In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

In [1]:
```
from google.colab import files
files.upload()  # Upload kaggle.json when prompted
```

Choose Files No file selected Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
```

Out[1]:
```
{'kaggle.json': b'{"username":"hamidrezaamin","key":"d15526e38f947f4469926effee3832aa"}'}
```

In [2]:
```
import os

# Create the .kaggle directory
os.makedirs("/root/.kaggle", exist_ok=True)

# Rename and move the file to the proper location
!mv "kaggle.json" /root/.kaggle/kaggle.json

# Set file permissions
!chmod 600 /root/.kaggle/kaggle.json
```

In [3]:
```
# Install the Kaggle CLI
!pip install -q kaggle

# Download the competition dataset
!kaggle competitions download -c dogs-vs-cats
```

```
Downloading dogs-vs-cats.zip to /content
 96% 779M/812M [00:05<00:00, 231MB/s]
100% 812M/812M [00:05<00:00, 163MB/s]
```

In [4]:
```
# Unzip the main dataset
!unzip dogs-vs-cats.zip -d /content
```

```
Archive:  dogs-vs-cats.zip
  inflating: /content/sampleSubmission.csv
  inflating: /content/test1.zip
  inflating: /content/train.zip
```

In [5]:
```
# Unzip the inner train and test archives
!unzip -q /content/train.zip -d /content/train
!unzip -q /content/test1.zip -d /content/test
```

In [6]:
```
import shutil
# Create the original_dataset_dir
original_dataset_dir = '/content/original_dataset_dir'
os.makedirs(original_dataset_dir, exist_ok=True)

# Move the train and test folders
shutil.move('/content/train', os.path.join(original_dataset_dir, 'train'))
shutil.move('/content/test', os.path.join(original_dataset_dir, 'test'))
```

Out[6]:
```
'/content/original_dataset_dir/test'
```

In [7]:
```
# Fix train folder
nested_train = '/content/original_dataset_dir/train/train'
correct_train = '/content/original_dataset_dir/train'

if os.path.exists(nested_train):
    for filename in os.listdir(nested_train):
        shutil.move(os.path.join(nested_train, filename), correct_train)
    os.rmdir(nested_train)  # remove the empty 'train' folder

# Fix test folder
nested_test = '/content/original_dataset_dir/test/test1'
correct_test = '/content/original_dataset_dir/test'

if os.path.exists(nested_test):
    for filename in os.listdir(nested_test):
        shutil.move(os.path.join(nested_test, filename), correct_test)
    os.rmdir(nested_test)  # remove the empty 'test1' folder
```

In [8]:
```
# check the number of cats and dogs in train folder
train_dir = '/content/original_dataset_dir/train'

cat_images = [f for f in os.listdir(train_dir) if f.startswith('cat')]
dog_images = [f for f in os.listdir(train_dir) if f.startswith('dog')]
```

```python
print(f"Number of cat images: {len(cat_images)}")
print(f"Number of dog images: {len(dog_images)}")
```

```
Number of cat images: 12500
Number of dog images: 12500
```

In [9]:

```python
# Create the base_dir
base_dir = '/content/base_dir'
os.makedirs(base_dir, exist_ok=True)
```

In [10]:

```python
# Define the base directory
base_dir = '/content/base_dir'
os.makedirs(base_dir, exist_ok=True)  # Create base_dir if it doesn't exist

# Define the subfolders to create
subfolders = ['train', 'test', 'validation']

# Loop and create each subfolder inside base_dir
for folder in subfolders:
    path = os.path.join(base_dir, folder)
    os.makedirs(path, exist_ok=True)
    print(f"Created: {path}")
```

```
Created: /content/base_dir/train
Created: /content/base_dir/test
Created: /content/base_dir/validation
```

In [11]:

```python
# Define base directory
base_dir = '/content/base_dir'
os.makedirs(base_dir, exist_ok=True)

# Top-level subfolders
top_subfolders = ['train', 'test', 'validation']

# Sub-subfolders to create inside each top-level folder
class_folders = ['cat', 'dog']

# Create the full directory structure
for sub in top_subfolders:
    sub_path = os.path.join(base_dir, sub)
    os.makedirs(sub_path, exist_ok=True)

    for class_folder in class_folders:
        class_path = os.path.join(sub_path, class_folder)
        os.makedirs(class_path, exist_ok=True)
        print(f"Created: {class_path}")
```

```
Created: /content/base_dir/train/cat
Created: /content/base_dir/train/dog
Created: /content/base_dir/test/cat
Created: /content/base_dir/test/dog
Created: /content/base_dir/validation/cat
Created: /content/base_dir/validation/dog
```

In [12]:

```python
# Source and target directories
source_dir = '/content/original_dataset_dir/train'
target_base_dir = '/content/base_dir/train'

# List all files in the source directory
all_filenames = os.listdir(source_dir)

# Filter out cat and dog images
cat_filenames = [f for f in all_filenames if f.startswith('cat')]
dog_filenames = [f for f in all_filenames if f.startswith('dog')]

# Sort and select the first 1000 of each
cat_filenames = sorted(cat_filenames)[:1000]
dog_filenames = sorted(dog_filenames)[:1000]

