Uppsala University

A hierarchical architecture of CNNs to increase the performance on a categorization problem
(Course Code: 1DT071)

Project Report

PROJECT GROUP 9

UPPSALA
UNIVERSITET

May 30, 2018

Professor: Olle Gllmo
Supervisor: Damian Matuszewski

Authors:
Aurélien Colin
Naresh Kumar Nagarajan

Abstract

In the field of deep learning, the image processing has recently know a huge improvement due to the use of convolutional neural networks. In this document, we present a structure, using an architecture of multiple convolutional neural networks working together, in order to study if the use of an hierarchical point of view could lead to an increase of the performance, compared to a system using only one of such network (a flat architecture).

Glossary

activation map Also called "feature map", an activation map is the result of the convolution of a matrix (in our case, a picture or another activation map) with a convolution kernel.. 2

booru Image hosting website using collaborative tagging.. 5

CNN Convolutional Neural Network.. 2

final label The label is final if it is associated with a leaf of the architecture graph.. 9

MICP The Mean In-Cluster Probability, which is the probability of a picture being classified in the right cluster.. 9

persona To disambiguate with written characters, such as the one used in the dataset MNIST, we will call a fictional character "persona".. 1, 6

root the network at the root of the architecture, which decides of the cluster.. 9

subnetwork a network which is not the root of the architecture.. 9

Contents

1 Introduction

During the previous years, the machine learning field has seen the increase in the use of deep learning to solve some problems that weren't efficiently worked out. One of them, the categorization of pictures, makes extensive use of CNN (Convolutional Neural Network). These tools have astonishing performance on some problems but are often long to train and requires a lot of memory.

In this project, we designed a method to decrease these requirements of both time and memory by designing an architecture of multiple CNNs, working together in a hierarchical structure. We assume that, by dividing the work that the network had to do (in other words, by specializing them), we will be able to improve the performance of the categorization. We also think that such architecture is more flexible since we can extend the set of labels without having to retrain the full architecture. Alternatively, it is also feasible to fuse such hierarchical architectures.

After describing the works on which our study is based on, we will present the dataset, consisting of various anime characters (that we will call "persona" to avoid any confusion with written characters) and explains several issues occurring when working on these data, such as the heterogeneity of the pictures and the presence of similar personae. Then, we will describe the architecture of our networks, the reason behind this choice and also the method used to create the clusters containing the labels. At last, we will present the results of this study in term of success rate, training time and the aftermath of its conclusions. We also present some hyperparameters that we have try but ultimately discarded.

2 Literature Review

2.1 What is a CNN?

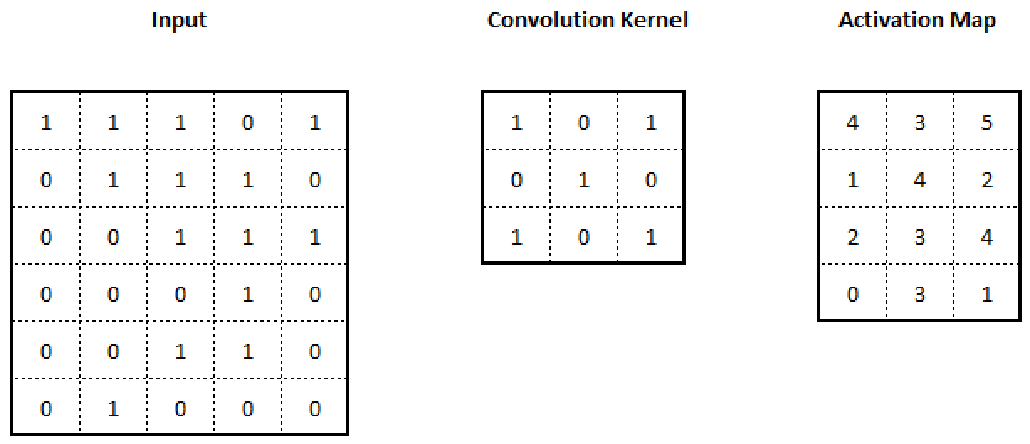
Convolutional Neural Network (CNN) is a specific kind of neural network widely use as deep learning models for image recognition. They are inspired by the biologic configuration of the visual cortex. This part of the brain has neurons receiving electro-chemical inputs from the retina. According to Hubel and Wiesel [HW59] discovered that some of these neurons are sensitive to shapes and organized in a columnar architecture. Applied in the machine learning field, this specificity of the visual cortex gave the idea of designing systems able to detect features (shapes, colors) and is a pillar of the CNNs.

2.1.1 A structure in four parts

A CNN is not dissimilar to ordinary neural networks in the sense that they are made of nodes that have biases, activation functions and weights on their connections with each other. The difference is that their task is to perform convolutions (in the case of image processing, on both spatial dimensions). They are usually made of four parts: the convolutional layers, the pooling layers, one flattening layer, and the fully connected layers.

Taking input as the actual image, which has, for example, a size of (H, W, C) for respectively the height, the width, and three colors channels, this layer will compute activation maps. We can see the kernel as a particular feature (for example a vertical line or a circle), and the activation map as a matrix whose value represents the presence of the feature depicted by the kernel. On the Figure 1, the kernel's feature is a cross, and the activation map shows that this feature is present in the up-right corner of the input, but it is totally absent in the lower-left corner. Of course, we don't want to detect only one feature, but a high number of them. Then, this means that the convolutional layer will, in fact, compute a number, that we note N, of filters (or kernels). Thus, the number of weights of this layer will be $(S + 1)(N \cdot C)$, with S the surface of the kernel (which will be 9 for a three-by-three matrix).

The size of the activation map will depend on two things: the padding and the stride. When convoluting the kernel, problems arise on the cells on the borders: it is not possible to convolve the kernel on a cell of the border of the input except by using padding, i.e. considering that the input can be extended on its borders by other values, usually 0. The second factor is the stride which is the length of the displacement of the kernel at each step. If we don't use padding and use a stride of 1, the dimension of the activation map is $(H - 2, W - 2, C)$ which is $(4, 3, 1)$ for the example of the Figure 1.



1	1	1	0	1
0	1	1	1	0
0	0	1	1	1
0	0	0	1	0
0	0	1	1	0
0	1	0	0	0

1	0	1
0	1	0
1	0	1

4	3	5
1	4	2
2	3	4
0	3	1

Figure 1: The activation map, after convolving the kernel on the input (stride=1, no padding)

The pooling layer: The convolutional layers can be followed by a pooling layer, which use will be to downsample the activation map (by applying a local max or mean for example, as seen in the Figure 2). The inputs will be the activation maps of the previous layer. The goal of the pooling is to double. The first effect of the dimension reduction is to increase the computation speed (by reducing the number of weights of the following layers). Second, it also reduces the risk of overfitting.

