Show code

Let's start with a model that's very effective at learning Cats v Dogs.

It's similar to the previous models that you have used, but I have updated the layers definition. Note that there are now 4 convolutional layers with 32, 64, 128 and 128 convolutions respectively.

Also, this will train for 100 epochs, because I want to plot the graph of loss and accuracy.

```
!wget --no-check-certificate \
    https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip \
    -O /tmp/cats_and_dogs_filtered.zip

import os
import zipfile
import tensorflow as tf
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator

local_zip·=·'/tmp/cats_and_dogs_filtered.zip'
zip_ref·=·zipfile.ZipFile(local_zip,·'r')
zip_ref.extractall('/tmp')
zip_ref.close()

base_dir·=·'/tmp/cats_and_dogs_filtered'
train_dir·=·os.path.join(base_dir,·'train')·#·/tmp/cats_and_dogs_filtered/train
validation_dir·=·os.path.join(base_dir,·'validation')·#·/tmp/cats_and_dogs_filtered/validatio

# Directory with our training cat pictures
train_cats_dir = os.path.join(train_dir, 'cats') # /tmp/cats_and_dogs_filtered/train/cats

# Directory with our training dog pictures
train_dogs_dir = os.path.join(train_dir, 'dogs') # /tmp/cats_and_dogs_filtered/train/dogs

# Directory with our validation cat pictures
validation_cats_dir = os.path.join(validation_dir, 'cats') # /tmp/cats_and_dogs_filtered/vali

# Directory with our validation dog pictures
validation_dogs_dir = os.path.join(validation_dir, 'dogs') # /tmp/cats_and_dogs_filtered/vali

model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
```

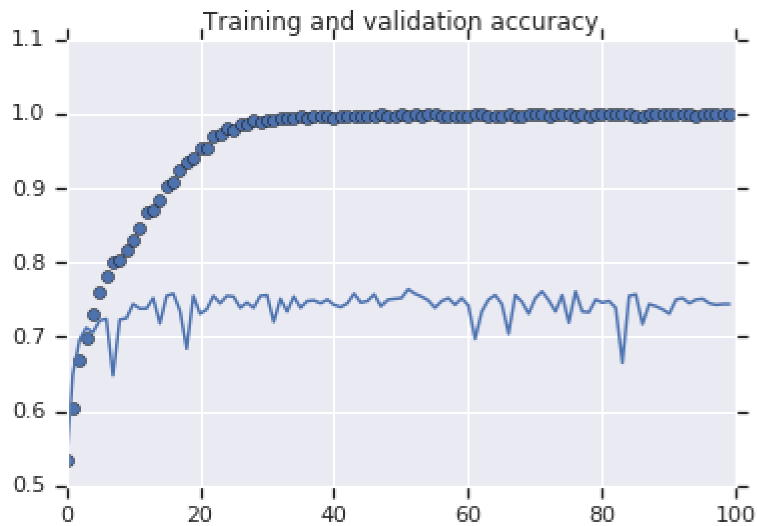
```
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(loss='binary_crossentropy',
              optimizer=RMSprop(lr=1e-4),
              metrics=['accuracy'])

# All images will be rescaled by 1./255
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

# Flow training images in batches of 20 using train_datagen generator
train_generator = train_datagen.flow_from_directory(
        train_dir,  # This is the source directory for training images
        target_size=(150, 150),  # All images will be resized to 150x150
        batch_size=20,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')

# Flow validation images in batches of 20 using test_datagen generator
validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')

history = model.fit(
      train_generator,
      steps_per_epoch=100,  # 2000 images = batch_size * steps
      epochs=100,
      validation_data=validation_generator,
      validation_steps=50,  # 1000 images = batch_size * steps
      verbose=2)
```

