Deep CNN and SVM Waste Classifier for Recyclability purposes

Chiara Giacanelli
(Sapienza University of Rome, Machine Learning course)

Abstract - As the years go by, it's becoming more and more necessary to take seriously into account the impact of our actions on the environment. For this reason, I've decided to use Convolutional Neural Networks (CNN), Support Vector Machine (SVM) models together with a combination of both (CNN having as softmax layer a SVM) in order to predict in which of the six recyclable categories (glass, garbage, plastic, paper, metal and cardboard) a particular object belongs to. I eventually evaluated such models and compared the result obtained.

1 Introduction

Waste is becoming a serious global issue. Infact, as the living standards rise, increasing amount of garbage is being generated and our planet is in need of a sustainable society in which the products we use and throw away are capable of being reconverted in new usable ones. The World Bank warns that global waste will increase up to 70 percent on current levels by 2050 unless urgent actions are undertaken [1].
This also comes up with costumers' confusion on how and where to throw away certain objects because they're insecure of their material.
For these reasons, I consider Computer Vision a very powerful tool to enhance the recycling process by creating an automatic method for sorting trash. For this problem, I used the TrashNet dataset, containing six classes of 400/500 images each, apart from the 'trash' one which contains about 200 images. TrashNet is used for supervised learning with various images of differentiated materials and categorizing these images.
The dataset can be found on Kaggle at this link: **https://www.kaggle.com/datasets/asdasdasasdas/garbage-classification**.
I used Google Colab to run the notebook in order to have GPU to get results in a reasonable amount of time.

2 Related work

For the sake of this project I did some research and found out many interesting past work to deepen:

- **Classification of Trash for Recyclability Status**[1]: it's a study conducted by Standford University that takes into account trash classification. I personally read the article as I was searching for ideas for the project and thought that it would be an interesting idea to take it as reference; this study in particular used support vector machines (SVM) with scale-invariant feature transform (SIFT) features and a convolutional neural network (CNN). However, their CNN didn't fully manage to train properly, so I decided to try to improve their results building a more robust neural network.

- **Fine-Tuning Models Comparisons on Garbage Classification for Recyclability**: this is a work done by Konya Technical University aimed to develop a deep learning application which detects types of garbage into trash in order to provide recyclability with vision system. AlexNet, GoogleNet, ResNet, VGG-16 and SquezeeNet were used as CNN fine tuned models (so pre-trained models) in the same dataset I used, but they didn't apply any data augmentation on the images[2].

Other various works can be found on Kaggle, an online community of data scientists and machine learning practitioners, in which people shared their trained models on the same dataset with different accuracy.

3 Work structure and methods

My work has been organized in this way: the first step has been data collection and pre-processing. Images were taken from the dataset and data augmentation was needed in order to improve the accuracy of the classifier (I will explain the process better in the following paragraph). Some examples of the techniques applied were: rescaling, zooming, horizontal and vertical flip and shear ranging.
I then imported all the needed libraries in order to train my models: I used *tensorflow* and *keras* to build and train the Convolutional Neural Network and *sklearn* to train the Support Vector Machine. Doing my researches, I came across an interesting study entitled: ***"Deep Learning using Linear Support Vector Machines"***[3], conducted by Yichuan Tang from the Univerisity of Toronto in which it's demonstrated a small but consistent advantage of replacing the softmax layer

with a linear support vector machine. I then thought that it would be good to implement it on my dataset and analyze the differences between the accuracy from the two separated models.

4 Data collection and pre-processing

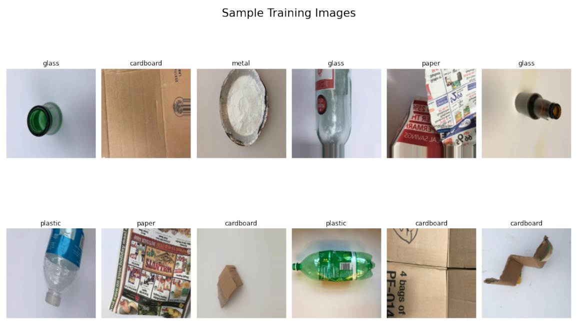
The first step I had to do was the data acquisition. I downloaded the directory containing images of the six categories from the GitHub repository: **https://github.com/garythung/trashnet**.

