In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
import warnings

warnings.filterwarnings('ignore')
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy v
ersion >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version
1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

Data analysis

In [2]:
```
# Define data paths

test_dir = "/kaggle/input/cats-and-dogs-image-classification/test"
train_dir = "/kaggle/input/cats-and-dogs-image-classification/train"
```

In [3]:
```
# Check size of each dataset

import os

def get_dataset_size(dir, type):
    print(f"Scanning {type} images...")
    size = 0
    for label in os.listdir(dir):
        size += len(os.listdir(os.path.join(dir, label)))
    print(f"Number of {type} images: {size}\n")
    return size

num_test = get_dataset_size(test_dir, "test")
num_train = get_dataset_size(train_dir, "train")
```

```
Scanning test images...
Number of test images: 140

Scanning train images...
Number of train images: 557
```

In [4]:
```
num_classes = len(os.listdir(train_dir))
print(f"Number of classes: {num_classes}")
```

```
Number of classes: 2
```

In [5]:
```
# Let's view a few training samples

import random
import cv2

# imgs to plot per class
m = 5
```

```
fig, axs = plt.subplots(nrows=num_classes, ncols=m, figsize=(m*4, num_classes*4))

for i in range(num_classes):

    label = os.listdir(train_dir)[i]

    for j in range(m):

        # get a random image from the chosen species

        img_name = random.choice(os.listdir(os.path.join(train_dir, label)))
        img_path = os.path.join(train_dir, label, img_name)
        image = cv2.imread(img_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        # plot the image
        axs[i][j].imshow(image)
        axs[i][j].set_title(label)
        axs[i][j].axis("off")

fig.show()
```

Create image data generators

In [6]:
```
# Let's create our image preprocessors

from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
    horizontal_flip=True,
)

test_datagen = ImageDataGenerator(
    validation_split=0.1
)
```

In [7]:
```
# Let's define the target size

input_size = (224, 224)
```

In [8]:
```
# Let's create our image generators

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=input_size,
    batch_size=32,
    shuffle=True,
)

val_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=input_size,
    batch_size=32,
    shuffle=True,
    subset="training"
)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=input_size,
    batch_size=32,
    shuffle=False,
    subset="validation"
)
```

```
Found 557 images belonging to 2 classes.
Found 126 images belonging to 2 classes.
Found 14 images belonging to 2 classes.
```

Set up the model

Load pretrained model

Let's load a Resnet50 model which has been pretrained on the ImageNet dataset

In [9]:
```
#from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input
from tensorflow.keras.applications.vgg19 import VGG19, preprocess_input

base_model = VGG19(
    include_top = False,
    weights="imagenet",
)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vg
g19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80134624/80134624 [==============================] - 0s 0us/step
```

In [10]:
```
# Let's print out a description of the model layers

base_model.summary()
```

```
Model: "vgg19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None, None, 3)]   0

 block1_conv1 (Conv2D)       (None, None, None, 64)    1792

 block1_conv2 (Conv2D)       (None, None, None, 64)    36928

 block1_pool (MaxPooling2D)  (None, None, None, 64)    0

 block2_conv1 (Conv2D)       (None, None, None, 128)   73856

 block2_conv2 (Conv2D)       (None, None, None, 128)   147584

 block2_pool (MaxPooling2D)  (None, None, None, 128)   0

 block3_conv1 (Conv2D)       (None, None, None, 256)   295168

 block3_conv2 (Conv2D)       (None, None, None, 256)   590080

 block3_conv3 (Conv2D)       (None, None, None, 256)   590080

 block3_conv4 (Conv2D)       (None, None, None, 256)   590080

 block3_pool (MaxPooling2D)  (None, None, None, 256)   0

 block4_conv1 (Conv2D)       (None, None, None, 512)   1180160

 block4_conv2 (Conv2D)       (None, None, None, 512)   2359808

 block4_conv3 (Conv2D)       (None, None, None, 512)   2359808

 block4_conv4 (Conv2D)       (None, None, None, 512)   2359808

 block4_pool (MaxPooling2D)  (None, None, None, 512)   0

 block5_conv1 (Conv2D)       (None, None, None, 512)   2359808

 block5_conv2 (Conv2D)       (None, None, None, 512)   2359808

 block5_conv3 (Conv2D)       (None, None, None, 512)   2359808

 block5_conv4 (Conv2D)       (None, None, None, 512)   2359808

 block5_pool (MaxPooling2D)  (None, None, None, 512)   0

=================================================================
Total params: 20,024,384
Trainable params: 20,024,384
Non-trainable params: 0
_________________________________________________________________
```

