Computer Vision and pattern recognition exam project

Eleonora Donadini

February 2020

1 Introduction

The aim of this project is to implement an image classifier based on convolutional neural networks trying each step to increase the accuracy of classification. The improvement are generated by the change of the parameters and approach adopted. The used dataset is the one provided by the assignment (from [Lazebnik et al., 2006]). It collects 15 categories and is already divided in training set and test set containing respectively 2985 imagines and 1500 imagines.

2 First task

The model in figure 1 represent the blueprint to follow to build a first network.

#	type	size
1	Image Input	64×64×1 images
2	Convolution	8 3×3 convolutions with stride 1
3	ReLU	
4	Max Pooling	2×2 max pooling with stride 2
5	Convolution	16 3×3 convolutions with stride 1
6	ReLU	
7	Max Pooling	2×2 max pooling with stride 2
8	Convolution	32 3×3 convolutions with stride 1
9	ReLU	
10	Fully Connected	15
11	Softmax	softmax
12	Classification Output	crossentropyex

Figure 1: Layout of the provided CNN

2.1 Data preprocessing

As we can see the imput image is 64x64x1. The first two numbers refers to the size while the third to the channel (color or black and white) of the image. For that reason in order to feed the images to the network we have to resize the them to 64x64 and to set the channel to 1 (black and white image).
We must be careful to perform an anisotropic transformation rescaling the whole image independently along x and y to get the proper size, unless we could lose image data. Python provide proper functions to perform these tasks.

The foward step is to split the provided training set in 85% for actual training set and 15% to be used as validation set.

```
X_train, X_validation, y_train, y_validation = train_test_split(
    X_train_raw, y_train_one_hot, train_size=0.85, random_state=42)
```

2.2 The model

As request we employ the stochastic gradient descent with momentum optimization algorithm, using the default parameters, except for those specified. We set mini batches of size 32 and initial weights drawn from a Gaussian distribution with a mean of 0 and a standard deviation of 0.01. We also set the initial bias values to 0. Our final model is:

```
base_model = Sequential([
    Conv2D(8, 3,strides=1, padding='same',input_shape=(64,64,1)),
    Activation('relu'),
    MaxPooling2D(pool_size=2,strides=2),
    Conv2D(16, 3,strides=1, padding='same'),
    Activation('relu'),
    MaxPooling2D(pool_size=2,strides=2),
    Conv2D(32, 3,strides=1, padding='same'),
    Activation('relu'),
    Flatten(),
    Dense(15, activation='softmax',
        kernel_initializer=norm,
        bias_initializer='zeros')
])

base_model.compile(loss='categorical_crossentropy',
                   optimizer=sgd,
                   metrics = ['accuracy'])
```

In this model we don't add any dropout layer, but to reduce overfitting we used the so called technique of early Stopping with patience = 10 and as monitor the val_loss. The monitor allows to specify the performance measure to monitor in order to end training, in this case we chose the value of the loss on the validation dataset, val_loss. The patience instead set the number of epochs with no improvement after which training will be stopped. This stopping criterion will be used also for the cnn developed in the following tasks.

2.3 Results

Here the plots of loss and accuracy over the epochs during the training, for both the training set and the validation set. The training stopped after 12 epochs. As we can see this model is probably affected by overfitting because the accuracy on the train set linearly increase while in the validation set stabilizes around 30%. This is probably due to fact that the data set uses for training is very small.

```
Train set dimintion:  1275 images
Validation set dimintion:  225 images
```

We reached the results shown on figure 2:

Figure 2: Accuracy and Loss if both training and validation set over the number of epochs.

In terms of test-loss and test accuracy, we reached the following results:

```
loss: 3.18
accuracy: 0.37
```

We can observe a very high loss. Due to the stochastic part of the model these results may vary but on average we can see that accuracy stabilizes around 30%.

We can now take a look to the confusion matrix, figure 3.
We can observe that the accuracy can vary from class to class, for example, class "Street", "Highway" and "TallBuilding" have an accuracy of over 50%, it means that these classes are easily identifiable. Other classes, such as "Kitchen"

or "LivingRoom" have a very low accuracy, so there are much more uncertainty to identify.

