UNIVERSITY OF TRIESTE

Computer vision project 3
Image classifier based on convolutional neural networks

Giovanni Pinna, Adriano Tumino

Academic Year 2020-2021

Summary

Introduction

In this report we will comment on the both practical and theoretical choices made during the implementation of the project. In particular, the code was written with the Python programming language and is organized in:

1. Point_One
2. Point_Two
3. Optional_Two
4. Point_three
5. Optional_three
6. Utils: function for data processing and plotting are implemented

Problem statement

This project requires the implementation of an image classifier based on convolutional neural networks. The provided dataset (from [Lazebnik et al., 2006]), contains 15 categories (office, kitchen, living room, bedroom, store, industrial, tall building, inside city, street, highway, coast, open country, mountain, forest, suburb), and is already divided into training set and test set. Images of our available dataset are split between 1500 in training set and a 2985 in test set.

Overview Workflow

1. In The First Part I simply implemented, using Keras, the shallow convolutional neural network requested by the exercise and I report the results in terms of accuracy, validation loss and confusion matrix. Obviously before training the CNN I had to preprocess my data in a way to make them "feasible" for my CNN (later on I will explain better how I dealt with data).
2. In the second part I tried to improve test accuracy of my CNN through several techniques, for example data augmentation, insertion of some dropout and batch normalization layer and the so called "ensemble of network". My aim is to reach at least 60% of the test accuracy. After that I applied data augmentation (crop, rotation and mirrored)and I compared the results.
3. In the last part I used a pre-trained CNN called VGG16 in two ways: first of all, I will freeze all weights except those of the last layer and I'll train only these ones and then I will exploit feature from the last convolutional layer in order to apply a linear SVM. After that we repeated all the process before the data augmentation.

Part One

Preprocessing

The data are organized as follows:

1. In the training dataset there are 1500 images divided into 15 categories
2. In the test dataset there are 2985 images divided into 15 categories

The images in the training dataset were used to train the neural network and choose the best parameters, while the test dataset to test neural network performance. To optimize the work, the function of *load_image(path,labels, dimension)* was written. This function inputs the path of where the images are, the labels are the various categories, and the size is used to define the size of the output image. The image size is set by default to 64x64. As output we get a numpy array the size of 64x64 and a category list where every 64x64 numpy array is

associated with a category of the initial 15. This function runs for both training images and images in the test dataset.
Last thing to do is to perform a reshaping of the train and test dataset. In particular, I add a dimension in order to specify the number of channel of each image (dealing with black and white images, it is equal to one).

Crating CNN

First of all I created the simplest neural network as required by the project. This neural network has been implementing using the Keras library available for the python programming language. The specifications of this simple neural network are:

1. The SGD ("stocastic gradient descendant") and the batch_size=32
2. Nesterov momentum equals 0.9 to increase model accurancy
3. The training dataset was divided into 2 subgroups, one to train the model (85%) and one to validate (15%), and then identify the best parameters for the model
4. No dropout level was entered but to decrease overfitting it was decided to use the technique called "early stopping" with a min_delta=0.10 and a patience=10

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 62, 62, 8)         80
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 31, 31, 8)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 29, 29, 16)        1168
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 12, 12, 32)        4640
_________________________________________________________________
flatten (Flatten)            (None, 4608)              0
_________________________________________________________________
dense (Dense)                (None, 15)                69135
_________________________________________________________________
dense_1 (Dense)              (None, 15)                240
=================================================================
Total params: 75,263
Trainable params: 75,263
Non-trainable params: 0
_________________________________________________________________
```

Figure1: Represents the structure of the first CNN implemented

Results

During the treaning thanks to the "early stopping" the training lasted 23 eras and the following values were obtained for test-loss and test-accurancy:

```
loss: 3.2441 - accuracy: 0.3715
```

Obviously being CNN based on a stochastic part with each execution the result will change a bit. But let's say that to be a very simple CNN you have already achieved good results and it can be said that it manages to correctly predict a third of the images that are provided to it.

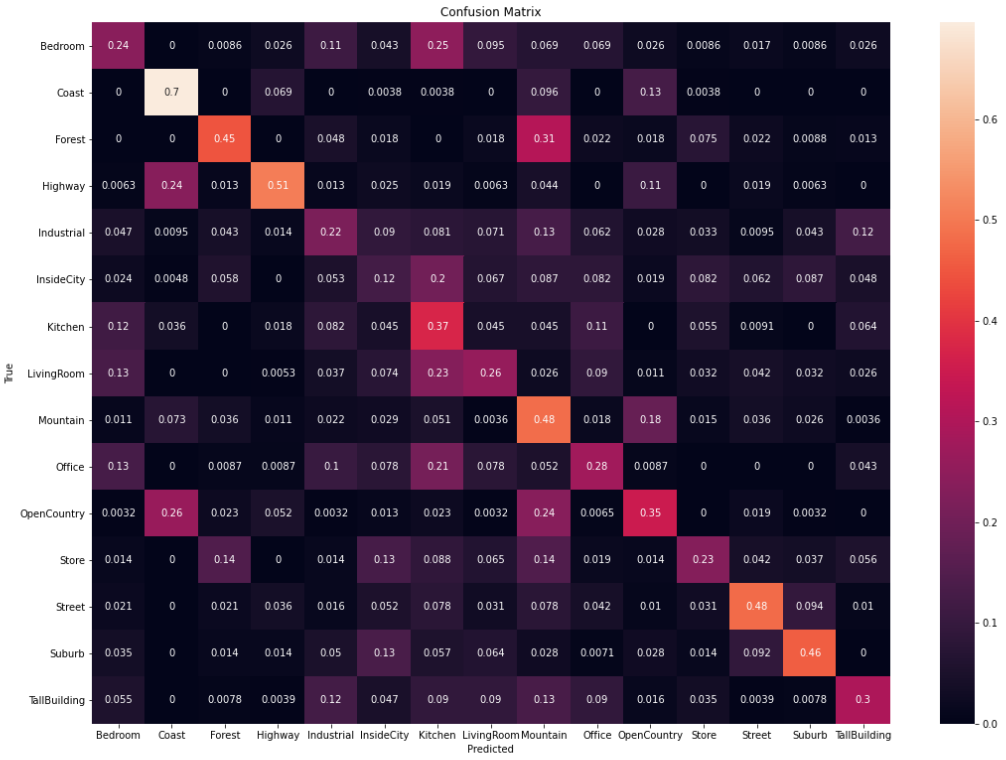

Figure2: Confusion matrix of the simplest CNN

We can observe in the confusion matrix that the accuracy can vary from class to class; that have an accuracy of near 50% and it means that these two class are easily identifiable. Other classes have a very low accuracy, so there are much more uncertainty on deciding the right class.

You can also go and see the charts: validation loss, validation accuracy, loss, accuracy

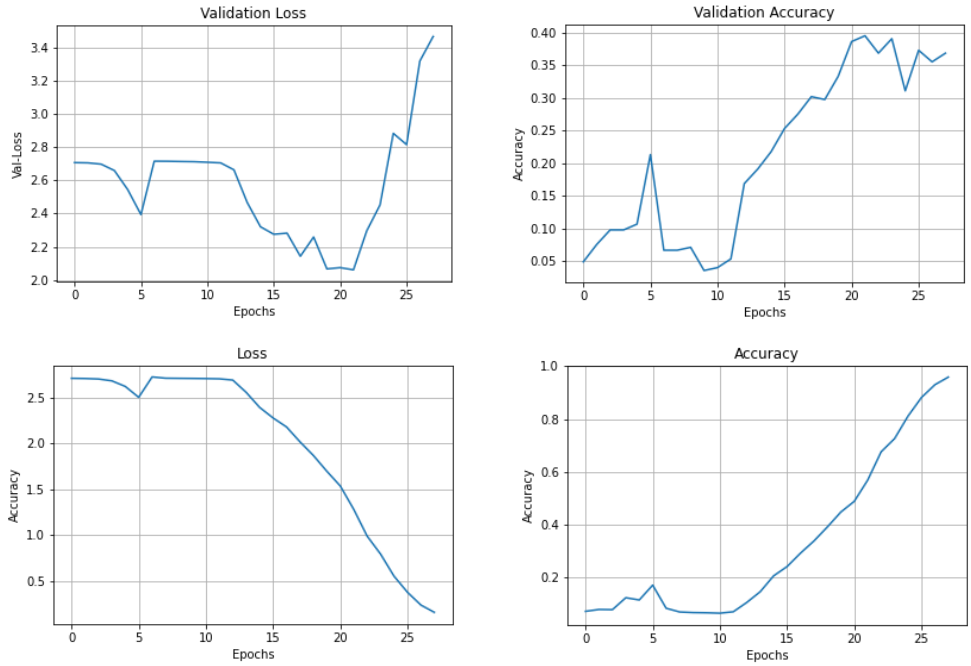

Figure3: valdation loss, validation accurancy, loss and accuracy of the simplest CNN

Part two

In this second part the same techniques will be implemented as the first, but to increase even more the accuracy we will increase our training set with data augmentation.

In particular, we will use these two techniques to increase data:

1. Left-to-right reflections
2. Cropping of images (takes only part of the image)

So by applying these techniques to our training set we find ourselves moving from a training set of 1500 images to one of 4500 images. (The algorithms for data augmentation have always been implemented in python and are located in the dataUtils script.)

Just with these simple modifications and using the same CNN developed in the first part, I reached a quite good implementation:

```
loss: 3.7471 - accuracy: 0.3960
```

Batch normalization

Another two very common technique to improve results of a CNN are dropout, which avoids the possibility of overfitting, and batch normalization, which controls the values of weights and bias avoiding the risk of gradient explosion. Moreover another thing to underline is the fact that I change the optimizer from SGD to ADAM with the default parameters. I add some layers of dropout and batch normalization on my previous CNN, obtaining the following structure:

```
Model: "sequential_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_45 (Conv2D)           (None, 62, 62, 8)         80
_________________________________________________________________
module_wrapper_39 (ModuleWra (None, 62, 62, 8)         248
_________________________________________________________________
max_pooling2d_30 (MaxPooling (None, 31, 31, 8)         0
_________________________________________________________________
dropout_26 (Dropout)         (None, 31, 31, 8)         0
_________________________________________________________________
conv2d_46 (Conv2D)           (None, 29, 29, 16)        1168
_________________________________________________________________
module_wrapper_40 (ModuleWra (None, 29, 29, 16)        116
_________________________________________________________________
max_pooling2d_31 (MaxPooling (None, 14, 14, 16)        0
_________________________________________________________________
dropout_27 (Dropout)         (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_47 (Conv2D)           (None, 12, 12, 32)        4640
_________________________________________________________________
flatten_15 (Flatten)         (None, 4608)              0
_________________________________________________________________
dense_30 (Dense)             (None, 15)                69135
_________________________________________________________________
module_wrapper_41 (ModuleWra (None, 15)                60
_________________________________________________________________
dense_31 (Dense)             (None, 15)                240
=================================================================
Total params: 75,687
Trainable params: 75,475
Non-trainable params: 212
_________________________________________________________________
```

Figure4: structure of the CNN model with adam optimizer 3x3

We also tried with The Adam5x5 and Adam7x7 models. Where the accuracy and loss of the various models is:

```
loss: 1.8093 - accuracy: 0.4834
loss: 1.7816 - accuracy: 0.5146
loss: 1.8224 - accuracy: 0.4576
```

Where we notice that everyone has similar performance and that the best accuracy in this case we have with the model that uses Adam5x5 which achieves 50% accuracy.

Ensemble Of Network

The idea is to combine a CNN and the use of the majority vote in the average of the prediction probabilities. We andnsemble a new network with 10 iterations.

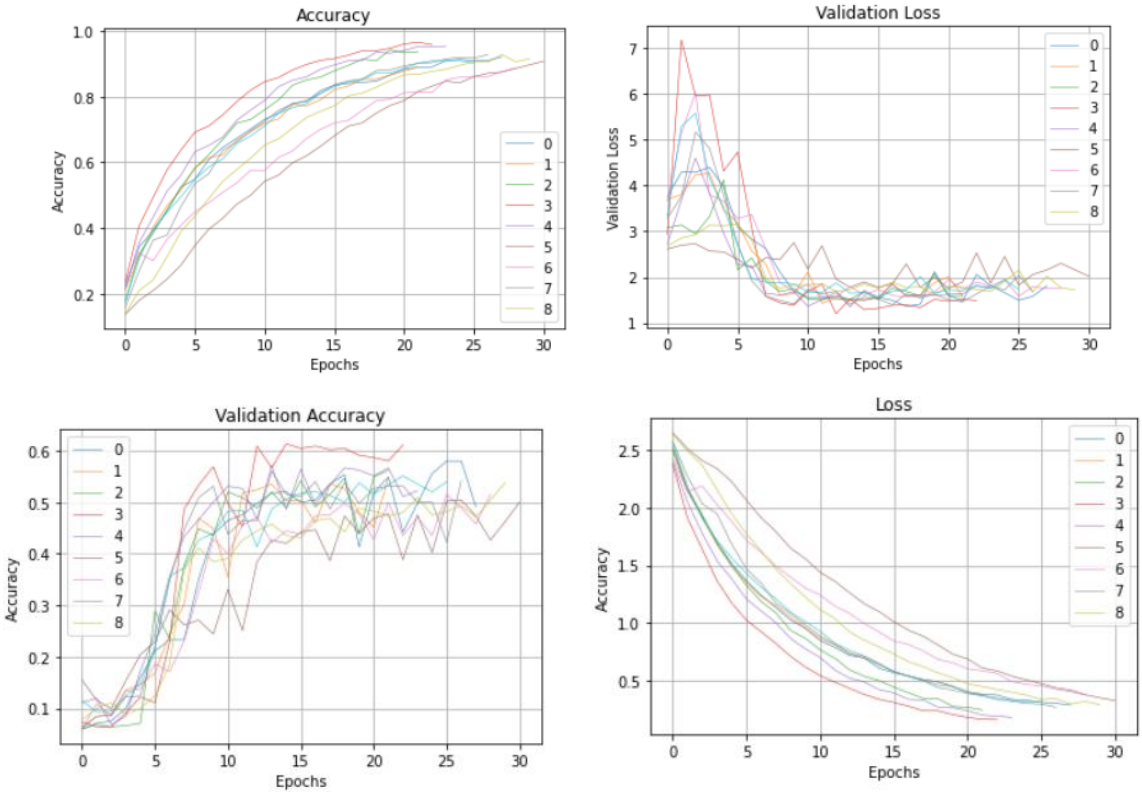

Figure5: accuracy, validation loss, valoidation accuracy and loss for each CNN used in the voting process

These above are the charts of the 10 networks created. In fact, from the graphs we see how with this technique we can achieve accurateness of up to 60%.

To achieve these results, I wrote some functions (listed below):

1. ensemble of network: This functions trains several independent and identical CNNs (the type of the CNN is a parameter) and then calculates the mean of the probability density function calculated in the test class (details are in the Jupyter notebook script).
2. calculate accuracy: It calculates the accuracy for the average ensemble of networks.
3. calculate accuracy with voting: It calculates the accuracy for the voting ensemble of networks.

Optional two

In this part we try to improve the results obtained by increasing the data even more, in particular by cropping and rotation of images randomly, so as to reach the volume of 6000 images to be used as training data. But where we get results similar to those we found earlier.

Another thing we tried to improve the results obtained above was to create a CNN network with multiple layers. In particular we tried with the following network:

```
Model: "sequential_28"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_84 (Conv2D)           (None, 62, 62, 8)         80
_________________________________________________________________
max_pooling2d_56 (MaxPooling (None, 31, 31, 8)         0
_________________________________________________________________
conv2d_85 (Conv2D)           (None, 29, 29, 16)        1168
_________________________________________________________________
max_pooling2d_57 (MaxPooling (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_86 (Conv2D)           (None, 12, 12, 32)        4640
_________________________________________________________________
max_pooling2d_58 (MaxPooling (None, 6, 6, 32)          0
_________________________________________________________________
conv2d_87 (Conv2D)           (None, 4, 4, 32)          9248
_________________________________________________________________
conv2d_88 (Conv2D)           (None, 2, 2, 32)          9248
_________________________________________________________________
flatten_28 (Flatten)         (None, 128)               0
_________________________________________________________________
dense_56 (Dense)             (None, 15)                1935
_________________________________________________________________
dense_57 (Dense)             (None, 15)                240
_________________________________________________________________
dense_58 (Dense)             (None, 15)                240
=================================================================
Total params: 26,799
Trainable params: 26,799
Non-trainable params: 0
_________________________________________________________________
```

Unfortunately, this model was more difficult and slower to train, without much better results. In fact, with this model we cannot exceed 50%.

Part three

In the last part I will do something different; instead of building my own CNN from scratch, I will exploit the features of pre-trained CNN in two different ways:

1. Freeze the weights of all the layers but the last fully connected layer and fine-tune the weights of the last layer with the same data as before (not augmented).
2. Employ the pre-trained network as a feature extractor, accessing the activation of the last convolutional layer and train a multiclass linear SVM.

I decided to use a convolutional neural network called VGG16, which is included in keras package and it has got the following structure:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
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
_________________________________________________________________
dense (Dense)                (None, 15)                61455
=================================================================
Total params: 134,321,999
Trainable params: 61,455
Non-trainable params: 134,260,544
_________________________________________________________________
```