In [11]:
```
len(base_model.trainable_variables)
```

Out[11]:
```
32
```

Create classification model

Let's add a classification head to this pretrained model

In [12]:
```
# Before we add the classification head,
# Let's get the output shape of the ResNet (without the classification head)

image_batch, label_batch = next(iter(train_generator))
feature_batch = base_model(image_batch)
print(f"Output shape of feature extractor: {feature_batch.shape}")
```

```
Output shape of feature extractor: (32, 7, 7, 512)
```

In [13]:
```
# Let's freeze the feature extractor

base_model.trainable = False
```

In [14]:
```
# Add a global pooling layer to flatten the output of the feature extractor

from tensorflow.keras.layers import GlobalAveragePooling2D

global_average_layer = GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
```

```
(32, 512)
```

In [15]:
```
# Add a fully connected layer

from tensorflow.keras.layers import Dense

prediction_layer = Dense(num_classes)
prediction_layer_output = prediction_layer(feature_batch_average)
print(prediction_layer_output.shape)
```

```
(32, 2)
```

In [16]:
```
# Now let's combine all the layers into a complete model

inputs = tf.keras.Input(shape=input_size + (3,))
x = preprocess_input(inputs)
x = base_model(x)
x = global_average_layer(x)
outputs = prediction_layer(x)

model = tf.keras.Model(inputs, outputs)
```

In [17]:
```
# Let's explore the model's layers

model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 tf.__operators__.getitem (S  (None, 224, 224, 3)      0
 licingOpLambda)

 tf.nn.bias_add (TFOpLambda)  (None, 224, 224, 3)      0

 vgg19 (Functional)          (None, None, None, 512)   20024384

 global_average_pooling2d (G  (None, 512)              0
 lobalAveragePooling2D)

 dense (Dense)               (None, 2)                 1026

=================================================================
Total params: 20,025,410
Trainable params: 1,026
Non-trainable params: 20,024,384
_________________________________________________________________
```

In [18]:
```
len(model.trainable_variables)
```

Out[18]:
```
2
```

Training

Training the classification model

In [19]:
```
# Let's compile the model


base_learning_rate = 0.0001

model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=base_learning_rate),
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)
```

In [20]:
```
initial_epochs = 50
```

In [21]:
```
loss0, accuracy0 = model.evaluate(val_generator)

print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
```

```
4/4 [==============================] - 13s 2s/step - loss: 5.3864 - accuracy: 0.5079
initial loss: 5.39
initial accuracy: 0.51
```

In [22]:
```
# Let's fit the model
# Remember, we are only training the classification head

history = model.fit(
    train_generator,
    epochs=initial_epochs,
    validation_data=val_generator
)
```

