UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia "Galileo Galilei"

Corso di Laurea in Fisica

Tesi di Laurea

STRIKE beamlet parameters retreiving via Convolutional Neural Network

Relatore

Dott. Gianluigi Serianni

Correlatori

Dott. Rita Delogu

Dott. Andrea Rigoni

Laureando

Nicola Lonigro

Anno Accademico 2018/2019

Abstract

STRIKE is a diagnostic calorimeter composed of 16 CFC tiles with unidirectional properties used to study the beams of particles generated in the SPIDER experiment. Two thermal cameras will be used to analyze the temperature of the tiles and reconstruct the bidimensional flux of energy striking the calorimeter. Most of the conventional methods used to evaluate the inverse heat flux are unbearably time consuming; since the objective is having a tool for heat flux evaluation for STRIKE real time operation, the need to have a ready-to-go instrument to understand the beam condition becomes stringent. For this reason a neural network was chosen to perform this analysis, as suggested in [2]. During the thesis work an existing convolutional neural network was optimized to retrieve the parameters of the beam from the thermographic images. In particular, starting from a network able to determine the position and the radius of a singular circular shape on a noiseless background, the possibility to recognize the position of bidimensional gaussians was added, needed to determine the beams divergence. The ability to determine the shape, the amplitude and the angle of rotation of the beams was then developed. Lastly the network was optimized to work in a noisy environment and with a non-constant number of gaussians.

In the first chapter the SPIDER experiment and the STRIKE calorimeter are briefly described, while in the second chapter an introduction to neural networks is given. In the third chapter the development of the thesis work is described and its performance is evaluated .

Contents

Chapter 1

Introduction

The search for more efficient and cleaner ways to generate energy has been at the core of many research projects and debates in the last decades. One of the most promising areas in this respect is related to nuclear energy and in particular to *nuclear fusion*. In fact, while many nuclear plants are already operative and in some countries contribute for over 70% of the energy produced (e.g. France), their operating principle is based on *nuclear fission*. In nuclear fission a heavy element is split in lighter nuclei and the difference in the energy between the starting nucleus and the sum of the energies of the daughter nuclei is released, mainly as fast neutrons and gamma photons. In most reactors ^{235}U is used as fuel and when bombarded with neutrons many decays are possible. A possible example of the fission of a nucleus of ^{235}U is

$$ {}^{1}_{0}\mathrm{n} + {}^{235}_{92}\mathrm{U} \rightarrow {}^{141}_{56}\mathrm{Ba} + {}^{92}_{36}\mathrm{Kr} + 3\,{}^{1}_{0}\mathrm{n} $$

However the usage of nuclear fission comes with a considerable number of problems, for example the scarcity of uranium and the very long lifetime of the radioactive waste generated. An alternative solution preserving the high power output of nuclear fission but with more manageable waste is nuclear fusion, a process in which two lighter nuclei fuse together to create a heavier element if given enough energy to overcome the coulomb barrier. A particularly interesting fusion process and the fulcrum of modern research regarding nuclear fusion as an energy source is given by the fusion of two hydrogen isotopes, the deuterium and the tritium:

$$ {}^{2}\mathrm{D} + {}^{3}\mathrm{T} \rightarrow {}^{4}\mathrm{He} + n $$

This particular reaction has been chosen due to its higher cross section as shown in Fig.1.1 and its lower activation energy, making it possible at temperatures as low as 20 keV. Moreover deuterium is a naturally occurring isotope, while tritium can be obtained from a nuclear reaction involving lithium, making the fuel needed for the reaction readily available.

Matter heated up to the energies required for the process to take place is said to be in a state called *plasma*, a highly ionized quasi-neutral gas of charged particles showing a collective behavior. In order to actually generate energy the point of *breakeven* must be

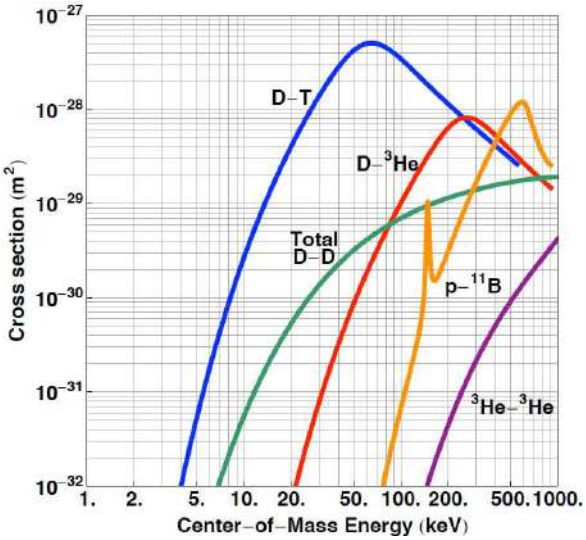

Figure 1.1: Cross section for many fusion processes interesting to nuclear fusion research