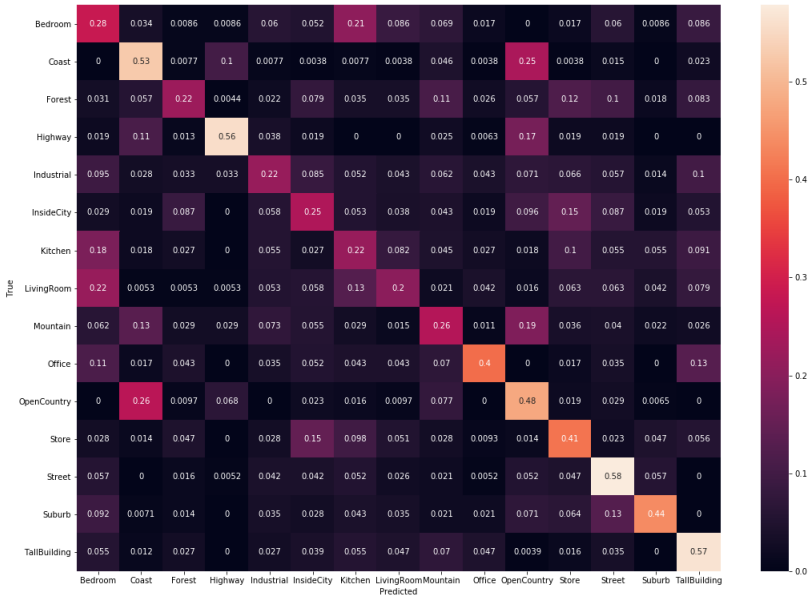

Figure 3: Confusion matrix base model.

Finally the following classification report summarize all of the score we got.

Classification Report	precision	recall	f1-score	support
Bedroom	0.24	0.28	0.26	116
Coast	0.40	0.44	0.42	260
Forest	0.37	0.34	0.35	228
Highway	0.65	0.56	0.60	160
Industrial	0.28	0.12	0.17	211
InsideCity	0.21	0.19	0.20	208
Kitchen	0.13	0.32	0.18	110
LivingRoom	0.28	0.27	0.27	189
Mountain	0.34	0.33	0.33	274
Office	0.33	0.46	0.38	115
OpenCountry	0.42	0.46	0.44	310
Store	0.47	0.19	0.27	215
Street	0.46	0.53	0.50	192
Suburb	0.51	0.51	0.51	141

TallBuilding	0.56	0.55	0.55	256
accuracy			0.37	2985
macro avg	0.38	0.37	0.36	2985
weighted avg	0.39	0.37	0.37	2985

3 Second Task

The aim of this part is to exploit some more techniques in order to obtain a test accuracy of about 60%. We focus on:

1. data augmentation
2. batch normalization
3. change the size and/or the number of the convolutional filters
4. switch to Adam optimizer
5. dropout
6. ensemble of networks technique

3.1 Data augmentation

To reduce overfitting we increase the training data set applying the so called technique "data augmentation". With this technique we augmented the number of data with simple transformation: left-to-right reflections and rotation of 45 degrees.

```
data_generator = ImageDataGenerator(rotation_range=45,
                                    horizontal_flip=True)
```

Just with these simple modifications and using the same cnn developed in the first part, we reached a quite good implementation of the results of accuracy and loss on the train and validation sets, figure 4: And also on the test set:

Figure 4: Accuracy and loss for training and validation set for base model with augmented data over the numbers of epochs.

```
loss: 1.68
accuracy: 0.46
```

In the figure 5 we can se the now the cnn presents some remarkable improvements in the identification of "Coast" and "Suburb".

