CNN 可視化方法

前言：

CNN 可視化可分成兩種:向前傳播法跟反向傳播法。

1. 向前傳播法是把模型學習的期間學到的東西圖示出來，比如說可視化 filter 跟 feature map。
2. 反向傳播法是從特定層反傳回輸入，例如 Deconvolution、Guided-Backpropagation、CAM(類激活映射)等。

第一部分我用下列三種可視化方法來了解 VGG16 模型是如何判斷圖片：

1. Visualization filter
2. Visualization feature map
3. Heatmap – Grad-CAM、Grad-CAM ++(方法解釋詳見程式碼說明)

第二部分是在前一部分第三點的基礎上做進一步的探討：

1. 不同目標層的比較：
   根據模型結構選擇數字最大的卷積層、最後分支中的卷積層、最後一個混和層等。
2. 不同結構模型的比較：
   選擇直線結構、殘差結構、堆疊結構、殘差堆疊結構四種不同模型，非直線結構的模型僅挑在前一點中效果最好的一個目標層進行比較。

參考資料：

- Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization
- Grad-CAM++: Improved Visual Explanations for Deep Convolutional Networks

環境配置：

tensorflow 2.4.0
keras 2.4.3
opencv 4.1.2

I. Visualization of VGG16 pre-trained model with ImageNet datas.

程式碼:

```
import tensorflow as tf
import keras
from keras.applications import vgg16
from keras.applications.vgg16 import preprocess_input, decode_predictions
from keras import backend as K
from keras import models
import skimage.io
import cv2
import matplotlib.pyplot as plt
import math
import numpy as np


#define some function.
#印出 tensor 的形狀、最大最小值(用來確認大小跟有沒有 normalize)
def tensor_summary(tensor):
    print("shape: {}  min: {}  max: {}".format(tensor.shape, tensor.min(),
 tensor.max()))

#Takes a tensor of 3 dimensions (height, width, colors)
#將值 normalize 到[0,1]
def normalize(image):
    image = image.astype(np.float32)
    return (image - image.min()) / (image.max() - image.min() + 1e-5)

def display_img(images, titles=None, cols=5, interpolation=None, cmap="Gre
ys_r"):
#images: A list of images.(圖片列表)
#ndarray:Numpy 數組的列表,每個數組代表一個圖像。
#concatenate array(images):Numpy 數組列表的列表(清單)。在這種情況下，內部列表中的圖
像將合併為一張圖像。
    titles = titles or [""] * len(images)
    rows = math.ceil(len(images) / cols)
    height_ratio = 1.2 * (rows/cols) * (0.5 if type(images[0]) is not np.n
darray else 1)
    plt.figure(figsize=(15, 15 * height_ratio))
    i = 1
    #用 zip 同時迭代 images 跟 titles，每個元素以一對一的方式配對起來/zip 對不同長度的
以短的為準
```

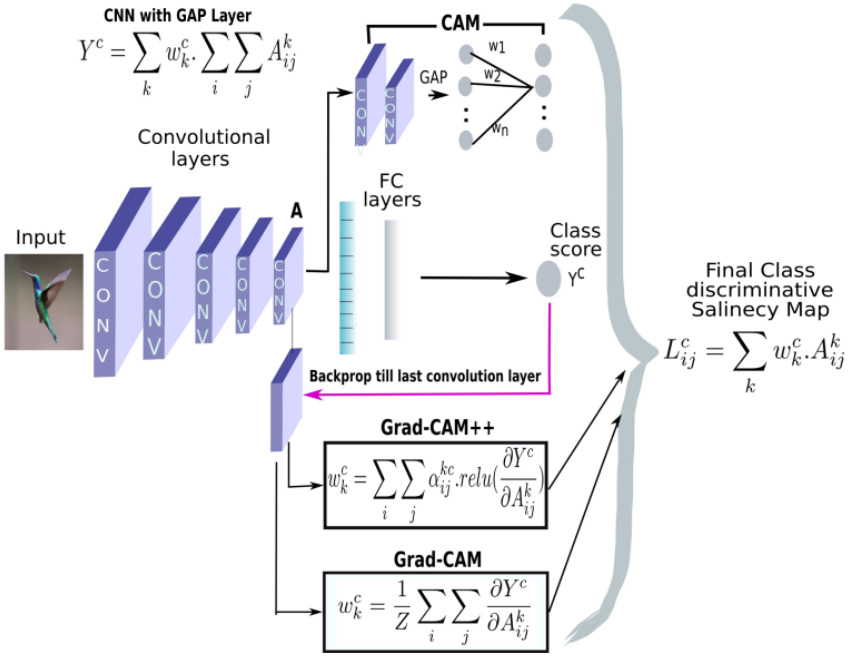
```
    for image, title in zip(images, titles):
        plt.subplot(rows, cols, i)
        plt.axis("off")#關閉軸

        if type(image) is not np.ndarray:
            image = [normalize(g) for g in image]
            image = np.concatenate(image, axis=1)
        else:
            image = normalize(image)
        plt.title(title, fontsize=9)
        plt.imshow(image, cmap=cmap, interpolation=interpolation)
        i += 1

def read_layer(model, x, layer_name):
    #返回指定層激活值
    get_layer_output = K.function([model.layers[0].input], [model.get_laye
r(layer_name).output])
    outputs = get_layer_output([x])[0]
    tensor_summary(outputs)
    return outputs[0]

#輸出特定層的權重跟捲積出的圖片
def view_layer(model, x, layer_name, cols=5):
    print(layer_name)
    outputs = read_layer(model, x, layer_name)
    display_img([outputs[:,:,i] for i in range(20)], cols=cols)
    plt.show()

#set model.
model = vgg16.VGG16(weights='imagenet')#載入 Keras 預訓練好的權重
model.summary()#顯示模型結構，主要是用來確認 layer 的名稱，方便之後看 feature map 的
時候用

#模型架構圖示
from keras.utils import plot_model
plot_model(model, to_file='model.png')
'''
讀圖方式 skimage.io.imread()和 cv2.imread()：
共通點:
➢ 讀出圖片格式是 uint8(unsigned int)
➢ value 是 np.array
通道值默認範圍 0~255
差異:
```

```
➢ skimage.io.imread:圖像數據以 RGB 的格式儲存
➢ cv2.imread:圖像數據以 BGR 的格式儲存，需要改成 RGB 的形式才能正常顯示原圖的顏色。
'''

#輸入圖片
dim=224
image=skimage.io.imread("husky3.jpg")
image = cv2.resize(image, (dim, dim), interpolation=cv2.INTER_AREA)
tensor_summary(image)
display_img([image], cols=2)


#visual filter
weights = model.get_layer("block1_conv1").get_weights()[0]
tensor_summary(weights)
display_img([weights[:,:,::,i] for i in range(weights.shape[3])], cols=16,
 interpolation="none")
plt.show()

#visual feature map
view_layer(model, x, "block1_conv1")
view_layer(model, x, "block2_conv1")
view_layer(model, x, "block3_conv1")
view_layer(model, x, "block4_conv1")
view_layer(model, x, "block5_conv1")
view_layer(model, x, "block5_conv3")


#預測圖片的標籤
x = image.astype(np.float32)
x = np.expand_dims(x, axis=0)#擴維
x = preprocess_input(x)
tensor_summary(x)
predictions = model.predict(x)#預測

#找到最大值對應的標籤
label_index = np.argmax(predictions)
print("label index: ", label_index)

#顯示預測前三
decode_predictions(predictions,top=3)

#可視化熱圖-展示模型是因為圖片的哪個區域而預測出此結果
```

'''

說明:

CAM、Grad-CAM、Grad-CAM++的差異:

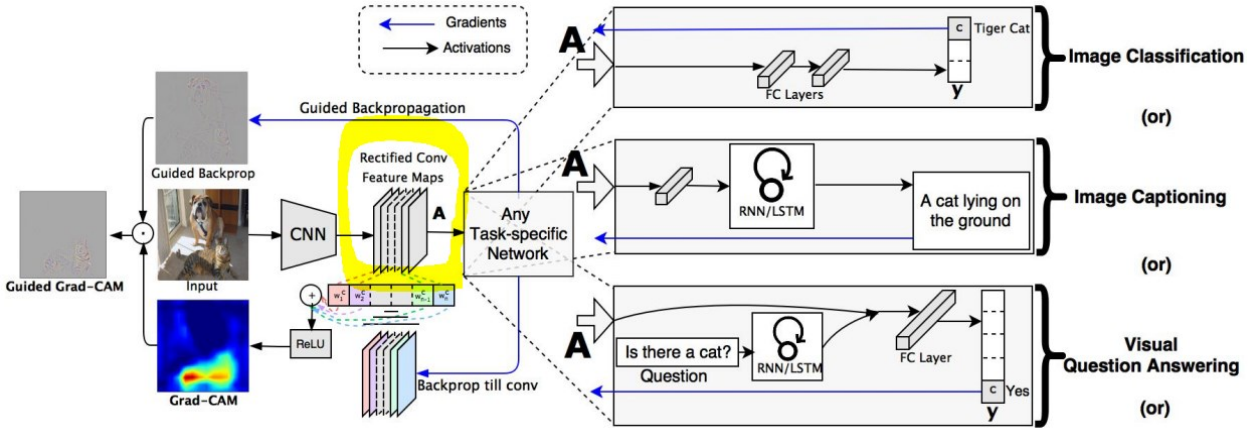


Grad-CAM(梯度加權類激活映射)比 CAM(類激活映射)方法來說更方便，因為 CAM 必須要有 GAP(global average pooling)，這表示如果原先模型沒有的話必須要改變模型結構並重新訓練。但是 Grad-CAM 沒有這個困擾，他可以適用所有類型的模型。

Grad-CAM:

流程圖:

(框起來的是目標層，**通常**是最後一個 Conv)



公式:

$$L_{\text{Grad-CAM}}^c = ReLU\underbrace{\left(\sum_k \alpha_k^c A^k\right)}_{\text{linear combination}} \qquad \alpha_k^c = \overbrace{\frac{1}{Z}\sum_i \sum_j}^{\text{global average pooling}} \underbrace{\frac{\partial y^c}{\partial A_{ij}^k}}_{\text{gradients via backprop}}$$

c: class

y^c: score of c

A: feature map activations of a convolutional layer
k: depth
L: heatmap

實現步驟:
1. 計算圖像的模型輸出和最後的捲積層輸出
2. 在模型輸出中找到預測類別的索引(前面設為 label_index)
3. 計算預測類和目標層的梯度。
4. 取全局平均，然後與最後一個卷積層相乘
5. 標準化 heatmap，轉換為 RGB 並將其疊加在原始圖像上

```
from google.colab.patches import cv2_imshow #cv2.imshow 在 Colab 上不起作用，改用 cv2_imshow

def Grad_CAM(orig,layer_name='block5_conv3', intensity=0.5, res=250):
  img=skimage.io.imread(orig)
  img = cv2.resize(img, (dim, dim), interpolation=cv2.INTER_NEAREST)
  x = img.astype(np.float32)
  x = np.expand_dims(x, 0)
  x = preprocess_input(x)
  preds = model.predict(x)
  print(decode_predictions(preds)[0][0][1]) #印出圖片的預測

#定義一個 tf.GradientTape，以便計算梯度（這是 tf.2 中的用法，tf.1 直接用 K.gradient 就可以了）
  with tf.GradientTape() as tape:
    conv_layer = model.get_layer(layer_name)#最後一個卷積層
    iterate = models.Model([model.inputs], [model.output, conv_layer.outpu
t])#列表:(圖像,模型輸出&最終一個卷積層)
    model_out, conv_layer = iterate(x)
    class_out = model_out[:, np.argmax(model_out[0])]
    grads = tape.gradient(class_out, conv_layer)#預測輸出和最後一個捲積的梯度
    pooled_grads = K.mean(grads, axis=(0, 1, 2))#在所有軸間求平均

  heatmap = tf.reduce_mean(tf.multiply(pooled_grads, conv_layer), axis=-1)
  heatmap = np.maximum(heatmap, 0)#ReLU
  #normalize
  max_heat = np.max(heatmap)
  if max_heat == 0:
      max_heat = 1e-10
  heatmap /= max_heat
  heatmap=np.squeeze(heatmap)
```

```
  plt.show()

  #結合熱圖跟用來預測的圖片
  img = cv2.imread(orig)
  heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0]))
  heatmap = cv2.applyColorMap(np.uint8(255*heatmap), cv2.COLORMAP_JET)
  img = heatmap * intensity + img
  #im2=cv2_imshow(cv2.resize(cv2.imread(orig), (res, res)))#原圖
  im3=cv2_imshow(cv2.resize(img, (res, res)))#gradCAM 圖


'''
論文公式:
```

$$\alpha_{ij}^{kc} = \frac{\frac{\partial^2 Y^c}{(\partial A_{ij}^k)^2}}{2\frac{\partial^2 Y^c}{(\partial A_{ij}^k)^2} + \sum_a \sum_b A_{ab}^k \{\frac{\partial^3 Y^c}{(\partial A_{ij}^k)^3}\}}$$

$$w_k^c = \sum_i \sum_j \alpha_{ij}^{kc}.relu(\frac{\partial Y^c}{\partial A_{ij}^k})$$

$$L_{ij}^c = relu(\sum_k w_k^c.A_{ij}^k)$$

```
Grad-CAM++是基於 Grad-CAM 的改良版，論文中說明與當時最好的方法相比，更好的解釋了目標定位和單個圖形上出現多個目標實例的解釋。
Grad-CAM++和 Grad-CAM 的差異在於 alpha 的計算方式不同
'''
#大致還原論文公式
def Grad_CAM_plus(orig,layer_name='block5_conv3',intensity=0.5,res=250):
  img=skimage.io.imread(orig)
  img = cv2.resize(img, (dim, dim), interpolation=cv2.INTER_NEAREST)
  x = img.astype(np.float32)
  x = np.expand_dims(x, 0)
  x = preprocess_input(x)
  preds = model.predict(x)
  print(decode_predictions(preds)[0][0][1]) #印出圖片的預測
```

```
  conv_layer = model.get_layer(layer_name)
  cam = models.Model([model.inputs], [conv_layer.output,model.output ])

  with tf.GradientTape() as gtape1:
      with tf.GradientTape() as gtape2:
          with tf.GradientTape() as gtape3:
            conv_layer,model_outs  = cam(x)
            class_out = model_outs[:, np.argmax(model_outs[0])]
            grad_1st = gtape3.gradient(class_out, conv_layer)
          grad_2nd = gtape2.gradient(grad_1st, conv_layer)
      grad_3rd = gtape1.gradient(grad_2nd, conv_layer)

  #alpha
  #分子
  alpha_num = grad_2nd[0]
  #分母
  global_sum = np.sum(conv_layer, axis=(0, 1, 2))
  alpha_denom =(2.0*alpha_num) + (global_sum*grad_3rd[0])
  alpha_denom = np.where(alpha_denom != 0.0, alpha_denom,1e-10)
   #np.where(condition,out1,out2)滿足 condition 輸出 out1，不滿足輸出 out2
   #normalize alpha
  alpha = alpha_num/alpha_denom
  alpha_normalization_constant = np.sum(alpha, axis=(0,1))
  alpha /= alpha_normalization_constant

  w = np.multiply(np.maximum(grad_1st[0], 0.0),alpha)
  w=np.sum(w,axis=(0,1))
#有的圖片拿來測試時在 sum 的步驟算完後出現 Nan，導致後續計算出全 Nan 的 heatmap，在此把
Nan 的部分替換成 0
  w[np.isnan(w)]=0.0

  heatmap = np.sum(w*conv_layer[0], axis=-1)
  heatmap = np.maximum(heatmap, 0)
  max_heat = np.max(heatmap)
  if max_heat == 0:
    max_heat = 1e-10
  heatmap /= max_heat
  plt.matshow(heatmap)

  #結合熱圖跟用來預測的圖片
  heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0]))
  heatmap = cv2.applyColorMap(np.uint8(255*heatmap), cv2.COLORMAP_JET)
  img = heatmap * intensity + img
```

```
    #show image.
    #cv2_imshow(cv2.resize(cv2.imread(orig), (res, res)))#原圖
    cv2_imshow(cv2.resize(img, (res, res)))

#grad-cam 跟 grad-cam++比較
Grad_CAM('husky1.jpg')
Grad_CAM_plus('husky1.jpg')
Grad_CAM('husky3.jpg')
Grad_CAM_plus('husky3.jpg')
Grad_CAM('husky7.jpg')
Grad_CAM_plus('husky7.jpg')
#dog.JPG 是從論文的圖片截圖下來的，測試 grad-cam++代碼寫的符不符合
Grad_CAM('dog.JPG')
Grad_CAM_plus('dog.JPG')
#多目標測試
Grad_CAM('many.jpg')
Grad_CAM_plus('many.jpg')
```