The dataset consisted of 2527 images divided in 6 directories for each category as follows:

- Paper images: 594
- Plastic images: 482
- Trash images: 137
- Cardboard images: 403
- Glass images: 501
- Metal images: 410

More specifically, I downloaded the resized dataset, in which all the pictures have been resized down to 512 x 384 pixels (that I eventually rescaled to the size of 256x256).
I loaded all the files and double checked with python.os if the amount of images was the right one.

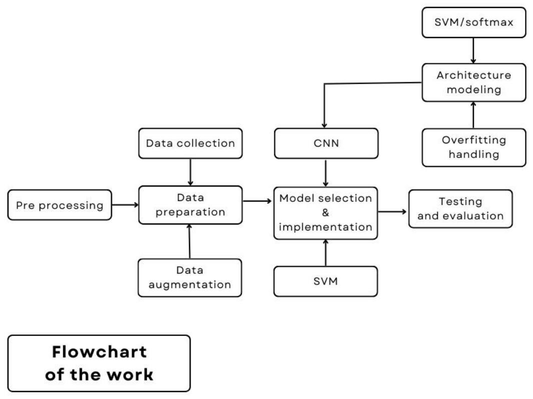
Since the dataset contained all images together, I had to perform a folder splittig in order to divide it in three directories: training, validation and testing, with a ratio of 0.8, 0.1 and 0.1 respectively.
Below you can see a sample of 12 random images of the dataset picked from every class.



4.1 Data augmentation

Being the dataset composed of a little amount of images, I thought it appropriate to apply data augmentation techniques on them.

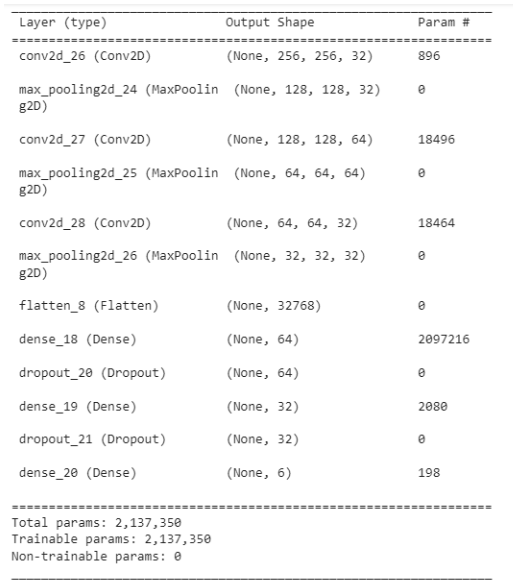
Data augmentation aims to increase the dimension of our dataset by transforming images we have in our training set and creating copies of them in order to let the model see something a little different than what it's seen before in each iteration. This extra variance in the training data is what helps the model on new data. These transformations involve various techniques such as random rotation, zooming, random shifting, flipping and adjusting the color or contrast.
To apply it on my images I used the *keras ImageDataGenerator class*[4]. I built an ImageDataGenerator with these data augmentation strategies: rescaling, zooming, shear ranging, flipping (horizontally and vertically) and shift ranging (width and height). I considered all the important data augmentation techniques in order to let the models train and learn with more robustness. I then used this class to generate batches of augmented data for training. Augmented techniques are done only on the training set because our aim is to decrease the generalization error during model training.