surpassed, with it being the point in which the energy extracted from the generator is greater than the one provided to create the process in the first place. Moreover the plasma is continually losing energy, mainly due to thermal conduction and bresstrahlung. Denoting the total power energy stored in the plasma as W and the total power loss due to transport phenomena as P_L, the *energy confinement time* can be defined as

$$\tau_e = \frac{W}{P_L}$$

and the *Lawson criterion* establishes a condition needed to ensure the power generated by the fusion process is greater than the power needed to heat the plasma, that is

$$n\tau_e \geq \frac{12k_bT}{E_\alpha \langle \sigma v \rangle}$$

where n is the plasma density, T its temperature, τ_e the energy confinement time, E_α the energy of the alpha particle product of the reaction (3.5 Mev for the D-T reaction) and $\langle \sigma v \rangle$ the reaction rate coefficient. A possible way to increase the left side of the inequality is to increase the confinement time, for example by confining the plasma with various configurations of magnetic fields, with the most used one being the tokamak.

1.1 ITER

Located in Cadarache (southern France), ITER (Fig. 1.2, [7]) is an experimental reactor born from the collaboration of 35 countries to prove the feasibility of using fusion as a reliable power source. Its main objective is to reach the breakeven point by using a D-T mixture heated to 10 keV.

The power output is expected to reach 500W while it's designed to have a Q factor, the ratio between the power generated and the heating power, greater than 10. In order to

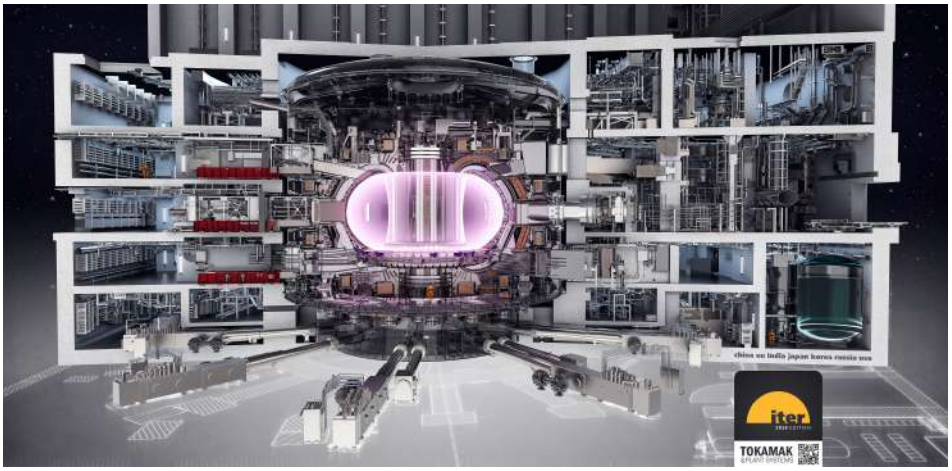

Figure 1.2: Project of the ITER tokamak and its plant systems

	Hydrogen	**Deuterium**
Beam Energy	$100keV$	$100keV$
Maximum Beam Source Pressure	$0.3Pa$	$0.3Pa$
Uniformity	$\pm 10\%$	$\pm 10\%$
Extracted Current Density	$> 355\frac{A}{m^2}$	$> 285\frac{A}{m^2}$
Beam on time	$3600s$	$3600s$
Co-extracted electro fraction	$0.5\frac{e}{H}$	$< 1\frac{e}{D}$

Table 1.1: Parameters of the SPIDER/ITER ion source

reach the temperatures required for the process to take place ohmic heating does not prove sufficient, due to the relationship between the plasma temperature and its conductivity, thus different ways of heating the plasma are needed. In RF heating the propagation of high frequency electromagnetic waves inside the plasma causes the electrons to oscillate, losing energy with each collision and gradually heating the plasma. In neutral beam injection a beam of high-energy particles is injected in the plasma to heat it via collisions. The particles have to be neutral to penetrate the magnetic confinement field without being deflected and, by choosing the same element as both fuel and as a heating system, the particles can just become part of the fusion process after being ionized. In order for the particles to be penetrating enough to reach the plasma ITER wil require two beam lines of 1 MeV each to supply 33 MW of heating power.

1.2 The PRIMA facility

The **P**adova **R**easerch on **I**TER **M**egavolt **A**ccelerator is a facility created to test and optimize the prototypes of the ion source and the neutral beam injector that will be used in ITER. The beam injector, called MITICA (**M**egavolt **ITER** **I**njector & **C**oncept **A**dvancement), is still under construction and its project is shown in Fig.1.4. The ion source, called SPIDER (**S**ource for **P**roduction of **I**on of **D**euterium **E**xtracted from **R**f plasma) and shown in Fig.1.3), will operate to the full extent of ITER's specifications, shown in Tab.1.1.

