Università degli Studi di Verona

Garbage Classification

Computer Engineering For Robotics and Smart Industry

Machine Learning And Artificial Intelligence

Raffaele Berardo - VR465726

Nicola Marchiotto - VR462317

2021

Contents

1 Motivation And Rationale

Nowadays machine learning has two main objectives, classification of data and making predictions for future outcomes.

We focused on **image classification**, trying to solve a real problem scenario which is the **classification of different materials in the proper garbage category**. We know that sorting waste in the proper category has a good impact on our environment. The classification of a single trashed item in the proper garbage category could help in a big way the recycling process, which main obstacle is in fact the separations of different materials in the undifferentiated recycling collection. Even with the right will power, people may not be able to rightly identify the proper garbage category.

We would ideally like to design a classifier able to recognize teh garbage category of the proposed item represented in an image. In the future this application could be implemented on a pick and place robot, in order to pick and put in the proper category an identified waste. You can find the used dataset at the following link.

You can download the code at the following github repository where the executable files are the 2 jupyter notebooks called *garbage_classification.ipynb* and *garbage_classification_CNN.ipynb*

2 State Of The Art

At the current state of the art there are several models able to classify images, putting them in the proper class. We wanted to test several models, also the simpler ones, in order to see what could be the results obtained using less advanced models. We performed tests using the Naive Bayes algorithm, the KNN and the SVM model with different strategies, expecting an increment in the accuracy starting from the first one and finishing with the SVM.

We then focused our attention on the CNN models which are able to extract very complex features from our images. We used *pretrained* models in order to have a better feature extractor and a faster training. The pretrained models used are:

- VGG19
- ResNet50

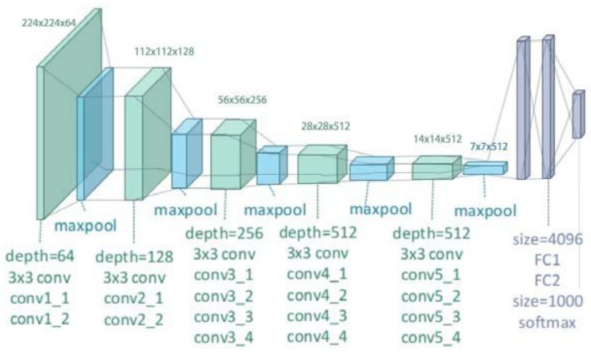
We'll now take a brief look to the architecture of these nets.

2.1 VGG19

The VGGNet was developed by *Karen Simonyan* and *Andrew Zisserman* from the Visual Geometry Group (VGG) research lab at Oxford University.

It had a very simple and classical architecture, with 2 or 3 convolutional layers and a pooling layers, then again 2 or 3 convolutional layers and a pooling layer, and so on reaching a total of just 16 or 19, called respectively VGG16 and VGG19.

There is a final dense network with 2 hidden layers and the output layer. It used only 3×3 filters, but many of them. Here you can see the model saved on the Keras framework.[1]



224x224x64
112x112x128
56x56x256
28x28x512
14x14x512
7x7x512
maxpool
maxpool
maxpool
maxpool
maxpool
depth=64 3x3 conv conv1_1 conv1_2
depth=128 3x3 conv conv2_1 conv2_2
depth=256 3x3 conv conv3_1 conv3_2 conv3_3 conv3_4
depth=512 3x3 conv conv4_1 conv4_2 conv4_3 conv4_4
depth=512 3x3 conv conv5_1 conv5_2 conv5_3 conv5_4
size=4096 FC1 FC2 size=1000 softmax

Figure 1: VGG19 Architecture

2.2 ResNet50

Researchers observed that it makes sense to affirm that *the deeper the better* when it comes to convolutional neural networks. This makes sense, since the models should be more capable of learning. However it has been noticed that after some depth, the performance degrades. This was one of the bottlenecks of the VGGNets. They couldn't go as deep as wanted because they started to lose generalization capability.

The ResNets solve this problem of *vanishing gradient*, called in this way because in the VGGNets very deep network, the gradients from where the loss function is calculated easily shrink to zero after several applications of the chain rule. This result on the weights never updating its values and therefore, no learning is being performed.

With ResNets, the gradients can flow directly through the skip connections backwards from later layers to initial filters, avoiding the cited problem.