# Copy cat images
for fname in cat_filenames:
    src = os.path.join(source_dir, fname)
    dst = os.path.join(target_base_dir, 'cat', fname)
    shutil.copyfile(src, dst)

# Copy dog images
for fname in dog_filenames:
    src = os.path.join(source_dir, fname)
    dst = os.path.join(target_base_dir, 'dog', fname)
    shutil.copyfile(src, dst)

print("Copied 1000 cat and 1000 dog images to base_dir/train/")
```

```
Copied 1000 cat and 1000 dog images to base_dir/train/
```

In [13]:

```
# Define source and destination paths
source_dir = '/content/original_dataset_dir/train'
validation_dir = '/content/base_dir/validation'

# List all files in the source directory
all_filenames = os.listdir(source_dir)

# Filter cat and dog images
cat_filenames = sorted([f for f in all_filenames if f.startswith('cat')])
dog_filenames = sorted([f for f in all_filenames if f.startswith('dog')])

# Select images from index 1001 to 2000 (i.e., next 1000 images)
cat_val_filenames = cat_filenames[1000:2000]
dog_val_filenames = dog_filenames[1000:2000]

# Copy cat validation images
for fname in cat_val_filenames:
    src = os.path.join(source_dir, fname)
    dst = os.path.join(validation_dir, 'cat', fname)
    shutil.copyfile(src, dst)

# Copy dog validation images
for fname in dog_val_filenames:
    src = os.path.join(source_dir, fname)
    dst = os.path.join(validation_dir, 'dog', fname)
    shutil.copyfile(src, dst)

print("Copied 1000 cat and 1000 dog validation images to base_dir/validation/")
```

```
Copied 1000 cat and 1000 dog validation images to base_dir/validation/
```

In [15]:

```
import os
import shutil

# Define source and target directories
source_dir = '/content/original_dataset_dir/test'
target_dir = '/content/base_dir/test'

# List and sort image files (to ensure consistent order like 1.jpg, 2.jpg, ...)
all_test_images = sorted(os.listdir(source_dir))[:1000]

# Copy first 1000 images
for fname in all_test_images:
    src = os.path.join(source_dir, fname)
    dst = os.path.join(target_dir, fname)
    shutil.copyfile(src, dst)

print("Copied first 1000 test images to base_dir/test/")
```

```
Copied first 1000 test images to base_dir/test/
```

(model architecture)

Since we are attacking a binary classification problem, we are ending the network with a single unit (a Dense layer of size 1) and a sigmoid activation. This unit will encode the probability that the network is looking at one class or the other.

In [16]:

```
from keras import layers, models


# Initialize a sequential model
model = models.Sequential()

# First convolutional block
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))  # Input layer for 150x150 RGB images
model.add(layers.MaxPooling2D((2, 2)))               # Downsampling with 2x2 max pooling

# Second convolutional block
model.add(layers.Conv2D(64, (3, 3), activation='relu'))  # 64 filters of 3x3
model.add(layers.MaxPooling2D((2, 2)))

# Third convolutional block
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# Fourth convolutional block
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# Flatten the 3D feature maps into 1D feature vector
model.add(layers.Flatten())
```

```
# Fully connected (dense) layer
model.add(layers.Dense(512, activation='relu'))  # High-capacity layer for learning complex patterns

# Output layer for binary classification
model.add(layers.Dense(1, activation='sigmoid'))  # Sigmoid gives a probability between 0 and 1
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pas
s an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)`
object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [17]:
```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_2 (Conv2D)	(None, 34, 34, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 17, 17, 128)	0
conv2d_3 (Conv2D)	(None, 15, 15, 128)	147,584
max_pooling2d_3 (MaxPooling2D)	(None, 7, 7, 128)	0
flatten (Flatten)	(None, 6272)	0
dense (Dense)	(None, 512)	3,211,776
dense_1 (Dense)	(None, 1)	513

Total params: 3,453,121 (13.17 MB)

Trainable params: 3,453,121 (13.17 MB)

Non-trainable params: 0 (0.00 B)

For our compilation step, we'll go with the binary_crossentropy optimizer as usual. Since we ended our network with a single sigmoid unit, we will use binary crossentropy as our loss (as a reminder, check out the table in Chapter 4, section 5 for a cheatsheet on what loss function to use in various situations).

In [18]:
```
from keras import optimizers

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(learning_rate=1e-4),
              metrics=['acc'])
```

(Data preprocessing)

Read the picture files. Decode the JPEG content to RGB grids of pixels. Convert these into floating point tensors. Rescale the pixel values (between 0 and 255) to the [0, 1] interval (as you know, neural networks prefer to deal with small input values). It may seem a bit daunting, but thankfully Keras has utilities to take care of these steps automatically. Keras has a module with image processing helper tools, located at keras.preprocessing.image. In particular, it contains the class ImageDataGenerator which allows to quickly set up Python generators that can automatically turn image files on disk into batches of pre-processed tensors. This is what we will use here.

In [19]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator


# Rescale pixel values to [0, 1] range
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

# Point to the correct directories
train_dir = '/content/base_dir/train'
validation_dir = '/content/base_dir/validation'

# Create the training data generator
train_generator = train_datagen.flow_from_directory(
        train_dir,              # This directory contains subfolders: 'cat', 'dog'
        target_size=(150, 150),  # Resize all images to 150x150
        batch_size=20,          # Number of images per batch
        class_mode='binary')    # Use binary labels (0 or 1)
```

```python
# Create the validation data generator
validation_generator = test_datagen.flow_from_directory(
        validation_dir,           # This directory also contains 'cat' and 'dog' folders
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')
```

```
Found 2000 images belonging to 2 classes.
Found 2000 images belonging to 2 classes.
```

