École polytechnique fédérale de Lausanne

Master project

Detection of strong lensing with convolutional neural networks

Abstract We participate to the Gravitational Lens Finding Challenge with a convolutional neural network and win the first place for one of the four categories out of 14 participants. We also develop a neural network architecture invariant under the dihedral group.

Mario Geiger

supervised by
Ph.D. Christoph Schäfer & Prof. Jean-Paul Kneib

April 2, 2017

Contents

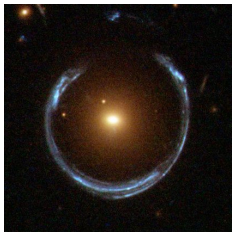


Figure 1: LRG 3-757 captured by the Hubble Space Telescope

Acknowledgements

I would like to thank Christoph Schäfer, Thibault Kuntzer and Jeanne Colbois for helping with the project. And also Jean-Paul Kneib for giving me this master thesis.

1 Introduction

Mass bends the light trajectory. Because of this, a cosmological object that lies behind an other one can be seen distorted and magnified. This effect is called *strong gravitational lensing* when it produces multiple images, arcs and even rings when the two objects are perfectly aligned in the line of sight (see figure 1). The strong gravitational lensing can be used as a natural telescope to observe the lensed object or to probe the mass distribution of the lensing object. Also, it has been discovered recently that the lensing of quasars can be used to determine the Hubble constant (Bonvin et al., 2017).

The aim of this project was to create an algorithm able to detect the presence of a strong gravitational lens on an image in which both the lens and the lensed object are galaxies. And to participate to the *Gravitational Lens Finding Challenge* (Metcalf, 2016) with this algorithm.

Convolutional neural networks are neural networks specialized in the analysis of images. In 2012 a convolutional neural network outperformed the state of the art algorithms for images recognition (Karpathy, 2016). In this report we present our approach to construct a neural network which is invariant under the rotations (multiples of 90 degrees) and mirrors of the input images, and compare it to a baseline algorithm.

Section 2 introduces neural networks and presents ideas which are applied in section 3. Section 3 proposes some implementations. Section 4 present and discusses the results.

2 Theory

A neural network (NN) is a generic function that has to be fitted over data. The usual steps for using a NN for a classifier are the following:

1. Collect data (or create it with simulations) and describe it using labels (for instance a number characterizing the presence or absence of a lensing effect).

2. Train the NN to predict the correct label given the data

Neural networks excel in image recognition, for instance the deep neural network "Inception" of Google is able to classify images among 1000 classes (it is able the distinguish between the Siberian husky and the Eskimo dog) (Szegedy et al., 2014). It has also be used in astrophysics to classify galaxies (gal, 2014). In this project the raw data was images of size 101×101 and the labels were 0 or 1, corresponding respectively to the absence or the presence of a strong gravitational lensing effect.

2.1 General structure of a neural network

A convolutional neural network (CNN) is a special case of a neural network (NN) in which we use convolutional layers (described later). A neural network is a complicated mathematical function (1) which takes in argument

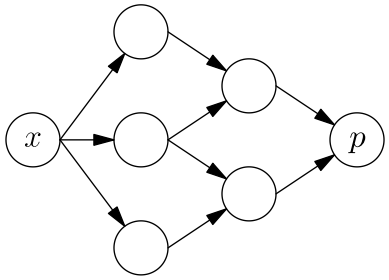


Figure 2: The graph of a neural network. The nodes represent the layers and the vertices represent the tensors.

some data x and lots of parameters W called *weights*.

$$f(x, W) \tag{1}$$

It is the composition of simpler functions called layers. The section 2.3 introduces the layers used in this project. The data (intput, output and intermediate) in a NN are called *tensors*, i.e. the input and the output of each layer is a tensor (see 2). The way the NN is built up is called its *architecture*. The output of the NN is the prediction which gets compared to the labels.

2.2 Training

The aim of the training is to optimize W such that $f(x, W) \approx \lambda$, where λ is the label of x. The set of couples (data, label) used to perform the training are called the *training set*.

2.2.1 Cost function

The cost function quantifies the accuracy of the NN. It is the quantity with respect to which W is optimized during the training. It is a function of the predictions $\{p\}$ and of their corresponding labels $\{\lambda\}$: $H(\{p\}, \{\lambda\})$. This function must have both the properties that is gets smaller if the predictions get better and that it is differentiable.

The most common cost function for a classifier is the *cross entropy*. It is defined for a single couple (data, label) as

$$h(p, \lambda) = \begin{cases} -\log(p) & \text{if } \lambda = 1 \\ -\log(1-p) & \text{if } \lambda = 0 \end{cases}$$

Then the total cost function is simply the mean over the entries (2).

$$H(\{p\}, \{\lambda\}) = \frac{1}{n} \sum_{i=1}^{n} h(p_i, \lambda_i) \tag{2}$$

2.2.2 Gradient descent

To minimize the cost function, the algorithm 1 is used. It is based on the fact that the NN is a mathematical function that can be derived with respect to its weights. Therefore the cost function can be minimized following the direction opposite to the gradient.

Data: training set (x, λ)
Result: weights W that minimize the cost-function $H(p, \lambda)$
initialize W randomly;
while *cost-function not small enough* **do**
  compute $g \longleftarrow \nabla_W H(\{x\}, \{\lambda\})$;
  update $W \longleftarrow W - \eta g$;
end

Algorithm 1: Gradient descent

where η is a parameter called the *learning rate*. If this parameters is too small, the training is slow, but if it is too high, there are risks of instabilities.

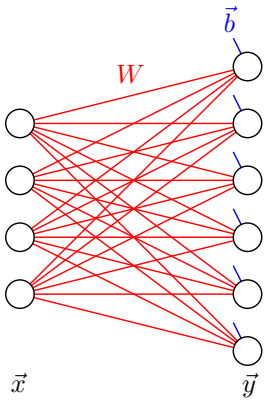


Figure 3: Fully-connected layer

2.2.3 Stochastic gradient descent

The stochastic gradient descent (SGD) algorithm 2 is the same as the gradient descent algorithm except that it uses an approximation of the gradient. This approximation is made by using a randomly chosen subpart of the training set, called *mini-batch*, instead of the full training set to compute the gradient.

Data: training set (x, λ)
Result: weights W that minimize the cost-function $H(p, \lambda)$
initialize W randomly;
while *cost-function not small enough* **do**
  pick randomly a mini-batch $(x', \lambda') \subset (x, \lambda)$;
  compute $g \longleftarrow \nabla_W H(\{x'\}, \{\lambda'\})$;
  update $W \longleftarrow W - \eta g$;
end

Algorithm 2: Stochastic gradient descent

2.2.4 Validation set and training set

The value of the cost function over all the training set tells how good the NN can predict the labels of these data. To determine whether the prediction of the NN can be generalized to other data it is a good practice to keep apart some data (the *validation set*), and not train with these data. We have two scores: the cost function of the validation set and the cost function of the training set. When the NN fits too well the training set and does not generalize anymore to other data, we say the NN *overfits*. The risks of overfitting can be detected with the validation set.

2.3 Layers

2.3.1 Fully-connected layer

The fully-connected layer is described in equation (3)

$$y_i = \sum_j x_j W_{ji} + b_i \tag{3}$$

where x is the input vector, y is the output vector, W is a matrix and b a vector. W and b are the weights of the layer. It is called fully-connected because if we represent each coefficient of the matrix W by a line that connects an element of x to an element of y, x and y are fully-connected by W (see figure 3). To motivate the next section, notice that stacking two of these layers is equivalent to only one of those: $y = W_2(W_1 x + b_1) + b_2 = W_2 W_1 x + (W_2 b_1 + b_2)$.

2.3.2 Activation function

If the architecture of a NN used only fully-connected layers, it would be equivalent to an affine function. For the NN to be able to take the shape of more complicated functions by stacking fully-connected layers, non linear functions

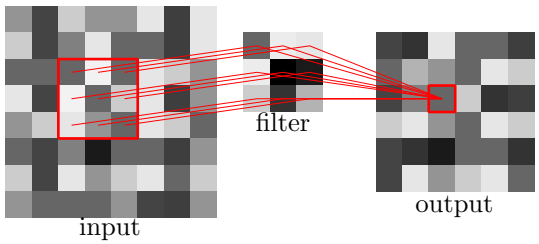


Figure 4: The convolutional layer uses the same filter for all the image.

are inserted between each layer. They are called *activation functions*. The most commonly used ones are: $max(0, x)$ (ReLU), $\frac{1}{1+e^{-x}}$ (sigmoid), Heaviside and $\frac{e^{x_i}}{\sum_j e^{x_j}}$ (soft-max).

2.3.3 Convolutional layer

When we work on images, the fully-connected layer suffers of 3 flaws

- It contains a huge number of weights: $width_{in} \times height_{in} \times width_{out} \times height_{out}$.
- It breaks down the spacial notion of the image: The result of a fully-connected layer is a set of numbers without meaningful structure.
- It doesn't use the fact that the ability to recognize an object shouldn't depend on its position in the image.

The convolutional layer solves these three problems. It takes as input an image and transforms it into another image. Only the neighborhood of a pixel from the input image (usually a square) is fully-connected to the corresponding pixel in the output image. The matrix containing the weights is called *filter*. The same filter is convoluted on the whole input image. The figure 4 illustrates this operation.

When performing a convolution, the borders can be handled in various manners. The *zero-padding* consists in adding a border of zeros to the input image in such a way that the output image has the same dimensions as the input image (depending on the shape of the filter). Without such padding, the dimensions of the output image are reduced by $w - 1$ in width and height for a $w \times w$ filter.