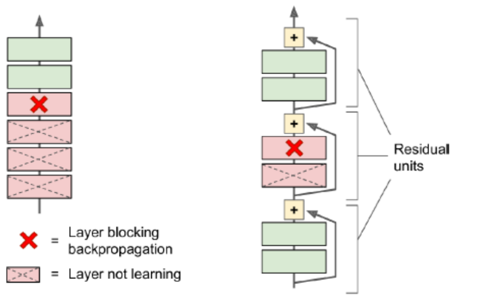
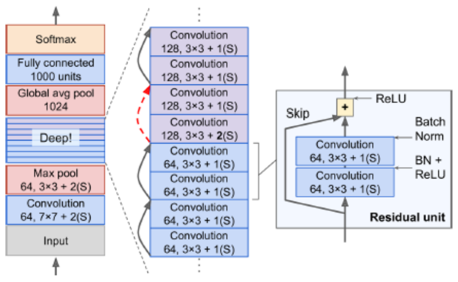


Figure 2: Regular DNN and deep residual network



Softmax
Fully connected 1000 units
Global avg pool 1024
Deep!
Max pool 64, 3×3 + 2(S)
Convolution 64, 7×7 + 2(S)
Input
Convolution 128, 3×3 + 1(S)
Convolution 128, 3×3 + 2(S)
Convolution 64, 3×3 + 1(S)
Skip
ReLU
Batch Norm
BN + ReLU
Residual unit

Figure 3: ResNet Architecture

The *skip connections* are used to add the input of a layer also to the output of a layer located a bit higher up the stack.

3 Objectives

The objective of our problem is to classify images representing objects made of several materials in the proper recycling category. To deal with our problem we found a dataset on kaggle website composed by 2.527 images divided in 6 different classes: *glass, metal, cardboard, plastic, metal, trash*.

Each image is labeled with a number that goes from 0 to 5, following the list order. The purpose of our project is to find a model able to classify the dataset pictures, testing machine learning and deep learning models.

We started by using the classical machine learning models *(Bayes classifiers, k-nn, SVM)* and then finished with the more advanced deep learning models.

4 Methodology

4.1 Dataset

Our dataset is composed by 2.527 colored images divided in 6 classes which are:

0. Glass
1. Paper
2. CardBoard
3. Plastic
4. Metal
5. Trash

Where each category is labeled with a number, from 0 to 5. You can see an example of how the images look like in the picture below.

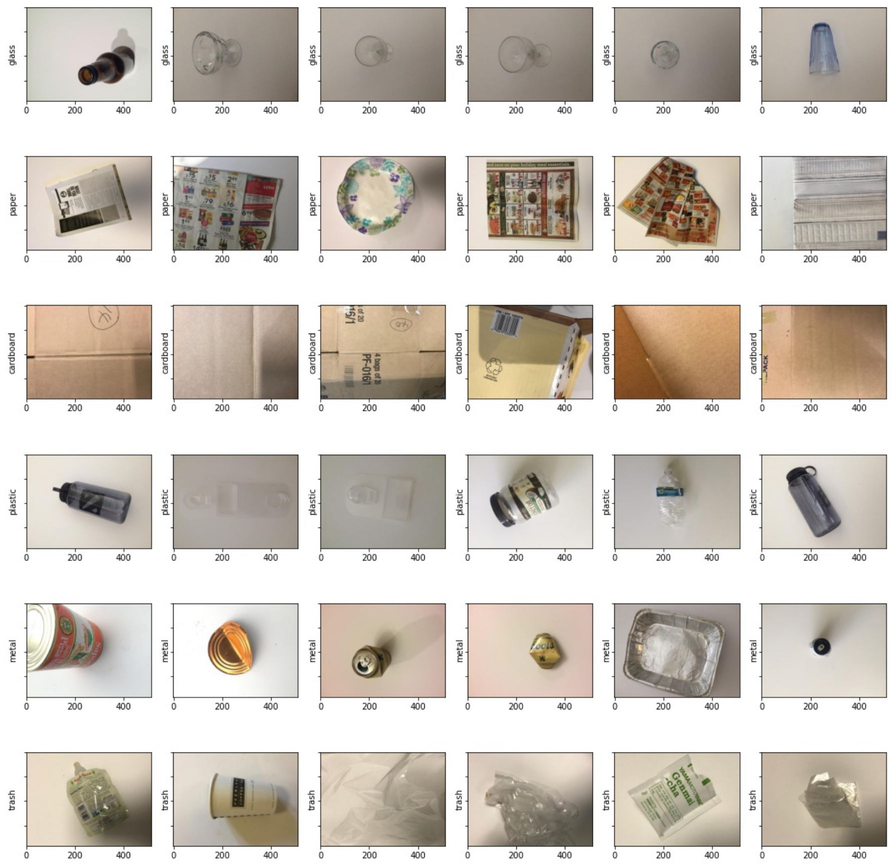


Figure 4: Samples images

The images share all the same size, which is 512×384 pixels and all pictures represent just a single item.

