Politecnico di Milano

Image Analysis and Computer Vision

Project Report

Course teacher
Professor Vincenzo Caglioti

Project supervisors
Professor Giacomo Boracchi
Professor Diego Carrera

Perugini Alex 876359
Re Marco 873564
Scotti Vincenzo 875505

Contents

Chapter 1

Problem formulation

The proposed project consisted in the implementation of a convolutional neural network for the classification of sea lions using pictures extracted from aerial images.
The idea of the project is based on a Kaggle competition featured in 2017 (`https://www.kaggle.com/c/noaa-fisheries-steller-sea-lion-population-count`) where the objective was to provide a sea lions population count using fully convolutional neural network to analyze the aerial images taken by drones. Moreover it was required also to distinguish five classes among sea lions based on age and sex:

- adult male
- adult female
- subadult male
- juvenile
- puppy

The original aim of this competition was to automatize work done by biologist to keep track of the sea lions population. This manual work takes up to four months to count sea lions from those images. Due to this long time needed, automatizing this work would allow biologist to focus more on sea lions problems, rather on this counting task.
The requirement assigned to our group for the project was to provide a classifier that was able to distinguish only 2 classes:

- sea lion
- background

without any distinction between the sea lions subclasses.
To accomplish to the task we used the dataset provided by kaggle that included,

for each image, the ground truth expressed as a point centered on each sea lion with different colors for the different classes.

Figure 1.1: Sample image from the dataset

Chapter 2

State of the art

2.1 Introduction

Image classification is a fundamental problem in computer vision since it forms the basis for other computer vision tasks such as localization, detection, and segmentation[4]. Although the task can be considered second nature for humans, it is much more challenging for an automated system.
Traditionally handcrafted features were first extracted from images using feature descriptors, and these served as input to a trainable classifier, in this way the accuracy of the classification task was profoundly dependent on the design of the feature extraction stage, and this usually proved to be a difficult task that required domain experts[7]. In recent years, deep learning models that exploit multiple layers of nonlinear information processing, for feature extraction and transformation as well as for pattern analysis and classification, have been shown to overcome these challenges. Among them, CNNs have become the leading architecture for most image recognition, classification, and detection tasks[8, 9, 10].

2.2 Convolutional Neural Networks

CNNs are feedforward networks in that information flow takes place in one direction only, from their inputs to their outputs. Just as artificial neural networks (ANN) are biologically inspired, so are CNNs. The visual cortex in the brain, which consists of alternating layers of simple and complex cells, motivates their architecture [5, 6]. CNN architectures come in several variations, however, in general, they consist of convolutional and pooling (or subsampling) layers, which are grouped into modules. Either one or more fully connected layers, as in a standard feedforward neural network, follow these modules. Modules are often stacked on top of each other to form a deep model.
Despite some early sucesses[8, 9, 11], deep CNN were brought into the limelight as a result of the deep learning reanaissance[12, 13, 14], which was fueled by

GPUs, larger data sets and better algorithms [16, 3, 1]. Several advances such as GPUs implementations and the application of maximum pooling have contributed to the recent popularity.
The most significant advance, which has captured intense interest in DCNNs, especially for image classification tasks, was achieved in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 [1]. Since then, DCNNs have dominated subsequent versions of the ILSVRC and, more specifically, its image classification component [3, 17, 18]. In addition to these advances, other improvements affected the performances of DCNN such as non linear activation functions [19, 20], regularization mechanisms [21, 22] and optimization techniques [23, 1].
Nowadays convolutional networks are driving the advances in recognition, in fact convnets are not only improving whole-image classification tasks [1, 3, 18], but also local tasks with structured output such as bounding box object detection[24, 25, 19], part and keypoint prediction [28, 27] and local correspondence [27, 26].

2.3 Fully convolutional networks

Due to the astonishing results in the aforementioned topics, the natural next step in the progression from coarse to fine inference yields to pixelwise prediction. A prior approach used convnets for semantic segmentation [29, 30, 31, 32, 33, 34, 35], however, during the training phase, this required that in the ground truth each pixel was labeled with the class of its enclosing object or region in the ground.
An important advance in this sense was introduced by fully convolutional neural networks [36]: the key idea was to extend convnets to arbitrary-sized inputs by model transfer. Differently from previous approaches that was based on sliding windows [24], separately computed feature extractors [19] or Recurrent CNN [25], FCNN exploit end-to-end supervised pre-training for pixelwise prediction [36].
Typical recognition nets were realized to take fixed-size inputs and produce non spatial outputs[1, 3, 18], the fully connected layers of these networks had fixed dimensions and threw away spatial coordinates. These fully connected layers can also be viewed as convolutions with kernels that cover the entire input regions, doing so it's possible to cast them into fully convolutional networks that take inputs of any size and give as outputs the classification maps[36].
Since the classification nets subsample to keep filters small and computational requirements reasonable, the output of a fully convolutional version of these nets results coarsen, reducing it from the size of the input by a factor equal to the pixel stride of the receptive fields of the output units. To cope with this problem and connect coarse outputs to dense pixels interpolation by upsampling can be adopted. Upsampling can be implemented by deconvolution and learned by backpropagation of the pixelwise loss[36].

Chapter 3

Solution Approach

The main idea for this project is to realize a Deep CNN classifier that is able to distinguish sea lions from their habitat, the choice of a Deep CNN is due to the great results that they can achive in this kind of tasks [1, 3, 19]. The successive step was to extend the classifier to a Fully CNN [36], in this way the object detection task proposed by the orginal Kaggle competition can be pursued.
Images in the provided data set were taken from drones, thus they are very large and contains a lot of sea lions and different background areas. Due to the impossibility of using the whole images to train the network, the first step was to create a suitable training dataset. To do this, we extracted both sea lions centered patches and background patches by cropping the images provided in the original data set and trained the binary classifier on these patches.
After having extracted them, we decided to apply data augmentation to obtain a more robust classifier. This allowed us to increase a lot the dimension of the original dataset, in particular for what concerns sea lions patches. In fact, due to the nature of the images, they contained much more background patches than sea lions ones. By data augmentation we were able to overcome, partially, this unbalance and to obtain better performance during testing phase, because it makes the classifier more insensitive to position, rotation and scaling.
Once performances over patches were satisfying, we moved to an higher level, modifying the network to take as input the whole image and providing an heatmap of it. This gives the possibility, given an image, to see where and how sea lions are distributed in the environment and to compute an estimate of how many sea lions are present in that image.