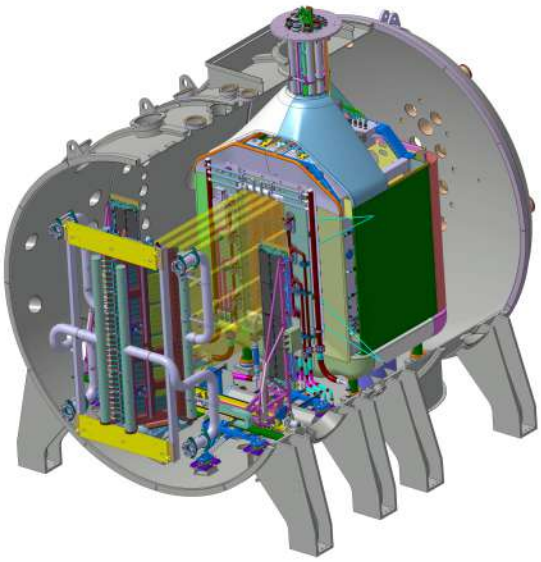

Figure 1.3: The SPIDER ion source

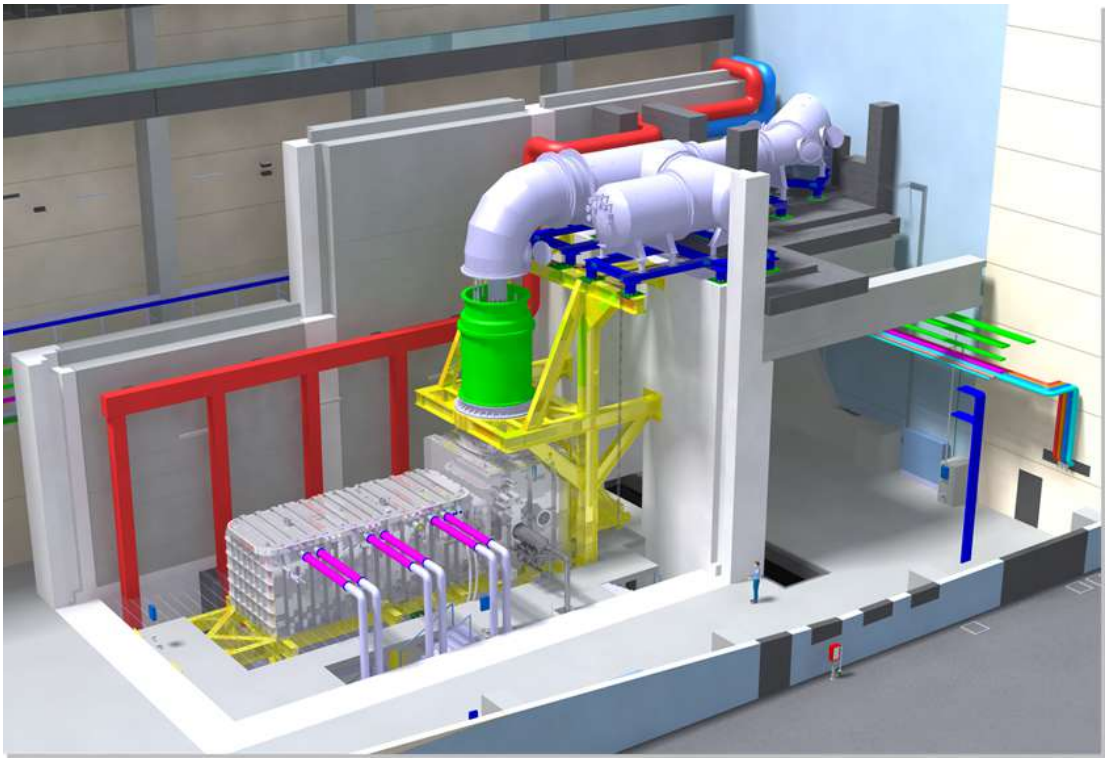

Figure 1.4: The MITICA accelerator

Initially the gas to be processed is injected in SPIDER's main chamber, then a source of electrons is used to free some electrons in the gas. An external coil generates an oscillating field that heats up the electrons, which then cede part of their energy to the surrounding atoms, ionizing them. The surface near the extraction grid, where the ions are pulled into the accelerator, is coated with Cs to improve the negative ion yield. The ions are then extracted and focused into 16 groups of 5x16 beamlets each through an extraction grid. The beamlets are then accelerated and to avoid their divergence a system of electrostatic lenses is used to compensate for the natural tendency of the ions to repel each other due to their negative charge. In the ITER experiment the beamlets would then be neutralized and injected into the plasma.