5 Model selection

Convolutional Neural Networks Convolutional Neural Networks are considered the neural networks that perform the image classification task at best. I will use the term **CNN** from now on to identify them.
To build my CNN I used the tensorflows' Sequential model. I built a twelve layer convolutional neural network, consisting of:

1. Convolutional 2D layer with 32 filters, size 3x3, 'relu' activation function and 'same' padding;
2. MaxPooling 2D layer
3. Convolutional 2D layer with 64 filters, size 3x3, 'relu' activation function and 'same' padding;
4. MaxPooling 2D layer
5. Convolutional 2D layer with 32 filters, size 3x3, 'relu' activation function and 'same' padding;
6. MaxPooling 2D layer
7. Flatten layer;
8. Dense layer with 64 units and 'relu' activation function;
9. Dropout of 0.2 to avoid overfitting;
10. Dense layer with 32 units and 'relu' activation function;
11. Dropout of 0.2 to avoid overfitting;
12. Output Dense layer;



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_26 (Conv2D)           (None, 256, 256, 32)      896

max_pooling2d_24 (MaxPoolin  (None, 128, 128, 32)      0
g2D)

conv2d_27 (Conv2D)           (None, 128, 128, 64)      18496

max_pooling2d_25 (MaxPoolin  (None, 64, 64, 64)        0
g2D)

conv2d_28 (Conv2D)           (None, 64, 64, 32)        18464

max_pooling2d_26 (MaxPoolin  (None, 32, 32, 32)        0
g2D)

