Faculty of Engineering
Cairo University

HEMN451 – MEDICAL PATTERN RECOGNITION

Term Project

CNN

Submitted by:

- Ahmed Helmy Ahmed 1170539
- Ala'a Osama Ahmed 1170185
- Amira Mahmoud Farid 1170498
- Salma Atia Ahmed 1170211

Table of Contents

Problem Definition and Importance 4
What are Neural Networks? 4
What is Convolutional Neural Network? 4
Why should you use a CNN? 4
Methods and Algorithms 5
How does the entire network look like? 5
Augmentation 6
Experimental Results and discussions 8
Structure Tests 9
Try 1: Change img SIZE= 200*200*3 9
Try 2: Normalize images to from 0 to 1 9
Try 3: increase filter size from (3, 3) to (5, 5) 9
Try 4: increase filter size from (5, 5) to (10, 10) 9
Try 5: decrease number of filters from 32 to 16 9
Try 6: increase strides from (1, 1) to (3, 3) 9
Try 7: increase strides from (1, 1) to (2, 2) 9
Try 8: change stride to (2, 1) 9
Try 9: change stride to (1, 2) 10
Try 10: change padding from "valid" to "same" 10
Try 11: change activation function from relu to tanh 10
Try 12: change activation function from tanh to sigmoid 10
Try 13: change pool_size from (2, 2) to (1, 1) (effect of removing pooling layer) 10
Try 14: change pool_size from (2, 2) to (5, 5) 10
Try 15: change output activation function from softmax to softplus 10
Try 16: change output activation function to softsign 11
Try 17: duplicate the layers before and after flatten and change pool_size to (2, 2) 11
Apply data augmentation script 11
Try 1: apply data augmentation script for our 10 objects' folders 11
Try 2: Trained the model on the new data and save the results 11
Final State 11
Results 13

Try to predict random image from the internet 14
Code 15
To start training the model: 15
To start using our model: 15
References 15

Problem Definition and Importance

What are Neural Networks?

Neural networks reflect the behavior of the human brain, allowing computer programs to recognize patterns and solve common problems in the fields of AI, machine learning, and deep learning.

What is Convolutional Neural Network?

The human brain is a very powerful machine. We see (capture) multiple images every second and process them without realizing how the processing is done. But, that is not the case with machines. The first step in image processing is to understand, how to represent an image so that the machine can read it?

In simple terms, every image is an arrangement of dots (a pixel) arranged in a special order. If you change the order or color of a pixel, the image would change as well.

The machine will basically break this image into a matrix of pixels and store the color code for each pixel at the representative location.

Here, we need to preserve the spatial arrangement in both horizontal and vertical direction. We can take the weight as a 2D matrix which takes pixels together in both horizontal and vertical direction. Also, keep in mind that since we have taken both horizontal and vertical movement of weights, the output is one pixel lower in both horizontal and vertical direction.

This is exactly what a Convolutional neural network does. We can take the input image, define a weight matrix and the input is convolved to extract specific features from the image without losing the information about its spatial arrangement.

Why should you use a CNN?