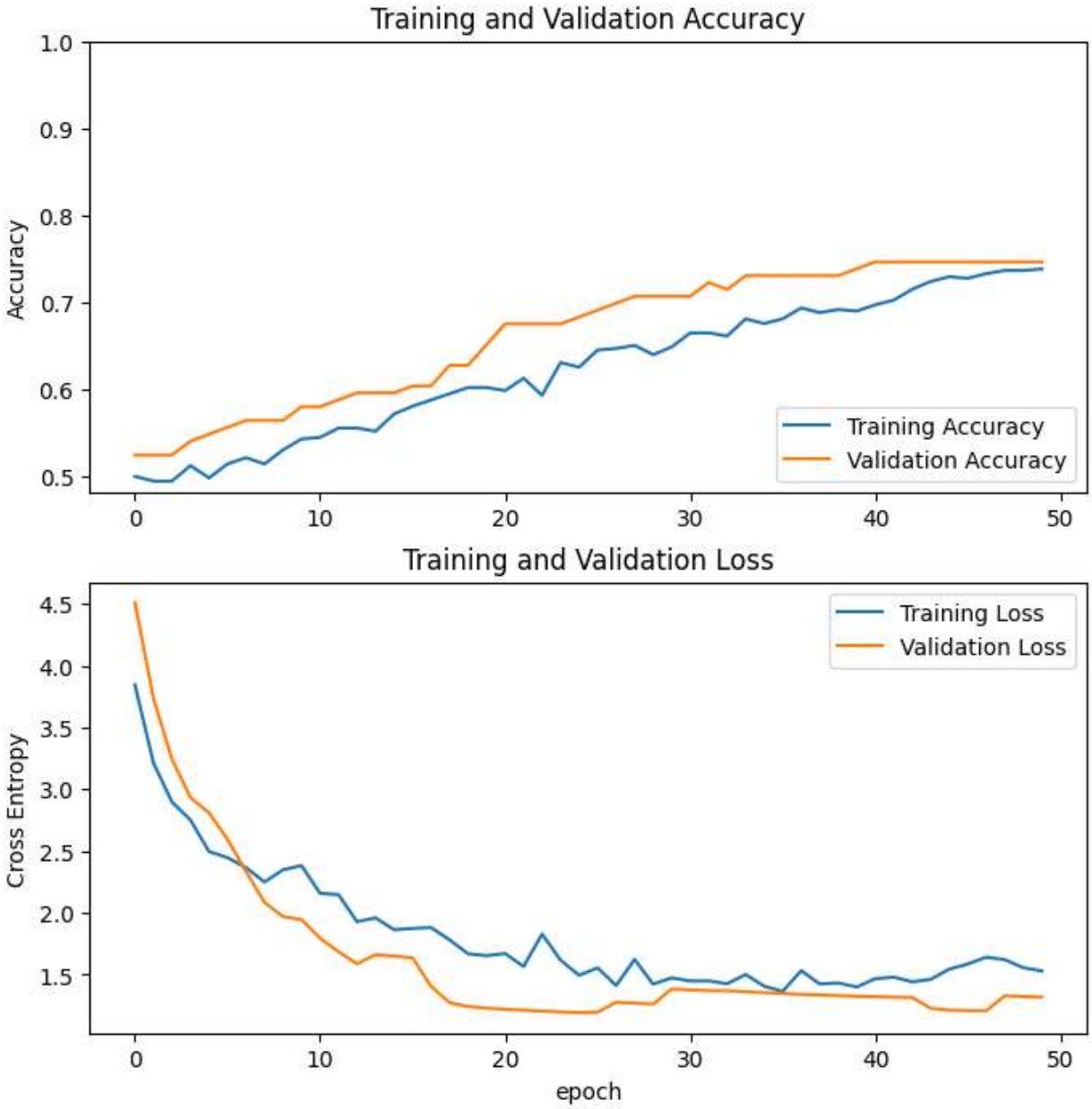
```
Epoch 1/50
18/18 [==============================] - 11s 515ms/step - loss: 3.8449 - accuracy: 0.
4991 - val_loss: 4.5109 - val_accuracy: 0.5238
Epoch 2/50
18/18 [==============================] - 5s 257ms/step - loss: 3.2140 - accuracy: 0.4
937 - val_loss: 3.7360 - val_accuracy: 0.5238
Epoch 3/50
18/18 [==============================] - 5s 277ms/step - loss: 2.8968 - accuracy: 0.4
937 - val_loss: 3.2440 - val_accuracy: 0.5238
Epoch 4/50
18/18 [==============================] - 5s 266ms/step - loss: 2.7518 - accuracy: 0.5
117 - val_loss: 2.9327 - val_accuracy: 0.5397
Epoch 5/50
18/18 [==============================] - 5s 264ms/step - loss: 2.4967 - accuracy: 0.4
973 - val_loss: 2.8090 - val_accuracy: 0.5476
Epoch 6/50
18/18 [==============================] - 5s 271ms/step - loss: 2.4465 - accuracy: 0.5
135 - val_loss: 2.5964 - val_accuracy: 0.5556
Epoch 7/50
18/18 [==============================] - 5s 272ms/step - loss: 2.3666 - accuracy: 0.5
206 - val_loss: 2.3351 - val_accuracy: 0.5635
Epoch 8/50
18/18 [==============================] - 5s 263ms/step - loss: 2.2494 - accuracy: 0.5
135 - val_loss: 2.0843 - val_accuracy: 0.5635
Epoch 9/50
18/18 [==============================] - 5s 256ms/step - loss: 2.3487 - accuracy: 0.5
296 - val_loss: 1.9697 - val_accuracy: 0.5635
Epoch 10/50
18/18 [==============================] - 5s 265ms/step - loss: 2.3821 - accuracy: 0.5
422 - val_loss: 1.9450 - val_accuracy: 0.5794
Epoch 11/50
18/18 [==============================] - 5s 263ms/step - loss: 2.1582 - accuracy: 0.5
440 - val_loss: 1.7945 - val_accuracy: 0.5794
Epoch 12/50
18/18 [==============================] - 5s 268ms/step - loss: 2.1474 - accuracy: 0.5
548 - val_loss: 1.6852 - val_accuracy: 0.5873
Epoch 13/50
18/18 [==============================] - 5s 273ms/step - loss: 1.9275 - accuracy: 0.5
548 - val_loss: 1.5876 - val_accuracy: 0.5952
Epoch 14/50
18/18 [==============================] - 5s 268ms/step - loss: 1.9596 - accuracy: 0.5
512 - val_loss: 1.6603 - val_accuracy: 0.5952
Epoch 15/50
18/18 [==============================] - 5s 266ms/step - loss: 1.8634 - accuracy: 0.5
709 - val_loss: 1.6504 - val_accuracy: 0.5952
Epoch 16/50
18/18 [==============================] - 5s 277ms/step - loss: 1.8736 - accuracy: 0.5
799 - val_loss: 1.6350 - val_accuracy: 0.6032
Epoch 17/50
18/18 [==============================] - 5s 267ms/step - loss: 1.8807 - accuracy: 0.5
