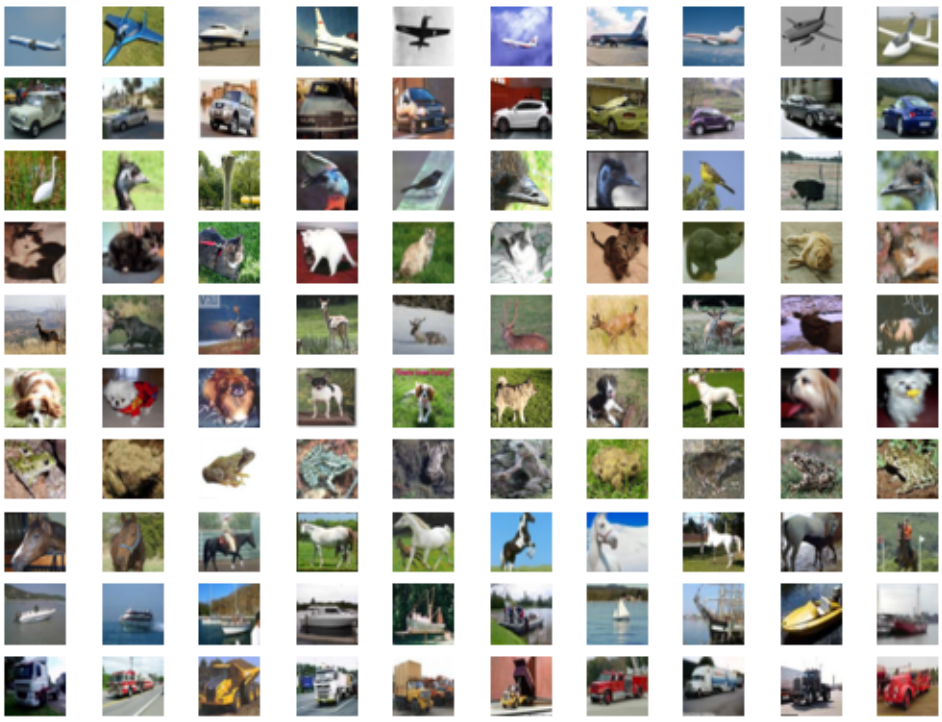
Tooploox Data Scientist Exercise

1. Download and process the data from the CIFAR-10 website
2. Plot 10 random images from each class in the dataset.

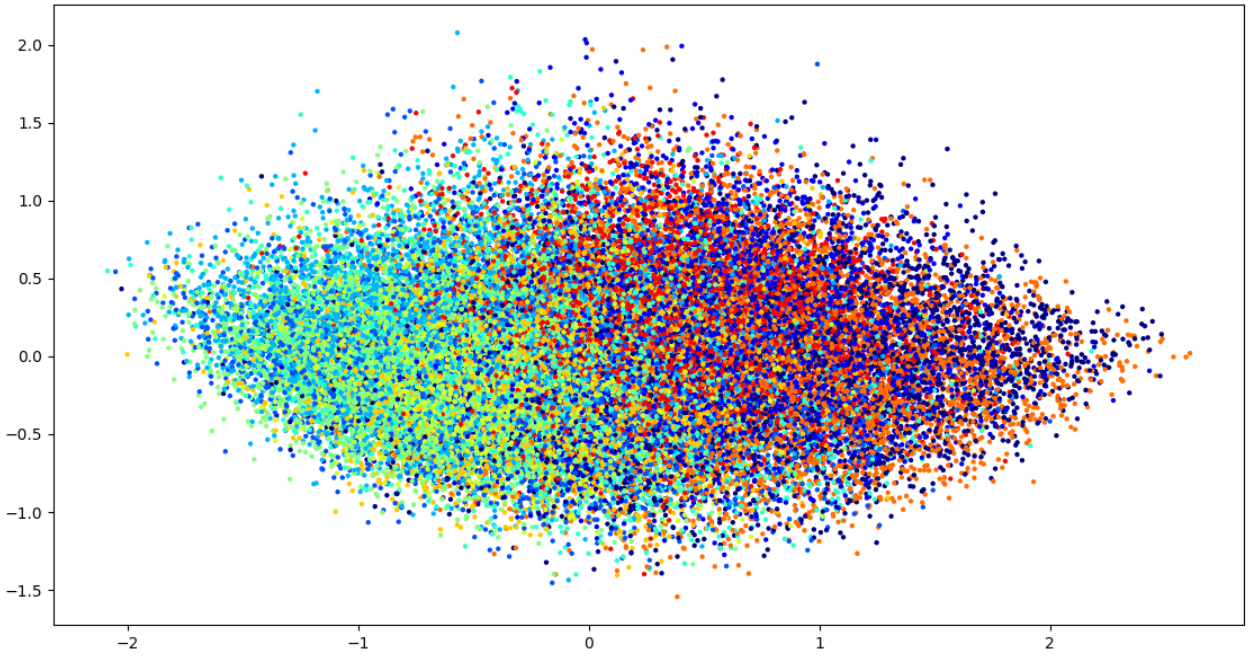
The resulting image:



3. Train a shallow classifier to serve as a benchmark.

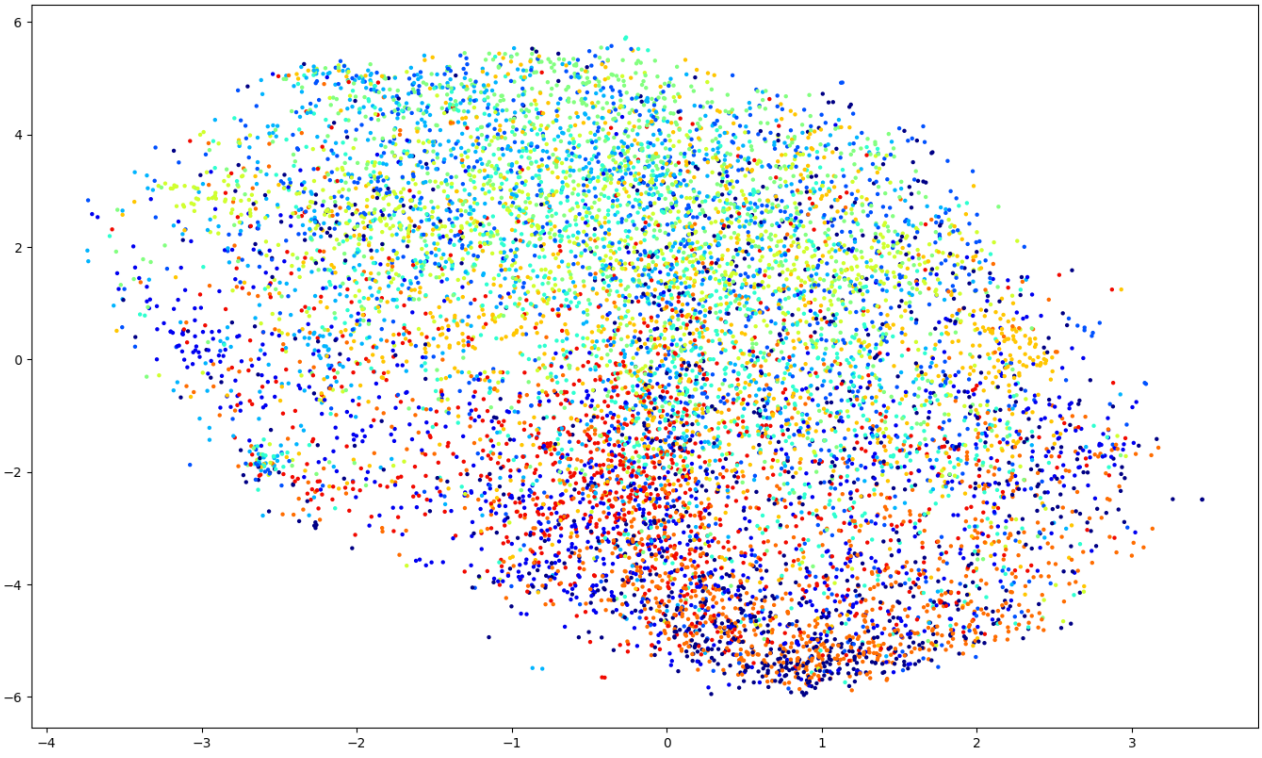
I extracted the HOG descriptors on the whole training dataset, using the ***skimage*** library. Every sample was represented as a 128-dimensional vector.

To check if the data is linear-separable, at the beginning I used the Principal Component Analysis.