More in detail, we have 393 cardboard images, 491 glass images, 400 metal images, 584 paper images, 472 plastic images and 127 trash images so they are more or less balanced. The dataset is available here.

4.2 Tools

Computational

To execute our project code we use the *Google Colaboratory* platform because allows to directly execute code on the cloud exploiting the Jupyter notebooks and also to share code with all the work group.

This platform supplies a **GPU** runtime environment for free, very useful sinces these GPUs are optimized for training artificial intelligence and deep learning models as they can process multiple computations simultaneously.

Libraries

To pursue the project objective we use the *python* programming language. It offers a large variety of useful tools when we have to treat machine learning problems. In particular we use the following libraries and frameworks:

- **Numpy**, a package for scientific computing;
- **Matplotlib** and **Seaborn** to plot the results;
- **Scikit-learn** to have already implemented machine learning models;
- **Keras**, a framework to build neural networks

4.3 Methods and Algorithms

We decided to use several approaches to obtain a proper classifier. We start with the most basic Machine Learning classification algorithms finishing with some more advanced models. The used models were:

- Naive Bayes Classifier
- K-Nearest Neighbor
- Support Vector Machine (One Vs One and One Vs Rest strategy)
- CNN Pre-Trained Models (VGG19 and ResNet50)

As expected we obtained an increment in the results applying the models in the given order, going from a 35% accuracy to a 87% accuracy.

K-Fold Cross Validation

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into.

It is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model. The general procedure is as follows:

1. Shuffle the dataset randomly.
2. Split the dataset into k groups
3. For each unique group:
    - Take the group as a hold out or test data set
    - Take the remaining groups as a training data set
    - Fit a model on the training set and evaluate it on the test set
    - Retain the evaluation score and discard the model
4. The error estimation is averaged over all k trials to get total effectiveness of our model

As can be seen, every data point gets to be in a validation set exactly once, and gets to be in a training set k-1 times. This significantly reduces bias as we are using most of the data for fitting, and also significantly reduces variance as most of the data is also being used in validation set.

Grid Search

Grid-searching is the process of scanning the data to configure optimal parameters for a given model. Depending on the type of model utilized, certain parameters are necessary. It is important to note that Grid-searching can be extremely computationally expensive and may take your machine quite a long time to run.

Grid-Search will build a model on each parameter combination possible. It iterates through every parameter combination and stores a model for each combination. In the end we'll use the model with the best score.

4.4 Pre-processing

Feature reduction

In the non neural network algorithms, we performed a feature reduction given the high number of available features. We had to face with RGB images, given their size, 512×384, we had $512 \times 384 \times 3 = 589.824$ features for a single image.

We decide to scale the images by a factor of 4 both in height and width and we consider each value of the 3 RGB channel as a feature. Then we performed the PCA algorithm on these features to extract the more relevant ones, choosing a good amounts of components to preserve considering the **elbow plot**. We ended up with 250 features per item.

At first we took as a single feature the mean of the values of the 3 RGB channels, but since this choice resolved in lower performances in the various algorithms, we discard it.

