In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [2]:
```
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[2]:

```
{'kaggle.json': b'{"username":"hamidrezaamin","key":"d15526e38f947f4469926effee3832aa"}'}
```


In [3]:
```
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 [4]:
```
# 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
 95% 775M/812M [00:03<00:00, 146MB/s]
100% 812M/812M [00:03<00:00, 219MB/s]
```

In [5]:
```
# 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 [6]:
```
# Unzip the inner train and test archives
!unzip -q /content/train.zip -d /content/train
!unzip -q /content/test1.zip -d /content/test
```

In [7]:
```
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[7]:
```
'/content/original_dataset_dir/test'
```

In [8]:
```
# 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 [9]:
```
# 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 [10]:

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

In [11]:

```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 [12]:

```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 [13]:

```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 [14]:
```python
# 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]:
```python
# 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/
```

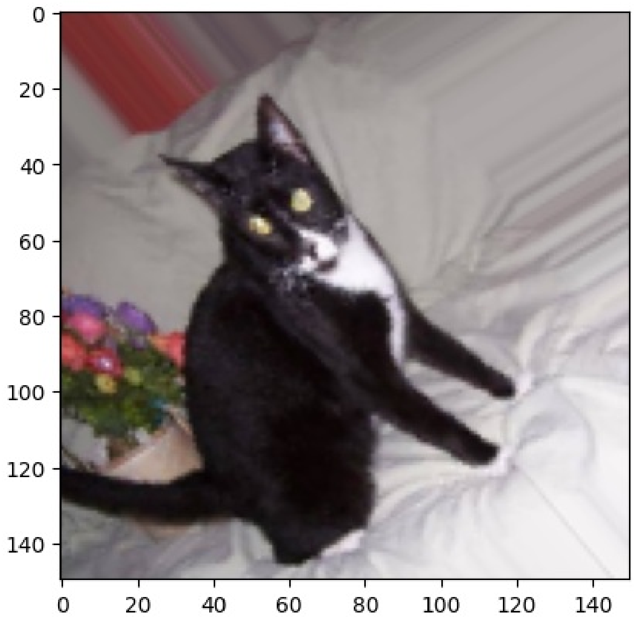
These are just a few of the options available (for more, see the Keras documentation). Let's quickly go over what we just wrote:

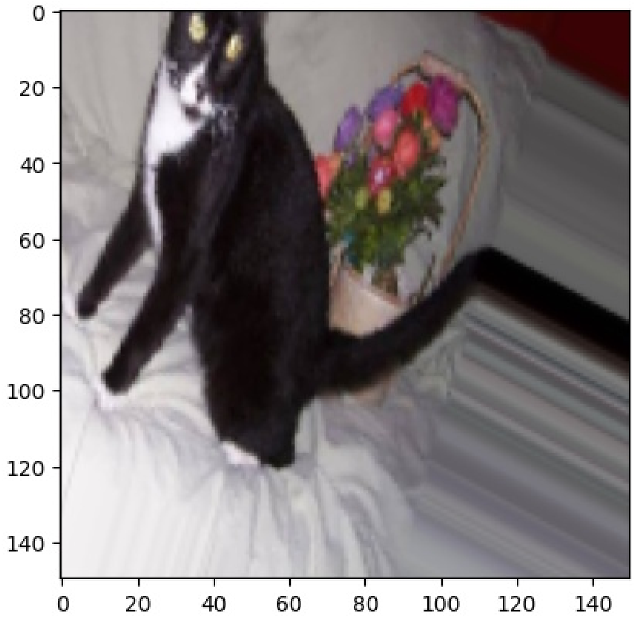
rotation_range is a value in degrees (0-180), a range within which to randomly rotate pictures. width_shift and height_shift are ranges (as a fraction of total width or height) within which to randomly translate pictures vertically or horizontally. shear_range is for randomly applying shearing transformations. zoom_range is for randomly zooming inside pictures. horizontal_flip is for randomly flipping half of the images horizontally -- relevant when there are no assumptions of horizontal asymmetry (e.g. real-world pictures). fill_mode is the strategy used for filling in newly created pixels, which can appear after a rotation or a width/height shift. Let's take a look at our augmented images:

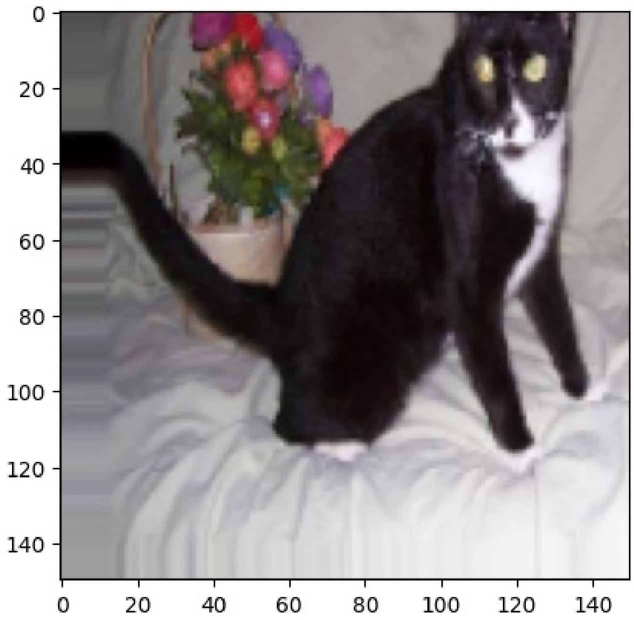
In [16]:
```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator


datagen = ImageDataGenerator(
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')
```

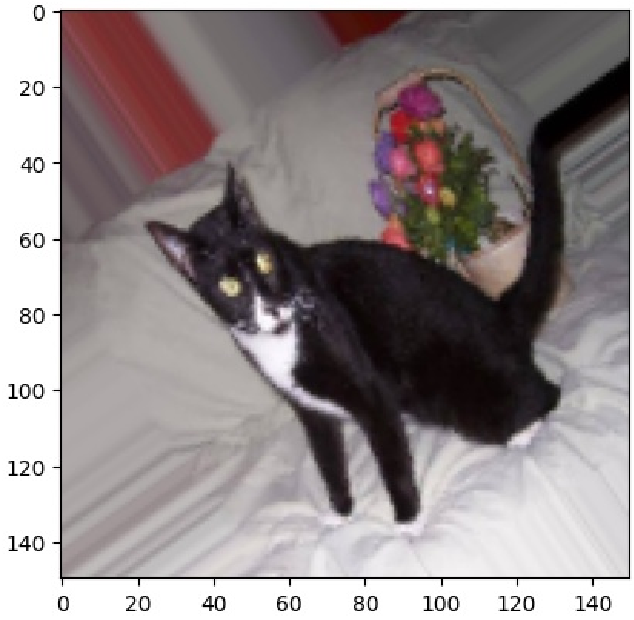
In [17]:
```python
# This is module with image processing utilities

import keras.utils as image
import matplotlib.pyplot as plt

train_cats_dir = '/content/base_dir/train/cat'

fnames = [os.path.join(train_cats_dir, fname) for fname in os.listdir(train_cats_dir)]

# We pick one image to "augment"
img_path = fnames[70]
```

```python
# Read the image and resize it
img = image.load_img(img_path, target_size=(150, 150))

# Convert it to a Numpy array with shape (150, 150, 3)
x = image.img_to_array(img)

# Reshape it to (1, 150, 150, 3)
x = x.reshape((1,) + x.shape)

# The .flow() command below generates batches of randomly transformed images.
# It will loop indefinitely, so we need to `break` the loop at some point!
i = 0
for batch in datagen.flow(x, batch_size=1):
    plt.figure(i)
    imgplot = plt.imshow(image.array_to_img(batch[0]))
    i += 1
    if i % 4 == 0:
        break

plt.show()
```









If we train a new network using this data augmentation configuration, our network will never see twice the same input. However, the inputs that it sees are still heavily intercorrelated, since they come from a small number of original images -- we cannot produce new information, we can only remix existing information. As such, this might not be quite enough to completely get rid of overfitting.