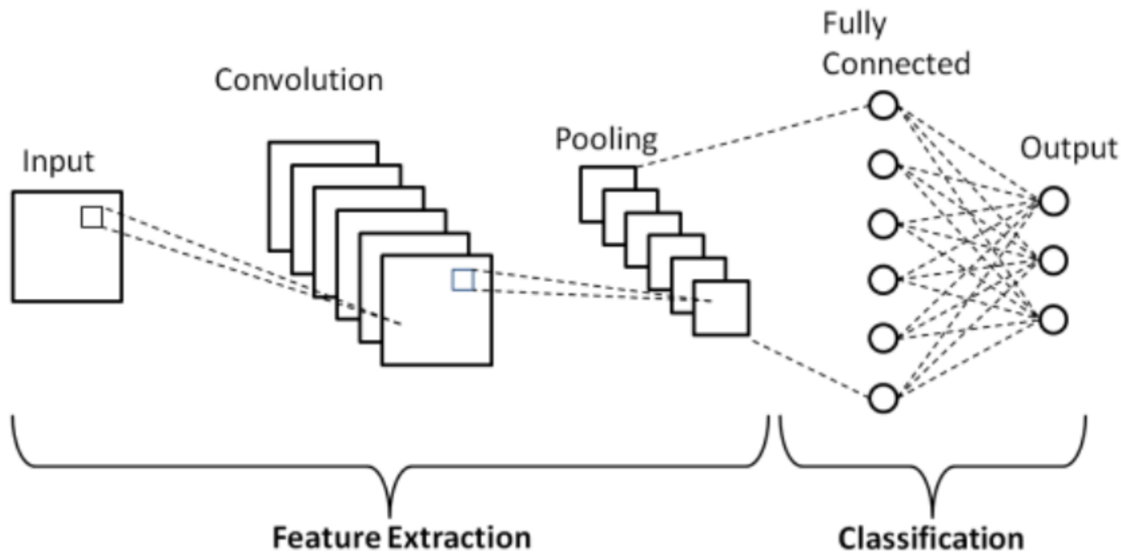
- ❖ CNN learns the filters automatically without mentioning it explicitly. These filters help in extracting the right and relevant features from the input data.

- ❖ CNN captures the spatial features from an image. Spatial features refer to the arrangement of pixels and the relationship between them in an image. They help us in identifying the object accurately, the location of an object, as well as its relation with other objects in an image.

Methods and Algorithms

How does the entire network look like?

- CNN is composed of various convolutional and pooling layers. Let's see how the network looks like.



- We pass an input image to the first convolutional layer. The convoluted output is obtained as an activation map. The filters applied in the convolution layer extract relevant features from the input image to pass further.
- Each filter shall give a different feature to aid the correct class prediction. In case we need to retain the size of the image, we use the same padding (zero padding), otherwise valid padding is used since it helps to reduce the number of features.
- Pooling layers are then added to further reduce the number of parameters.
- Several convolution and pooling layers are added before the prediction is made. Convolutional layer help in extracting features. As we go deeper in the network more specific features are extracted as compared to a shallow network where the features extracted are more generic.
- The output layer in a CNN as mentioned previously is a fully connected layer, where the input from the other layers is flattened and sent so as the transform the output into the number of classes as desired by the network.
- The output is then generated through the output layer and is compared to the output layer for error generation. A loss function is defined in the fully connected output layer to compute the mean square loss. The gradient of error is then calculated.
- The error is then backpropagated to update the filter (weights) and bias values.

- One training cycle is completed in a single forward and backward pass.

Augmentation

The performance of deep learning neural networks often improves with the amount of data available.

Data augmentation is a technique to artificially create new training data from existing training data. This is done by applying domain-specific techniques to examples from the training data that create new and different training examples.

Image data augmentation is perhaps the most well-known type of data augmentation and involves creating transformed versions of images in the training dataset that belong to the same class as the original image.

Transforms include a range of operations from the field of image manipulation, such as shifts, flips, zooms, and much more. The intent is to expand the training dataset with new, plausible examples.

Modern deep learning algorithms, such as the convolutional neural network, or CNN, can learn features that are invariant to their location in the image. Nevertheless, augmentation can further aid in this transform invariant approach to learning and can aid the model in learning features that are also invariant to transforms.

A convolutional neural network that can robustly classify objects even if it is placed in different orientations is said to have the property called invariance. More specifically, a CNN can be invariant to translation, viewpoint, size or illumination.

This essentially is the premise of data augmentation. In the real-world scenario, we may have a dataset of images taken in a limited set of conditions. But our target application may exist in a variety of conditions, such as different orientation, location, scale, brightness etc. We account for these situations by training our neural network with additional synthetically modified data.

Image data augmentation is typically only applied to the training dataset, and not to the validation or test dataset. This is different from data preparation such as image resizing and pixel scaling; they must be performed consistently across all datasets that interact with the model.

