Image Classification: Boat Classifier using CNN

Riccardo, Chiaretti, 1661390

23/11/2017

1 ARGOS Project Overview

The ARGOS Project, was a project started in order to control the traffic of boats within the city of Venice. Indeed, since Venice is a city built over the water, waves produced by the passage of boats inside canals are probably the main cause of the erosion of the base where the buildings are built.
In order to solve this problem, 13 survey cells are installed along the Grand Canal and what each survey cell does, is the following:

- Background Estimation and Subtraction in order to be able to eliminate the water in the background and to highlight the boats;
- Segmentation by exploiting the foreground image and the optical flow image they gather information about the boat in the image;
- then all these information is given to the tracking module to keep track of the boat while it is moving.

2 Boat Classification

The problem we are going to discuss in the following, is almost one of the problem that the designers of ARGOS encounter during the development of the software, that is the boat classification. Indeed, to be able to know if a boat should be sanctioned or not, at first we should be able to distinguish if the boat under consideration is, for example, a Cacciapesca rather than a Mototopo or a Gondola.
Therefore, as the general classification problem in machine learning, we want to find a function such that, given the set of possible boats, we are able to recognize which kind of boat it is, which means, classify it correctly.

2.1 MarDCT Boat Classification dataset [2]

The dataset we are going to use in this classification problem is called MarDCT dataset. In particular, there are two separated datasets: one is used for the training part of our problem (training set) while the other is used as the test set for evaluating "how good" is our classifier.
The dataset which we consider as training set is the one contained in the *sc5* folder: it contains 24 different sub-directories each one corresponding to a specific boat and containing all pictures showing that boat.
The other set of images, contained in the *sc52013MarAprTest20130412* directory, is our test set which is structured differently from the previous one. Indeed, in this folder, there is a set of 1900 images depicting all 24 classes of boats with a text file containing for each image the corresponding class. Clearly this dataset cannot be given directly to our classifier, thus we should organize it in such a way to be readable for our algorithm.

3 Convolutional Neural Networks (CNN)

The model we are going to use to build our image classifier, is the **Artificial Neural Network** model. Basically, a neural network, is a set of *units* (called *neurons*) organized in one or more groups (called *layers* of the network) such that each layer has a specific task even if all the neurons concur to solve the same problem. In particular, the neural network we are going to build, is the **Convolutional Neural Network (CNN)** which is a particular *feedforward* artificial neural network (meaning that all information goes through the network starting from the input nodes to the output ones without having cycles among units) widely used for analyzing visual imagery. A CNN is composed of one input layer as well as one output layer and multiple hidden layers which typically consist of convolutional layers, pooling layers and fully connected layers.
The goal of a CNN is to learn a set of parameters called *weights* (or also *filters* or *kernels*) and *biases* such that, at the end of the training phase, applying those filters to a new image, the network should be able to correctly classify it. The main advantage of this approach with the respect to other image classification algorithms is that, while in those algorithms filters should be "explicitly handled", in CNNs filters are self-learned during the training phase regardless of the human hand or a priori knowledge.

4 TensorFlow

The software for the image classifier is provided in Python programming language since it enables to import the TensorFlow library.
TensorFlow[1] is an open source library and a software tool for deep learning which uses data flow graphs to build models. It was developed by Google and actually used in lots of fields such as classification, perception and prediction indeed, some of Google products which use TensorFlow include: Google search, Google's voice recognition app, and Google Translate and Inception. In the following we are going to show the neural network used for the boat classifier.

4.1 CNN implementation using TensorFlow

Briefly, the network we built using TensorFlow, is composed of two convolutional layers, each one followed by a max-pooling layer, and two fully connected layers (also called dense layers). Below, we are going to see in a more detailed way all the passes we followed to build the classifier, starting from the construction of the dataset and describing all the layers that make our Convolutional Neural Network.