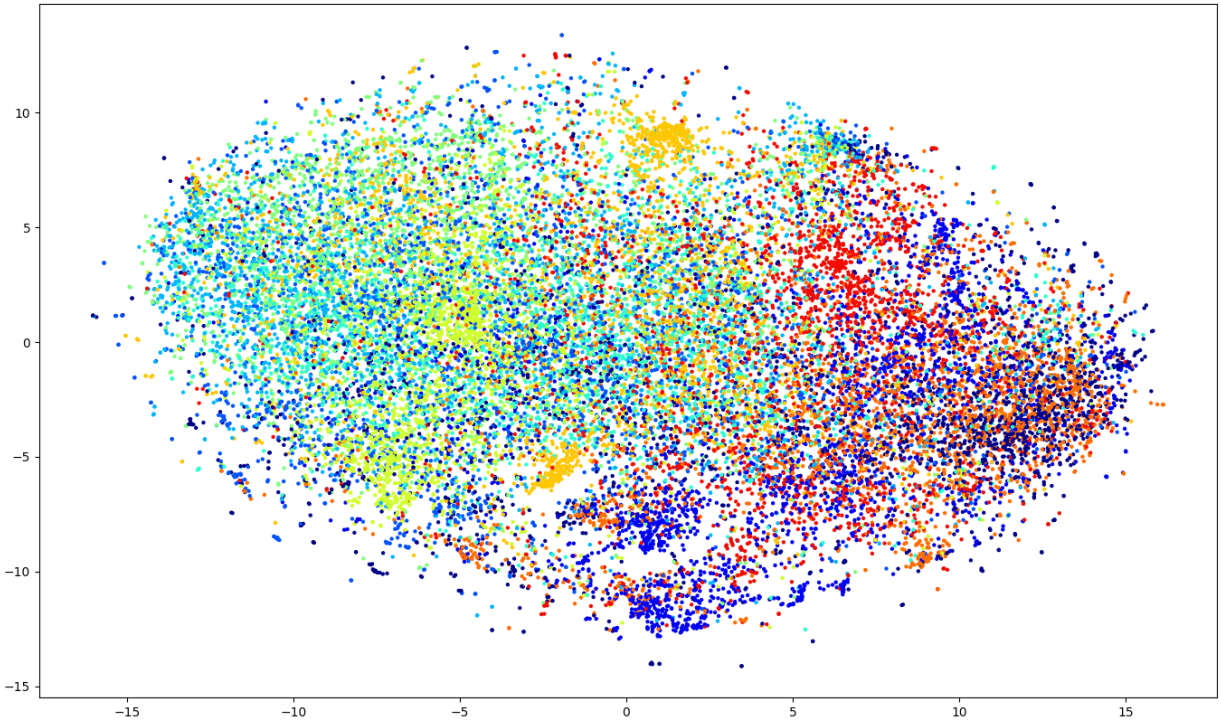


The percentage of the variance explained by each of the two main components is respectively: 9.71% and 3.62%. As we can see, it is almost completely impossible to decide if the data is linear separable, based on the 2D plot (not very much of the variance is focused in the main 2 components). I then tried to visualize the data with t-SNE method.

For the first 10.000 samples in the training set:



For the first 25.000 samples in the training set:



Despite the fact that the non-linear dimensionality reduction technique like t-SNE is sometimes quite hard to interpret, we can see that different classes have its data almost focused. However, it could be impossible to distinguish between classes with a linear classifier so I used the RBF function kernel for SVM classification.

As a fast benchmark, for RBF kernel, trained with the one-vs-rest strategy I got the **resulting accuracy** of **0.4858**. The one-vs-one (much slower) strategy gave the same exact result. This is almost 5 times better than a random classifier for 10 classes. However, it is much less than we would like to achieve :)

4. Extract visual features using a pre-trained CNN network.

For a start I tried to use Keras as a machine learning framework, as it provides an easy high-level API for deep – learning, based on other frameworks like Tensorflow. To extract the CNN codes I decided to use pre-trained Inception-v3 network.

