FDS Report - Architectural styles CNN

Michael Cozzolino
cozzolino.1928667@studenti.uniroma1.it

Mattia Capparella
capparella.1746513@studenti.uniroma1.it

Federico Fontana
fontana.1744946@studenti.uniroma1.it

Abstract

The aim of this project consists in the implementation of an Architectural Style Classifier, improving the results published in Zhe Xu's paper, and disproving the results obtained by Marian Dimitru's solution posted on Kaggle. Our results show that data preparation generally improves the performance, but the choice of a deeper network might outperform alone even composite and more complex approach.

1 Problem overview

The main task is to distinguish an architectural style from another. The presence of mutual influences among styles, re-interpretation, revival and so on, can easily lead to low inter-class variations, making this kind of classification a particular one.

2 Related work

2.1 Original paper

Xu's solution consists in adopting a Deformable Part-based Models (DPM) to detect the position of a facade in the image while capturing the morphological characteristics of each style, sided by Multinomial Latent Logistic Regression (MLLR) as the learning algorithm to carry out the classification with probabilistic analysis. DPM models both global and local features enabling a flexible configuration of local patches introducing some deformation costs, while the introduction of MLLR enables soft assignment results and simultaneous training phases for all the classes classifiers. To tackle this task, neither a *ad hoc* pre-processing step nor the canonical augmentation techniques have been performed on the dataset.

2.2 Dimitru's variation

Dimitru proposes a slight paradigm shift: he adopts a *convolutional neural network* to perform the classification task. The original dataset is enlarged up to 10.000 images (5000 new images have been scraped automatically from Google). No custom models have been created: he experimented on both *resnet34* and *resnet50*, obtaining better results on the latter. Our first assumption of biased results due to a bad augmentation process turned out to be apparently wrong: there are some discrepancies between the description of what has been done and the code itself on Dumitru's GitHub page that let us think no strange input manipulation occurred and the boosted performance are a consequence of the dataset enlargement.

3 Proposed method explained

3.1 Preprocessing

This is the first contribution of our work: we introduce an offline preprocessing algorithm to refine training and test data. Even if the images gathered by Xu are in general good quality photos, yet they present some noise and other elements of disturb that could degrade both training and testing phase. The algorithm can be summarized in 4 main steps:

3.1.1 Image Denoising

We use a **Non-local Means** denoising algorithm with a dual purpose: first to achieve much greater post-filtering clarity while preserving more details in the image (wrt *local mean algorithm*) and second to reduce the color palette of similar hues: scanning wider portions of the image while computing the average values for similar pixels, we start the convergence of slightly different colors into a single one.

3.1.2 Conversion into HSV colorspace

We do convert the images into the HSV color space as a standard procedure in *image analysis*: when dealing with *image segmentation*, having a color space whose elements are as independent as possible is a great advantage, and separating a color in

hue, lightness and saturation is the most effective procedure. The benefit we derive from this operation comes from the fact the hue component of same or similar colors will be very close, regardless external effects such as bright lights, shadows, reflections and so on.

3.1.3 Color Extraction

Once defined the ranges of the hues to be removed, we can compute the masks of these colors and subtract them from the original image. What's left once this operation is done is an image in which the sky and greenery are *blacked out*. This approach is as naive as it seems, but has its own interesting advantages:

- It is straightforward to implement.
- It is fast enough to be used also for online learning and preprocess new test data once the model has been deployed.
- If original photo is exposed correctly, the blue sky and the greenery are removed optimally.

the downsides are:

- The absence of morphological analysis prevent the detection and the extraction of more complex elements, such as manmade object, human shapes and so on.
- It may happen that if a building has either many windows or it is a *glass building*, then the reflection of the sky on it may trick the algorithm causing the erosion of important portion of image, *i.e. loss of useful information*.
- It may happen that, even if not visible to a the naked eye, hues of colors to be removed are present in some area we would like to preserve.