Let's take a look at the output of one of these generators: it yields batches of 150x150 RGB images (shape (20, 150, 150, 3)) and binary labels (shape (20,)). 20 is the number of samples in each batch (the batch size). Note that the generator yields these batches indefinitely: it just loops endlessly over the images present in the target folder. For this reason, we need to break the iteration loop at some point.

In [20]:

```python
# The generator yields batches of images and labels endlessly.
# We use a loop to get just ONE batch and break immediately after.

for data_batch, labels_batch in train_generator:
    print('data batch shape:', data_batch.shape)     # Shape: (20, 150, 150, 3)
    print('labels batch shape:', labels_batch.shape) # Shape: (20,)
    break  # Stop after the first batch
```

```
data batch shape: (20, 150, 150, 3)
labels batch shape: (20,)
```

Training the CNN Model Using Generators with .fit()

Let's fit our model to the data using the generator. We do it using the fit_generator method, the equivalent of fit for data generators like ours. It expects as first argument a Python generator that will yield batches of inputs and targets indefinitely, like ours does. Because the data is being generated endlessly, the generator needs to know, for example, how many samples to draw from the generator before declaring an epoch over. This is the role of the steps_per_epoch argument: after having drawn steps_per_epoch batches from the generator, i.e., after having run for steps_per_epoch gradient descent steps, the fitting process will go to the next epoch.

In our case, batches are 20-sample large, so it will take 1,125 batches until we see our target of 22,500 samples.

When using fit_generator, one may pass a validation_data argument, much like with the fit method. Importantly, this argument is allowed to be a data generator itself, but it could be a tuple of Numpy arrays as well. If you pass a generator as validation_data, then this generator is expected to yield batches of validation data endlessly, and thus you should also specify the validation_steps argument, which tells the process how many batches to draw from the validation generator for evaluation.

In our case, since we have 2,500 validation samples, and the batch size is 20, we'll use 125 validation steps.

In [21]:

```python
history = model.fit(
      train_generator,                # Training generator (yields batches of 20 images)
      epochs=30,                      # Train the model for 30 full passes (epochs) through the dataset
      validation_data=validation_generator  # Validation generator used after each epoch
)
```

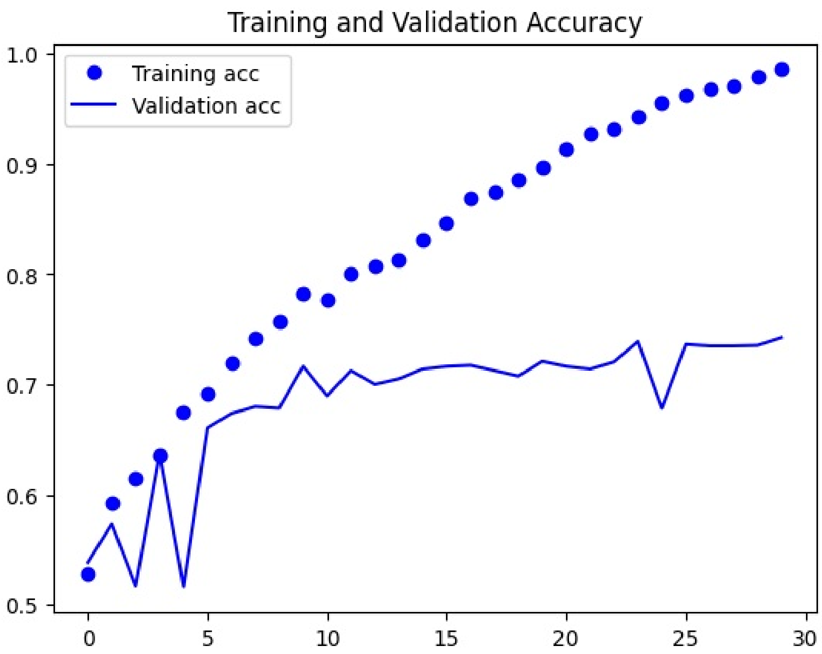
```
Epoch 1/30
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
```

