Unsupervised Learning of visual representations

by solving Jigsaw Puzzles

SAPIENZA
UNIVERSITÀ DI ROMA

Capece Michela
Ferranti Flavia
Ferraro Jacopo

Experiment objectives and how to achieve them

The goal was to build a convolutional neural network that can be trained to solve Jigsaw puzzles as a pretext task (with no manual labeling) and then later repurposed to solve object classification.

This is done through a context-free network CFN that will learn a feature mapping of every object parts as well as their correct spatial arrangement.

This means that the CFN learned features can be reused for classification because they capture semantically relevant content and this allows to obtain high performance when transferred to classification task.

We talk about self-supervised learning because we exploited a freely available type of labeling obtained from the structure of the data that is the pixel arrangement, given by the Jigsaw puzzle reassembly problem.

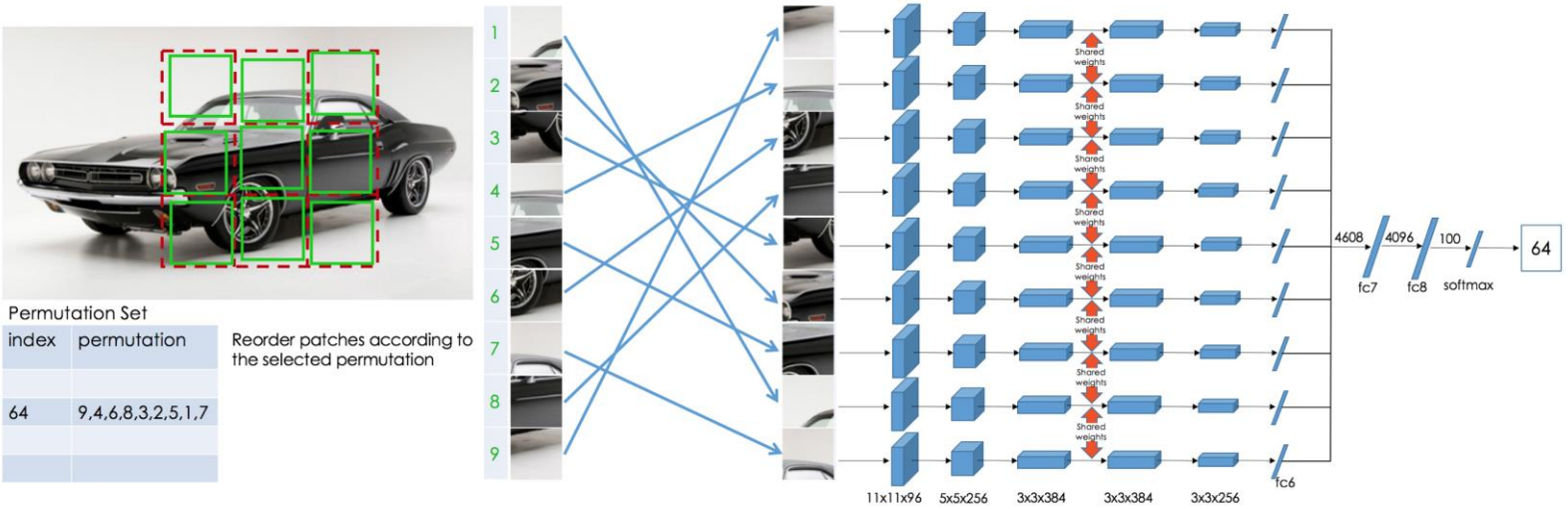
Once divided an image into parts, the jigsaw puzzle problem is solved by observing all the tiles composing the image at the same time. So the network first computes features based only on the pixels within each tile and then finds the parts arrangement by using these features.

The main idea is to force the network to learn features that are as representative and discriminative as possible of each object part for the purpose of determining their relative location, so we are not focusing on the similarities between the tiles but rather on their differences.

