Applied Machine Learning Challenge

Anna Maria Fetz, Luisa Lo Presti, Gioia Tessarolo, Daria Mochalkina

13 June 2021

1 Introduction

1.1 Task

The task of this project is to create an algorithm able to match the images in a query set with the images in a much larger dataset called gallery. Given some unlabelled training data, the algorithm will be able to extract its main features and compute the distance between the features of query and gallery images. The expected output is a dictionary having as key the query image and as associated value a list of the top n similar gallery pictures, thus, the images having the least distance.

1.2 Main ideas

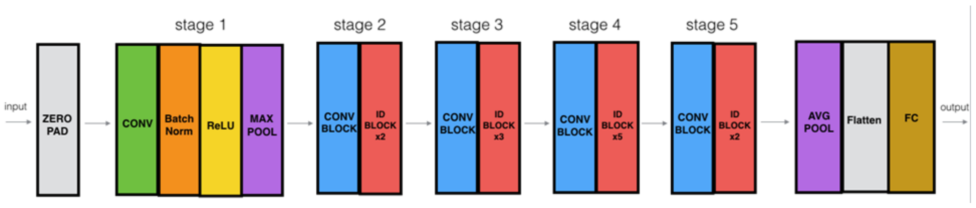
Different approaches have been used:

1. ResNet50: a pre-trained model has been chosen. The task has been accomplished using pre-saved weights.
2. Siamese Networks: contains two sub-networks used to generate feature vectors for each input. The vectors are then compared to evaluate their similarity.
3. Convolutional Autoencoder: grown fairly deep with some added Gaussian Noise layers to avoid over-fitting.
4. Variational Autoencoder: this was only a very broad idea. Since convolutional autoencoders assume not only a uniform probability on each point in the latent space, but also lack of dimensional construction, this approach could cause serious problems. Here, each image in a VAE is mapped to a multivariate normal distribution around a point on the latent space.
5. Convolutional Neural Network: we used this deep neural network because it is the most commonly used in image recognition. Simplifying its architecture, it consists of an input layer, hidden layers and an output layer.
6. Speeded-Up Robust Features (SURF): a descriptor that, dealing with the SIFT, has the advantage to be faster and the ability to recognise more key points.

1.2.1 ResNet 50 - Based Model

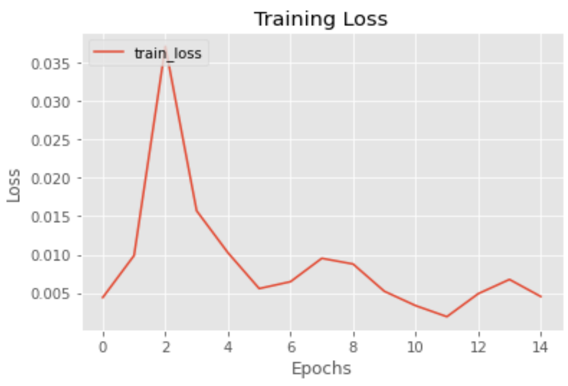
This model uses ResNet50, a pre-trained model implemented in Keras, which was trained over more than a million images from the ImageNet dataset. Thus, it has its own weights and it is able to classify objects in several different categories. ResNet50 is a convolutional neural network with 50 layers and accepts as input images of size of 224x224. In convolutional neutral networks (CNN), at least one layer, but usually more than one, uses a convolution instead of the general matrix multiplication, meaning that a kernel is used as filter: the kernel moves through the whole matrix, and each element within the kernel is multiplied by the corresponding element in the matrix. The results are summed and inserted in the output matrix. There are three types of layers: the convolutional layers described above, the activation layers, often using a ReLu or a Sigmoid function, and the pooling layers, used for sub-sampling to reduce the dimensions of the input. ResNet is a particular kind of CNN which uses batch normalization for better generalization and introduces *identity blocks*. These are a peculiarity of ResNet, used to decide whether some layers should be skipped for the overall performance to be improved. Indeed, identity blocks help easing the issue of *vanishing gradient* (issue we have been facing with our Autoencoder model), and ensure that the following layer will perform at least as good as the previous one.