Chapter 4

Implementation

4.1 Dataset creation

The first thing to do with images provided by Kaggle is to divide them in train and test set, in particular the first 750 images were used for training and the remaining ones (from 751 to 947) for testing.
The extraction of patches from Kaggle images has been done performing the absolute difference between the original image and its corresponding ground truth with colored dots on the sea lions, in this way it's possible to gather from each image coordinates and class of all the sea lions.

(a) Original image (b) Ground truth image

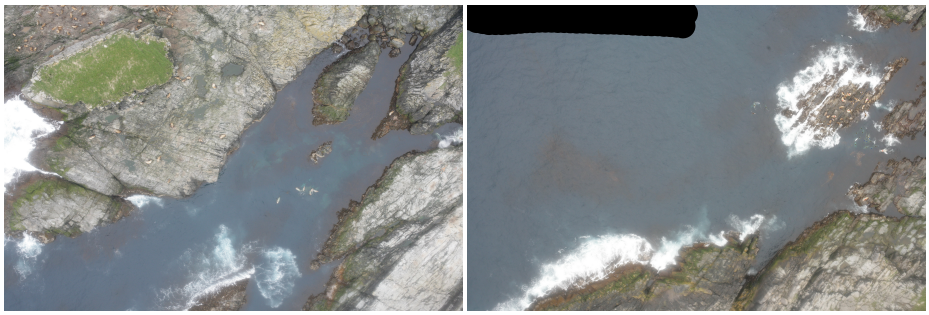
Figure 4.1: Comparison of train image and ground truth provided by kaggle

Given these coordinates it's easy to cut from the original images a 96×96 area around them and save the patches labeling them as 'sea lion'.

To extract the background we used a sliding window of size 96×96 over the image and cut all the patches which were not intersecting with a sea lion patch and save them with label 'background'.

The train set at the end of the extraction procedure includes 50079 'sea lions' and 1139531 'background' patches which will be split again in 40411, 9668 and 1129863, 9668 respectively for proper train and validation procedures. The test set instead has 13539 'sea lions' and 277390 'background' patches. Given this dataset divided into classes, train, validation and test set it's possible to train the model.
Going deep in the training and testing procedures we noticed some important problems with the dataset:

- mismatches in the ground truth provided by kaggle, we listed and removed all these images



(a) Original image '21' (b) Ground truth image '21'

Figure 4.2: Example of mismatch between original image and ground truth provided by Kaggle

- blob detection technique provided some false positives that ended up labeled in the wrong way, so we also inspected manually the patches to cope with this problem
- some images have areas covered with black color so we added a check to discard all patches with more than 2% of pure black color, in this way the sliding window gathering the background excludes black patches

(a) Ground truth image '22' (b) Ground truth image '26'

Figure 4.3: Example of two images covered with black areas

As can be seen from these two images avoiding balck patches is fundamental to have a correct dataset.

Once the extraction procedure has been fully completed and all its related problems have been solved, we detected new problems related to the patches themselves. First of all, if we used the 40411 sea lion patches, what happens is that they might not be sufficient to correctly classify a sea lion. Also, the two classes are really unbalanced, there are a lot more background patches than sea lion ones. So, if we used all the patches, there was the risk of overfitting over the sea lions ones.

4.2 Data augmentation

As explained before, to increase the dimension of the dataset and generalize more over position, rotation and scale we applied data augmentation to the sea lions patches [39]. This technique has been applied to solve the problems related to the dataset that came up after patches extraction (4.1), in fact it allows to expand the dataset creating new patches modifying the extracted ones, this is fundamental especially due to the poorness that characterize the sea lion class. Moreover, training a model with augmented patches makes it more robust and consistent in classifying previously unseen samples. In particular, what we applied is a random combination of four transformations [40]:

- Rotation of a random degree in the range 0 to 360 degrees
- Flipping both vertically and horizontally
- Shifting of a maximum of 10 pixels, both vertically and horizontally
- Zooming of a maximum of 20 pixels

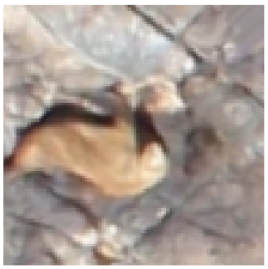
The transformation process is, as said before, completely random so the number of new patches that can be generated is really high, that's why with these

transformations is possible to overcome the problem of lack of sea lion patches and class unbalance.

The first implementation we tried for the augmentation procedure was to build our own handcrafted tool which behaved exactly as described before. This tool was built to receive as input higher size sea lion patches (144×144) to avoid interpolation of eventual missing pixels due to the transformations and to have a final transformed patch as much precise as possible. Of course patches must be modified at runtime, during the network training, because it's impossible to save all the patches that can be generated with augmentation. At this point we noticed that this handcrafted tool was not as fast as expected so we decided to replace it with the Keras built-in data generator which allows to apply augmentation as well. Data generator provided by Keras doesn't require bigger patches as input beacuse automatically applies interpolation to fill eventual missing pixels and mantain sizes of the input image. With this augmentation tool the process of loading modified patches at runtime speeded up consistently while the applied transformations are the same, so the whole train process became less time consuming.

Here follows samples of modified patches

(a) Original image



(b) Augmented image

(c) Augmented image

(d) Augmented image

(e) Augmented image

(f) Augmented image

(g) Augmented image

Figure 4.4: Augmentation samples

4.3 Batch generator

Starting from the second implementation of the network we added a custom batch generator to improve memory management and handle augmentation. This was a necessary consequence of the training approach we adopted, in fact for what concerns the sea lion patches we decided to leave in memory both the training and validation sets, while for what concerns background patches, due to the huge amount, only the validation set was maintained. On the other hand, the training patches were instead periodically reloaded.

