Computer vision with Keras

...

Content

- Libraries installation
- Neural network intro.
- Image
- Neural network structure
- Sequential model
- Conv2D layer
- Feature map
- Max Pooling layer
- Flatten and Dense layers
- Image classification model using VGG16

Libraries needed

- TensorFlow:

  A framework from Google have many libraries can be used in computer vision tasks.

- Keras:

  An API works on top of tensorflow.

- OpenCV:

  Library used to perform image processing and computer vision tasks.

- Installation:

```
from tensorflow import keras
import cv2
```

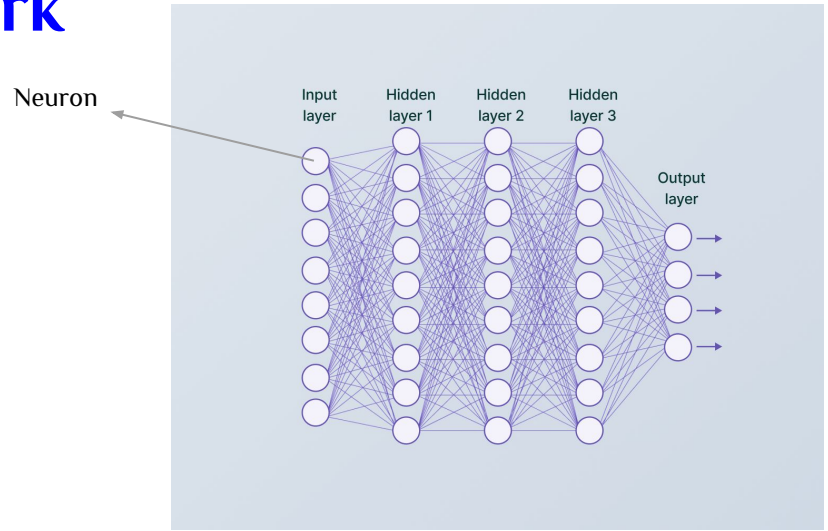
- Checking version:

```
print(tensorflow.__version__)
print(keras.__version__)
print(cv2.__version__)
```

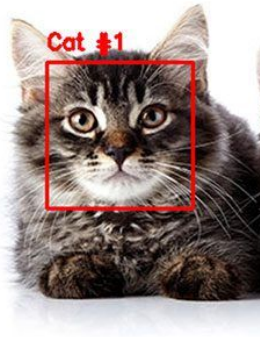
Content

- ~~Libraries installation~~
- Neural network intro.
- Image
- Neural network structure
- Sequential model
- Conv2D layer
- Feature map
- Max Pooling layer
- Flatten and Dense layers
- Image classification model using VGG16

Neural Network



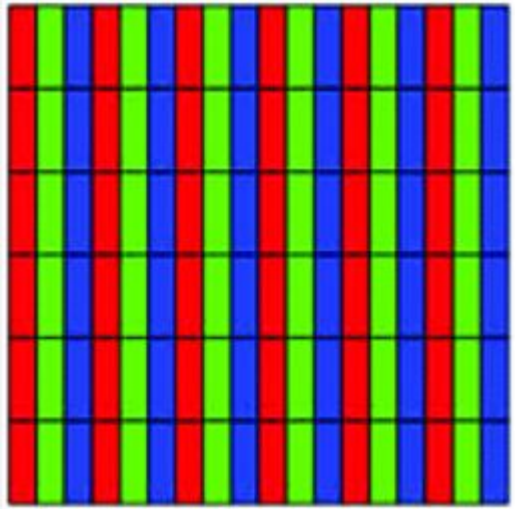
It consists of many layer, the first layer takes the input image, then the image passes into one/more hidden layers to perform some process on it, then the output layer returns the input image concatenated with the specified task. For example, if we are performing a classification task to return if the image if for cat or dog, it will return the image of the animal concatenated with a boundary box around the animal with a text of the type of this animal either cat or dog.



Content

- ~~Libraries installation~~
- ~~Neural network intro.~~
- Image
- Neural network structure
- Sequential model
- Conv2D layer
- Feature map
- Max Pooling layer
- Flatten and Dense layers
- Image classification model using VGG16

Image



It is a list of rows and columns, each pixel (row, column) represents the brightness of each main color, in other words, represents the intensity of the RGB colors (Red Green Blue)

Ex:

```
[[[216 170  69]   (Row0, column 0) [R G B] -> Pixel 0
  [216 170  69]   (Row 0, column 1) [R G B] -> Pixel 1
  [216 170  69]   (Row 0, column 2) [R G B] -> Pixel 2
  ...             .....
  [202 159  80]   (Row 0, column n-2) [R G B] -> pixel n-2
  [197 159  87]   (Row 0, column n-1) [R G B] -> pixel n-1
  [197 159  94]]  (Row 0, column n) [R G B] -> pixel n

 [[214 171  62]   (Row 1, column 0) [R G B]
  [214 171  62]   (Row 1, column 1) [R G B]
  [214 170  63]   (Row 1, column 2) [R G B]
  ...             .....
```

To know information about the image:

```
14  # Image info
15  print(img.shape)  # Size
16  print(img)  # List
```

In detection task, there is no need for colors. if we want to detect something in an image, we can convert the input image into grayscale image which makes the image in black and white only which is one channel of RGB, this results that each pixel will have one value instead of three.

- Reading the image in grayscale:

```
18  # Read the image in gray scale
19  grayImg = cv2.imread("Car.png", cv2.IMREAD_GRAYSCALE)
20  cv2.imshow("Car image in gray scale", grayImg)
21  # cv2.waitKey(0)
22  print(grayImg.shape)
23  print(grayImg)
```

- Img output:

```
[[145 145 145 ... 140 141 143]
 [143 143 143 ... 139 140 142]
 [140 141 140 ... 139 141 143]
 ...
```

This image will be the input layer of neural network.

Content

- ~~Libraries installation~~
- ~~Neural network intro.~~
- ~~Image~~
- Neural network structure
- Sequential model
- Conv2D layer
- Feature map
- Max Pooling layer
- Flatten and Dense layers
- Image classification model using VGG16

Neural Network structure

❖ How to append layer/s into the neural network?