The realization of this work was done using python libraries such as *tensoflow*, *keras* and *numpy*

Implementation

How we arrived at a convolutional architecture capable of solving Jigsaw puzzles while learning general-purpose features



Firstly we built a siamese-ennead convolutional network where each row up to the first fully connected layer uses the AlexNet architecture with shared weights. The output of all fc6 layers are concatenated and given as input to fc7. All the layers in the rows share the same weights up to and including fc6. So the context is handled only in the last fully connected layers (context-free network). The idea is that we divide the image in different parts where each patch is given to a different AlexNet separately until the very last fully connected layer. The fc7 is responsible for the understanding the context of the objects and so the rearrangement of the tiles.
Let's see the structure more in details

CFN structure

For what concern the definition of the 9 AlexNet block the number of layers and the dimension of the kernels were strictly defined. This is why we left the construction as the one of the paper with all relu activation and kernel dimensions and parameters number as the one saw in the previous image. This configuration is similar to a regular AlexNet but with some differences for example a stride of 2 in the first layer.
After the definition of this model we created the function ***CFN*** that is the one that, after taking as input the AlexNet, it passes the tiles to the AlexNet block by recalling it 9 times, as the number of the tiles per image, and then concatenate the obtained output from each fc6. The concatenation can be now provided to the fc7 and we can proceed with the classifications. We created this configuration such that it could be easy to save and restore the weights for the next phase of the project.
At this point the model was ready to be compiled. For what concern the loss function we decided to use the categorical cross entropy while for the optimizers we used Adam.
At this point the only thing left was the training but what structure should the dataset have in order to make everything work?

Images manipulation

First of all we loaded a bunch of images directly from ImageNet and we saved them in a drive repository accessible through google colab. The images from ImageNet had all different size and resolution this is why while saving them in the directory we resize the shortest side to the image to 256 and we modify the other side so that it could be bigger than 256 and the the aspect ratio would be maintained.
Then through ***path_to_array*** we were able to extract each image and normalize it.
With ***square_image*** the images were cropped to a square shape of 225x225.
At this point they were ready to be splitted into 9 different pieces of size 75x75 and of each of this pieces we selected a tile inside of it of size 64x64. All of this was taken care by the function ***create_ordered_tiles***
At this point they were almost ready to be fed to the network, the only thing that missed was the label.

Permutations

To train the CFN we defined a set of Jigsaw puzzle permutations then, by choosing randomly in this set, we selected a possible configuration.
This configuration must determine how the tiles of an image are shuffled and so the selected permutation becomes the encoded **one_hot** label of the image. Once the CFN receive this dataset, the task is to predict the index of the chosen permutation.
The set of the possible permutations is not casual because we have to make sure that the tiles are shuffled as much as possible and we would like that the same tile would have to be assigned to multiple positions, ideally all 9.
So the permutation set is extremely important because controls the ambiguity of the task, in fact, if the permutations are close to each other the Jigsaw puzzle task is more challenging and ambiguous.
We created the set considering the Hamming distance that measures the number of substitutions required to change one permutation into the next, in other words it measures the edit distance between two sequences and so how dissimilar they are.
Among the various possibility we considered the average, the minimum and the maximum hamming distance, as we will see later on.
In order to pass to the alexnets the tiles correctly permutated it was used the function ***create_tiles*** that returned the shuffled tiles and the index of the permutation used

Avoiding shortcuts

In order to solve the Jigsaw Puzzle task the CFN can take some shortcuts. This kind of easy solution must be avoided because they can be considered suitable to solve the pre-text task but are not equally good for classification task. To avoid this, other than generate more than 1 Jigsaw puzzle per image (so that the network doesn't learn only the absolute position), we manipulated each image of the dataset through some function.
First of all we can find the normalization of the mean and the standard deviation of each tiles. This is done through ***normalize_tiles*** and is needed because adjacent patches have similar low-level statistics.
Then we needed to contrast chromatic aberration that is a relative spatial shift between color channels that increases from the images center to the borders. Since it helps to estimate the tile positions we eliminated this possibility in to different ways:

- the 30% of images were transformed into grayscale images through ***gray*** function
- the remaining colored images had the color channels spattialy jittered randomly by +-0, +-1, +-2 pixels

Finally, as said even before the 75x75 initial piece was randomly cropped to 64x64 so that the gap of about 22 pixels could help to avoid tiles positions thanks to edge continuity.

