OBJECT CLASSIFICATION USING CNN -Acc 0.8235

Given a set of images that are all labeled with a single category, I predicted these categories for a set of images and measure the accuracy of the predictions and generared the confusion matrix.

Instead of trying to specify what every one of the image categories of interest look like directly in code, they provide the computer with many examples of each image class and then develop learning algorithms that look at these examples and learn about the visual appearance of each class. In other words, they first accumulate a training dataset of labeled images, then feed it to the computer in order for it to get familiar with the data.

Given that fact, the complete image classification pipeline can be formalized as follows:

- Our input is a training dataset that consists of *421* images, each labeled with one of *4* different classes.
- Then, we use this training set to train a classifier to learn what every one of the classes looks like.
- In the end, we evaluate the quality of the classifier by asking it to predict labels for a new set of images that it has never seen before. We will then compare the true labels of these images to the ones predicted by the classifier.
- Finally calculated the confusion matrix.

Here I used a 5 layers Sequential Convolutional Neural Network for object classification trained on the dataset downloaded from IKEA site (https://www.ikea.com/). I choosed to build it with keras API (Tensorflow backend) which is very intuitive. Firstly, I will prepare the data then I will focus on the CNN modeling and evaluation.

I achieved **82.35%** of accuracy with this CNN trained.

Define the model

I used the Keras Sequential API, where you have just to add one layer at a time, starting from the input.

Here I used 5 convolution layers. The first is the convolutional (Conv2D) layer. It is like a set of learnable filters. I choosed to set 32 filters for the first three conv2D layers and 64 filters for the next ones. The last conv2D layer have 128 filters. Each filter transforms a part of the image

(defined by the kernel size) using the kernel filter. The kernel filter matrix is applied on the whole image. Filters can be seen as a transformation of the image.

The second important layer in CNN is the pooling (MaxPool2D) layer. This layer simply acts as a downsampling filter. It looks at the 2 neighboring pixels and picks the maximal value. These are used to reduce computational cost, and to some extent also reduce overfitting. We have to choose the pooling size (i.e the area size pooled each time) more the pooling dimension is high, more the downsampling is important.

Combining convolutional and pooling layers, CNN are able to combine local features and learn more global features of the image.

Dropout is a regularization method, where a proportion of nodes in the layer are randomly ignored (setting their wieghts to zero) for each training sample. This technique also improves generalization and reduces the overfitting.

'Relu' is the rectifier (activation function max(0,x). The rectifier activation function is used to add non linearity to the network.

The Flatten layer is use to convert the final feature maps into a one single 1D vector. This flattening step is needed so that you can make use of fully connected layers after some convolutional/maxpool layers. It combines all the found local features of the previous convolutional layers. In the end I used the features in two fully-connected (Dense) layers which is just artificial an neural networks (ANN) classifier. In the last layer(Dense(10,activation="softmax")) the net outputs distribution of probability of each class.

Once our layers are added to the model, we need to set up a score function, a loss function and an optimisation algorithm. I used a specific form for categorical classifications (>2 classes) called the "categorical_crossentropy" as a loss function.

I choosed Adam as optimizer, it is a Straightforward to implement,Computationally efficient, Little memory requirements and Well suited for problems that are large in terms of data and/or parameters.

Comparision with other architectures :-

I also created another two architectures for the object classification. They given very poor accuracy compared with this architecture(explained above). One of it given 68.235 % and another one given 24.7 % accuracy. They also specified in the folder **Another_Architectures** with the models.

Observations

The parameters of the model:-

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 398, 398, 32)      320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 199, 199, 32)      0
_________________________________________________________________
dropout_1 (Dropout)          (None, 199, 199, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 197, 197, 32)      9248
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 98, 98, 32)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 98, 98, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 96, 96, 32)        9248
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 48, 48, 32)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 48, 48, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 46, 46, 64)        18496
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 23, 23, 64)        0
_________________________________________________________________
dropout_4 (Dropout)          (None, 23, 23, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 21, 21, 128)       73856
_________________________________________________________________
dropout_5 (Dropout)          (None, 21, 21, 128)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 56448)             0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               7225472
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 516
=================================================================
Total params: 7,337,156
Trainable params: 7,337,156
Non-trainable params: 0
_________________________________________________________________
```

The total parameters of the model = 73,37,156
Accuracy = 82.35 %

Confusion Matrix:-

```
Confusion Matrix :
[[51  0  0  0]
 [ 0 28  0  0]
 [ 0  0 62  0]
 [ 0  0  0 29]]
Accuracy Score : 1.0
Report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        51
           1       1.00      1.00      1.00        28
           2       1.00      1.00      1.00        62
           3       1.00      1.00      1.00        29

   micro avg       1.00      1.00      1.00       170
   macro avg       1.00      1.00      1.00       170
weighted avg       1.00      1.00      1.00       170
```