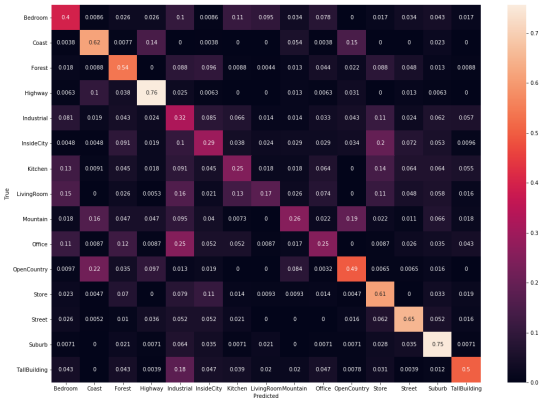

Figure 5: Confusion matrix of base model with augmented data

3.2 Batch normalization

Batch normalization enables the use of higher learning rates, greatly accelerating the learning process. We develop the following cnn.

```
norm_model = Sequential([
        Conv2D(8, 3,strides=1, padding='same',input_shape=(64,64,1)),
        BatchNormalization(axis=-1),
        Activation('relu'),
        MaxPooling2D(pool_size=2,strides=2),
        Conv2D(16, 3,strides=1, padding='same'),
        BatchNormalization(axis=-1),
        Activation('relu'),
        MaxPooling2D(pool_size=2,strides=2),
        Conv2D(32, 3,strides=1, padding='same'),
        BatchNormalization(axis=-1),
        Activation('relu'),
        Flatten(),
        Dense(512),
        Activation('relu'),
        Dense(NUM_CLASSES, kernel_initializer=norm,
        bias_initializer='zeros'),
        Activation('softmax')])

    norm_model.compile(loss='categorical_crossentropy',
    optimizer=sgd,
```

```
    metrics = ['accuracy'])
```

We try to run the cnn with the base training set and we get fast an accuracy equals to 1, figure 6. With this cnn we get one of the best performance on the

Figure 6: Accuracy and loss for training and validation set for model with added normalization layers over the numbers of epochs.

test set:

```
loss: 2.04
accuracy: 0.55
```

The confusion matrix as in figure 7:

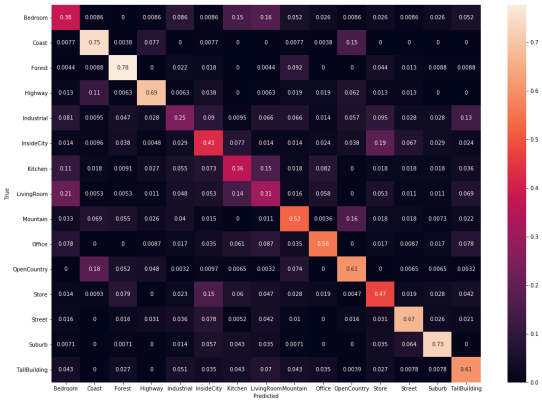

Figure 7: Confusion matrix for cnn with normalization layers.

A high accuracy in the training set may be a sign of overfitting so to try to reduce it we run the same algorithm with augmented data and we get a better behaviour but a less accuracy on the test set:

```
loss: 2.25
accuracy: 0.40
```

3.3 Adam Optimizer and change in number of convolutional filters

The Adam optimization algorithm is an extension to stochastic gradient descent that has recently seen broader adoption for deep learning applications in computer vision and natural language processing. Stochastic gradient descent maintains a single learning rate (termed alpha) for all weight updates and the learning rate does not change during training while update network weights iterative based in training data. We train the following cnn.

```
model_res_norm = Sequential([
    Conv2D(8, 3,strides=1, padding='same',input_shape=(64,64,1)),
    BatchNormalization(axis=-1),
    Activation('relu'),
    MaxPooling2D(pool_size=2,strides=2),
    Conv2D(16, 3,strides=1, padding='same'),
    Conv2D(16, 5,strides=1, padding='same'),
    Activation('relu'),
    MaxPooling2D(pool_size=2,strides=2),
    Conv2D(32, 7,strides=1, padding='same'),
    Activation('relu'),
    Flatten(),
    Dense(NUM_CLASSES, kernel_initializer=norm,
    bias_initializer='zeros'),
    Activation('softmax')
    ])

    model_res_norm.compile(loss='categorical_crossentropy',
    optimizer='adam',
    metrics = ['accuracy'])
```

We apply the same optimization with the base data set and the augmented ones and we get quite good results in both cases, for validation and training set as we can see in the figure, 8. In average we get an accuracy of 45% in the test

Figure 8: Accuracy and loss model with adam optimizer and an increase number of convolutional filters over the numbers of epochs.

set. The best confusion matrix that we obtain with this design is in figure, 9.