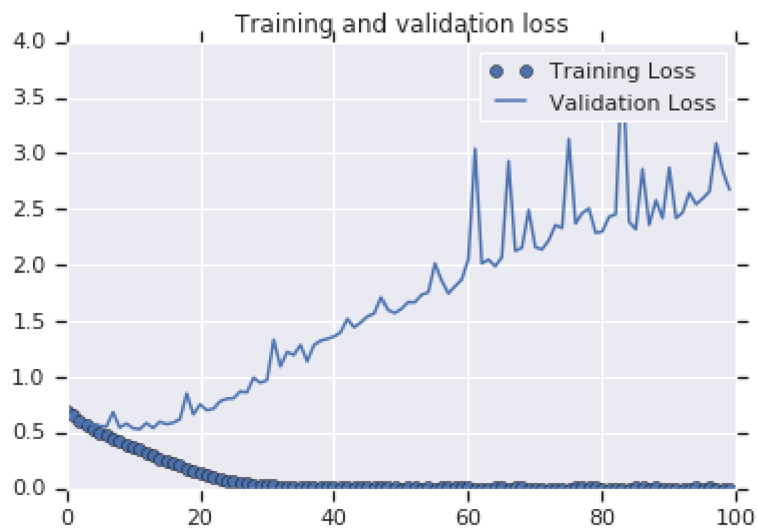
```
--2022-06-13 14:30:08--  https://storage.googleapis.com/mledu-datasets/cats_and_dogs_fi
Resolving storage.googleapis.com (storage.googleapis.com)... 172.217.193.128, 172.217.2
Connecting to storage.googleapis.com (storage.googleapis.com)|172.217.193.128|:443... co
HTTP request sent, awaiting response... 200 OK
Length: 68606236 (65M) [application/zip]
Saving to: ‘/tmp/cats_and_dogs_filtered.zip’

/tmp/cats_and_dogs_ 100%[===================>]  65.43M   113MB/s    in 0.6s
```

```
2022-06-13 14:30:09 (113 MB/s) - ‘/tmp/cats_and_dogs_filtered.zip’ saved [68606236/6860

/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/rmsprop.py:130: UserWarning: 
  super(RMSprop, self).__init__(name, **kwargs)
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
Epoch 1/100
100/100 - 130s - loss: 0.6927 - accuracy: 0.5185 - val_loss: 0.6762 - val_accuracy: 0.5
Epoch 2/100
100/100 - 108s - loss: 0.6598 - accuracy: 0.6035 - val_loss: 0.6584 - val_accuracy: 0.6
Epoch 3/100
100/100 - 110s - loss: 0.6175 - accuracy: 0.6590 - val_loss: 0.6063 - val_accuracy: 0.6
Epoch 4/100
100/100 - 108s - loss: 0.5738 - accuracy: 0.7090 - val_loss: 0.5753 - val_accuracy: 0.7
Epoch 5/100
100/100 - 106s - loss: 0.5294 - accuracy: 0.7400 - val_loss: 0.5605 - val_accuracy: 0.7
Epoch 6/100
100/100 - 107s - loss: 0.4987 - accuracy: 0.7505 - val_loss: 0.5455 - val_accuracy: 0.7
Epoch 7/100
100/100 - 108s - loss: 0.4663 - accuracy: 0.7790 - val_loss: 0.5845 - val_accuracy: 0.7
Epoch 8/100
100/100 - 108s - loss: 0.4351 - accuracy: 0.7980 - val_loss: 0.5522 - val_accuracy: 0.7
Epoch 9/100
100/100 - 108s - loss: 0.4129 - accuracy: 0.8060 - val_loss: 0.6638 - val_accuracy: 0.6
Epoch 10/100
100/100 - 107s - loss: 0.3933 - accuracy: 0.8250 - val_loss: 0.5327 - val_accuracy: 0.7
Epoch 11/100
100/100 - 106s - loss: 0.3624 - accuracy: 0.8380 - val_loss: 0.5260 - val_accuracy: 0.7
Epoch 12/100
100/100 - 109s - loss: 0.3380 - accuracy: 0.8575 - val_loss: 0.5354 - val_accuracy: 0.7
Epoch 13/100
100/100 - 109s - loss: 0.3035 - accuracy: 0.8705 - val_loss: 0.5411 - val_accuracy: 0.7
Epoch 14/100
100/100 - 108s - loss: 0.2804 - accuracy: 0.8795 - val_loss: 0.5468 - val_accuracy: 0.7
Epoch 15/100
100/100 - 108s - loss: 0.2540 - accuracy: 0.9035 - val_loss: 0.5484 - val_accuracy: 0.7
Epoch 16/100
```

```
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

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

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()
```





The Training Accuracy is close to 100%, and the validation accuracy is in the 70%-80% range. This is a great example of overfitting -- which in short means that it can do very well with images it has seen before, but not so well with images it hasn't. Let's see if we can do better to avoid overfitting -- and one simple method is to augment the images a bit. If you think about it, most pictures of a cat are very similar -- the ears are at the top, then the eyes, then the mouth etc. Things like the distance between the eyes and ears will always be quite similar too.

What if we tweak with the images to change this up a bit -- rotate the image, squash it, etc. That's what image augementation is all about. And there's an API that makes it easy...