871 - val_loss: 1.4055 - val_accuracy: 0.6032
Epoch 18/50
18/18 [==============================] - 5s 270ms/step - loss: 1.7818 - accuracy: 0.5
943 - val_loss: 1.2725 - val_accuracy: 0.6270
Epoch 19/50
18/18 [==============================] - 5s 269ms/step - loss: 1.6684 - accuracy: 0.6
014 - val_loss: 1.2418 - val_accuracy: 0.6270
Epoch 20/50
18/18 [==============================] - 5s 264ms/step - loss: 1.6530 - accuracy: 0.6
014 - val_loss: 1.2286 - val_accuracy: 0.6508
```

```
Epoch 21/50
18/18 [==============================] - 5s 258ms/step - loss: 1.6706 - accuracy: 0.5
978 - val_loss: 1.2190 - val_accuracy: 0.6746
Epoch 22/50
18/18 [==============================] - 5s 265ms/step - loss: 1.5656 - accuracy: 0.6
122 - val_loss: 1.2114 - val_accuracy: 0.6746
Epoch 23/50
18/18 [==============================] - 5s 259ms/step - loss: 1.8282 - accuracy: 0.5
925 - val_loss: 1.2042 - val_accuracy: 0.6746
Epoch 24/50
18/18 [==============================] - 5s 256ms/step - loss: 1.6159 - accuracy: 0.6
302 - val_loss: 1.1975 - val_accuracy: 0.6746
Epoch 25/50
18/18 [==============================] - 5s 287ms/step - loss: 1.4943 - accuracy: 0.6
248 - val_loss: 1.1924 - val_accuracy: 0.6825
Epoch 26/50
18/18 [==============================] - 5s 263ms/step - loss: 1.5544 - accuracy: 0.6
445 - val_loss: 1.1956 - val_accuracy: 0.6905
Epoch 27/50
18/18 [==============================] - 5s 274ms/step - loss: 1.4114 - accuracy: 0.6
463 - val_loss: 1.2760 - val_accuracy: 0.6984
Epoch 28/50
18/18 [==============================] - 5s 263ms/step - loss: 1.6233 - accuracy: 0.6
499 - val_loss: 1.2690 - val_accuracy: 0.7063
Epoch 29/50
18/18 [==============================] - 5s 268ms/step - loss: 1.4226 - accuracy: 0.6
391 - val_loss: 1.2611 - val_accuracy: 0.7063
Epoch 30/50
18/18 [==============================] - 5s 277ms/step - loss: 1.4738 - accuracy: 0.6
481 - val_loss: 1.3829 - val_accuracy: 0.7063
Epoch 31/50
18/18 [==============================] - 5s 263ms/step - loss: 1.4483 - accuracy: 0.6
643 - val_loss: 1.3763 - val_accuracy: 0.7063
Epoch 32/50
18/18 [==============================] - 5s 260ms/step - loss: 1.4491 - accuracy: 0.6
643 - val_loss: 1.3699 - val_accuracy: 0.7222
Epoch 33/50
18/18 [==============================] - 5s 270ms/step - loss: 1.4255 - accuracy: 0.6
607 - val_loss: 1.3684 - val_accuracy: 0.7143
Epoch 34/50