This is done using function `Input()` and passing the shape of the input image into its shape parameter. If we print the shape of the input layer, it will be the same as the shape of the input image, but will have another dimension. This dimension refers to number of batches, which are images inputted into the input layer.

```
25  # Passing the input layer into the neural network
26  rows, columns = grayImg.shape  # Image height and width
27  input_layer = keras.Input(shape=(rows, columns))
```

After the input layer, there is/are hidden layer/s where the processing is done, one of these layers is called “dense” layer, to append it to the neural network, we have to create it by passing the number of neurons(depends on the model itself), and then connect it to the input layer.

```
29  # Append Dense layer
30  from keras.layers import Dense
31  layer_1 = Dense(64)(input_layer)
32  layer_2 = Dense(32)(layer_1)
33  output = Dense(2)(layer_2)
```

These layers and connection between them is called Neural Network Structure.

```
29      # Neural network structure
30      input_layer = keras.Input(shape=(rows, columns))
31      layer_1 = Dense(64)(input_layer)
32      layer_2 = Dense(32)(layer_1)
33      output = Dense(2)(layer_2)
```

After building our neural network layers, we have to define the model to apply on it, by passing the input layer and output layer as parameters to the `Model()` function.

```
35      # Define the model
36      model = keras.Model(inputs=input_layer, outputs=output)
```

To get a summary of the model, layers and shapes:

```
38    # Model summary to make sure that everything is on
39    model.summary()
```

The output:

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 410, 728)]        0

 dense (Dense)               (None, 410, 64)           46656

 dense_1 (Dense)             (None, 410, 32)           2080

 dense_2 (Dense)             (None, 410, 2)            66

=================================================================
```

Code

Content

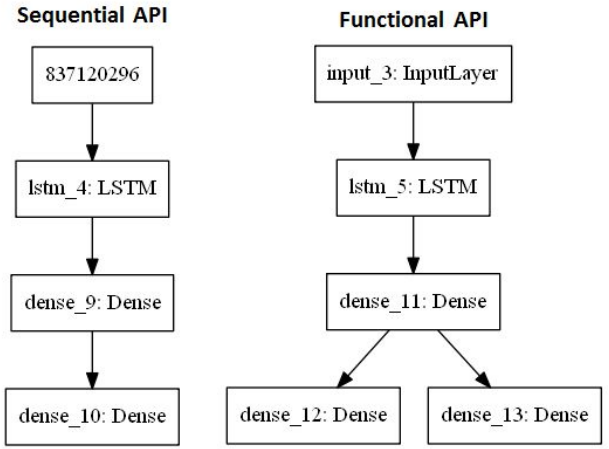
- ~~Libraries installation~~
- ~~Neural network intro.~~
- ~~Image~~
- ~~Neural network structure~~
- Sequential model
- Conv2D layer
- Feature map
- Max Pooling layer
- Flatten and Dense layers
- Image classification model using VGG16

Sequential model

Keras provides two ways to create the model, Functional API model and Sequential API model. Sequential API model is simpler but less flexible than functional API model.

In sequential model, we need a sequence of layers of the top of eachother, so if we want for example to split the model on the 2nd layer and separately use this layer to connect to other layers, we can not do this with sequential model, we will need to use functional model. [This is called layers sharing and branching].

Most of the time in computer vision, for example in YOLO models and other models, we use the sequential API models.



How to build the sequential API model?

```
12  # Create a sequential API model
13  model = keras.Sequential()
14  model.add(layers.Input(shape=(height, width, channels)))
15  model.add(layers.Dense(32))
16  model.add(layers.Dense(16))  # Adding a new layer on the top of the first layer and connect between them
17  model.add(layers.Dense(2))
```

To make sure that our neural network is running well without any errors, we need to pass our image into the model.

```
21  # Passing our image into the model
22  result = model(preprocessed_img)
23  print(result)
```

The output of the model will be multi-dimensional list which doesn't represent anything rightnow, but it will have meanings once we use a real model for specific task like detection, classification, etc...

We may get an error if the shape of our image doesn't match the expected shape to the model, so we may need a simple preprocessing step to change the image shape. To do this, we simply put the image inside an array.

```
18    # Change the size of image if needed
19    preprocessed_img = np.array([img])
```

Output of the model be like:

```
tf.Tensor(
[[[[105.110405   74.165985 ]
   [105.110405   74.165985 ]
   [105.110405   74.165985 ]
   ...
   [101.87345    74.700935 ]
   [101.28979    74.97924  ]
   [102.90784    77.39818  ]]

  [[102.57669    70.64288  ]
   [102.57669    70.64288  ]
   [102.84284    71.236374 ]
   ...
```

Code

Content

- ~~Libraries installation~~
- ~~Neural network intro.~~
- ~~Image~~
- ~~Neural network structure~~
- ~~Sequential model~~
- Conv2D layer
- Feature map
- Max Pooling layer
- Flatten and Dense layers
- Image classification model using VGG16

Conv2D Layer

Introduction

- The convolution operation is used in computer vision usually to extract information from the image or to manipulate the image (extract features like horizontal/ vertical/ etc.. edges & manipulation like blurring).
- In the specific case of convolution neural networks, it get more complex; because we will use this to extract features from an image.
- The first thing we do usually in convolution neural networks is the convolution operation, so the first layer that takes the images will be the convolution layer.
- We need to resize the shape of our image, as it is simpler to be processed [Note that we can detect the dog from the image either the image is big or small].

So, we will at first load and resize our image:

```
5   img = cv2.imread("Car.png", cv2.IMREAD_GRAYSCALE)
6   img = cv2.resize(img, (224, 224))  # Common size for classification neural network
7   height, width = img.shape
8   cv2.imshow("image", img)
9   cv2.waitKey(0)
```

To create the convolution layer:

```
11  model = keras.Sequential()
12  # Parameters essential for creating the Conv2D layer are:
13  # input shape, filters, kernel_size
14  model.add(layers.Conv2D(input_shape=(height, width)), filters=64, kernel_size=(3, 3))
```

In order to extract the information from this layer, we need to access the wights that are created each time we add a layer. If we check the summary of the model, we get params for each layer. These are values that we can access.

```
14    model.summary()
```

The params are the filters that is generated as a result from features extraction, each one of them represents an feature.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 64)      640

=================================================================
Total params: 640
Trainable params: 640
Non-trainable params: 0
_________________________________________________________________
```

To access the first filter:

```
16  # Access filters/parameters/weights/features
17  filters, _ = model.layers[0].get_weights() # The index is 0 for the 1st layer, 1 for the 2nd layer, and so on
18  print("Filters shape = ", filters.shape)  # (3, 3, 1, 64) => 64 images of size 3x3
```

The output will be:

```
Filters shape =  (3, 3, 1, 64)
```

So, the kernel_size parameter is the size of the images that will be used as features and the filiters argument is the number of filters/features generated.

- *Another way to get the 3rf filter(feature image):*

```
19  f = filters[:, :, :, 2]
20  cv2.imshow("img", f)
21  cv2.waitKey(0)
```

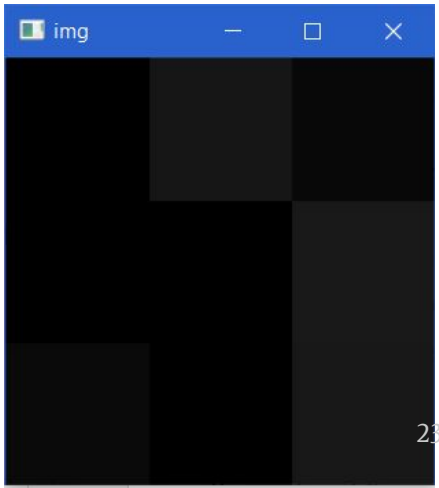
The output be like: 

The kernal_size which is the size of each filter (feature) is 3x3 which is very small.
So, we can resize it to be more clear to see:

```
16  # Access filters/parameters/weights/features
17  filters, _ = model.layers[0].get_weights() # The index is 0 for the 1st layer, 1 for the 2nd layer, and so on
18  print("Filters shape = ", filters.shape)  # (3, 3, 1, 64) => 64 images of size 3x3
19  f = filters[:, :, :, 2]
20  f = cv2.resize(f, (250, 250), interpolation=cv2.INTER_NEAREST)
21  cv2.imshow("img", f)
22  cv2.waitKey(0)
```

To increase the pixels size
without blurring occurrence.