2.3.4 Pool max

The input of the NN being an image and the output being a single number, a layer is needed to reduce the spacial dimensions of the data flow. This so-called pool max layer with a tile of 2×2 consists in keeping only the maximum pixel in each 2×2 square of the image. The size of the image is then reduced by 2 in width and height.

2.4 Useful techniques

2.4.1 Dropout

The dropout (Srivastava et al., 2014) aims to reduce the overfitting. It consists in randomly disabling a portion of the NN at each training iteration. We define a dropout probability and at each iteration, the entry of each tensor affected by the dropout is randomly set to zero with this probability.

2.4.2 Views

An idea is to feed the NN with parts of the image (called *views*) instead of the full image (Dieleman et al., 2015). In section 3, we will use two views, the full image and the central part, and feed them to two NNs that are connected at the end to generate a single prediction. To ensure that both NNs learn, the training phase begins by train only one, then the other one, and finally both at the same time.

2.4.3 Batch normalization

It is important in NN to have well normalized data. Otherwise, as the data flow deep into the NN, depending on the initialization of the weights, it can go far out of the domains where the activation functions vary. And therefore all the intermediate tensors need to be well normalized. To ensure this, the *batch normalization* (Ioffe and Szegedy, 2015) computes the mean and variance over the batch at each iteration and uses it to renormalize the tensors into the NN.

The batch normalization is very useful to start quickly the training but it suffers from two flaws

- The prediction depends on the other images in the batch
- If the batch is too small, its mean and variance can be very different from the mean and variance of the full training set.

To avoid these two problems a "cumulative" batch normalization is used. It consists in using accumulators for the mean and the variance, the value of the accumulator is updated as follows: $\bar{\mu}_i \leftarrow (1-\alpha)\bar{\mu}_{i-1} + \alpha\mu_i$ where at iteration i, $\bar{\mu}_i$ is the accumulator of the mean, μ_i is the mean computed on the batch and α is an hyperparameter[1] between 0 and 1. To remove dependence to the batch, α is set to 0 when making a prediction. At the beginning of the training, the batch normalization is very useful. But once the training is finished, it is not wanted. Therefore α is typically set to an exponential decaying with the training steps. When the training goes on With this cumulative batch normalization, the NN depends less and less on the other images in the batch. And there is no more dependence on the batch during the predictions.

2.4.4 Normalization propagation

Considering the input and output tensors of the layers as random variables, assuming that the input tensor consists of independent random variable distributed with the normal law, the layers can be modified such that the output tensor consists of random variables distributed with the normal law (Arpit et al., 2016). Mathematically we want each layer f to satisfy $x \sim \mathcal{N} \Rightarrow f(x) \sim \mathcal{N}$.

For instance, for the fully-connected layer ($y_j = \sum_i x_i W_{ij}$) we have a variance of

$$\langle y_j^2 \rangle = \sum_i \sum_k \underbrace{\langle x_i x_k \rangle}_{\delta_{ik}} W_{ij} W_{kj} = \sum_i W_{ij} W_{ij}$$

Therefore, to apply the normalization propagation to the fully-connected layer, we need to impose $\sum_i W_{ij} W_{ij} = 1$.

2.5 Symmetry

In image recognition, it happens frequently that the labels are invariant under some spacial transformation. For instance, the vertical mirror is often used in the recognition of common objects (cars, cats, dogs, ...). With astrophysical images, the labels are often invariant under rotations, mirrors and scales. Instead of taking all the symmetries into account, it can be numerically interesting to restrict them to a symmetry group which is computationally simple. In this project, the dihedral group has been used. It is formed of rotations of 90 degrees and mirrors (see figure 5). This group is computationally simple because it does not require any interpolation. The absence of interpolation ensures no loss of information.

2.5.1 Augmentation

The symmetry group can be used to generate more images to train the NN, this is called *augmentation*. By augmenting the training set with the transformations of the dihedral group, we get 8 times more images for the training. It can be also used to improve the prediction by computing a result for each transformation of the image and merge all these in one final prediction.

[1] A parameter that defines the architecture.

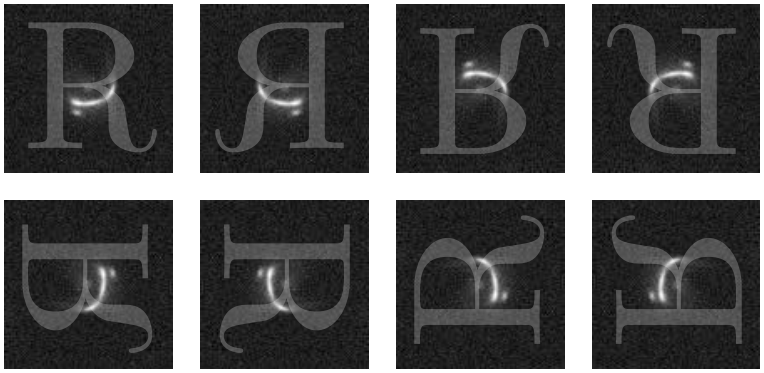


Figure 5: The dihedral group is composed of 8 elements

2.5.2 Invariant neural network

At the end of the training, the NN will eventually discover by itself all the symmetries of the problem.

Rather than relying on the training to discover the symmetries we can enforce some of them from the beginning (Dieleman et al., 2016). The idea is to create an architecture that is itself invariant under some symmetry group G, that is to say a function f satisfying equation 4 for all set of parameters W.

$$\forall W,\ f(x, W) = f(\tau x, W) \quad \forall \tau \in G \tag{4}$$

To build up an invariant NN, the output of each layer must transform with a representation of the group $(G, \circ)$. A representation D is a mapping from the group to linear transformations such that

- $D[e] = 1$ where e is the identity of the group and 1 is the identity application on the vectorial space.
- $D[\tau_1 \circ \tau_2] = D[\tau_1]D[\tau_2]$

As pictured in equation (5), if the input x is changed into τx, each intermediate tensor must transform with some representation. By doing so, we can ensure that the output changes with the trivial representation[2]; in other words, it will be invariant.

$$\begin{array}{ccccccc} x & \xrightarrow{\text{layer 1}} & y_1 & \xrightarrow{\text{layer 2}} & y_2 & \longrightarrow \cdots \longrightarrow & p \\ \tau x & \xrightarrow{\text{layer 1}} & D_1[\tau]y_1 & \xrightarrow{\text{layer 2}} & D_2[\tau]y_2 & \longrightarrow \cdots \longrightarrow & p \end{array} \tag{5}$$

Therefore each layer must be *equivariant*. A function g is called G-equivariant if

$$\exists D_1, D_2, \forall x \forall \tau \in G, \quad g(D_1[\tau]x) = D_2[\tau]g(x)$$

where D_1 and D_2 are two representations.

The following representations will be used:

- *Trivial* or *invariant representation*: on any vectorial space, it is the representation which associates the identity application to any element of the group.
- *Defining representation*: The ensemble of square images made of pixels forms a vectorial space of dimension equal to the number of pixels. The dihedral transformation (rotations by 90 degrees, mirrors) acting of such images can be viewed as permutation matrices. These matrices form a representation. We call this representation the defining representation.
- *Regular representation*: it acts on a vectorial space of dimension equal to the order of the group. On a basis in which each element is associated with an element of the group, the representation is defined as $D[\tau]|\sigma\rangle = |\tau \circ \sigma\rangle$.

It is also important to note that the tensor product of two representations is a representation: $D[\tau] = D_1[\tau] \otimes D_2[\tau]$.

Now that we have defined properly these notions, we can use a lighter notation. In the following, the representation which is being used can be inferred from the space on which the group is acting; therefore we will use τ instead of $D[\tau]$.

[2]The trivial representation is $\forall \tau \in G, D[\tau] = 1$

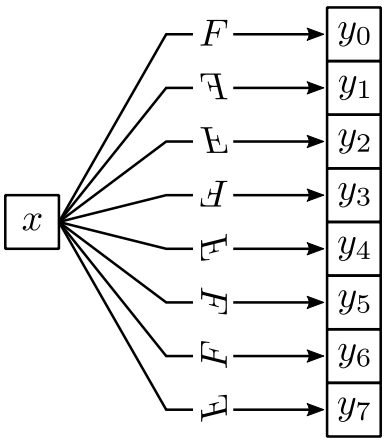


Figure 6: Dihedral convolution from representation $x \longrightarrow \tau x$ to $y_i \longrightarrow \tau y_{\tau^{-1} \circ i}$

2.5.3 Dihedral convolution

The aim of this section is to build a modification of the convolutional layer equivariant under the dihedral group.

For the sake of simplicity, we will consider that the images and the filters are square, this is not a real restriction [3].

Let us denote by $Conv(x, F)$ the convolution of the image x with the filter F. A property of the convolution is that $Conv(\tau x, \tau F) = \tau Conv(x, F)$ for τ an element of the dihedral group.

First consider that the input transforms with the defining representation $x \longrightarrow \tau x$. This will be the case for at least the first layer. The result of the convolutional layer transforms as follows (6)

$$Conv(x, F) \longrightarrow Conv(\tau x, F) = \tau Conv(x, \tau^{-1} F) \tag{6}$$

It is not equivariant because there is no representation of G (that is to say no linear transformation depending only on τ) which does this transformation.