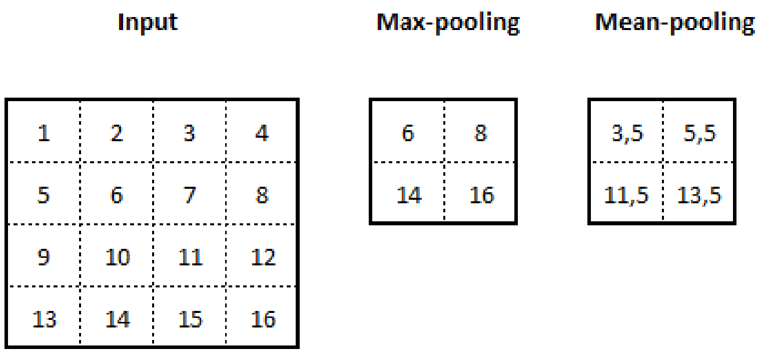


Figure 2: Two kinds of pooling: max-pooling and mean-pooling)

The flattening: We can see that the output of the convolutional layers will be in three dimensions (two spatial dimensions, and one for the number of feature maps). Ultimately, the output of the network for a task like categorization will be a one-dimensional vector. The data from the feature maps thus needs to be flattened to one dimension in order to be applied as the input of the fully connected layer. This is done by the flattening layer.

The fully connected layer: These layers, whose number of weight is $(I + 1) * N$ with I the size of the input and N the number of nodes, are classical layers where each node is connected to all inputs.

2.1.2 Activation Function and Back-Propagation

The activation functions are an important choice of a neural network since they are the functions applied on the output of a node. Without them, or by using a linear activation function, the network would not perform better than a linear regression, which has limited use in the image processing field.

An activation function also has to be differentiable (or at least, piecewise differentiable) to be able to apply the back-propagation algorithm.

The back-propagation algorithm works as follow. First, the network receives a new input, and the desired target $\dot{y}$. From the input x, it computes the output y (this is the feed-forward part). Then, it computes a loss (this loss can be an absolute difference, a square difference, a cross-entropy, etc...) between $\dot{y}$ and y. After that, we back-propagate the error by computing the error on each node (beginning by the output layers) and on each weight of this node.

With w_{ji} the weight from the node i to the node j, x_i an input to the node, y_i the output of the node i, $\dot{y}_i$ the target of the node i, E the loss function, f the activation function, S_i the aggregation function (usually a Cartesian product) of the node i, $\sum_k$ the sum over the nodes of the following layer and η the learning rate, the update equation can be written as:

$$\Delta w_{ji} = \eta x_i \partial_j \text{ where } \partial_j = E(\dot{y}_j, y_j) f'(S_j) = f'(S_j) \sum_k w_{kj} \partial_k [\text{G18}]$$

In the deep learning area, where several layers are sequenced, the activation function is confronted with the issue of vanishing gradient. The reason is that during the minimization of the loss function, the network had to compute the derivative that could exponentially decrease depending on the activation functions of the following layers. This is the reason why, in deep learning, we are not using activation functions like a sigmoid or the hyperbolic tangent but rather ReLU or SELU (Figure 3).

2.2 Regularization techniques

Because of a potentially very large number of weights, several methods are used in deep learning to avoid overfitting. These are called regularization methods.

2.2.1 Dropout

Dropout forces the model to only train on a portion of the network. At each iteration of the training, the nodes of a layer subject to dropout will have some probability (usually 0.5) to be deactivated. This way, the nodes will be dynamically altered, lowering the risk of the network to "learn by heart" the inputs.

On the Figure 3 we can see that the network became very good at classifying the pictures of the training set without dropout, but does not generalize as good on the validation.

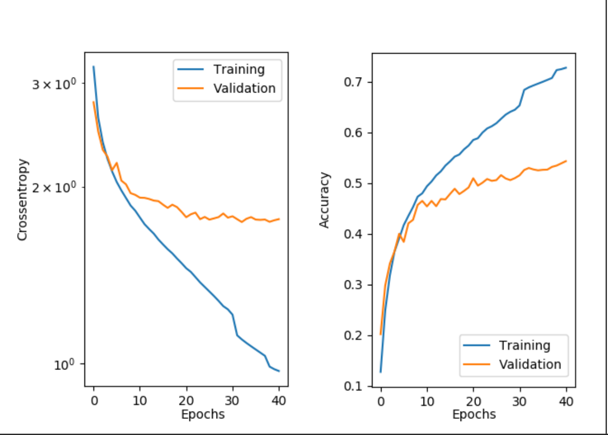
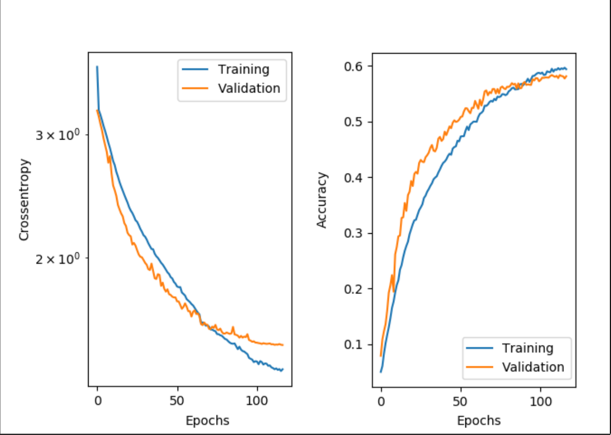
Without dropout	With Dropout
	

Figure 3: Comparison of training without (left) and with dropout (right)

2.2.2 Weight Decay

The weight decay is the decrease of the weights over the time. It has been showed [KH92] that higher weight value will decrease the capacity of the network to generalize. Moreover, this decay can give us the option of trimming the weights if their value is too low, reducing the size of the network.

2.2.3 Noise inputs and online data augmentation

We can also alter the input of the network so that the data on which the CNN works will be different at each iteration. This is done by applying noise (usually white and Gaussian).

It is also possible to alter the pictures each time they enter the network by applying small modifications on them. These modifications can be a zoom, a translation, a rotation, a color variation, an axial symmetry, etc...

2.2.4 Early stopping

To avoid overfitting, we also have the possibility to detect when it happens and stop the training beyond this point. For this, we can estimate the point where overfitting happens by the minimum of the loss function on the validation set.

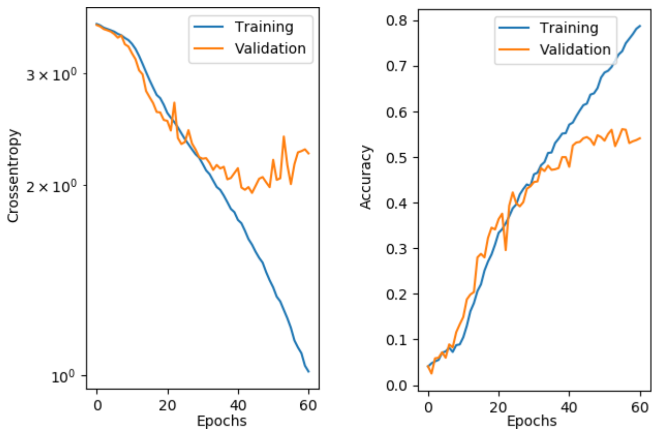
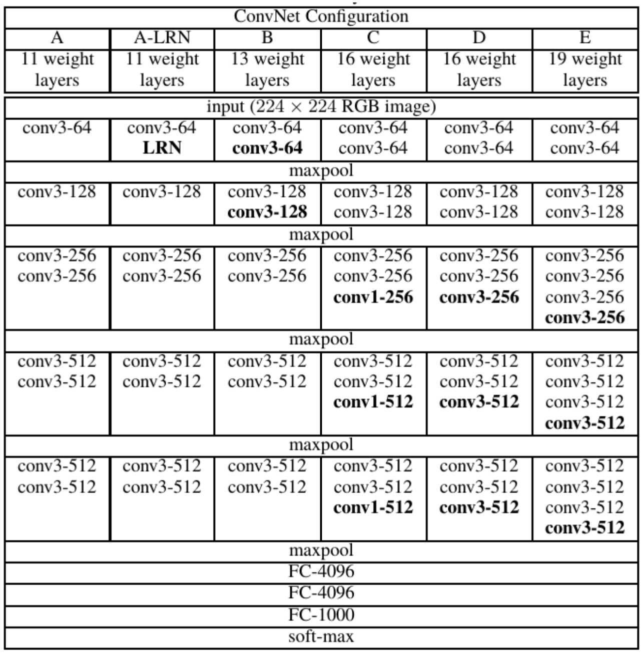


Figure 4: At some point, the validation loss stop decreasing as the network overfit

2.3 VGG, a large but efficient CNN

As the baseline of our first network, we choose VGG [SZ15] to test the networks. It is a very large, but highly efficient structure whose number of weights is between 133 and 144 million depending on the configuration. As we can see in Figure 5, the number of nodes in the last layer, which is the number of labels, is one thousand. Even though, they were able to reach astonishing performance, which is the reason why we choose this network to begin our project.