Output

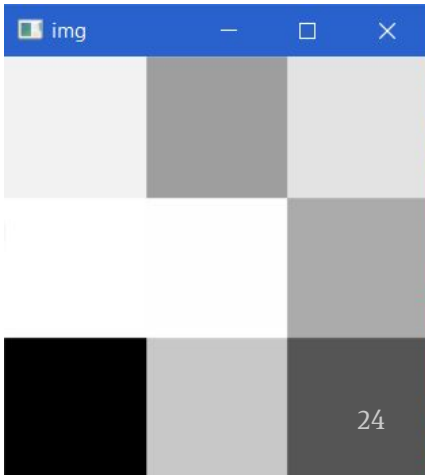


The divergence in pixels colors is not clear, so to solve this, we need to perform normalization which means limiting the pixels brightness intensity range from between 0 and 1, where 1 is the maximum and 0 is the minimum, this is done by:

```
24    print("F maximum and F minimum before normalization: ", f.min(), ", ", f.max())
25    # Normalize each filter (feature extraction image)
26    f_min, f_max = f.min(), f.max()
27    f = (f - f_min) / (f_max - f_min)
28    print("After normalization: ", f.min(), ", ", f.max())
```

The output:

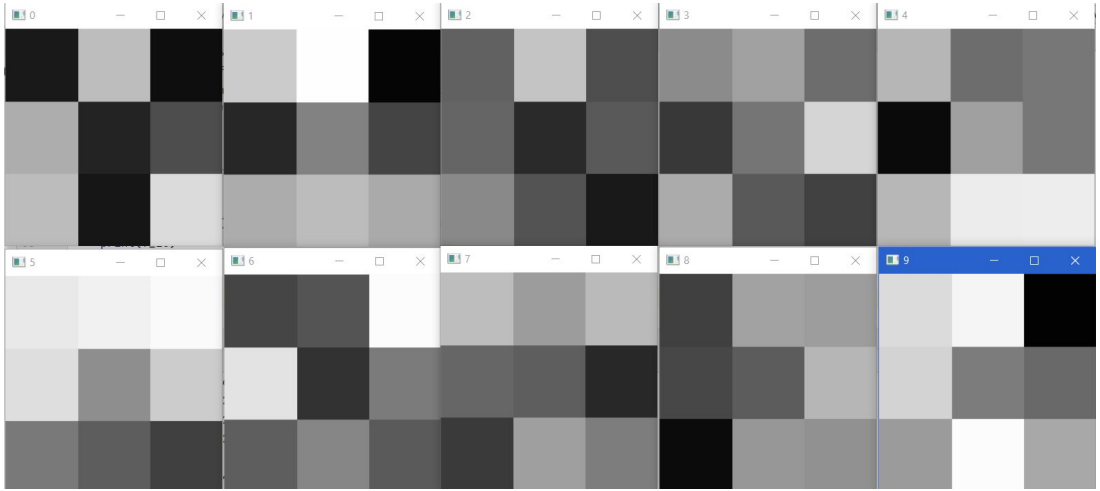
```
F maximum and F minimum before normalization:  -0.08586654 ,  0.08595742
After normalization:  0.0 ,  1.0
```



To get the first 10 filters/ params/ feature images:

```
32  # Get the first 10 filters/ feature images
33  for i in range(10):
34      f_10 = filters[:, :, :, i]
35      f_10 = cv2.resize(f_10, (250, 250), interpolation=cv2.INTER_NEAREST)
36      print(f_10)
37      cv2.imshow(str(i), f_10)
38
39  cv2.waitKey(0)
```

Output:



What does this output represent?

Features extraction from an image to create a feature map. In other words:

- The kernel size represents the complexity of the feature, so the bigger kernel_size, the greater feature complexity.
- The number of filters represents the number of features that we can get.

Note that:

- The kernel_size parameter is usually a square, ((3, 3)/ (5, 5)/ (7, 7)/ etc...).
- The filters parameter depends on the type of neural network, (detection/ classification/ etc...).

Code

Content

- ~~Libraries installation~~
- ~~Neural network intro.~~
- ~~Image~~
- ~~Neural network structure~~
- ~~Sequential model~~
- ~~Conv2D layer~~
- Feature map
- Max Pooling layer
- Flatten and Dense layers
- Image classification model using VGG16

Feature map

The feature map is the result of the features that we extract from the convolution layer.

Usually in computer vision, we shrink all images to a standard square size. Each model has its own size.

If the size is a rectangle not a square, we've to resize it as it is the standard in CV. Surely, we will lose some proportion as if the image is wide and we shrink it, some proportion will be lost, but it is how it works.

- The first step: Load the image and resize it

```
7   # Load image
8   img = cv2.imread("Car.png", cv2.IMREAD_GRAYSCALE)
9   img = cv2.resize(img, (224, 224))
10  height, width = img.shape
11  cv2.imshow("img", img)
12  cv2.waitKey(0)
```

The model takes the input image/layer and gives the output of the last layer.

- The second step is building the model

```
14  # Model
15  model = keras.Sequential()
16  model.add(layers.Conv2D(input_shape=(height, width, 1), filters=64, kernel_size=(3, 3)))
17
18  model.summary()
```

3rd step: we will pass the input image into the model, to start features extraction after convolution layer

```
20  # pass the image into the model
21  feature_map = model.predict(np.array([img]))
22  print(feature_map.shape)  # 64 feature images of size 222x222 for one input image
23  print(feature_map)  # The output is the feature map
```

`np.array()` method takes the input image in a list, as the neural network is designed to get multiple images as input layer not only one image.

4th step: Extract the features and visualize them

```
25  # Visualize the feature map (64 images)
26  #  Extract only onr
27  feature_img = feature_map[0, :, :, 0]
28  plt.imshow(feature_img, cmap="gray")  # To display the image in grayscale
29  plt.show()
```

We can also get all the features extracted whose number is defined in the code in `filters=64` parameter.

```
# Extract all images
for i in range(64):
    feature_img = feature_map[0, :, :, i]
    ax = plt.subplot(8, 8, i+1)  # Put number of image, foe example: for cell 0, column 0 => image 1
    # For `subplot` function, we need to pass 3 parameter:
    # no.of vertical images, no.of horizontal images, index of each single image
    # Remove the ticks (0 -> 222) that are around each image
    ax.set_xticks([])
    ax.set_yticks([])
    plt.imshow(feature_img, cmap="gray")
plt.show()
```

Note that:

It is better to visualize the features images in Matplotlib then OpenCV.