Freezing the Weights

As I mentioned before, The first part is dedicated to train only the last fully connected component of the VGG16, while the remaining weights are freezed. Obviously, since VGG16 has been written for resolving a classification problem between 1000 different classes, I had to change the last classification layer with a layer conform to our problem. For what concern the processing of data, I used the package image from *keras.preprocessing* in order to make data suitable for the VGG16. Notice that, respect to the previous part, images now has a shape of 224x224. In particular, function preprocess input is meant to adequate your image to the format the model requires. I wrote a function, called vgg16 model, which initializes the right model and Icompile it. As Optimizer I used
Adam with default values and as loss function I decide to use categorical cross-entropy.

Results

I trained the weights of the last fully connected layer for 10 epochs8 and using a batch-size equal to 64 and I obtained good results, better than all the other one.

```
loss: 2.8106 - accuracy: 0.7652
```

The related matrix confusion is:

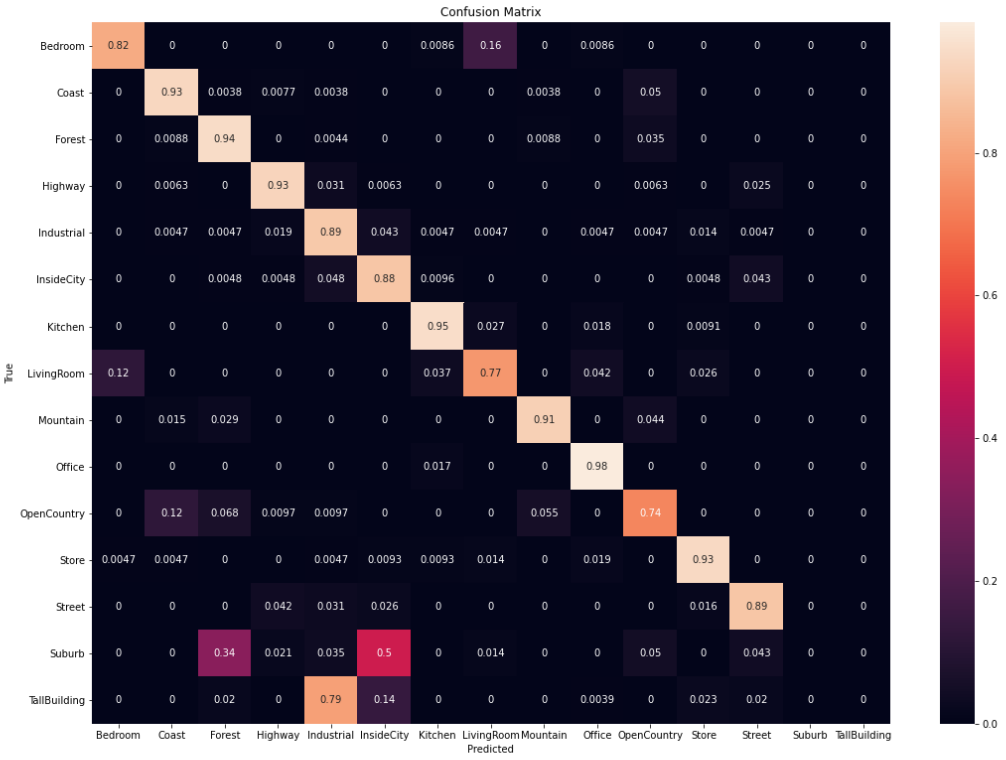

Figure6: confusion matrix before used the VGG16 network

Features extractions

Last thing to try now is to exploit the CNN not for classifying directly classes of images, but to extract features to use after in another classification technique. In our case we will perform a linear SVM. In particular we will extract features obtained in the last convolutional layer of the VGG16. The kind of structure of the features that we have, for each image, is: we will obtain 512 features, each of them is a 7x7 grid of values. To make these feature suitable for an SVM, we will attend all the 512 features related to a specific image in a unique vector.
Once we have obtained features from images, we simply use it in a SVM. This is done in the function called predict with linear SVM, which returns the accuracy of technique based on SVM. Being a multiclass classification problem, it is necessary to decide what strategy one has to use for classification. We adopted a one-against-rest approach in which the winning class is the one that maximizes the distance from the separating hyperplane. For what concern the obtained result, we can observe that the test accuracy overcomes 86 %, indeed:

```
Accuracy value:  0.8991624790619765
```