flatten_8 (Flatten)          (None, 32768)             0

dense_18 (Dense)             (None, 64)                2097216

dropout_20 (Dropout)         (None, 64)                0

dense_19 (Dense)             (None, 32)                2080

dropout_21 (Dropout)         (None, 32)                0

dense_20 (Dense)             (None, 6)                 198

=================================================================
Total params: 2,137,350
Trainable params: 2,137,350
Non-trainable params: 0
_________________________________________________________________
```

Figure 1: Architecture of the CNN model

Support Vector Machines Being SVM a very good machine learning model for classification, I wanted to test the accuracy of it on an image dataset and comparing it with the more complex model that's the CNN.
I started by implementing a function that collected all the images in a numpy data structure, because I had to train the model with one-dimensional data and I couldn't work with the ImageDataGenerator class. I cycled over my dataset and created a *numpy* array for my flat data and the targets. Flat data corresponds to all the flattened images (one dimensional arrays), because SVM doesn't work with 2D arrays. The target has been assigned by giving an index to all the classes (from 0 to 5). I then splitted the dataset in training and testing, with the relative dimension of 80/20.
Since the 'trash' feature had just 104 samples, I applied the SMOTE oversampling technique to balance my dataset, avoid under-generalization and increase the training performance.
I trained the SVM model with these parameters: C = 1, *'rbf'* kernel and *'scale'* gamma. I was aimed to perform hyperparameter tuning but my Google Colab notebook crashed everytime for little RAM space so I managed to train a simple SVM, but I still obtained quite good results (62% of accuracy).

Convolutional Neural Networks with SVMs In this study, I used multiclass SVMs to train my Deep CNN for my classification. To do this, I had to differentiate the SVM with respect to the activation function of the softmax layer.

To better explain it, I have to remind how SVM learning works. SVMs learning consists of the following optimization function:

$$\min_{\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} + C\sum_{n=1}^{N} \max(1 - \mathbf{w}^\mathsf{T}\mathbf{x}_n t_n, 0)$$

The objective of this function is known as the L1-SVM, or primal form problem, with the standard hinge loss. Since L1-SVM is not differentiale, a common version is known as the L2-SVM, that's differentiable and imposes a squared hinge loss for points that violate the margin:

$$\min_{\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} + C\sum_{n=1}^{N} \max(1 - \mathbf{w}^\mathsf{T}\mathbf{x}_n t_n, 0)^2$$

With these premises, I built a the same CNN model as the first one, but I applied a regularizer to apply a penalty on the last layer's kernel (the L2 keras regularizer). I also compiled my model with a *squared hinge* loss function, because I have to perform a multi class classification (for binary classifications I have a hinge loss).

6 Evaluation phases and results

Convolutional Neural Networks The CNN was trained an image size of 256x256, 50 epochs, 71 steps per epoch (32 samples per batch) and 8 validation steps. Plotting the accuracy and loss levels, results were quite good: I achieved a val_accuracy of around 70% (77% on the 49th epoch and 66% in the following one). The val_accuracy indicates the accuracy of the predictions of a randomly separated validation set after each epoch. However, the general accuracy continued to increase. It's important to say that accuracy and val_accuracy became quite stable after adding the Dropout layers: infact, at the beginning I wasn't aware of the possible overfitting that could occur and these values had too high difference between eachother. So, my model would fit training data very well but couldn't generalize with validation data.

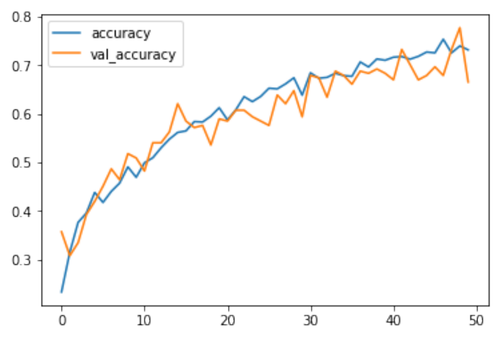


Figure 2: Accuracy levels

The loss levels were actually similiar: general loss tended to decrease epoch per epoch.

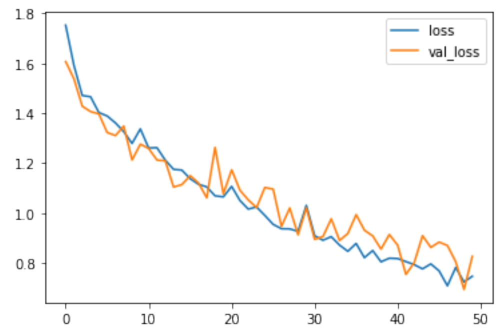
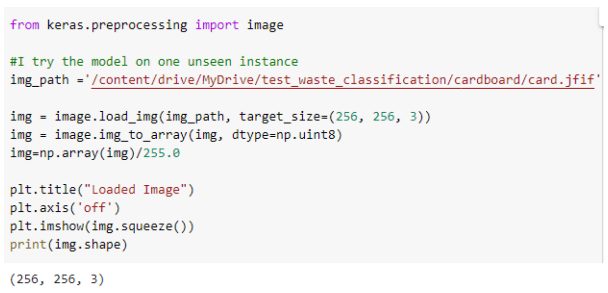


Figure 3: Loss levels

I then tested the model on random images not belonging to the dataset: I created another folder in my drive called *'test_waste_classification'* in which I downloaded many images found online for each of the class labels. I could have also downloaded themselves directly from Google Images but I thought that using the drive could be easier to test the images on. This is an example of how the model worked on an unseen cardboard image:



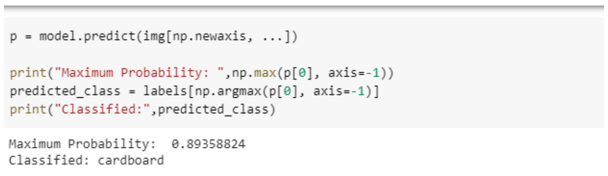
```
from keras.preprocessing import image

#I try the model on one unseen instance
img_path ='/content/drive/MyDrive/test_waste_classification/cardboard/card.jfif'

img = image.load_img(img_path, target_size=(256, 256, 3))
img = image.img_to_array(img, dtype=np.uint8)
img=np.array(img)/255.0

plt.title("Loaded Image")
plt.axis('off')
plt.imshow(img.squeeze())
print(img.shape)
```