Moreover, to take advantage of Keras queue system for batch processing, we actually performed augmentation just before yielding the batch to the queue, in this way the training could proceed without waiting that the entire data set of the current epoch had undergone transformations and we avoided to fill all the memory.

To be more precise, the first step was to load the entire validation set. Subsequently all the sea lions patches from the training set were loaded in memory as soon as the first batch request was dispatched and then periodically, an equal amount of background patches from the train set was randomly sampled and loaded. At this point augmentation was applied batch-wise until the end of the training.

A key problem related to memory management was that even though pixels required $8bit$ for each color channel, since we needed to normalize the values in a $[0, 1]$ range we were forced to use a $32bit$ floating point representation that would make the current data sets four times bigger. Thankfully as a side effect of the use of batch generator only the current batch images were changed to that representation avoiding the explosion of required memory.

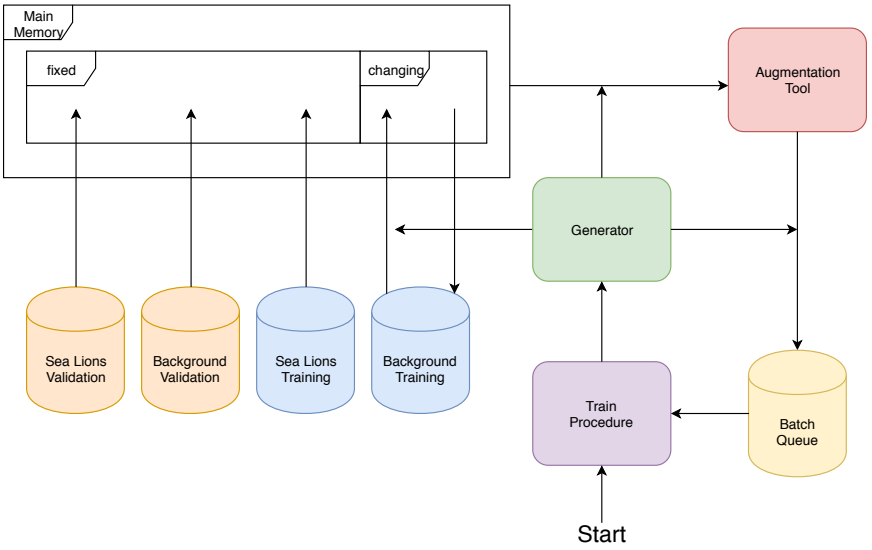


Figure 4.5: Batch generator working diagram

4.4 Model

The DCNN used to classify patches has input size of $96 \times 96 \times 3$ and output size of 2, where each output corresponds to the probability of belonging to a class. The network is made up of 9 layers

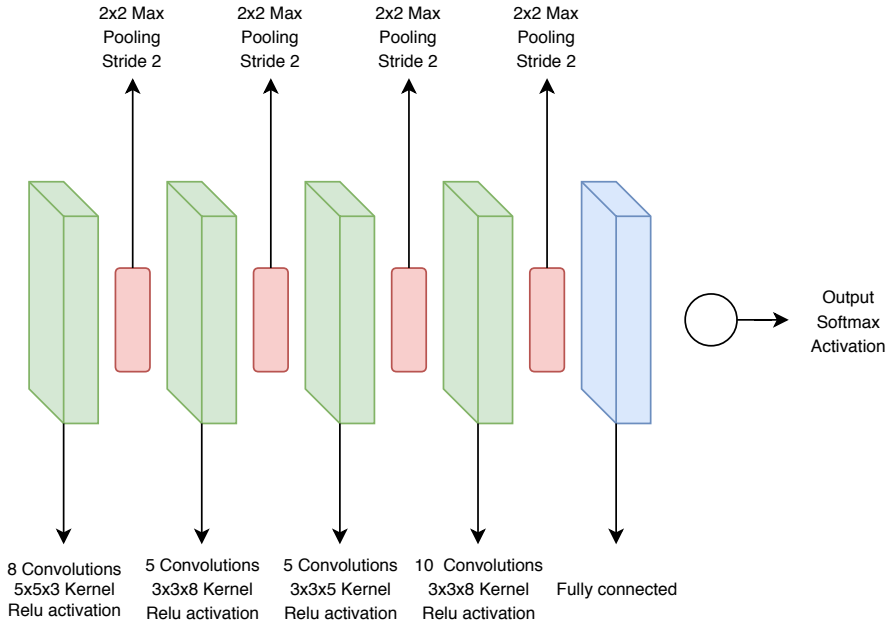


Figure 4.6: CNN model

The optimizer we chose is Adam, a memory and computationally efficient algorithm for first-order gradient-based optimization of stochastic objective functions. Defining α the step size, $\hat{m}_t$ the bias corrected first moment and $\hat{v}_t$the bias corrected second raw moment, the update rule at timestep t can be written as $\Delta_t = \alpha \cdot \hat{m}_t / \sqrt{\hat{v}_t}$. In more common scenarios, the effective magnitude of the steps can be approximately bounded by the stepsize α and moreover this update rule also provides smaller steps in proximity of local optima, due to the ratio of first and secondo order moments, which goes to 0 in this regions. Finally, in the context of Deep Convolutional Neural Networks, Adam shows performances similar or better than SGD with momentum, including also the benefit of the adaptive learning rate[42].

Binary cross entropy is the loss function we decided to use for training. It measures performances of a binary classifier by increasing the evaluated loss as the predicted probability diverges from the true label. For model parameters θ, true labels y and predicted probabilities p, the binary cross entropy is computed as:

$$L(\theta) = -(y \cdot \log(p) + (1 - y) \cdot \log(1 - p))$$

As a metric to evaluate the performances of the model, we used the binary accuracy. For each test sample, it takes as input the predicted probability p and the true label y. Then it means over all the samples the number of correctly classified ones, i.e. if $p > 0.5$ and $y = 1$, the labeling is considered correct, otherwise it isn't. In particular, this is Keras implementation of it, while the formal definition is

$$acc = \frac{TP + TN}{P + N}$$

where P, N, TP and TN are respectively total number of positive samples, total number of negatives, correctly classified true positives and correctly classified true negatives. This means that the accuracy is defined as the proportion of true results among the total number of cases examinated.