CFN dataset construction

We decided to work with an extremely small dataset to keep the computation fast, containing around 200 images. Of this images the 80% was used as training images and the remaining 20% as validation images.
To obtain this two dataset, used to train end test the CFN, we used the ***structured_dataset*** function**.**
The function take a *batch_size* number of images at once and start the computation.
First of all it creates the structure that will contain the labels and the images, that are respectively an array of *batch_size* images and an array containing 9 different arrays, one per alexnet, where there was allocated the space for containing in each of them *batch_size* images of size 64x64x3
After that, at every image of the batch were applied the functions seen before to arrive having 9 tiles per images free of shortcuts. This tiles were distributed on each of the 9 pre-allocated array before moving to the next image.
Then each label, containing the index of the permutation of the corresponding image, was encoded into a *one_hot* label composed of one and zeros.
At this point the datasets were ready to yield to the network one batch at the time.

Transfer Learning

Transfer learning is defined as the process of updating the weights that are obtained from the training of a task to solve another task. The procedure is the following:

- Freeze the layers that are supposed to be leveraged from the pre-trained model as is and no weights updates are expected on these layers
- Fine-tuning allowing to update the weights of some layers that will participate in the transfer-learning process

This means that we can take advantage of these learned feature maps without having to start from scratch, in fact, what we did was to evaluate our learned features as pre-trained weights for classification. This means that after training the CFN on the self-supervised learning task, we used the CFN weights to initialize all the conv layers of a standard AlexNet network and then we retrained the rest of the network from scratch for the object classification.