Instead, if we build the new layer so that it computes various convolutions (not only one), we can force the result of the layer to transform with a representation. To keep the mathematical expressions short, let us choose an ordering of the dihedral group and make a bijection between the integers from 0 to 7 with the elements of the group. The following order has been chosen: $\{\mathrm{R}, \mathrm{Я}, \mathrm{R}, \mathrm{Я}, \mathrm{R}, \mathrm{Я}, \mathrm{R}, \mathrm{Я}\}$[4]. The new layer is described in equation (7)

$$y_i = Conv(x, iF) \quad i \in \{0, \ldots, 7\} \tag{7}$$

Now when $x \longrightarrow \tau x$ then $y_i \longrightarrow y'_i = \tau Conv(x, \tau^{-1} \circ iF) = \tau y_{\tau^{-1} \circ i}$ which is a representation. Indeed, it is the tensor product of the defining representation acting on the spacial components with the regular representation acting on the channel components. So we managed to build an equivariant layer (equation (7), illustrated in figure 6).

But we are only done for the case when the input transforms with the defining representation. We now want to stack layers, so we need an equivariant operation which takes as input a tensor transforming with our new representation: $x_i \longrightarrow x'_i = \tau x_{\tau^{-1} \circ i}$. We have room to make some choices. Let us impose that

- The input (x_i) and the output (y_i) transform with the same representation
- We use eight filters $F_i, i \in \{0...7\}$
- $y_j = \sum_i Conv(x_i, \mu(i, j) F_{\sigma(i,j)})$ where μ is an element of G and σ takes integer values between 0 and 7
- $\mu(i, 0) = 0, \sigma(i, 0) = i$ which is equivalent to $y_0 = \sum_i Conv(x_i, F_i)$

[3] To handle mathematically the case of non square images or filters, they can be padded with zeros to fit in a square.

[4] This choice is arbitrary.

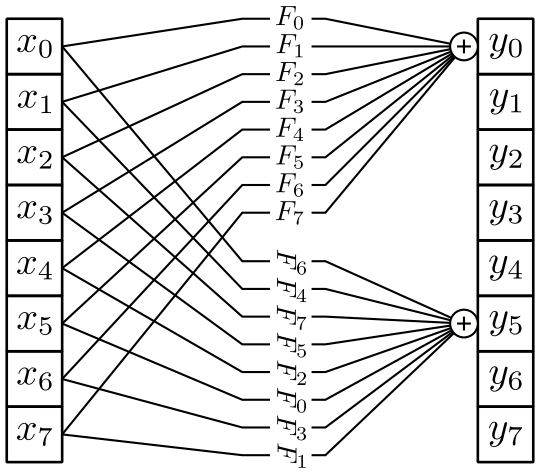


Figure 7: Dihedral convolution from representation $x_i \longrightarrow \tau x_{\tau^{-1}\circ i}$ to $y_i \longrightarrow \tau y_{\tau^{-1}\circ i}$

We just have to find expressions for μ and σ. We can compute the transformation induced by the transformation of the input:

$$y'_j = \sum_i Conv(\tau x_{\tau^{-1}\circ i}, \mu(i,j)F_{\sigma(i,j)}) = \sum_i \tau Conv(x_{\tau^{-1}\circ i}, \tau^{-1}\circ\mu(i,j)F_{\sigma(i,j)}) = \sum_i \tau Conv(x_i, \tau^{-1}\circ\mu(\tau\circ i,j)F_{\sigma(\tau\circ i,j)}) \tag{8}$$

where we used the property of the convolution and reordered the sum. Using that we imposed how the y_js transform:

$$y'_j = \tau \sum_i Conv(x_i, \mu(i,\tau^{-1}\circ j)F_{\sigma(i,\tau^{-1}\circ j)}) \tag{9}$$

By identification of equations (8) and (9):

$$\begin{cases} \tau^{-1}\circ\mu(\tau\circ i,j) = \mu(i,\tau^{-1}\circ j) & \Leftrightarrow \mu(\tau\circ i,\tau\circ j) = \tau\circ\mu(i,j) \\ \sigma(\tau\circ i,j) = \sigma(i,\tau^{-1}\circ j) & \Leftrightarrow \sigma(\tau\circ i,\tau\circ j) = \sigma(i,j) \end{cases}$$

$\mu(i,j) = j$ and $\sigma(i,j) = j^{-1}\circ i$ fulfill all the constraints. So we have found a new equivariant layer

$$y_j = \sum_{i=0}^{7} Conv(x_i, jF_{j^{-1}\circ i}) \quad j \in \{0,\dots,7\}$$

The figure 7 illustrates this layer.

2.5.4 Other layers

Other equivariant layers can be easily derived from the two treated in the last section. The table 1 lists all the equivariant layers used in this project.

The pool max with a $s \times s$ tile is equivariant for both $x \longrightarrow \tau x$ and $x_i \longrightarrow \tau x_{\tau^{-1}\circ i}$ representations as long as the dimension of the input image is a multiple of s.

2.5.5 Domain of application and generalization

Rectangular images All the layers listed in table 1 apply dihedral transformations to the filters but not to the images. Therefore, all the layers can be used (without any modification) to implement invariant NN also for input images that are rectangular.

Other groups Among the properties of the dihedral group, the only one which has been used above is the identity $Conv(\tau x, \tau F) = \tau Conv(x, F)$. Therefore, theses layers can be modified for any group that satisfies this identity. As far as we know, the dihedral group is the larger group satisfying this property. But it has four subgroups, the rotations and the horizontal/vertical/both mirror.

Representation		Equation	Comment
Input	Output		
$x \longrightarrow \tau x$	$y_i \longrightarrow \tau y_{\tau^{-1} \circ i}$	$y_i = Conv(x, iF)$	useful to start the NN (see figure 6)
$x_i \longrightarrow \tau x_{\tau^{-1} \circ i}$	$y_i \longrightarrow \tau y_{\tau^{-1} \circ i}$	$y_j = \sum_i Conv(x_i, jF_{j^{-1} \circ i})$	(see figure 7)
$x_i \longrightarrow \tau x_{\tau^{-1} \circ i}$	$y \longrightarrow \tau y$	$y = \sum_i x_i$	
$x_i \longrightarrow \tau x_{\tau^{-1} \circ i}$	$y \longrightarrow \tau y$	$y = \sum_i Conv(x_i, iF)$	useful to get a smaller tensor (see figure 8)
$x_i \longrightarrow x_{\tau^{-1} \circ i}$	$y_i \longrightarrow y_{\tau^{-1} \circ i}$	$y_j = \sum_i x_i \cdot W_{j^{-1} \circ i}$	fully-connected layer
$x_i \longrightarrow x_{\tau^{-1} \circ i}$	$y \longrightarrow y$	$y = \sum_i x_i$	useful at the end
$x_i \longrightarrow x_{\tau^{-1} \circ i}$	$y \longrightarrow y$	$y = \sum_i x_i \cdot W$	useful at the end

Table 1: Some equivariant layers under the dihedral group. With these layers, invariant NN under the dihedral group can be built.

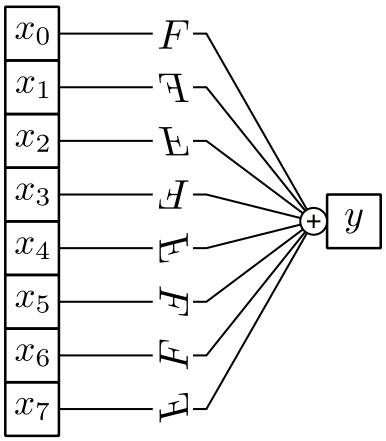


Figure 8: Dihedral convolution from representation $x_i \longrightarrow \tau x_{\tau^{-1} \circ i}$ to $y \longrightarrow \tau y$

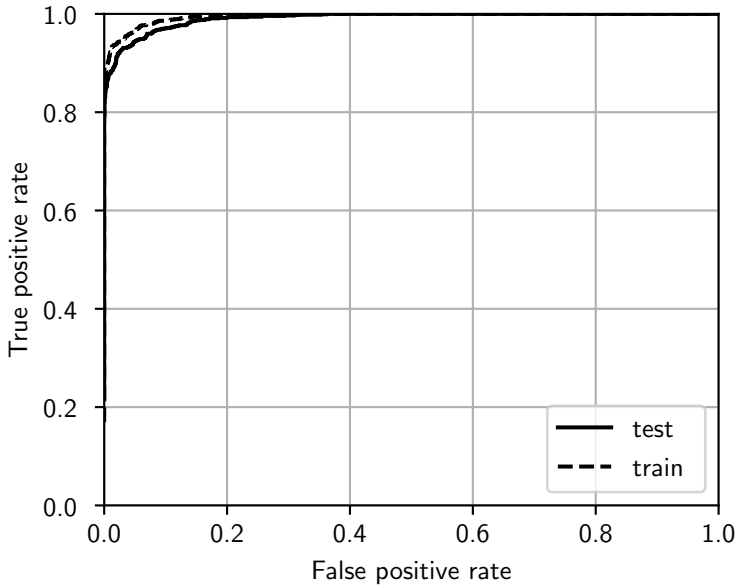


Figure 9: Example of a ROC curve (the architecture with view for the ground based data).

3 Implementation

Now that the bases have been clarified, let us study in more details how they are applied to participate in the *Gravitational Lens Finding Challenge*.

3.1 Challenge Information

In the challenge, 20k images with labels are provided for training our algorithm (split into the training set (17k) and validation set (3k)). When we were ready, we had to download 100k unlabeled images, compute the predictions and upload them within the 48h following the download.

3.1.1 Roc curve

Because the result of our NN is a number between 0 and 1, a detection threshold is needed to make a prediction (i.e. to choose whether the answer is 0 or 1). The choice of the threshold is arbitrary. For each threshold there will be a different amount of false positive and false negative. The *true positive rate* is the ratio of the number of true positive over the number of labeled as positive. The *false positive rate* is the ratio of the number of false positive over the number of labeled as negative. These two ratios are bounded in $[0, 1]$. As illustrated in figure 9, a ROC (receiver operating characteristic) curve is a path from $(1, 1)$ to $(0, 0)$ parametrized by the detection threshold.