結果：

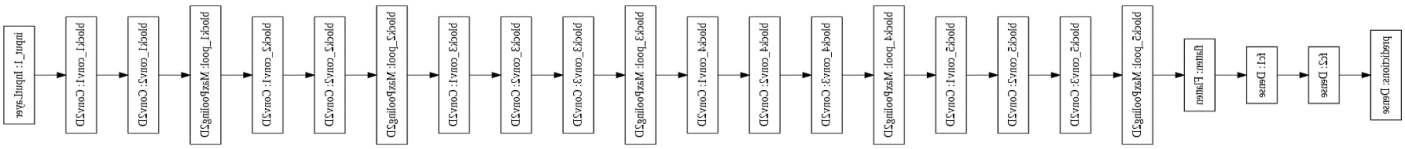
1. Model:

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels.h5
553467904/553467096 [==============================] - 19s 0us/step
```

VGG16 模型結構:

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
```

```
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
_________________________________________________________________
```



2. 圖片：

shape: (224, 224, 3) min: 5 max: 255

3. 預測

A. 改變圖片輸入維度:

shape: (1, 224, 224, 3) min: -116.68000030517578 max: 151.06100463867188

B. 圖片預測出的標籤:

標籤 label index: 249

C. 預測前三類標籤跟機率:

Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/imagenet_class_index.json

40960/35363 [==============================] - 0s 0us/step

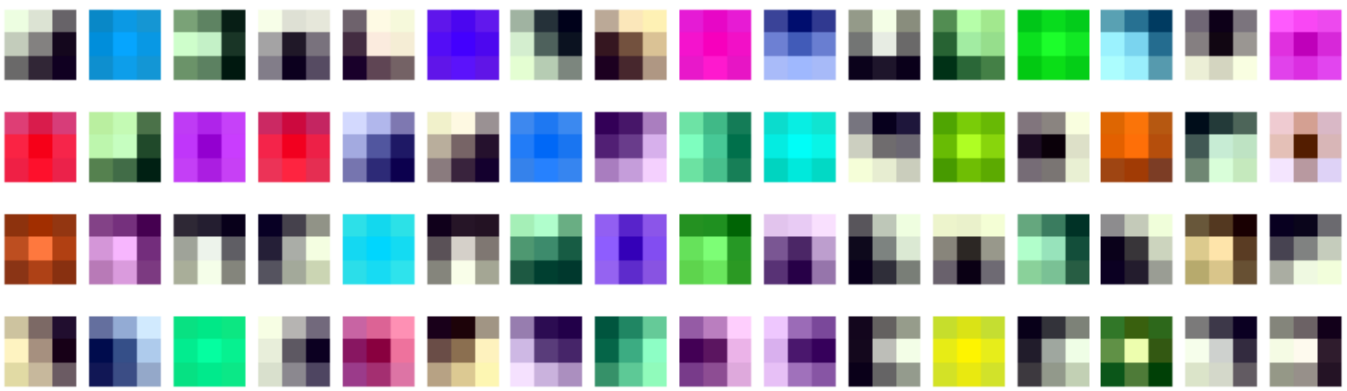
```
[[('n02110063', 'malamute', 0.7351754),
  ('n02110185', 'Siberian_husky', 0.17270559),
  ('n02109961', 'Eskimo_dog', 0.08778877)]]
```

說明: 1.阿拉斯加雪橇犬 73.5% 2.西伯利亞哈士奇 17.2% 3.愛斯基摩犬 8%

4. Visualize:

A. Filter

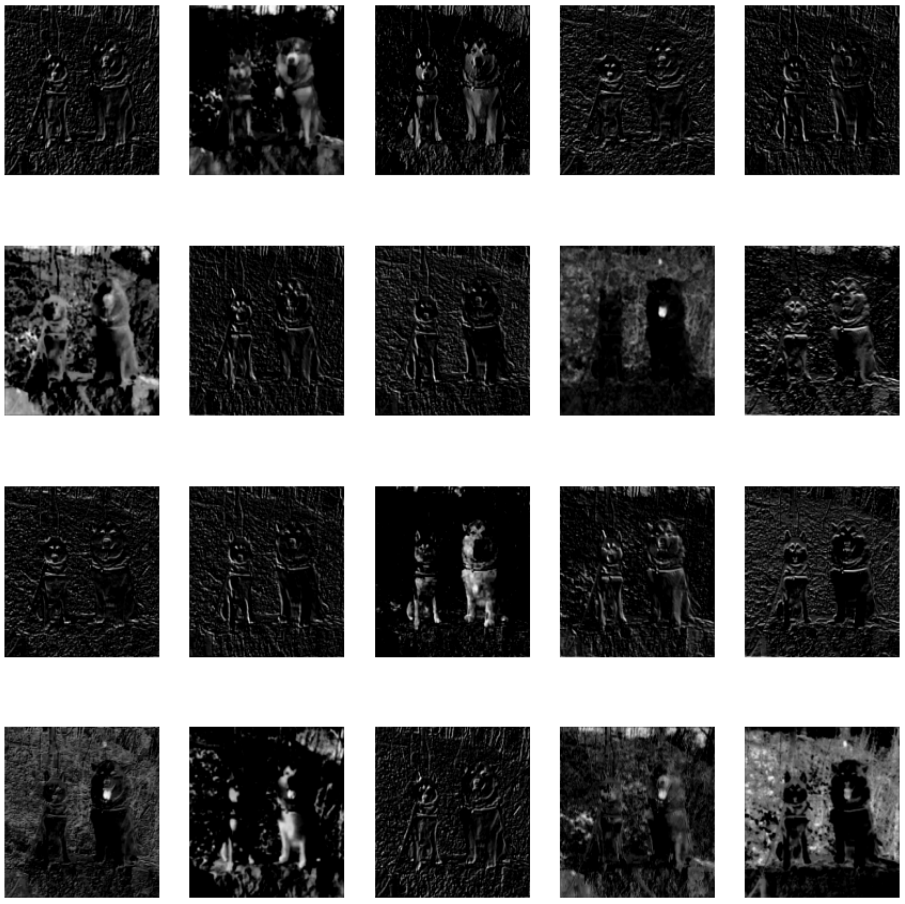
說明:VGG16 第一個卷積的 filter 用來提取圖片顏色，顏色越深表示越重要。