18/18 [==============================] - 5s 263ms/step - loss: 1.5010 - accuracy: 0.6
804 - val_loss: 1.3612 - val_accuracy: 0.7302
Epoch 35/50
18/18 [==============================] - 5s 269ms/step - loss: 1.4057 - accuracy: 0.6
750 - val_loss: 1.3541 - val_accuracy: 0.7302
Epoch 36/50
18/18 [==============================] - 5s 266ms/step - loss: 1.3644 - accuracy: 0.6
804 - val_loss: 1.3461 - val_accuracy: 0.7302
Epoch 37/50
18/18 [==============================] - 5s 265ms/step - loss: 1.5318 - accuracy: 0.6
930 - val_loss: 1.3397 - val_accuracy: 0.7302
Epoch 38/50
18/18 [==============================] - 5s 272ms/step - loss: 1.4241 - accuracy: 0.6
876 - val_loss: 1.3349 - val_accuracy: 0.7302
Epoch 39/50
18/18 [==============================] - 5s 268ms/step - loss: 1.4324 - accuracy: 0.6
912 - val_loss: 1.3294 - val_accuracy: 0.7302
Epoch 40/50
18/18 [==============================] - 5s 268ms/step - loss: 1.3993 - accuracy: 0.6
894 - val_loss: 1.3244 - val_accuracy: 0.7381
```

```
Epoch 41/50
18/18 [==============================] - 5s 272ms/step - loss: 1.4666 - accuracy: 0.6
966 - val_loss: 1.3214 - val_accuracy: 0.7460
Epoch 42/50
18/18 [==============================] - 5s 250ms/step - loss: 1.4791 - accuracy: 0.7
020 - val_loss: 1.3173 - val_accuracy: 0.7460
Epoch 43/50
18/18 [==============================] - 5s 295ms/step - loss: 1.4412 - accuracy: 0.7
145 - val_loss: 1.3146 - val_accuracy: 0.7460
Epoch 44/50
18/18 [==============================] - 5s 275ms/step - loss: 1.4616 - accuracy: 0.7
235 - val_loss: 1.2268 - val_accuracy: 0.7460
Epoch 45/50
18/18 [==============================] - 5s 250ms/step - loss: 1.5441 - accuracy: 0.7
289 - val_loss: 1.2118 - val_accuracy: 0.7460
Epoch 46/50
18/18 [==============================] - 5s 253ms/step - loss: 1.5854 - accuracy: 0.7
271 - val_loss: 1.2089 - val_accuracy: 0.7460
Epoch 47/50
18/18 [==============================] - 5s 260ms/step - loss: 1.6410 - accuracy: 0.7
325 - val_loss: 1.2072 - val_accuracy: 0.7460
Epoch 48/50
18/18 [==============================] - 5s 280ms/step - loss: 1.6212 - accuracy: 0.7
361 - val_loss: 1.3290 - val_accuracy: 0.7460
Epoch 49/50
18/18 [==============================] - 5s 277ms/step - loss: 1.5556 - accuracy: 0.7
361 - val_loss: 1.3227 - val_accuracy: 0.7460
Epoch 50/50
18/18 [==============================] - 5s 292ms/step - loss: 1.5297 - accuracy: 0.7
379 - val_loss: 1.3173 - val_accuracy: 0.7460
```