We can pass the input image in 3-channel colors or in grayscale, but the no.of params/weights will increase in case of 3-channel colors. Choosing between them depends on the model.

```python
# Load image
img = cv2.imread("Car.png", cv2.IMREAD_GRAYSCALE)
img = cv2.resize(img, (224, 224))
height, width = img.shape
cv2.imshow("img", img)
cv2.waitKey(0)

# Model
model = keras.Sequential()
model.add(layers.Conv2D(input_shape=(height, width, 1), filters=64, kernel_size=(3, 3)))

model.summary()
```

Params for grayscale →

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 64)      640

=================================================================
```

```python
# Load image
img = cv2.imread("Car.png")
img = cv2.resize(img, (224, 224))
height, width, channels = img.shape
cv2.imshow("img", img)
cv2.waitKey(0)

# Model
model = keras.Sequential()
model.add(layers.Conv2D(input_shape=(height, width, channels), filters=64, kernel_size=(3, 3)))

model.summary()
```

Params for 3-channel colors →

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 64)      1792

=================================================================
```

Feature map:

What is the purpose of feature map?

The purpose of the feature map is to extract features; as the further we go inside the model close to the output level, the less information we have.

- The concept is for example, for face recognition, the face will be a series of multiple features, for example will be the eyelashes, the mouth, the nose, and so on...
- So, ideally, each image from the feature map will be something different from the face and when we have a lot of features that make a face, then it says this is a face, otherwise, it is not a face.
- Each image in feature map is going to extract something different from the other image.

Code

Content

- ~~Libraries installation~~
- ~~Neural network intro.~~
- ~~Image~~
- ~~Neural network structure~~
- ~~Sequential model~~
- ~~Conv2D layer~~
- ~~Feature map~~
- Max Pooling layer
- Flatten and Dense layers
- Image classification model using VGG16

Max Pooling layer

Simply, it is most about taking a square of pixels and then get the maximum value into a new pixel.

It is done by using `MaxPooling2D()` function, by passing two arguments into it which are:

- pool_size
- strides

Creation:

```
19      model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(1, 1)))
```

For better understanding, let us explain the code steps and get the convolution layer output image list and the Max Pooling layer output image list, then compare them together:

For this, we will create two files one for convolution layer only and the second is for the convolution + Pooling layers to see the difference clearly...

For the convolution layer file:

1. Firstly, we will load our image and perform the preprocessing step right now:

```
12  # Load image
13  img = cv2.imread("Car.png")
14  img = cv2.resize(img, (7, 7))
```

We set the image size to (7, 7) only to understand the difference between the two layers output.

2. Secondly, we will build the CNN model including the convolution layer:

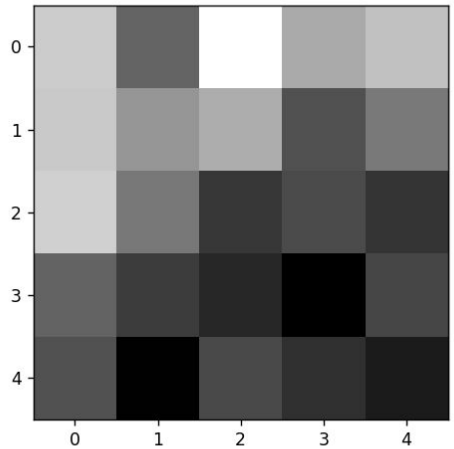
```
16  # CNN Model
17  model = keras.Sequential()
18  model.add(layers.Conv2D(input_shape=(7, 7, 3), filters=64, kernel_size=(3, 3)))
19  feature_map = model.predict(np.array([img]))
20
21  feature_img = feature_map[0, :, :, 0]
22  print("Image pixels after convolution\n", feature_img)
23  plt.figure("Image after convolution")
24  plt.imshow(feature_img, cmap="gray")
25  plt.show()
```

We have to keep the note that Keras generates its weights randomly, so each time we run it, we will get different values for the image list, in this way we can't compare between the two layers' output. Thus, we need to fix the weights generated by it and to do this, we set the tensorflow funcion `random.set_seeds()` into 0.

```
8   # Set random seed
9   tf.keras.utils.set_random_seed(0)
```

The output of the convolution layer file will be:

```
1/1 [==============================] - 0s 358ms/step
Image pixels after convolution
 [[ 81.65944    35.159847  104.92497    66.53307    76.99078  ]
 [ 80.51706    57.484264   67.63798    26.399046   44.563976 ]
 [ 83.406845   43.980988   14.842911   23.666834   13.444102 ]
 [ 34.611973   17.3805      7.6288323 -10.278886   21.565739 ]
 [ 26.769157   -9.508825   23.399143   11.605007    2.0221872]]
```



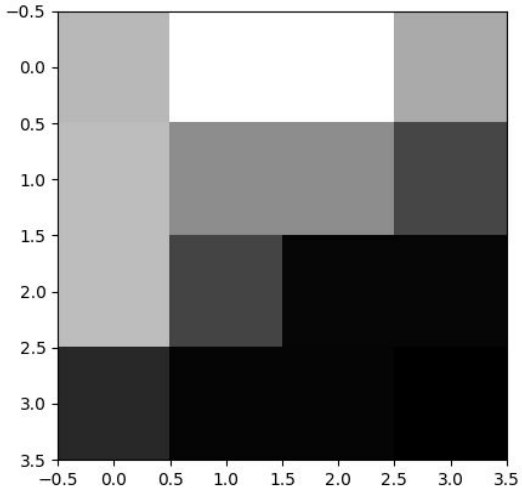
For the convolution + pooling layer file:

After loading the image and setting fixed seeds, we will build the model:

```
16    # CNN Model
17    model = keras.Sequential()
18    model.add(layers.Conv2D(input_shape=(7, 7, 3), filters=64, kernel_size=(3, 3)))
19    model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(1, 1)))
20    feature_map = model.predict(np.array([img]))
21
22    feature_img = feature_map[0, :, :, 0]
23    print("Image pixels after convolution and pooling\n", feature_img)
24    plt.figure("Image after convolution and pooling")
25    plt.imshow(feature_img, cmap="gray")
26    plt.show()
```

The output of this file will be:

```
1/1 [==============================] - 0s 85ms/step
Image pixels after convolution and pooling
 [[ 81.65944  104.92497  104.92497   76.99078 ]
 [ 83.406845  67.63798   67.63798   44.563976]
 [ 83.406845  43.980988  23.666834  23.666834]
 [ 34.611973  23.399143  23.399143  21.565739]]
```



To understand what had happened, we have to understand the parameters of the 'MaxPooling2D' function:

1. *pool_size* ⇒ It is the subset matrix from the convolution matrix we will apply the max pooling on.
2. *strides* ⇒ It is the step that each max pooling will move to the next one.

Max pooling means to find the max pixel value from the subset matrix can create a new pixel with this max value.