4.5 Object Detection

As soon as the CNN building and training phases ended the fully convolutional model extraction could proceed. To do so the fully connected final layer was reshaped to another convolutional layer maintaining the same connections and related weights of the original network, the new final convolutional layer used 2 filters (one for each label) and a 4×4 kernel, with a softmax activation function. Since the constraints on the input shape were dropped it was possible to feed an entire image to the network that produced as output two heatmaps that highlighted respectively the pixels belonging to the sea lion and the background classes according to the probabilities predicted by the classifier.
It is important to stress out that any shift-and-stitch or upsampling via deconvolution to yield the dense predictions was used [41], instead was applied a simple interpolation to the results.
The final step was to use the computed heatmap of an entire image to locate all the sea lions. To do so at first the heatmap was hard thresholded (with pixel probabilty threshold set to 0.9) then, in order to have a clearer distinction from the background, a blob detector based on the Laplacian of the Gaussian was applied to results [1].
Given an input image $f(x, y)$, this image is convolved by a Gaussian kernel

$$g(x, y, t) = \frac{1}{2\pi t} e^{-\frac{x^2+y^2}{2t}}$$

where t is the scale give a scale space representation.

$$L(x, y; t) = g(x, y, t) \times f(x, y)$$

Then, the result of applying the Laplacian operator

$$\nabla^2 L = L_{xx} + L_{yy}$$

is computed, which usually results in strong positive responses for blobs of radius $r = \sqrt{2t}$.

Since this technique presents some problems with automatic detection of blobs with different (unknown) size in the image domain, the normalization of the Laplacian was adopted alongside a multi-scale approach implemented by defining a range in which the different possible scales could be tested.
In the end the points of interest $(\hat{x}, \hat{y})$ are retrived as

$$(\hat{x}, \hat{y}; \hat{t}) = argmaxminlocal_{(\hat{x},\hat{y};\hat{t})}((\nabla^2_{norm} L)(x, y; t))$$

Chapter 5

Experimental activity and results

5.1 Developement tools

In order to perform the tasks required we developed the project in Python, using PyCharm IDE. In particular, we used Jupyter notebooks that allowed to write the code and to execute it section by section, in this way we were able to monitor the results as soon as each step was completed.
Thanks to the choice of Python we managed to use Keras, an high-level neural networks API, with the TensorFlow framework as backend. The choice of Keras helped to simplify the development while TensorFlow allowed us to take advantage of GPU acceleration that sped up the entire training process significantly. Another aspect that characterized the development was the use of dataframes that allowed a better management of all the informations about the data set.

5.2 Balanced dataset

Experimental activity started by extracting patches in order to create a train set. By applying the procedure explained in section 4.1, we obtained a train set including 50079 'sea lions' and 1139531 'background' patches which are split again in 40411, 9668 and 1129863, 9668 respectively for proper train and validation procedures. The test set instead has 13539 'sea lions' and 277390 'background' patches.

As can be seen from the numbers of extracted patches the dataset is heavily unbalanced and there are a lot more background patches than sea lion ones. Rather than using all the background patches, we decided as a first attempt to randomly sample them and produce a balanced train set. Extracted sea lions patches were 50079, so an equal number of background patches were taken. The reduced dimensions of the used train set, allowed it to be fit all in memory,

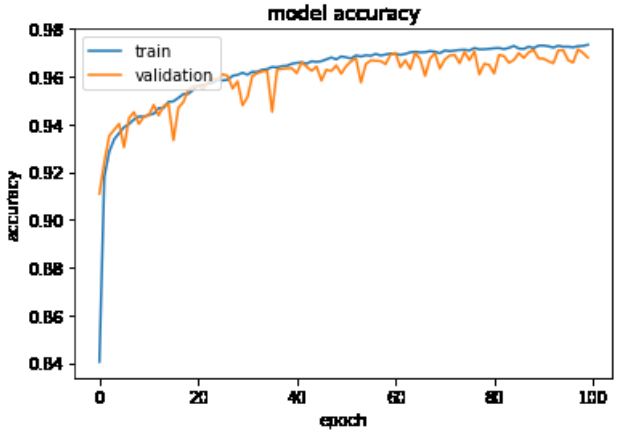
producing good performances in terms of minutes per epoch.

The simplicity of this first training attempt allowed us to tune the network in diffenrent ways and test the results:

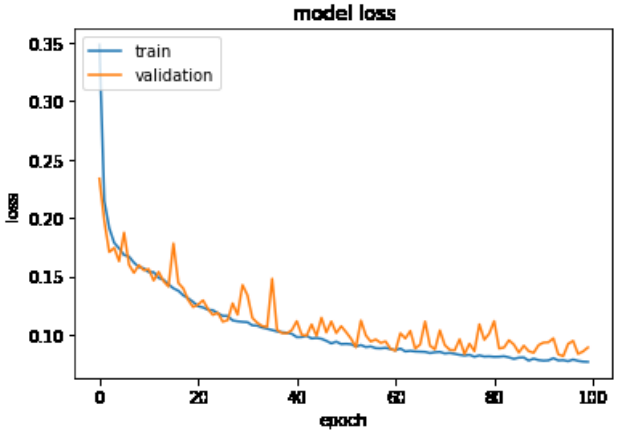
- learning rate: the best learning rate to improve at each epoch ended out to be 0.001, in fact a smaller one seemed not sufficient to learn while an higher one made loss values fluctuating up and down not converging
- optimizer: we tried replacing Adam with stochastic gradient descent (SGD) but with poor results, in fact it didn't give the same improvement performaces of Adam so we kept the latter
- dropout: a technique we tested to avoid overfitting was adding a dropout layer between each layer of the network, this didn't brought any improvement, the network wasn't able to learn enough during training so we decided to stick to the original model

After all these trials and considerations we were able to train the network for 100 epochs using a learning rate of 0.001.

Testing of the model trained in this way was performed using a balanced test set, using once as before the total number of sea lions patches as an upper bound to the background ones.



(a) Model accuracy



(b) Model loss

Figure 5.1: Loss and accuracy history

To measure performances we considered not only the accuracy, but also the area under the curve (AUC) of the receiver operating characteristic (ROC) curve. Up to this point, we achieved an AUC of 99.29%. Here follows more detailed tables about the results.