1.3 STRIKE

In order to study the characteristics of the beamlets, especially regarding the divergence and the bidimensional energy flux, a calorimeter called STRIKE (**S**hort-**T**ime **R**etractable **I**nstrumented **K**alorimeter **E**xperiment) will be used. Composed of 16 Carbon Fibre Composite (CFC) tiles with anisotropic properties, the calorimeter will be observed by two thermographic cameras pointed at its back. The reason for this positioning lies in the fact that the observation of the front would be disturbed by the optically emitting layer created by the beam interaction with the background gas, located between the beam source and STRIKE, and by the debris coming off the tile surface. The anisotropy of the materials making up the tiles is needed to preserve as much information about the beamlets as possible. Due to the impossibility of observing the front of the calorimeter, the energy flux will be reconstructed from the heat diffused through the tiles, so using a material with an isotropic thermal conductivity would lead to a shape on the rear side very different from the one striking the front of the calorimeter.

The calorimeter operates in a closed configuration but it can also be opened by moving apart the two sets of tiles, letting the beamlets pass through. In Fig. 1.5 a schematic representation of the calorimeter in its open configuration is given, while in Fig. 1.6 the position of the two thermal cameras is shown. It is clear from the figures that the images will be taken at an angle and thus strongly affected by the perspective but another

Figure 1.5: Project of the STRIKE calorimeter in its open configuration.

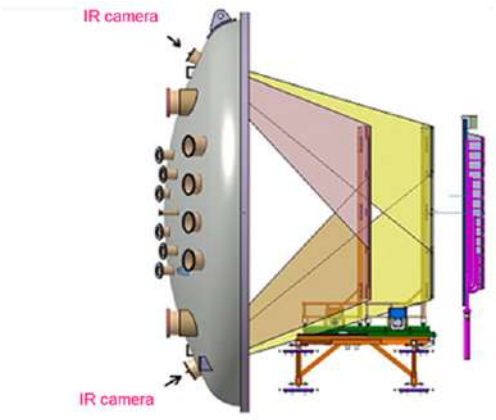

Figure 1.6: The thermal cameras pointed at STRIKE

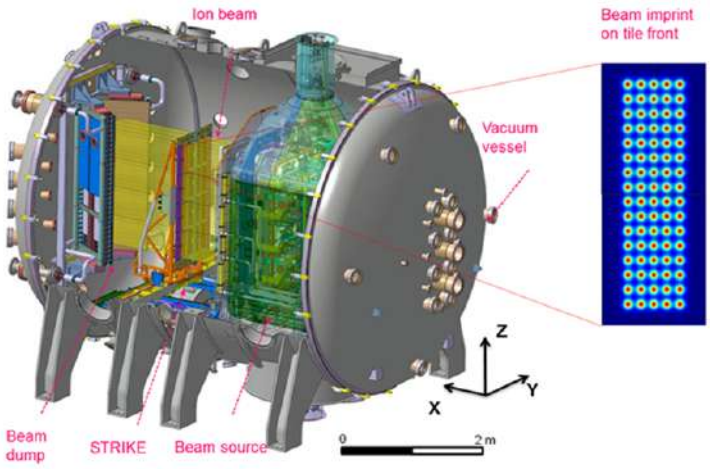

Figure 1.7: A simulation of the expected reconstructed image from the thermal cameras

convolutional neural network will account for this effect and reconstruct an orthogonal image from the two skewed views. The object of this thesis will be to prepare a convolution neural network that will take the reconstructed images and determine the parameters of the beamlets striking the calorimeter. An example of a possible image that will be passed to the neural network is shown in Fig. 1.7.

Chapter 2

Neural Networks

A neural network is a collection of elementary units called *neurons*, in analogy with their biological counterparts, organized in groups called layers. The idea behind the initial development of neural networks was that complex tasks could be achieved using a large number of elementary units, performing basilar operations. As the name implies, this idea is based on the evidence that biological cognitive functions must be driven by similar processes, given how a single neuron by itself is only able to perform very simple tasks.

While the first developments of neural networks were strongly linked to neuroscience and they were used as a mean to study and try to replicate the brain functions, nowadays research into deep neural networks has become a science in and of itself. Putting aside the implications towards the understanding of cognitive processes, a more utilitaristic approach to deep learning has become very common, recognizing its importance not for the insight it may provide in understanding the way the brain works but for its ability to perform much better than traditional machine learning in a variety of processes including computer vision, speech recognition,machine translation, data clustering and data generation. In the last few years neural networks have become ever more important in our society and are widely used for many common applications, while many tech giants are investing in the development of faster and better networks, both software-wise and hardware-wise. In particular their use is becoming more common in the field of research where the ability to quickly process a vast amount of data is invaluable, for example in predicting the interactions between molecules, thus helping the production of new drugs, and in the field of high energy physics where they have become a useful tool in the search for subatomic particles. For a more thorough review on the history of neural networks see [3].