'pool_size' parameter further explanation:

If it is set to (2, 2) ⇒ The subset matrix size if (2, 2)

```
1/1 [==============================] - 0s 358ms/step
Image pixels after convolution
 [[ 81.65944    35.159847  104.92497    66.53307    76.99078  ]
 [ 80.51706    57.484264  67.63798    26.399046   44.563976 ]
 [ 83.406845   43.980988  14.842911   23.666834   13.444102 ]
 [ 34.611973   17.3805     7.6288323 -10.278886   21.565739 ]
 [ 26.769157   -9.508825   23.399143   11.605007    2.0221872]]
```

'strides' parameter further explanation:

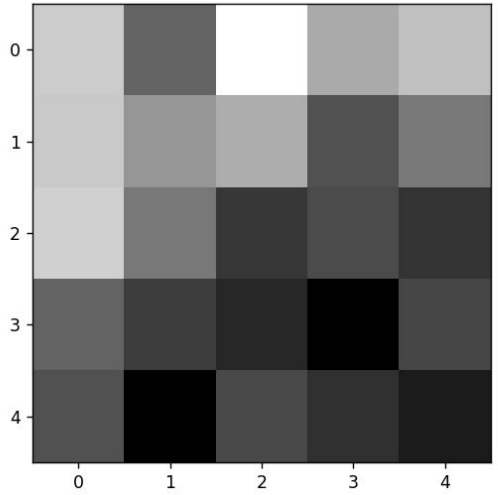
If it is set to (1, 1) ⇒ Each subset will move by 1 in height and 1 in width to the next subset:

```
1/1 [==============================] - 0s 358ms/step
Image pixels after convolution
 [[ 81.65944    35.159847  104.92497    66.53307    76.99078  ]
 [ 80.51706    57.484264  67.63798    26.399046   44.563976 ]
 [ 83.406845   43.980988  14.842911   23.666834   13.444102 ]
 [ 34.611973   17.3805     7.6288323 -10.278886   21.565739 ]
 [ 26.769157   -9.508825   23.399143   11.605007    2.0221872]]
```

Colored rectangles represents each movement of the pooling.

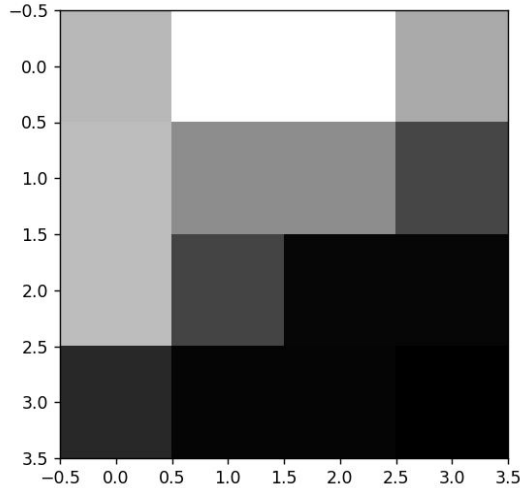
Comparing between the two files' outputs:

The convolution layer file:




```
1/1 [==============================] - 0s 358ms/step
Image pixels after convolution
 [[ 81.65944    35.159847  104.92497    66.53307    76.99078  ]
 [ 80.51706    57.484264   67.63798    26.399046   44.563976 ]
 [ 83.406845   43.980988   14.842911   23.666834   13.444102 ]
 [ 34.611973   17.3805      7.6288323 -10.278886   21.565739 ]
 [ 26.769157   -9.508825   23.399143   11.605007    2.0221872]]
```

The convolution+pooling layer file:




```
1/1 [==============================] - 0s 85ms/step
Image pixels after convolution and pooling
 [[ 81.65944  104.92497  104.92497   76.99078 ]
 [ 83.406845  67.63798   67.63798   44.563976]
 [ 83.406845  43.980988  23.666834  23.666834]
 [ 34.611973  23.399143  23.399143  21.565739]]
```

```
1/1 [==============================] - 0s 358ms/step
Image pixels after convolution
 [[ 81.65944    35.159847  104.92497    66.53307    76.99078  ]
 [ 80.51706    57.484264  67.63798    26.399046   44.563976 ]
 [ 83.406845   43.980988  14.842911   23.666834   13.444102 ]
 [ 34.611973   17.3805     7.6288323 -10.278886   21.565739 ]
 [ 26.769157   -9.508825  23.399143   11.605007    2.0221872]]
```

Max. of the 1st subset matrix = 81.65944

Max. of the 2nd subset matrix = 104,92497

And so on.....

```
1/1 [==============================] - 0s 85ms/step
Image pixels after convolution and pooling
 [[ 81.65944  104.92497  104.92497   76.99078 ]
 [ 83.406845  67.63798   67.63798   44.563976]
 [ 83.406845  43.980988  23.666834  23.666834]
 [ 34.611973  23.399143  23.399143  21.565739]]
```

code:

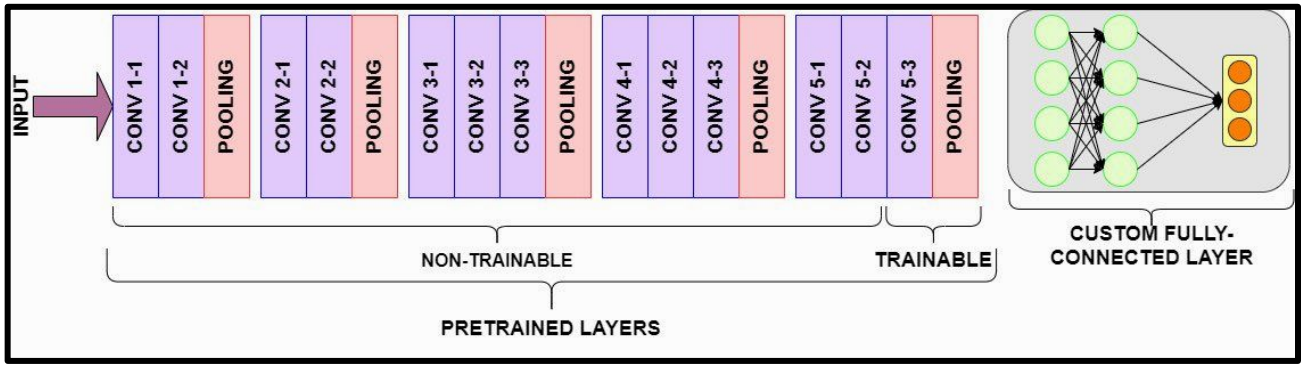
- Convolution layer matrix
- Convolution + pooling layer matrix code

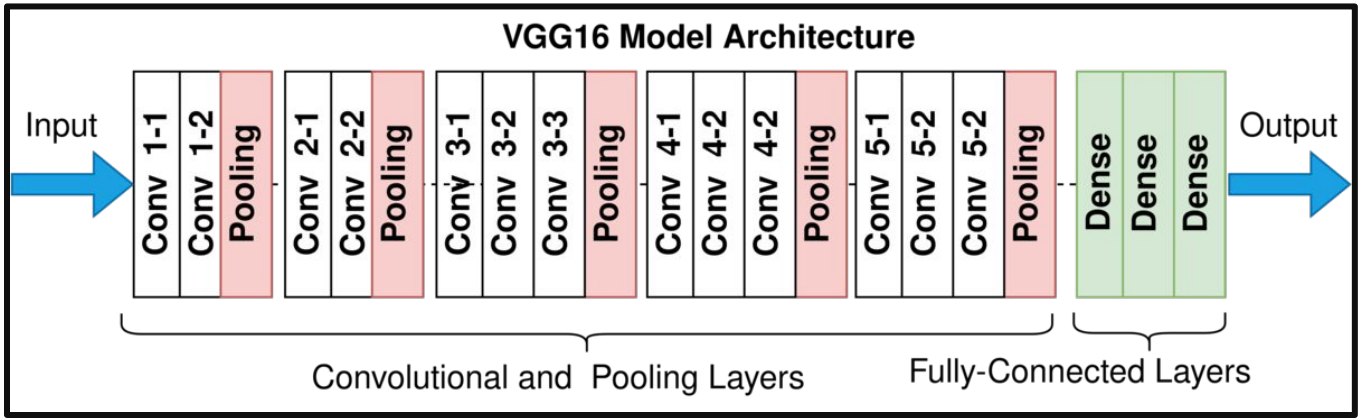
Content

- ~~Libraries installation~~
- ~~Neural network intro.~~
- ~~Image~~
- ~~Neural network structure~~
- ~~Sequential model~~
- ~~Conv2D layer~~
- ~~Feature map~~
- ~~Max Pooling layer~~
- Flatten and Dense layers
- Image classification model using VGG16

