Image classification report

Describe the architecture used in the lab

The architecture I used in the lab was a fairly basic one that is very common. I alternated different layers of convolution and max pooling.

Basic network

The first basic network looks like this:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_17 (Conv2D)           (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 7, 7, 128)         0
_________________________________________________________________
flatten_5 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_9 (Dense)              (None, 512)               3211776
_________________________________________________________________
dense_10 (Dense)             (None, 5)                 2565
=================================================================
Total params: 3,455,173
Trainable params: 3,455,173
Non-trainable params: 0
```

This network suffers overfitting problem and only has an accuracy of 64%.

Basic network with dropout to fight overfit

To resolve the problem of overfitting, I added a dropout layer which has improved the performance.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_21 (Conv2D)           (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 7, 7, 128)         0
_________________________________________________________________
flatten_6 (Flatten)          (None, 6272)              0
_________________________________________________________________
dropout_5 (Dropout)          (None, 6272)              0
_________________________________________________________________
dense_11 (Dense)             (None, 512)               3211776
_________________________________________________________________
dense_12 (Dense)             (None, 5)                 2565
=================================================================
Total params: 3,455,173
Trainable params: 3,455,173
Non-trainable params: 0
_________________________________________________________________
```

The accuracy bumped from 64% to 72%, which represents a 8% gain of accuracy with just one dropout layer added to the end of the network.

Using a pretrained convolution base

Using the pretrained InceptionV3 base, I have tried two approaches.

Use base output as input for my own network

I use the features extracted using the pretrained convolution as input for my network which only has 3 layers.

```
model_3 = models.Sequential()
model_3.add(layers.Dense(256, activation='relu', input_dim=3*3*2048))
model_3.add(layers.Dropout(0.5))
model_3.add(layers.Dense(5, activation='softmax'))
```

```
model_3.compile(optimizer=optimizers.RMSprop(learning_rate=2e-5),
loss='categorical_crossentropy', metrics=['acc'])
```

The network stopped converging after only 5 epochs, with a gain of 10% of performance resulting to an accuracy of 82%. This is a very notable gain which does not require a powerful machine and a long training time.

Integrate the base convolution to my network with data augmentation

It is also possible to integrate the the pretrained base convolutional to our own network. This approach allows us to use data augmentation during training which was not available when using the features extraction as I previously did.

```
from keras import models, layers

model_4 = models.Sequential()
model_4.add(conv_base)
model_4.add(layers.Flatten())
model_4.add(layers.Dense(256, activation='relu'))
model_4.add(layers.Dense(5, activation='softmax'))

model_4.compile(loss='categorical_crossentropy',
optimizer=optimizers.Adam(), metrics=['acc'])
```

Here we are extending the model and I have freezed the base so that the weight are not influenced by the other layers that I have added after. On this model I have an accuracy of 71%.

Experiment Chollet's 5.4 Notebook and read article

Being able to understand the way the network take decision is important, especially when it comes to debugging. Using Class Activation Map visualization, a technique that produces a heatmap indicating how important each location in the image is with respect to a given class.

In the Chollet's notebook, the example was on a photo of two elephants with a natural background. Knowing that the prediction 'African_elephant' has the highest propability, the Grad-CAM process was used to highlight the region where the activation was the strongest. Indeed, when stacking the activation heatmap on the original image, the most important zones were located at the elephants which is what we expected. A little interesting detail was that the ears of the elephants were strongly highlighted which gave us some insights about how the network could tell it was 'African' elephants.