Two scores are used in the challenge.

- the area under the ROC curve abbreviated ROC AUC. A perfect classifier will have a ROC AUC of 1. A random classifier will have a ROC AUC of 0.5.
- the true positive rate for a threshold at which there is no false positive for the 10k first images of the test set.

In our algorithms, we focused on maximizing the ROC AUC.

3.1.2 Data set

In this challenge, two types of images must be analyzed. For both type 20k labeled images are provided to the participants to train their algorithm. These are mock data, they are not coming from actual observations but have been numerically generated. They are images of galaxies lensed by galaxies (no clusters and no quasars). A lensed

Architecture	Amount of parameters	Memory (float 32 bit)
Baseline	5674145	21.6 Mb
Views	7035308	26.8 Mb
Invariant	1753705	6.7 Mb
Residual	4863021	18.6 Mb

Table 2: The amount of training variables (variables affected by SGD) in each architecture. It is interesting to notice that the more complex architectures do not necessarily contain more parameters. In particular, the residual architecture is made of a large number of layers but contains less parameters than the baseline.

image is simulated from a galaxy image and a mass model, then it is merged with an image of another galaxy (the lens) and finally some noise is added.

- **Space based** 101×101 images (gray-scale), 13968 labeled as lens. The figure 10 shows a random selection of the images of this type.
- **Ground based** $101 \times 101 \times 4$ images (4 color channels), 8021 labeled as lens. The four channels correspond to the optical filters: red (R), infrared (I), green (G) and ultraviolet (U). The channels sometimes contain masked region. The figure 11 shows a random selection of the images of this type.

3.1.3 Preprocessing of image

Normalization For each dataset, the mean and the variance of the whole training set has been computed to normalize the input images. After normalization, the mean and the variance of the images are 0 and 1, respectively.

Handling of masked regions For the ground based data type, the masked region are handled by simply setting them to 0. This method is not good because it generates false positive for images like this one 12. An other idea consisting in adding an extra channel with value 0 for not masked region and 1 for masked region (even partially) was tested but abandoned.

3.2 Data augmentation

The presence of a lens is invariant under the rotations, mirrors and scales. To ensure no loss of information due to interpolation, only the rotations of 90 degrees and mirrors have been used to augment the training set. For the architectures made invariant under the dihedral group, this augmentation is useless.

Other types of transformation of the images can be considered. The modification of the color, contrast and brightness is also highly used for images recognition (Krizhevsky et al., 2012). The modifications of the colors were not used because we were worried about loosing any spectral information. Therefore, only a global scale and shift of all the values of the pixels has been imlemented because it cannot remove information in the image.

3.3 Architecture

During the project, a lot of architectures with small changes has been tested. The training containing randomness, multiple instances of training must be executed to properly compare two architectures. Here four architectures are presented.

3.3.1 Baseline

A quite simple architecture constitutes our baseline for comparison. It is made of 8 conventional convolutional layers and 4 fully-connected layers (See figure 13a). The sigmoid activation function of the single value output gives the prediction of the NN. It uses the cumulative batch normalization, the dropout and data augmentation.

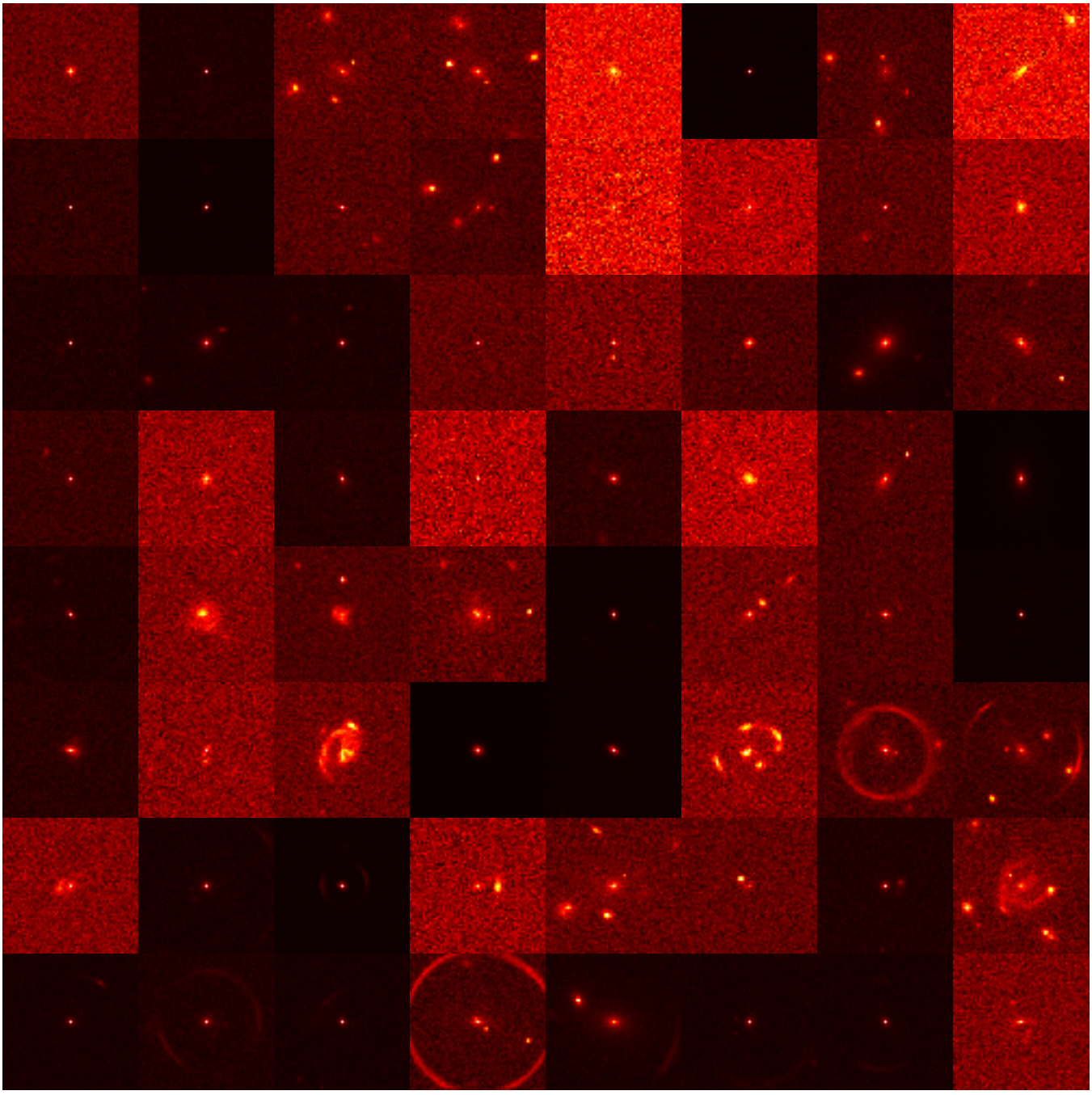


Figure 10: Samples from the **space based** images. The 4 top rows are labeled as containing no lens. The 4 bottom rows are labeled as containing a lens.