In the project, we tested the accuracy for the model before data augmentation which resulted in a low accuracy with a maximum 30%-33%. After applying data augmentation, the resulted accuracy greatly increased to reach 95%. We applied brightness, zoom, and rotate techniques. For each of these techniques, we also specify the factor by which the size of dataset would get increased. For the brightness and zoom techniques the augmentation factor = 9x. And as for rotate technique the augmentation factor = 3x

We chose 10 objects from Caltech 101 dataset

File name	Grand_biano	Brain	Buddha	Butterfly	Ewer
Number of images	99	98	85	91	85
File name	helicopter	kangaroo	menorah	starfish	trilobite
Number of images	88	86	87	86	86

Then,

1. Loop over folders in dataset folder
2. Save label for every category and label name with folder name
3. Read every image after normalize it (image/255) and push it into images array
4. Resize all images with same size (SIZE variable)
5. Convert images and labels into np.array
6. Split images and labels to train and test
7. convert labels to categorial
8. define the hyperparameter (filters filtersize epochs batchsize input_shape strides padding activation pool_size rate)
9. initial model
10. add input layer
11. add hidden layers
    - convolutional
    - MaxPooling2D
    - BatchNormalization
    - Dropout
    - convolutional
    - MaxPooling2D
    - BatchNormalization
    - Dropout
    - Flatten
    - Dense
    - BatchNormalization
    - Dropout
    - Dense
    - BatchNormalization
    - Dropout
12. Add output layer
13. Fit the model

Experimental Results and discussions

We put the code and data on google colab and run it there.
At the beginning we initiate all model's parameters randomly.

Initial model summary:
Number of hidden layers = 15
Total number of images = 846 (15 % as testing data, 85% as training data, 15 % of the training data as validation data)
Number of filters = 32
Filter size = (3, 3)
Stride = (1, 1)
Padding = valid
Activation function = relu
Output activation function = softmax
Img SIZE= 128*128*3
5 epochs
Accuracy is = 11.811023950576782 %
Time = 60 sec

