Group Project

Giulia Amenduni Scalabrino, Isotta Magistrali, Luca Gandolfi and Gabriele Mura

May 29, 2024

1 Introduction

1.1 Motivation

The whole task of the project, i.e. the construction of a model which is able to correctly gather pieces, was inspired by a primordial and profound concept, well known several centuries ago. In Ancient Rome, despite the absence of the puzzle game, a similar idea was expressed through the word *symbolon*, which derives from the Greek word $\sigma\acute{\upsilon}\mu\beta o\lambda o\nu$, ($\sigma\acute{\upsilon}\nu$, 'together', and $\beta\alpha\lambda\lambda\omega$, 'put'). It was an object which was split into two halves (sometimes even more), so that two individuals that previously met and then separated could recognize each other by 'putting together' the two parts. In a broader sense, a symbolon is something that splits, but eventually assembles. It embodies a pre-existing order and unity, which was then perturbed and destroyed, but which can still be restored.
As a matter of fact, this is the starting point of the work. We have 'demolished' original images and we have trained two models which exploit different techniques, hoping that they could bring us back to the initial equilibrium.

1.2 The Dataset

The dataset (https://huggingface.co/datasets/huggan/wikiart) contains 81,444 pieces of visual art from various artists, taken from WikiArt.org, along with class labels for each image.

2 Structure

The initial question we posed to ourselves was: how do we, as human beings, would approach this game? If we had to do a jigsaw puzzle of this type, where each piece has the same quadratic shape, we would probably focus on two main things in order to analyse a single piece: *in primis*, the context, i.e. what's drawn inside the piece, the color used, etc, and *in secundis* the borders: it is known for sure that the borders of two neighboring pieces must match exactly, so this information will be largely employed to check whether two pieces are next to each other.
In this project we will follow these two approaches separately: firstly we construct a model which focuses only on borders, and reconstructs the picture starting from those, and secondly we construct a model which analyses the whole content of each piece of the image and reconstructs the picture from the results of the analysis. With a general approach, we thought of two possible procedures to solve the puzzle:

1. A two-stage approach: first, build a model which, for a given pair of pieces (and a given relative position of the two), is able to predict whether the two pieces are next to each other; second, build a model that, using the results obtained in the first part, reconstructs the image

2. A one-stage approach: build a model that directly, from the original image, reconstructs the correct image

It is natural that, when considering only borders, it is convenient to follow the two-stage approach. However, when considering the content of the image, the choice is not so obvious, as other choices could be modeled. We have decided to stick with the one-stage approach for the second scenario.

3 First model: ContourWatcher

3.1 Introduction

Let's follow the two-stage approach:

1. We build a model (in this case is a convolutional neural network), which, given two pieces, is able to predict whether the first is located to the left with respect the second. It can be also be employed for other relative positions up (or down), by rotating the pieces.

2. We build an algorithm that reconstructs the image using the information obtained in step 1. In this project we decided to use a greedy algorithm, despite several other choices could have been made (e.g. using simulated annealing or a neural network with a Sinkhorn layer, as described in section 4).

In this part we will resize the original pictures to 600x600, and our pieces will be of dimension 100x100, so that we will have in total 36 pieces to order.