4.1.1 Building the dataset

Since that the MarDCT dataset, as it is, cannot be given in input to any network and since that in TensorFlow everything should be a tensor, we have to organize both the training set and the test set in a way that it is acceptable by the TensorFlow framework.
At first, in order to identify the membership class of each image, we should fetch all names of all the classes from the directories contained in the *sc5* folder.
In order to build a correct dataset, we should fill two different arrays: one containing the path of all images both for the training set and for the test set while the other should be the list of the membership class of images contained in the previous array. To do this, we built a dictionary data structure where the key is the name of the membership class, while the value is an integer from 0 to 24. In this way, since we have to build a dataset where labels are represented in one-hot encoding, instead of assigning to each image the entire name of its membership class, we assign to it the integer value stored in the dictionary. Then, calling the `one_hot()` function, we convert the integer into a one-hot tensor of dimension 24 (i.e. the label 3 will be transformed

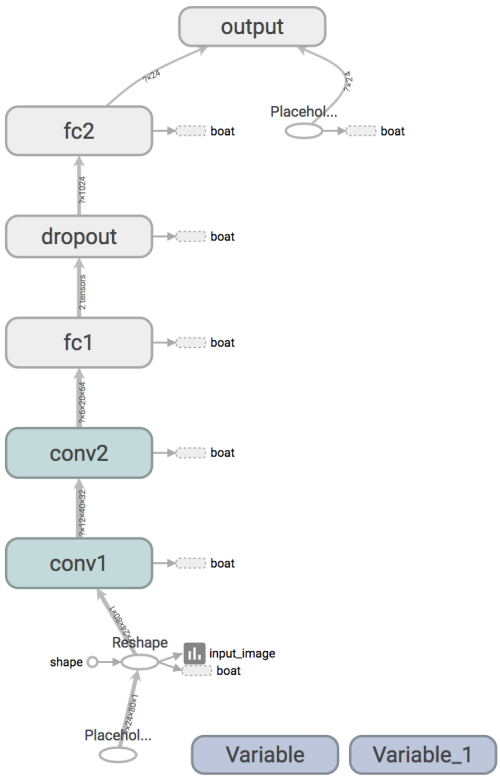

Figure 1: This figure shows the layers of the CNN we used to build our classifier.

into `[0, 0, 0, 1, 0, ..., 0]`).
For what concern the training set, the approach is almost the same apart from the parsing of the ground truth. Indeed, unlike the training set, now we should read and parse the file containing the membership class for each image in the test set.
The task is not difficult in fact, since that some membership class have not the same name of the ones in the training set or they do not appear at all, the only thing we should to do is to change those names in the correct ones while skip images whose membership class is not in the training samples.

4.1.2 Convolutional layers

In the first **convolutional layer** (which is created by calling the `conv2d()` function) we are applying 32 filters (or kernels) each one of size 5×5, in order to get exactly 32 features from the input layer. Since everything in TensorFlow is a tensor, here the input layer is a tensor whose shape is `[-1, 24, 80, 1]` where: -1 is a special value meaning that the batch size of images should be computed dynamically, 24×80 is the (re-sized) input image while the last value is the number of color channels in the extracted image (if equal to 1 means greyscale). The `strides=[1,1,1,1]` parameter defines the "policy" with which the kernel should move while applied to the image; in this case we move by one unit on all four dimensions. To apply the kernel to all the image, we set `padding='SAME'` meaning that the output should be of the same size of the input. Indeed, in this way we are filling we are adding 0s to the edges of he input tensor to preserve the 24×80 tensor. At the end of each convolutional layer we use the `max_pool()` function that applies the pooling to the tensor in input taking the maximum pixel of a 2×2 grid of pixels. This means that given the input image whose size is $width \times height$, the output will be of size $\frac{width}{2} \times \frac{height}{2}$; the intuition under this layer is to combine the output of groups of neurons in one layer to a single neuron in the following layer.
The second convolutional layer has exactly the same structure of the first one indeed, the only difference is the size of the input and the output. Now the input is the pooled image represented with its 32 features, while the output is another pooled image with half size, each one represented with 64 features.
The activation function used at the end of each convolutional layer, before the max pooling, is the ReLU.