In [18]:

```
# Import necessary Keras modules for building and training the model
from keras import layers, models, optimizers
from keras.layers import BatchNormalization

# Build a Convolutional Neural Network (CNN) using the Sequential API
model = models.Sequential()

# First convolutional block:
# - 32 filters with size 3x3, ReLU activation
# - Input shape is 150x150 RGB images
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)))

# Apply batch normalization to stabilize and speed up training
model.add(BatchNormalization())
```

```python
# Downsample the feature maps using max pooling
model.add(layers.MaxPooling2D((2, 2)))

# Second convolutional block:
# - 64 filters with size 3x3
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))

# Third convolutional block:
# - 128 filters with size 3x3
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))

# Fourth convolutional block:
# - Another 128 filters with size 3x3
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))

# Flatten the 3D feature maps to 1D before feeding into dense layers
model.add(layers.Flatten())

# Add dropout layer to reduce overfitting by randomly turning off 50% of neurons during training
model.add(layers.Dropout(0.5))

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

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

# Compile the model:
# - Use binary_crossentropy for binary classification
# - Use RMSprop optimizer with a small learning rate
# - Track accuracy as the performance metric
model.compile(
    loss='binary_crossentropy',
    optimizer=optimizers.RMSprop(learning_rate=1e-4),
    metrics=['acc']
)

# Display the architecture summary including the number of parameters per layer
model.summary()
```

```
/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)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 32)	896
batch_normalization (BatchNormalization)	(None, 148, 148, 32)	128
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 72, 72, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_2 (Conv2D)	(None, 34, 34, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 34, 34, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 17, 17, 128)	0
conv2d_3 (Conv2D)	(None, 15, 15, 128)	147,584
batch_normalization_3 (BatchNormalization)	(None, 15, 15, 128)	512
max_pooling2d_3 (MaxPooling2D)	(None, 7, 7, 128)	0
flatten (Flatten)	(None, 6272)	0
dropout (Dropout)	(None, 6272)	0
dense (Dense)	(None, 512)	3,211,776
dense_1 (Dense)	(None, 1)	513

Total params: 3,454,529 (13.18 MB)

Trainable params: 3,453,825 (13.18 MB)

Non-trainable params: 704 (2.75 KB)

In [19]:

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,)

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

# Note that the validation data should not be augmented!
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        # This is the target directory
        train_dir,
        # All images will be resized to 150x150
        target_size=(150, 150),
        batch_size=20,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')

# Fit model
history = model.fit(
      train_generator,
      epochs=100,
      validation_data=validation_generator)
```