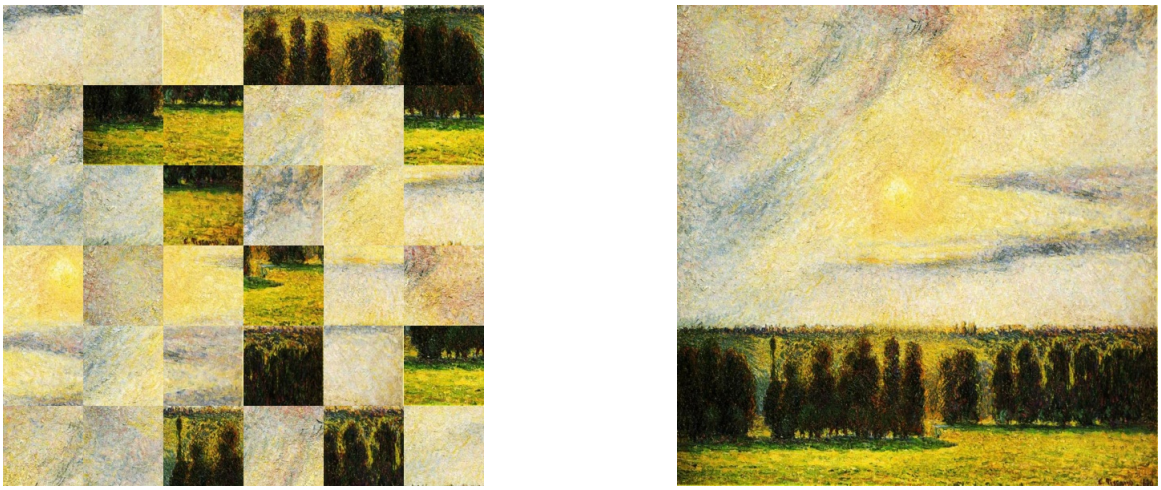

Figure 1: On the left, it is possible to observe the shuffled image, while on the right the one predicted by our model

3.2 The Convolutional neural network

The first step is to prepare the dataset, indeed we cannot use directly the dataset from WikiArt because we are not interested in whole images, but rather in pairs of pieces, one "glued" left-right to the other. For every picture in the WikiArt dataset, we decompose it in 36 little squares and, a priori we look at every possible combination of two pieces (considering also the relative positions). If in the original image the first is next to the second (with correct positions) we label it as +1, otherwise we label it as 0. We would a priori consider not only left-right, but also top-botton, and we could also reflect the image to have a symmetric dataset. Therefore, the number of possible entries in the dataset for a single image is really high, and, using the same amount of memory, we prefer to increase the number of original images, which we set to 1000, keeping a fraction of 0.3 of all possible horizontal combinations (but keeping all the correct ones); we thus end up with a training set composed by 400'000 pairs of blocks.

Since a priori we know that we want to consider only borders in our approach, we impose a very strong inductive bias. When we predict whether two blocks are next to each others, instead of considering the whole blocks and concatenating them, obtaining a dimension of 200x100, we maintain only the 3 pixels closest to the borders for each block, so that the final dimension of each data point is 6x100, made on the left side by the last 3 pixels of the first block, and on the right side, by the first 3 pixels of the second block.
The CNN, which takes as input the 6x100 image, is composed by two convolutional layers, the first one with 128 output channel and kernel size $(y, x) = (5, 6)$ (thus we don't convolve along the x

direction, with the idea that half of the parameters of each kernel are related exclusively to one block, and half exclusively to the other) and the second with 256 output channels and kernel size $(4, 1)$. As nonlinearity we use ReLu and as regularizer we use BatchNorm, which are just before the nonlinearities. Between the first and the second convolutional layers there is a Maxpool layer, while after the second convolutional layer, we use Average Polling, then we flatten, and then, the 256 outputs are fed to a 2-layer MLP, which has the hidden layer composed by 512 units, and clearly the last layer by 2 units. In the MLP we use BatchNorm once and Dropout twice, and as activation function a ReLU.
We trained our model on our dataset. We used ADAM first, and SGD with Nesterov Momentum later. We obtained an accuracy of 0.971 in the training set and of 0.960 in the test set (which is composed of pairs generated from other 300 images)

3.3 The Greedy algorithm

Now that we have a model which is able to predict whether two blocks are next to each others, we can proceed with reconstructing the image. We use a greedy algorithm, whose idea is to start from a piece and add one by one the correct pieces in the correct positions. Briefly, the algorithm starts with a block taken at random, it assigns a position $(x, y) = (0, 0)$ to it, and computes for every possible available block (all the other pieces for now) the score obtained from the CNN in all four directions. Then, it takes the block and the direction with maximum score, say the block is block i and the direction is on the right, assigns to i the position $(1, 0)$, since we moved by one on the right, it removes i from the list of available pieces, and goes on, in the same way, starting from block i, i.e. choosing a move starting from i. There are, however, some caveats which must be highlighted:

1. Not all directions are always available: for instance, there could already be a block in a particular position, and, therefore, we need to exclude those when looking for the next piece to attach. Moreover, since the final image must be composed of 6x6 blocks, once we have reached a width of 6 blocks, we have to make unavailable all the positions which would increase the total width, and the same is true for the y axis: once we have reached a height of 6, we cannot go up, or down, anymore, therefore we make unavailable all the positions which would increase the total height.

2. There could be situations where all directions inside the 6x6 square are unavailable, however, since we haven't reached the maximal width or length, we would be tempted to go beyond the border: a priori, indeed, we don't know the current position with respect to the full image. To avoid this issue we consider a threshold th: if the proposed move has a score less than th, then we reject the move, we take another block at random and we look for a new block to attach starting from it. However, if all the blocks with free positions next to them propose a move with a score less than th, then we take the move with the highest score among all those and we jump directly to the piece which proposed that move. th is a parameter, which was set to 0.9, that is, we want to accept a move only if we are very confident it is correct; however, a valid possibility is also to set it to 1.0, which basically makes the algorithm every time choose the move with highest score considering all the known blocks. Nevertheless, it slows down the procedure (even though it could be optimized) and it does not seem to lead to any improvement in the results.

Once every block has been allocated to a position, the algorithm stops, it translates the positions in order to have the upper-left corner at $(x, y) = (0, 0)$ and it returns the obtained order of blocks.

3.4 Final results: discussion and limitations

Testing the models on the first 200 paintings from which the test set was created, the algorithm correctly orders 179 paintings, obtaining an accuracy of 0.895. Examples are shown in Table 1.
Even though it is quite a simple model, it is able to order also complicated paintings.
However, there are some intrinsic limitations. First of all, the greedy algorithm is heavily dependent on the correctness of the CNN: as soon as the CNN is wrong and classifies an incorrect piece as the one with the highest score, then the algorithm fails miserably. This could be alleviated in two ways: one, considering not only the score that a proposed block obtains relative to only a single existing block, but rather doing an average of the scores of the proposed block with all the existing neighbours, possibly favouring when there are more than one neighbours. Two, letting the algorithm be able to

Figure 2: An extreme example where the algorithm fails due to the painting's frame. Left: correct image, Right: predicted one.

make changes when it recognizes it has made mistakes (however this would require larger modifications and the algorithm would not be greedy anymore).
A second limitation, which is relative to the fact that we are only looking at borders, arises when images have a frame. Indeed in these cases sometimes (actually, a small percentage of times) the algorithm fails because it couples two opposite borders of the frame. An extreme example is shown in figure 2. This is an intrinsic problem of considering only the borders of the pieces, even though it seems reasonable to think that it could be solved with a non-greedy algorithm.

4 Second Model: AlexNet vs Transformer

One of the fundamental assumptions of the first model we employed is that it focuses on the borders. Unfortunately, this is not always ideal. For instance, when humans solve puzzles they don't just look at the contour, but they try to locate the piece bearing in mind some sort of context. Among the many ideas that came to our mind, the one that we thought could be the best was to implement a transformer architecture. In theory, one could build an encoder able to embed patches and extract the context, hoping it would learn also some sort of proper positional embedding. Afterwards, the desired result would be obtained by using a decoder able to reconstruct the original image. In pratice, however, a potential obstacle would be that all transformers, meant to solve a similar tasks to ours, have been trained in the context of NLP (e.g. reorder words in a sentence or groups of sentences) and, on the other hand, all the pre-trained visual transformers we could find were trained on image classification, which means that the encoder took into account the position of patches in the order they were received (which, again, is not ideal for our goal). Moreover, since training a transformer from scratch would require too much computational complexity, we thought this would not be the best idea.
Going through the literature (mainly [1]), we decided to build the following model. We firstly preprocessed each image in the appropriate way. Then we permuted it following a permutation matrix (i.e. a double stochastic matrix which represents a sort of one-hot-encoded version of the permutation), which will also represent the label of our dataset. We used the pretained AlexNet up to the first fully connected layer (fc6) and applied it separately to all the patches. At this stage, the patches don't communicate between each other, which implies that our aim is to do some sort of feature extraction within the patch. Then, we concatenated the output and passed it to two fully connected layers. The last layer's output was 16, which also allowed us to apply a Sinkhorn Layer, an algorithm taken from optimal transportation, which produces continuous and differentiable relaxations of permutation matrices. In particular, it has the objective to transform our array into a 4x4 double stochastic matrix, i.e. a format comparable with the initial label. It then came the time to decide which loss to use, and this was one of the main challenges we encountered. The issue here is that we have to compare two