2.1 Neurons, layers and activation functions

A neuron is the elementary component of a neural network and its role is to apply a parameter-dependent function, called *activation function*, to its inputs. In practice, many neurons are organized in layers, with each layer using the output of the previous one as input for its neurons. A network is then composed of a first layer, called *input layer*, whose neurons take the values of the input of the network and a succession of layers, called *hidden layers*, that end with a final layer, called *output layer*, having a number of neurons corresponding to the number of variables that need to be estimated, each outputting a single

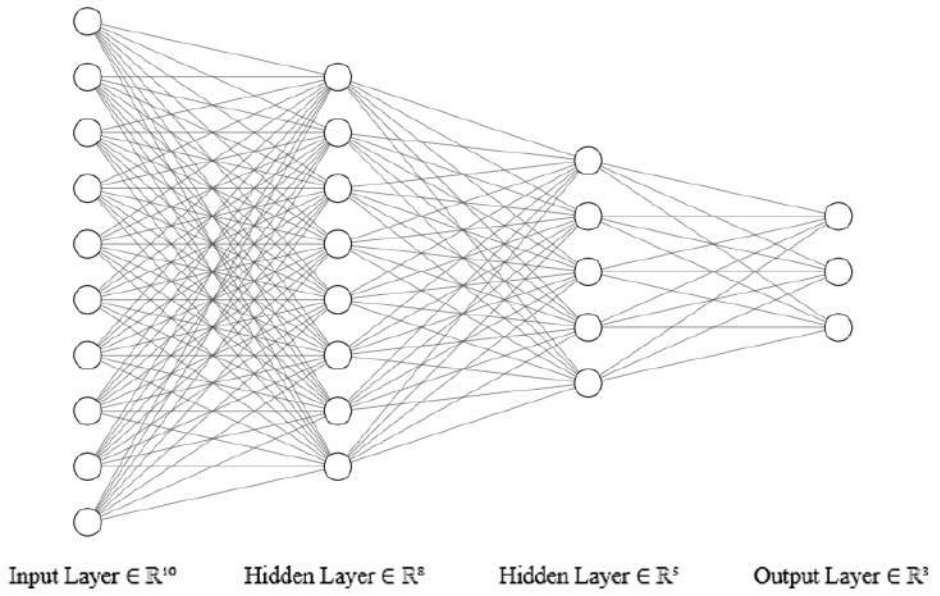

Figure 2.1: Schematic representation of a fully connected network with 2 hidden layers, in which every neuron is connected to all the neurons in the preceding layer

value and together making up the output of the network[1].A schematic representation of a network is given in Fig.2.1.

A commonly used practice is to use a linear transformation governed by parameters learned by the network followed by a fixed non linear transformation. The most widely used activation function is the rectified linear function or ReLU defined by $f(x) = max\{0, x\}$, then the complete function a neuron applies to its input becomes

$$f(x) = max\{0, W^T x + b\}$$

where W and b are respectively a matrix called the *weight matrix* and a vector, called *bias vector*, of learned parameters. Other activation functions still in use are the linear activation function, the Leaky ReLU described by $f(x) = \begin{cases} x, & \text{if } x > 0 \\ ax, & \text{if } x < 0 \end{cases}$, and the exponential linear unit or ELU described by $f(x) = \begin{cases} x, & \text{if } x > 0 \\ a(e^x - 1), & \text{if } x < 0 \end{cases}$. A graphic representation of the mentioned activation functions is given in Fig.2.2.

2.2 Training a network

The adaptability of neural networks derives from the great number of functions that can be represented with a composition of activation functions by changing their weights. In

[1]It is important to note that in ordinary networks the number of output variables must remain constant, as that will become important later on in the thesis work

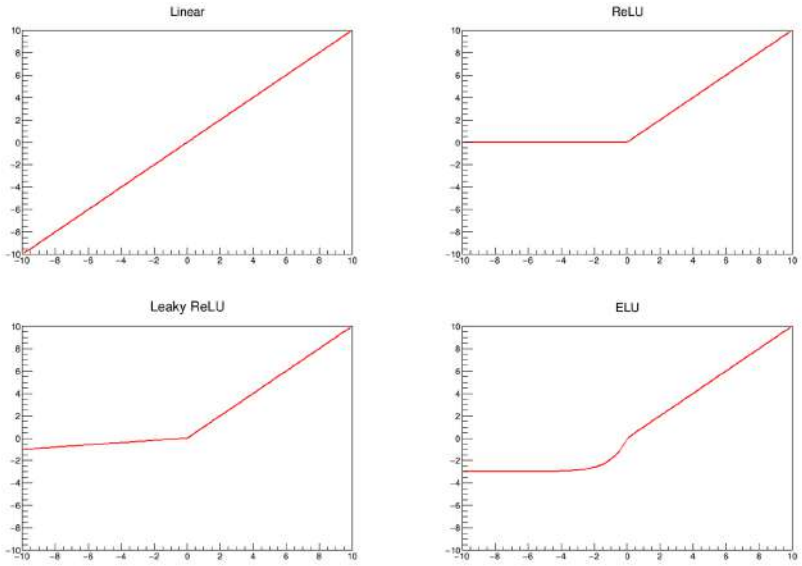

Figure 2.2: Graphs of various activation functions

particular the *universal approximation theorem* ensures that a network with a linear output layer and at least one hidden layer with any "squashing" activation function (e.g. the logistic sigmoid [2]) can approximate any Borel measurable function from one finite-dimensional space to another with an error that can be made as little as desired, while still being greater than zero, if enough hidden units are used (see [4]). The problem then becomes finding the right weights, a tedious if not impossible task to perform manually, that fortunately is not required. In fact the training of the network is an automatic procedure, performed by the network itself, that can be divided in two broad types: *supervised learning* and *unsupervised learning*. In the following sections a brief description of supervised learning[3] applied to neural networks is given, while for a more thorough review of supervised learning and a description of some unsupervised learning algorithms[4] see [3].