3.1.4 K-means Clustering

To tackle the last downside, applying colors *k-means clusterization* and reducing the color palette to the dominant ones, generally improves the correct isolation of the parts we want to save from the ones that can be discarded (in this case: *blacked out*). Several experiments have been conducted to choose the best k value, and the results suggest that an interval from 6 to 12 seems to work fine; however, notice that the closer we get to the lower bound, the faster the algorithm will be.

3.2 Data Augmentation Online

This technique helps us to increase dataset variability and avoid overfitting. The transformations we are introducing are normal photo noises like rotation, cropping, changing in brightness, contrast, tonality. Our focus was in Erasing random zones on images, we notice helps the model to get better performance in the test dataset avoiding some local minima. For image data, online augmentation is motivated by observing that we can translate or add noise or otherwise distort an image but retain its key semantic content. The hypothesis of online augmentation is that the model probably will not see the exact same image twice, so memorization is unlikely, so the model will generalize well.

3.3 Building the model

Our model is Convolutional neural network (CNN) composed by 7 levels and one fully collected layer. Each level is composed by 5 times this structure: Convolutional layer, batch normalization and ReLu Activation. Between two levels there is a Max-Pooling layer, but in the final one we have an Average-Pooling layer.

4 Performance evaluation

4.1 Comparison with other algorithms

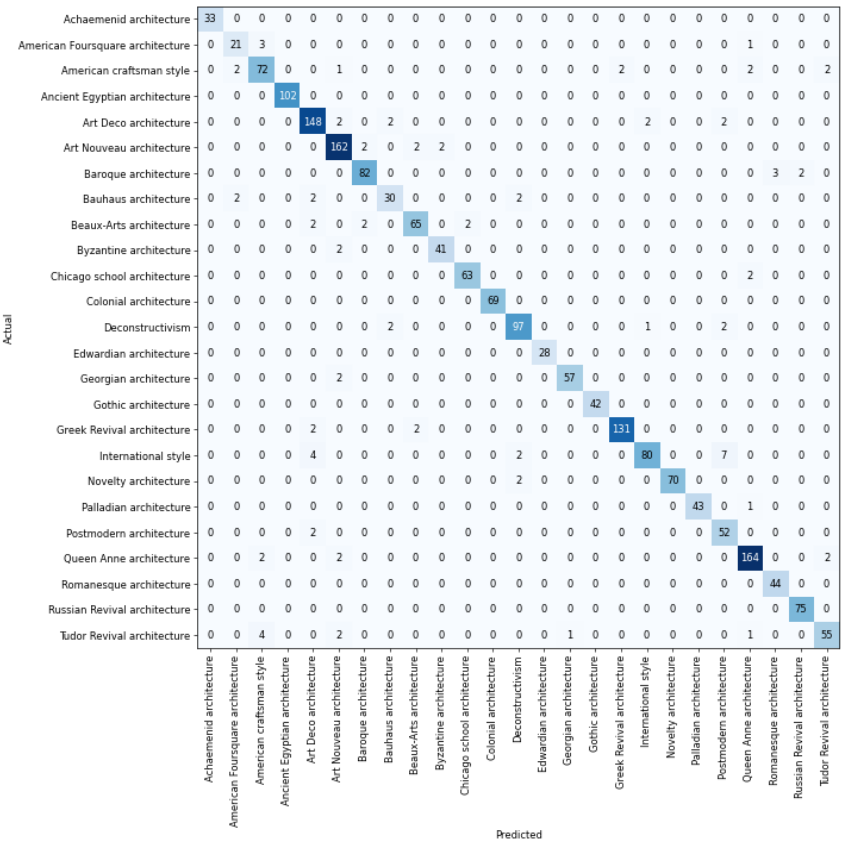
Table 1: Accuracy table

# classes	DPM-LSVM	MLLR+SP	Our Solution
10	65.67%	69.17%	-