	Loss	*Accuracy*
Train set	0.0768	0.9733
Validation set	0.0891	0.9679

(a) End of training performances

Prediction accuracy	96.60%
AUC	0.9929

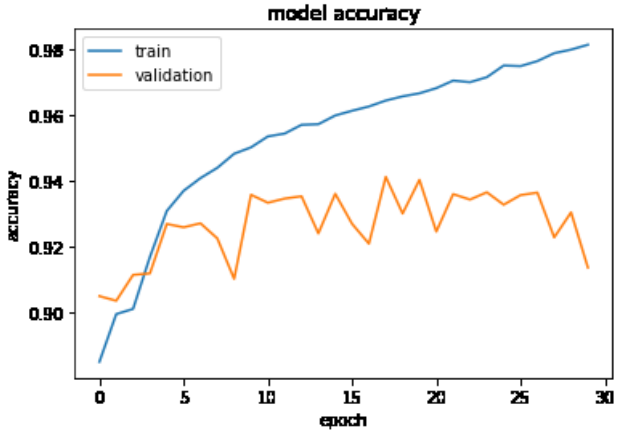
(b) Testing results

Table 5.1: First training results

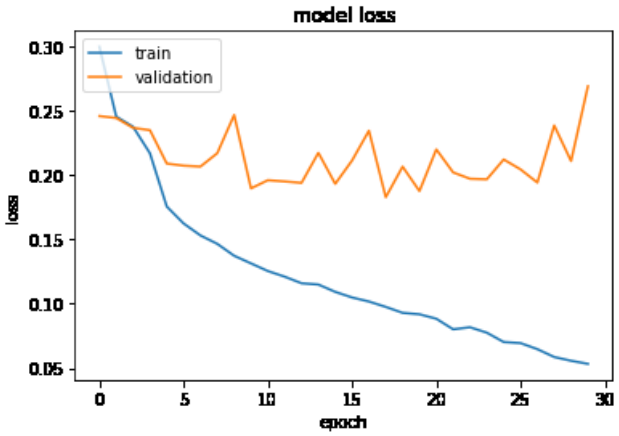
5.3 Batch generator and augmentation

After that, we decided to use the whole dataset both for training and testing rather than a balanced one because the real problem is to identify sea lions among a lot of background. This caused different problems, in particular due to memory usage and to high number of background patches with respect to sea lions ones.
Memory related problems were linked to the fact that training dataset was of some Gigabytes, thus it couldn't fit all in the memory at the same time. To overcome this we created a batch generator which, as explained in section 4.3, retrieved a limited number of patches from the memory and fed them to the network at each epoch. In this way only a little amount of patches were moved into the RAM at the same time.
This generator also enabled us to introduce data augmentation in this training step. It was needed due to the high difference in number between sea lions and background patches and also to obtain more robust results in classification, see section 4.2 for more details.
To train the network we used the generator to create a balanced data set composed of all the sea lions patches with augmentation and an equal number of background patches randomly sampled from all the background patches. This dataset is used for one epoch and then changed, creating a new one with the same criteria. In this way the network sees all the sea lions patches different times but overfitting is prevented by the wide amount of modifications introduced by the augmentation. While the background patches are used in an efficient way because network sees a wide variety of them while keeping the procedure memory efficient.
Although these premises, results were not good and the network very bad, being not so better with respect to a random classifier.
This training allowed us to identify three new problems: changing the dataset at each epoch is time consuming, because loading the patches from disk to memory is a bottleneck in the whole procedure; moreover due to the handcrafted augmentation tool even the batch creation was slowed down; puppies are really hard to be identified by the network, with their color and shape can be easily mistaken as background also by human eye.

Given that this training was particularly time consuming for the reasons discussed before, the network was trained for only 30 epochs with a 0.0005 learning rate. The procedure took a lot of time and was not sufficient to learn enough from the data provided.



(a) Model accuracy



(b) Model loss

Figure 5.2: Loss and accuracy history

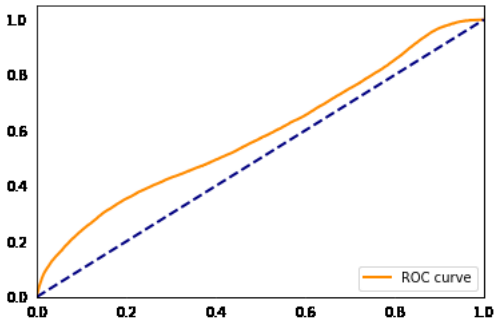


Figure 5.3: *ROC* curves

As shown by the following tables the presence of a much higher number of background patches influenced a lot the metrics used to estimate the model performances, in particular while the accuracy was still close to that of the previous training the AUC shows a significant drop and gets close to 0.5 (the random classifier limit).
We can also directly compare the accuracy of this training with the previous one. If we restrict the analysis to the first 30 epochs and on the validation set, to have a meaningful comparison, the first training provides an accuracy of about 96.6%, while this one only of the 91.38%. Also, the variation of accuracy from the beginning of the training is about 4%, while this one is of less than the 2%.

	Loss	*Accuracy*
Train set	0.0533	0.9813
Validation set	0.2689	0.9138

(a) End of training performances

Prediction accuracy	94.91%
AUC	0.5864

(b) Testing results

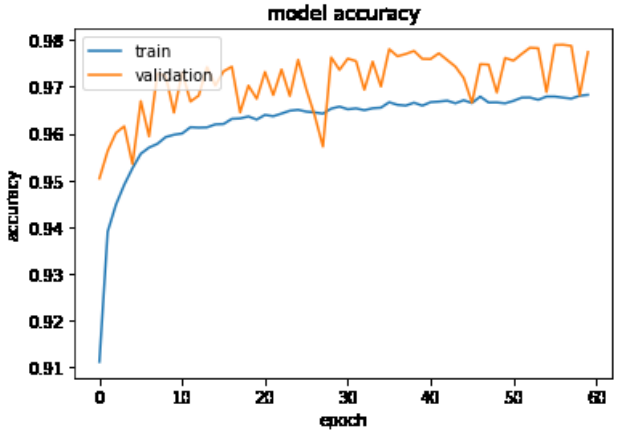
	Precision	*Recall*
Sea lions	0.2948	0.0228
Background	0.9516	0.9972