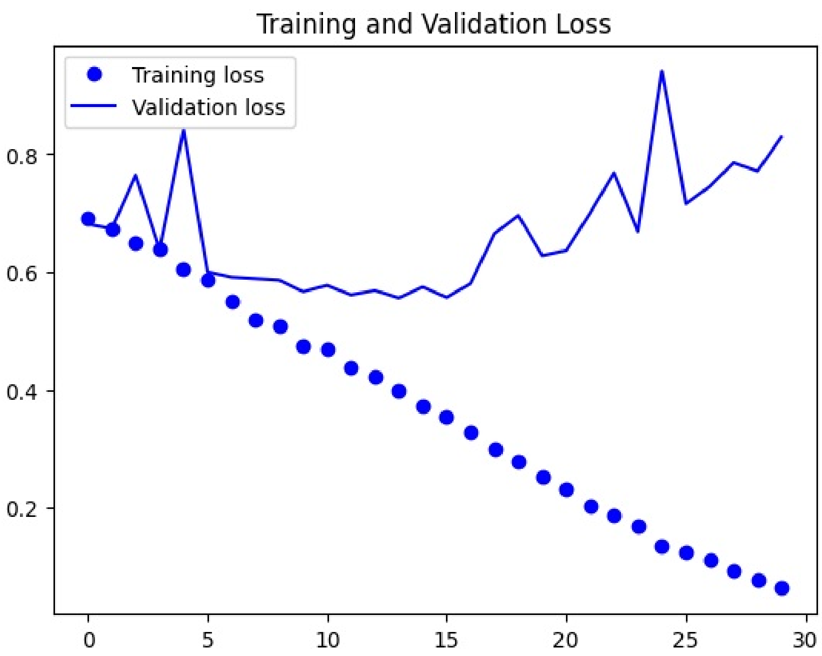
```
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 71ms/step - acc: 0.5066 - loss: 0.6918 - val_acc: 0.5385 - val_loss: 0.6819
Epoch 2/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 16s 67ms/step - acc: 0.5913 - loss: 0.6752 - val_acc: 0.5735 - val_loss: 0.6735
Epoch 3/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 59ms/step - acc: 0.6046 - loss: 0.6561 - val_acc: 0.5170 - val_loss: 0.7644
Epoch 4/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 98ms/step - acc: 0.6251 - loss: 0.6501 - val_acc: 0.6365 - val_loss: 0.6387
Epoch 5/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - acc: 0.6709 - loss: 0.6119 - val_acc: 0.5165 - val_loss: 0.8420
Epoch 6/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 68ms/step - acc: 0.6784 - loss: 0.6110 - val_acc: 0.6605 - val_loss: 0.6005
Epoch 7/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 83ms/step - acc: 0.6992 - loss: 0.5602 - val_acc: 0.6735 - val_loss: 0.5914
Epoch 8/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 82ms/step - acc: 0.7411 - loss: 0.5232 - val_acc: 0.6800 - val_loss: 0.5889
Epoch 9/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 90ms/step - acc: 0.7516 - loss: 0.5089 - val_acc: 0.6785 - val_loss: 0.5864
Epoch 10/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 82ms/step - acc: 0.7855 - loss: 0.4767 - val_acc: 0.7165 - val_loss: 0.5669
Epoch 11/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 59ms/step - acc: 0.7940 - loss: 0.4566 - val_acc: 0.6895 - val_loss: 0.5777
Epoch 12/30

100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 93ms/step - acc: 0.8011 - loss: 0.4322 - val_acc: 0.7125 - val_loss: 0.5613
Epoch 13/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 83ms/step - acc: 0.7988 - loss: 0.4311 - val_acc: 0.7000 - val_loss: 0.5689
Epoch 14/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 94ms/step - acc: 0.8176 - loss: 0.3963 - val_acc: 0.7050 - val_loss: 0.5561
Epoch 15/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 66ms/step - acc: 0.8315 - loss: 0.3659 - val_acc: 0.7140 - val_loss: 0.5749
Epoch 16/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - acc: 0.8467 - loss: 0.3512 - val_acc: 0.7165 - val_loss: 0.5570
Epoch 17/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 83ms/step - acc: 0.8749 - loss: 0.3248 - val_acc: 0.7175 - val_loss: 0.5806
Epoch 18/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 58ms/step - acc: 0.8725 - loss: 0.3048 - val_acc: 0.7125 - val_loss: 0.6654
Epoch 19/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 67ms/step - acc: 0.8924 - loss: 0.2603 - val_acc: 0.7075 - val_loss: 0.6960
Epoch 20/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 59ms/step - acc: 0.8991 - loss: 0.2594 - val_acc: 0.7210 - val_loss: 0.6280
Epoch 21/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 68ms/step - acc: 0.9145 - loss: 0.2371 - val_acc: 0.7165 - val_loss: 0.6367
Epoch 22/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - acc: 0.9368 - loss: 0.1889 - val_acc: 0.7140 - val_loss: 0.7002
Epoch 23/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 68ms/step - acc: 0.9316 - loss: 0.1939 - val_acc: 0.7205 - val_loss: 0.7684
Epoch 24/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - acc: 0.9439 - loss: 0.1677 - val_acc: 0.7390 - val_loss: 0.6687
Epoch 25/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 84ms/step - acc: 0.9610 - loss: 0.1388 - val_acc: 0.6785 - val_loss: 0.9409
Epoch 26/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 82ms/step - acc: 0.9562 - loss: 0.1362 - val_acc: 0.7365 - val_loss: 0.7159
Epoch 27/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 57ms/step - acc: 0.9640 - loss: 0.1186 - val_acc: 0.7350 - val_loss: 0.7452
Epoch 28/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 65ms/step - acc: 0.9750 - loss: 0.0874 - val_acc: 0.7350 - val_loss: 0.7856
Epoch 29/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 60ms/step - acc: 0.9808 - loss: 0.0675 - val_acc: 0.7355 - val_loss: 0.7713
Epoch 30/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 82ms/step - acc: 0.9897 - loss: 0.0579 - val_acc: 0.7425 - val_loss: 0.8293
```