25	37.69%	46.21%	63.43%

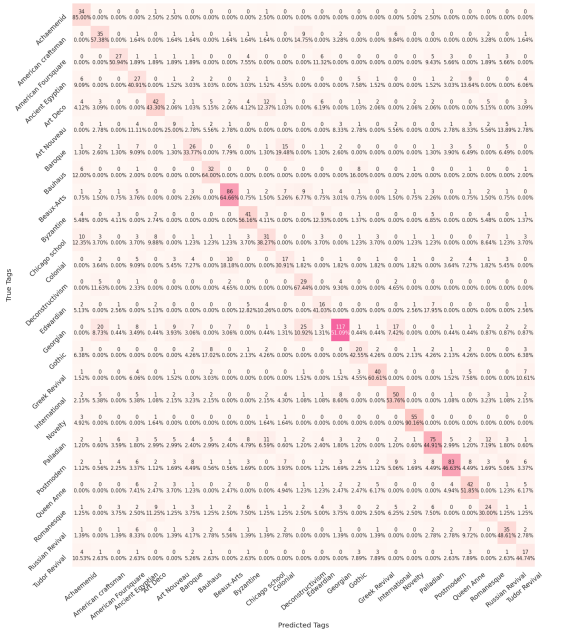
Performances obtained by using the Xu's original dataset

Our system outperforms all the others when using the original dataset and comparing the *accuracy*.

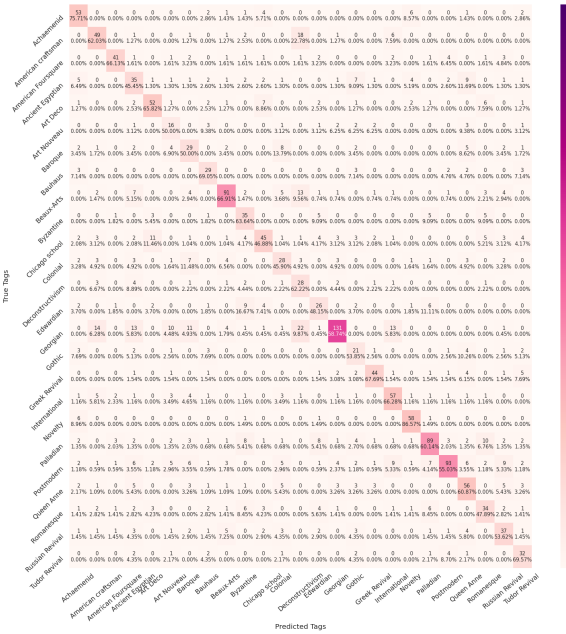
However, taking in consideration the kind of task we are dealing with and the different sizes of datasets involved, we believe that the *accuracy* is a poor choice when estimating the level of performance: computing the accuracy of an imbalanced problem such this one, the results are obviously skewed. For this reason, we have adopted the more (graphically) intuitive *Confusion Matrix*. Note that for every class, we have added the *recognition rate*: $\frac{\#correct\,identifications}{\#class\,instances\,in\,test\,set}$ to better understand how does the system perform when analyzing more images per class.



(a) Dumitru's result on enlarged dataset



(b) Our result on enlarged dataset



(c) Our result on enlarged pre-processed dataset

Figure 1: Comparing the confusion matrices

5 Final notes

At this stage, from **(a)**, **(b)** and **(c)** it is immediately visible that Dumitru's system does a better job in categorizing each architectural style. However, from **(b)** and **(c)**, we can see that better results are obtained when the algorithm described in Preprocessing is applied. As supposed by Dumitru when experimenting with different version of *resnet*, to choose of a deeper neural network can heavily influence the outcomes: more layers and parameters probably lead to better results. Further investigation and experimenting could easily take to even higher performances. A first step in this direction would be merging the two approach: prepare the data with a refined algorithm and train a deeper network.