Machine Learning

Home Work II

Venice boat classification

Ahmed Mohamed Galal Osman

Master of Engineering in Computer Science

Matricola: 1853277

Objective

The objective of the project is to build an image classifier with CNN using only a few training examples - just a few hundred or thousands of images from each class of different categories of boats navigating the City of Venice (Italy).

In other words, given an input image, with a computer automatically classify it into one from a set of categories, say “Gondola”, “raccolta rifutti” or “ambulanza”, etc.

Data Exploration

The dataset used here is the MarDCT from the Sapienza University of Rome.

The training dataset contains 4.774 images divided into 24 classes.

The test/validation dataset contains 1.969 images, and Ground truth text file which contains images divided into their respective classes.

Techniques

I will use Convolutional neural network as our classifier. To implement it, I will use Keras library for Python with TensorFlow as backend.

In order to get full potential out of our CNN, I will have to preprocess the images, then train a convolutional neural network on all the samples.

I’ll get to apply what I learned in the class and build a convolutional, max pooling, dropout, and fully connected layers.

At the end, I'll get to see our neural network's predictions on the sample images.

Data Preprocessing

1- As a requirement of Keras library is to have the data in a specific folder structure, and because test data provided by MarDCT is differently structured, I used Ground truth text file as a base, and I wrote a script which will read Ground truth

text and create subfolders with the names of the classes of the boats. Then it will take the images and reorganize putting each image in its respective class, where the directory's name is taken as the label of all the images presented in it. For example all the images inside 'cats' folder will be considered as cats by Keras.

Remark: the script will also put any image with the class snapshot in Ground truth text file and it will put it in the new category called Water.

2nd remark: The classes provided in the test folder are fewer than the classes in the training folder. The number of given classes is 20 and training classes is 24. This fact created a problem as Keras specification requires of us to have the same number of categories/classes between the test and training folders.

(When we use the flow_from_directory properties)

I solved this issue by creating empty folders automatically. Note that this script is modula,r meaning if Ground truth file has the same number of classes as the files provided in training, no empty folders will be created.

2- To prevent over-fitting: Overfitting is when you get a great training accuracy and very poor test accuracy due to overfitting of nodes from one layer to another. So before I fit our images to the neural network, I need to perform some image augmentation on them, which is basically synthesizing the training data. I am going to do this using keras.preprocessing library for applying the synthesizing part as well as to prepare the training set and the test set of images that are present in a properly structured directories.

Build the Convolutional neural networks

1- I've Imported Sequential from keras.models, to initialize our neural network model as a sequential network.
2- I've imported Conv2D from keras.layers, this is to perform the convolution operation i.e the first step of a CNN, on the training images. Since I am working on images here, which are basically 2-Dimensional arrays, I am using Convolution 2-D. I may have to use Convolution 3-D while dealing with videos, where the third dimension will be time.

3- I've imported MaxPooling2D from keras.layers, which is used for pooling operation. That is the second step in the process of building a CNN. For building this particular neural network, I are using a Maxpooling function. There are different types of pooling operations like Min Pooling, Mean Pooling, etc. Here in MaxPooling I need the maximum value pixel from the respective region of interest.

4- I've imported Flatten from keras.layers, which is used for flattening. Flattening is the process of converting all the resultant 2-dimensional arrays into a single long continuous linear vector.

5- I've imported Dense from keras.layers that is used to perform the full connection of the neural network, which is the fourth step in the process of building a CNN.

Now that we have a better sense of the dataset we're working with, let's move onto the machine learning bits.

I define the loss function to measure how poorly this model performs on images with known labels. I use a specific form called the categorical_crossentropy.

Using the magic of blackbox optimization algorithms provided by TensorFlow and Keras, I can define a single step of the *rmsprop* optimizer (to improve our parameters for our score function and reduce the loss) in one line of code.