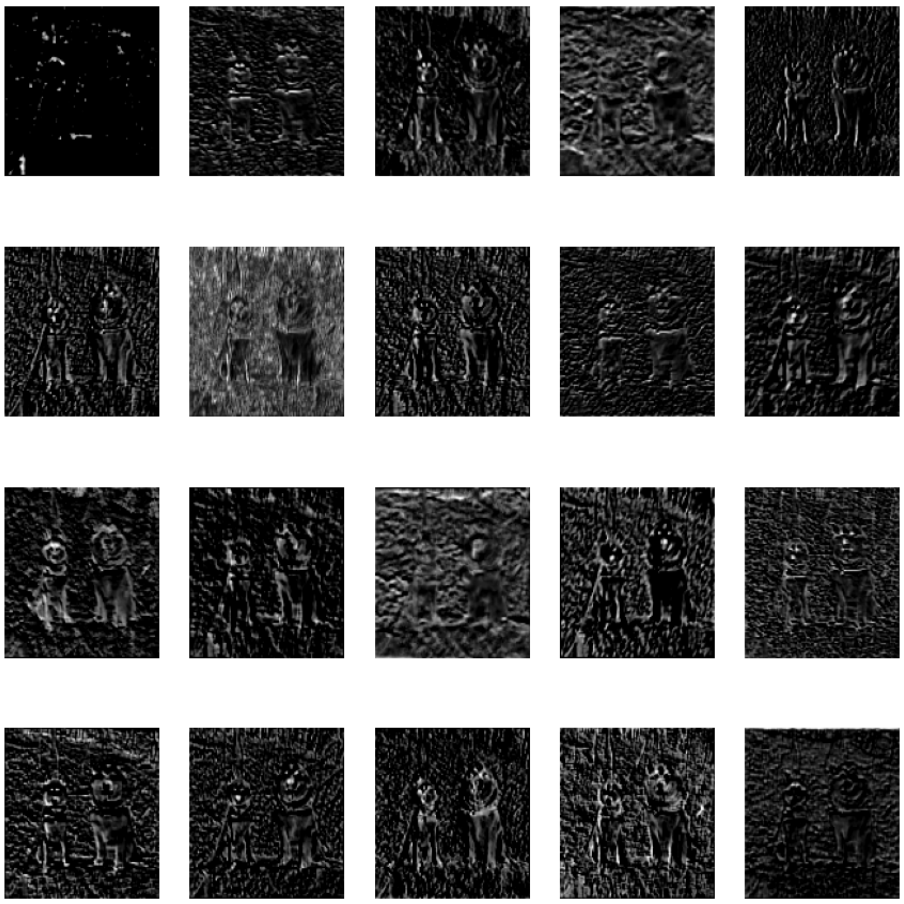
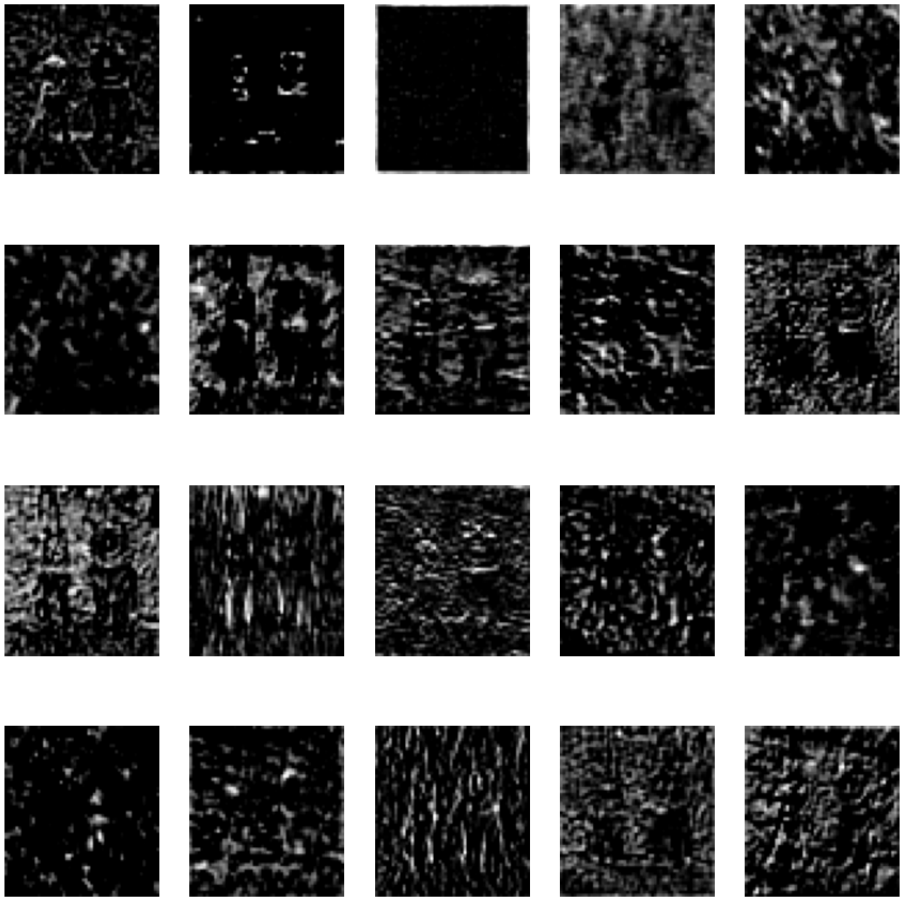