```
(256, 256, 3)
```



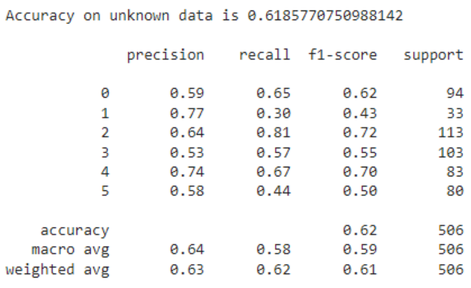
```
p = model.predict(img[np.newaxis, ...])

print("Maximum Probability: ",np.max(p[0], axis=-1))
predicted_class = labels[np.argmax(p[0], axis=-1)]
print("Classified:",predicted_class)
```

```
Maximum Probability:  0.89358824
Classified: cardboard
```

The model made the right prediction with (assigned to the class label 0.89 percent of probability, whereas 0 for 'glass', 'trash' and 'plastic' and a little more on 'metal' and 'paper'). I eventually cycled over 20 unseen images and tested the model on them, and as a result the CNN predicted well 17 of them.

Support Vector Machines To analyze how the training went, I printed out the classification report which showed 62% of overall accuracy. The 0 corresponds to the plastic class, the 1 to the trash, 2 to the paper, 3 to the glass, 4 to the cardboard and 5 to the metal one.
The class with higher precision is the trash one. The ones with the lowest precision were glass and metal, and I noticed that these are the classes that also the CNN easily misclassified.



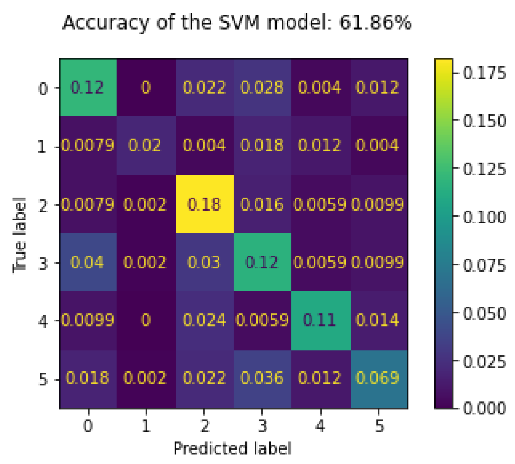
```
Accuracy on unknown data is 0.6185770750988142

              precision    recall  f1-score   support

           0       0.59      0.65      0.62        94
           1       0.77      0.30      0.43        33
           2       0.64      0.81      0.72       113
           3       0.53      0.57      0.55       103
           4       0.74      0.67      0.70        83
           5       0.58      0.44      0.50        80