Figure 1: ResNet's structure



ResNet and in general CNN are very popular choices when dealing with image recognition since they are able to capture the most important features of the images, granting great performances. Moreover, as we were using ImageNet weights, we took advantage of the fact that ResNet was already trained on a number of images that we could have never reached for computational and memory issues. Indeed, having a huge dataset is essential to improve the results and thanks to the pre-trained model we could solve this issue quite easily. Moreover, we added a few dense layers at the end of the ResNet50 and we made sure that the units value of the last dense layer were equal to the number of classes in the training set. We also used a different activation function, the softmax, in the last layer, as common practise. Finally, we fitted the model using a limited number of epochs given that already at the first iteration the model performed really well and there was no need to increment the epochs due to ResNet50 being already trained. Before applying this model, we used ImageDataGenerator to generate new augmented images from the original training set. Firstly, we actually explored all the possible attributes that the function could take and performed several modifications on the training images; then, before computing the fitted model, we actually used ImageDataGenerator inside a generators function, which flipped the images and also applied the ResNet50 pre-processing function to the training data. By doing so we introduced a bit of noises to the original training set to render the image matching more challenging. Unfortunately, due to limitation of the functions we used for fitting the model, we could not use a validation set to evaluate the error and the optimal number of epochs, so we decided to estimate the goodness of the model directly on the gallery-query matching, knowing the general accuracy of a ResNet50 model, as reported in the Keras documentation. Being it a pre-trained model, the validation step was somehow already executed for us. However, we produced a plot of the training process, as shown below.

Figure 2: ResNet50 loss during training



1.2.2 Siamese Network

This version of the code uses the so-called Siamese network. The main point of using a Siamese network is that it works well when having fewer data and it is robust to class imbalance, which was our starting point. It is often used in image recognition with the aim of finding similarities between the inputs and it takes advantage of convolutional layers as well as fully connected layers, although it might perform better when recognizing human features. This type of neural network works on distances: two different images are passed through the network, which extracts their features. Finally, the difference between the features is computed: similar images, meaning images representing the same object, must have distance tending to zero. Given its architecture, the Siamese network can be considered as an encoder. This model is computationally more expensive than classic neural networks, as it works with *pairs* (thus, it has a quadratic cost) and it does not output probabilities but *distances*. Therefore, for the purpose of finding the top-10 matching gallery images, we needed to select the ones with the smallest distance from the chosen query image. In our first implementation, we added some noises generated through a Gaussian distribution at the top of the network to improve generalization, followed by several convolutional and dense layers. The performances of this model did not seem to be optimal, thus we tried to change the Siamese network function by introducing the pre-trained ResNet50 at the top of the network, expanding it with dense layers and batch normalizations. For the activation function we chose to keep the Relu, except for the last layer where we used a Sigmoid function. Even though the query-gallery matching were not always successful, the final training and validation loss became quite low and they were both around 0.28, which is good since it implies that the difference in the two errors is minimal, suggesting the low overfitting achieved. During the training, we decided to take advantage of the EarlyStopping and ReduceLROnPlateau available in Keras, used respectively to stop the training and to reduce the learning rate when the validation loss stops improving significantly. This was done in order to try to minimize overfitting, which could have taken place if we set an arbitrary number of epochs. Indeed, the training of the model stopped at 80/100 iterations.

1.2.3 Convolutional Autoencoder

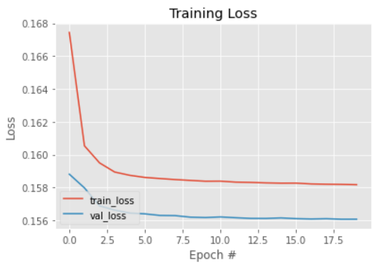
The proposed model, has been chosen for three main reasons:

- We decided to try a deeper decomposition and try to create a model able to reconstruct the whole input, thus forcing the encoder part to learn as much information as possible. The reason behind this thinking was to be able to understand to what extend the model was able to fragment and memorize images features to reconstruct them properly, which could not have been done without a decoder.
- It has been proven to be a successful for lossy compression and noise removal (see the link:Performance Comparison of Convolutional AutoEncoders, Generative Adversarial Networks and Super-Resolution for Image Compression).
- It succeeds in linking three distorted images concerning the same object to convey a unique picture.