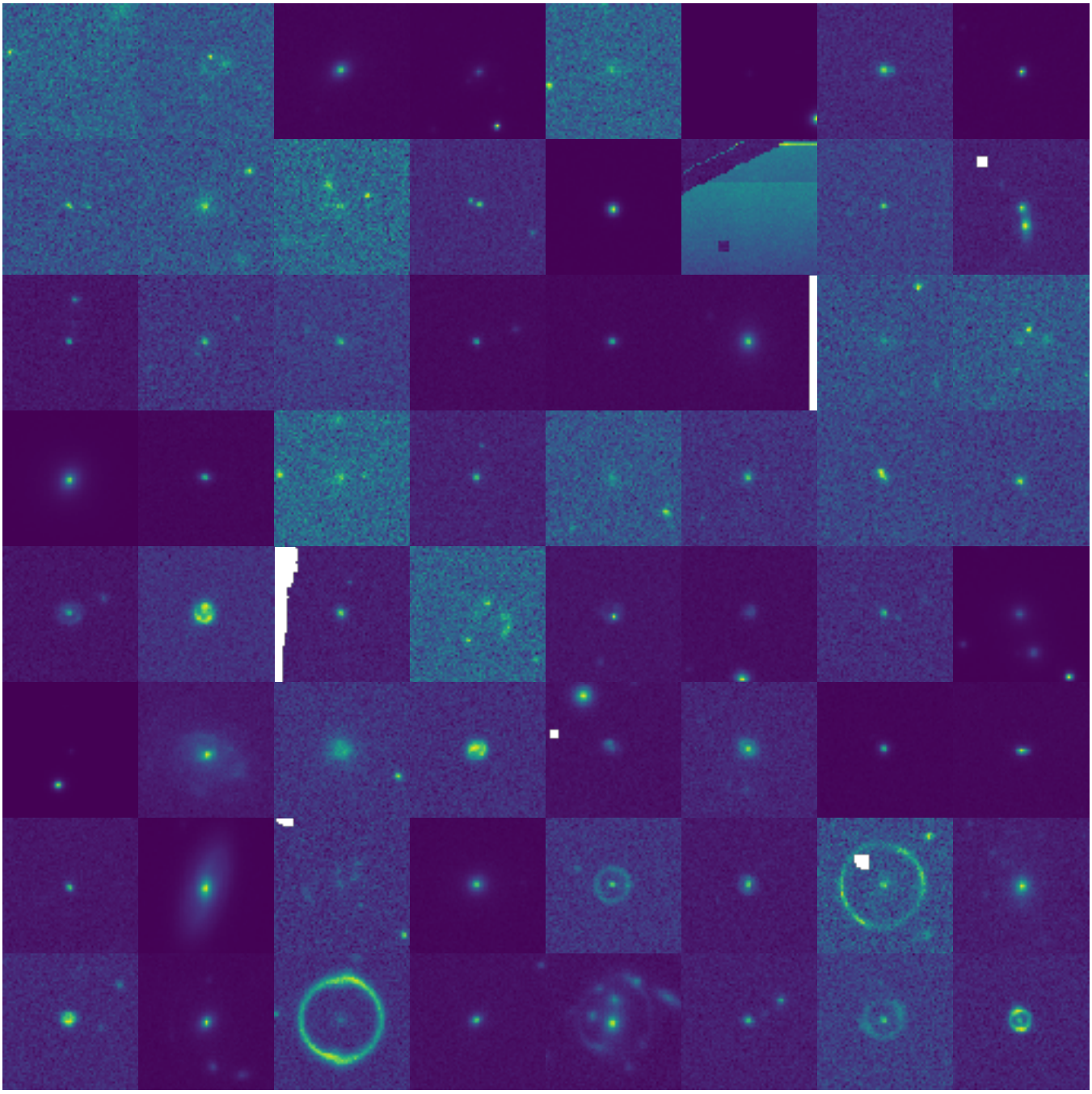


Figure 11: Samples from the **ground based** images. Only the band R is shown. The 4 top rows are labeled as containing no lens. The 4 bottom rows are labeled as containing a lens.

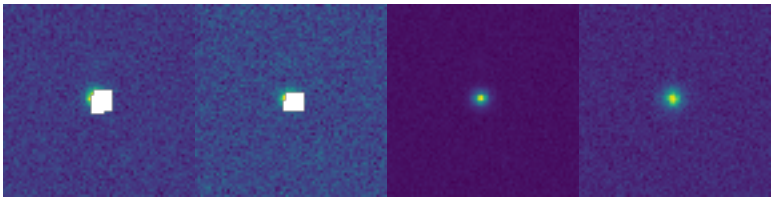
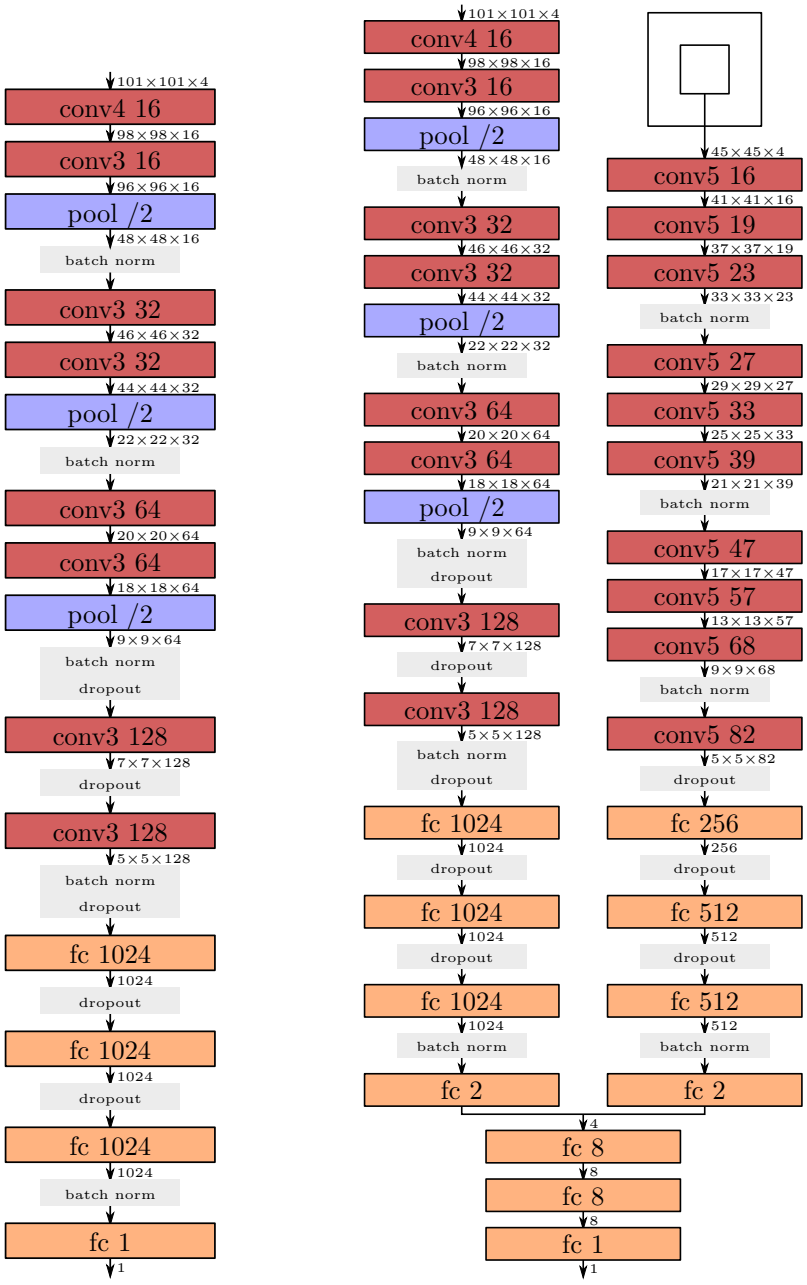
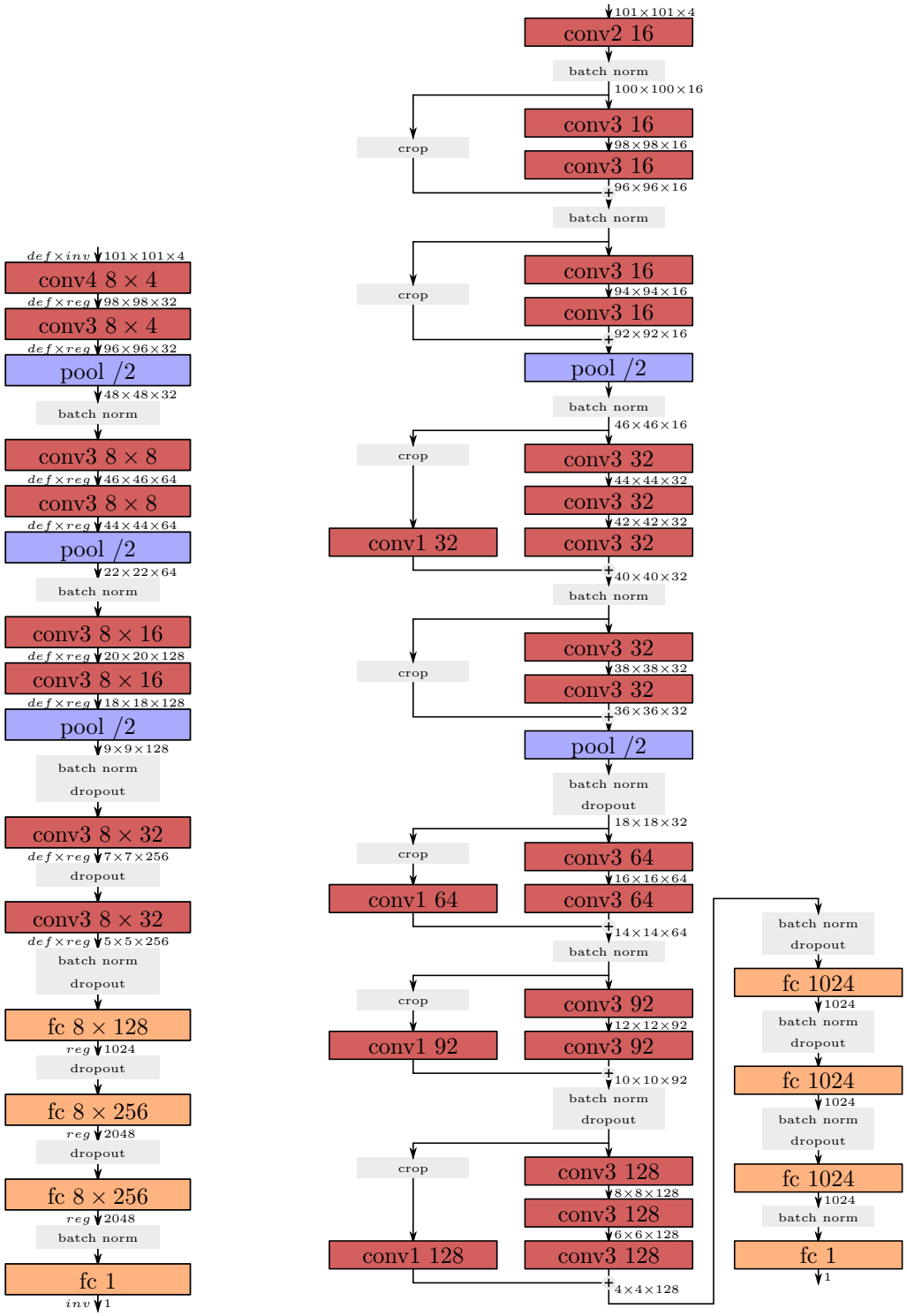


Figure 12: The four channels of an image likely to generate a false positive.



(a) Baseline (b) Views

Figure 13: Architectures: `conv{w} {f}` denotes a convolutional layer with tile $w \times w$ and f output channels. `pool /2` denotes the max pool with tile of 2×2. `fc f` denotes a fully-connected layer with f output channels. Except the last one (sigmoid), each convolutional and fully-connected layers is followed by the activation function ReLU.