(c) Precision and recall scores

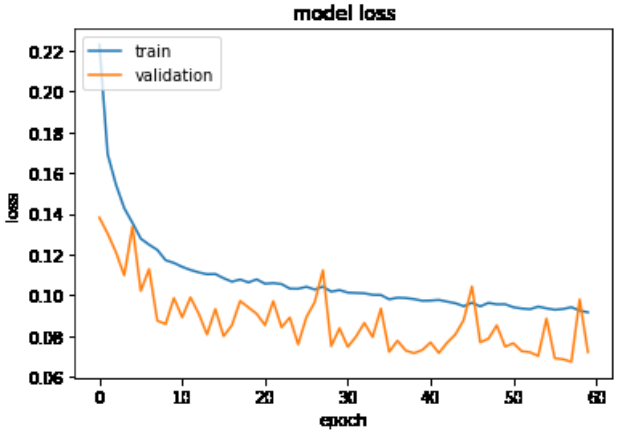
Table 5.2: Second training results

5.4 Keras augmentation and puppies exclusion

The purpose of this further modifications was to solve all the problems that came up in the previous one.
The first adjustment was to change the generator and make it reload a different dataset from memory every 15 epochs and not at each epoch. This choice speeded up the whole training phase allowing us to train for more epochs. The problem related to puppies was solved analyzing where and how they appear in the images, in fact it can be noticed that this particular class is often next to a female sea lion.
Thanks to this information we were able to apply a second important change: remove puppies from the dataset. In this way for the network it's easier to learn and distinguish between all the other classes of sea lions and the background so it's possible to achieve better performances. At the end, in the final result, the number of puppies can be estimated from the number of sea lions detected.
Another remarkable difference with respect to the last training was that we dropped the handcrafted augmentation tool in favor of Keras built-in one that enabled a much higher throughput in the creation of the training batches, as seen in section 4.2.
Performances of this model were measured again using accuracy, that is 97.27%, and *AUC* of the *ROC* curve, which is 99.64%, plus other metrics such as precision and recall respectively 62.53% and 97.68% over the sea lions.



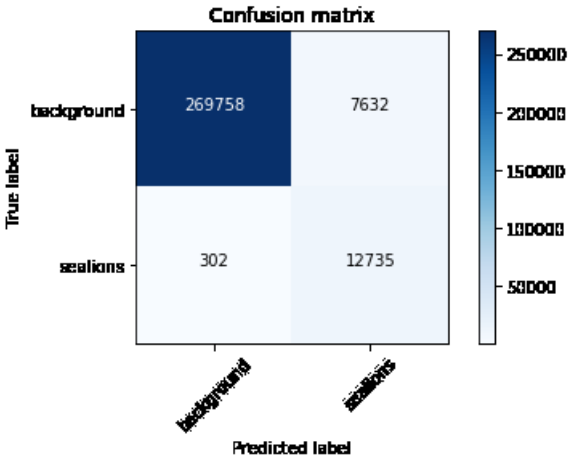
(a) Model accuracy



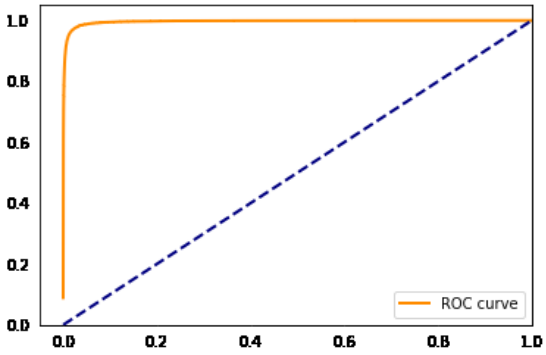
(b) Model loss

Figure 5.4: Loss and accuracy history

During this phase we set the learning rate to 0.0005 and we trained for 60 epochs.
The following graphs and tables highlight the enhancements of the new training: the ROC nearly covers all the graph area, resulting in a AUC higher than first training one, the same applies for the precision-recall curve.



(a) Confusion matrix



(b) *ROC*

Figure 5.5: Confusion matrix and *ROC* curve

Prediction accuracy	97.27%
AUC	0.9964

(a) Testing results

	Precision	*Recall*
Sea lions	0.6253	0.9768
Background	0.9989	0.9275

(b) Precision and recall scores

Table 5.3: Third training results

We also performed some testing in the case of a balanced test set. Here are

reported the results in this case and we can see that the precision in classification of sea lions increases highly with respect to the unbalanced test set case. On the other hand, there is a slight decrease in the precision of background classification.

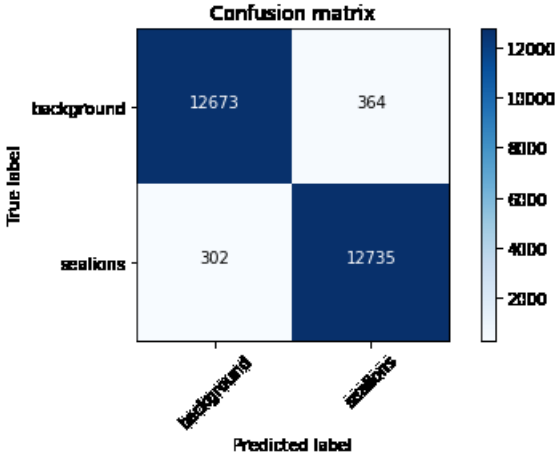


Figure 5.6: Confusion matrix with the balanced test set

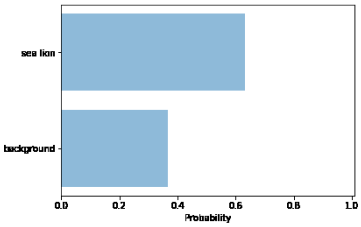
	Precision	*Recall*
Sea lions	0.9722	0.9768
Background	0.9767	0.9721