Labeled dataset

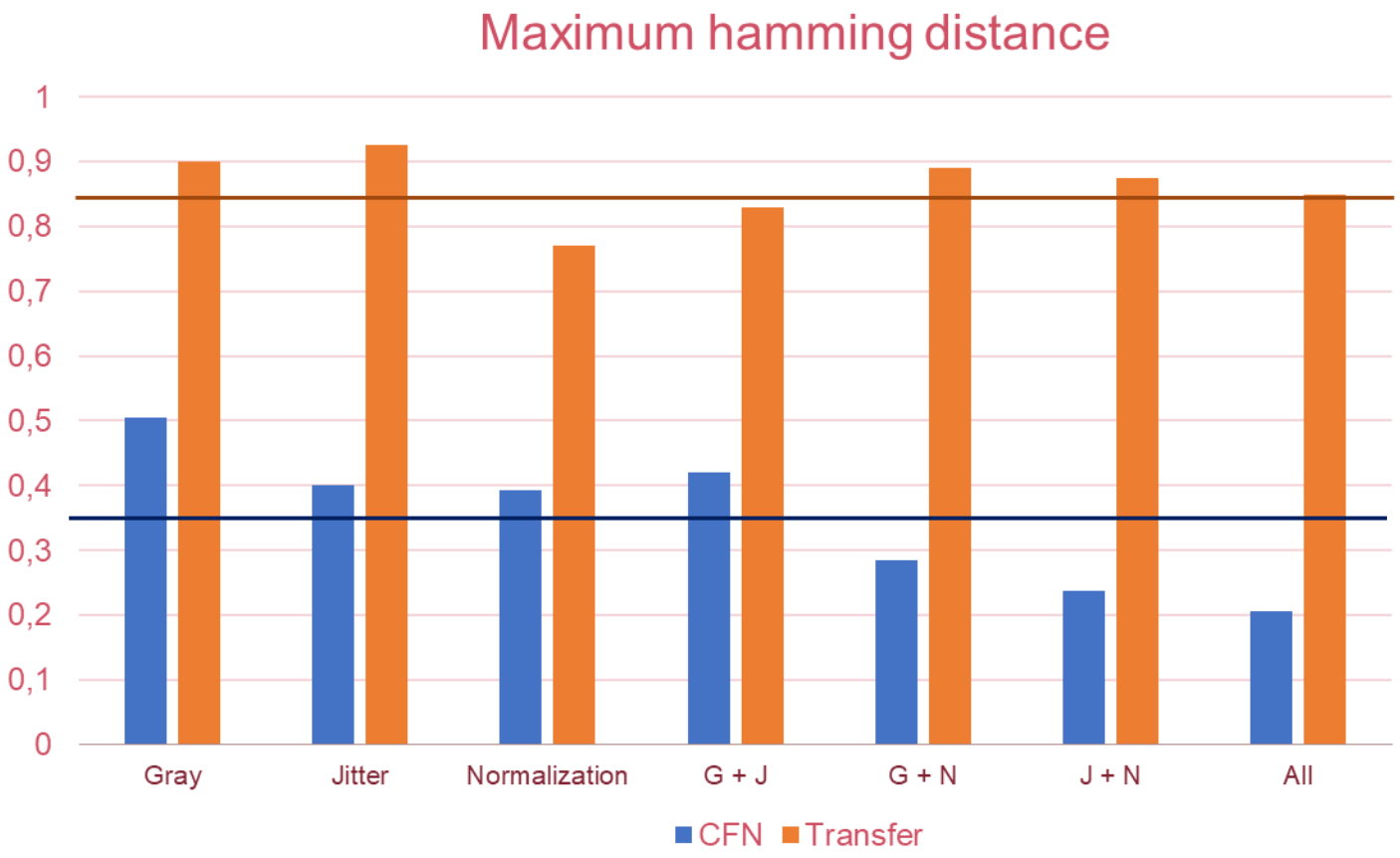
Here the images used are the same as before organized such that images belonging to the same class are placed in the directory named after the name of the class. This allows us to create a labeled data.
The dataset, training and validation, were created through the help of tensorflow API in particular *tf.data.dataset* and the split were done as before: 20% validation and 80% training.
The label of each image is taken by the name of the directory in which the image is placed and is encoded as an array of length equal to the number of the classes where the True value correspond to the right class.
Then for the training images is done a random crop to 224x224 and for the validation is directly taken the center of the image. No other data augmentation technique were applied because firstly we wanted to see the results.
At this point a shuffled, prefetched and batched dataset was ready for the classification task.