(a) Invariant

(b) Residual

Figure 14: Architectures: `conv{w} {f}` denotes a convolutional layer with tile $w \times w$ and f output channels. `conv{w} {8×f}` denotes an **equivariant convolutional layer**. `pool /2` denotes the max pool with tile of 2×2. `fc f` denotes a fully-connected layer with f output channels. Except the last one (sigmoid), each convolutional and fully-connected layers is followed by the activation function ReLU.

Initialization This architecture does not use normalization propagation. (The normalization propagation has been tested, allowing to train without batch normalization, but the results were not better.) Instead, the variables of the layers are only initialized in such a way that if the input tensor is normalized (zero mean and variance of 1) then the output is also normalized.

No padding We choose to not use any padding for our convolutional layers (also called `valid padding` in some libraries). This does not introduce arbitrary border value (usually 0) that creates artifacts especially annoying when the images get small. Also it reduces the size of the data flow which has been very useful to avoid the usage of more max pool. The reduction of the size was also very useful for making invariant NN, as it allows to ensure that the max pool gets an even dimension of image to be dihedral-equivariant.

SGD The Adaptive Moment Estimation (Kingma and Ba, 2015) has been used.

3.3.2 Views

This variant (based on section 2.4.2) adds a second NN fed with the center of the image to the baseline one. (See figure 13b).

3.3.3 Invariant

It uses the layers described in section 2.5. It is based on the baseline architecture. The main difference is that all the layers are replaced with their dihedral-equivariant ones. Because the regular representation requires tensors with channel size multiple of 8, the size of the tensors has been multiplied by 2. On the other hand, the filters of the equivariant layers uses 8 times less parameters compared to the baseline layers for the same input and output channel size, hence the amount of parameter per convolution is divided by 2 (see table 2). This architecture is sketched in figure 14a. This architecture makes more parameters sharing that the baseline, it has therefore a smaller amount of parameters (table 2). In principle, having less parameters reduces the risks of overfitting. The validity of equation (4) has been tested with random weights.

3.3.4 Residual

In the challenge, the winner of the category "ground based and ROC AUC" used a residual network. For curiosity, such an architecture has been quickly built and tested here. As NN grow deeper (i.e. include more layers), the training phase becomes more delicate. In order to still be able to train such networks, a solution is to shortcut several layers with one simpler layer(He et al., 2016) (see figure 14b). This architecture contains 17 convolutional layers and 4 fully-connected layers.

4 Results

The training time varies with the complexity of the architecture. At the beginning of the project we used CPUs on a cluster; later, we bought a GPU for gammer (GTX 1060 6G) and the time of computation was reduced by 15[5]. To give an order of magnitude, the training on the GPU took about 4h for 200k iterations with the baseline architecture, 5h for the invariant architecture (this slow-down is only a matter of lack of optimization in the library used[6]), 5h30 for the architecture with the views and 7h for the residual architecture. Therefore, to compare the architectures, they have been trained several times. The figures 15 and 16 (respectively for space and ground based data), show the results of the architectures presented in the previous section. These figures are generated from at least 4 runs per architecture and only the best run is shown.

[5]Probably mainly because the development of Tensorflow is focused on the GPUs.

[6]We used Tensorflow and they are developing a just-in-time compiler which merges and optimizes some portions of the graph. The invariant architecture consists of a large amount of simple operations and therefore is a good candidate for optimization in the future.

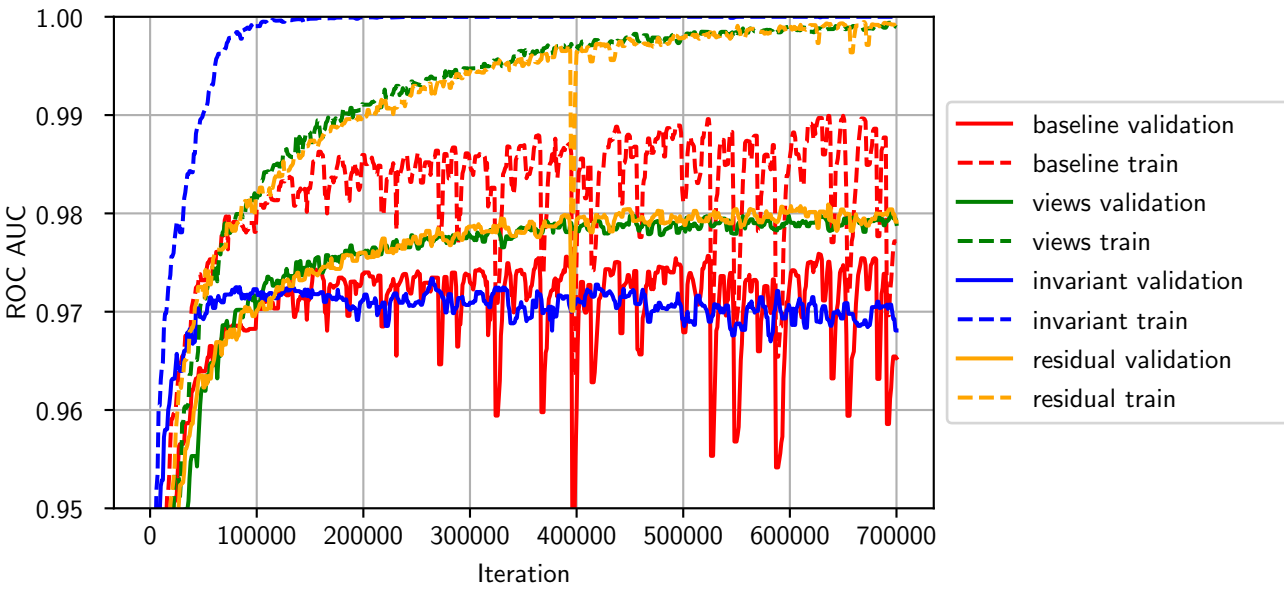


Figure 15: Area under the ROC curve for **space based** data in function of the iteration of the SGD algorithm. The plain lines are computed on the validation set (3k). The broken lines are computed on the first 3k images of the training set.