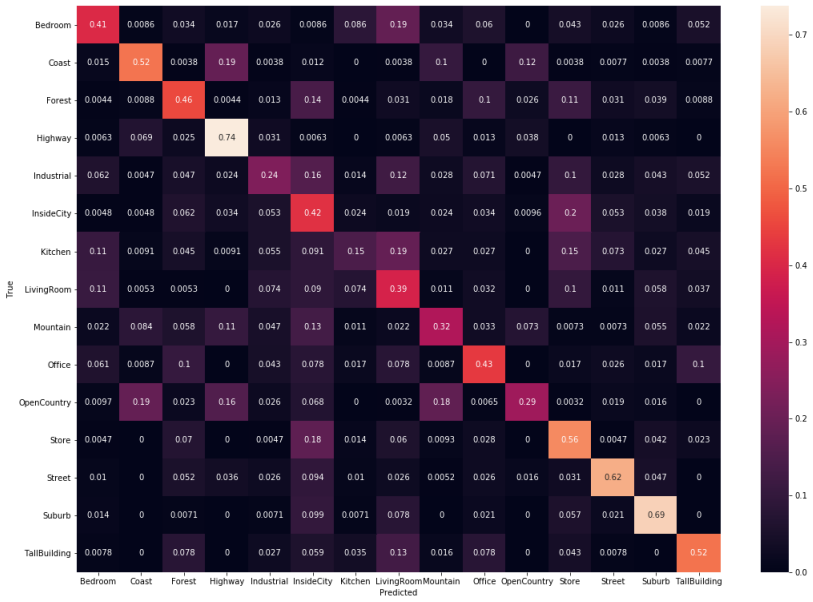

Figure 9: Confusion matrix for cnn with Adam optimizer.

The choice of optimization Adam algorithm can sped up the performance for that reasons we have increase the number of convolutional filters as we moved from input to output (we set 3x3, 5x5, 7x7 convolutional filters), without much increasing the computational time. For that reasons for the following models we skip to Adam optimizer.

3.4 Dropout

We add some dropout layer to improve regularization. Introducing dropout layers to the network is a form of regularization that forces the weights in the network to take only small values, which makes the distribution of weight values more regular and the network can reduce overfitting on small training examples.

```
model_dr = Sequential([
    Conv2D(8, 3,strides=1, padding='same',input_shape=(64,64,1)),
    BatchNormalization(axis=-1),
    Activation('relu'),
    MaxPooling2D(pool_size=2,strides=2),
    Dropout(0.2),
    Conv2D(16, 3,strides=1, padding='same'),
    Conv2D(16, 5,strides=1, padding='same'),
```

```
    Activation('relu'),
    MaxPooling2D(pool_size=2,strides=2),
    Dropout(0.2),
    Conv2D(32, 7,strides=1, padding='same'),
    Activation('relu'),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(NUM_CLASSES, kernel_initializer=norm,
    bias_initializer='zeros'),
    Activation('softmax')
    ])
    model_dr.compile(loss='categorical_crossentropy',
    optimizer='adam',
    metrics = ['accuracy'])
```

With this changes we get an accuracy and a loss in the test set of:

```
loss: 2.23
accuracy: 0.40
```

And a confusion matrix as the figure, 10

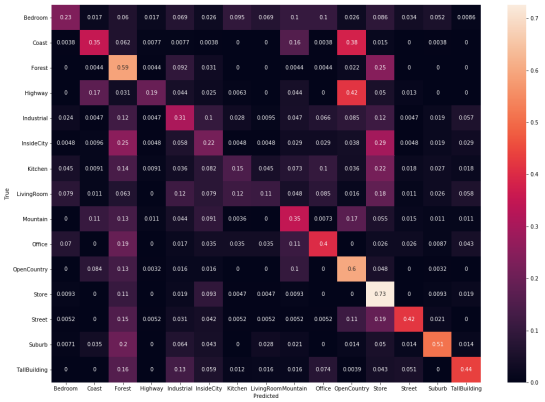

Figure 10: Confusion matrix of model with added dropout layers.

