Detect clean from dirty streets to ask for maintenance in Montevideo

Naomie Halioua
M2 Artificial Intelligence and Advanced Visual Computing
Ecole Polytechnique
naomie.halioua@polytechnique.edu

Abstract

During the first quarantine in 2020 neighbors were worried about the garbage in the Montevideo container. The main objective of this project is to implement an AI that understands if a street is dirty or clean and interpret the results in order to request the automatic maintenance of cleaning trucks in the dirtiest streets. A CNN algorithm is used to classify the dirty and clean streets, and the Grad-Cam algorithm is used in a second step to visualize the waste to better understand which pixels of that image have contributed to the final output of the model.

Key Words: Convolutional Neural Network; Computer Vision; GradCam ;Artificial Intelligence; Classification; Image Classification; Deep Learning; Supervised Learning;Class Activation Maps

1 The DataSet

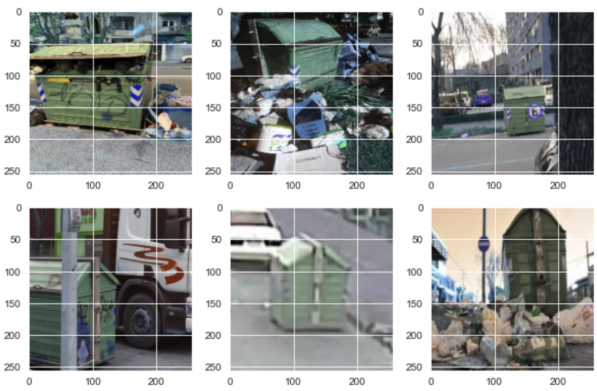

Figure 1: Samples of the Montevideo dataset

The data is divided into two parts, one for training and one for testing, and they are independent. The data is labeled as clean or dirty and it's shape is (3412,1). I down-sampled the images to a fixed resolution of 256 × 256 to trained the network on the (centered) raw RGB values of the pixels. The size of the images is very different from one image to another. They come from three main sources:

- Images from Google Street View, they are 600x600 pixels, collected automatically via its API.
- Images provided by individuals, most of which were taken by the author himself.
- Images from social networks (Twitter and Facebook) and news.

You can find the original Montevideo Container Dataset on Kaggle. [1].

2 Image Classification with CNN

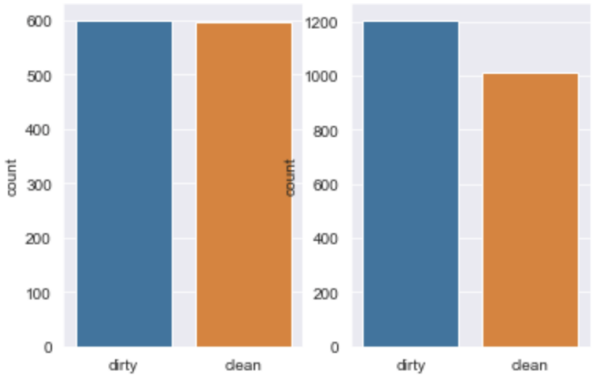

Figure 2: Samples in each class for the train (left) and test (right) sets

The instructions for this part were to train a classifier with clean and dirty images and evaluate the accuracy as well as analyze its failure cases. In this section, I build a convolutional neural network (CNN) that, once trained, is able to automatically classify new images into one of these categories. I

[1] https://www.kaggle.com/rodrigolaguna/clean-dirty-containers-in-montevideo

use the Keras library which provides a high-level interface to TensorFlow. To properly train and evaluate the implemented systems, the dataset is divided into a training set of 2217 images (75%) and a test set, that contains 1195 images (35%). In Fig. 2 it can be observed that the data samples are more or less equally distributed across the different classes, so the dataset is balanced.

2.1 Network Architecture

The architecture of our network is summarized in Figure 3. It contains eight learned layers — five convolutional and three fully-connected. I first add a convolutional 2D layer with 32 filters, a kernel of 3x3, the input size as our image dimensions, 256x256, and the activation as ReLU.