4.1.3 Fully connected layers

At the end of the the second convolutional layer we have that each image has size a quarter of the original (thanks to having applied two pooling layers) and represented by 64 features. This output tensor is reshaped such that to have only two dimensions: $(batch\ size) \times (features\ size)$ and then it is passed through two layers called **fully connected layers**. A fully connected layer has the peculiarity that each neuron belonging to that layer is connected to all activation in the previous layer.
In our case, the first of that layers is composed of 1024 units and the activation function is always the ReLU. Notice that before passing to the last

layer (where the classification is done) we apply what is called a dropout to the first dense layer: this is a regularization technique which consists in eliminating some units (in this case with probability 50%) of the layer in order to prevent the network to overfit during the training phase.
The last dense layer is the one in which the classification is done indeed, this layer, is composed of 24 units which is exactly the number of classes of our classification problem.
The classification is done by the `softmax_cross_entropy_with_logits()` function that given the output tensor of the last fully connected layer (containing the predictions of our network), it produces another tensor of the same dimension where each entry is a value between 0 and 1 such that all these values sum up to 1. This peculiarity is due to the *softmax* function which is widely used in multiclass class classification. Indeed, in this case, given the 24 values produced by this function, we can interpret the index (from 0 to 23) containing the highest value, as the predicted class of our model. Now we should define a method for estimating how good is our model in classifying boats thus, in the following, we are going to define what is our loss function.

4.1.4 Loss function and optimizer

The loss function used in our network is the *cross entropy* which is applied to the output of the softmax function. Basically, the cross entropy loss, determines how far is the predicted distribution from the true distribution. Indeed, given the probability distribution predicted by applying the softmax and the real distribution (given in one-hot representation), using the cross entropy formula we can understand how "wrong" is our prediction with the respect to the real distribution.
Then, given a measure of how far we are from the right classification, we can use an optimizer in order to minimize the loss function. In this case we choose the Adam optimizer: we can say that it is similar to a Stochastic Gradient Descent (SGD) algorithm apart from the way in which the gradient is updated in each iteration. In particular, while the SGD uses only a single learning rate which is used to update the gradient, in Adam we have a different learning rate for each parameter of the network which is also adapted during the learning phase.

4.1.5 Accuracy evaluation

In this section, we are going to discuss and to compare some executions of our boat classifier when some parameters are changed or even when the network itself is modified.
Before starting with the discussion, in Figure 2 you can see the plot of the loss function; clearly the right moment to stop our algorithm is when the loss function reaches 0 meaning that there should not be anything more to learn but, since that the reached value is almost 2 (after 1000 iterations) which is smaller enough, we can consider ourselves satisfied. All our runs[1]

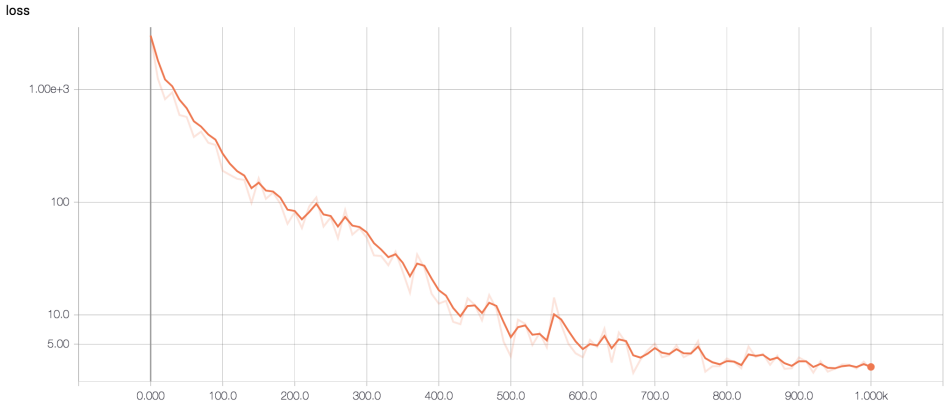

Figure 2: In this figure you can see the plot of the loss function.

are based on images whose size is 24×80 instead of its original size which is 240×800 so, the features extracted by our network are not comparable to the ones extracted from an image in its original size. Therefore, the maximum accuracy we can obtain, is less than the one we could obtain considering the pictures with their original size even because, almost all our runs, consider greyscale images.
In Figure 3 you can find a plot of the accuracy reached by the CNN. In the following we are also going to point out the accuracy when evaluating samples from the training set, just to highlight the fact that our network doesn't overfit the training set itself. At first, the result obtained considering the whole 24 classes and only one channel (images in greyscale), is an accuracy of 0.55% on the training set while 0.45% on the test set.