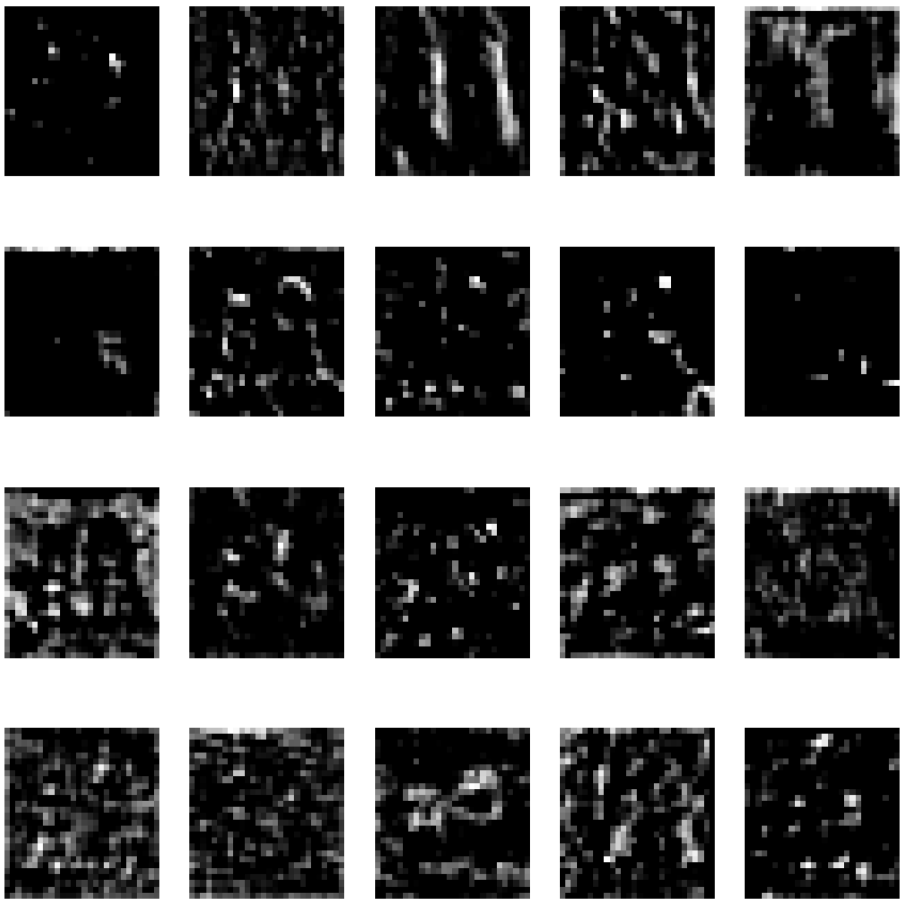
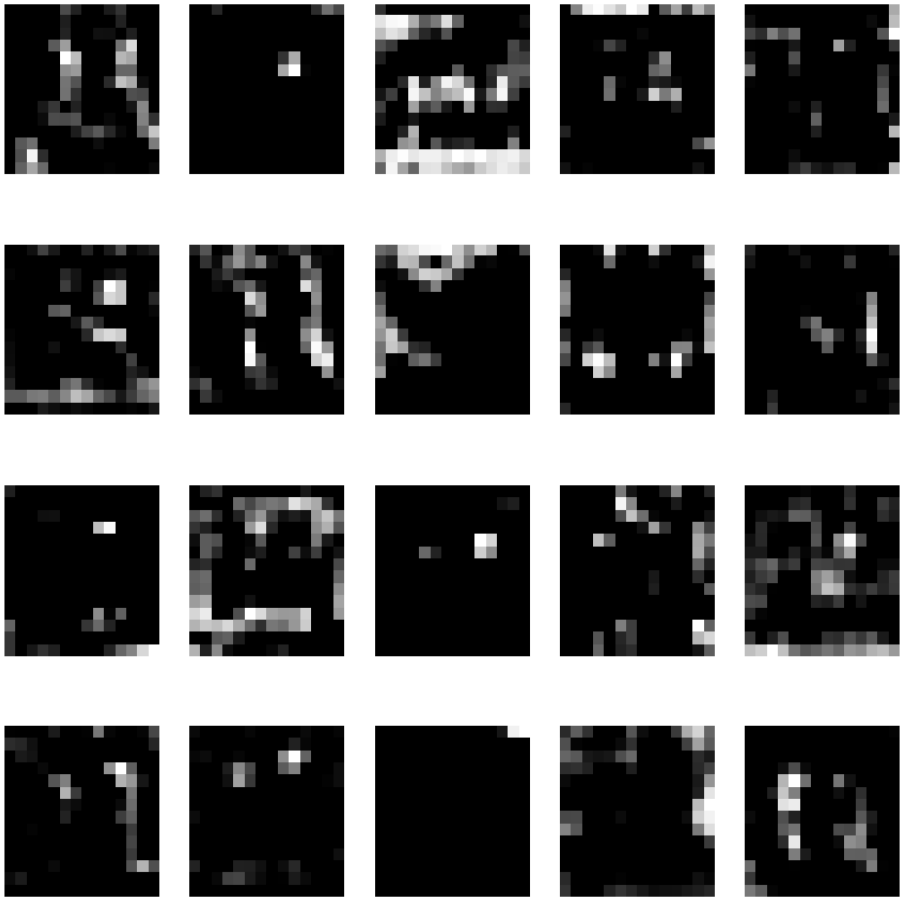
shape: (3, 3, 3, 64) min: -0.6714000701904297 max: 0.6085159182548523

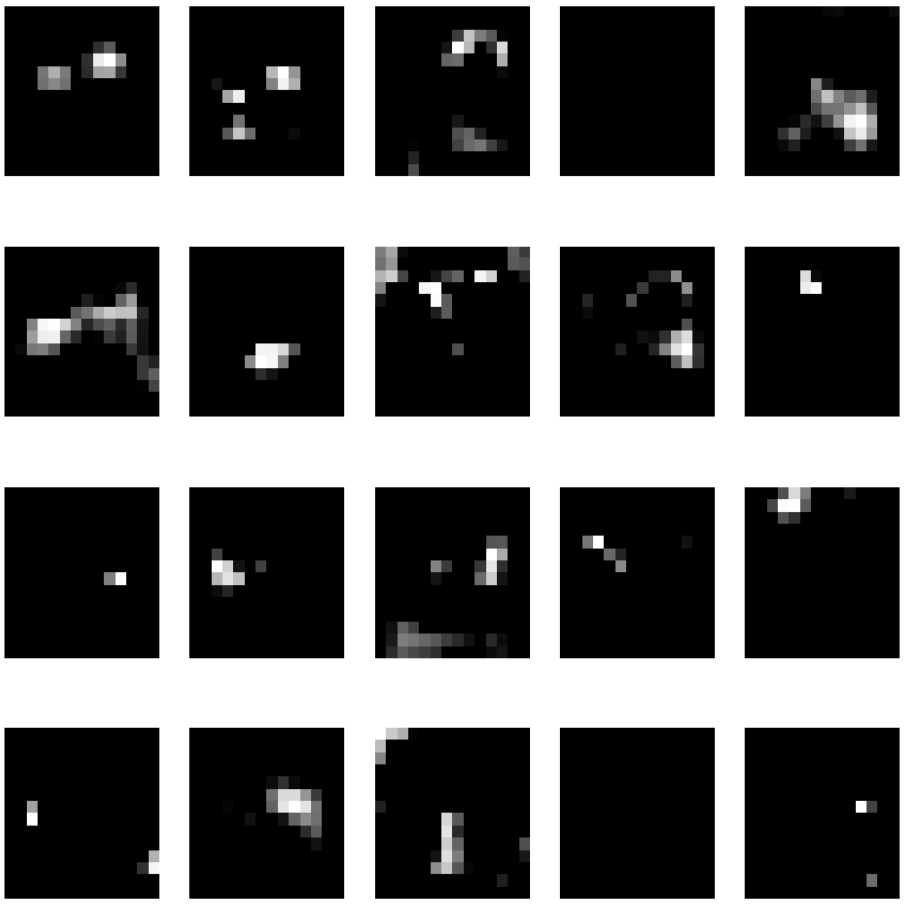
B. Feature map

說明:由表內的圖可見，隨著模型深度越深，原本還看得出原圖的輪廓，越後面越看不出原貌，但對特徵的提取也越精細。

block1_conv1	 shape: (1, 224, 224, 64) min: 0.0 max: 676.1041259765625

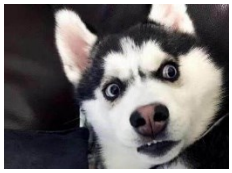
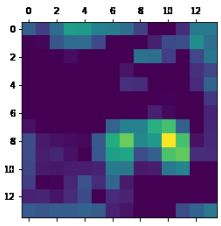
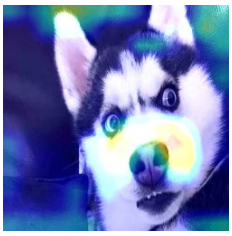
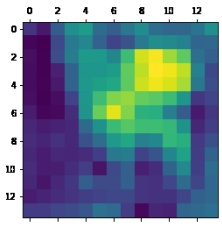
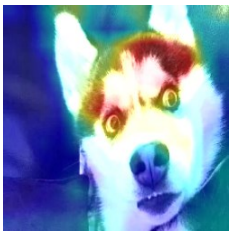
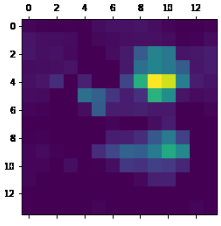
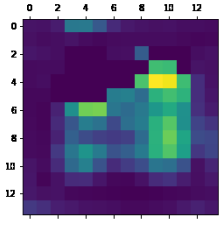
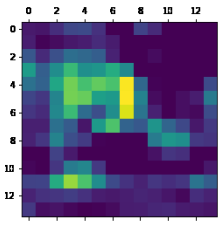
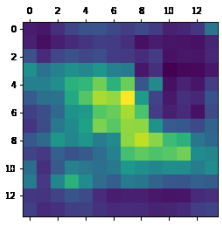
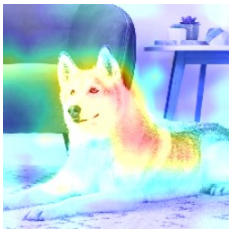
block2_conv1	 shape: (1, 112, 112, 128) min: 0.0 max: 4251.1103515625
block3_conv1	 shape: (1, 56, 56, 256) min: 0.0 max: 9868.501953125