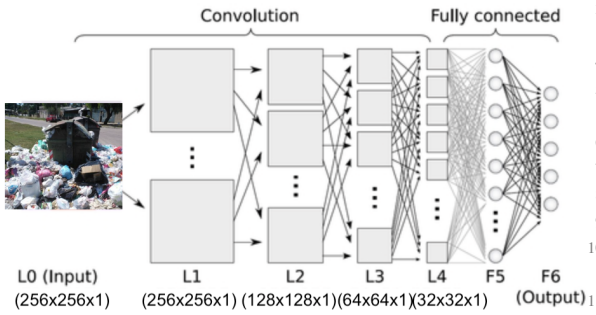

Figure 3: CNN Architecture

Convolutional Layers (Conv Layers)

```
model.add(layers.Conv2D(32, (3, 3),
    input_shape=(256, 256, 3)))
```

This layer is the first to extract information from the image by moving over it, a few pixels at a time and applying the convolutional operation. The output of these pixels is the dot product of the filter value and the pixel value. This process continues for the rest of the pixels and end with the entire image convolved to a feature map. In simple terms, a convolutional layer detects features/information from the image by placing and moving filters consecuetively. After that, I add a max pooling layer that halves the image dimension, so after this layer, the output is 128x128x3.

Pooling Layer (MaxPooling)

```
model.add(layers.MaxPooling2D(pool_size
    =(2, 2)))
```

The Pooling layer reduces the size of this feature map to only retain important information and make the training process easier and quicker.I use a Max Pooling layer in the CNN which moved over the feature map in a similar fashion like the previous filter and only take the maximum value of the cluster over which it is focusing.

Fully-Connected layer

Finally, the model flatten the output of the CNN layers, feed it into a fully-connected layer, and then to a sigmoid layer for binary classification. In this case, it is composed of 2 layers; one with 64 nodes and the last one with a single output node for binary classification. I add a Dropout layer which randomly sets input units to 0 with a frequency of 50% at each step during training time, to helps prevent over fitting. 3 of these layers were stack together, with each subsequent CNN adding more filters.

```
model = models.Sequential()
# This is the first convolution
model.add(layers.Conv2D(32, (3, 3),
    input_shape=(256, 256, 3)))
model.add(layers.Activation('relu'))
model.add(layers.MaxPooling2D(pool_size
    =(2, 2)))

# This is the second convolution
model.add(layers.Conv2D(32, (3, 3)))
model.add(layers.Activation('relu'))
model.add(layers.MaxPooling2D(pool_size
    =(2, 2)))

# This is the third convolution
model.add(layers.Conv2D(64, (3, 3)))
model.add(layers.Activation('relu'))
model.add(layers.MaxPooling2D(pool_size
    =(2, 2)))

# Flatten the results to feed into a DNN
model.add(layers.Flatten())

# 64 neuron hidden layer
model.add(layers.Dense(64))
model.add(layers.Activation('relu'))

# Dropout with a frequency of 50\%
model.add(layers.Dropout(0.5))

# 1 output neuron.
model.add(layers.Dense(1))
model.add(layers.Activation('sigmoid'))
```

Next, I configure the specifications for model training. I trained the model with the binarycrossentropy loss and using the RMSProp optimizer. RMSProp is a sensible optimization algorithm because it automates learning-rate tuning for us. I added accuracy to metrics so that the model will monitor accuracy during training.

```
model.compile(loss='binary_crossentropy'
    ,
              optimizer='rmsprop',
              metrics=['accuracy'])
```

2.2 Results

I trained the network for 10 and 50 epochs and compared the result with different batch size. This approach gets us to a validation accuracy of 0.68-0.76 after 50 epochs, I choose that number arbitrarily because the model is small and uses aggressive dropout, it does not seem to be over fitting too much by that point. I trained it using three different batch size: batch mode: where the batch size is equal to the total dataset thus making the iteration and epoch values equivalent

- batch mode: where the batch size is equal to the total dataset thus making the iteration and epoch values equivalent
- mini-batch mode: where the batch size is greater than one but less than the total dataset size. Usually, a number that can be divided into the total dataset size.
- stochastic mode: where the batch size is equal to one. Therefore the gradient and the neural network parameters are updated after each sample.

10 epochs	=1	=32	=64	=128	=256
Recall	.87	.52	.57	.62	.70
Precision	.66	.78	.77	.77	.70
Accuracy	.71	.69	.68	.72	.71
F1	.75	.62	.65	.70	.72

Table 1: Results after 10 epochs for different Batch Size

50 epochs	=1	=32	=64	=128	=256
Recall	.77	.72	.74	.58	.73
Precision	.65	.64	.70	.75	.74
Accuracy	.68	.77	.76	.69	.74
F1	.70	.70	.73	.65	.73

Table 2: Results after 50 epochs for different Batch Size

What I notice is that the result is not drastically different when training the network through different epochs while using mini-batch mode and stochastic batch. On the contrary, the result become more stable result when training the NN with a batch mode for 50 epochs.