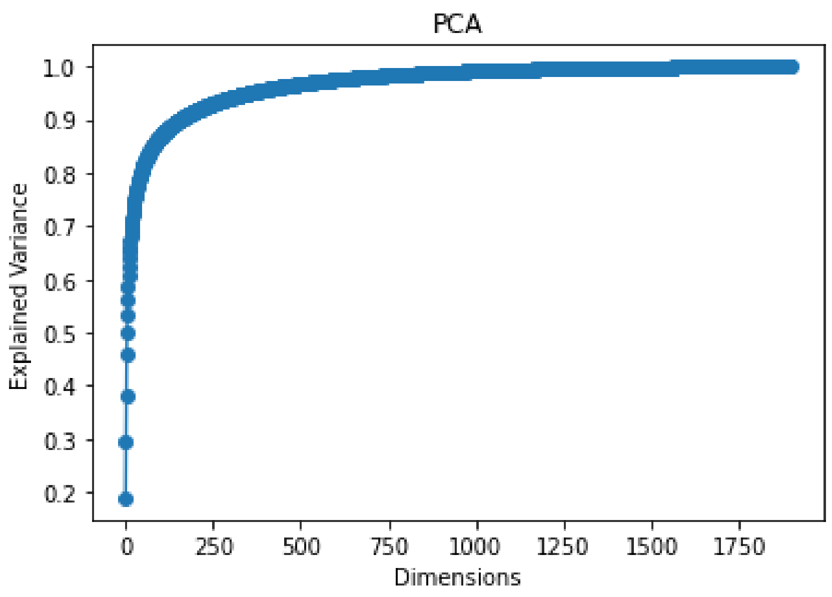


Figure 5: Elbow Plot

Resizing

Often, machine learning will recommend or require that you prepare your data in specific ways before fitting a machine learning model.

In the CNNs models we used,we had to **resize** the images of dimension 512×384, to be a suitable input for our architectures, which by default accepts images of input size 224×224 pixels.

One-Hot Encoding

In the neural network cases, we converted our labels from an integer format (0 to 5) to a one-hot encoding format where each number is encoded in a one row vector of dimension 6 where we have just 0 and 1 values.

We have the value of 1 just in the index place which represents the label, so for example the 0 label is represented as $[1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]^T$. We did this due to the fact that we don't have to consider an order between our categories.

Image Data Augmentation

In the CNN algorithms, we made use of image data augmentation which is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset, often used in this field due to the voracity of these algorthms. In fact augmentation techniques can create variations of the images that can improve the ability of models to generalize what they have learned to new images.

The Keras deep learning neural network library provides the capability to fit models using image data augmentation via the **ImageDataGenerator class**.

The class may be instanciated and the configuration for the types of data augmentation are specified by arguments to the class constructor. The main types of data augmentation techniques for image data that we used are: **rotation_range**, **width_shift_range**, **height_shift_range**, **zoom_range**, **bright-**

ness_range, **horizontal_flip**, **vertical_flip**.

Here you can see a picture example of what our augmentation looks like.

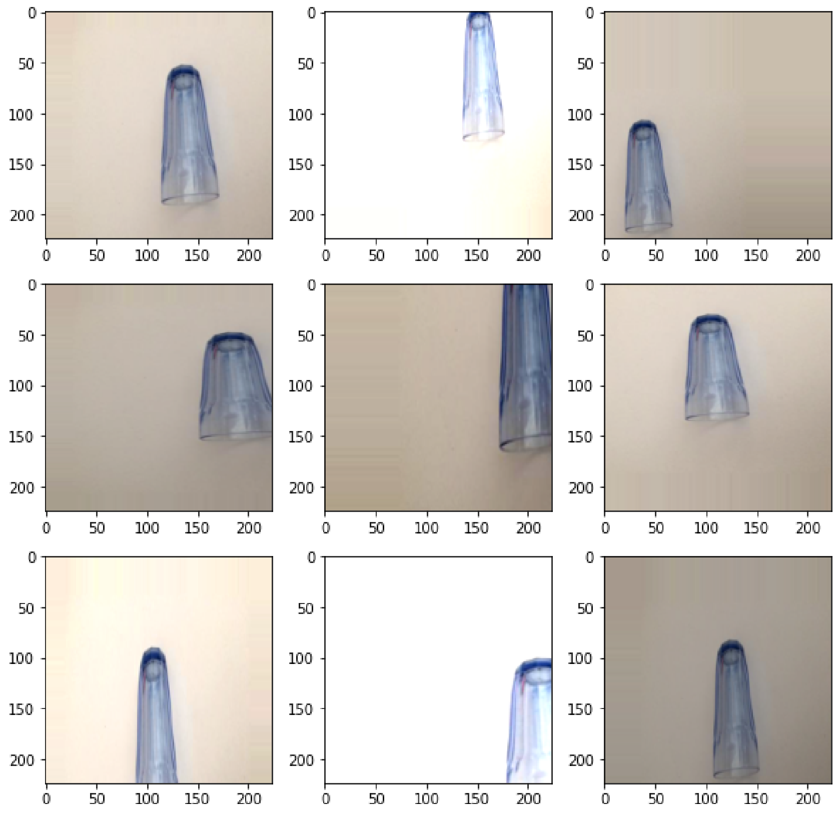


Figure 6: Image Data Augmentation for Garbage Images

Transfer Learning

For our purposes we use the VGG and ResNet architectures already trained, so **pretrained**, on the **imagenet** dataset which is a large visual database designed for use in visual object recognition software research.