Run GCAM on a flower image

See PDF file attached

In [108…
```
import os
import random
import shutil
from tqdm import tqdm

# The machine name
machine_name = 'colab'

# To create the same dataset
random.seed(0)

base = '/content/drive/My Drive/Colab Notebooks/lab_3_cnn/'
```

In [109…
```
from google.colab import drive
drive.mount("/content/drive")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

In [110…
```
from keras.applications import VGG16
from keras import backend as K
from keras.models import load_model
K.clear_session()


model = VGG16(weights='imagenet')
```

In [131…
```
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input, decode_predictions
import numpy as np

im_path = base + 'flowers/sf3.jpg'
img = image.load_img(im_path, target_size=(224,224))

x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)
```

In [132…
```
preds = model.predict(x)
```

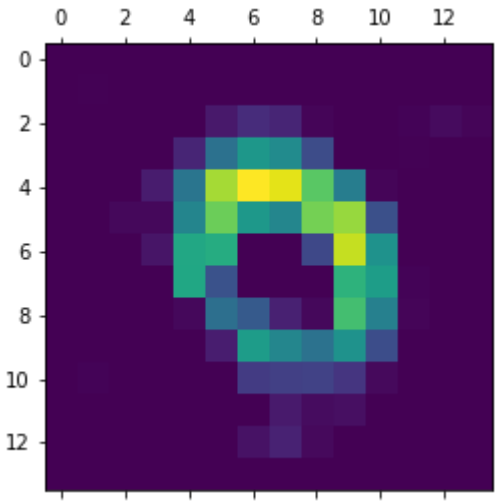
In [132…
```

```

In [133…
```
import tensorflow as tf
tf.compat.v1.disable_eager_execution()

flower_output = model.output[:, np.argmax(preds[0])]
last_conv_layer = model.get_layer('block5_conv3')
grads = K.gradients(flower_output, last_conv_layer.output)[0]
pooled_grads = K.mean(grads, axis=(0, 1, 2))
iterate = K.function([model.input], [pooled_grads, last_conv_layer.output[0]]
pooled_grads_value, conv_layer_output_value = iterate([x])
for i in range(512):
    conv_layer_output_value[:, :, i] *= pooled_grads_value[i]
heatmap = np.mean(conv_layer_output_value, axis=-1)
```

In [134…
```
heatmap = np.maximum(heatmap, 0)
heatmap /= np.max(heatmap)
plt.matshow(heatmap)
plt.show()
```



In [147…
```
import cv2

img = cv2.imread(im_path)
heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0]))
heatmap = np.uint8(255 * heatmap)
heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)
# 0.4 here is a heatmap intensity factor
superimposed_img = heatmap * 0.4 + img
cv2.imwrite(base +'flower_cam.jpg', superimposed_img)
```

Out[147… True

In [148…
```
import matplotlib.pyplot as plt
```

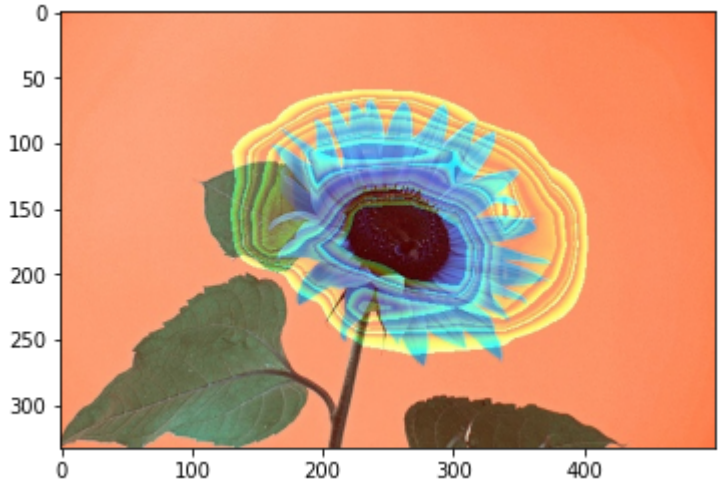
In [149…
```
org_im = cv2.imread(im_path)
plt.imshow(org_im)
```

Out[149… <matplotlib.image.AxesImage at 0x7f563f3673c8>

In [150…

```
im = cv2.imread(base +'flower_cam.jpg')
plt.imshow(im)
```

Out[150… <matplotlib.image.AxesImage at 0x7f563f5c8ba8>



In [150…