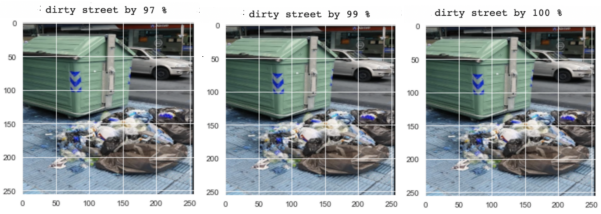

Figure 4: Classification of a dirty street for 10 epochs with a mini-batch mode/ stochastic mode/ batch mode (from left to right)

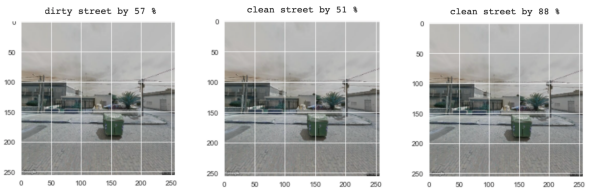

Figure 5: Classification of a clean street for 10 epochs with a mini-batch mode/stochastic mode/ batch mode (from left to right)

Here is what it means from two concrete examples: the first one is an obvious example of a dirty street that is interpreted as such in all our examples, and the second one is a clean street often interpreted as dirty according to our different parameters. As the images in Figure 5 are taken from a distance, the algorithm classifies this street as dirty with a mini-batch, and as the batch size increases, this case of a clean street is seen as such. We will examine the metrics listed in the table to better understand this result.

Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations.Accuracy is a great measure but only when you have symmetric datasets where values of false positive and false negatives are almost same. Since it's no't the case (except for the case we train a NN with a batch of 64 during 50 epochs or when we're using batch mode) we will pay attention of others metrics in order to evaluate the model.

Precision talks about how precise/accurate your model is out of those predicted positive, how many of them are actual positive. There is a tendency to increase the precision when the batch size increase.

Recall actually calculates how many of the Actual Positives our model capture through labeling it as Positive, there is a tendency to decrease when the

batch size increase.

F1 Score might be a better measure to use if we need to seek a balance between Precision and Recall AND there is an uneven class distribution (large number of Actual Negatives).

The results are therefore in our case more stable in batch mode, training the algorithm between 10 and 50 epochs does not have much difference. After having understood the principle of metrics, we understand that accuracy is not the most important metric, but we need an armony between the different metrics. Part 2 will allow us to understand how to improve our model by focusing on the pixels taken into account during the classification.

3 Network interpretation with Grad-Cam

In this section, we'll show how we interpret the previous result using **Class Activation Maps (CAM)**, a powerful technique used in Computer Vision for classification tasks. It allows to inspect the categorized image and understand which parts/pixels of an image have contributed more to the final output of the model. That means that we would be able to see which parts of the picture activate the "Dirty" class the most. This is very useful since it help to understand which layers need to be modified to improve the accuracy of our model.

3.1 Architecture

I used **Grad-Cam** for our model, an improve of CAM providing better localization. The main difference is that Grad-Cam, unlike CAM, uses the gradient information flowing into the last convolutional layer of the CNN to understand each neuron for a decision of interest.

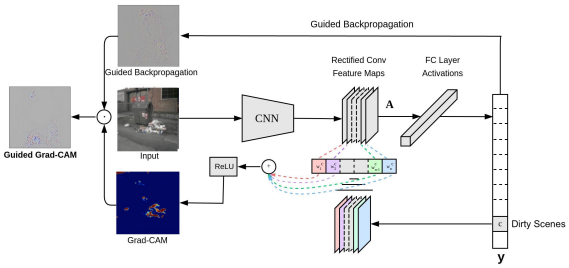

Figure 6: Grad-CAM Architecture

The architecture is complex but the output is intuitive. From a high-level, we take an image as input and create a model that is cut off at the layer for which we want to create a Grad-CAM heat-map. We attach the fully-connected layers for prediction. We then run the input through the model, grab the layer output, and the loss. Next, we find the gradient of the output of our desired model layer. From there, we take sections of the gradient which contribute to the prediction, reduce, resize, and rescale so that the heat-map can be overlaid with the original image.

3.2 Results

I started out creating Grad-CAM heat-maps for the last convolutional layer in our model, Conv1. In theory, the heat-map for this layer should display the most accurate visual explanation of the object being classified by the model.