Using transfer learning we can have great performance in terms of training time and overall results.

The most common incarnation of transfer learning in the context of deep learning is the following workflow:

1. Take layers from a previously trained model.
2. Freeze them, so as to avoid destroying any of the information they contain during future training rounds.
3. Add some new, trainable layers on top of the frozen layers. They will learn to turn the old features into predictions on a new dataset.
4. Train the new layers on your dataset.

A last, optional step that we perform, is fine-tuning, which consists of unfreezing the entire model you obtained above (or part of it), and re-training it on the new data with a very low learning rate. This can potentially achieve meaningful improvements, by incrementally adapting the pretrained features to the new data.

In **keras** we have a set of functions that allow us to freeze the pretrained layers easily. Then we can add our modeled layers and train just those for our purposes. Here we can see how we construct our network for the VGG and ResNet cases.

Here's the code for the VGG19 architecture:

```python
# Base pretrained model with no top layers for VGG19
base_model = keras.applications.VGG19(weights = "imagenet",
                                      include_top = False)

# Begin to add layers for our multiclass classification
avg = keras.layers.GlobalAveragePooling2D()(base_model.output)
dropout = keras.layers.Dropout(rate=0.3)(avg)
output = keras.layers.Dense(6, activation='softmax')(dropout)
model = keras.Model(inputs=base_model.input, outputs=output)
model.summary()
```

Here's the code for the ResNet50 architecture:

```python
# Base pretrained model with no top layers for ResNet50
base_model = keras.applications.ResNet50(weights="imagenet",
                                         include_top=False)


# Begin to add layers for our multiclass classification
avg = keras.layers.GlobalAveragePooling2D()(base_model.output)
dropout = keras.layers.Dropout(rate=0.3)(avg)
output = keras.layers.Dense(6, activation='softmax')(dropout)
model = keras.Model(inputs=base_model.input, outputs=output)
model.summary()
```

Global pooling can be used in a model to aggressively summarize the presence of a feature in an image. It is also sometimes used in models as an alternative to using a fully connected layer to transition from feature maps to an output prediction for the model.

Dropout is a technique where randomly selected neurons are ignored during training. They are "dropped-out" randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.

The effect is that the network becomes less sensitive to the specific weights of neurons. This in turn results in a network that is capable of better generalization and is less likely to overfit the training data.

5 Experiments And Results

5.1 Naive Bayes Classifier

As already mentioned, after feature reduction we end up working with 250 features. We divided our dataset in two parts, 1895 images for training and validation, 632 for testing.

We tried to fit our data in a Gaussian function and predict the test label through the Naive Bayes decision rule. We K-fold cross validated our test set to get better result, imposing k=10, so dividing our training set in 10 parts and using one of them to validate our model at each iteration.

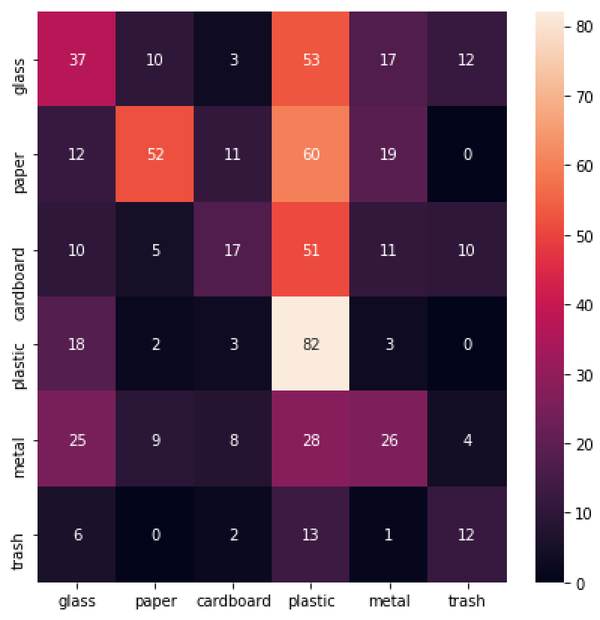


Figure 7: Naive Bayes classifier confusion matrix

	Precision	Recall	F1-score
Glass	0.34	0.28	0.31
Paper	0.67	0.34	0.45
Cardboard	0.39	0.16	0.23
Plastic	0.29	0.76	0.42
Metal	0.34	0.26	0.29
Trash	0.32	0.35	0.33
Accuracy	0.36		