Table 5.4: Precision and recall scores on a balanced test set

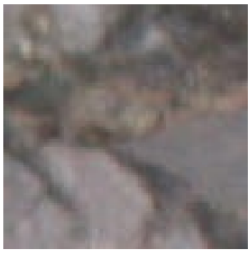
5.4.1 Misclassified patches

In this section are presented some errors made by the network in classifying patches

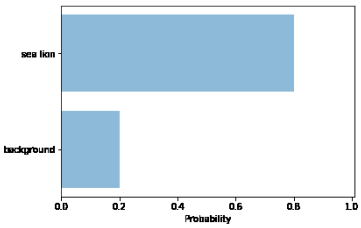
(a) Background patch



(b) Prediction values

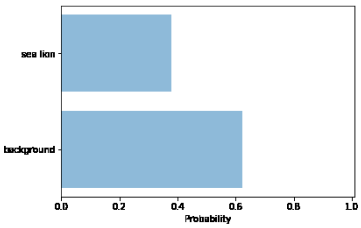


(c) Background patch

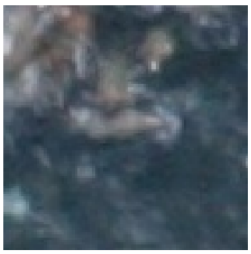


(d) Prediction values

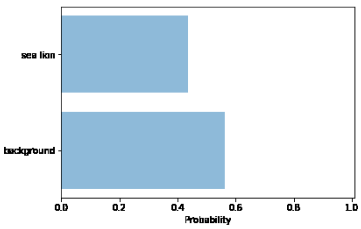
(e) Sea lion patch



(f) Prediction values



(g) Sea lion patch



(h) Prediction values

Figure 5.7: Misclassified patches

As can be seen looking at the misclassified patches, the errors committed by the network are reasonable, in fact sometimes there are sea lions whose color and shape is hardly distinguishable from the background therefore are classified as background. In other cases rocks or stones have color and shape such that they are interpreted as sea lions by the network.

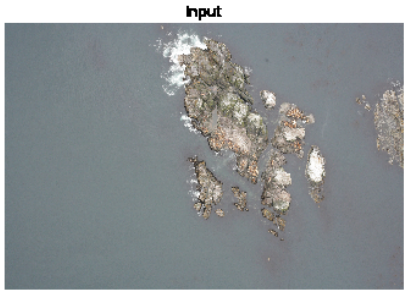
5.5 Object detection and sea lions counting

At this point we took the last trained network and, as explained before in (4.5), replaced the last fully connected layer with a convolutional one maintaining all the weights. With this reshaped network we are able to evaluate an entire image and obtain a heatmap highlighting zone where is more probable to have a sea lion.
Then we decided to apply a threshold on the heatmap to create a grayscale image with white zones corresponding to sea lions. On this new image it was possible to apply blob detection, refined through parameter tuning as for the heatmap threshold, to have a rough count of how many sea lions are present in the considered image. To do so we used the built in function from the scikit-learn library and after some tests we decided for the following parameters:

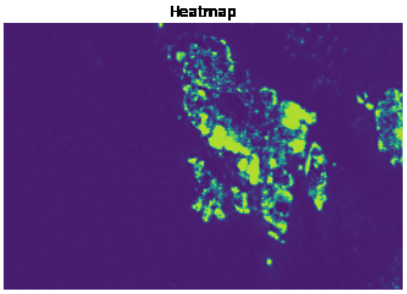
Heatmap *threshold*	0.9	*LoG* scale minimum value	20
		LoG scale maximum value	35

Table 5.5: Detection Parameters

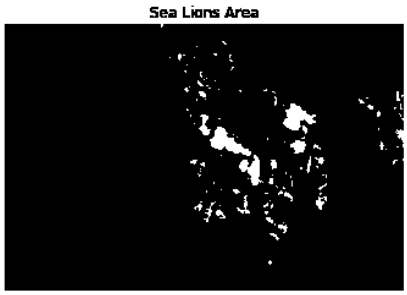
The entire procedure is depicted in the following figures. As shown the results are quite satisfying even though the blobs don't always overlap correctly with the sea lions.



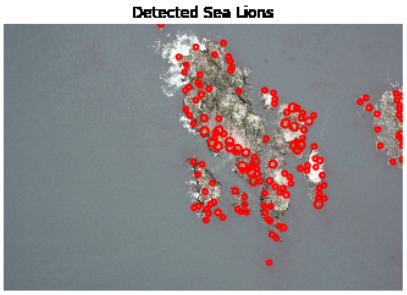
(a) Original image



(b) Heatmap



(c) Blobs



(d) Detected sea lions

Figure 5.8: Sea lions detection

Here follows a comparison of blob counting and true number of sea lions

	True	*Predicted*
Image 755	212	233
Image 771	149	141
Image 773	391	411
Image 907	93	123

Table 5.6: Sea lions counting based on blobs

As can be seen from these numbers, even if the prediction on the nuber of sea lions isn't performed with a deeply studied strategy, final results aren't too bad. In particular, looking at figure 5.8d, can be noticed that some animals are not covered by blobs and there are some mistaken blobs over the background, therefore we can say this method is not the best to count sea lions with high precision due to the presence of these errors.

Another final consideration that can be done is that the predicted number is clearly an overestimate of the real one, but the presence of puppies, which

are not considered in this network, makes the two nubers be closer one to each other.

Chapter 6

Conclusions

The work done for this project produced a convolutional neural network able to achieve good performance on the classification of sea lions over background. In particular we want to point out how the two major enhancements we did to improve the performance of the network, i.e. data augmentation and the management of the dataset during the training, have actually set out the condition for a good classification.
The first approach to augmentation didn't produce the expected results. This was mainly caused by the high variety of data introduced by it and the complexity for the network to learn it. In fact, the high computational cost of the training didn't allow us to run an high number of epochs. Thus, the benefits of augmentation were overcome by its disadvantages. After we introduced generator and by it optimized the memory usage, we found out that the augmentation provided the expected improvements.
There are still some open problems linked to this project.
First of all we have overcome the recognition of puppies due to the intrinsic difficulty of the task. Further works could reintroduce them with a correct classification.
Our task was to produce a binary classifier which could distinguish between a sea lion and the background. An extension to this could be a multiclass classifier which can effectively recognise among the classes of the original competition.
Another approach that could produce an improvement in the results is to exploit transfer learning from models trained with imageNet, for example, since they provide optimal performances in low level feature extraction [41].
The last thing we imagine could work better is a direct object recognition approach without going through the classification task, however this would require a much more detailed ground truth [36].