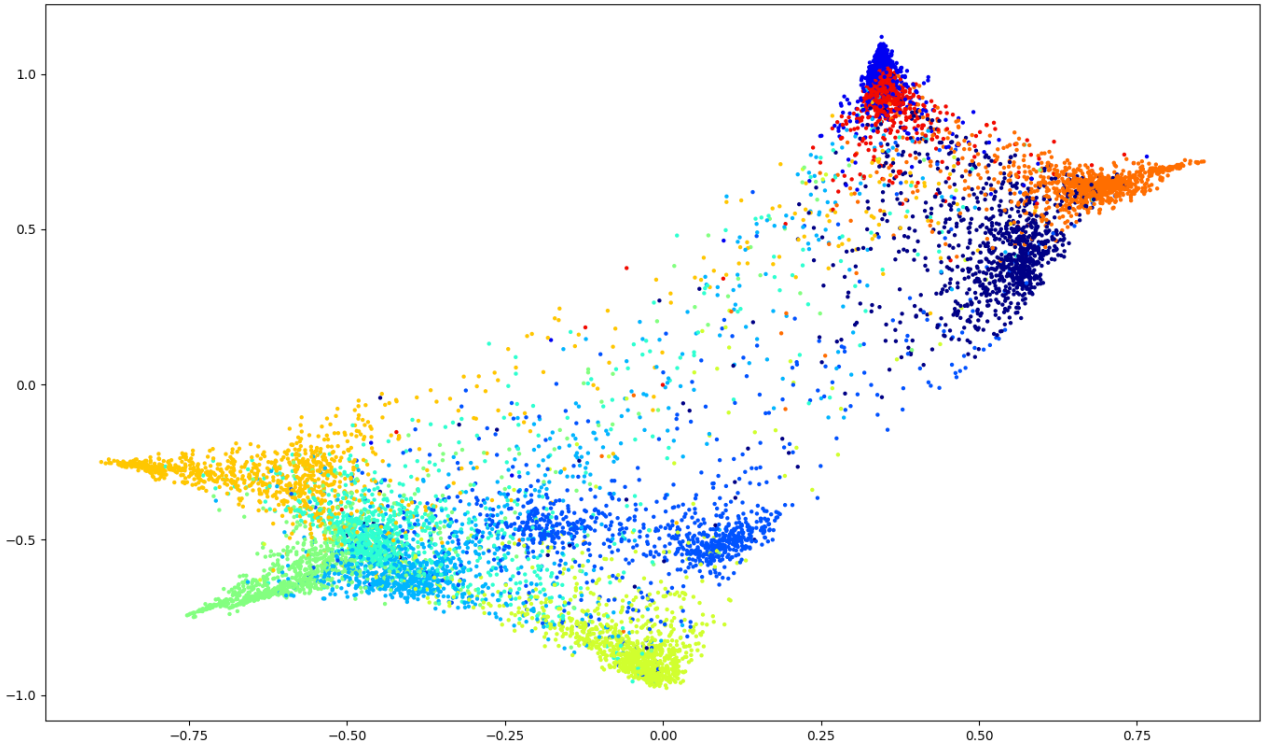
```
import keras
from keras.datasets import cifar10
import numpy as np
from keras.applications.inception_v3 import InceptionV3, preprocess_input
import scipy
from scipy import misc
import os

(x_train, y_train), (x_test, y_test) = cifar10.load_data()
y_train = np.squeeze(y_train)
model = InceptionV3(weights='imagenet', include_top=False, input_shape=(139, 139, 3))

inception_features_filename = 'inception_features_test.npz'
if os.path.exists(inception_features_filename):
    print('Bottleneck features detected')
    features_test = np.load('inception_features_test.npz')['features_test']
else:
    print('No bottleneck features saved')
    # pre-process the test data
    resized_images = np.array([scipy.misc.imresize(x_test[i], (139, 139, 3))
                    for i in range(0, len(x_test))]).astype('float32')
    inception_input_test = preprocess_input(resized_images)

    features_test = model.predict(inception_input_test)
    features_test = np.squeeze(features_test)
    np.savez(inception_features_filename, features_test=features_test)
print('Bottleneck features saved.')
```

Unfortunately, after some experiments I found out that there is no possibility to extract the penultimate layer just before the soft-max layer for Inception-v3 in Keras. To find it out, I printed a list of all layers' names of the network. The last layer before the final predictions layer was 'avg_pool'. It outputs a (3, 3, 2048)-dimensional vector of features (18 432 features). I experimented with the “include_top” parameter but it looks like Keras does not show the “pool_3:0” layer in any case.
Therefore, I needed to use the whole Tensorflow API.