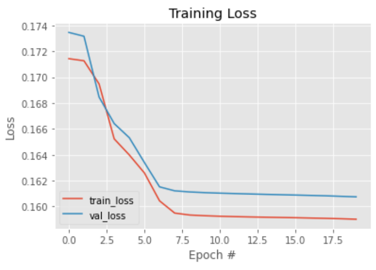
There were two main approaches while training and implementing this model. However, the common structure of the encoder remained the same: the input layer and the subsequent outputs are fed to alternating Convolutional and Max-Pooling Layers. At some randomly chosen index, we decided to implement a Gaussian noise layer, to artificially augment data (mainly due to the fact that feeding augmented datasets required huge memory, which we could not have at our disposal). Firstly, there were three main Conv2D layers (with a ReLu activation function, to obtain good granularity), three MaxPooling layers and two Gaussian Noise layers intended to extract the features grid out of the input image and to avoid great loss of information. Filtering has been applied with 32, 64 and 128 filters firstly. However, the model seemed not only to be overfitting (Figure 2), but also to be ”stuck” in a local minima, while not learning as much as expected (the vanishing gradient descent mentioned above). Therefore, our approach has been threefold:

1. The training dataset was increased up to 2000 pictures, as there were still few pictures per class;
2. The model was simplified: instead of three convolutional layers, we set two using 32 and 64 filters, shifting of 2 pixels at a time;
3. The starting learning rate was reduced from 10e-3 to 10e-6.

The Max-Pooling layer is added to shrink images size, and has been preferred over the Average Pooling due to the fact that it extracts the most important features like edges better, whereas, average pooling extracts features more slowly and smoothly. For what concerns the decoder, we decided to insert fewer layers, as it would have been detached anyway after some training of the whole model. The main reason for using the decoder was to understand how well the model was able to reconstruct the starting picture, to eventually modify some layers while building the encoder. Nevertheless, we chose to keep the decoder as symmetric as possible to the encoder, although no real advantage has been proved. Moreover, we intended to separate the encoder and the decoder model mainly because of the main purpose of the given task, and most importantly to be able to eventually try different loss functions (one specific for the encoder, and an overall reconstruction loss).



(a) Overfitting Model, three convolutions, higher Adam



(b) Better Model, two convolutions, lower Adam

There have been implemented a first fully connected layer, and immediately after a reshaping layer, whose dimensions have been willingly left arbitrary and filter-dependent in order to be able to try different model-depths with different filters, without having to change reshaping dimensions at every iteration. Next, there are three Conv2DTranspose layers and an additional one having a different activation function. This latter layer uses a different activation function, namely the softmax function instead of the Relu, after having noticed an increased performance when switching between these. We chose the softmax activaton function over the Sigmoid one for two main reasons:

- The softmax function is build like this:

$$softmax(z_j) = \frac{e^(z_j)}{\sum_{k=1} e^(z_k)} forj = 1, ..., K$$

  which is very similar to the Sigmoid. However, in the denominator, we sum together **all** of the values: when calculating the value of softmax on a single raw output, we have to take into account all the output data. Therefore, it ensures that the sum of all our output probabilities is equal to one.

- Output are related to each other, given the sum at the denominator. Therefore, if the probability of one class increases, it has to decrease for a different class.

Additionally, we decided to save weights for both the encoder and the decoder throughout every accomplished iteration: the model has been trained starting with zero information, and therefore we needed to build some.

We decided to implement EarlyStopping over the training loss in this model as well, in order to prevent overfitting and avoid useless iterations.

1.2.4 Variational Autoencoder

One of the main issues linked to ”simple” Convolutional Autoencoders is assuming a generalized probability distribution for each point within the latent space. Therefore, we decided to implement a (simpler) Variational Autoencoder, which provides a probabilistic manner for describing an observation in latent space. This could be proved to be useful mainly for ”latent” attributes one picture may have (especially the distractors), because they are now described with a proper probability distribution. The encoder does not output a single value describing the latent space of the attribute, but it describes the probability distribution for each point in the latent space. We tried to implement the so called ”re-parametrization trick” to allow random nodes to be passed as inputs to the decoder. Moreover, within the latent space we are constraining our points to fall within a defined probability distribution, which limits the issues we have depicted earlier. Our main goal was to try to understand the impact of a robust distribution for each point in the latent space, compared to a more generalized and soft assumption of the standard CAE. Thus, we wanted to try analysing the trade-off between a more complex convolutional autoencoder, and a simpler VAE, to understand which ground-hypothesis would suit better the task. The following model was built over four main convolutions, starting with a 32 filter convolution and three 64-filter convolutions. No further layers have been added before the fully connected layer in order to understand whether the main underlying assumption of a non-generic probability distribution per point within the latent space could have been more successful than a well-trained Convolutional Autoencoder. The decoder has been built even more synthetically by introducing one transpose convolutional layer and one normal convolution to get a feature map of the same size as the original input.