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

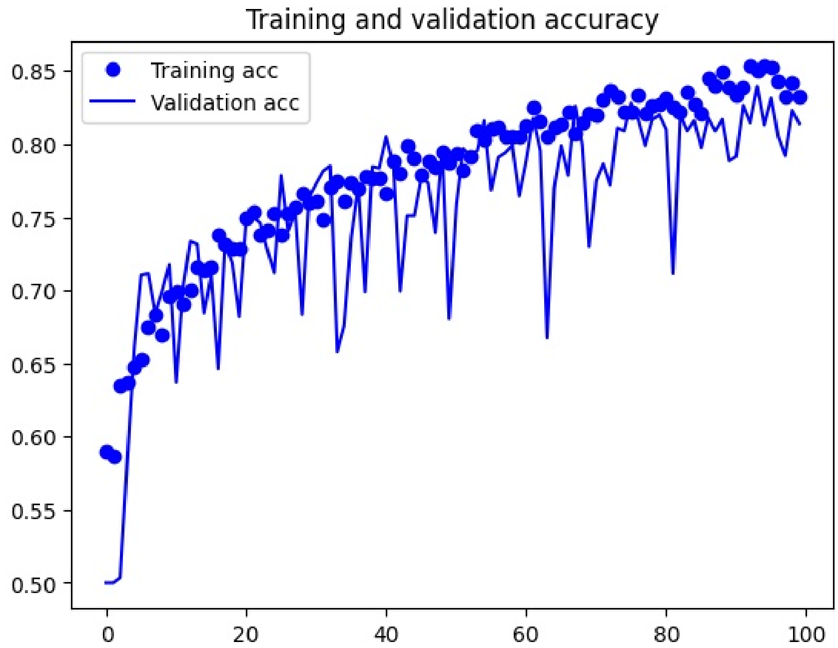
```
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
Epoch 1/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 31s 236ms/step - acc: 0.5823 - loss: 1.2250 - val_acc: 0.5000 - val_loss: 0.8231
Epoch 2/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 17s 170ms/step - acc: 0.5890 - loss: 0.9546 - val_acc: 0.5000 - val_loss: 1.8098
Epoch 3/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 18s 176ms/step - acc: 0.6502 - loss: 0.7812 - val_acc: 0.5035 - val_loss: 1.5591
Epoch 4/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 22s 191ms/step - acc: 0.6059 - loss: 0.7954 - val_acc: 0.5825 - val_loss: 0.9954
Epoch 5/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 205ms/step - acc: 0.6530 - loss: 0.7382 - val_acc: 0.6605 - val_loss: 0.6783
Epoch 6/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 19s 193ms/step - acc: 0.6449 - loss: 0.6791 - val_acc: 0.7105 - val_loss: 0.5760
Epoch 7/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 17s 167ms/step - acc: 0.6674 - loss: 0.6464 - val_acc: 0.7115 - val_loss: 0.5679
Epoch 8/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 18s 181ms/step - acc: 0.6972 - loss: 0.6325 - val_acc: 0.6840 - val_loss: 0.5922
Epoch 9/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 17s 173ms/step - acc: 0.6730 - loss: 0.6258 - val_acc: 0.6995 - val_loss: 0.5770
Epoch 10/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 17s 173ms/step - acc: 0.6989 - loss: 0.5831 - val_acc: 0.7175 - val_loss: 0.5620
Epoch 11/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 19s 195ms/step - acc: 0.7126 - loss: 0.5692 - val_acc: 0.6370 - val_loss: 0.6576
Epoch 12/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 17s 172ms/step - acc: 0.6922 - loss: 0.5926 - val_acc: 0.7025 - val_loss: 0.5728
Epoch 13/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 17s 173ms/step - acc: 0.6913 - loss: 0.5857 - val_acc: 0.7335 - val_loss: 0.5579
Epoch 14/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 18s 180ms/step - acc: 0.7178 - loss: 0.5653 - val_acc: 0.7315 - val_loss: 0.5381
Epoch 15/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 197ms/step - acc: 0.6950 - loss: 0.5759 - val_acc: 0.6845 - val_loss: 0.6145
Epoch 16/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 196ms/step - acc: 0.7314 - loss: 0.5407 - val_acc: 0.7100 - val_loss: 0.5723
Epoch 17/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 18s 178ms/step - acc: 0.7355 - loss: 0.5330 - val_acc: 0.6465 - val_loss: 0.7234
Epoch 18/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 197ms/step - acc: 0.7183 - loss: 0.5677 - val_acc: 0.7320 - val_loss: 0.5482
Epoch 19/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 19s 186ms/step - acc: 0.7246 - loss: 0.5417 - val_acc: 0.7185 - val_loss: 0.5736