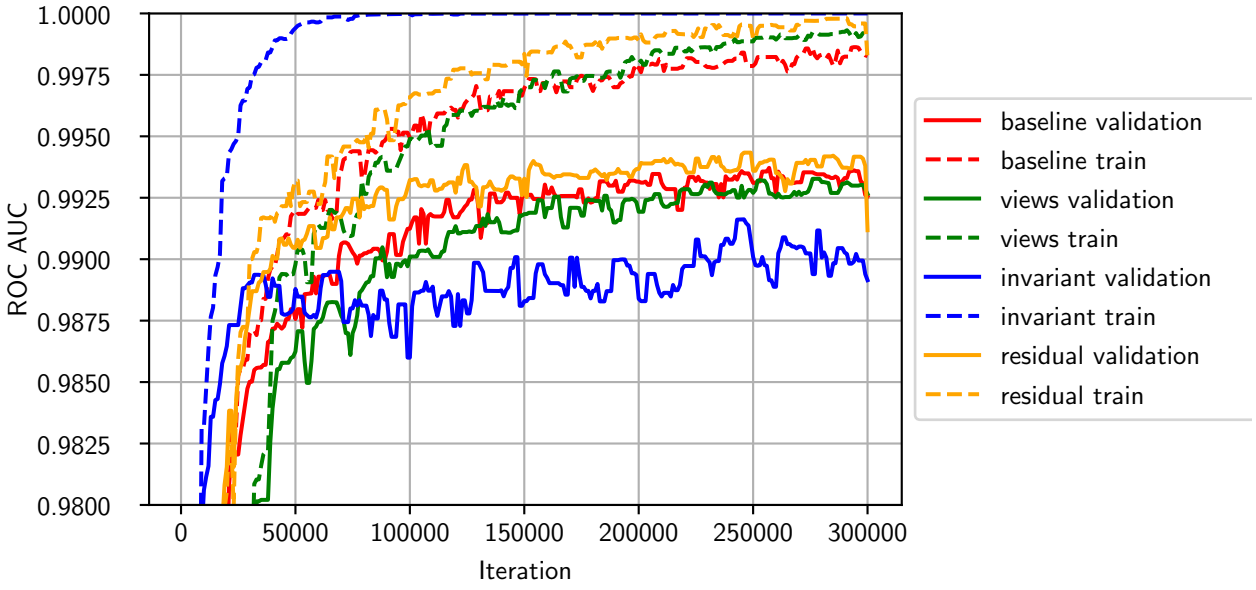
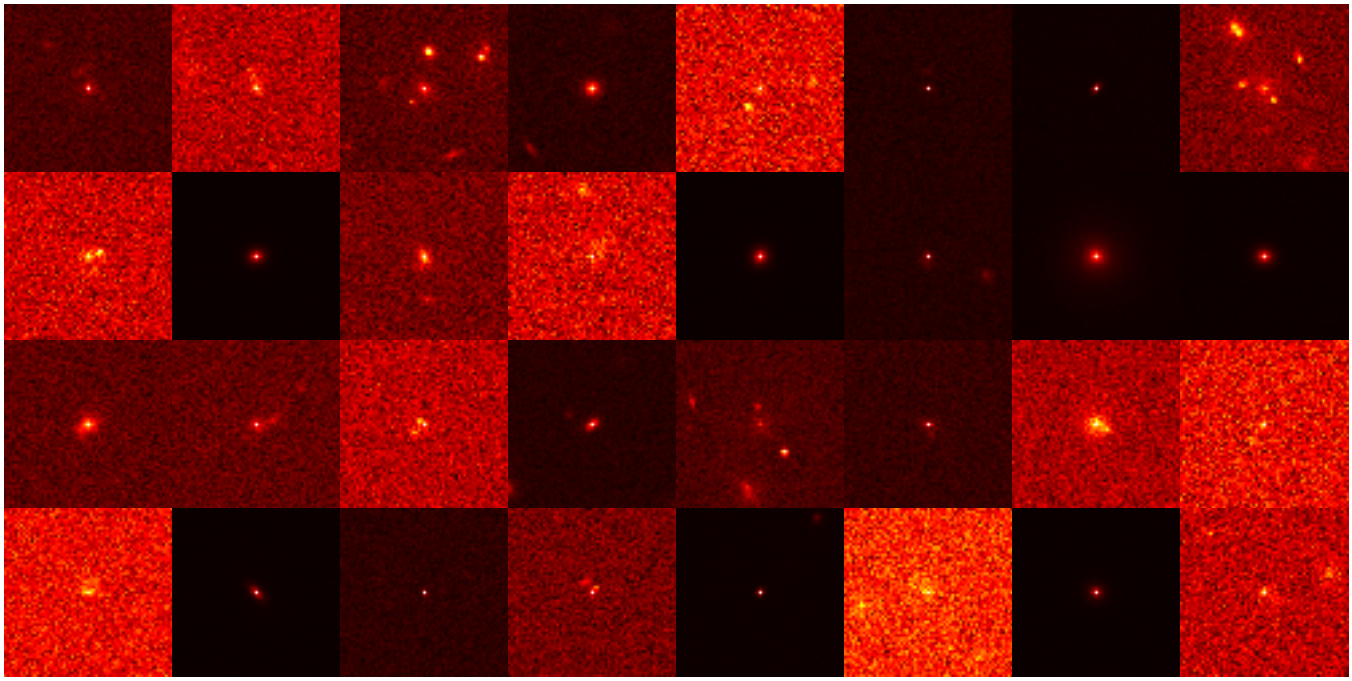
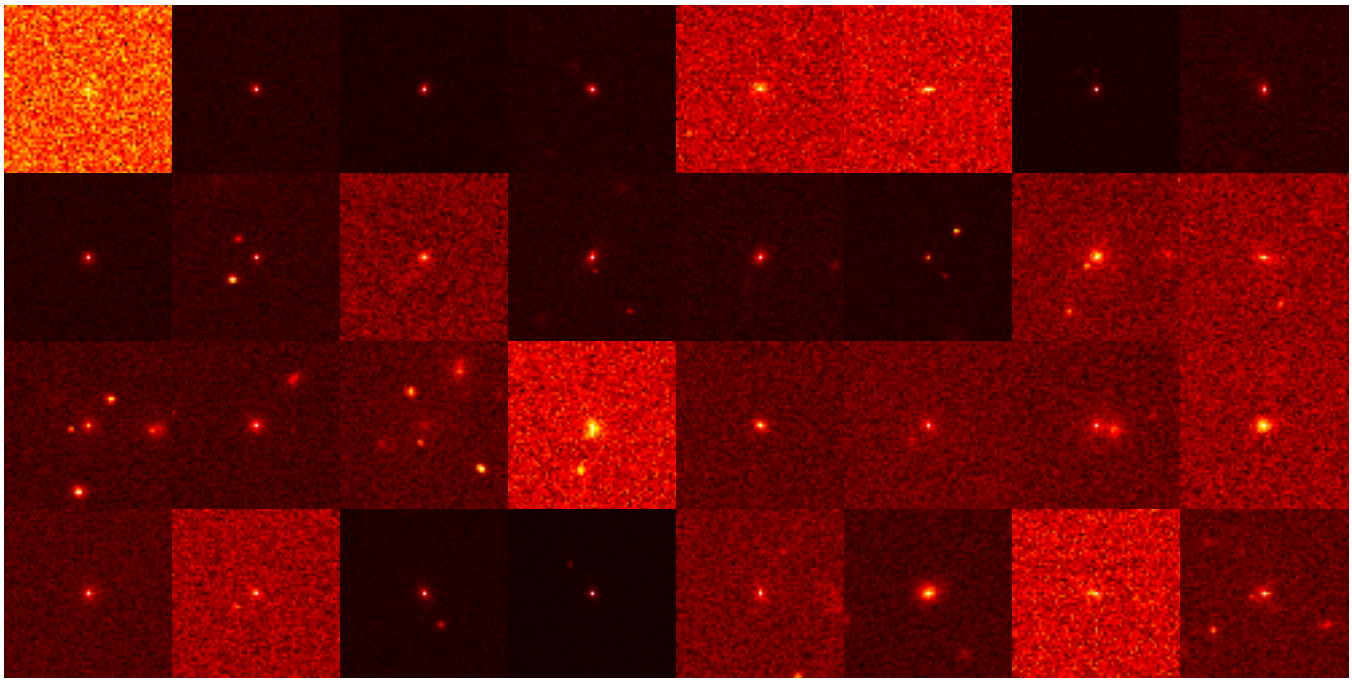


Figure 16: Area under the ROC curve for **ground based** data in function of the iteration of the SGD algorithm. The plain lines are computed on the validation set (3k). The broken lines are computed on the first 3k images of the training set.