In [22]:

```python
import matplotlib.pyplot as plt

# Retrieve accuracy and loss values from training history
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

# Define the number of epochs (x-axis)
epochs = range(len(acc))

# Plot Training & Validation Accuracy
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and Validation Accuracy')
plt.legend()

# Start a new figure
plt.figure()

# Plot Training & Validation Loss
plt.plot(epochs, loss, 'bo', label='Training loss')
```

```python
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and Validation Loss')
plt.legend()

# Display both plots
plt.show()
```





These plots are characteristic of overfitting. Our training accuracy increases linearly over time, until it reaches nearly 100%, while our validation accuracy stalls at 86–88%. Our validation loss reaches its minimum after about 10–12 epochs, then starts increasing, while the training loss keeps decreasing linearly until it reaches nearly 0.

Because we only have relatively few training samples (22,500), overfitting is going to be our number one concern. Let's train our network using data augmentation and dropout:

CNN Model with Dropout for Overfitting Prevention :

In [23]:

```python
from tensorflow.keras import layers, models, optimizers

# Initialize the model
model = models.Sequential()

# First convolutional block
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))  # Input layer for 150x150 RGB images
model.add(layers.MaxPooling2D((2, 2)))               # Downsamples to reduce spatial size

# Second convolutional block
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

```python
model.add(layers.MaxPooling2D((2, 2)))

# Third convolutional block
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# Fourth convolutional block
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# Flatten feature maps into a 1D vector
model.add(layers.Flatten())

#  Dropout layer to reduce overfitting
# Randomly drops 50% of the neurons during training
model.add(layers.Dropout(0.5))

# Fully connected dense layer
model.add(layers.Dense(512, activation='relu'))

# Output layer for binary classification (cat = 0, dog = 1)
model.add(layers.Dense(1, activation='sigmoid'))

# Compile the model with binary crossentropy and RMSprop optimizer
model.compile(
    loss='binary_crossentropy',
    optimizer=optimizers.RMSprop(learning_rate=1e-4),  # use learning_rate (not lr)
    metrics=['acc']
)

# Train the model using the image data generators
history = model.fit(
    train_generator,             # Generator for training data
    epochs=100,                  # Train for 100 epochs
    validation_data=validation_generator  # Generator for validation data
)
```