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 5: Some configurations proposed by VGG

2.4 The boorus, a source of picture for image processing

We think that the website such as the boorus can be an relevant source of datasets. A recent dataset [AtDcBG18] have shown that Danbooru (the original booru), hosting more than three millions pictures, more than hundred of thousands of tags, with a mean of 26 tags per picture, can be used for various goals such as categorization or image generation. As another example, Danbooru was used to design a semantic morphing algorithm [SM17].

As such, we have decided to use this website to design the dataset that we will use during this project.

3 Specification of the datasets

To test the architecture of the networks, we have created two datasets by downloading pictures from the website Danbooru. The pictures should (if they are correctly tagged on the website) share the following characteristics :

- only one persona is present on each picture ;
- the picture is not a manga page (professional or self-published), but must be an illustration ;
- the picture is not monochromatic ;
- the persona is not cosplaying another persona.

The first dataset contains 25 labels, one for each persona, with 58470 pictures. They were chosen based on the number of pictures on the website, and four of them were selected to purposely fool the network. These are presented in the Figure 6.

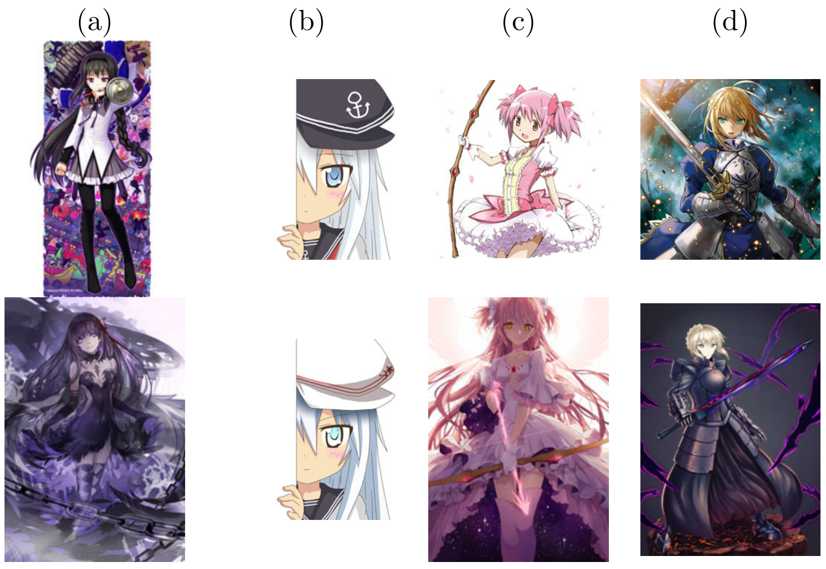


Figure 6: Similar personae chosen to fool the network: Akemi and Akuma Homura (a), Hibiki and Verniy (b), Kaname and Ultimate Madoka (c) and Saber and Saber Alter (d).
The original source of the pictures is indicated by links into the figure.

The second dataset contains 123 labels and 168580 pictures. These personae are those who have more than one thousand images on Danbooru which meet the requirements, they were not chosen on any other criteria.

Each of these two datasets has a second selection in which the background of all pictures is white, this reduces greatly the number of pictures for each label, and therefore is only used for tests.

The pictures are downloaded in the form of thumbnails, with a maximum dimension of 150x150 pixels to decrease the load on the internet, reducing the size of the dataset to 585 MB and 1.59 GB respectively. They enter the network with a size of 128x128 pixels, the nearest power of two.