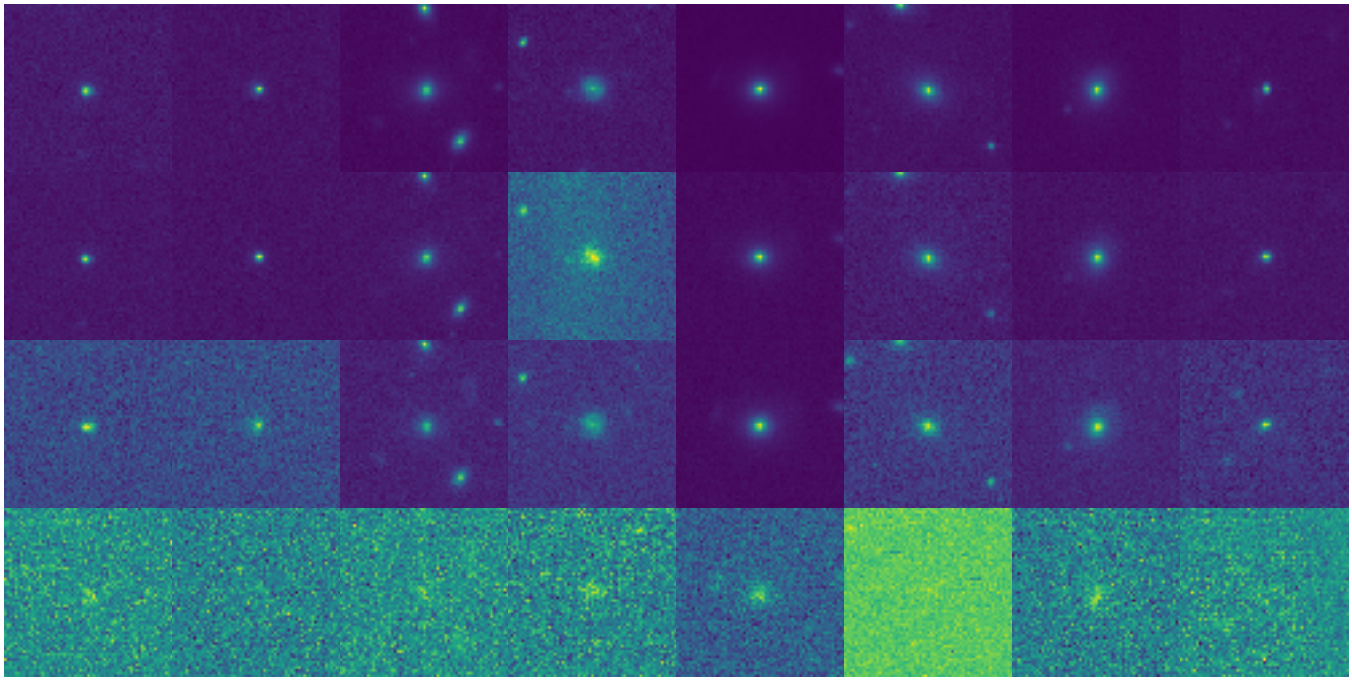


(a) False positive

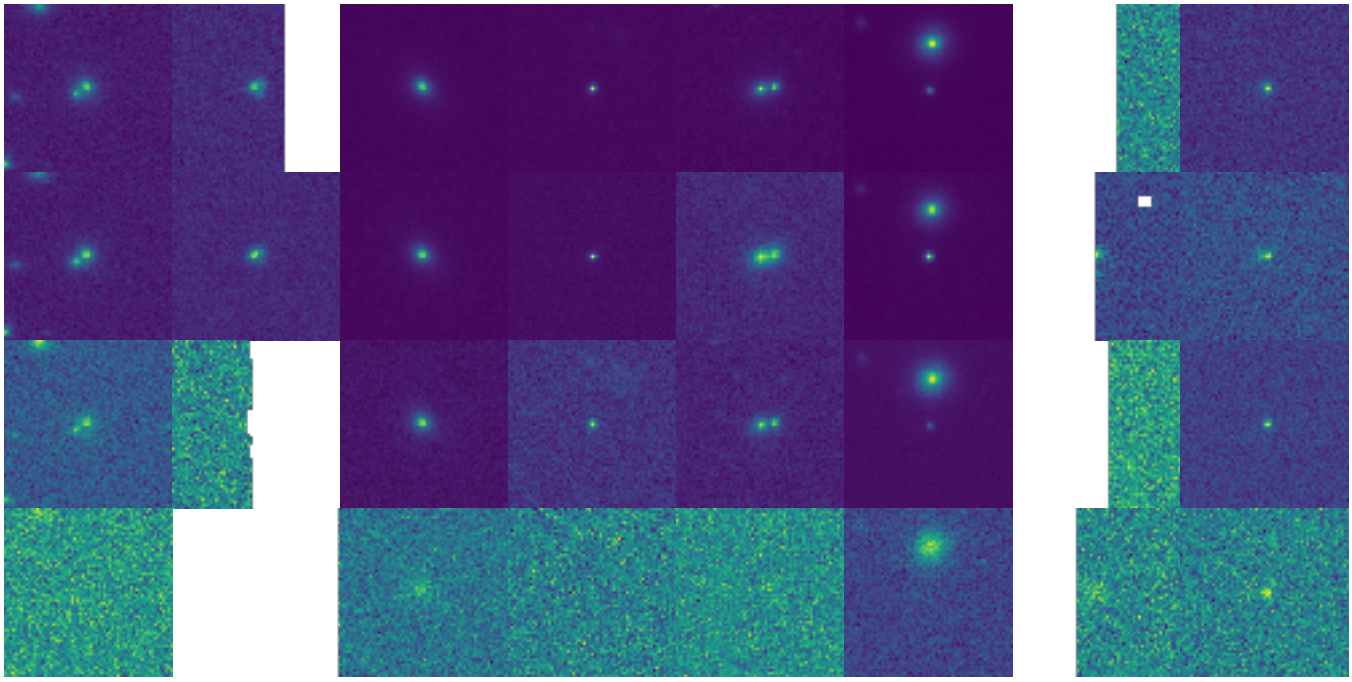


(b) False negative

Figure 17: These are the 32 most wrong false positive and false negative for the best iteration of the best run on **space based**. They are sorted starting with the most wrong.



(a) False positive



(b) False negative

Figure 18: These are the 8 most wrong false positive and false negative for the best iteration of the best run on **ground based**. The four rows correspond to the four bands: R, I, G, U. They are sorted starting with the most wrong.

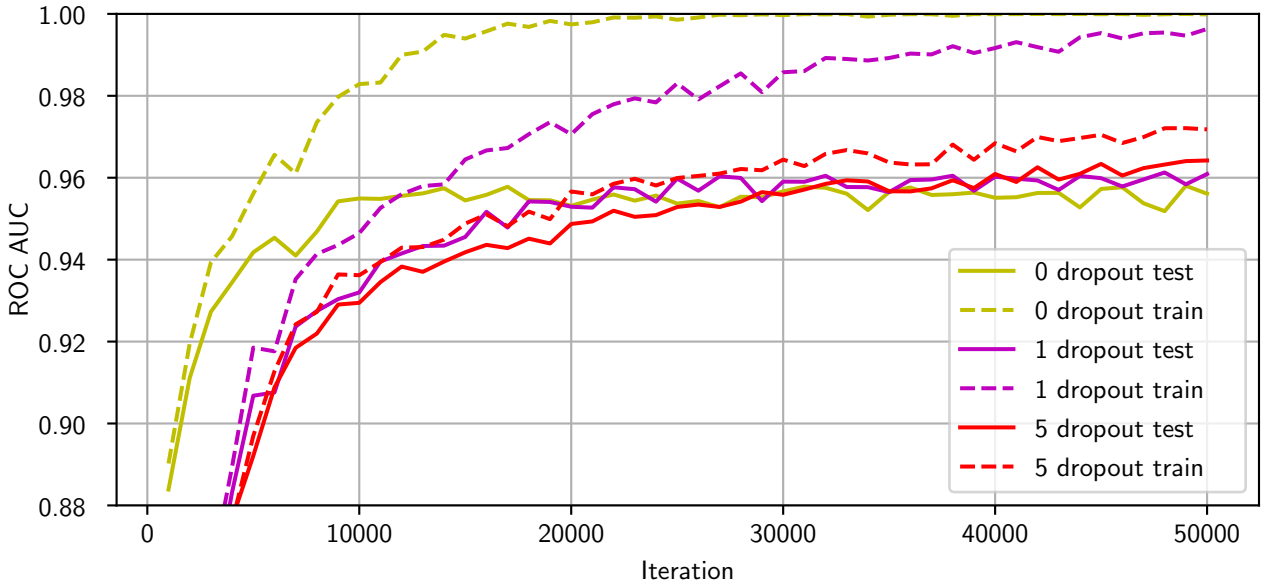


Figure 19: This graph shows why dropout has been used in this project.

4.1 Baseline

4.1.1 Dropout

The figure 19 shows the ability of the dropout to reduce the overfitting. But there is a trade-off between no dropout and dropout each tensor in the graph. The dropout reduces the capacity of learning.

4.1.2 Batch normalization

The batch normalization is very important to avoid the following problem. It happens when there is not enough batch normalization that the training gets stuck with prediction all equal to 0.5.

4.2 Views

It is, at least for the space based data, a bit better than the baseline. Despite the higher number of parameters, we think that the risk of overfitting is not too high because the two parallel NN used in this architecture have a bottleneck at their end before the merge, reducing the level of "collaboration".

4.3 Invariant

Despite having fewer parameters than baseline, which in principle reduces the risk of overfitting, the invariant versions overfit. It is maybe due to the fact that, as we cannot use the dihedral transformations for augmentation, the training set is in some sense eight times smaller.

4.4 Residual

Like in the challenge, the residual network gives better results than the baseline. On space based images, it gives almost the same results as the algorithm with views; however, at the challenge, our architecture was just better. This suggests that residual networks are the best path. Another fact which supports this is that the winner of ILSVRC 2015 used a residual network (He et al., 2016).

50k more iterations have been added to the two better architecture to distinguish them (15). It appear that as long as they do not overfit, increasing the number of iterations improves the results.

Architecture	ROC AUC	TPR0
Space based	0.9325338	0.004773626 (0.2206807)
Ground based	0.9749255 (0.9814321)	0.07493794 (0.2237273)

Table 3: Scores of the submission. In parentheses, the best score of the challenge

4.5 Challenge

The baseline architecture has been submitted to the challenge. To evaluate the algorithm, the challenge gave us 100k images to analyze in 48h. The results of the submission are in the table 3. In this table, the TPR0 (true positive rate at 0 false positive) were computed with the threshold ensuring that there was no false positive in the first 10k images of the challenge test set. Fortunately we got the best ROC AUC over the space based data.

5 Conclusion

The neural networks are very simple to setup and do not need any physical knowledge of the studied phenomena. It needs a good training set which can be generated with a good knowledge of the studied phenomena.

It gives good results. Indeed, Dr Neal Jackson from the university of Manchester also participate to the challenge with human inspection. Our NN gives better results than his human inspection for both data type.

However it is not an easy task to found the perfect architecture.

Another interesting aspect of the NN is that the same architecture can solves various different problem. For example, it was easy to adapt our NN to test it on the galaxy zoo challenge (gal, 2014). In the galaxy zoo challenge, the objective is to classify images of galaxies by answering 11 questions about the morphology of the galaxy. The answers are encoded into 37 values (the output of the NN is therefore a vector of 37 numbers).

The dihedral-equivariant layers, used to build invariant architectures, produce too much overfitting for the gravitational lensing effect recognition. As nothing prevents the creation of an invariant residual network, this might be a very well working combination.

A possibility to improve the TPR0 would be to use an asymmetric modification of the cross entropy as cost function. Indeed, the cross entropy penalizes severely the false positive and the false negative (the cost diverges to plus infinity in the tow cases). It could greatly improve TPR0 an hopefully not reduce too much the ROC AUC if the false negative cost tends to a finite value instead of infinity.

A repository with the minimal code and jupyter documents for usage example is available at `https://github.com/antigol/lensfinder-euclid`.

References

Galaxy Zoo - The Galaxy Challenge, 2014. URL `https://www.kaggle.com/c/galaxy-zoo-the-galaxy-challenge`. Accessed: 2017-02-23.

D. Arpit, Y. Zhou, B. U. Kota, and V. Govindaraju. Normalization Propagation: A Parametric Technique for Removing Internal Covariate Shift in Deep Networks. *International Conference on Machine Learning (ICML)*, 2016.

V. Bonvin, F. Courbin, S. H. Suyu, P. J. Marshall, C. E. Rusu, D. Sluse, M. Tewes, K. C. Wong, T. Collett, C. D. Fassnacht, T. Treu, M. W. Auger, S. Hilbert, L. V. E. Koopmans, G. Meylan, N. Rumbaugh, A. Sonnenfeld, and C. Spiniello. H0licow v. new cosmograil time delays of he04351223: H0 to 3.8percent precision from strong lensing in a flat cdm model. *Monthly Notices of the Royal Astronomical Society*, 465(4):4914, 2017. doi: 10.1093/mnras/stw3006. URL `+http://dx.doi.org/10.1093/mnras/stw3006`.

S. Dieleman, J. De Fauw, and K. Kavukcuoglu. Exploiting Cyclic Symmetry in Convolutional Neural Networks. *International Conference on Machine Learning (ICML)*, 2016.

Sander Dieleman, Kyle W. Willett, and Joni Dambre. Rotation-invariant convolutional neural networks for galaxy morphology prediction. *Monthly Notices of the Royal Astronomical Society*, 450(2):1441, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Journal of Machine Learning Research*, February 2015.

Andrej Karpathy. Cs231n convolutional neural networks for visual recognition, 2016. URL `http://cs231n.github.io/convolutional-networks/`. Accessed: 2017-02-23.

D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations*, 2015.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

Robert Benton Metcalf. Gravitational Lens Finding Challenge, 2016. URL `http://metcalf1.bo.astro.it/blf-portal/gg_challenge.html`. Accessed: 2017-02-23.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going Deeper with Convolutions. *ArXiv e-prints*, September 2014.