Supervised Learning

In supervised learning a set of values called *labels* and consisting in the "solutions" to the problem (or the values taken by the function that is being learned) is passed to the network in addition to the training-data in the form of a dataset of points (*input*,*label*). These labels are manually chosen, thus making the creation of large datasets costly and time consuming. In the absence of a dataset large enough for training purposes or if samples of the images on which the system will be used are unavailable, synthetically generated images can be used. For example in classification problems, where the network task is to find the right classification for the input among a fixed set of classes, the labels can be chosen as a set of zeros with a single one in the position corresponding to the correct class, as determined by a human operator. The training procedure then consists in modifying the weights to minimize the difference between the labels and the network output. A suitable function, called *cost function* or loss function, must be chosen to evaluate this difference. In general

[2] $\sigma(x) = \frac{1}{1+exp(-x)}$

[3] Used during the thesis work

[4] A type of learning algorithm not requiring the labeling of the data

different functions are best suited to different tasks, but the most common choices are the *mean squared error* for continuous data, shown in (2.1), and the *cross-entropy* for categorical data, shown in (2.2)

$$f(x) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i(x))^2 \tag{2.1}$$

$$f(x) = -\sum_{i=1}^{N} y_i log(\hat{y}_i(x)) + (1 - y_i) log(1 - \hat{y}_i(x)) \tag{2.2}$$

where N is the number of data-points, y_i the output of the network and $\hat{y}_i$ the corresponding label. Additional terms can be added to the cost function to perform different kinds of *regularizations*, expressing different kinds of prior belief about the structure of the function being learned (e.g. the L^2 parameter normalization[5] adds a term proportional to the sum of the L^2-norm of the weights, thus favoring smaller values for the parameters).

Gradient Descent and Backpropagation

Having chosen a cost function, an iterative algorithm to find the optimal parameters is still needed. Defining the optimal weights as the ones that minimize the cost function and denoting them as w^* we have

$$f_{min}(x) = f(w^*; x)$$

and remembering from calculus that a minimum is a stationary point, in which

$$\nabla_w f(w; x) = \mathbf{0}$$

and that the gradient of a function points in the direction of maximum increase, an iterative algorithm to find the minimum can be written as a succession of steps

$$f_{n+1}(w; x) = f_n(w; x) - \epsilon \nabla_w f_n(w; x)$$

where ϵ is the *learning rate* that determines how big are the steps taken with each iteration. A greater value of ϵ will mean faster convergence to a minimum but also a greater chance of "stepping over it", ending up in a point even further away than the one in the previous step as pictorically shown in Fig.2.3.

In reality the high number of parameters translates into a high-dimensional parameter space, where most of the stationary points are saddles and there is a high number of local minima that not necessarily lead to a low enough value of the cost function. Another consideration is that for large datasets computing the gradient for every element would be a very slow and computationally expensive process. A variation called *Stochastic gradient*

[5] Also known as weight decay

Figure 2.3: Effect of the learning rate during training. (a) too small, (b) optimal, (c) too large

descent(SGD), more thoroughly explained in [1], was developed as a way to bypass this and other problems. In SGD the gradient is computed only on a randomly selected mini-batch of a fixed number of elements, that can vary from a few units to a couple hundreds. Using reasonably large mini-batches the gradient will point on average in the same direction as the gradient computed using the entire dataset, moreover the stochasticity added to the network by randomly choosing the elements has been shown to have a regularizing effect, preventing the convergence to sharp minima and thus limiting overfitting[6]. In practice reaching the absolute minimum is not required and we are satisfied with the cost function settling in a local minimum with a low enough value. The method for computing the gradients is called *backpropagation*, firstly introduced in [8] and is based on the chain rule of calculus and the layered structure of the networks. Denoting with w^l_{jk} the weight of the connection between the k-th neuron of the layer l-1 and the j-th neuron of the l-th layer and with b^l_j the bias of the j-th neuron of the l-th layer, the activation function of this neuron can be written as

$$a^l_j = f(\sum_k w^l_{jk} a^{l-1}_k + b^l_j) = f(z^l_k) \tag{2.3}$$

with $z^l_k = \sum_k w^l_{jk} a^{l-1}_k + b^l_j$. Then defining the error of the j-th neuron in the l-th layer as