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 126, 126, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 63, 63, 32)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 63, 63, 32)        128
_________________________________________________________________
dropout (Dropout)            (None, 63, 63, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 61, 61, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 30, 30, 32)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 30, 30, 32)        128
_________________________________________________________________
dropout_1 (Dropout)          (None, 30, 30, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 28800)             0
_________________________________________________________________
dense (Dense)                (None, 512)               14746112
_________________________________________________________________
batch_normalization_2 (Batch (None, 512)               2048
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
batch_normalization_3 (Batch (None, 256)               1024
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                2570
=================================================================
Total params: 14,893,482
Trainable params: 14,891,818
Non-trainable params: 1,664
```

Then we made

some experiments on our model to obtain the optimal structure of our model and parameters values.

Structure Tests

Try 1: Change img SIZE= 200*200*3

Same accuracy = 11.811023950576782 %
Time = 150 sec, so back to SIZE 128*128*3

Try 2: Normalize images to from 0 to 1

Accuracy is = 18.897637724876404 %
Time = 60 sec
Higher accuracy so normalize images before training.

Try 3: increase filter size from (3, 3) to (5, 5)

Accuracy is = 21.259842813014984 %
Time = 60 sec
Higher accuracy so apply the change

Try 4: increase filter size from (5, 5) to (10, 10)

Accuracy is = 22.047244012355804 %
Time = 330 sec = 5:30 min!!!!
So back to filter size (5, 5)

Try 5: decrease number of filters from 32 to 16

Accuracy is = 18.897637724876404 %
Time = 30 sec
Smaller accuracy so back to number of filters = 32

Try 6: increase strides from (1, 1) to (3, 3)

Accuracy is = 17.322835326194763 %
Time = 6 sec
Smaller accuracy so back to strides from (1, 1)

Try 7: increase strides from (1, 1) to (2, 2)

Accuracy is = 20.472441613674164 %
Time =15 sec

Try 8: change stride to (2, 1)

Accuracy is = 18.897637724876404 %

Time =15 sec

Try 9: change stride to (1, 2)

Accuracy is = 14.960630238056183 %
Time = 15 sec
Smaller accuracy so back to Stride (1, 1)

Try 10: change padding from "valid" to "same"

Accuracy is = 18.110236525535583 %
Time = 55 sec
Smaller accuracy so back to “same”

Try 11: change activation function from relu to tanh

Accuracy is = 24.409449100494385 %
Higher accuracy but we will retest it after applying augmentation

Try 12: change activation function from tanh to sigmoid

Accuracy is = 12.598425149917603 %
Time = 55 sec
Smaller accuracy so back to relu

Try 13: change pool_size from (2, 2) to (1, 1) (effect of removing pooling layer)

Accuracy is = 14.173229038715363 %
Time = above 10 min!!
Smaller accuracy so back to pool_size (2, 2)

Try 14: change pool_size from (2, 2) to (5, 5)

Accuracy is = 33.07086527347565 %
Time = 50 sec
Higher accuracy and fastest time till now so we try to increase pool_size to:

(10, 10) => Accuracy is = 16.535432636737823 %
(7, 7) => Accuracy is = 28.346458077430725 %
(4, 4) => Accuracy is = 27.559053897857666 %