Now take a look at the ImageGenerator. There are properties on it that you can use to augment the image.

```
# Updated to do image augmentation
train_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')
```

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

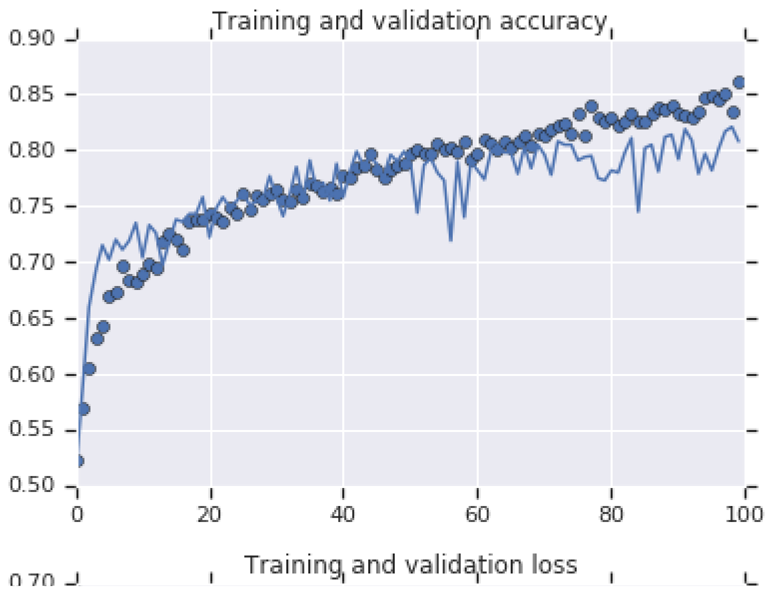
- rotation_range is a value in degrees (0–180), a range within which to randomly rotate pictures.
- width_shift and height_shift are ranges (as a fraction of total width or height) within which to randomly translate pictures vertically or horizontally.
- shear_range is for randomly applying shearing transformations.
- zoom_range is for randomly zooming inside pictures.
- horizontal_flip is for randomly flipping half of the images horizontally. This is relevant when there are no assumptions of horizontal assymmetry (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.