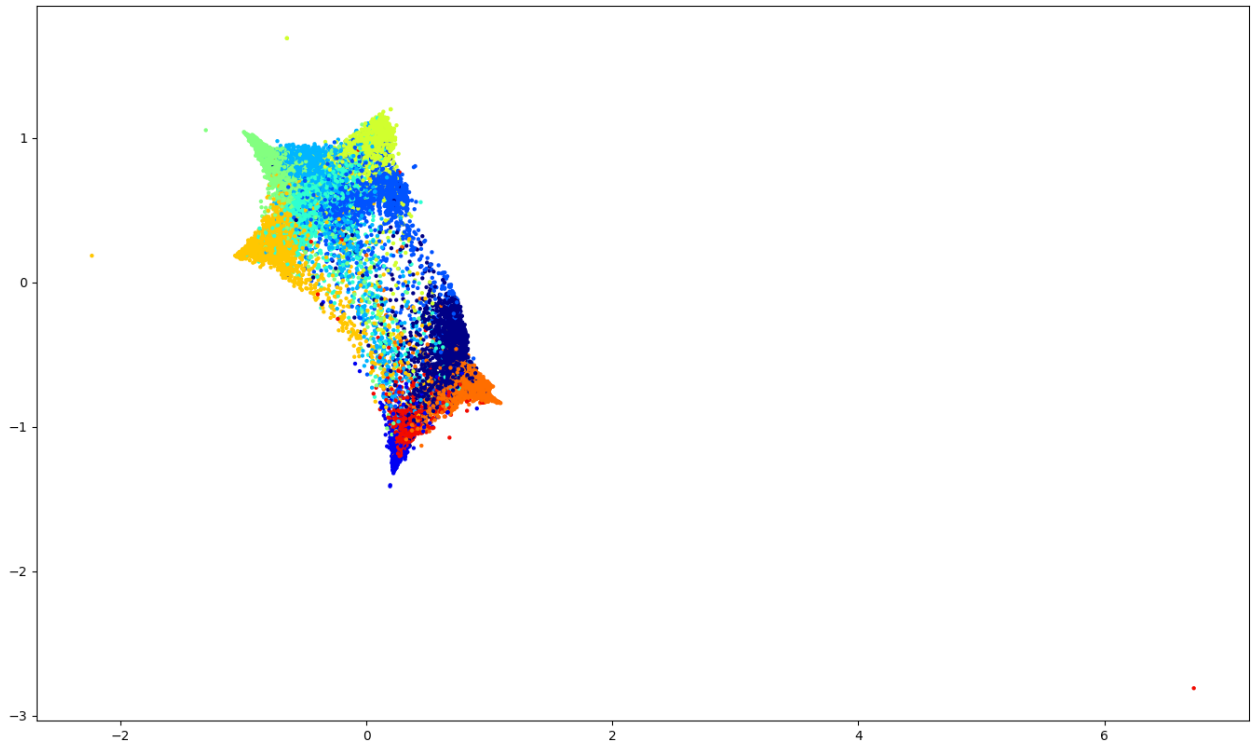
As a result, I got a 2048-dimensional vector for each sample of the whole dataset.

5. Visualize the CNN codes by embedding them in two dimensions.

For a benchmark classification, I used t-SNE as a dimensionality reduction technique. Below you can find the resulting scatter plot for the first 10000 samples from the training set:



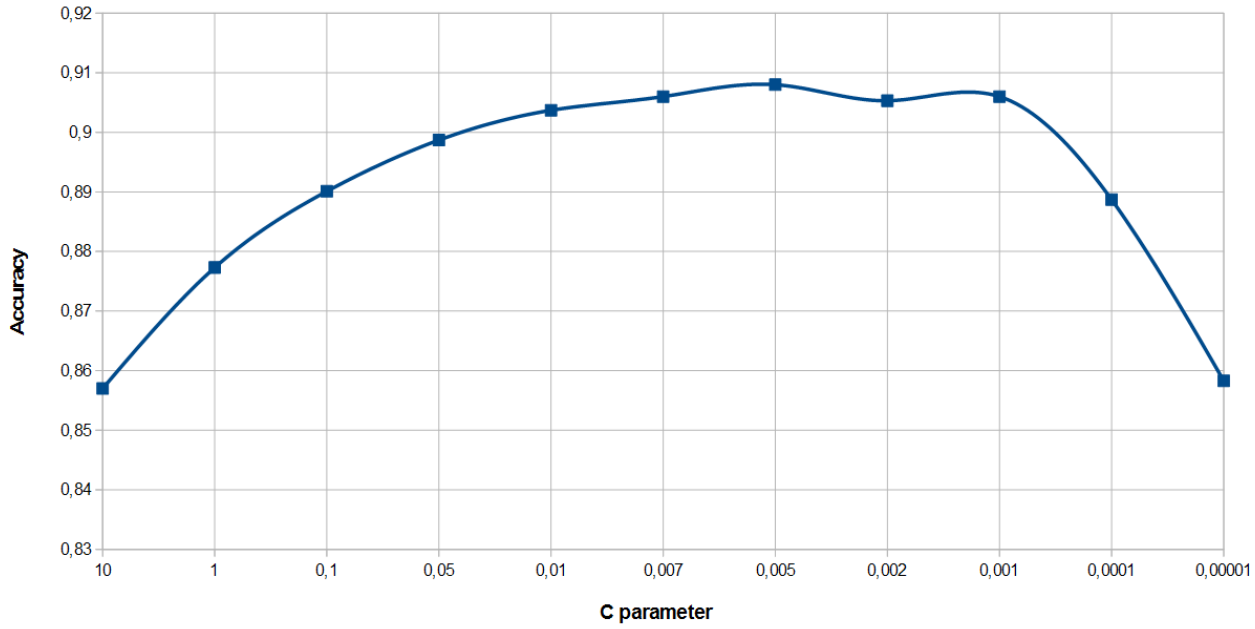
And another one for the first 25000 samples (the first half of the training dataset):