Epoch 20/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 196ms/step - acc: 0.7261 - loss: 0.5312 - val_acc: 0.6820 - val_loss: 0.6073
Epoch 21/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 17s 173ms/step - acc: 0.7539 - loss: 0.4851 - val_acc: 0.7470 - val_loss: 0.5317
Epoch 22/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 172ms/step - acc: 0.7451 - loss: 0.5411 - val_acc: 0.7500 - val_loss: 0.5299
Epoch 23/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 18s 175ms/step - acc: 0.7217 - loss: 0.5531 - val_acc: 0.7460 - val_loss: 0.5591
Epoch 24/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 19s 186ms/step - acc: 0.7508 - loss: 0.5110 - val_acc: 0.7280 - val_loss: 0.5441
Epoch 25/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 198ms/step - acc: 0.7649 - loss: 0.4940 - val_acc: 0.7120 - val_loss: 0.5721
Epoch 26/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 18s 177ms/step - acc: 0.7277 - loss: 0.5317 - val_acc: 0.7785 - val_loss: 0.4753
Epoch 27/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 177ms/step - acc: 0.7447 - loss: 0.5270 - val_acc: 0.7410 - val_loss: 0.5325
Epoch 28/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 19s 195ms/step - acc: 0.7732 - loss: 0.5089 - val_acc: 0.7535 - val_loss: 0.5544
Epoch 29/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 18s 172ms/step - acc: 0.7692 - loss: 0.5113 - val_acc: 0.6835 - val_loss: 0.6029
Epoch 30/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 17s 171ms/step - acc: 0.7706 - loss: 0.4839 - val_acc: 0.7640 - val_loss: 0.5004
Epoch 31/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 19s 195ms/step - acc: 0.7675 - loss: 0.5087 - val_acc: 0.7730 - val_loss: 0.4892
Epoch 32/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 17s 171ms/step - acc: 0.7674 - loss: 0.4931 - val_acc: 0.7815 - val_loss: 0.4992
Epoch 33/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 17s 173ms/step - acc: 0.7557 - loss: 0.5043 - val_acc: 0.7850 - val_loss: 0.5066
Epoch 34/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 196ms/step - acc: 0.7658 - loss: 0.4896 - val_acc: 0.6580 - val_loss: 0.7265
Epoch 35/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 17s 172ms/step - acc: 0.7582 - loss: 0.5091 - val_acc: 0.6760 - val_loss: 0.6316
Epoch 36/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 196ms/step - acc: 0.7683 - loss: 0.5079 - val_acc: 0.7365 - val_loss: 0.5420
Epoch 37/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 18s 179ms/step - acc: 0.7800 - loss: 0.4690 - val_acc: 0.7720 - val_loss: 0.5182
Epoch 38/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 17s 173ms/step - acc: 0.7859 - loss: 0.4590 - val_acc: 0.6990 - val_loss: 0.6240
Epoch 39/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 17s 174ms/step - acc: 0.7849 - loss: 0.4501 - val_acc: 0.7845 - val_loss: 0.4985
Epoch 40/100
```