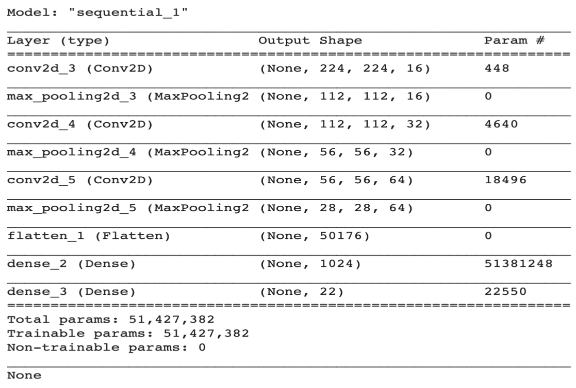
The main and most relevant difference between the two models is the built-in custom variational layer, which corresponds to the above-mentioned re-parametrization. This layer is called on the input and the decoded output and computes the loss between the estimated probability distribution and the expected one. This latter loss function is based on two main terms:

1. A reconstruction error penalizer;
2. A newly-learned distribution enhancer, for the distribution to be similar to the ground truth one.

Moreover, the optimizer was used to contrast the typical oscillations given by the VAE training. Unfortunately, this autoencoder would have required a way larger dataset than the one used to train the convolutional autoencoder. Moreover, we noticed training to be highly unstable (probably due to the little size of available pictures) and decided to not use it during our challenge.

1.2.5 Convolutional Neural Network

This neural network is characterised by the presence of the optimizer: we choose to use the Adam function for its large number of benefits: it has little memory requirements, it is computationally efficient and appropriate for problems with very noisy and/or sparse gradients. It is used instead of the classical stochastic gradient descent as it update network weights more efficiently. In order to be functional, we used the most common pattern implemented in tensorflow: some Conv2D and MaxPooling2D layers. Conv2D is a layer that creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. MaxPooling2D is used for max pooling operation for 2D spatial data. We chose Max Pooling over the Average one for the same reasoning as above. As input CNN takes tensors of shape. Dense layers take vectors as input (which are 1D), while the current output is a 3D tensor. First, one will flatten (or unroll) the 3D output to 1D, then add one or more Dense layers on top.



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_3 (Conv2D)            (None, 224, 224, 16)      448       
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 112, 112, 16)      0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 112, 112, 32)      4640      
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 56, 56, 32)        0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 56, 56, 64)        18496     
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 28, 28, 64)        0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 50176)             0         
_________________________________________________________________
dense_2 (Dense)              (None, 1024)              51381248  
_________________________________________________________________
dense_3 (Dense)              (None, 22)                22550     
=================================================================
Total params: 51,427,382
Trainable params: 51,427,382
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 3: Summary of the model

1.2.6 Speeded Up Robust Features

We choose this descriptor because:

- it is computationally more efficient
- the descriptor size is reduced from 128 to 64
- It brings faster training and recognition.
- Compared to the SIFT, it also includes larger surrounding of the key-point for the descriptor creation. SURF features are thus less local than the SIFT ones.

With a similar architecture to the more famous SIFT. It matches only key-points inside corresponding regions in the compared images which also reduces the computational coasts, then it is calculated the difference between this distance and the distance of the second nearest key-point.

2 Methodology

2.1 Training dataset

Firstly, one main consideration has to be made regarding the overall approach with the training dataset, since all proposed models were trained from the very beginning, besides the one based on ResNet50. We not only decided to enhance every sub-folder of the given starting set, and to add some more classes, such as McDonalds buildings, Clothes, animals etc. for the model to learn as much as possible, but we trained the Autoencoder and the Siamese Network model on very different datasets as well, using *transfer learning.* Additionally:

- The **CAE** model has been trained on both "pure" and "distorted" images, thanks to an additional Gaussian noise with 0.3 standard deviation (which has been randomly chosen).
- Both the **Siamese Network** model and **ResNet50** have been dealing with original and augmented images for features extraction.