Here the table that contain the true positive, true negative, false positive and false negative:

	labels	True positive	True negative	False positive	False negative
0	Bedroom	91	2843	26	25
1	Coast	237	2695	30	23
2	Forest	220	2734	23	8
3	Highway	151	2815	10	9
4	Industrial	169	2765	9	42
5	InsideCity	178	2761	16	30
6	Kitchen	101	2863	12	9
7	LivingRoom	147	2762	34	42
8	Mountain	256	2691	20	18
9	Office	113	2856	14	2
10	OpenCountry	254	2633	42	56
11	Store	202	2755	15	13
12	Street	180	2775	18	12
13	Suburb	140	2835	9	1
14	TallBuilding	245	2706	23	11

Figure7: features extracted form the VGG16 model

And the related matrix confusion for this model.

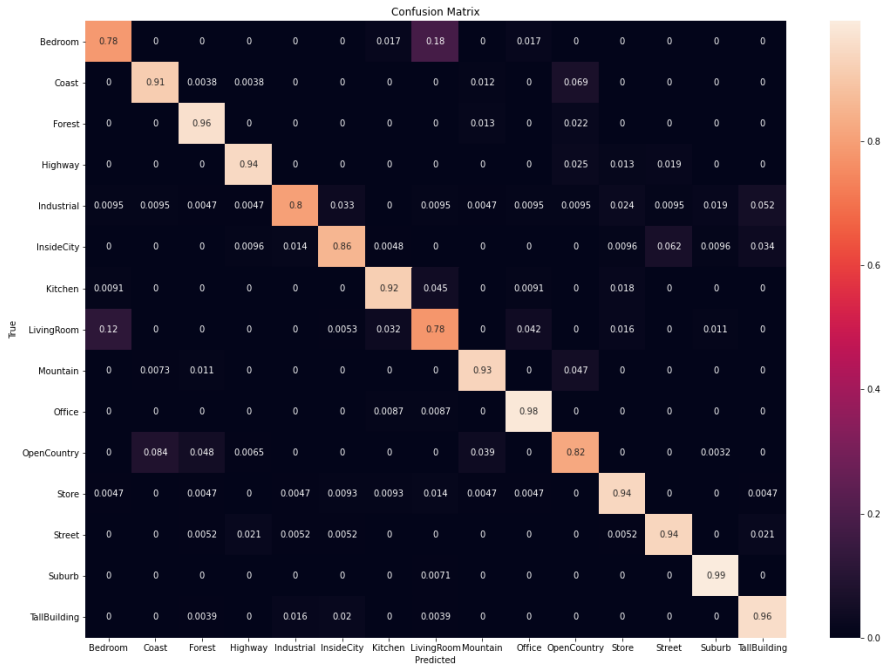

Figure8: confusion matrix before the training of the SVM on the VGG16

It can be seen that in prediction in each category is very accurate. Touching the accuracy rate around 90%. The only two classes that the model predicts with the most difficulty (we deduce from the fact that they have an accuracy of only 78%) only "Bedroom" and "Livingroom". This seems quite normal since they are two very similar entities and are located in the same environment and may have similar objects that make them up (furniture, bed and sofa are similar).

In any case, it can be said with certainty that transfert learning is the technique that has brought the best results.

Conclusion

In this project we have seen how a very simple CNN is able to categorize the various images with sum-up good results. These performance can be easily increased by using techniques such as batch normalization and data augmentation. This allows you to easily achieve an accuracy of around 60%. In any case, as already pointed out in point three the best technique, that is, the technique that in this case has reached the highest degree of accuracy is that of transfert learning. In particular using the VGG16 network, which has already been trained for many hours and with a lot of data, we are able to achieve a very high degree of accuracy. This is mainly due to the fact that it is only necessary to train the last layer of the network since it is the one that captures the details of the images, so it is the layer that we are interested in customizing. In addition, by extracting the characteristics of the last layer and taking advantage of the linear SVM we obtain a very accurate precision.