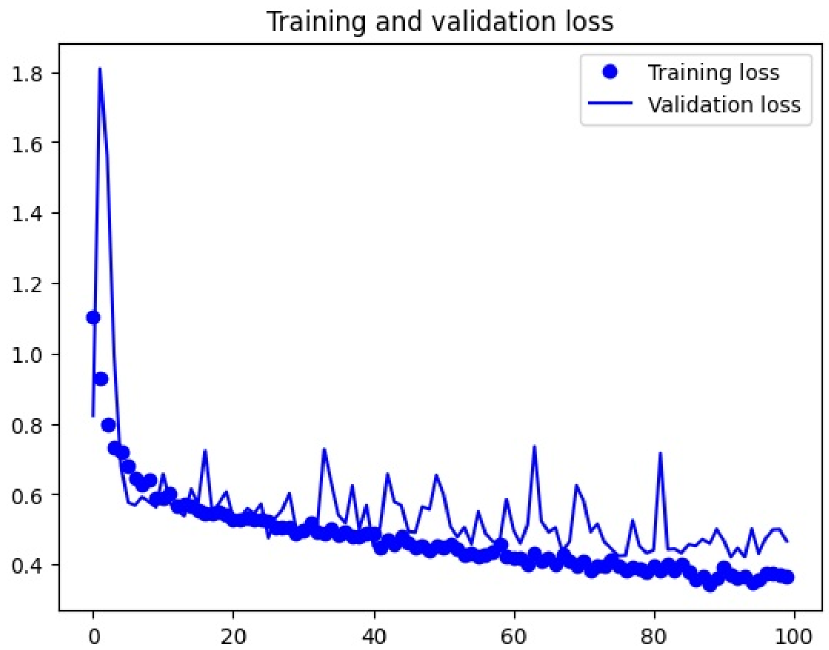
```
100/100 ━━━━━━━━━━━━━━━━━━━━ 19s 187ms/step - acc: 0.7812 - loss: 0.4872 - val_acc: 0.7835 - val_loss: 0.5684
Epoch 41/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 17s 175ms/step - acc: 0.7597 - loss: 0.5196 - val_acc: 0.8050 - val_loss: 0.4513
Epoch 42/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 18s 178ms/step - acc: 0.7987 - loss: 0.4511 - val_acc: 0.7860 - val_loss: 0.5072
Epoch 43/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 19s 187ms/step - acc: 0.7586 - loss: 0.5186 - val_acc: 0.6995 - val_loss: 0.6580
Epoch 44/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 197ms/step - acc: 0.7979 - loss: 0.4596 - val_acc: 0.7510 - val_loss: 0.5777
Epoch 45/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 17s 175ms/step - acc: 0.7908 - loss: 0.4781 - val_acc: 0.7510 - val_loss: 0.5681
Epoch 46/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 173ms/step - acc: 0.7676 - loss: 0.4733 - val_acc: 0.7770 - val_loss: 0.4937
Epoch 47/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 18s 176ms/step - acc: 0.8128 - loss: 0.4198 - val_acc: 0.7735 - val_loss: 0.4915
Epoch 48/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 19s 186ms/step - acc: 0.7788 - loss: 0.4674 - val_acc: 0.7395 - val_loss: 0.5639
Epoch 49/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 17s 174ms/step - acc: 0.8119 - loss: 0.4091 - val_acc: 0.7975 - val_loss: 0.5562
Epoch 50/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 18s 177ms/step - acc: 0.7856 - loss: 0.4435 - val_acc: 0.6805 - val_loss: 0.6545
Epoch 51/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 18s 185ms/step - acc: 0.7882 - loss: 0.4603 - val_acc: 0.7580 - val_loss: 0.6024
Epoch 52/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 19s 174ms/step - acc: 0.7909 - loss: 0.4404 - val_acc: 0.7970 - val_loss: 0.5073
Epoch 53/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 17s 173ms/step - acc: 0.7810 - loss: 0.4713 - val_acc: 0.7920 - val_loss: 0.4785
Epoch 54/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 19s 193ms/step - acc: 0.8084 - loss: 0.4204 - val_acc: 0.7955 - val_loss: 0.5051
Epoch 55/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 19s 195ms/step - acc: 0.8027 - loss: 0.4287 - val_acc: 0.8160 - val_loss: 0.4573
Epoch 56/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 197ms/step - acc: 0.8088 - loss: 0.4213 - val_acc: 0.7685 - val_loss: 0.5505
Epoch 57/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 19s 195ms/step - acc: 0.8107 - loss: 0.4319 - val_acc: 0.7910 - val_loss: 0.4885
Epoch 58/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 197ms/step - acc: 0.8163 - loss: 0.4178 - val_acc: 0.7940 - val_loss: 0.4660
Epoch 59/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 196ms/step - acc: 0.8116 - loss: 0.4438 - val_acc: 0.7990 - val_loss: 0.4584
Epoch 60/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 18s 175ms/step - acc: 0.8017 - loss: 0.4411 - val_acc: 0.7645 - val_loss: 0.5849
Epoch 61/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 23s 196ms/step - acc: 0.8037 - loss: 0.4239 - val_acc: 0.7885 - val_loss: 0.4984
Epoch 62/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 196ms/step - acc: 0.8224 - loss: 0.4113 - val_acc: 0.8175 - val_loss: 0.4595
Epoch 63/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 17s 175ms/step - acc: 0.8074 - loss: 0.4171 - val_acc: 0.7955 - val_loss: 0.5125
Epoch 64/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 18s 177ms/step - acc: 0.8036 - loss: 0.4411 - val_acc: 0.6675 - val_loss: 0.7350
Epoch 65/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 18s 185ms/step - acc: 0.7993 - loss: 0.4214 - val_acc: 0.7700 - val_loss: 0.5234
Epoch 66/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 177ms/step - acc: 0.8106 - loss: 0.4250 - val_acc: 0.7990 - val_loss: 0.4919
Epoch 67/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 18s 178ms/step - acc: 0.8300 - loss: 0.3844 - val_acc: 0.7785 - val_loss: 0.5046
Epoch 68/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 205ms/step - acc: 0.8053 - loss: 0.4309 - val_acc: 0.8260 - val_loss: 0.4364
Epoch 69/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 18s 175ms/step - acc: 0.8136 - loss: 0.3965 - val_acc: 0.7995 - val_loss: 0.4657
Epoch 70/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 199ms/step - acc: 0.8166 - loss: 0.4005 - val_acc: 0.7300 - val_loss: 0.6247
Epoch 71/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 18s 176ms/step - acc: 0.8212 - loss: 0.4061 - val_acc: 0.7750 - val_loss: 0.5828