3.5 Ensemble of Networks

Another very important technique to improve results is to train independently a several number of cnn of the same type and then find a method to "merge" different outputs in a smart way to make the results better and much robust. To get better results we must take into account that the cnn chosen, must have precision about the same. For that reason we chose 8 of the cnn implemented in the previous part and we merge the different outputs. The chosen cnn have

different number of layers and different optimizers: sgd and adam and as for example the following cnn.

```
model_dr_mod = Sequential([
    Conv2D(8, 3,strides=1, padding='same',input_shape=(64,64,1)),
    BatchNormalization(axis=-1),
    Activation('relu'),
    MaxPooling2D(pool_size=2,strides=2),
    Dropout(0.2),
    Conv2D(16, 3,strides=1, padding='same'),
    Activation('relu'),
    MaxPooling2D(pool_size=2,strides=2),
    Conv2D(32, 5,strides=1, padding='same'),
    Activation('relu'),
    MaxPooling2D(pool_size=2,strides=2),
    Dropout(0.2),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(NUM_CLASSES, kernel_initializer=norm,
    bias_initializer='zeros'),
    Activation('softmax')
    ])
    model_dr_mod.compile(loss='categorical_crossentropy',
    optimizer='adam',
    metrics = ['accuracy'])
```

Doing that we use the voting approach: determine the class of a test images by voting, so the class which is chosen most of the time by the different convolutional neural network. We get the following results:

```
Classification Report
              precision    recall  f1-score   support

     Bedroom       0.42      0.34      0.38       116
       Coast       0.65      0.60      0.62       260
      Forest       0.52      0.82      0.64       228
     Highway       0.74      0.67      0.70       160
  Industrial       0.29      0.41      0.34       211
  InsideCity       0.34      0.55      0.42       208
     Kitchen       0.40      0.21      0.28       110
  LivingRoom       0.58      0.24      0.34       189
    Mountain       0.60      0.40      0.48       274
      Office       0.58      0.40      0.47       115
 OpenCountry       0.57      0.67      0.61       310
       Store       0.51      0.54      0.53       215
      Street       0.86      0.65      0.74       192
      Suburb       0.68      0.74      0.71       141
TallBuilding       0.74      0.65      0.69       256

    accuracy                           0.55      2985
   macro avg       0.57      0.53      0.53      2985
weighted avg       0.57      0.55      0.54      2985
```

Note the averal accuracy in the traing set around 55%.

4 Third Task

In this last part we exploit the features of a 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.
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: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
=================================================================
Total params: 14,714,688
```

```
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

4.1 Freezing the weighs and tuning

The first part is dedicated to train only the last fully connected component of the VGG16, while the remaining weights are freezed.
VGG16 has been written for resolving a classification problem between 1000 different classes, we have to change the last classification layer with a layer conform to our problem. The imput images have shape 224x224 so we adequate our image to the format the model requires, moreover this cnn accepts only 3-channel images, so we make the black and white images (1 channel) into colored ones (3 channel) by a function provided by openCV:

```
... test_x.append(cv2.cvtColor(cv2.imread(im), cv2.COLOR_BGR2RGB))
... train_x.append(cv2.cvtColor(cv2.imread(im), cv2.COLOR_BGR2RGB))
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten_2 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_2 (Dense)              (None, 15)                376335
=================================================================
Total params: 15,091,023
Trainable params: 376,335
Non-trainable params: 14,714,688
```

We train the weights of the last fully connected layer for 20 epochs, figure 11: We

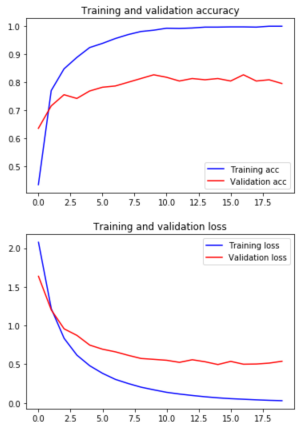

Figure 11: Accuracy and loss of the modified vgg16.

use a batch-size equal to 32. As Optimizer we use optimizers.RMSprop(lr=1e-4) that divide the learning rate for a weight by a running average of the magnitudes of recent gradients for that weights. As loss function we decide to use categorical crossentropy.
Only the last weights are trainable, but despite it is an expensive computation as in fact there are a lot of weights, but we obtain very good results. For the test set we get the following results:

```
loss: 0.48
accuracy: 0.83
```