So pool_size (5, 5) is the highest accuracy and lowest time.

Try 15: change output activation function from softmax to softplus

Accuracy is = 28.346458077430725 %
Time = 50 sec
Smaller accuracy

Try 16: change output activation function to softsign

Accuracy is = 9.448818862438202 %
Time = 50 sec
Smaller accuracy so back to softmax

Try 17: duplicate the layers before and after flatten and change pool_size to (2, 2)

Accuracy is = 25.196850299835205 %
Time = 50 sec
Smaller accuracy so back to the previous number of layers

But after all this tests and changes in the model structure the accuracy is still too small so we tried to get more training data, the number of images for each object was small so we applied augmentation to the images we had to get more training data.

Apply data augmentation script

We generated and trained the model locally on PC because google colab resources weren't enough to read 18000 images

Try 1: apply data augmentation script for our 10 objects' folders

Time = 5 min
Input -> 846 output-> 18654
Uses 12gb/12/gb ram to read all images

Try 2: Trained the model on the new data and save the results

Final State

Final model summary:

Total number of images = 18654 (15 % as testing data, 85% as training data, 15 % of the training data as validation data)
Filters number = 32
Filter size = (5, 5)
Epochs = 100
Batch size = 128
Input shape = (SIZE, SIZE, 3)
Strides = (1, 1)
Padding = 'same'
Activation = 'tanh'
pool_size = (5, 5)
Rate = 0.2

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 128, 32)	2432
max_pooling2d (MaxPooling2D)	(None, 25, 25, 32)	0
batch_normalization (BatchNo	(None, 25, 25, 32)	128
dropout (Dropout)	(None, 25, 25, 32)	0
conv2d_1 (Conv2D)	(None, 25, 25, 32)	25632
max_pooling2d_1 (MaxPooling2	(None, 5, 5, 32)	0
batch_normalization_1 (Batch	(None, 5, 5, 32)	128
dropout_1 (Dropout)	(None, 5, 5, 32)	0
flatten (Flatten)	(None, 800)	0
dense (Dense)	(None, 512)	410112
batch_normalization_2 (Batch	(None, 512)	2048
dropout_2 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131328
batch_normalization_3 (Batch	(None, 256)	1024
dropout_3 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 10)	2570

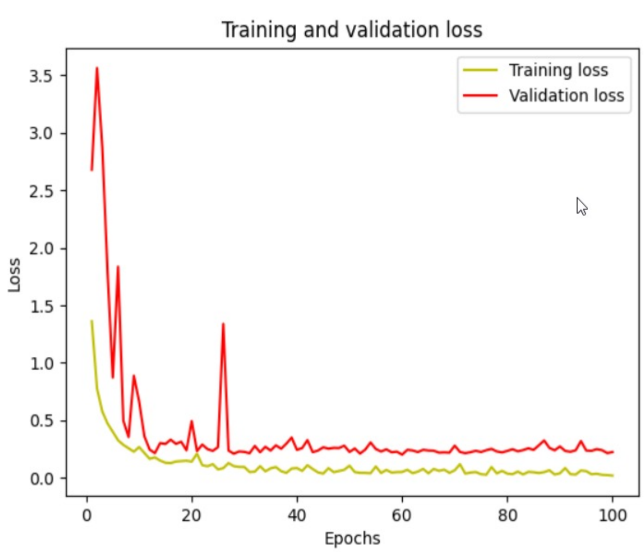
Total params: 575,402
Trainable params: 573,738
Non-trainable params: 1,664

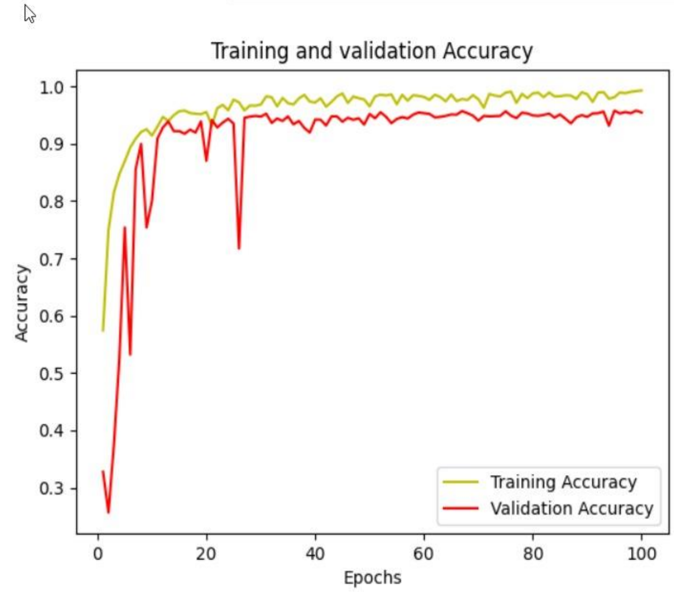
Results

Accuracy = 95.59455513954163 %

Time = 100 min

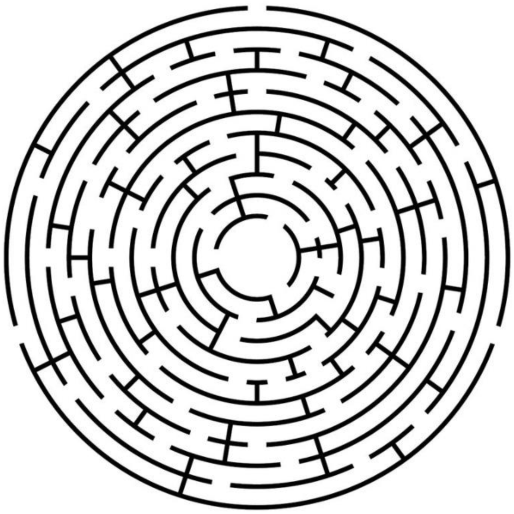
```
Epoch 80/100
106/106 [==============================] - 61s 578ms/step - loss: 0.0405 - accuracy: 0.9870 - val_loss: 0.2357 - val_accuracy: 0.9494
Epoch 81/100
106/106 [==============================] - 61s 578ms/step - loss: 0.0358 - accuracy: 0.9884 - val_loss: 0.2501 - val_accuracy: 0.9486
Epoch 82/100
106/106 [==============================] - 61s 579ms/step - loss: 0.0695 - accuracy: 0.9770 - val_loss: 0.2322 - val_accuracy: 0.9503
Epoch 83/100
106/106 [==============================] - 61s 577ms/step - loss: 0.0328 - accuracy: 0.9889 - val_loss: 0.2427 - val_accuracy: 0.9524
Epoch 84/100
106/106 [==============================] - 61s 577ms/step - loss: 0.0654 - accuracy: 0.9793 - val_loss: 0.2583 - val_accuracy: 0.9456
Epoch 85/100
106/106 [==============================] - 61s 578ms/step - loss: 0.0662 - accuracy: 0.9783 - val_loss: 0.2448 - val_accuracy: 0.9511
Epoch 86/100
106/106 [==============================] - 61s 580ms/step - loss: 0.0523 - accuracy: 0.9814 - val_loss: 0.2845 - val_accuracy: 0.9435
Epoch 87/100