$$\Delta^l_j = \frac{\partial E}{\partial z^l_j} = \frac{\partial E}{\partial a^l_j} f'(z^l_j) = \frac{\partial E}{\partial b^l_j} \tag{2.4}$$

with E as the cost function and f' as the derivative of f evaluated at x. Using the chain rule we can also write Δ^l_j as

$$\Delta^l_j = \frac{\partial E}{\partial z^l_j} = \sum_k \frac{\partial E}{\partial z^{l+1}_k} \frac{\partial z^{l+1}_k}{\partial z^l_j} = (\sum_k \Delta^{l+1}_k w^{l+1}_{kj}) f'(z^l_j) \tag{2.5}$$

and by differentiating E by w^l_{jk} we have

[6]A network is said to be *overfitting* if it starts to learn features specific to the dataset used in the training and not representative of the general distribution, thus leading to a higher error on samples taken from the general distribution than on samples taken from the training dataset

$$\frac{\partial E}{\partial w_{jk}^l} = \frac{\partial E}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{jk}^l} = \Delta_j^l a_k^{l-1} \tag{2.6}$$

The backpropagation algorithm then consists in consequently computing the a^l's and z^l's for each layer, starting from the input one, then calculating the error Δ^L in the last layer using (2.3) and then propagating it backwards using (2.5) to determine the Δ^l's for each layer and then the cost function partial derivatives with (2.4) and (2.6). For a more detailed explanation see [6] Having determined the cost function and the activation functions, backpropagation allows a fast and computationally efficient way to find reasonable values for the network parameters.

2.3 Convolutional networks

A convolutional neural network (CNN) is a type of neural network, typically used for the analysis of data with a known, grid-like topology. The need for CNNs arises from the great number of input variables present in some forms of data (e.g. images) and in the possibility of using the translational invariance of features, that would be much harder to learn with a feed-forward neural network. In particular, CNNs use a mathematical operation called convolution (Appendix A.) in layers, called *convolutional layers*, that apply a filter, represented by a matrix called *kernel*, to the preceding layer. This operation is performed as a matrix multiplication between the kernel, usually a squared matrix, and an equally-sized matrix composed of a sub-set of the output values of the preceding layer. A convolutional layer is then composed of a certain number of filters and a fixed number of neurons, depending on the number of neurons in the preceding layer and the dimension of the kernel, for each filter. The number of kernels determine the *depth* of the layer and to each filter a bidimensional set of neurons is assigned. The output of each neuron is given by the multiplication of the kernel corresponding to the filter by the output of the neurons in the preceding layer surrounding the neuron in the same position as the one whose output is being computed. A visual representation is given in Fig.2.4. Each filter is applied to the entire picture, a group of neurons at a time, but the weights defining the filter, equal in number to the elements in the kernel, are shared in the corresponding layer, thus strongly reducing the number of parameters needed to describe the network. For example in the case of a 200x300 pixel input image, the number of inputs would be equal to the number of pixel and the number of weights required by a single fully connected layer of N neurons would be N*6 $\cdot 10^4$. Using a convolution layer with a 10x10 kernel and N filters the number of weights would be N*100. Different approaches can be taken when dealing with neurons close to the edge of the input layer, where there aren't enough neighboring neurons to complete the convolution operation: the missing entries may be replaced with zeros, a process called *zero-padding* , or the network may drop such neurons, leading to a reduction of the following layer that will have (N-$\frac{K}{2}$)x(N-$\frac{K}{2}$) neurons, with K being the dimension[7] of the kernel. To further reduce the dimensionality of the problem, *pooling layers* are used to retain a single value out of the outputs of multiple neighboring neurons, re-sizing a layer of a N neurons to a layer of $\frac{N}{P}$ neurons, with P the size of the pooling window,