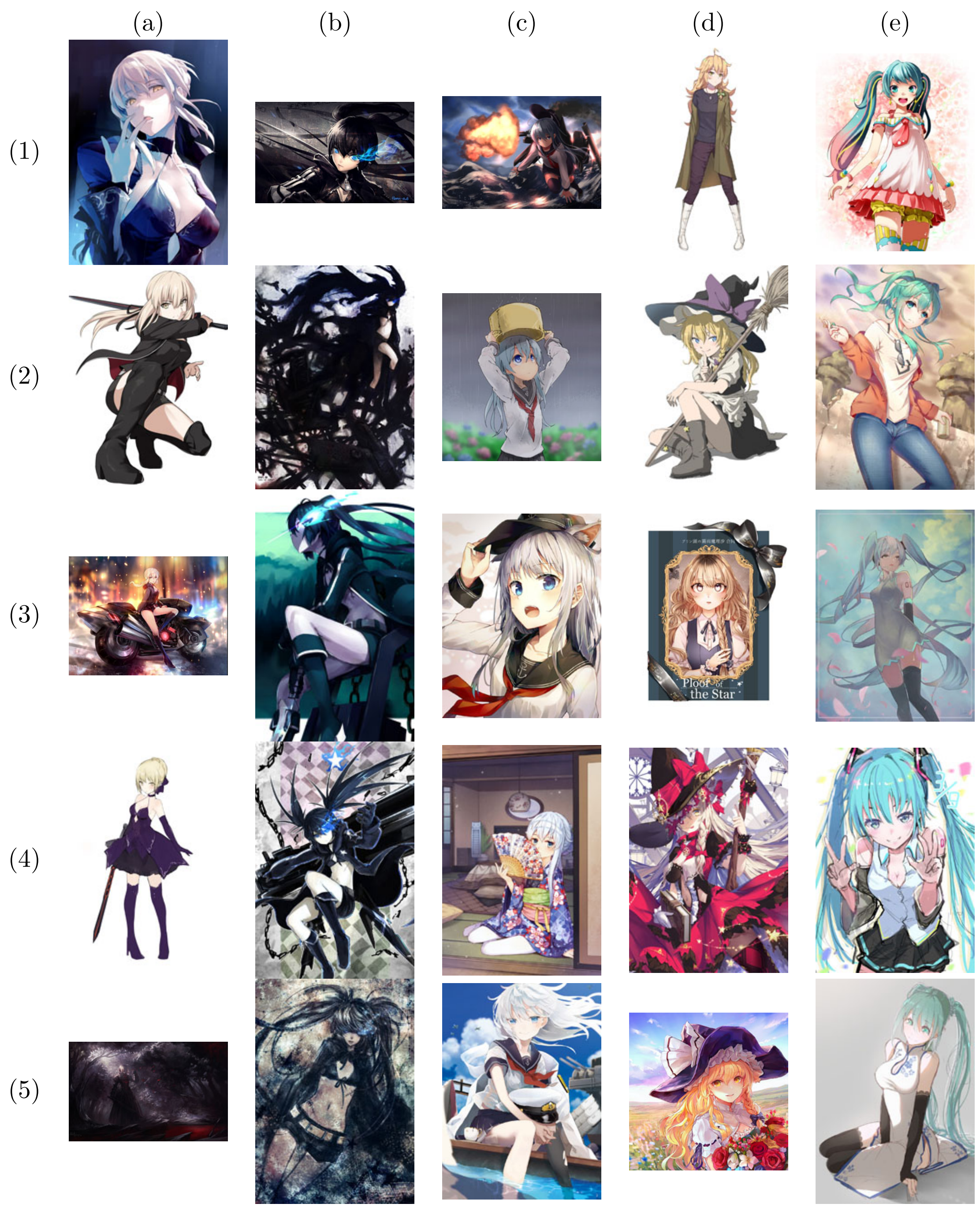


Figure 7: Dataset sample for the personae Saber Alter (a), Black★Rock Shooter (b), Hibiki (c), Kirisame Marisa (d) and Hatsune Miku (e)

As we can see, it seems to be a difficult problem for several reasons:

- the original pictures are usually quite big (more than 10% of the original pictures being more than 3200 pixels wide or 2400 pixels tall), the downscale will thus lead to major detail loss ;
- the heterogeneity (variety of clothes, of hairstyle, of composition, etc..) inside a class can be high for some personae (Hatsune Miku for example) but almost inexistent for another one (Black★Rock Shooter) ;
- the artstyle can be anything from photorealistic to sketches.

4 Two architectures of CNNs

4.1 Flat architecture

Initially, the CNN that we used was the VGG [SZ15] network. We have progressively reduced its size until we reached the network described in Figure 12 without any performance loss. With this model, the number of weights was decreased from one hundred million to two million, making both the occupation of the memory and the computation time lower.

In addition, we have used the following hyperparameters :

- optimizer: SGD with Nesterov momentum [Nes83];
- learning rate: 10^{-3} ;
- momentum: 0.5 ;
- batch size: 128 ;
- learning rate decay: 10^{-5} ;
- weight decay: 4.10^{-5} ;
- dropout probability: 50% ;
- noise input standard deviation: 0.01 ;
- activation function: ReLU ;
- loss function: crossentropy.

This configuration is the baseline that we will use for all the CNN of the clustered architecture that we will describe now.

4.2 Hierarchical architecture

4.2.1 General Idea

Lets suppose that we have four labels l_1, l_2, l_3 and l_4. We also suppose that there are some kinds of features shared between l_1 and l_2, and between l_3 and l_4, leading the members of these two doublets to be more frequently misclassified with the other members than with the members of the other doublet. The idea of the clustered architecture is to begin by recognizing the doublet, then the label. For example, with comic's heroes, we could have the Figure 8 (with l_1 as Batman, l_2 as Catwoman, l_3 as Iron Man and l_4 as Spider-man). A picture has first to enter in the root, where it would be decided whether it enter in the network specialized for black-clothed characters or red-clothed ones. In this diagram, the CNNs are represented by rectangular nodes while the labels are inside ellipses.

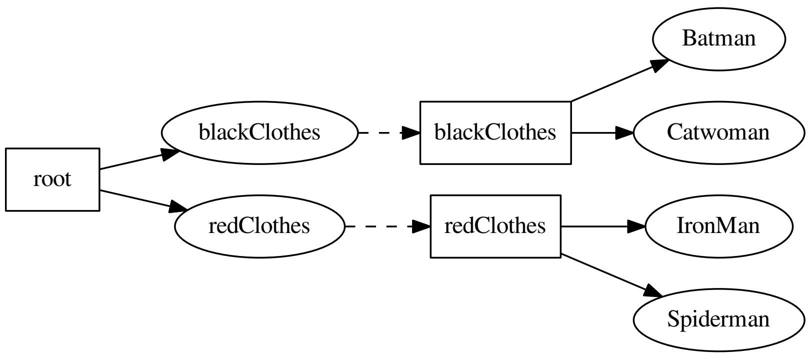


Figure 8: A small cluster architecture

In this case, the first CNN will conclude about the clothing of the hero, direct the picture to a specialized CNN which will decide about the final label. We make the assumption that, using this method, we can make smaller networks, making them computable by a smaller GPU, with the same performance as the bigger, but flat, ones.

In the subsequent part of the report, we will call "root" the root of the architecture (ie the first network in which the input are fed) and "subnetworks" all the other networks.

We think that such an architecture will be more flexible because it enables to extend the set of labels and to merge several architectures. To extend the set of labels, we have to add branches on the architecture tree, this can be done by retraining severals CNNs but it will still be only a part of all of them. Merging the architectures can be done by considering the root of each as a subnetwork (which means that we consider each of the architecture as a cluster on its own), and differentiate them inside a newly trained root.

4.2.2 Clusterization

Noting l the label associated with a picture, d the detection from the network and $\{L_n\}_{n \in [\![1,N]\!]}$ a series of N labels, the confusion matrix is the matrix M with $M_{i,j} = \mathbb{P}(d = L_j | l = L_i)$. We can deduce easily that we have $\sum_{j=1}^{N} M_{i,j} = 1 \ \forall i \in [1, N]$. Increasing the quality of the categorization can be seen as getting closer to 1 on the diagonal, which is equivalent to have 0 on any other cell.

To decide about the method used for the clusterization, and based on our assumption made in the previous part, we have to find the clusters C maximizing the quantity $\frac{1}{N} \sum\limits_{C} \sum\limits_{d,l \in C^2} \mathbb{P}(d|l)$. This quantity is the probability to be classified in the right cluster, we will call it MICP (Mean In-Cluster Probability) in the rest of the document, while there is no *a priori* on the cardinal of the clusters other that they form a partition of the set of labels.

To solve this problem, commonly referred as "community detection", we use an implementation of the algorithm proposed by Mark Newman [New13], using the confusion matrix depicted in the Figure 9.

Of course, we can repeat the operation of clusterization after the formation of the first layer of clusters to create a multi-layered architecture.

About the size of the clusters

If a network is trained with a different number of pictures in some classes, its predictions will be biased in favor of the label with the most data. This leads us to carefully balance the size of the clusters. Assuming that we train on the same number of pictures in each class, a too large asymmetry will have an effect on the total number of picture that we can use for the training. If the labels are unbalanced but that we want to train all labels on the same amount of picture, we have to specify that the number chosen for training in each class have to be less than the size of the smallest class. Alternatively, we can inject knowledge about the distribution of these classes in the real world (if a persona is overly more popular than another, the network should favor the first one), but this is an *a priori* that we want to avoid.