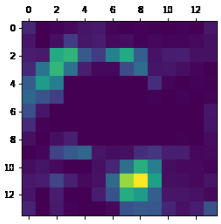
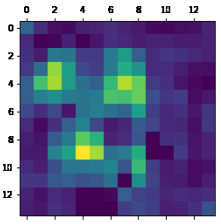
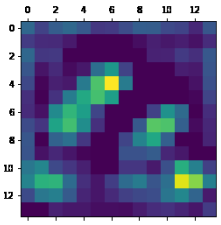
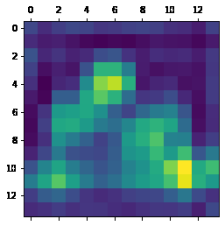
block4_conv1	 shape: (1, 28, 28, 512) min: 0.0 max: 7569.845703125
block5_conv1	 shape: (1, 14, 14, 512) min: 0.0 max: 2017.851806640625

block5_conv3	 shape: (1, 14, 14, 512) min: 0.0 max: 319.8306579589844

C. 熱圖可視化方法比較

說明:熱圖是用來了解圖片中哪些部分對於預測出此結果是比較重要的，顏色越紅表示越重要。從下表可看出，Grad-CAM++對於多目標的圖片解釋性的確比較好。

Original picture &prediction	Grad-CAM		Grad-CAM++	
Siberian_husky 				
malamute 				
Eskimo_dog				

				
Border_collie 				
Siberian_husky 				

第二部分說明:

Grad-CAM 和 Grad-CAM++兩者的概念都是以"目標層"得到的輸入會是最接近分類特徵的。論文中以及關於這兩者的分享文章都是直接用 model. summary()裡面 conv2D 數字最大的，也是用最後一個卷積(VGG16 用 block5_conv3)。但我有個疑問，對於模型的結構不是跟 VGG 一樣是一條直線到最後的結構，這樣有分支跟殘差結構模型的目標層要如何選擇。會有這樣的疑問是因為在殘差還有分支結構提出後，模型的卷積會持續分開跟連結，可能會有最後面試好幾個卷積卷完再合併，或是數字最大的卷積不是最深的情況，而且也想要了解其他層學到特徵的重要度。

關於不同的位置的目標層選擇會有什麼樣的影響，在 Grad-CAM 這篇論文的後面有提到，他們在基於 ResNet 的 VQA 模型可視化時發現 Grad-CAM 中大部分相鄰層次的微小變化以及涉及降維的層次之間有較大的變化。而在 Grad-CAM++則是使用 AlexNet 跟 VGG16 來測試，主要還是以不同數據集來測試，文末說明關於其他類型的模型只提說希望可以延伸使用。

以下使用不同目標層以及不同結構的模型再做進一步探討。

II. Visualization of other pre-trained models with ImageNet datas.

(在以上 VGG16 的代碼基礎上，把標示的的部分做更改，比較不同模型、不同目標層的 heatmap)

(僅示更改部分)

程式碼:

1. ResNet50，有殘差結構

- `from keras.applications import ResNet50`
- `from keras.applications.resnet50 import preprocess_input, decode_predictions`

- `model = ResNet50(weights='imagenet')`

- `layer_name=' conv5_block3_3_conv'`
- `layer_name='conv5_block3_add'`
- `layer_name='conv5_block3_out'`

2. InceptionV3，分支訓練，重複堆疊結構
   - `from tensorflow.keras.applications.inception_v3 import InceptionV3`
   - `from tensorflow.keras.applications.inception_v3 import preprocess_input, decode_predictions`

   - `model = InceptionV3(weights='imagenet')`

   - `dim=299`

   - `layer_name='conv2d_85'`
   - `layer_name='conv2d_88'`
   - `layer_name='conv2d_91'`
   - `layer_name='conv2d_92'`
   - `layer_name='conv2d_93'`
   - `layer_name='mixed10'`

3. NASNetLarge，結合殘差跟分支、重複堆疊的結構
   - `from keras.applications.nasnet import NASNetLarge`
   - `from keras.applications.nasnet import preprocess_input, decode_predictions`

   - `model = NASNetLarge(weights='imagenet')`

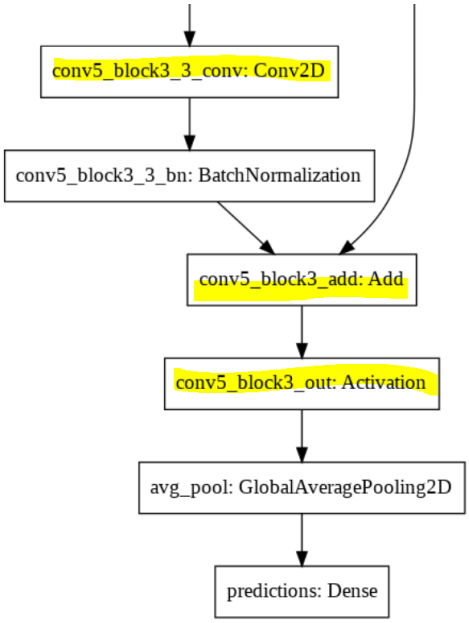
   - `dim=331`

   - `layer_name=' separable_conv_2_normal_left2_18'`
   - `layer_name=' separable_conv_2_normal_right2_18'`
   - `layer_name=' separable_conv_2_normal_right1_18'`
   - `layer_name=' separable_conv_2_normal_left1_18'`
   - `layer_name=' separable_conv_2_normal_left5_18'`
   - `layer_name='normal_concat_18'`

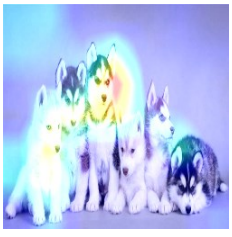
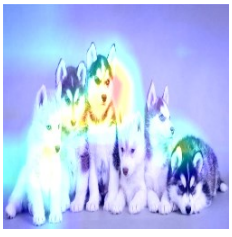
結果：