In [23]:

```
# Let's plot the results of the training

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Fine-tuning

Now that we've trained the classification head, let's train some of the top layers to greatly improve the performance of the model.

In [24]:
```
base_model.trainable = True
```

In [25]:
```
# Let's take a look to see how many layers are in the base model
print("Number of layers in the base model: ", len(base_model.layers))

# Fine-tune from this layer onwards
fine_tune_at = 18

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False
```

```
Number of layers in the base model:  22
```

In [26]:
```
model.compile(
    loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
    optimizer = tf.keras.optimizers.RMSprop(learning_rate=base_learning_rate/10),
    metrics=['accuracy']
)
```

In [27]:
```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 tf.__operators__.getitem (S  (None, 224, 224, 3)      0
 licingOpLambda)

 tf.nn.bias_add (TFOpLambda)  (None, 224, 224, 3)      0

 vgg19 (Functional)          (None, None, None, 512)   20024384

 global_average_pooling2d (G  (None, 512)              0
 lobalAveragePooling2D)

 dense (Dense)               (None, 2)                 1026

=================================================================
Total params: 20,025,410
Trainable params: 7,080,450
Non-trainable params: 12,944,960
_________________________________________________________________
```

In [28]:
```
len(model.trainable_variables)
```

Out[28]:
```
8
```

In [29]:
```
earlystopping = tf.keras.callbacks.EarlyStopping(
    monitor="val_loss",
    patience=5,
    restore_best_weights=True
)
```

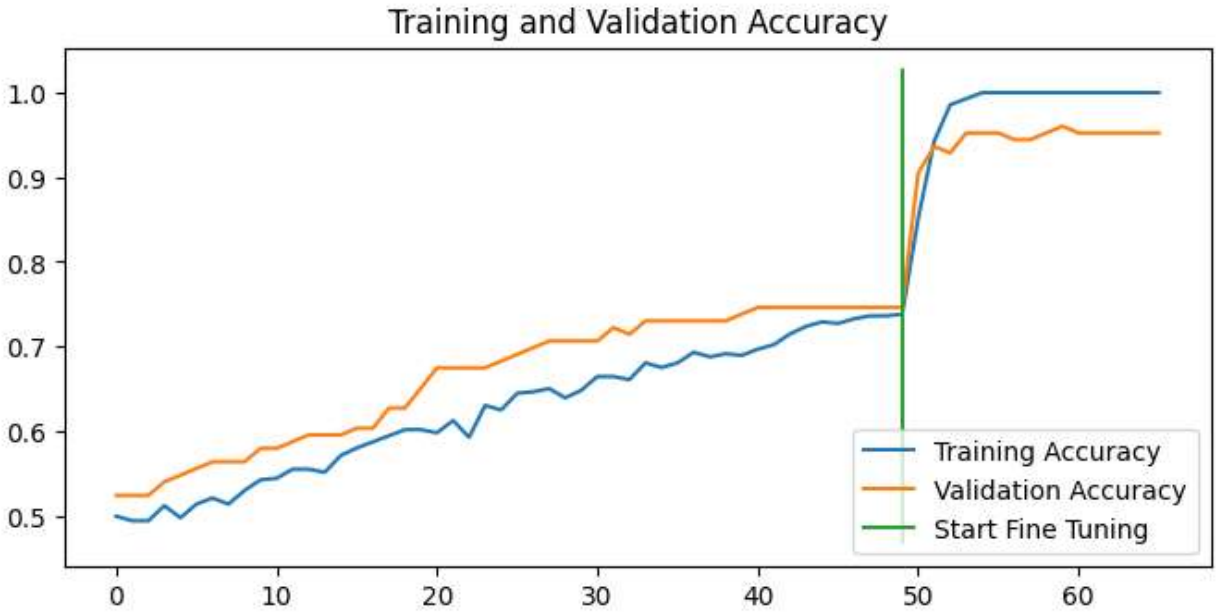
In [30]:
```
# Let's fine tune the model