matrices that are doubly stochastic, with values summing up to n for an nxn matrix. We tried implementing different loss functions, here we cite the two most promising ones: the first one was computing the Frobenious norm between the two matrices, which consists in the euclidean norm of the vector of differences between all entries in the matrices. Then, we also tried computing the cross entropy on all n rows of the matrices, which can be interpreted as n different classification problems done at the same time. In this framework, we can see that each row represents the vector of probabilities of the positions of a patch inside the grid.
Unfortunately, the model does not perform good and, after few epochs, it actually looks like the loss starts going up (instead of decreasing). Another issue is that there were cases in which, when transforming the matrix of probabilities into an actual permutation taking the argmax, it is possible for one patch to be the most probable in more than one position (we displayed an example in the Jupyter notebook). We didn't try to solve the issue since a well functioning model shouldn't even present it, and we believe the issue might lay at the root of the model.
We tried different combinations of losses, number of epochs, training set sizes and model structures but we were not able to train a proper model following this approach. There could be various issues leading to this. It could be that the model was not big enough to extrapolate all the relevant information for properly classifying the images. It could also be that the training steps required to reach good scores is way too large for our capabilities and thus we didn't reach promising results. Another issue could be in the definition of our loss functions or even in the size of the MLP layers. One other approach could be changing the dataset format, even though we believe that giving the permutation matrix as a target label was the best choice we could make, as explained at the beginning. We believe that better results could be achieved with a more extensive training and more experimentation on the model parameters and structure. However, due to time constraints, we were not able to push our research far enough to reach our intended goal. One last comment is that, inspired by the literature, we decided to exploit one of the most famous CNNs, namely AlexNet, which, however, is now *old*. A possible implementation would be to use *younger* networks, such as VGG16 or VGG19, observing whether it would lead to some impovement.
A further try we made was to substitute the AlexNet part with a pretrained visual transformer (as discussed a few lines above), in particular using the encoder part without the MLP head: when passing a single patch into an encoder, it divided it further in smaller patches which should, indeed, be processesed in the same way an encoder would processes full images before doing the classification. We hoped to have a better embedding and/or feature extraction but, again, the improvement was very little. One venue for further research and implementation would be to try to implement the transformer described at the beginning.

5 Conclusion

Nowadays, research has been done on solving jigsaw puzzles, most of the times obtaining very bad accuracies (which did not overcome 60%) using these type of networks. As a matter of fact, this is not at all an easy task: when it comes to general pre-trained models the result seems to be worse compared to smaller (in size and depth) networks, which however were built for our specific purpose (Contour Watcher). In particular, the worst outcomes might also be due to the fact that AlexNet was trained on a large dataset of images, thus not specializing in art portraits (that be be extremely complicated). Nevertheless, the results we obtained are, in general, promising: with the right implementations and more sophisticated algorithms, we could, in principle, instruct the model to solve the puzzle in a 'human' way and, eventually, to lead us back to pre-existing equilibrium.

References

[1] Mehdi Noroozi and Paolo Favaro. *Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles*. Springer International Publishing, Cham, 2016.