```
Epoch 1/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 85ms/step - acc: 0.4826 - loss: 0.6967 - val_acc: 0.6090 - val_loss: 0.6806
Epoch 2/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - acc: 0.5739 - loss: 0.6679 - val_acc: 0.6100 - val_loss: 0.6556
Epoch 3/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 66ms/step - acc: 0.6137 - loss: 0.6491 - val_acc: 0.6275 - val_loss: 0.6427
Epoch 4/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 59ms/step - acc: 0.6246 - loss: 0.6391 - val_acc: 0.6305 - val_loss: 0.6336
Epoch 5/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 82ms/step - acc: 0.6559 - loss: 0.6152 - val_acc: 0.6435 - val_loss: 0.6242
Epoch 6/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - acc: 0.6566 - loss: 0.6145 - val_acc: 0.6305 - val_loss: 0.6433
Epoch 7/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 67ms/step - acc: 0.6874 - loss: 0.5886 - val_acc: 0.6565 - val_loss: 0.6051
Epoch 8/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - acc: 0.6882 - loss: 0.5825 - val_acc: 0.6845 - val_loss: 0.5818
Epoch 9/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 67ms/step - acc: 0.7156 - loss: 0.5443 - val_acc: 0.7110 - val_loss: 0.5640
Epoch 10/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 59ms/step - acc: 0.7289 - loss: 0.5328 - val_acc: 0.6680 - val_loss: 0.6305
Epoch 11/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 63ms/step - acc: 0.7165 - loss: 0.5197 - val_acc: 0.7135 - val_loss: 0.5626
Epoch 12/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 69ms/step - acc: 0.7656 - loss: 0.4838 - val_acc: 0.7165 - val_loss: 0.5623
Epoch 13/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 12s 89ms/step - acc: 0.7785 - loss: 0.4806 - val_acc: 0.7185 - val_loss: 0.5519
Epoch 14/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 83ms/step - acc: 0.7562 - loss: 0.4811 - val_acc: 0.7395 - val_loss: 0.5387
Epoch 15/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - acc: 0.8035 - loss: 0.4342 - val_acc: 0.7360 - val_loss: 0.5494
Epoch 16/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 66ms/step - acc: 0.7966 - loss: 0.4410 - val_acc: 0.7435 - val_loss: 0.5282
Epoch 17/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 59ms/step - acc: 0.8040 - loss: 0.4331 - val_acc: 0.7315 - val_loss: 0.5321
Epoch 18/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 66ms/step - acc: 0.7956 - loss: 0.4291 - val_acc: 0.7455 - val_loss: 0.5270
Epoch 19/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 60ms/step - acc: 0.8115 - loss: 0.4091 - val_acc: 0.7455 - val_loss: 0.5460
Epoch 20/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 67ms/step - acc: 0.7954 - loss: 0.4245 - val_acc: 0.7465 - val_loss: 0.5242
Epoch 21/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - acc: 0.8353 - loss: 0.3743 - val_acc: 0.7510 - val_loss: 0.5185
Epoch 22/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 67ms/step - acc: 0.8410 - loss: 0.3588 - val_acc: 0.7470 - val_loss: 0.5268
Epoch 23/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 60ms/step - acc: 0.8495 - loss: 0.3582 - val_acc: 0.7475 - val_loss: 0.5313
Epoch 24/100
```

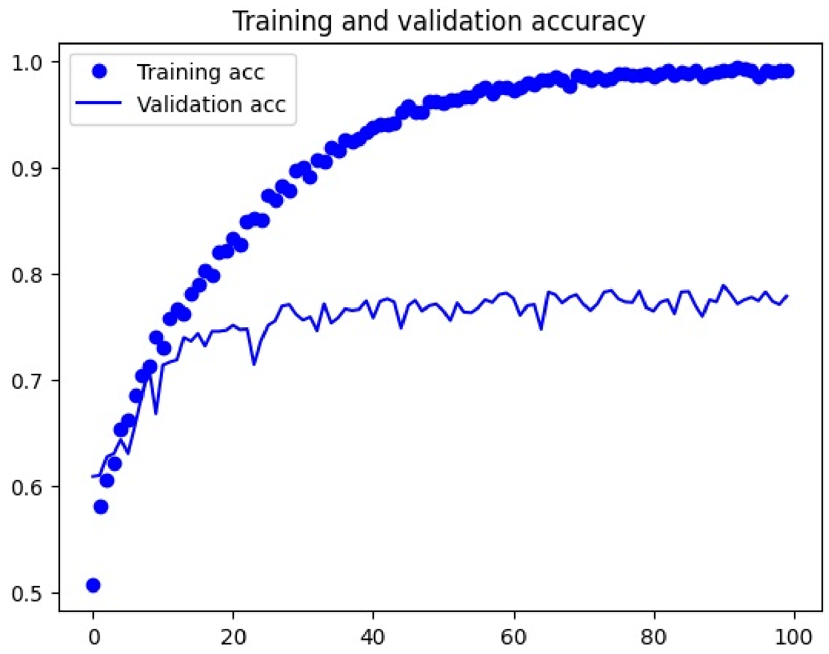
```
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 69ms/step - acc: 0.8606 - loss: 0.3396 - val_acc: 0.7140 - val_loss: 0.6213
Epoch 25/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - acc: 0.8457 - loss: 0.3258 - val_acc: 0.7365 - val_loss: 0.5770
Epoch 26/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 82ms/step - acc: 0.8794 - loss: 0.3128 - val_acc: 0.7510 - val_loss: 0.5308
Epoch 27/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 59ms/step - acc: 0.8606 - loss: 0.3140 - val_acc: 0.7550 - val_loss: 0.5309
Epoch 28/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 68ms/step - acc: 0.8857 - loss: 0.2698 - val_acc: 0.7690 - val_loss: 0.5275
Epoch 29/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - acc: 0.8681 - loss: 0.2983 - val_acc: 0.7705 - val_loss: 0.5279
Epoch 30/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 83ms/step - acc: 0.8975 - loss: 0.2676 - val_acc: 0.7615 - val_loss: 0.5377
Epoch 31/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 83ms/step - acc: 0.9036 - loss: 0.2478 - val_acc: 0.7560 - val_loss: 0.5526
Epoch 32/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 92ms/step - acc: 0.8945 - loss: 0.2501 - val_acc: 0.7590 - val_loss: 0.5298
Epoch 33/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 62ms/step - acc: 0.9117 - loss: 0.2372 - val_acc: 0.7460 - val_loss: 0.5874
Epoch 34/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 62ms/step - acc: 0.9160 - loss: 0.2269 - val_acc: 0.7710 - val_loss: 0.5343