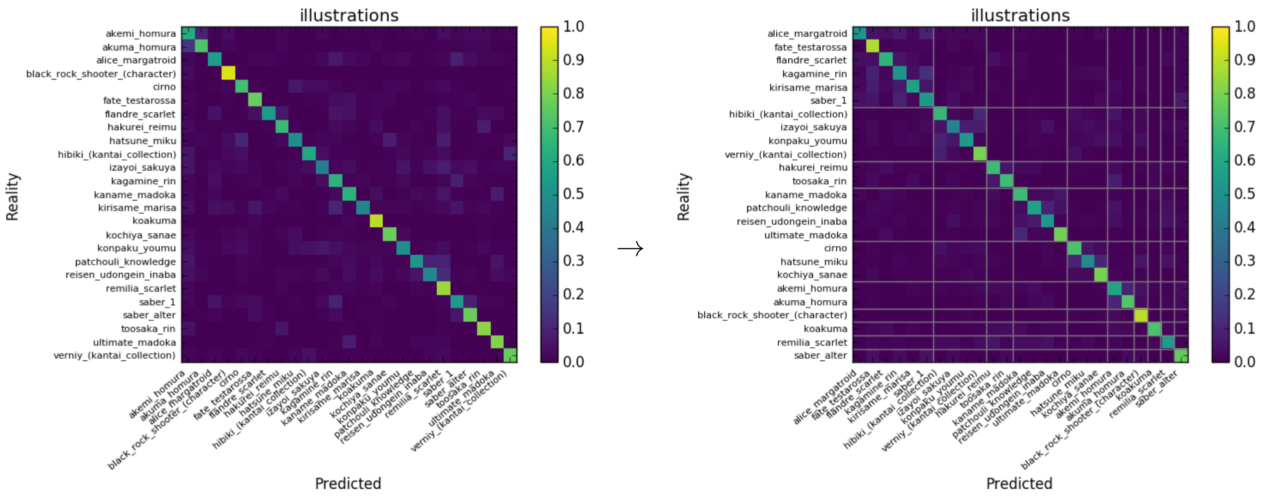


Figure 9: The labels are reordered to form clusters (ideally, in a block diagonal matrix)

4.2.3 Training specifications

The first network to be trained can use the entire dataset. Thus, the quantity of data at its disposal is several times larger than the ones of the inner networks. For example, using 50% (for computational reasons and equity between the networks) of all pictures on each training (on the 25 labels dataset, 20% on the 123 labels dataset), the root of the first dataset will train on almost thirty thousands pictures, while in a in-depth network like StayNight (Figure 13) will only have a little more than two thousand illustrations.

To prevent this reduction of the training data to affect negatively the architecture, we use two mechanisms: data augmentation and fine-tuning.

Data augmentation is the process to alter the pictures on which the CNN is trained at each epoch. The pictures can be rotated (from -20° to 20°), zoomed-in (with a maximum of 120%), shifted on either ax (up to 20% of the picture dimension) or be flipped on the vertical axis.

Fine-tuning concerns the re-utilization of a previously trained network as a basis for a new one. In our case, the root has been trained on all pictures of the training set, we can suppose that its convolution layers have learned the features that would be needed for the categorization. Thus, it is not necessary to train the convolution layers of the other networks: we only have to reuse what we have previously trained. This leads to an important increase of training speed and even an improvement of the success rate.

The training makes use of early stopping to prevent overfitting, waiting for a period of eight epochs without decrease of the validation loss to stop. The selected weights are those with the minimum validation loss. The number of eight epochs can be seen as relatively high but is conditioned to another feature of the training: the reduction of the learning rate on plateau. Each time that the validation loss doesn't decrease during four epochs, the learning rate is divided by two.

5 Results and analysis

5.1 Performance

5.1.1 The Mean Success Rate

The main indicator of performance that we use is the Mean Success Rate, which is the mean probability for an image to be classified correctly. Using this indicator, and as we can see on the Figure 1, the use of the hierarchical structure does not improve the performance. On the contrary, the MSR is degraded most of the time. Moreover, the depth of the structure degrades the performance even more.

We also computed the MSR on the picture with a white background to confirm that the network was not learning the background's patterns. A picture with a white background will often have a simpler composition (images such as 4a, 5a or 1c in the Figure 7 are problematic) and thus is easier to decide of a persona. Only one persona ("Black★Rock Shooter") was more often wrong-classified. The reason is that this particular persona is represented over a checkered background most of the time, leading the network to learn this checkered pattern instead of the persona's features.

5.1.2 The Mean In-Cluster Probability

A second indicator is the Mean In-Cluster Probability that we have used in the clustering process. We do not compute it for the deep cluster because its cluster number is very different to the other architectures (two clusters at the root and eleven final clusters) and depends on the depth that we consider.

We can draw several conclusions from the MICP. First, the use of the hierarchical architecture does not improve it, which would have been logical to assume so. Second, even if the MICP is higher in the clustered version, the MSR is not significantly larger (in the case of the 123 labels dataset, it decreases). Whereas the previous statement was related to the quality of the root network, the last one shows that the last networks of the structure also have performance issues.

Table 1: Mean Success Rate (MSR) for each network and dataset and the Mean In-Cluster Probability (MICP)

Architecture	Dataset	MSR	MICP	MSR (white background)
flat	25 labels	65.3%	77.8%	71.6%
hierarchical	25 labels	66.0%	77.0%	67.5%
deep hierarchical	25 labels	57.9%	NA	61.2%
flat	123 labels	42.5%	64 0%	46.1%
hierarchical	123 labels	41.0%	68.6%	46.0%

We can look more precisely into the details of the prediction by subtracting the confusion matrix of the hierarchical and flat architectures. On the Figure 10 the colorbar is rescaled to values in [-0.3, 0.3] to highlight the differences. The improvement (respectively decrement) of the quality of the categorization is shown by a positive (respectively negative) value on the diagonal and a negative (respectively positive) value on any other cell. We can draw two conclusions from this illustration.

First, the classes in single-label clusters have a tendency to increase their probability of success to the detriment of the others labels. This can be mitigated by a strict application of the equality of the clusters training set. In the Figure 10, this is quite flagrant for the persona "remilia_scarlet" and is a factor of the decrease of the categorization's quality on few others personae.

Second, whereas the diminution of the categorization seems to be mainly due to the quality of the root (a blue diagonal cell is accompanied by light green cells on the same row, not necessarily in the same cluster), the increase of the performance is connected to a decrease of the in-cluster miss-categorization. However, these observations don't validate our hypothesis since the overall MSR has not increased.

