIZE, IMG_SIZE),
  batch_size=batch_size)

# Define the model
model = tf.keras.Sequential([
    # Input and preprocessing
    tf.keras.layers.Rescaling(1./255, input_shape=(IMG_SIZE, IMG_SIZE, 3)),

    # First Conv Block
    tf.keras.layers.Conv2D(32, 3, padding='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(32, 3, padding='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Dropout(0.25),  # Small dropout after each block

    # Second Conv Block
    tf.keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Dropout(0.25),

    # Third Conv Block
    tf.keras.layers.Conv2D(128, 3, padding='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(128, 3, padding='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Dropout(0.25),

    # Dense layers
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu',
                         kernel_regularizer=regularizers.l2(0.001)),  # Reduced regularization strength
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),

    tf.keras.layers.Dense(2, activation='softmax')
])
```

```
# Compile with adjusted learning rate
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(
    optimizer=optimizer,
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=['accuracy']
)

# Add callbacks for training improvement
callbacks = [
    EarlyStopping(
        monitor='val_loss',
        patience=10,
        restore_best_weights=True,
        verbose=1
    ),
    ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.5,
        patience=5,
        min_lr=1e-6,
        verbose=1
    )
]

# Train the model
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs,
    callbacks=callbacks,
    verbose=1
)

# Plotting with improved visualization
plt.figure(figsize=(12, 5))

# Plot accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.grid(True)

# Plot loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper right')
plt.grid(True)

plt.tight_layout()
plt.show()

# Save the trained model for later
model.save('saved_model.keras')
```

APPENDIX 2. Main test code

```
import tensorflow as tf
import numpy as np
import os

# Load our pre-trained model
model = tf.keras.models.load_model('saved_model.keras')

# Directory containing test images
test_dir = 'images/test'

# Loop through all images in the test directory
for filename in os.listdir(test_dir):
    if filename.endswith(".jpg") or filename.endswith(".png"):  # Handle image formats
        # Load single image
        test_image_path = os.path.join(test_dir, filename)
        test_image = tf.keras.preprocessing.image.load_img(test_image_path, target_size=(150,
150))
        test_array = tf.keras.preprocessing.image.img_to_array(test_image)
        test_array = np.expand_dims(test_array, axis=0)  # Add batch dimension

        # Predict using model
        prediction = model.predict(test_array)

        # Interpret the results
        print(f"Test image {filename} evaluation: {prediction}")
        if prediction[0][0] > prediction[0][1]:
            print(f"Test image {filename} is of category: cat")
        else:
            print(f"Test image {filename} is of category: dog")
```

APPENDIX 2. Simple test code

```
import tensorflow as tf
import numpy as np

# Load our pre trained model
model = tf.keras.models.load_model('saved_model.keras')

# Load single image
test_image = tf.keras.preprocessing.image.load_img('images/mypet.jpg', target_size=(150,
150))
test_array = tf.keras.preprocessing.image.img_to_array(test_image)
test_array = np.array([test_array])

# Predict using model
prediction = model.predict(test_array)

# Interpret the results
print ("Test image evaluation: ", prediction)
if prediction[0][0] > prediction[0][1]:
    print ("Test image is of category: cat")
else:
    print ("Test image is of category: dog")
```