106/106 [==============================] - 61s 580ms/step - loss: 0.0570 - accuracy: 0.9817 - val_loss: 0.3266 - val_accuracy: 0.9355
Epoch 88/100
106/106 [==============================] - 61s 580ms/step - loss: 0.0914 - accuracy: 0.9717 - val_loss: 0.2621 - val_accuracy: 0.9461
Epoch 89/100
106/106 [==============================] - 62s 582ms/step - loss: 0.0324 - accuracy: 0.9887 - val_loss: 0.2408 - val_accuracy: 0.9499
Epoch 90/100
106/106 [==============================] - 62s 581ms/step - loss: 0.0469 - accuracy: 0.9847 - val_loss: 0.2744 - val_accuracy: 0.9469
Epoch 91/100
106/106 [==============================] - 62s 581ms/step - loss: 0.1480 - accuracy: 0.9585 - val_loss: 0.2356 - val_accuracy: 0.9528
Epoch 92/100
106/106 [==============================] - 62s 583ms/step - loss: 0.0324 - accuracy: 0.9892 - val_loss: 0.2281 - val_accuracy: 0.9532
Epoch 93/100
106/106 [==============================] - 62s 586ms/step - loss: 0.0254 - accuracy: 0.9914 - val_loss: 0.2412 - val_accuracy: 0.9562
Epoch 94/100
106/106 [==============================] - 62s 582ms/step - loss: 0.0764 - accuracy: 0.9754 - val_loss: 0.3214 - val_accuracy: 0.9313
Epoch 95/100
106/106 [==============================] - 62s 581ms/step - loss: 0.0810 - accuracy: 0.9754 - val_loss: 0.2390 - val_accuracy: 0.9579
Epoch 96/100
106/106 [==============================] - 62s 580ms/step - loss: 0.0298 - accuracy: 0.9898 - val_loss: 0.2365 - val_accuracy: 0.9528
Epoch 97/100
106/106 [==============================] - 62s 581ms/step - loss: 0.0463 - accuracy: 0.9835 - val_loss: 0.2505 - val_accuracy: 0.9553
Epoch 98/100
106/106 [==============================] - 62s 582ms/step - loss: 0.0270 - accuracy: 0.9909 - val_loss: 0.2420 - val_accuracy: 0.9532
Epoch 99/100
106/106 [==============================] - 62s 581ms/step - loss: 0.0275 - accuracy: 0.9911 - val_loss: 0.2169 - val_accuracy: 0.9579
Epoch 100/100
106/106 [==============================] - 61s 580ms/step - loss: 0.0228 - accuracy: 0.9923 - val_loss: 0.2258 - val_accuracy: 0.9545
88/88 [==============================] - 3s 38ms/step - loss: 0.1975 - accuracy: 0.9559
Accuracy is =  95.59455513954163 %
```





Try to predict random image from the internet

Maze has features similar to brain so the model predicted it as a brain.



```
Image category maze
prediction label 0
label Name brain
```


```
Image category helicopter
prediction label 5
label Name helicopter
```

```
Image category kangaroo
prediction label 6
label Name kangaroo
```

Code

You can find the code here:
https://github.com/Ahmad-Helmy/pattern_recognation_cnn/tree/master

To start training the model:

Add any number of images folders (any number of objects) in the dataset folder then run the code of augmentation and training.

To start using our model:

Add the 10 folders we mentioned before (even they are empty) then run the load model.

References

- http://www.vision.caltech.edu/Image_Datasets/Caltech101/
- https://www.youtube.com/watch?v=R9PPxpzj5tI&list=PLZsOBAyNTZwbIjGnolFydAN33gyyGP7lT&index=74
- https://courses.analyticsvidhya.com/courses/convolutional-neural-networks-cnn-from-scratch?utm_source=blog&utm_medium=learn-image-classification-cnn-convolutional-neural-networks-3-datasets
- https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/
- https://keras.io/api/layers/convolution_layers/convolution2d/
- https://keras.io/api/layers/activations/
- https://keras.io/api/layers/core_layers/dense/
- https://keras.io/api/layers/
- https://keras.io/api/layers/pooling_layers/max_pooling2d/