A. ResNet 不同目標層比較

部分結構圖（標註的是選擇用來測試的目標層）：



比較：

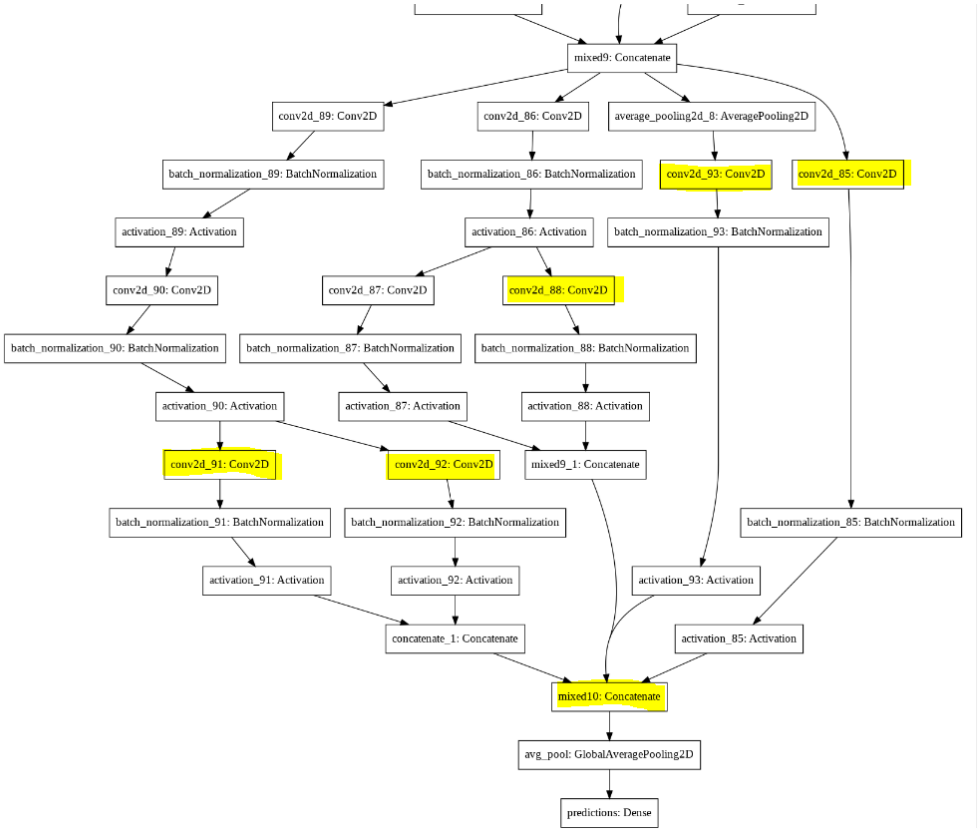
layer name	conv5_block3_3_conv	conv5_block3_add	conv5_block3_out
Grad-CAM			
Grad-CAM++			
Grad-CAM			
Grad-CAM++			
Grad-CAM			

Grad-CAM++	shape: (7, 7) min: 0.0 max: 0.0		

說明：從上表大致可看出，最後一層卷積所學到的新特徵和沒學到新特徵的舊特徵加在一起後，再經過一層 activation，各個階段的熱圖差異。conv5_block3_out 這個目標層所涵蓋的範圍最廣，但對於最後一個 row 的部分我有點迷惑，感覺像突然蹦出來的。

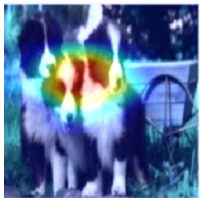
B. InceptionV3 不同目標層比較:

部分結構圖（標註的是選擇用來測試的目標層）：



比較：

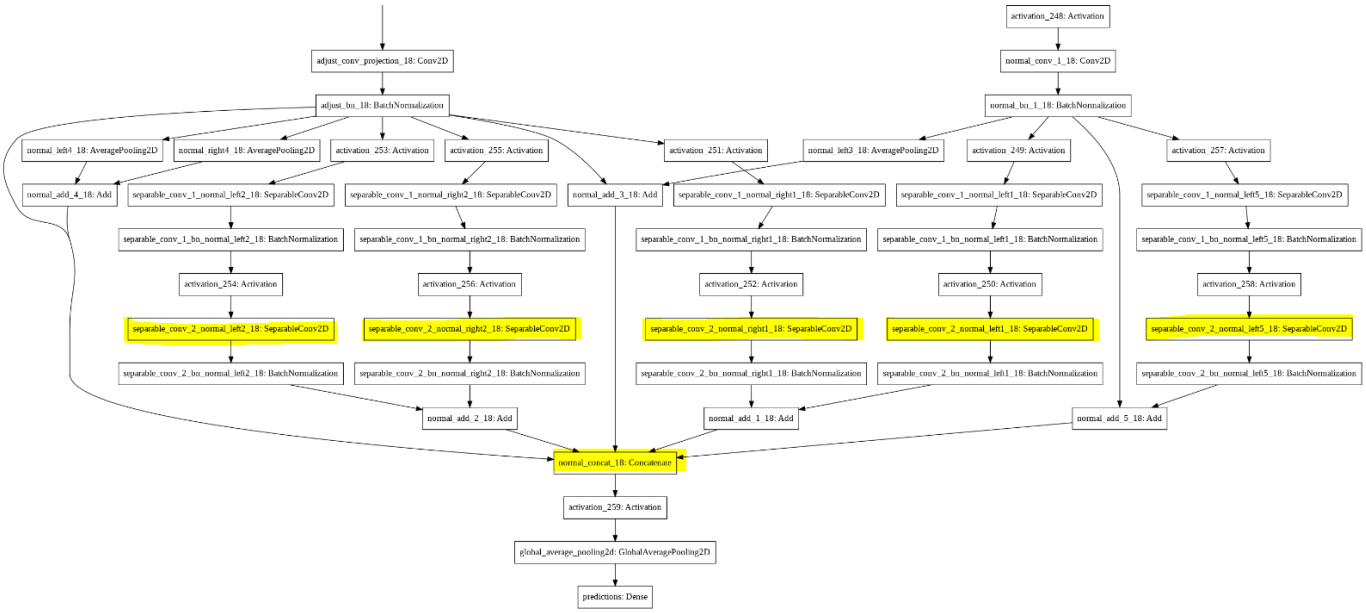
	conv2d_85	conv2d_88	conv2d_91	conv2d_92	conv2d_93	mixed10
G C						

GC++						
GC						
GC++						
GC						
GC++						

說明：在上一個連接之後，85 是直接接 BN 跟 Activation 就到連接層，93 比 85 的前面多加一個 AP，而 88、91、92 的前面還有經過其他卷積，最後都接到 mixed10。在幾種不同學習特徵的方式下，從上表的 heatmap 看出每種的情況。透過 85 跟 93 的比較可以了解在前方加了平均池化層後的效果不錯，而 88、91、92 也學習不少新的特徵。對於數字最大的卷積(93)跟最接近 Softmax 的連接(mixed10)，一個範圍縮得比較精確，一個是幾乎包含該目標的整體，如果只是要了解模型是根據圖片哪個部分來預測的，我覺得這兩者(93&mixed10)皆可。

C. NASNetLarge 不同目標層比較:

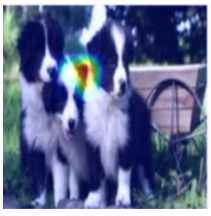
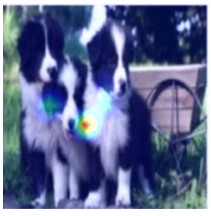
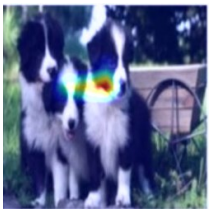
部分結構圖（標註的是選擇用來測試的目標層）：



比較：

(layer name 順序以圖示來看，從左到右，從上到下)