As we can see, the pre-trained Convolutional Neural Network does a real magic with the data. The classes are now quite easily separable and possibly the linear SVM kernel could give us some promising results. Of course, we have to remember that t-SNE is quite hard to interpret, so we need to apply a cross-validation to choose the best SVM model and its hyper-parameters.

6. Train a SVM classifier on top of the CNN codes.

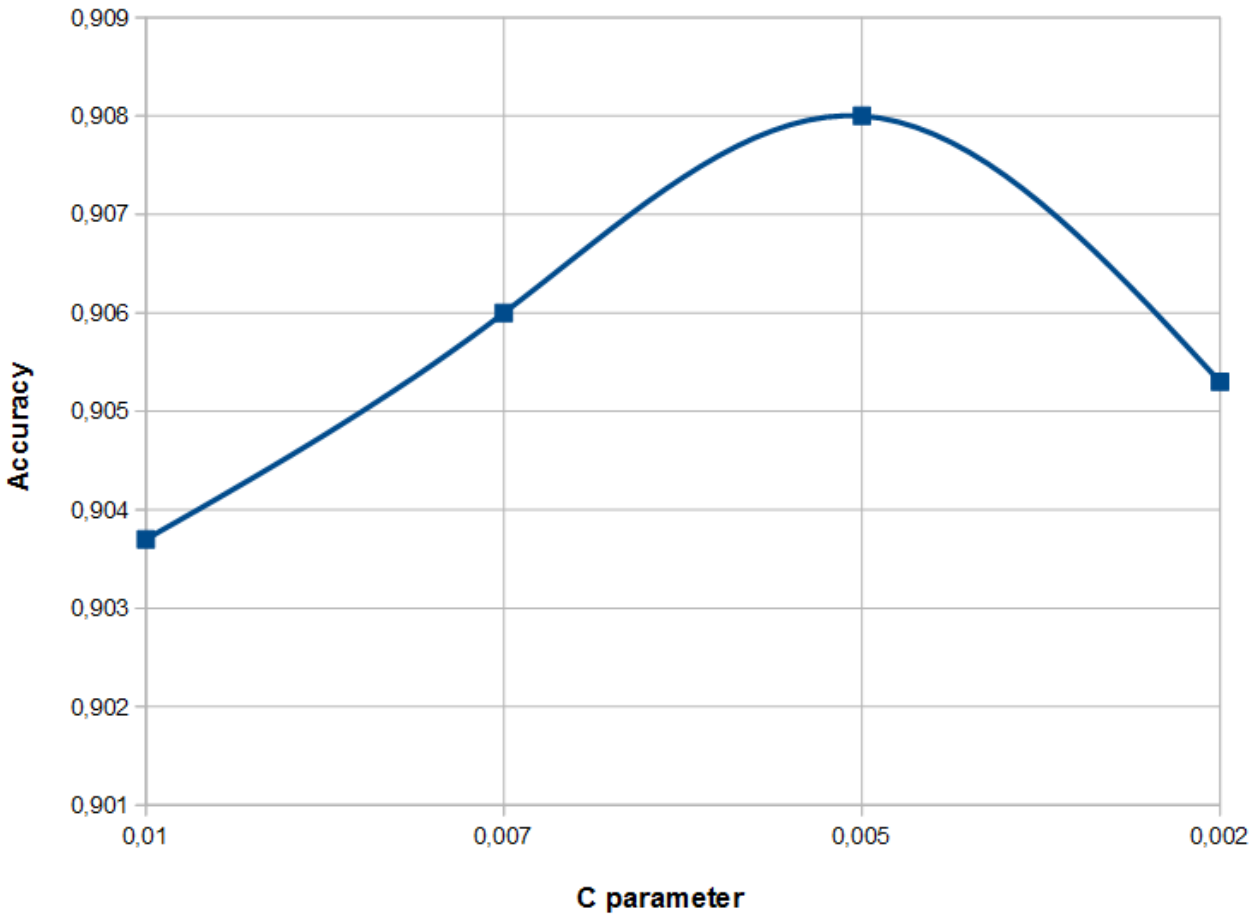
For a start, I chose a linear SVM classifier, based on a Python LinearSVC class (in the ***sklearn.svm*** library). At first, I wanted to roughly estimate the order of magnitude of the ***C*** parameter (the penalty parameter) of the Support Vector Machine. In order to do that I randomly divided the training dataset into two sets (the training set: 80% and the validation set: 20%). I took a default value of the ***C*** parameter, computed the model on the training set and evaluated it on the validation set. Then I took another value of ***C*** and repeated the process. It helped me to create this plot for a start:



This rough estimation of the ***C*** parameter was used to make all the calculations faster (cross-validation is much slower because it trains a model a couple of times). I saved the validation data scores, confusion matrices and values of precision, recall and F1 score in the file: ***single_step_validation.txt***. Now when I knew that I should look for the optimal ***C*** parameter close to the value of 0.005 (which gave the accuracy of 0.908 on the validation set), I could apply a real cross-validation.

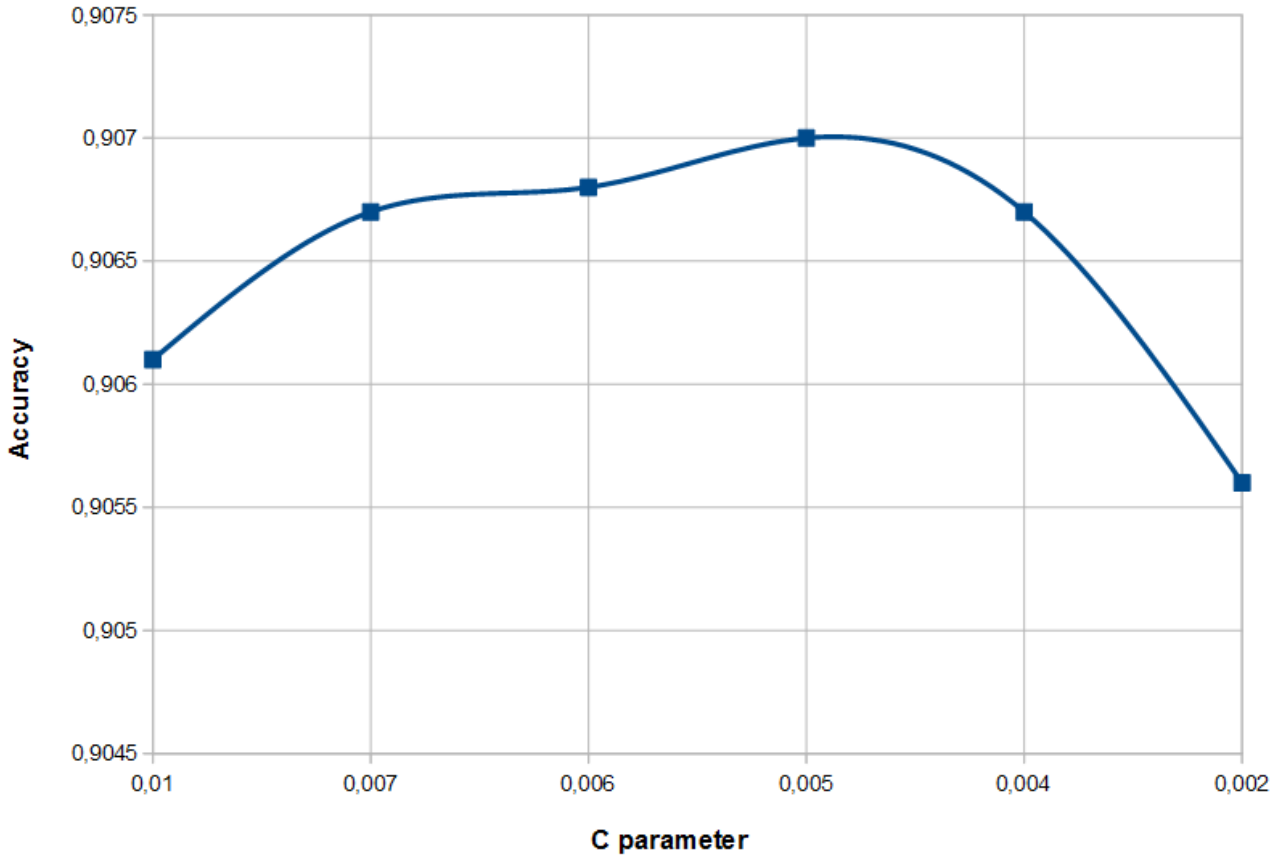
Unfortunately, the laptop I used in this task wasn't definitely the fastest machine in the world, so I started with a cross-validation with 4 iterations. I realize it could be too little but for a start I had to ensure I would get proper results around the C=0.005 point. Moreover, I used a computer with exactly 4 CPU's and a cross-validation algorithm that could be parallelized on a separate processors.