The main reason this was put into practise is to force our models to learn to extract the most meaningful features and to leave noises behind. To conclude, we decided to both enhance the number of pictures in our training data, and to introduce some random noise within each class. In general, at first, models were implemented over the given training dataset in order to understand how to build the general frame. Next, we introduced more than 5 augmented pictures to each class, to increase the learning difficulty for the model. After having saved those weights, we shifted the learning over a much different, broader dataset of 2000 pictures circa. This dataset included pokémons, clothes, cats, dogs, boats, and buildings. After having trained our models over this newly-created dataset, we decided to enlarge it more, reaching 6000 pictures circa, with more or less 60 images per class.

It has to be highlighted, that during most of our training we chose to keep an 80%-20% split, and sometimes 70%-30% split, to understand the extent of model's goodness and to compute a partial validation loss.

2.2 Feature extraction and matching methodologies

In all our models, to match query and gallery images, we defined a feature extractor, and the distance between features has been computed to assess whether a gallery picture could be a match for the query one or not. However, this process has been led differently on the different models proposed.

2.2.1 ResNet50

The feature extractor function firstly pre-processes the images given as input and then applies the model to make predictions. In other words, after the pre-processing of the images is completed, the model can be applied to extract the features. The features extractor takes as input a list of images and the defined model and returns a list of normalized features. Then, we defined a function to match the images based on the Euclidean distance between the features. This function takes as input a query image and returns the top five matching pictures in the gallery, together with a score and the accuracy of the matching.

2.2.2 Siamese Network

Given its architecture, the Siamese network can be considered as an encoder (and feature extractor). We chose to implement, instead of the triplet loss, a contrastive loss, which is based on the distance between the true label/class and the predicted one, considering a margin which specify a tolerable distance between similar images, or, in other words, a minimum distance that needs to be found to consider two images different. The triplet loss would be an extension of the contrastive loss, which would consider not two but three images: an anchor, a positive image, which would represent the same object of the anchor, and a negative image, representing a different object. The aim of the triplet loss is to make the distance between the anchor and the positive smaller than the distance between the anchor and the negative. However, we preferred to implement the contrastive loss not only for computational costs, but also for the main goal of the task being identifying places instead of the typical use for facial recognition. Once the Siamese model and the loss were defined, we implemented a function able to produce pairs of images taken from the training set, either composed by different or similar images. These pairs were used to train the model to recognize that similar images must have small distance. In this version of the code, we also implemented a function to save the computed model through checkpoints and to restore it for later use.

2.2.3 Convolutional Autoencoder

The chosen loss function for this model is the MSE, as:

- we wanted to minimize the difference between the decoded image and the ground truth one, meaning that the Autoencoder has been able to learn the relevant features well;
- we assumed our target is continuous and normally distributed, and want to maximize the likelihood of the output of the net under this assumption.

However, it has to be taken into account that MSE does not punish misclassifications enough. Instead, it is the right loss for regression, where the distance between two possibly predicted values is small, which is our case when passing *encoded features*. One further step we could have implemented having more time is the fine-tune of the Sparse Cross Entropy loss function. To add more variability, "noisy" and "pure" training data has been shuffled randomly. Moreover, training after training, the weights for both the encoder and the decoder have been saved, and the first encoder layers were re-trained by anchoring to each layer the respective previously-saved ones. The following step was implementing a feature-extractor function, in which encoded features are extracted for both query and gallery images. We computed distances among query and gallery features through a simpler matrix/vector norm.

2.2.4 Convolutional Neural Network

In contrast with what we have tried on the Convolutional Autoencoder, this time we have tried implementing our CNN over the categorical cross-entropy loss function, as the model presents multiple classes.

2.2.5 Speeded Up Robust Features

With a similar architecture and performance to the one presented in class. You will find keypoints and descriptor and with a brute force feature matching it will find the top3 more accurate images (with more similar features) to the query one.