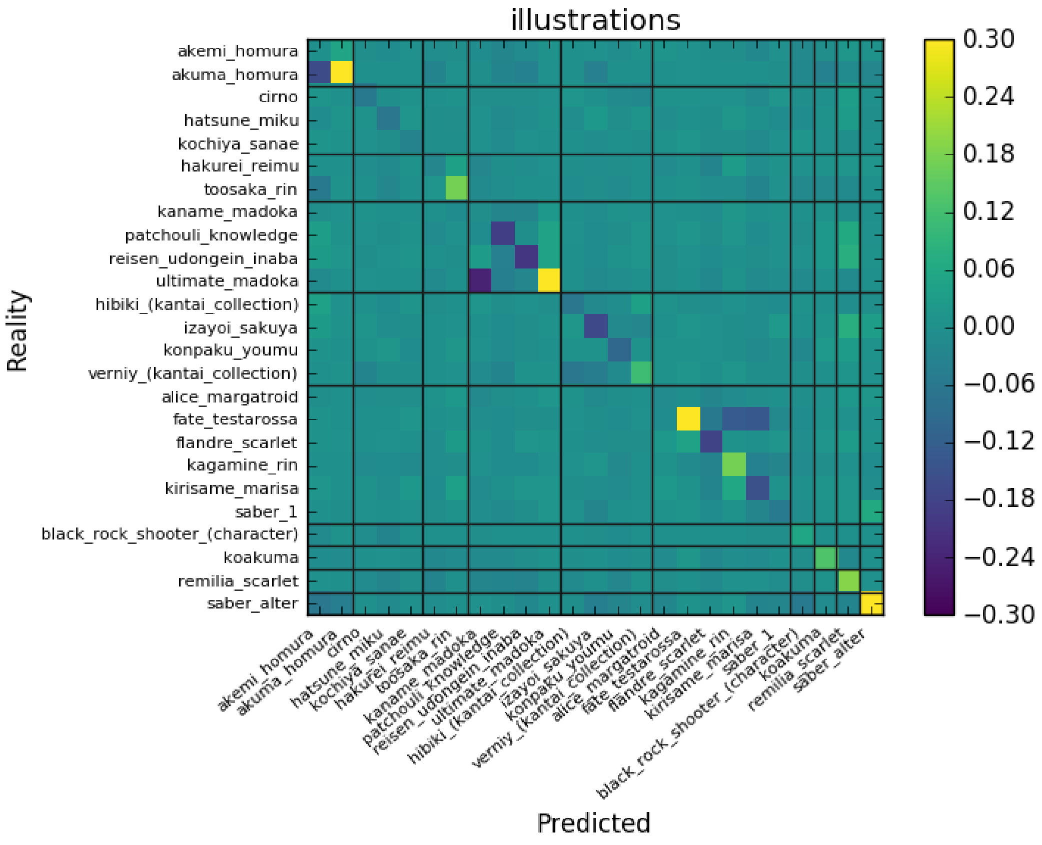


Figure 10: Subtraction of the hierarchical and flat confusion matrix

From these results, we conclude that the different classes of the root have to be drastically different to allow it to have near perfect results. This was not the case in our datasets, restricting the usefulness of the architecture. Nonetheless, we think that a slight increase of the MSR could be obtained by a more skillful clustering.

5.2 Computation time

Table 2: Training duration (in seconds) of each architecture

# labels	Flat	Root	Subnetworks (mean | max)	Total time (parallelized)
25 labels (depht=1)	9900	3900	470 | 720	6620 (4620)
25 labels (depht=5)	9900	2220	406 | 1500	9528 (3720)
123 labels	9451	3540	805 | 1476	8370 (5016)

The training is quicker with the hierarchical architecture, despite the need to compute a larger number of CNNs. Also, the mean time of the subnetworks training at a high depth is heavily impacted by the configuration of the early stopping. Indeed, these networks inside this structure only need between ten and twenty epochs to converge. Thus, the discarded epochs of the early stopping (the last eight with our configuration) can represent 50% of the computation time, whereas, on the flat architecture, the models need more than sixty epochs, leading these discarded epochs to be less than 10% of the total computation time.

An advantage of the hierarchical architecture is that the trainings of the subnetworks can be done on different machines and only need the root because of the fine-tuning. By parallelizing, the total training duration of the architecture is only the sum of the time needed for the root and the largest subnetwork.

However, the prediction of the hierarchical architecture is slower than the prediction of the flat one, because an image has to pass successively in several networks. Besides, since all the networks have to be loaded in the memory, it is more efficient to create batches of images, which will reduce the time dedicated to the network load.

On a side note, this hierarchical architecture is also more flexible since that, if we want to add some classes in the categorization, it could be achieved easily by retraining only the root and the appropriate cluster.

5.3 Effect of the hyper-parameters

During this project, we have tried severals hyperparameters to design the network that we finally use. We will now discuss the reason for not choosing them in the final network.

5.3.1 SELU or ReLU ?

Alternatively to ReLU, it is possible to use SELU[KUM17] as an activation function.

It has the advantage of converging (on the validation loss) quicker but gets slightly lower results in the categorization.

Table 3: SELU and ReLU

Activation Function	Derivative
$SELU(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}$	$d_x SELU(x) = \begin{cases} \lambda & \text{if } x > 0 \\ \lambda\alpha e^x & \text{if } x \leq 0 \end{cases}$
$ReLU(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$	$d_x ReLU(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$

We can note that $d_x SELU(x)$ is not monotonous if $\alpha > 1$, since $\lim\limits_{x \to 0^-} d_x SELU(x) = \lambda\alpha$ whereas $\lim\limits_{x \to 0^+} d_x SELU(x) = \lambda$

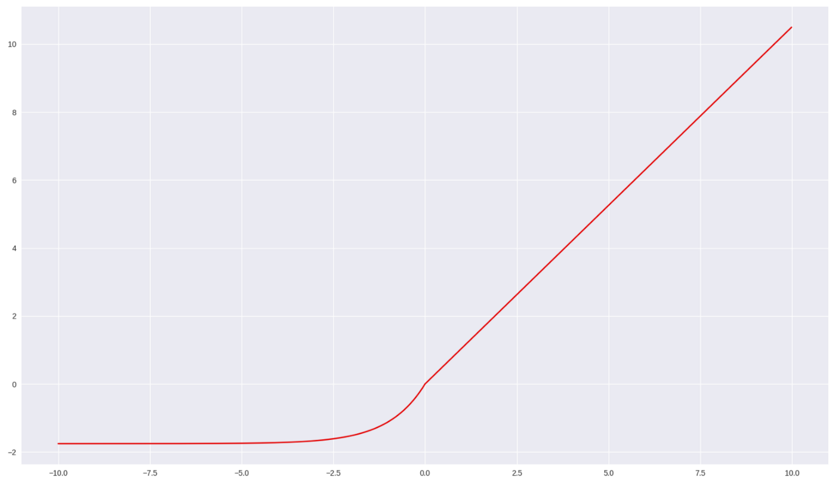


Figure 11: SELU with $\alpha = 1.6732, \lambda = 1.0507$

5.3.2 Smaller networks ?

We have also tried to reduce the number of weights (2285721 in the final version of the network) to values around two hundred thousand. These modifications were discarded because of an observed diminution of the performance rate, even if such networks were able to converge quicker, meaning with a lower training computation time.

5.3.3 Adam or SGD ?

As indicated previously, we have chosen to use a version of SGD with Nesterov momentum. We have also try to use another optimizer called Adam (Adaptative Momentum), but this method had been ultimately discarded because the time required to converge was higher and that the performances were lower than using SGD. In the Table 4 are the update rules for several optimizers, we can see that Adam requires three parameters, against two for SGD with Nesterov momentum, making the latter easier to adjust quickly.

Table 4: Different kind of optimizers [Rud16]

Optimizer	Parameters	Update rule's $\delta_t, \delta_0 = 0$
SGD	η	$\delta_t = -\eta g_t$
SGD with momentum	m, η	$\delta_t = m\delta_{t-1} - \eta g_t$
SGD with Nesterov momentum	m, η	$\delta_t = m^2\delta_{t-1} - (1+m)\eta g_t$
Adam	$\beta_1\beta_2, \epsilon$	$\delta_t = -\frac{\eta}{\sqrt{\frac{v_t}{1-\beta_2^t}+\epsilon}} \frac{m_t}{1-\beta_1^t}$
		$m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t, m_0 = 0$
		$v_t = \beta_1 v_{t-1} + (1-\beta_2)g_t^2, v_0 = 0$

With g_t the gradient at the step t.

6 Discussion and conclusion

Aiming at increasing the quality of the categorization of personae, our project took the approach of designing a hierarchical architecture of CNNs and testing it against a flat architecture. We hoped that, if the hierarchical structure was performing better, it could lift some constraints of both training time and memory space.