-: shape: (11, 11) min: 0.0 max: 0.0

GC	-			-		
GC++	-	-	-	-		
GC			-	-		
GC++	-	-	-	-		
GC		-	-	-		

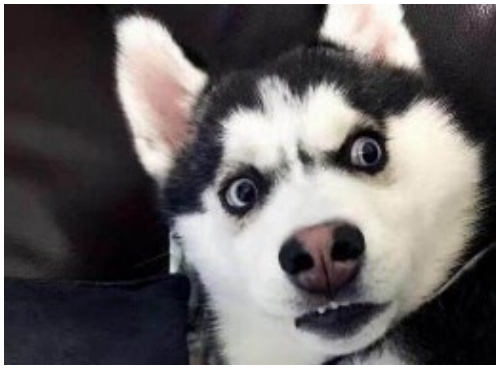
G C + +	-	-	-	-		

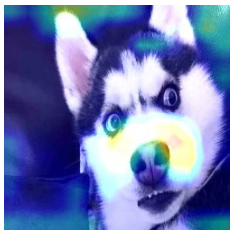
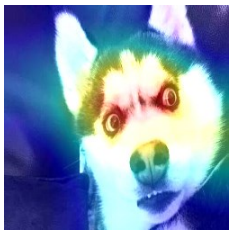
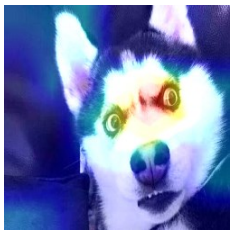
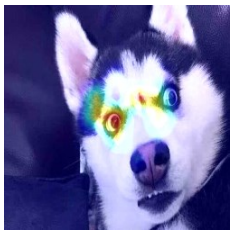
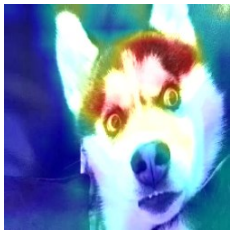
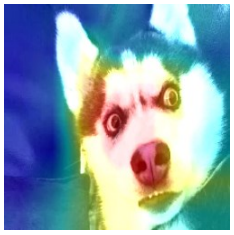
說明：從 heatmap 看出，分開訓練的卷積各學習到的特徵不同，有的沒有學習到新的特徵，最後連接層把前面幾個分支的特徵結合在一起。對於這類型的結構，我覺得比起把卷積層當作目標層，使用最後一個連接層會更能瞭解該模型對於圖片特徵提取的位置。

D. 不同模型比較:

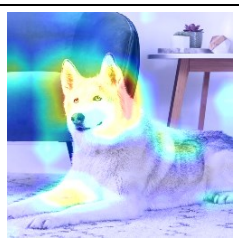
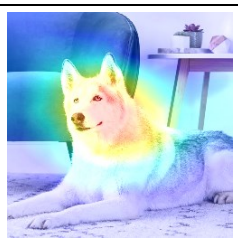
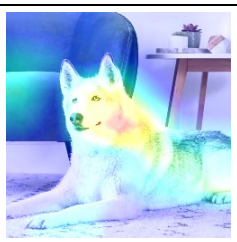
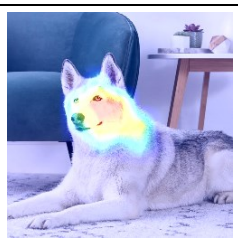
i. 不同模型提取特徵的方式都不同，對於不同圖片在各個模型裡的熱圖來看，要預判出是某類的特徵大致是差不多的，只是各個模型判出的重要度範圍不同。

ii. 下面以與目標覆蓋面積以及紅色範圍為標準再進行比較。

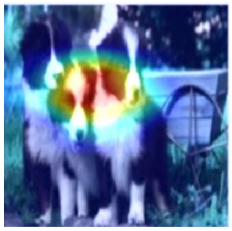


Model & layer name	VGG16 block5_conv3	InceptionV3 conv2d_93	ResNet50 conv5_block3_out	NASNetLarge normal_concat_18
prediction	Siberian_husky	Siberian_husky	Siberian_husky	Siberian_husky
Grad-CAM				
	比較:InceptionV3 > ResNet50 > NASNetLarge > VGG16			
Grad-CAM++				
	比較: ResNet50 > VGG16 > InceptionV3 > NASNetLarge			

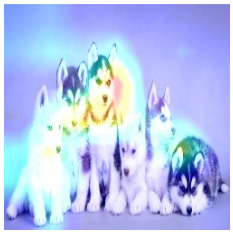
Model & layer name	VGG16 block5_conv3	InceptionV3 conv2d_93	ResNet50 conv5_block3_out	NASNetLarge normal_concat_18
prediction	Malamute	Malamute	Malamute	Malamute
Grad-CAM				
	比較:InceptionV3 > ResNet50 > VGG16 > NASNetLarge			
Grad-CAM++				
	比較:InceptionV3 ≧ ResNet50 > VGG16 > NASNetLarge			

Model & layer name	VGG16 block5_conv3	InceptionV3 conv2d_93	ResNet50 conv5_block3_out	NASNetLarge normal_concat_18
prediction	Eskimo_dog	Eskimo_dog	Eskimo_dog	Eskimo_dog
Grad-CAM				

	比較: InceptionV3 > ResNet50 > VGG16> NASNetLarge			
Grad-CAM++				
	比較: VGG16 > ResNet50 ≧ InceptionV3 > NASNetLarge			

Model & layer name	VGG16 block5_conv3	InceptionV3 conv2d_93	ResNet50 conv5_block3_out	NASNetLarge normal_concat_18
prediction	Border_collie	Border_collie	hare	Border_collie
Grad-CAM				
	比較: InceptionV3 > NASNetLarge > VGG16 ,ResNet50(預測錯誤)			
Grad-CAM++				
	比較: ResNet50 > InceptionV3 > VGG16> NASNetLarge			

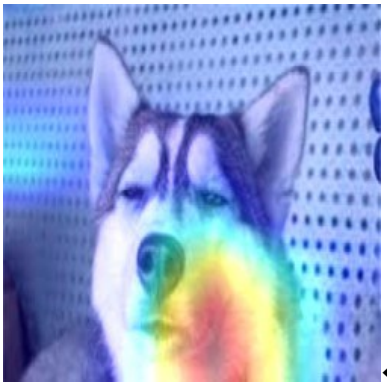
Model & layer name	VGG16 block5_conv3	InceptionV3 conv2d_93	ResNet50 conv5_block3_out	NASNetLarge normal_concat_18

prediction	Siberian_husky	Siberian_husky	Siberian_husky	Siberian_husky
Grad-CAM				
	比較: InceptionV3 > ResNet50 > VGG16 ≧ NASNetLarge			
Grad-CAM++				
	比較: ResNet50 > VGG16>InceptionV3 > NASNetLarge			

結果: 以覆蓋面積來說，InceptionV3 跟 ResNet50 對於目標的覆蓋率都很大，但有幾張偏紅的部分標示的地方有部分並不是目標，還有預測錯誤成另一種動物的。另一方面，VGG16 和 NASNetLarge 的熱圖覆蓋比較零散，但是重要特徵紅區是在正確的位置，像是在告訴我們如何分辨 A 品種跟 B 品種。

心得:

其實會寫這個主題純粹是意外，一開始我只是想找把特徵圖畫出來的方法，看一下我嘗試寫的模型到底怎麼把飛機認成青蛙的，結果在可視化研究裡面發現了不少有趣的方法。使用反傳播法來可視化的方法其實非常多，不僅有在前言提到的幾個。在這些之中我會選擇 heatmap 來進一步研究的原因是因為在嘗試幾種反傳播可視化的方法時，發生了很奇妙的事情(如圖)。


←疑似遭到毆打


←花圈?

由於論文提供的程式是用 tensorflow1.X 的版本，而相關的分享文章都是用 pytorch，我再次深深感受到版本不兼容的尷尬之處。弄出這兩張"效果"圖的時候其實我還在嘗試用 tensorflow2.X 版寫出 grad-cam++的程式碼。這段期間，製造出了不少奇形怪狀的圖片。在經歷了打錯層數、梯度算到沒數值、矩陣算到變成 Nan 等等意外後，我終於做出了理想中的圖，但也在錯誤中發現了如果是用熱圖的話更能清楚的感受到對於增加某些特殊層的效果。