We plot the confusion matrix in the figure 12

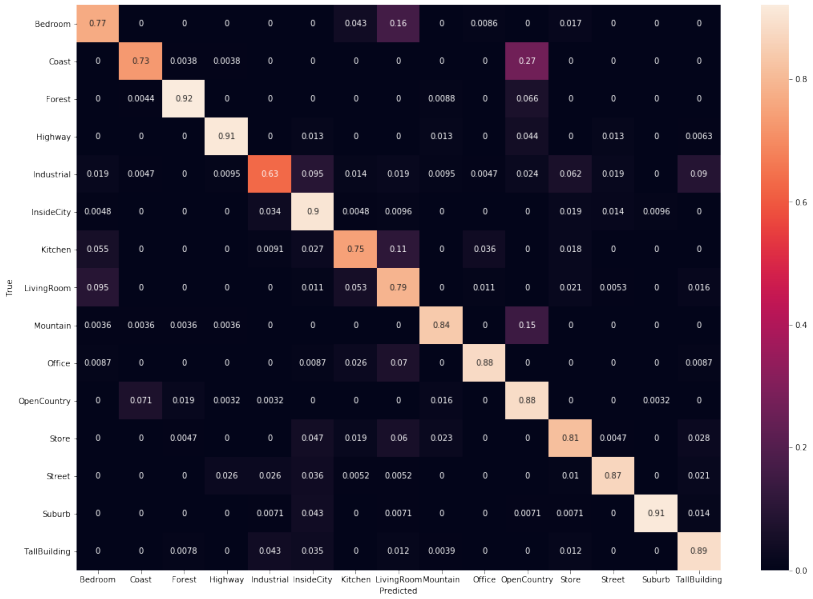

Figure 12: Confusion matrix related to the results obtained using VGG16 over the numbers of epochs.

4.2 Features extractor

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 perform a linear SVM, using the function Linear SVC (Support Vector Classifier), a function that the Python library sklearn provide to fit to the data, and returns a "best fit" hyperplane that divides, or categorizes, data. In particular we extract features obtained in the last convolutional layer of the VGG16. For each image, we will obtain 512 features, each of them is a 7x7 grid of values, so for each image we associate a feature represented by a 25088-dimension vector.

After that we simple use the features in a SVM. Being a multiclasses classification problem, it is necessary to decide what strategy one has to use for classification; I personally adopted a one-against-rest approach which is implemented by the method fit of the library that we use.
We obtain very good results, on the test set an accurancy of:

```
accuracy: 0.84
```

We can see the accuracy matrix in the figure 13

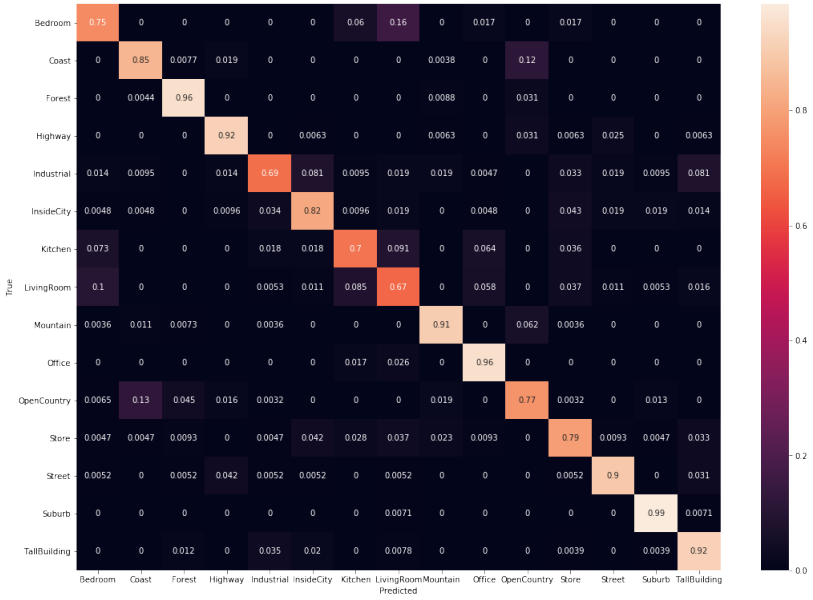

Figure 13: Confusion matrix related to the results obtained using linear SVM.

5 Conclusion

In this project, we will exploit all the most important feature of cnn for a multiclass classification problem and we have observed that techniques based on cnn are very powerful. A very shallow convolutional neural network was enough to obtain a quite good results and it has been possible improve these results with simple tricks like batch normalization or data augmentation. In the end, we applied transfer learning, in particular we extract features obtained by the last convolutional layer of VGG16 and we exploited them in a SVM, obtaining a very high test accuracy.