Epoch 35/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 60ms/step - acc: 0.9247 - loss: 0.2105 - val_acc: 0.7530 - val_loss: 0.6370
Epoch 36/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 59ms/step - acc: 0.9130 - loss: 0.2076 - val_acc: 0.7580 - val_loss: 0.6064
Epoch 37/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 66ms/step - acc: 0.9263 - loss: 0.1841 - val_acc: 0.7665 - val_loss: 0.5839
Epoch 38/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 82ms/step - acc: 0.9326 - loss: 0.1919 - val_acc: 0.7650 - val_loss: 0.5907
Epoch 39/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 10s 99ms/step - acc: 0.9327 - loss: 0.1684 - val_acc: 0.7660 - val_loss: 0.5997
Epoch 40/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 59ms/step - acc: 0.9386 - loss: 0.1689 - val_acc: 0.7740 - val_loss: 0.5830
Epoch 41/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 66ms/step - acc: 0.9355 - loss: 0.1749 - val_acc: 0.7580 - val_loss: 0.6317
Epoch 42/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 82ms/step - acc: 0.9415 - loss: 0.1512 - val_acc: 0.7735 - val_loss: 0.5871
Epoch 43/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 87ms/step - acc: 0.9372 - loss: 0.1470 - val_acc: 0.7760 - val_loss: 0.5962
Epoch 44/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 83ms/step - acc: 0.9409 - loss: 0.1543 - val_acc: 0.7730 - val_loss: 0.6115
Epoch 45/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 57ms/step - acc: 0.9584 - loss: 0.1295 - val_acc: 0.7485 - val_loss: 0.6681
Epoch 46/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 67ms/step - acc: 0.9636 - loss: 0.1147 - val_acc: 0.7695 - val_loss: 0.6263
Epoch 47/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 59ms/step - acc: 0.9607 - loss: 0.1057 - val_acc: 0.7745 - val_loss: 0.6379
Epoch 48/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 68ms/step - acc: 0.9542 - loss: 0.1240 - val_acc: 0.7645 - val_loss: 0.6580
Epoch 49/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 56ms/step - acc: 0.9627 - loss: 0.1129 - val_acc: 0.7695 - val_loss: 0.6968
Epoch 50/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 83ms/step - acc: 0.9638 - loss: 0.1047 - val_acc: 0.7710 - val_loss: 0.6552
Epoch 51/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - acc: 0.9732 - loss: 0.0864 - val_acc: 0.7650 - val_loss: 0.7396
Epoch 52/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 67ms/step - acc: 0.9671 - loss: 0.0942 - val_acc: 0.7555 - val_loss: 0.7382
Epoch 53/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 57ms/step - acc: 0.9671 - loss: 0.0946 - val_acc: 0.7720 - val_loss: 0.7377
Epoch 54/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 67ms/step - acc: 0.9622 - loss: 0.1015 - val_acc: 0.7635 - val_loss: 0.6961
Epoch 55/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 59ms/step - acc: 0.9701 - loss: 0.0853 - val_acc: 0.7630 - val_loss: 0.7255
Epoch 56/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 82ms/step - acc: 0.9747 - loss: 0.0763 - val_acc: 0.7675 - val_loss: 0.7369
Epoch 57/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - acc: 0.9779 - loss: 0.0636 - val_acc: 0.7750 - val_loss: 0.7490
Epoch 58/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 67ms/step - acc: 0.9676 - loss: 0.0900 - val_acc: 0.7725 - val_loss: 0.7293
Epoch 59/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 83ms/step - acc: 0.9741 - loss: 0.0773 - val_acc: 0.7800 - val_loss: 0.7326
Epoch 60/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 67ms/step - acc: 0.9815 - loss: 0.0579 - val_acc: 0.7815 - val_loss: 0.7559
Epoch 61/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - acc: 0.9709 - loss: 0.0727 - val_acc: 0.7765 - val_loss: 0.7229
Epoch 62/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 68ms/step - acc: 0.9750 - loss: 0.0677 - val_acc: 0.7605 - val_loss: 0.8211
Epoch 63/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - acc: 0.9832 - loss: 0.0547 - val_acc: 0.7695 - val_loss: 0.7562
Epoch 64/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 67ms/step - acc: 0.9818 - loss: 0.0570 - val_acc: 0.7705 - val_loss: 0.8045
Epoch 65/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - acc: 0.9802 - loss: 0.0601 - val_acc: 0.7475 - val_loss: 0.9116
```

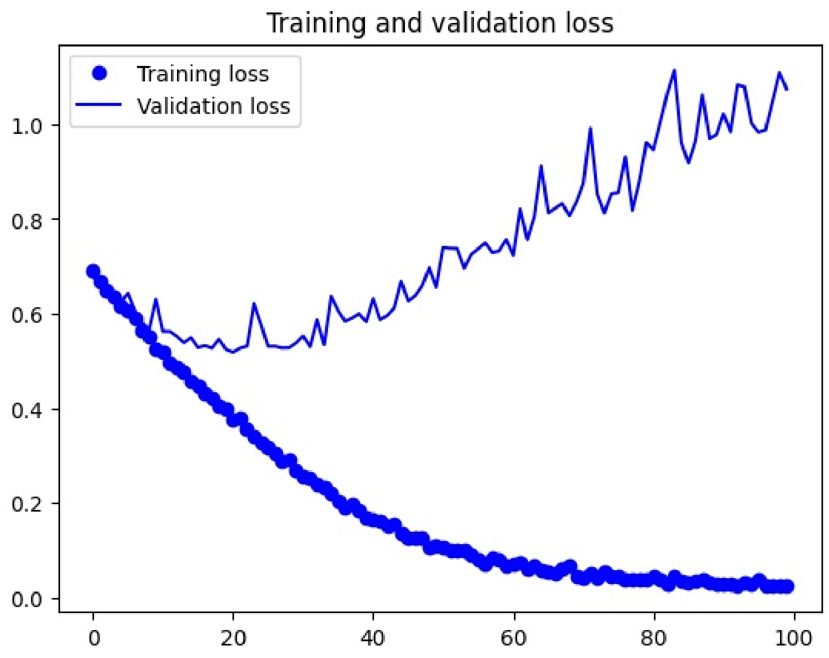
```
Epoch 66/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 65ms/step - acc: 0.9772 - loss: 0.0612 - val_acc: 0.7825 - val_loss: 0.8128
Epoch 67/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 11s 68ms/step - acc: 0.9838 - loss: 0.0504 - val_acc: 0.7800 - val_loss: 0.8227
Epoch 68/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 68ms/step - acc: 0.9774 - loss: 0.0670 - val_acc: 0.7720 - val_loss: 0.8322