3 Results

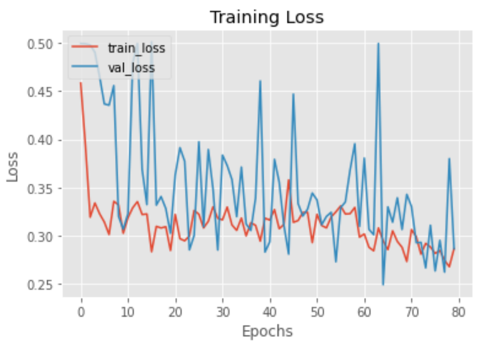
Having many different models, we tried to use them as a playground for both learning rate, epochs, and batch-size fine tuning.

For what concerns the ResNet50 based-model, we observed that it was harder for the model to match images after the introduction of the augmentation of the data. However, it has to be noticed that, being it a pre-trained model, we did not save the model after each run because we used the already computed Imagenet weights. This choice could have affected the overall performance, but it was still a justified choice since having a huge dataset, such as the Imagenet dataset, for training the weights is one of the most important part when building this kind of models. Indeed, the model always performed well and kept a good training accuracy. As mentioned in the Keras documentation, ResNet50 has an accuracy around 0.75 if asked to find the top-1 matching image, but it reaches 0.92 when considering the top-5. This found some sort of confirmation in our results since the matching were correct and led to the following accuracy values:

accuracy is: {'top1': 0.5333333333333333, 'top3': 0.7333333333333333, 'top10': 0.8666666666666667}

On the other hand, the Siamese network did not perform as expected. We decided to plot the loss on the training and validation set at the variation of the number of epochs to check the performance of our model: the model still struggled to learn the correct features and to reduce the loss, thus its performances were not great, as depicted below.

Figure 4: Siamese Network performance with contrastive loss



Concerning the Convolutional Autoencoder model, training results were surprising. With regards to the training phase, the first issue we faced was to find a reliable way to save and re-train our model. Due to the way in which the Convolutional Autoencoder has been built, we encountered some difficulties in loading the whole model again. Therefore, we decided to train on different datasets the whole autoencoder, to save the weights (encoder-decoder separated), and to re-train the encoder only later on (after all, the decoder part is not needed for the purposes of the task). By varying our training dataset, simplifying some layers, and decreasing the starting learning rate for the Adam Optimizer, the immediate results were quite satisfactory: the model stopped overfitting and started to learn, reaching its highest accuracy at 68% with 50 epochs and a batch size of 16.

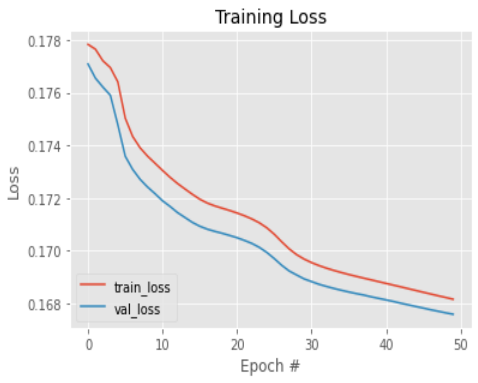


Figure 5: Final Autoencoder model's performance

However, when applying these results on a different training set, we faced an overfitting issue, and the model did not seem to learn at all. Therefore, we decided to intervene on the batch size, the number of epochs, and the learning rate. An accuracy of 53% was reached. Despite the low rate, and the distractor-sensitivity, we consider this a good result for two main reasons:

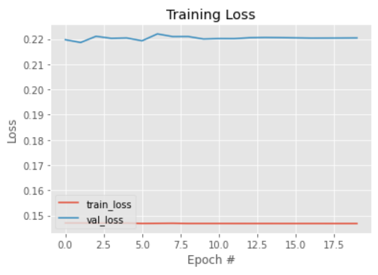
1. Firstly, despite the large dataset, the size of each sub-folder contained 100 pictures maximum, which is still to be considered low. When enhancing our dataset, accuracy increased (on average) from 55% to 65%, remaining close to 53% when switching among datasets.

2. Due to computational costs, we could not run the model a greater number of times, on even larger and more differentiated datasets.