Table 1: Precision and Recall results for each class

5.2 K-NN

The data fed to this method were in the same format of those we used for the Bayes Classifier, we still performed K-fold cross validation over the data and employed the Grid Search method to find the best k for the nearest neighbours procedure. we tested with $k \in [2, 3, 5, 10, 100]$. The best performance was with $k = 3$

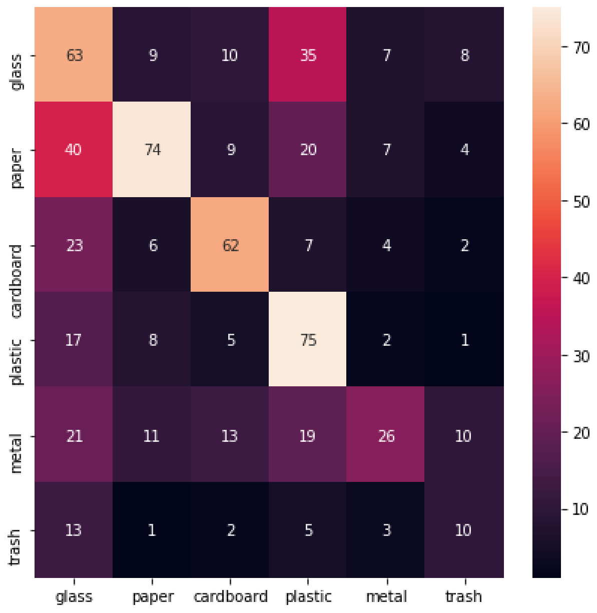


Figure 8: K-NN classifier confusion matrix with $k = 3$

	Precision	Recall	F1-score
Glass	0.36	0.48	0.41
Paper	0.68	0.48	0.56
Cardboard	0.61	0.60	0.60
Plastic	0.47	0.69	0.56
Metal	0.53	0.26	0.35
Trash	0.29	0.29	0.29
Accuracy	0.49		

Table 2: Precision and Recall results for each class

5.3 SVM

Since SVM are binary classifies, *One VS Rest* and *One Vs One* techniques were used to achieve multi class classification.

- **One Vs Rest:** is a heuristic method for using binary classification algorithms for multi-class classification. It involves splitting the multi-class dataset into multiple binary classification problems. A binary classifier is then trained on each binary classification problem and predictions are made using the model that is the most confident.
- **One Vs One:** Like one vs rest, one vs one splits a multi-class classification dataset into binary classification problems. Unlike one vs rest that splits it into one binary dataset for each class, the one vs one approach splits the dataset into one dataset for each class versus every other class.

The data fed to these methods were in the same format of those we used for the Bayes Classifier and K-NN algorithm. We performed cross validation in both methods but only grid search, with parameter the type of kernel used, in the One Vs Rest case, due to the big time of execution necessary for this latter. The kernel searched where of the type [*poly, linear, rbf*] with possible degree of the polynomial kernel belonging to [*2,3*]. The best resulting kernel was the *radial basis function*.