Below you can find the validation data scores I got:



C = 0.01, Mean accuracy: 0.9053 +/- 0.0070
C = 0.007, Mean accuracy: 0.9057 +/- 0.0071
C = 0.005, Mean accuracy: 0.9059 +/- 0.0067
C = 0.002, Mean accuracy: 0.9050 +/- 0.0063
(file ***cv4_results.txt***)
The possible error (e.g. +/- 0.0070) was calculated as 2 times standard deviation of the accuracies for each iteration.

Now I had a possibility to check the ***C*** hyper-parameter with the cross-validation of 8 iterations. Below you can find the results:



C = 0.01, Mean accuracy: 0.9061 +/- 0.0080
C = 0.007, Mean accuracy: 0.9067 +/- 0.0078
C = 0.006, Mean accuracy: 0.9068 +/- 0.0081
C = 0.005, Mean accuracy: 0.9070 +/- 0.0077
C = 0.004, Mean accuracy: 0.9067 +/- 0.0085
C = 0.002, Mean accuracy: 0.9056 +/- 0.0073
(file ***cv8_results.txt***)

Again, C=0.005 gave the best result. The distributions of the each iteration's scores were quite compact (the standard deviations were quite small). Thanks to that, we can suppose that the iterations were reliable.

This wasn't enough for me and I wanted to check the results of the cross-validation for a measure other than the classifier's accuracy. In order to find the balance between the precision and recall I chose the ***F1*** measure and reapplied the 8-steps cross-validation. Here are the results:

C = 0.01, F1 score: 0.9061 +/- 0.0079
C = 0.005, F1 score: 0.9069 +/- 0.0076
C = 0.002, F1 score: 0.9055 +/- 0.0072
(file ***f1_cv8_results.txt***)
Now I was totally convinced that C=0.005 is the optimal hyper-parameter. In fact, I used the LinearSVC Python classifier (it has a lower number of hyper-parameters because it is based on the ***liblinear*** library), so I decided to leave other hyper-parameters default because they generally have less impact on the final score than the penalty parameter.

I also wanted to check the RBF kernel of the SVM classifier. In order to do that I applied the 8-steps cross-validation for the initial value of the ***C*** parameter equal to 1.
The computations took many hours but finally I got the result:
C = 1, Mean accuracy: 0.8940 +/- 0.0065
(file ***rbf_cv8_c1.txt***)

Then I wanted to try the best parameter of the linear model, C=0.005, for the RBF kernel. I'm pretty sure the computations took almost one century and I finally couldn't evaluate them. So I decided that I won't further check the RBF model (neither the polynomial nor the sigmoid one) and take the linear model as the final one.

7. Evaluate your model on the test set.

I finally trained the SVM classifier on the whole training dataset (50 000 samples). The **test set accuracy** was **0.9055** which is a very great result including fact that it was the best result for the CIFAR-10 dataset in 2012. The transfer learning approach did here a really good job including fact, that I didn't have to train the parameters of the whole, big CNN.

8. Further improvement.

If I would have much time for this task, I would definitely try data augmentation to decrease a generalization error. However, I wouldn't try to scale images because they are already very, very small (32x32) and the downsampling could create only a noise. Moreover, I probably wouldn't try to add any noise to the images – for the same reason. What I would try is to simply flip images horizontally. Maybe I would also change some lighting conditions e.g by histogram equalization. Possibly this could improve the general accuracy a little bit.
However, it is also good to stop for a while and spend some time with the family :)