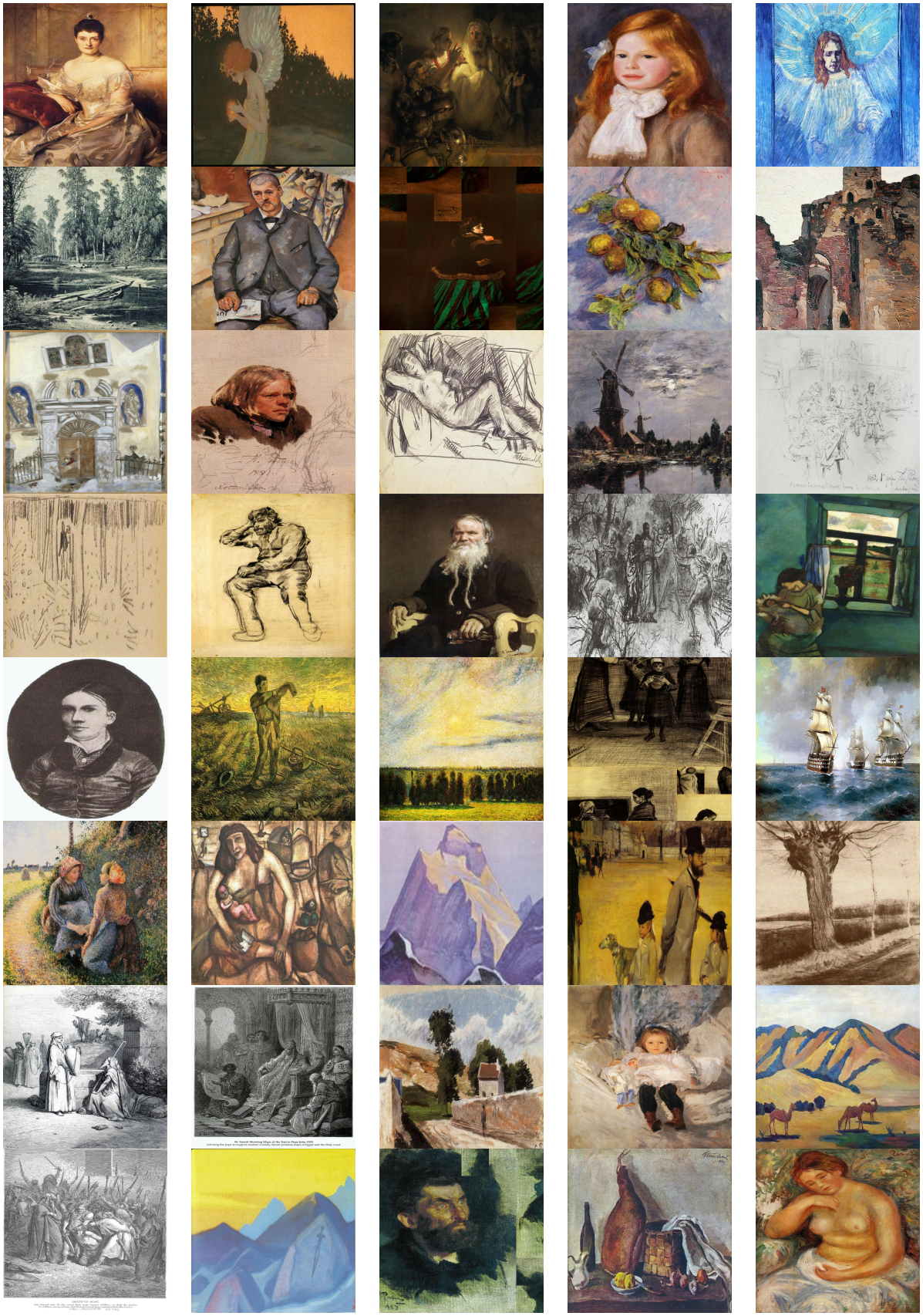

Table 1: Images reconstructed by ContourWatcher: out of 40 samples, 5 are reconstructed in the wrong way