AlexNet

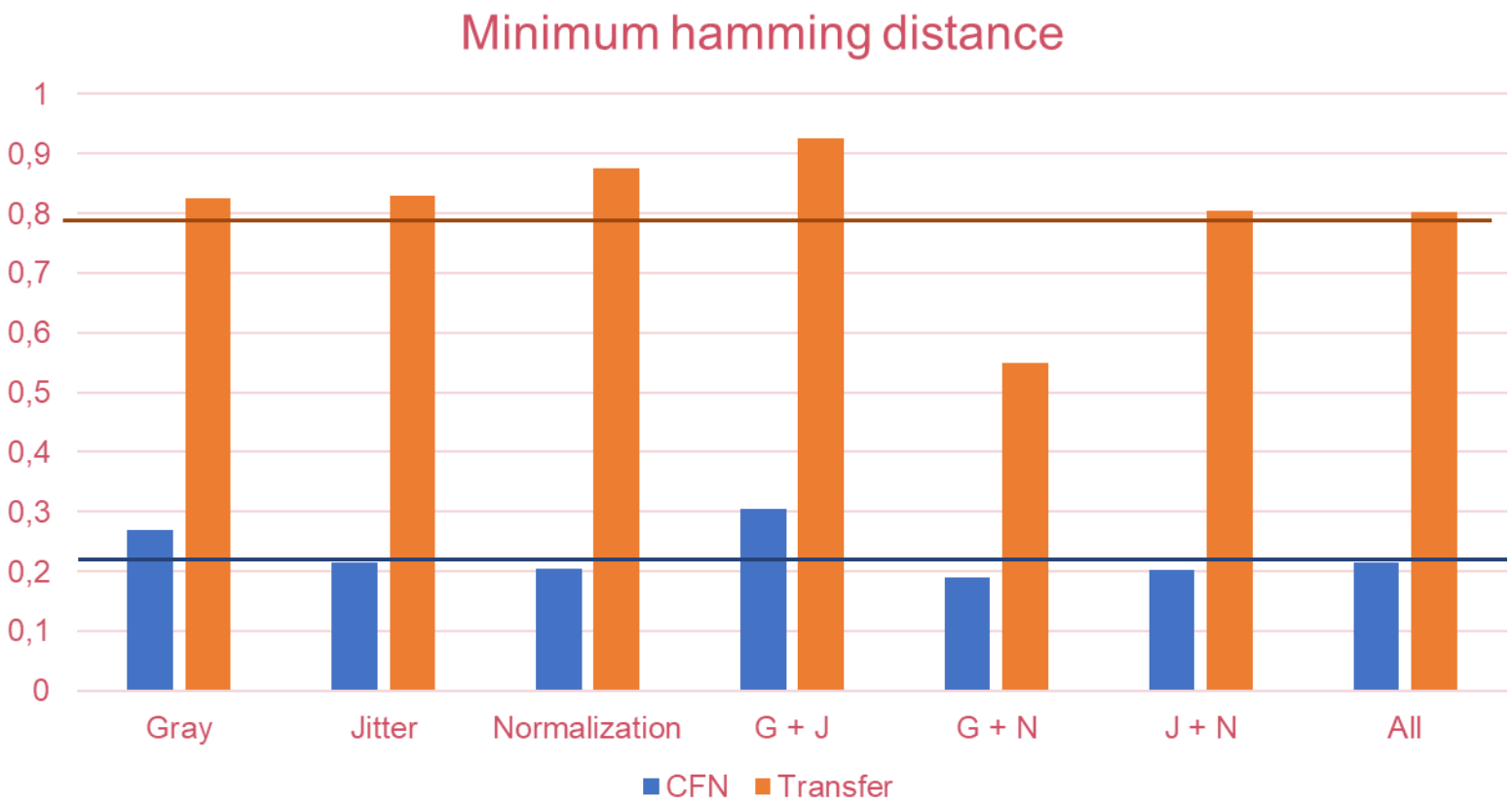
To proceed with transfer learning first we had to built our AlexNet and since Keras does not provide it we built that from scratch. Then, before starting the training , we got the weights, found at the end of the training of the CFN, and we took the one of the convolution layers to set them as the initialization weights for the real AlexNet conv layers while the rest were initialized with random weights.
The most important part was to understand which layer should be allowed to get updated during the training and which one should be left frozen, to allow a correct fine tuning of the Jigsaw task features.
We decided to allow the training of the layers below the last convolutional one included and the above were set as not trainable.
At this point everything was ready to allow us to evaluate the obtained results.