Epoch 72/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 17s 174ms/step - acc: 0.8359 - loss: 0.3696 - val_acc: 0.7865 - val_loss: 0.4928
Epoch 73/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 199ms/step - acc: 0.8495 - loss: 0.3670 - val_acc: 0.7720 - val_loss: 0.5151
Epoch 74/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 18s 176ms/step - acc: 0.8312 - loss: 0.3958 - val_acc: 0.8110 - val_loss: 0.4636
Epoch 75/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 18s 181ms/step - acc: 0.8122 - loss: 0.4425 - val_acc: 0.8090 - val_loss: 0.4430
Epoch 76/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 19s 187ms/step - acc: 0.8299 - loss: 0.3966 - val_acc: 0.8280 - val_loss: 0.4248
Epoch 77/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 18s 177ms/step - acc: 0.8288 - loss: 0.3886 - val_acc: 0.8165 - val_loss: 0.4259
Epoch 78/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 198ms/step - acc: 0.8323 - loss: 0.3753 - val_acc: 0.7990 - val_loss: 0.5259
Epoch 79/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 19s 185ms/step - acc: 0.8452 - loss: 0.3560 - val_acc: 0.8165 - val_loss: 0.4522
Epoch 80/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 18s 175ms/step - acc: 0.8329 - loss: 0.3714 - val_acc: 0.8200 - val_loss: 0.4328
Epoch 81/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 23s 198ms/step - acc: 0.8404 - loss: 0.3708 - val_acc: 0.8100 - val_loss: 0.4409
```

```
Epoch 82/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 18s 176ms/step - acc: 0.8356 - loss: 0.3656 - val_acc: 0.7115 - val_loss: 0.7168
Epoch 83/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 18s 176ms/step - acc: 0.8235 - loss: 0.3912 - val_acc: 0.8200 - val_loss: 0.4426
Epoch 84/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 18s 185ms/step - acc: 0.8477 - loss: 0.3503 - val_acc: 0.8090 - val_loss: 0.4440
Epoch 85/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 17s 174ms/step - acc: 0.8263 - loss: 0.3979 - val_acc: 0.8160 - val_loss: 0.4326
Epoch 86/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 21s 182ms/step - acc: 0.8224 - loss: 0.3758 - val_acc: 0.7975 - val_loss: 0.4569
Epoch 87/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 22s 198ms/step - acc: 0.8463 - loss: 0.3587 - val_acc: 0.8180 - val_loss: 0.4526
Epoch 88/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 197ms/step - acc: 0.8569 - loss: 0.3407 - val_acc: 0.8090 - val_loss: 0.4721
Epoch 89/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 18s 178ms/step - acc: 0.8514 - loss: 0.3309 - val_acc: 0.8170 - val_loss: 0.4592
Epoch 90/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 17s 173ms/step - acc: 0.8339 - loss: 0.3708 - val_acc: 0.7885 - val_loss: 0.5005
Epoch 91/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 19s 190ms/step - acc: 0.8447 - loss: 0.3618 - val_acc: 0.7915 - val_loss: 0.4696
Epoch 92/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 19s 176ms/step - acc: 0.8508 - loss: 0.3473 - val_acc: 0.8265 - val_loss: 0.4213
Epoch 93/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 197ms/step - acc: 0.8518 - loss: 0.3495 - val_acc: 0.8145 - val_loss: 0.4467
Epoch 94/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 18s 175ms/step - acc: 0.8423 - loss: 0.3776 - val_acc: 0.8395 - val_loss: 0.4219
Epoch 95/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 17s 172ms/step - acc: 0.8555 - loss: 0.3471 - val_acc: 0.8130 - val_loss: 0.5013
Epoch 96/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 197ms/step - acc: 0.8473 - loss: 0.3507 - val_acc: 0.8315 - val_loss: 0.4310
Epoch 97/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 17s 175ms/step - acc: 0.8434 - loss: 0.4019 - val_acc: 0.8050 - val_loss: 0.4748
Epoch 98/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 17s 174ms/step - acc: 0.8368 - loss: 0.3489 - val_acc: 0.7920 - val_loss: 0.4986
Epoch 99/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 20s 197ms/step - acc: 0.8467 - loss: 0.3734 - val_acc: 0.8230 - val_loss: 0.4996
Epoch 100/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 18s 176ms/step - acc: 0.8404 - loss: 0.3537 - val_acc: 0.8140 - val_loss: 0.4666
```

In [20]:

```
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()
```


Training and validation accuracy
Training acc
Validation acc
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0
20
40
60
80
100


Training and validation loss
Training loss
Validation loss
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0
20
40
60
80
100