One Vs One

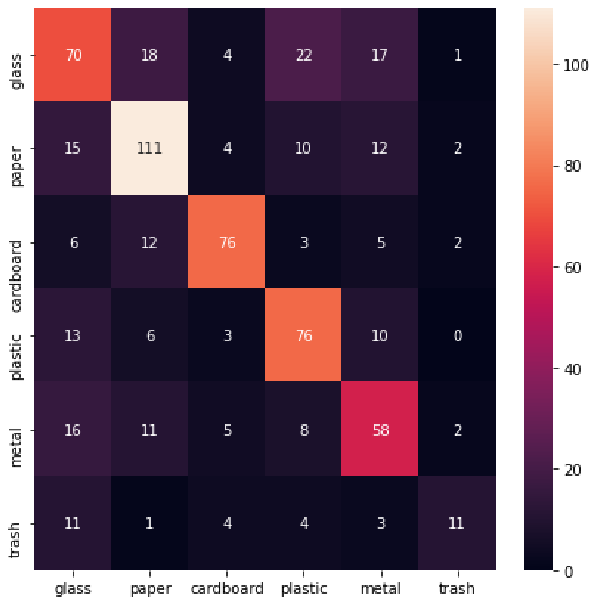


Figure 9: SVM One Vs One confusion matrix with rbf kernel

	Precision	Recall	F1-score
Glass	0.53	0.53	0.53
Paper	0.70	0.72	0.71
Cardboard	0.79	0.73	0.76
Plastic	0.62	0.70	0.66
Metal	0.55	0.58	0.57
Trash	0.61	0.32	0.42
Accuracy	0.64		

Table 3: Precision and Recall results for each class

One Vs Rest

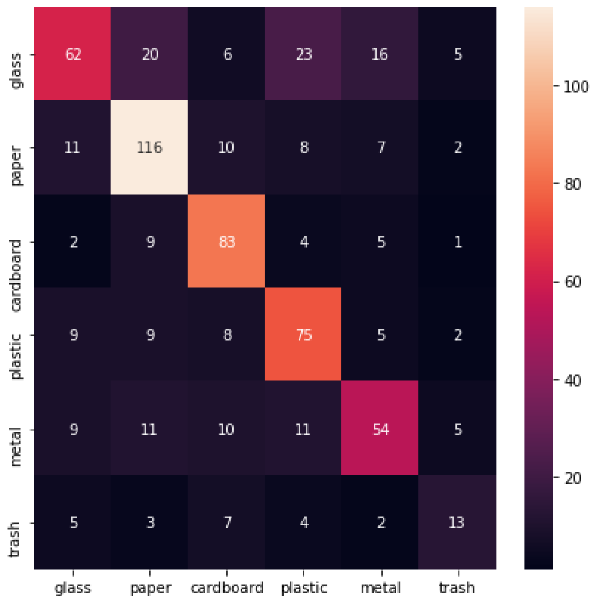


Figure 10: SVM One Vs rest confusion matrix

	Precision	**Recall**	**F1-score**
Glass	0.63	0.47	0.54
Paper	0.69	0.75	0.72
Cardboard	0.67	0.80	0.73
Plastic	0.60	0.69	0.64
Metal	0.61	0.54	0.57
Trash	0.46	0.38	0.42
Accuracy	0.64		

Table 4: Precision and Recall results for each class

5.4 CNN

We trained each network for 50 epochs with an Adam optimizer, passing the training images and augmenting them. To train our deep learning models we split the dataset in *train set*, *validation set* and *test set*. We use the train set to train our network while the validation set is used to adjust the hyperparameters. In the end we evaluate the resulting model on the test set.

We initially **froze** the base layers and train just the final ones. Then we **fine tuned** also the base layers with a lower learning rate, but we end up to have worst results from this last operation, so we obtmitted the results in this report.

To avoid a long training time and overfitting we also implement the **early stopping** technique using some keras callbacks functions like **keras.callbacks.ModelCheckpoint** and **keras.callbacks.EarlyStopping**.