Here's some code where we've added Image Augmentation. Run it to see the impact.

```
!wget --no-check-certificate \
    https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip \
    -O /tmp/cats_and_dogs_filtered.zip

import os
import zipfile
import tensorflow as tf
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator

local_zip = '/tmp/cats_and_dogs_filtered.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('/tmp')
zip_ref.close()

base_dir = '/tmp/cats_and_dogs_filtered'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
```

```
# Directory with our training cat pictures
train_cats_dir = os.path.join(train_dir, 'cats')

# Directory with our training dog pictures
train_dogs_dir = os.path.join(train_dir, 'dogs')

# Directory with our validation cat pictures
validation_cats_dir = os.path.join(validation_dir, 'cats')

# Directory with our validation dog pictures
validation_dogs_dir = os.path.join(validation_dir, 'dogs')

model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(loss='binary_crossentropy',
              optimizer=RMSprop(lr=1e-4),
              metrics=['accuracy'])

# This code has changed. Now instead of the ImageGenerator just rescaling
# the image, we also rotate and do other operations
# Updated to do image augmentation
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,
······fill_mode='nearest')

test_datagen = ImageDataGenerator(rescale=1./255)

# Flow training images in batches of 20 using train_datagen generator
train_generator = train_datagen.flow_from_directory(
        train_dir,  # This is the source directory for training images
        target_size=(150, 150),  # All images will be resized to 150x150
        batch_size=20,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')
```

```
# Flow validation images in batches of 20 using test_datagen generator
validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')

history = model.fit(
      train_generator,
      steps_per_epoch=100,  # 2000 images = batch_size * steps
      epochs=100,
      validation_data=validation_generator,
      validation_steps=50,  # 1000 images = batch_size * steps
      verbose=2)


import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

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

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()
```




```
!wget --no-check-certificate \
    https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip \
    -O /tmp/cats_and_dogs_filtered.zip

import os
import zipfile
import tensorflow as tf
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator

local_zip = '/tmp/cats_and_dogs_filtered.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('/tmp')
zip_ref.close()

base_dir = '/tmp/cats_and_dogs_filtered'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')

# Directory with our training cat pictures
train_cats_dir = os.path.join(train_dir, 'cats')

# Directory with our training dog pictures
train_dogs_dir = os.path.join(train_dir, 'dogs')

# Directory with our validation cat pictures
validation_cats_dir = os.path.join(validation_dir, 'cats')

# Directory with our validation dog pictures
validation_dogs_dir = os.path.join(validation_dir, 'dogs')

model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
```

```
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(loss='binary_crossentropy',
              optimizer=RMSprop(lr=1e-4),
              metrics=['accuracy'])

# This code has changed. Now instead of the ImageGenerator just rescaling
# the image, we also rotate and do other operations
# Updated to do image augmentation
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,
      fill_mode='nearest')

test_datagen = ImageDataGenerator(rescale=1./255)

# Flow training images in batches of 20 using train_datagen generator
train_generator = train_datagen.flow_from_directory(
        train_dir,  # This is the source directory for training images
        target_size=(150, 150),  # All images will be resized to 150x150
        batch_size=20,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')

# Flow validation images in batches of 20 using test_datagen generator
validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')

history = model.fit(
      train_generator,
      steps_per_epoch=100,  # 2000 images = batch_size * steps
      epochs=100,
      validation_data=validation_generator,
```

```
 validation_steps=50,  # 1000 images = batch_size * steps
 verbose=2)
```

```
--2019-02-12 07:59:45--  https://storage.googleapis.com/mledu-datasets/cats_and_dogs_
Resolving storage.googleapis.com... 2607:f8b0:4001:c1c::80, 173.194.197.128
Connecting to storage.googleapis.com|2607:f8b0:4001:c1c::80|:443... connected.
WARNING: cannot verify storage.googleapis.com's certificate, issued by 'CN=Google Int
  Unable to locally verify the issuer's authority.
HTTP request sent, awaiting response... 200 OK
Length: 68606236 (65M) [application/zip]
Saving to: '/tmp/cats_and_dogs_filtered.zip'

/tmp/cats_and_dogs_ 100%[===================>]  65.43M   243MB/s   in 0.3s

2019-02-12 07:59:46 (243 MB/s) - '/tmp/cats_and_dogs_filtered.zip' saved [68606236/68

Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
Epoch 1/100
100/100 - 14s - loss: 0.6931 - acc: 0.5350 - val_loss: 0.6907 - val_acc: 0.5080
Epoch 2/100
100/100 - 14s - loss: 0.6855 - acc: 0.5400 - val_loss: 0.6660 - val_acc: 0.6200
Epoch 3/100
100/100 - 13s - loss: 0.6702 - acc: 0.5810 - val_loss: 0.6665 - val_acc: 0.5650
Epoch 4/100
100/100 - 13s - loss: 0.6541 - acc: 0.6000 - val_loss: 0.6342 - val_acc: 0.6300
Epoch 5/100
100/100 - 14s - loss: 0.6415 - acc: 0.6180 - val_loss: 0.6457 - val_acc: 0.5920
Epoch 6/100
100/100 - 13s - loss: 0.6248 - acc: 0.6495 - val_loss: 0.5875 - val_acc: 0.6840
Epoch 7/100
100/100 - 13s - loss: 0.6115 - acc: 0.6575 - val_loss: 0.5864 - val_acc: 0.6810
Epoch 8/100
100/100 - 13s - loss: 0.6010 - acc: 0.6780 - val_loss: 0.5550 - val_acc: 0.7130
Epoch 9/100
100/100 - 14s - loss: 0.5972 - acc: 0.6670 - val_loss: 0.5640 - val_acc: 0.7020
Epoch 10/100
100/100 - 14s - loss: 0.5877 - acc: 0.6920 - val_loss: 0.5830 - val_acc: 0.6900
Epoch 11/100
100/100 - 14s - loss: 0.5761 - acc: 0.7055 - val_loss: 0.5663 - val_acc: 0.7030
Epoch 12/100
100/100 - 14s - loss: 0.5708 - acc: 0.7100 - val_loss: 0.5662 - val_acc: 0.7030
Epoch 13/100
100/100 - 14s - loss: 0.5810 - acc: 0.6935 - val_loss: 0.5600 - val_acc: 0.6980
Epoch 14/100
100/100 - 14s - loss: 0.5734 - acc: 0.7025 - val_loss: 0.5253 - val_acc: 0.7220
Epoch 15/100
100/100 - 13s - loss: 0.5616 - acc: 0.7150 - val_loss: 0.6329 - val_acc: 0.6470
Epoch 16/100
100/100 - 14s - loss: 0.5487 - acc: 0.7150 - val_loss: 0.5577 - val_acc: 0.7160
Epoch 17/100
100/100 - 13s - loss: 0.5575 - acc: 0.7180 - val_loss: 0.5160 - val_acc: 0.7390
Epoch 18/100
100/100 - 13s - loss: 0.5481 - acc: 0.7250 - val_loss: 0.5057 - val_acc: 0.7360
Epoch 19/100
100/100 - 14s - loss: 0.5398 - acc: 0.7285 - val_loss: 0.5052 - val_acc: 0.7320
```

```
Epoch 20/100
100/100 - 13s - loss: 0.5448 - acc: 0.7240 - val_loss: 0.4988 - val_acc: 0.7560
Epoch 21/100
100/100 - 13s - loss: 0.5321 - acc: 0.7345 - val_loss: 0.5014 - val_acc: 0.7500
```

```
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

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

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()
```

Executing (33m 2s) … › error_han… › … › error_han… › __cal… › _c… › __cal… › _call_… › c… › quick_exec… •••