The Training

- ### First approach

  Here I trained the model using 3 convolutional layers, I will apply max pooling for each layer, and I will add 3 fully connected layers. In the hidden layers the ReLu activation function will be used. In the output layer, the softmax activation function is used, because there are more than 2 classes

  ```
  Softmax = Softmax Layer
  ReLU = Rectified Linear Unit Activation
  ```

  After 20 epochs, the **accuracy** is almost **60%** and **the loss** is almost **9.0009.**

- ## Second approach

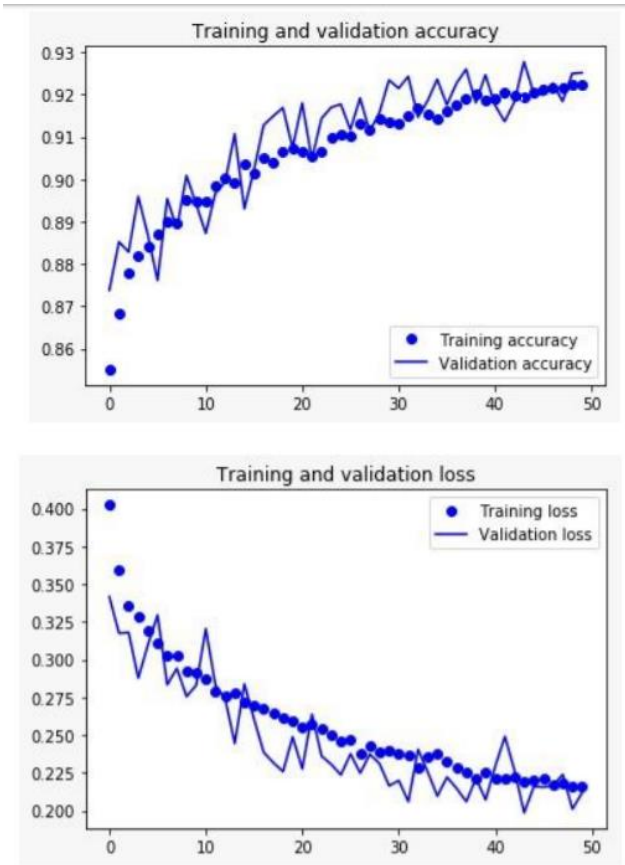
  In the second approach, I have changed the settings, such as adding a new convolutional layers, making the model use 4 convolutional layers.
  I Increased the epochs to 30, and the batch size to 128, after which we can notice a slight improvement in accuracy which turns out to be 65%.

  The **accuracy** is almost **65%** and **the loss** is almost **7.125.**

- ## Third approach

  In the third approach, in which I set back the CNN settings to 3 convolutional layers, batch size of 32 and 20 steps of epochs. More importantly I decreased the number of classes in both training and test datasets to be 10.
  Here we can notice a great accuracy improvement with accuracy being 92.63%, and loss of 0.1776999668269291

  For visual purpose, we plotted the training and validation accuracy and loss:



The Prediction

To make a prediction, i.e. to classify a new image that it is not present in the training set, follow the next steps

1- There is an empty folder called (predictions)
2- Put the image you want to classify inside it
3- Run the code

CONCLUSION & SUMMARY

We can notice the accuracy and loss vary in the three approaches, due to the number of images in the categories fluctuating. Keras and CNN work better if the training folder categories are equally distributed. That's not our case. There's a big difference. The Water category, for example has 9000 images, and Gondola has 10 images. This makes out training model a bias one.

Some remarks on pooling and dropouts

Pooling

The first secret sauce that has made CNNs very effective is pooling. Pooling is a vector to scalar transformation that operates on each local region of an image, just like convolutions do. However, unlike convolutions, they do not have filters and do not compute dot products with the local region. Instead, they compute the average of the pixels in the region (Average Pooling) or simply pick the pixel with the highest intensity and discards the rest (Max Pooling).

I used a 2 x 2 pooling, as it will effectively reduce the size of feature maps by a factor of 2.

Dropouts

Overfitting is a phenomenon whereby a network works well on the training set, but performs poorly on the test set. This is often due to excessive dependence on the presence of specific features in the training set. Dropout is a technique for combating over-fitting. It works by randomly setting some activations to 0, essentially killing them. By doing this, the network is forced to explore more ways of classifying the images instead of over-depending on some features. This was one of the key elements in the AlexNet.