Flatten and Dense layers

Example of a classification neural network architecture (VGG 16 Model):





The Dense layer is a fully connected layer where the final goal is to bring an image of large and multiple values into a small values.

For example, the cat-dog classification problem, we say it is a cat if the output of all the layers is close to 0, and dog if the output of all the layers of neural network is close to 1.

After running our model on our input image, we get a multi-dimensional matrix includes features of the image after processing at layers of neural network, with size of (1, 29, 29, 64)

```
Extracted features after convolution and pooling layers:
 [[[[-2.1591686e+01 -3.7260395e+01 -1.0190188e+01 ...  7.0343231e+01
     1.2846365e+02  5.9149475e+01]
    [-2.1591686e+01 -3.5279182e+01 -1.1650402e+01 ...  7.1195938e+01
     1.3102802e+02  5.9608414e+01]
    [-2.3199327e+01 -3.1714319e+01 -1.0789754e+01 ...  7.1427460e+01
     1.3283560e+02  6.0225540e+01]
    ...
```

Now, how can we transform these features/images into just numbers?

This is done through two operations in sequence:

1. Flatten
2. Dense

1. Flatten layer:

- *Creation:*

```
13    # CNN Model
14    model = keras.Sequential()
15    model.add(layers.Conv2D(input_shape=(img_size, img_size, 3), filters=64, kernel_size=(3, 3)))
16    model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(1, 1)))
17    model.add(layers.Flatten())
```

This layer puts a lot of images together (flattening in English).

- *The output:*

```
Extracted features after convolution and pooling layers:
 [[ 54.734535   -0.8900752 -59.906216  ...  -5.4965096   7.329387
   15.682692 ]]
The features shape: (1, 53824)
```

→ (1, 29*29*64)

Random example for Flatten layer:
If we have such an image of a white vertical line, the img data will be:

2. Dense layer:

- *Creation:*

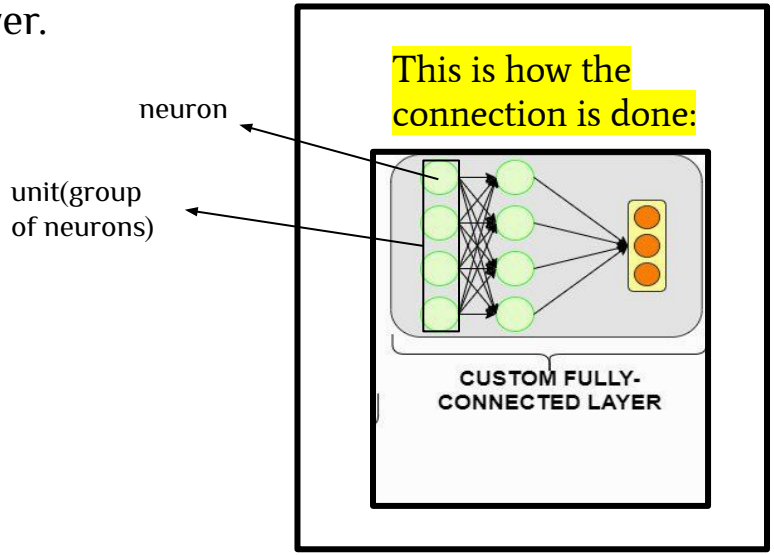
```
18    model.add(layers.Dense(units=10))
```

This step is passing the `units` parameter equals to 'number of neurons' units, the purpose is to simplifying the information to come with a small numbers.
The *shape* will be: (1, units number)

The output will be:

```
Extracted features after convolution and pooling layers:
 [[ -64.54524    30.235044  -10.594951   33.504795   54.81801    64.98638
    -3.649742   50.02368    72.70337  -159.30592 ]]
The features shape:  (1, 10)
```

Here all +53000 features resulted from the convolution and pooling layers will be connected somehow to the units of the dense layer.



Code

Content

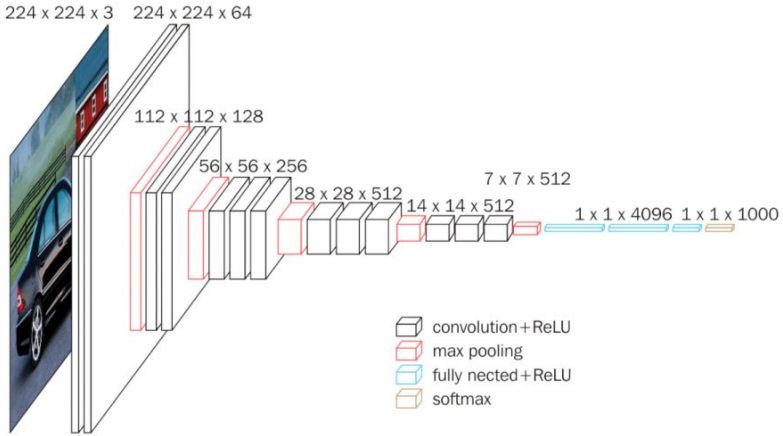
- ~~Libraries installation~~
- ~~Neural network intro.~~
- ~~Image~~
- ~~Neural network structure~~
- ~~Sequential model~~
- ~~Conv2D layer~~
- ~~Feature map~~
- ~~Max Pooling layer~~
- ~~Flatten and Dense layers~~
- Image classification model using VGG16

Image classification model using VGG 16

1. *VGG 16 model:*

A classification model for large-scale image recognition. It achieved 92.7 % accuracy on a dataset with 1000 classes, which means that it could detect 1000 different types of objects on a huge image dataset with +14 million images.

2. *VGG 16 layers:*



Model architecture:

Input

images of size 224*224*3

Block #1

Two convolution layers:
- Images of size 224*224
- 64 filters
- Activation: relu

One pooling layer:
- Images of 112*112

Block #2

Two convolution layers:
- Images of size 112*112
- 128 filters
- Activation: relu

One pooling layer:
- Images of size 56*56

Block #3

Three convolution layers:
- Images of size 56*56
- 256 filters
- Activation: relu

One pooling layer:
- Images of size 28*28

Block #4

Three convolution layers:
- Images of size 28*28
- 512 filters
- Activation: relu

One pooling layer:
- Images of size 14*14

Model architecture:

Block #5

Three convolution layers:
- Images of size 14*14
- 512 filters
- Activation: relu

One pooling layer:
- Images of size 7*7

Block #6

One flatten layer

Three dense layers:
- 2 layers of 4096 units and relu activation.