fine_tune_epochs = 50
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model.fit(
    train_generator,
    epochs=total_epochs,
    initial_epoch=history.epoch[-1],
    validation_data=val_generator,
    callbacks=[earlystopping]
)
```

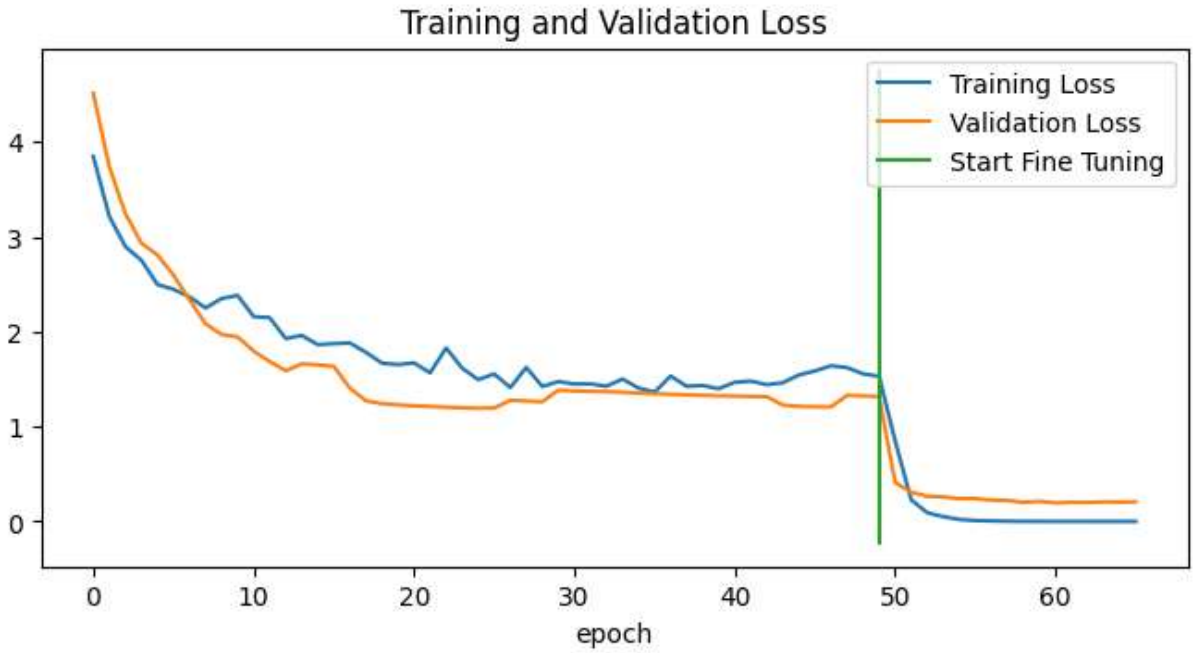
```
Epoch 50/100
18/18 [==============================] - 8s 316ms/step - loss: 0.8486 - accuracy: 0.8
510 - val_loss: 0.4111 - val_accuracy: 0.9048
Epoch 51/100
18/18 [==============================] - 5s 273ms/step - loss: 0.2286 - accuracy: 0.9
425 - val_loss: 0.3084 - val_accuracy: 0.9365
Epoch 52/100
18/18 [==============================] - 5s 299ms/step - loss: 0.0925 - accuracy: 0.9
856 - val_loss: 0.2653 - val_accuracy: 0.9286
Epoch 53/100
18/18 [==============================] - 5s 277ms/step - loss: 0.0507 - accuracy: 0.9
928 - val_loss: 0.2595 - val_accuracy: 0.9524
Epoch 54/100
18/18 [==============================] - 5s 279ms/step - loss: 0.0211 - accuracy: 1.0
000 - val_loss: 0.2401 - val_accuracy: 0.9524
Epoch 55/100
18/18 [==============================] - 5s 272ms/step - loss: 0.0099 - accuracy: 1.0
000 - val_loss: 0.2412 - val_accuracy: 0.9524
Epoch 56/100
18/18 [==============================] - 5s 264ms/step - loss: 0.0056 - accuracy: 1.0
000 - val_loss: 0.2255 - val_accuracy: 0.9444
Epoch 57/100
18/18 [==============================] - 5s 275ms/step - loss: 0.0029 - accuracy: 1.0
000 - val_loss: 0.2221 - val_accuracy: 0.9444
Epoch 58/100
18/18 [==============================] - 5s 280ms/step - loss: 0.0012 - accuracy: 1.0
000 - val_loss: 0.2012 - val_accuracy: 0.9524
Epoch 59/100
18/18 [==============================] - 5s 261ms/step - loss: 8.9268e-04 - accuracy:
1.0000 - val_loss: 0.2130 - val_accuracy: 0.9603
Epoch 60/100
18/18 [==============================] - 5s 301ms/step - loss: 5.3802e-04 - accuracy:
1.0000 - val_loss: 0.1956 - val_accuracy: 0.9524
Epoch 61/100
18/18 [==============================] - 5s 267ms/step - loss: 4.1647e-04 - accuracy:
1.0000 - val_loss: 0.2023 - val_accuracy: 0.9524
Epoch 62/100
18/18 [==============================] - 5s 255ms/step - loss: 3.6208e-04 - accuracy:
1.0000 - val_loss: 0.2008 - val_accuracy: 0.9524
Epoch 63/100
18/18 [==============================] - 5s 269ms/step - loss: 2.6654e-04 - accuracy:
1.0000 - val_loss: 0.2069 - val_accuracy: 0.9524
Epoch 64/100
18/18 [==============================] - 5s 271ms/step - loss: 2.0695e-04 - accuracy:
1.0000 - val_loss: 0.2041 - val_accuracy: 0.9524
Epoch 65/100
18/18 [==============================] - 5s 292ms/step - loss: 1.8368e-04 - accuracy:
1.0000 - val_loss: 0.2075 - val_accuracy: 0.9524
```

Wow! That really boosted performance!

In [31]:
```
# Update the accuracy and loss

acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']

loss += history_fine.history['loss']
val_loss += history_fine.history['val_loss']
```

In [32]:
```
# Plot the results!

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.plot([initial_epochs-1,initial_epochs-1],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.plot([initial_epochs-1,initial_epochs-1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```





Evaluate the final model

In [33]:
```
loss, acc = model.evaluate(test_generator, verbose=False)
print(f"Validation accuracy: {acc*100}%")
```

```
Validation accuracy: 92.85714030265808%
```