[1]Notice that all the tests have been done on a machine with 8 GB of RAM with2 GHz Intel Core i5 as CPU and without a dedicated GPU

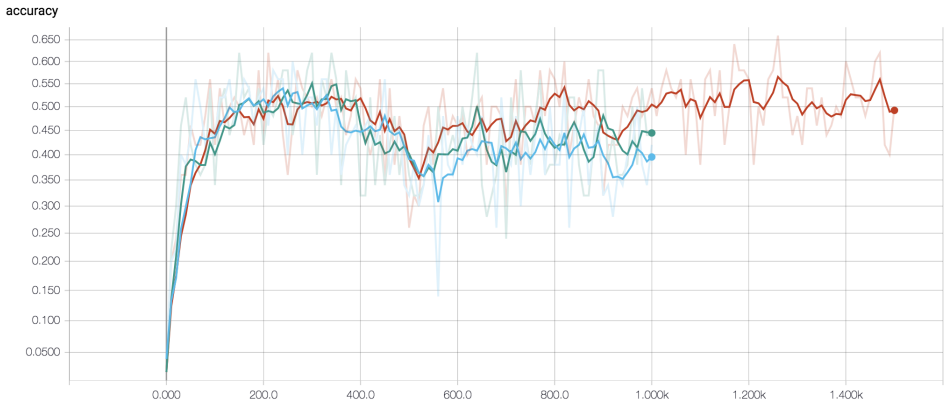

Figure 3: In this figure you can see the plot of the accuracy obtained by some configurations. The red one is the plot of the accuracy considering 3 convolutional layers and 1500 iterations, the green one is the result obtained by the network in Figure 1 and the last one (light blue colored) is the accuracy obtained considering 3 channels.

We can say that a very strange result is the one obtained by considering three channels (thus an image in RGB encoding) instead of one. Indeed, since that the CNN has more detailed features that can be extracted from each image, the accuracy should be greater that the case in which we had only greyscale images. However, running the same network and considering three channels, the observed accuracy is smaller than the one obtained in the previous case, that is 0.53% on the training set and 0.40% on the test set.
Another interesting result is the one obtained by classifying all boats apart from "Water" samples (using one channel): in this case, since that the 19% of the training set is composed of images belonging to that class, removing them bring to a worse accuracy than the other case. This means that the classifier was "very good" in classifying water samples than other images thus, removing them bring to have an accuracy of 0.46% on the training set while 0.38% on the test set.
Up to now we have discussed all results considering the network composed of two convolutional layers, two max pooling layers and two dense layers; now we are going to consider some runs made with different combination of layers and always one channel.
At first we add one more convolutional layer plus max pooling before the

dense layers. In this case the expected result should be better than the other cases indeed, adding those layers, the extracted features are more than the case in which we had only two convolutional layers (previously we had 64 features that were used by the fully connected layers to classify images, now we have 128 extracted features), even if the image is smaller. However, after 1000 iterations, we saw that the loss function doesn't reach the right value (we said that in our case the value 2 is good) thus the obtained accuracy is not better than the first case. Therefore, we try with 1500 iterations in order to have the loss function as smaller as possible, having the best accuracy we ever obtain that is, 0.55% on the training set and 0.49% on the test set.
Another configuration we tried was also the one with three convolutional layers but, this time, we put only one max pooling layer in order to have a bigger image when reaching the fully connected layers (indeed, in the previous case, we arrived to have an image of size 3×10). However, the processing time of images was too long in order to be considered a good classifier (given the resources of this machine) even because, when the image reaches the fully connected layers, since that there are 1024 neurons (without considering the dropping), they should work on too many parameters while, in previous cases, images were small enough to make a good classification. Therefore, in this case and given these resources, it is not a good idea to build a network with these characteristics.

References

[1] TensorFlow web page: *www.tensorflow.org*

[2] Bloisi, Domenico D.; Iocchi, Luca; Pennisi, Andrea; Tombolini, Luigi, "ARGOS-Venice Boat Classification," in 12th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), 2015, pp.1-6, 2015. doi: 10.1109/AVSS.2015.7301727