To be able to create this architecture, we needed a method to form clusters i.e. grouping class often miss-classified with each other and maximizing the Mean In-Cluster Probability. We used the Spectral Community Detection algorithm and were able to design several architectures with varied depths and numbers of labels, using pictures from the website Danbooru as datasets.

The results of this architecture show us that this architecture is not as efficient as we would think since the Mean Success Rate was not highly affected, but we were able to observe small variations, either negative or positive, depending on the dataset. However, a high-depth structure has to be avoided since it decreases the success rate substantially.

We have also concluded that a hierarchical architecture could be used to decrease the computation time of the training.

Further works could be required to study different methods to design the clusters, as they are critical in the performance of the architecture. It could also be interesting to leave a fully hierarchical network of CNNs (forming a tree) to implement a kind of rollback allowing a CNN to detect a picture which has followed a wrong branch.

References

[AtDcBG18] Anonymous, the Danbooru community, Gwern Branwen, and Aaron Gokaslan. Danbooru2017: A large-scale crowdsourced and tagged anime illustration dataset. `https://www.gwern.net/Danbooru2017`, January 2018. Accessed: 2018-05-22.

[G18] Olle Gllmo. Machine learning, lesson 4, 2018.

[HW59] David Hubel and Torsten Wiesen. Receptive fields of single neurones in the cat's striate cortex, 1959.

[KH92] Anders Krogh and John Hert. A simple weight decay can improve generalization. *Advances in Neural Information Processing Systems*, 4:950–957, 1992.

[KUM17] Gnter Klambauer, Thomas Unterthiner, and Andreas Mayr. Self-normalizing neural networks. 2017.

[Nes83] Yurii Nesterov. A method of solving a convex programming problem with convergence rate $o(1/k^2)$. *Soviet Mathematics Doklady*, 27:372–376, 1983.

[New13] Mark Newman. Spectral methods for network community detection and graph partitioning. 2013.

[Rud16] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[SM17] Masaki Saito and Yusuke Matsui. Illustration2vec: A semantic vector representation of illustrations. 2017.

[SZ15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. 2015.