1 layer of 1000 units; for custom models: it is the number of our categories to be classified. VGG16 has been trained on 1000 classes, so the units in its architecture is 1000.

The model takes 224x224x3 images, that how it was developed, so we have to shrink the image to the size suited for the model.

```
# Set seed
tf.keras.utils.set_random_seed(0)

# Load image
img = cv2.imread("Car.png")
img = cv2.resize(img, (224, 224))
```

- Building the first block:

```
# VGG 16 Model
model = keras.Sequential()
# Block 1
# 1. convolution layer with 64 filters
model.add(layers.Conv2D(input_shape=(224, 224, 3),                    filters=64, kernel_size=(3, 3)))
# 2. Convolution layer similar to the first one
model.add(layers.Conv2D(filters=64,                    kernel_size=(3, 3)))
# 3. MaxPooling layer
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(1, 1)))
```

After running the first two layers, we have to follow the output shape of the features extracted from the convolution layers and make sure they are as the model is setting:

Our output summary:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 64)      1792

 conv2d_1 (Conv2D)           (None, 220, 220, 64)      36928

=================================================================
```

The model sets the shape must be (224, 224), so *why we lose 2 pixels for every convolution layer?* The answer is by knowing the convolution layer. The solution is to apply the padding.

```
13  # 1. convolution layer with 64 filters
14  model.add(layers.Conv2D(input_shape=(224, 224, 3), padding="same", filters=64, kernel_size=(3, 3)))
15  # 2. Convolution layer similar to the first one
16  model.add(layers.Conv2D(64, padding="same", kernel_size=(3, 3)))
```

The output now will be:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 224, 224, 64)      1792

 conv2d_1 (Conv2D)           (None, 224, 224, 64)      36928

=================================================================
```

Then, we will display the feature map after the first block in this way:

```
# result
# Predict the result of the image
feature_map = model.predict(np.array([img]))
for i in range(64):
    feature_img = feature_map[0, :, :, i]
    ax = plt.subplot(8, 8, i+1)
    ax.set_xticks([])
    ax.set_yticks([])
    plt.imshow(feature_img, cmap="gray")
plt.show()
```

The feature map will be like:

The activation function is by default `activation="vgg16"`, but the model uses “ReLU”:

- For example, we can set it as “ReLU” which is “Rectified Linear Unit activation”.
- Some other activation functions give us some result whether it is positive, how positive the number is, and of the result is negative, then how negative the number is.
- “relu” ⇒ if that result is positive, it says how much is positive. If it is negative, it just says 0, so it just cut-off the connection with the next neuron.
- To set the relu activation:

```
14  # 1. convolution layer with 64 filters
15  model.add(layers.Conv2D(input_shape=(224, 224, 3), activation="relu", padding="same", filters=64, kernel_size=(3, 3)))
16  # 2. Convolution layer similar to the first one
17  model.add(layers.Conv2D(filters=64, padding="same", activation="relu", kernel_size=(3, 3)))
```

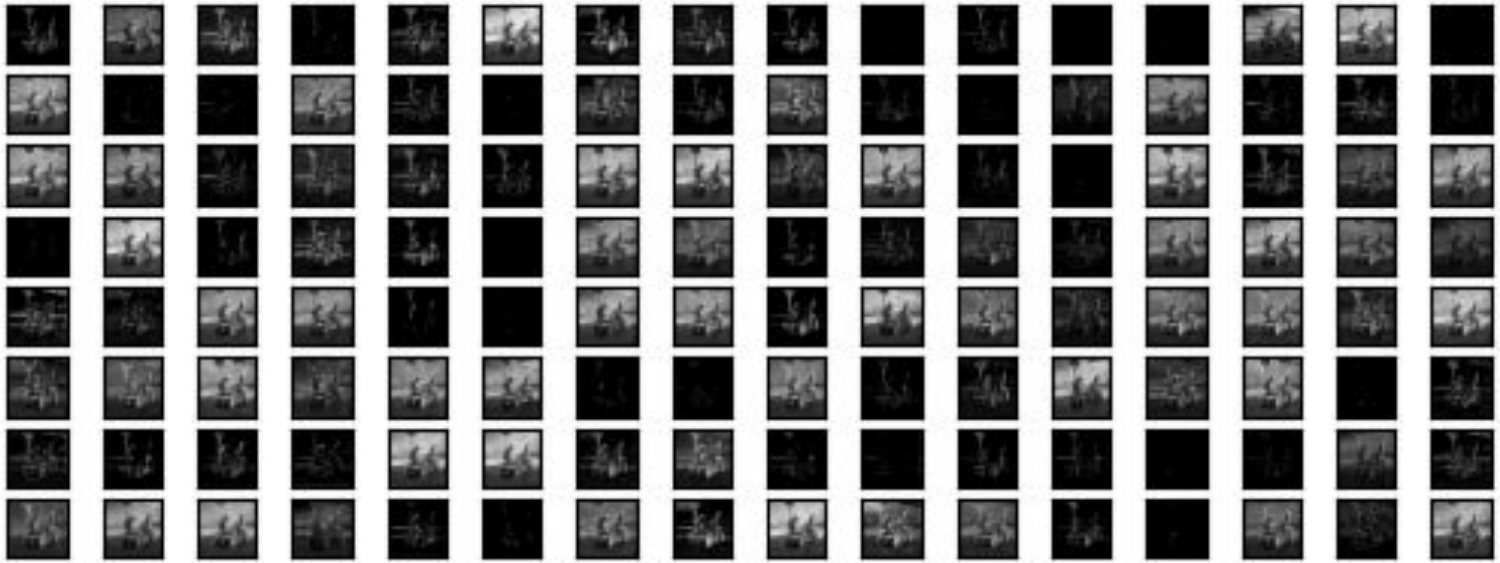
Then the feature map will be:

The feature map will have many black images, This also takes the concept of cutting off the neuron if the result number is negative.