Epoch 69/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 82ms/step - acc: 0.9746 - loss: 0.0696 - val_acc: 0.7775 - val_loss: 0.8068
Epoch 70/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 85ms/step - acc: 0.9831 - loss: 0.0522 - val_acc: 0.7800 - val_loss: 0.8347
Epoch 71/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 85ms/step - acc: 0.9906 - loss: 0.0324 - val_acc: 0.7710 - val_loss: 0.8742
Epoch 72/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - acc: 0.9853 - loss: 0.0423 - val_acc: 0.7650 - val_loss: 0.9918
Epoch 73/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 67ms/step - acc: 0.9854 - loss: 0.0431 - val_acc: 0.7710 - val_loss: 0.8523
Epoch 74/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 59ms/step - acc: 0.9861 - loss: 0.0471 - val_acc: 0.7825 - val_loss: 0.8127
Epoch 75/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 67ms/step - acc: 0.9891 - loss: 0.0335 - val_acc: 0.7840 - val_loss: 0.8535
Epoch 76/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - acc: 0.9852 - loss: 0.0444 - val_acc: 0.7760 - val_loss: 0.8556
Epoch 77/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 67ms/step - acc: 0.9907 - loss: 0.0292 - val_acc: 0.7730 - val_loss: 0.9306
Epoch 78/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 59ms/step - acc: 0.9870 - loss: 0.0375 - val_acc: 0.7725 - val_loss: 0.8180
Epoch 79/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 65ms/step - acc: 0.9887 - loss: 0.0336 - val_acc: 0.7835 - val_loss: 0.8789
Epoch 80/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 85ms/step - acc: 0.9890 - loss: 0.0384 - val_acc: 0.7675 - val_loss: 0.9609
Epoch 81/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 67ms/step - acc: 0.9859 - loss: 0.0464 - val_acc: 0.7645 - val_loss: 0.9461
Epoch 82/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - acc: 0.9854 - loss: 0.0438 - val_acc: 0.7725 - val_loss: 1.0048
Epoch 83/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 67ms/step - acc: 0.9912 - loss: 0.0294 - val_acc: 0.7750 - val_loss: 1.0663
Epoch 84/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - acc: 0.9745 - loss: 0.0627 - val_acc: 0.7620 - val_loss: 1.1146
Epoch 85/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 82ms/step - acc: 0.9911 - loss: 0.0291 - val_acc: 0.7825 - val_loss: 0.9612
Epoch 86/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 59ms/step - acc: 0.9890 - loss: 0.0278 - val_acc: 0.7830 - val_loss: 0.9178
Epoch 87/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 67ms/step - acc: 0.9932 - loss: 0.0301 - val_acc: 0.7695 - val_loss: 0.9646
Epoch 88/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - acc: 0.9857 - loss: 0.0353 - val_acc: 0.7595 - val_loss: 1.0624
Epoch 89/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 83ms/step - acc: 0.9925 - loss: 0.0234 - val_acc: 0.7750 - val_loss: 0.9697
Epoch 90/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - acc: 0.9891 - loss: 0.0290 - val_acc: 0.7730 - val_loss: 0.9779
Epoch 91/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 67ms/step - acc: 0.9927 - loss: 0.0253 - val_acc: 0.7885 - val_loss: 1.0220
Epoch 92/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - acc: 0.9910 - loss: 0.0351 - val_acc: 0.7805 - val_loss: 0.9841
Epoch 93/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 83ms/step - acc: 0.9898 - loss: 0.0322 - val_acc: 0.7710 - val_loss: 1.0831
Epoch 94/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - acc: 0.9930 - loss: 0.0277 - val_acc: 0.7750 - val_loss: 1.0792
Epoch 95/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 71ms/step - acc: 0.9936 - loss: 0.0216 - val_acc: 0.7775 - val_loss: 1.0014
Epoch 96/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 9s 57ms/step - acc: 0.9843 - loss: 0.0470 - val_acc: 0.7740 - val_loss: 0.9836
Epoch 97/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 68ms/step - acc: 0.9931 - loss: 0.0229 - val_acc: 0.7825 - val_loss: 0.9879
Epoch 98/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - acc: 0.9888 - loss: 0.0281 - val_acc: 0.7735 - val_loss: 1.0484
Epoch 99/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 67ms/step - acc: 0.9893 - loss: 0.0295 - val_acc: 0.7705 - val_loss: 1.1094
Epoch 100/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 57ms/step - acc: 0.9937 - loss: 0.0199 - val_acc: 0.7785 - val_loss: 1.0745
```

In [24]:

```
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
```

```
plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```





In [26]:
```
# Save Model
model.save('cats_and_dogs_small_1.keras')
```

In [27]:
```
import os
print(os.getcwd())  # Should output: /content
```

```
/content
```

In [28]:
```
# Save the model to HDF5 format (.h5)
model.save('cats_and_dogs_small_1.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js