7 Appendix

7.1 Annex 1 - Configuration of the network

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
B1C1 (Conv2D)                (None, 128, 128, 64)      1792
_________________________________________________________________
B1P (MaxPooling2D)           (None, 64, 64, 64)        0
_________________________________________________________________
B2C1 (Conv2D)                (None, 64, 64, 128)       73856
_________________________________________________________________
B2P (MaxPooling2D)           (None, 32, 32, 128)       0
_________________________________________________________________
B3C1 (Conv2D)                (None, 32, 32, 128)       147584
_________________________________________________________________
B3C2 (Conv2D)                (None, 32, 32, 128)       147584
_________________________________________________________________
B3P (MaxPooling2D)           (None, 16, 16, 128)       0
_________________________________________________________________
B4C1 (Conv2D)                (None, 16, 16, 128)       147584
_________________________________________________________________
B4C2 (Conv2D)                (None, 16, 16, 128)       147584
_________________________________________________________________
B4P (MaxPooling2D)           (None, 8, 8, 128)         0
_________________________________________________________________
B5C1 (Conv2D)                (None, 8, 8, 128)         147584
_________________________________________________________________
B5C2 (Conv2D)                (None, 8, 8, 128)         147584
_________________________________________________________________
B5P (MaxPooling2D)           (None, 4, 4, 128)         0
_________________________________________________________________
gaussianNoise (GaussianNoise (None, 4, 4, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
fc21 (Dense)                 (None, 512)               1049088
_________________________________________________________________
do2 (Dropout)                (None, 512)               0
_________________________________________________________________
fc51 (Dense)                 (None, 512)               262656
_________________________________________________________________
do3 (Dropout)                (None, 512)               0
_________________________________________________________________
predictions25 (Dense)        (None, 25)                12825
=================================================================
Total params: 2,285,721
Trainable params: 2,285,721
Non-trainable params: 0
_________________________________________________________________
```

Figure 12: CNN Configuration, configuration of 25 labels

7.2 Annex 2 - Hierarchical Structure Graph (25 labels dataset), depth = 5

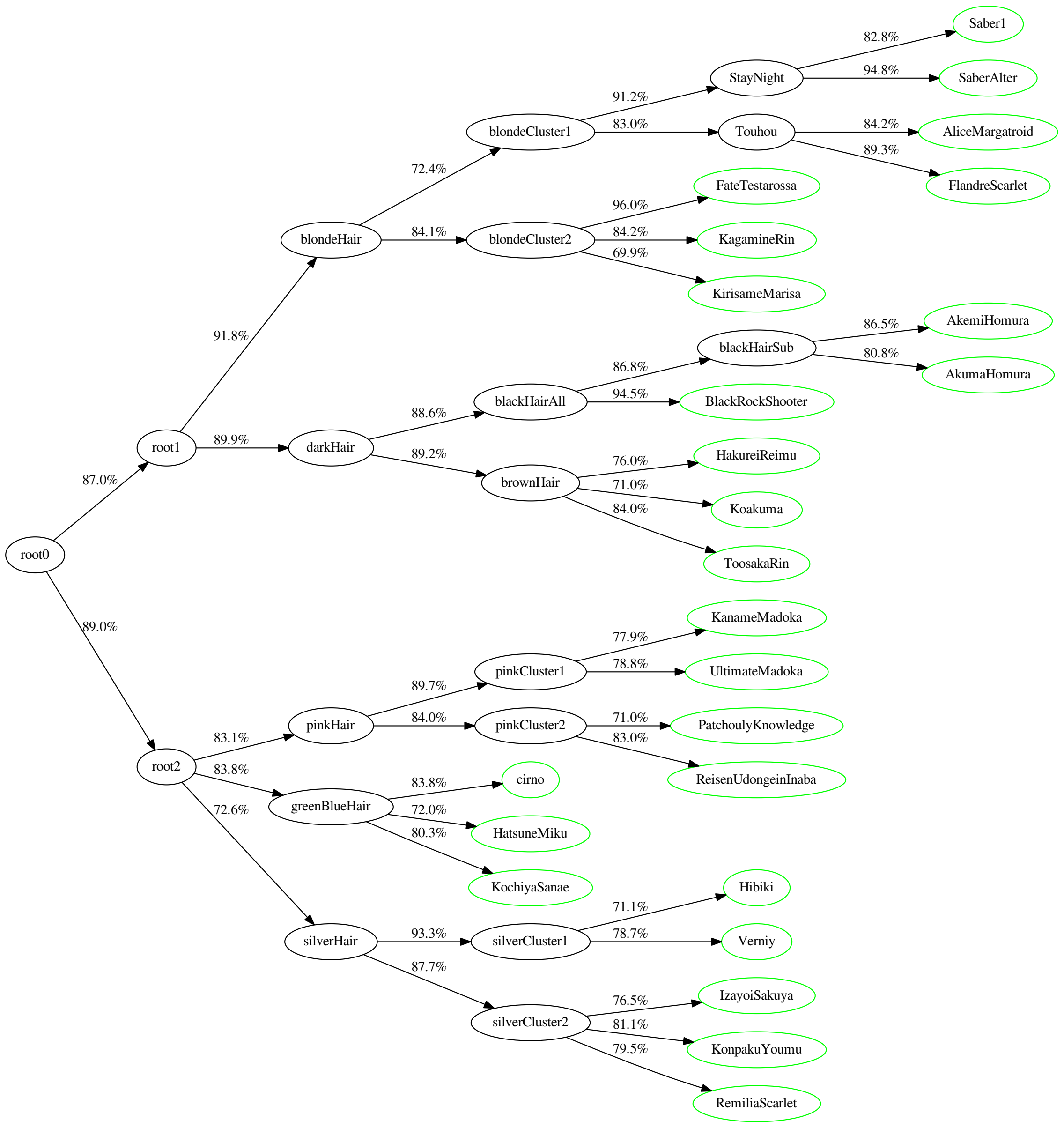


Figure 13: Hierachical structure and success rate associated to each network

7.3 Annex 3 - Hierarchical Structure Graph (25 labels dataset), depth = 2

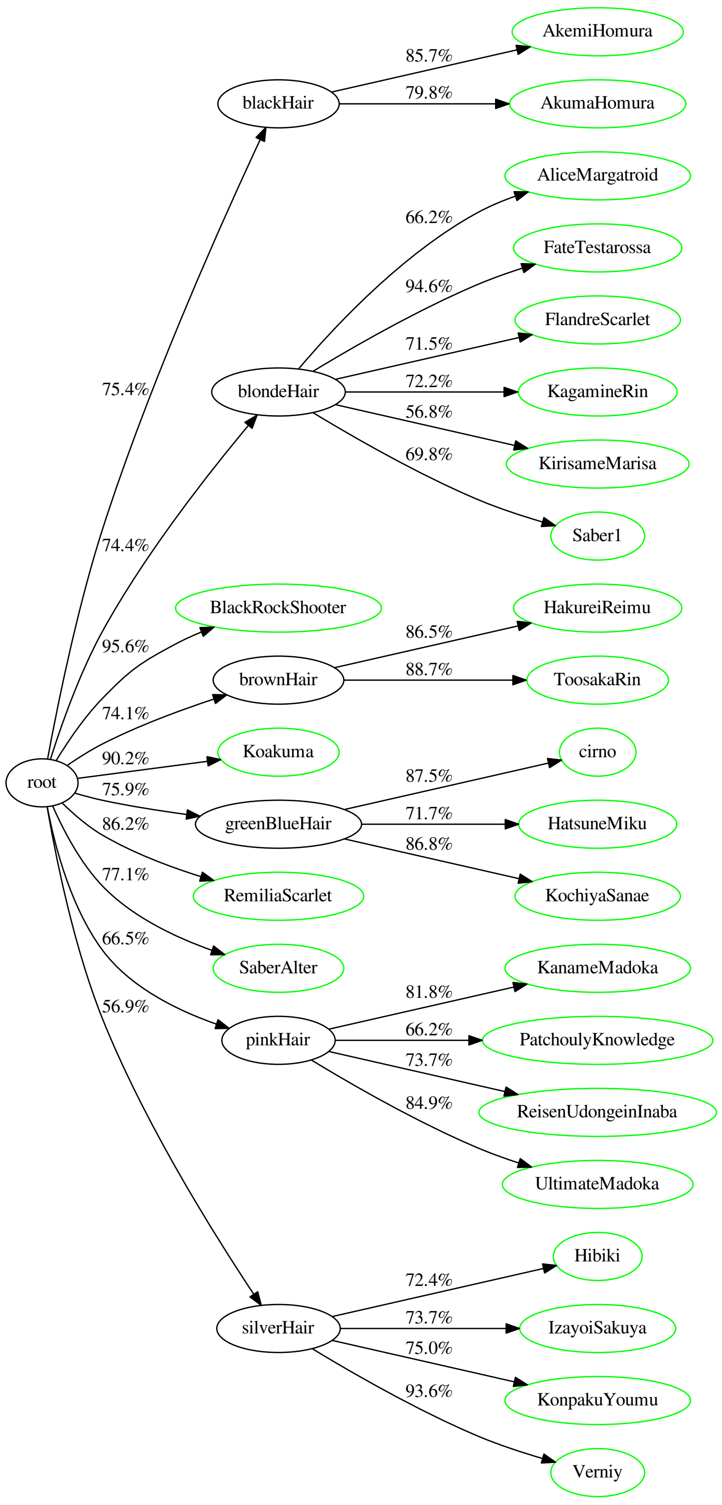


Figure 14: Hierarchical structure and success rate associated to each network

7.4 Annex 4 - Hierarchical Structure (123 labels dataset)

Table 5: Hierarchical structure

root	cluster 0	nakano_azusa, akemi_homura, akiyama_mio black_rock_shooter_(character), d.va_(overwatch) fubuki_(kantai_collection), ganaha_hibiki haruna_(kantai_collection), hestia_(danmachi) houraisan_kaguya, kaga_(kantai_collection) kikuchi_makoto, murasa_minamitsu ryuujou_(kantai_collection), shameimaru_aya shibuya_rin, shigure_(kantai_collection) takagaki_kaede, tenryuu_(kantai_collection) toujou_nozomi, yazawa_nico, reiuji_utsuho
	cluster 1	kaku_seiga, kawashiro_nitori, kisaragi_chihaya, ayanami_rei, komeiji_koishi, konpaku_youmu, miki_sayaka cirno, sonoda_umi, tatara_kogasa, wriggle_nightbug
	cluster 2	fujiwara_no_mokou, hata_no_kokoro hibiki_(kantai_collection), hinanawi_tenshi inubashiri_momiji, izayoi_sakuya jeanne_d'arc_(alter)_(fate), kamishirasawa_keine kashima_(kantai_collection) mononobe_no_futo, murakumo_(kantai_collection) nagae_iku, nazrin, patchouli_knowledge, rem_(re-zero) remilia_scarlet, shijou_takane, suigintou yagokoro_eirin, yasaka_kanako
	cluster 3	hatsune_miku, kagiyama_hina, kasodani_kyouko kazami_yuuka, kochiya_sanae, nero_claudius_(fate) gumi shiki_eiki, suzuya_(kantai_collection)
	cluster 4	alice_margatroid, atago_(kantai_collection) ayase_eli, clownpiece, flandre_scarlet hamakaze_(kantai_collection), hijiri_byakuren hoshiguma_yuugi, hoshii_miki, ibuki_suika kagamine_rin, kirisame_marisa, kurodani_yamame megurine_luka, mizuhashi_parsee, moriya_suwako northern_ocean_hime, prinz_eugen_(kantai_collection) ro-500_(kantai_collection), rumia, saber_alter saber_true, shimakaze_(kantai_collection) super_sonico, tamamo_no_mae_(fate), tomoe_mami toyosatomimi_no_miko, yakumo_ran yakumo_yukari, yuudachi_(kantai_collection)
	cluster 5	amami_haruka, chen, hakurei_reimu himekaidou_hatate, hirasawa_yui, hong_meiling imaizumi_kagerou, inaba_tewi, inazuma_(kantai_collection) kaenbyou_rin, kaname_madoka, kongou_(kantai_collection) minami_kotori, minase_iori, misaka_mikoto souryuu_asuka_langley, suzumiya_haruhi
	cluster 4	houjuu_nue, ibaraki_kasen, kanzaki_ranko koakuma, komeiji_satori, mash_kyrielight mystia_lorelei, nagato_yuki, nishikino_maki onozuka_komachi, reisen_udongein_inaba saigyouji_yuyuko, sakura_kyouko, toosaka_rin