Building the second block:

```
25    # Block 2
26    model.add(layers.Conv2D(filters=128, kernel_size=(3, 3), padding="same", activation="relu"))
27    model.add(layers.Conv2D(filters=128, kernel_size=(3, 3), padding="same", activation="relu"))
28    model.add(layers.MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))
```

The feature map will be:



The model summary:

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d (Conv2D)              (None, 224, 224, 64)      1792

 conv2d_1 (Conv2D)            (None, 224, 224, 64)      36928

 max_pooling2d (MaxPooling2D  (None, 223, 223, 64)      0
 )

 conv2d_2 (Conv2D)            (None, 223, 223, 128)     73856

 conv2d_3 (Conv2D)            (None, 223, 223, 128)     147584

 max_pooling2d_1 (MaxPooling  (None, 111, 111, 128)     0
 2D)
```

Building the third block:

```
30    # Block 3
31    model.add(layers.Conv2D(filters=256, kernel_size=(3, 3), padding="same", activation="relu"))
32    model.add(layers.Conv2D(filters=256, kernel_size=(3, 3), padding="same", activation="relu"))
33    model.add(layers.Conv2D(filters=256, kernel_size=(3, 3), padding="same", activation="relu"))
34    model.add(layers.MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))
```

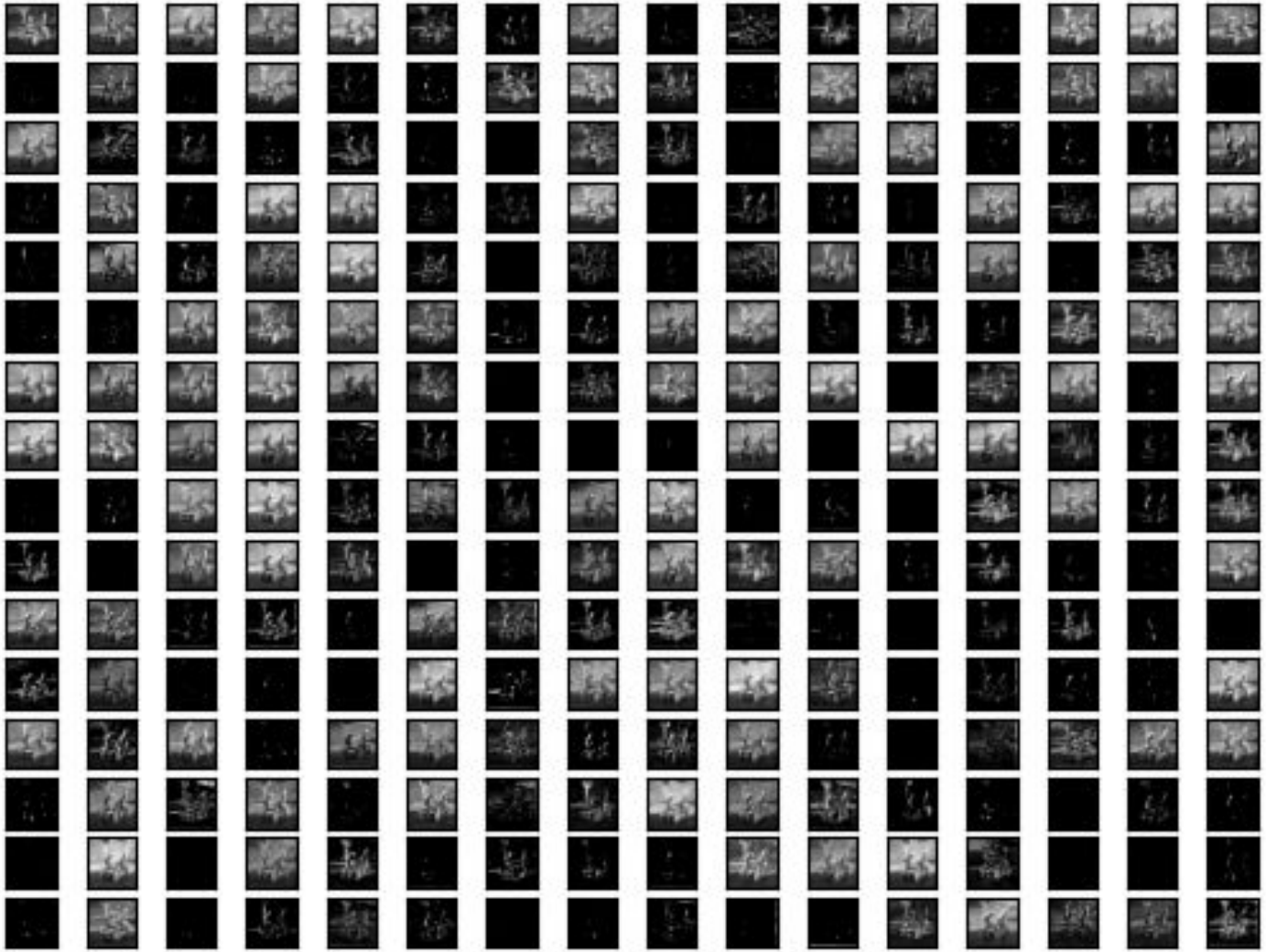
The model summary:

```
conv2d_4 (Conv2D)           (None, 111, 111, 256)     295168

conv2d_5 (Conv2D)           (None, 111, 111, 256)     590080

conv2d_6 (Conv2D)           (None, 111, 111, 256)     590080

max_pooling2d_2 (MaxPooling  (None, 55, 55, 256)      0
2D)
```

The feature map:

So clear that:
The further we go into the network, the features are extracted more in detail.

&& The more we are going depth, the smaller the features are getting the more simpler, but the number of features is increasing



Building 4th, 5th, and 6th blocks:

```
38  # Block 4
39  model.add(layers.Conv2D(filters=512, kernel_size=(3, 3), padding="same", activation="relu"))
40  model.add(layers.Conv2D(filters=512, kernel_size=(3, 3), padding="same", activation="relu"))
41  model.add(layers.Conv2D(filters=512, kernel_size=(3, 3), padding="same", activation="relu"))
42  model.add(layers.MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))
43  # The more we are going depth, the smaller the features are getting the more simpler, but the number of features is increasing
44
45  # Block 5
46  model.add(layers.Conv2D(filters=512, kernel_size=(3, 3), padding="same", activation="relu"))
47  model.add(layers.Conv2D(filters=512, kernel_size=(3, 3), padding="same", activation="relu"))
48  model.add(layers.Conv2D(filters=512, kernel_size=(3, 3), padding="same", activation="relu"))
49  model.add(layers.MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))
50
51  # Block 6
52  model.add(layers.Flatten())
53  model.add(layers.Dense(units=4096, activation="relu"))
54  model.add(layers.Dense(units=4096, activation="relu"))
55  model.add(layers.Dense(units=3, activation="softmax"))
```

The model summary:

```
conv2d_7 (Conv2D)           (None, 55, 55, 512)       1180160

conv2d_8 (Conv2D)           (None, 55, 55, 512)       2359808

conv2d_9 (Conv2D)           (None, 55, 55, 512)       2359808

max_pooling2d_3 (MaxPooling  (None, 27, 27, 512)      0
2D)

conv2d_10 (Conv2D)          (None, 27, 27, 512)       2359808

conv2d_11 (Conv2D)          (None, 27, 27, 512)       2359808

conv2d_12 (Conv2D)          (None, 27, 27, 512)       2359808

max_pooling2d_4 (MaxPooling  (None, 13, 13, 512)      0
2D)

flatten (Flatten)           (None, 86528)             0

dense (Dense)               (None, 4096)              354422784

dense_1 (Dense)             (None, 4096)              16781312

dense_2 (Dense)             (None, 3)                 12291
```

How can we understand the output matrix?

```
1/1 [==============================] - 7s 7s/step
[[0.305629   0.29119745 0.40317357]]
```

We passed 3 units in the last dense layer, so this model will classify between 3 categories, if we passed for example, cat, dog, horse categories in this order, so the order of elements is [cat=0.305629 dog=0.29119745 horse=0.40317357]. The detected animal will have the maximum value. So, for this example, the model classified this animal as horse.

Code