Results

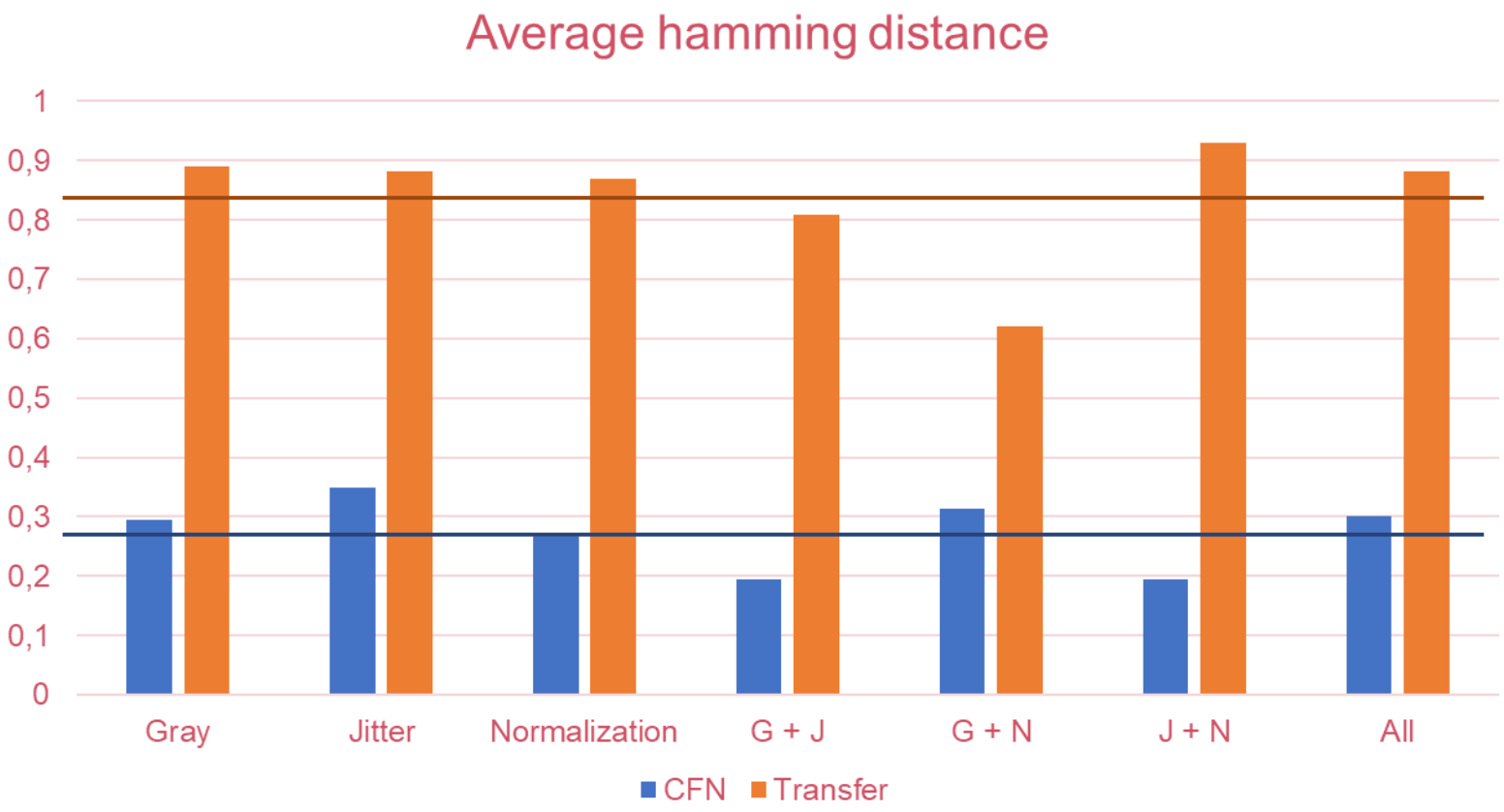
We performed different experiments on what we have discuss so far to see the impact of each component during the training of Jigsaw task. We trained the networks under different scenarios to evaluate the performance of the classification task on the ImageNet dataset.



More on the results



More on the results



Conclusion

Since the best results was provided with the average hamming set and a combination of jitter and normalization of the tiles we decided to study also its cardinality. And after this is clear that with less permutation the task with the CFN is easier but the transfer result are worst. The best are still the one with 10 type of different permutation. From this results and the one on the type of permutation we got that the best performing permutation set is a trade off between the number of permutations and how dissimilar they are.