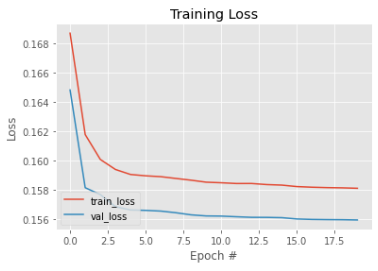
After the test phase, we soon realized three main problems our model had:

1. Firstly, we kept training the model over the same dataset with very low and very different learning rates for too long. Therefore, the autoencoder had slowly learned some specific features of that dataset (and similar ones), but when faced with different pictures, it was not able to learn anything.

2. Secondly, gradient descent was too low, together with batch size. When re-parametrizing the Adam optimizer (still on the basis of pre-saved weights), increasing batch size and lowering the number of epochs to 20 and then 10, the model started really learning something without getting stuck into some irrelevant local minima.



(a) Not converging model



(b) Starting to learn, but still overfitting

3. Increasing the number of filters to [32, 64, 128, 256] was not proved to be successful, especially on a noisy training dataset. Therefore we avoided implementing it, to avoid overfitting when re-training the model multiple times.

Although the model had been trained on 600, 2000 and 6000 images (with very different classes, ranging from clothes, to pokémons, to famous buildings), we noticed that it wasn't performing well during the test phase, barely reaching 0.05% on the top1, top3 and top10. Hence, whilst keeping in mind that many more pictures for training the model in a suitable way for the challenge would have been needed, we decided to create an additional Convolutional layer in the encoder, over 128 filters, (which we previously erased), and to increase the starting learning rate consistently (from 10e-6 to 10e-2). This proved to be partially successful (although there was still high overfitting), mainly given the fact that there were already many saved weights our group could benefit from.

This intuition helped us to boost performances a bit and obtain the following results:

accuracy is: {'top1': 0.06666666666666667, 'top3': 0.06666666666666667, 'top10': 0.3333333333333333}

One further step that could have been accomplished was not to increase the number of epochs, (we decided to set it to 20 due to the amount of ties we have been training the model to avoid overfitting), and increase the batch size from 16 to 32 (under 32 training got very slow due to significantly lower computational speed,not exploiting vectorization to the full extent).

Talking about the CNN considering the fact that it is the most used algorithm it gave us some problem, especially in term of time; the accuracy of the model was pretty low: around 40%. We didn't trust this model too much because it gave everytime very different accurancy.

Regarding the SURF model we obtained the following results:

accuracy is:{'top1': 0.557, 'top3': 0.743, 'top10': 0.871}

Which can be considered a very good outcome, when considering the number of images it has been trained on.

4 Conclusion

As expected, the best performing model has been ResNet50, succeeding around 53% of times for the top-1, 73% at top-3, and 86% at top-10. Unluckily, we did not manage to use it during the challenge since we decided to test first our manually-built models, namely the Autoencoder, the Siamese Network and the CNN, since they were the result of our experimentation. The outcome was predictable because ResNet50 is a very deep network, which has been already trained over more than one million images. Therefore, the correct weights were already saved and the remaining main task to accomplish was to choose a measure of similarity (in our case the Euclidean distance) and compute the distance among features. Surprisingly, we obtained very good results with the SURF model as well (almost as much competitive). Thus, we consider this to be a very good outcome, although it might be less solid than Resnet50. However, SURF seems to work quite well in detecting specific features, way faster than the SIFT one. A better performance would have been expected by both the CNN and the Convolutional Autoencoder. However, with further training on larger datasets, the CAE model could have still provided good results. Unfortunatly, as pointed out before even the Convolutional Neural Network gave a not so good accuracy and the train was really hard and big amount of time, maybe the model could best fit the dataset. For what concerns the Siamese network, the training was quite hard and it seems that the model did not learn the right weights; it may be that this kind of network was not the best choice for our task and that it would have needed way more images than the ones in our training set. It is likely that it would have worked better with simpler images, such as letters or graphic signs.

5 Contributions

Member's Name	Model	Tasks
Luisa Lo Presti	ResNet50	Research, implementation,fine-tuning,training
	Siamese Network	Research, implementation,fine-tuning, training
Anna Fetz	Convolutional Autoencoder	Research, implementation, fine-tuning, training
	Variational Autoencoder	Implementation, theoretical research
	Siamese Network	Research, implementation
Gioia Tessarolo	CNN	Research,implementation, fine-tuning, training
	SURF	Research,implementation, fine-tuning, training
Daria Mochalkina		Research