Results for VGG19

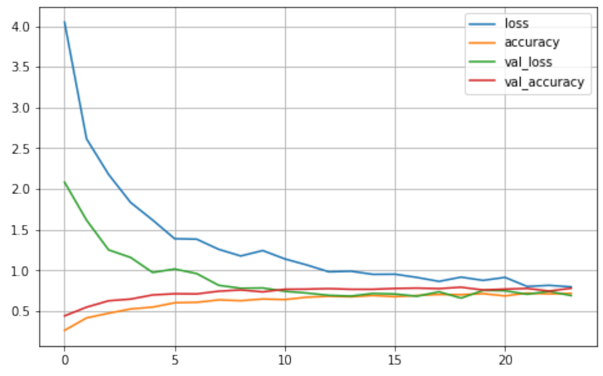


Figure 11: VGG19 training history

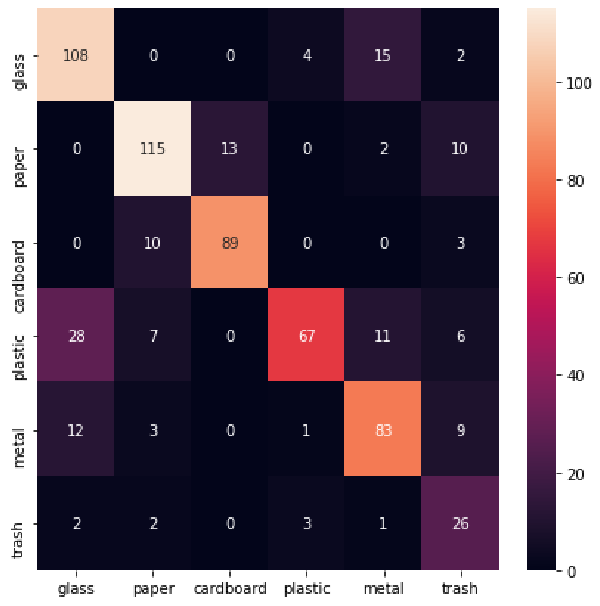


Figure 12: VGG19 confusion matrix

	Precision	Recall	F1-score
Glass	0.72	0.84	0.77
Paper	0.84	0.82	0.83
Cardboard	0.87	0.87	0.87
Plastic	0.89	0.56	0.69
Metal	0.74	0.77	0.75
Trash	0.46	0.76	0.58
Accuracy	0.77		

Table 5: Precision and Recall results for each class

Results for ResNet50

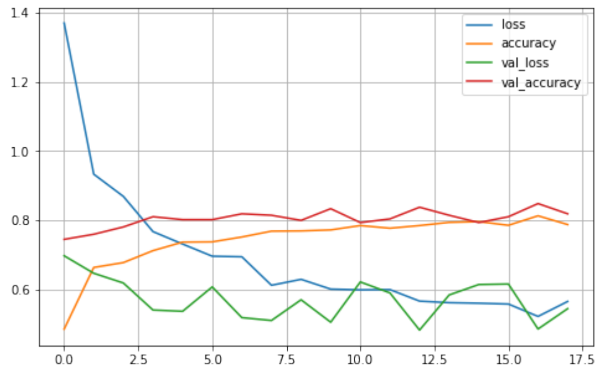
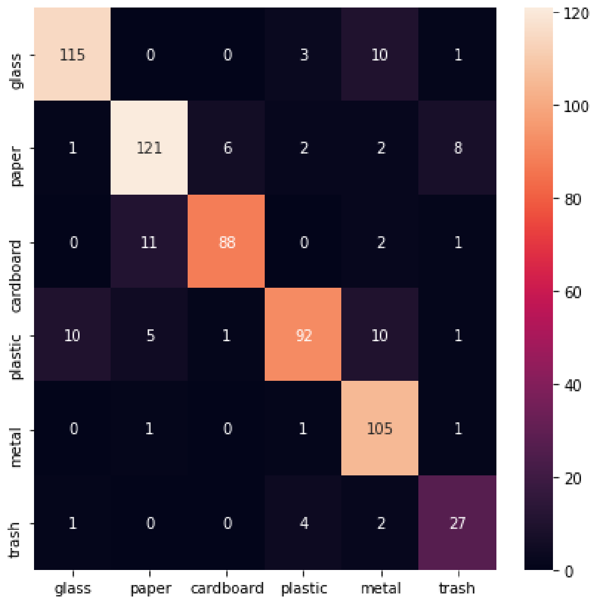


Figure 13: ResNet50 training history



	glass	paper	cardboard	plastic	metal	trash
glass	115	0	0	3	10	1
paper	1	121	6	2	2	8
cardboard	0	11	88	0	2	1
plastic	10	5	1	92	10	1
metal	0	1	0	1	105	1
trash	1	0	0	4	2	27

Figure 14: ResNet50 confusion matrix

	Precision	**Recall**	**F1-score**
Glass	0.91	0.89	0.90
Paper	0.88	0.86	0.87
Cardboard	0.93	0.86	0.89
Plastic	0.90	0.77	0.83
Metal	0.80	0.97	0.88
Trash	0.69	0.79	0.74
Accuracy	0.87		

Table 6: Precision and Recall results for each class

6 Conclusions

As expected we achieve better performances as we used more advanced classification methods, we achieved an accuracy of 0.87 using the ResNet architecture. Given these performances we can be reasonably satisfied of the work done.

References

[1] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.