Bibliography

[1] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), Advances in neural information processing systems, 25 (pp. 1097–1105). Red Hook, NY: Curran.

[2] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to hand-written zip code recognition. In Neural Computation, 1989.

[3] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv 1409.1556.

[4] Karpathy, A. (2016). CS231n: Convolutional neural networks for visual recognition.

[5] Hubel, D. H., & Wiesel, T. N. (1959). Receptive fields of single neurones in the cat's striate cortex. Journal of Physiology, 148(1), 574–591.

[6] Hubel, D. H., & Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. Journal of Physiology, 160(1), 106–154.

[7] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278–2324.

[8] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989a). Handwritten digit recognition with a back-propagation net- work. In D. S. Touretzky (Ed.), Advances in neural information processing systems, 2 (pp. 396–404). Cambridge, MA: MIT Press.

[9] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989b). Backpropagation applied to handwritten zip code recogni- tion. Neural Computation, 1(4), 541–551.

[10] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436– 444.

[11] Simard, P. Y., Steinkraus, D., & Platt, J. C. (2003, August). Best practices for convolutional neural networks applied to visual document analysis. In Proceedings of the 7th International Conference on Document Analysis and Recognition (vol. 3, pp. 958– 963). Washington, DC: IEEE Computer Society.

[12] Hinton, G. E., Osindero, S., & Teh, Y. (2006). A fast learning algorithm for deep belief nets. Neural Computation, 18(7), 1527–1554.

[13] Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. Science, 313(5786), 504–507.

[14] Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2006). Greedy layer-wise training of deep networks. In J. C. Platt, D. Koller, Y. Singer, & S. T. Roweis (Eds.), Advances in neural information processing systems, 19 (pp. 2814–2822). Red Hook, NY: Curran.

[15] H. Kong, H. Akakin, and S. Sarma. A generalized Laplacian of Gaussian filter for blob detection and its applications. In IEEE Xplore, 2013

[16] Deng, L., & Yu, D. (2014). Deep learning: Methods and applications. Foundations and Trends in Signal Processing, 7(3–4), 197–387.

[17] Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional net- works. In Proceedings of the European Conference on Computer Vision (pp. 818–833). Berlin: Springer.

[18] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., . . . Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 1–9). Los Alamitos, CA: IEEE Com- puter Society.

[19] He, K., Zhang, X., Ren, S., & Sun, J. (2015a). Deep residual learning for image recognition. arXiv 1512.03385.

[20] Xu, B., Wang, N., Chen, T., & Li, M. (2015). Empirical evaluation of rectified activations in convolutional network. arXiv 1505.00853v2.

[21] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. arXiv 1207.0580.

[22] Zeiler, M. D., & Fergus, R. (2013). Stochastic pooling for regularization of deep convolu- tional neural networks. arXiv 1301.3557.

[23] Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feed-forward neural networks. In Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (pp. 249–256). jmlr.org/proceedings/papers/v9 /glorot10a/glorot10a.pdf

[24] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In ICLR, 2014.

[25] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In Computer Vision and Pattern Recognition, 2014.

[26] P. Fischer, A. Dosovitskiy, and T. Brox. Descriptor matching with convolutional neural networks: a comparison to SIFT. CoRR, abs/1405.5769, 2014.

[27] J. Long, N. Zhang, and T. Darrell. Do convnets learn correspondence? In NIPS, 2014.

[28] N. Zhang, J. Donahue, R. Girshick, and T. Darrell. Part- based r-cnns for fine-grained category detection. In Computer Vision–ECCV 2014, pages 834–849. Springer, 2014.

[29] F. Ning, D. Delhomme, Y. LeCun, F. Piano, L. Bottou, and P. E. Barbano. Toward automatic phenotyping of developing embryos from videos. Image Processing, IEEE Transactions on, 14(9):1360–1371, 2005.

[30] D.C.Ciresan,A.Giusti,L.M.Gambardella,andJ.Schmid- huber. Deep neural networks segment neuronal membranes in electron microscopy images. In NIPS, pages 2852–2860, 2012.

[31] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2013.

[32] P. H. Pinheiro and R. Collobert. Recurrent convolutional neural networks for scene labeling. In ICML, 2014.

[33] B. Hariharan, P. Arbela´ez, R. Girshick, and J. Malik. Simul- taneous detection and segmentation. In European Confer- ence on Computer Vision (ECCV), 2014.

[34] S. Gupta, R. Girshick, P. Arbelaez, and J. Malik. Learning rich features from RGB-D images for object detection and segmentation. In ECCV. Springer, 2014.

[35] Y. Ganin and V. Lempitsky. N4-fields: Neural network nearest neighbor fields for image transforms. In ACCV, 2014.

[36] J. Long, E. Shelhamer and T. Darrel. Fully Convolutional Networks for Semantic Segmentation. In IEEE Xplore, 2015.

[37] B. Hariharan, P. Arbela´ez, R. Girshick, and J. Malik. Hypercolumns for object segmentation and fine-grained localization. In Computer Vision and Pattern Recognition, 2015.

[38] J. Tompson, A. Jain, Y. LeCun, and C. Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. CoRR, abs/1406.2984, 2014.

[39] J. Wang and L. Perez. The Effectiveness of Data Augmentation in Image Classification using Deep Learning. Google Scholar, 2017.

[40] E. Jannik Bjerrum. SMILES Enumeration as Data Augmentation for Neural Network Modeling of Molecules. ArXiv e-prints, Mar. 2017.

[41] M. Oquab, L. Bottou, I. Laptev and J. Sivic. Learning and Transferring Mid-Level Image Representations using Convolutional Neural Networks. In IEEE Xplore, 2014.

[42] D. P. Kingma, J. L. Ba. Adam: A method for stochastic optimization. ICLR, 2015