    accuracy                           0.62       506
   macro avg       0.64      0.58      0.59       506
weighted avg       0.63      0.62      0.61       506
```

I also plotted the confusion matrix, useful for evaluating graphically the performance of the classification model.



Convolutional Neural Networks with SVMs This CNN was trained with an image size of 256x256, 80 epochs, 71 steps per epoch and 8 validation steps. I trained this neural network with more epochs because I noticed that the accuracy had the tendency to increase more slowly than the first CNN, and I wanted to analyze its behaviour within more training cycles.
The overall accuracy of this model was lower than the CNN with the softmax layer: infact, at the 80th epoch the validation accuracy was 66%. As we can see in the plot of the accuracy levels, in this case the validation accuracy struggled a little bit more in being close to the value of the accuracy, and it's probabily because of a too little available data to work on (infact, I already tried to manage overfitting by putting two Dropout layers after each Dense layer). Despite this, we can clearly see that accuracy tended to increase also in this case.

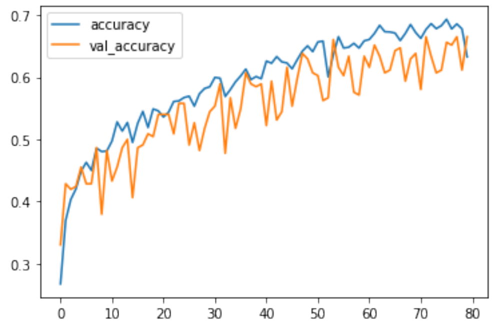


Figure 4: Accuracy levels of the CNN with SVMs

We can also see that also loss values tended to decrease.

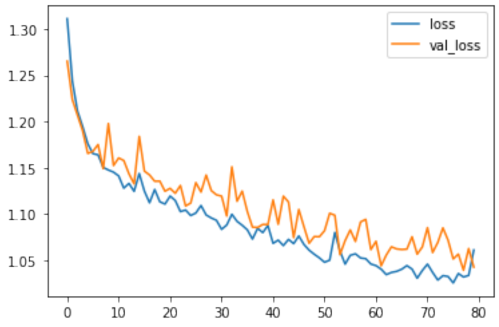
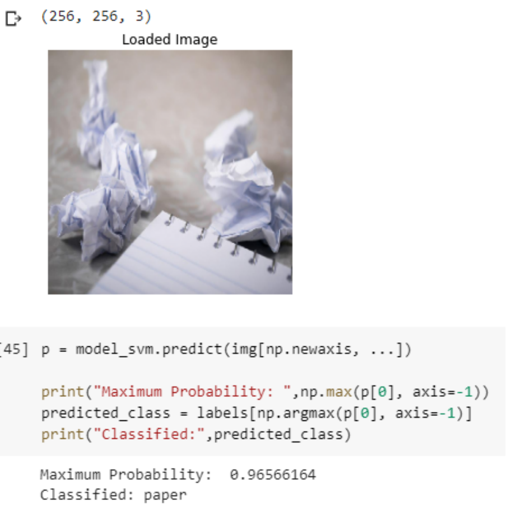


Figure 5: Loss levels of the CNN with SVMs

I tried to test the model on multiple unknown images, obtaining quite good results, as we can see in the image below, in which the model predicted the right class label with 96% of probability (it wasn't the case for all the images in the unknown test set, but I was glad that in some of them it worked properly).



```
(256, 256, 3)
```

```
[45] p = model_svm.predict(img[np.newaxis, ...])

     print("Maximum Probability: ",np.max(p[0], axis=-1))
     predicted_class = labels[np.argmax(p[0], axis=-1)]
     print("Classified:",predicted_class)

     Maximum Probability:  0.96566164
     Classified: paper
```

Even in this case, I eventually tested the model on a random set of 20 test images, and it managed to correctly predict 12 of them.

7 Conclusions

My work was aimed to create image classification models for trash recognition. I trained three different models aiming to analyze the behaviour of a Deep Neural Network on image classification against a more traditional classification algorithm, that's the Support Vector Machines. I then built a CNN having SVMs as last layer and compared the results obtained. Convolutional Neural Networks achieved the best results on the dataset that I used, and in this case having the softmax as last layer showed to be the best choice for classification.

What's here to say is that one of the biggest pain points is the lack of bigger datasets to train the models with. Therefore, in order to create a more accurate system, there needs to be a large and continuously growing data source. Despite this, I can say that the classifiers achieved a good overall accuracy ($> 60\%$), which had the tendency to increase for CNNs and CNNs with SVMs. This tells that, training with more epochs, a more powerful GPU and bigger RAM, could lead to an even better classifier.

Notes

1. *Classification of Trash for Recyclability Status*, https://tinyurl.com/dkck2sz2
2. *Fine-Tuning Models Comparisons on Garbage Classification for Recyclability:* https://arxiv.org/ftp/arxiv/papers/1908/1908.04393.pdf
3. *"Deep Learning using Linear Support Vector Machines"*: https://arxiv.org/pdf/1306.0239v4.pdf
4. Documentation: *https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator*

References

[1] Sultana Yesmin. Global waste crisis: A rising threat to environment. *moderndiplomacy*, 2019.