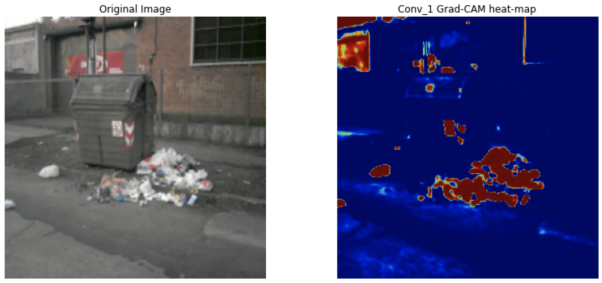

Figure 7: Grad-CAM heat-map of conv1 layer

I noticed that while the heat-map does emphasize the classified object, a wast in our case, it is very precise but no't mostly right. The emphasized region (red) encapsulates regions of the wast with much precision. The region includes part of the container and the background. We know the model sees a wast, but we are not quite sure what it is about the wast that convinces the model that this is, in fact, a wast. I explore model layers using this function to make the Grad-CAM heat-map more precise.

Studied the heat-maps make us understand the logic in how the model learns. The first layers (blocks 1) is detecting contours and borders in the image. Depthwise layers de-emphasize objects while project expand layers de-emphasize contours. The next layers (blocks 2 and 3) are detecting concepts in the image. The last convolution block sees all the results upside down and seems to detect what is not a wast and classifies the final result as this.

At this point in the architecture of the model, the shapes that define the characteristics of a wast around a trash begin to stand out from the overall

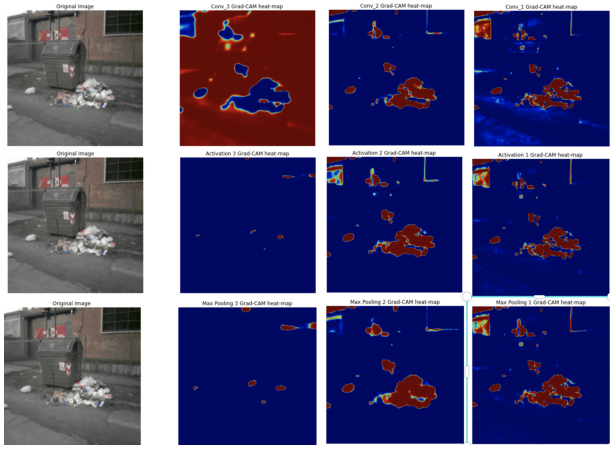

Figure 8: Grad-CAM heat-map of all layer(Conv/Activation/MaxPooling)

contours of the image. You can see how the wast are accentuated in shades of red and yellow, while the rest of the image is colored in blue and green. You can see how the model reasons how the characteristics of a wast differentiate this image as we move into the layers of the model. The last layer of the model, Conv1, correctly identifies the general region of the object, but it does not capture the nuances of the object that exist in the Grad-CAM heatmaps of the previous layers.

To incorporate earlier layers, I averaged together Grad-CAM heat-maps from all model layers. You can see the result below.

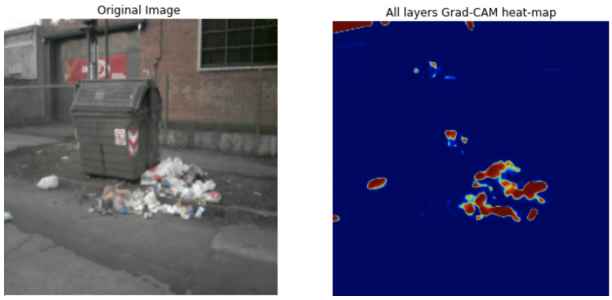

Figure 9: Grad-CAM heat-map of all layer averaged together

The Grad-CAM heat-map now emphasizes the dirt, and de-emphasizes the trash and the background. Overall, we have a much more precise region of emphasis that locates the dirt. We know that the model classifies this image as dirty due to its intrinsic features, not a general region in the image. Through the all-layer Grad-CAM I've learned about the models strengths. The techniques presented here helped us to provide more transparency and create more trust in models throughout the improvement process.

4 Conclusion

In this paper, I attempted to analyse neural network characteristics. A simplified model that classifies the two objects. The classification of higher-class information can be further investigated. We have seen a new technique for interpreting convolutional neural networks which are a state-of-the-art architecture, especially for image-related tasks.

References

R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, D. Batra. 2016. *Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization*,

B. Zhou, A. Khosla, A.Lapedriza, A. Oliva, A. Torralba. 2015. *Deep Features for Discriminative Localization*. Computer Science and Artificial Intelligence Laboratory, MIT.

K.Srinivas, B.Kavitha Rani, M.Varaprasad Rao, G.Madhukar *Convolution Neural Networks for Binary Classification*. 2019. Journal of Computational and Theoretical Nanoscience

Yann Le Cun, Léon Bottou, Yoshua Bengio, and Patrick Haffner *Gradient-based learning applied to document recognition. Proceedings of the IEEE.* 1998.