[7] assuming a square convolution window

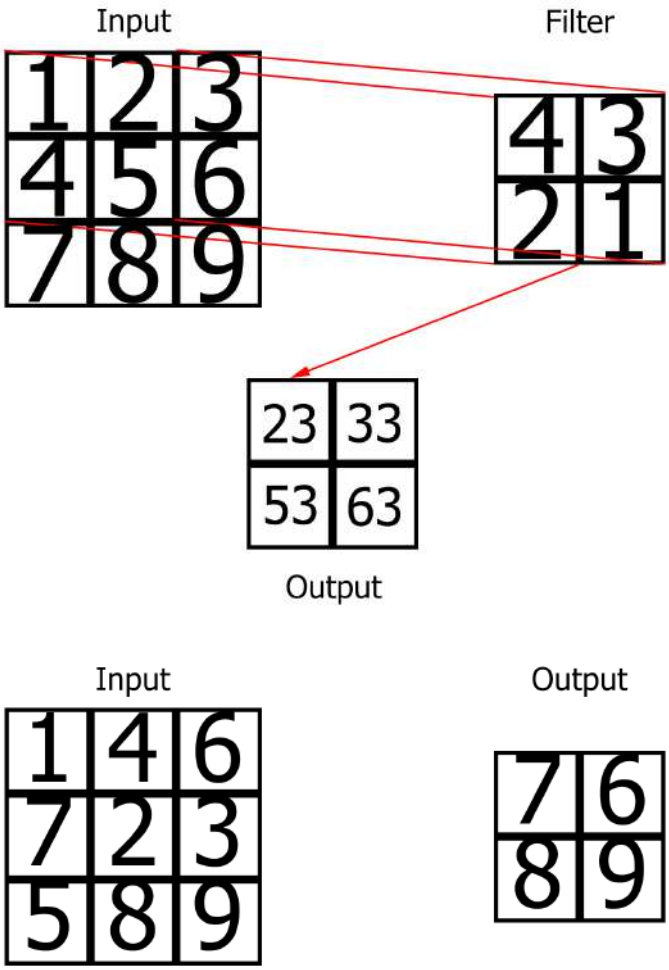

Figure 2.4: Representation of (a) a convolution operation with no padding (b) a maxpooling operation with a 2x2 pooling window

while maintaining the spatial structure of the information. A visual representation of the maxpooling operation, in which only the highest value in the pooling window is kept, is given in Fig.2.4. One of the main strengths of CNNs is their ability to easily learn the translational invariance of features. Consider, for example, a classifying task consisting in the recognition of various types of objects in a picture; the network will have to learn the features of each object but it will not be concerned with the position in which these features appear. By encoding a peculiar feature in a filter, it can quickly "scan" the image and register the results about the position and the amount of confidence with which the feature has been identified in the following layer. In the next section the thesis work is explained, describing the different network structures and strategies used to optimize the performance.

Chapter 3

Development of the neural network

The work developed for this thesis consisted in optimizing an already existing CNN able to estimate the position and the HWHM (half width half maximum) of energy profiles of the beamlets well represented by a 3D gaussian curve. As STRIKE was not operating at the beginning of this work, the CNN has been trained with a set of synthetically generated images representing expected beamlet shapes. A percentage of noise, corresponding to that expected during operation, has been added.

3.1 Starting network characteristics

The code has been developed on *Collaboratory*, an online Jupyter notebook environment, written in python and implemented using the Tensorflow library.

Starting training Dataset

The network was trained using synthetic images due to a lack of a sizable dataset of images taken from other experiments and the inability to gather more images from STRIKE, not operational during the thesis work. The images were generated by superimposing on a 200x300 pixels black image a gaussian with randomly generated parameters, sampled from the following distributions:

- **Position :** Uniform distribution over the image dimensions
- **Dimensions :** The 2 σ characterizing a bidimensional gaussian distribution, dubbed σ_x and σ_y in the following, were taken as the square root of their respective variances, sampled from a uniform distribution in the interval [120;400]
- **Angle :** Uniform distribution in $[0{:}\frac{\pi}{2}]$
- **Amplitude :** Fixed, always equal to one

An example of an image used for training and the corresponding labels are shown in (Fig.3.1 , Tab.3.1)

The labels were normalized in the interval [0:1] before being passed to the network, avoiding unnecessary bias and ensuring the same weight was given to all variables during training. This has been shown to improve performance and is now a standard procedure in the field.

Figure 3.1: Example of an image used for training

Parameter	Value
Y	63.68
X	215.07
σ_y	239.32
σ_x	332.64
θ	0.1579

Table 3.1: Parameters used to generate Fig.3.1

The dataset is composed of 20,000 images and after every 20 training steps a batch was used to test the loss function, determining the improvement from the last check.

Training algorithm

The mean squared error was used as cost function and the parameters were optimized using the ADAM algorithm [5]. ADAM is an algorithm based on the stochastic gradient descent discussed in (2.2) but including a momentum-based term. In general algorithms using momentum add the gradient of the cost function to an exponentially decaying sum of the gradients of previous learning steps, this adds stability to the training procedure and allows the training to "gain speed" in a certain direction in parameters space if successive steps have parallel gradients, allowing a faster navigation of flat regions. In particular, ADAM scales the momentum term by the squared root of an exponentially decaying sum of the squares of the previous gradients. An example of a training iteration is shown in (3.1).

$$\begin{aligned} g &= \frac{\Delta_\theta \sum_i L(f(x^{(i)};\theta), y^{(i)})}{n} \\ s &= \beta_1 s + (1-\beta_1) g \\ r &= \beta_2 r + (1-\beta_2) g \cdot g \\ \hat{s} &= \frac{s}{1-\beta_1^t} \\ \hat{r} &= \frac{r}{1-\beta_2^t} \\ \Delta_\theta &= -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}} + \sigma} \end{aligned} \tag{3.1}$$

with n as the batch size, t the number of the iterations, σ a small parameter to avoid numerical errors when dividing by small numbers and ϵ the learning rate.

Starting Architecture

The network was initially composed of 2 convolutional layers of 64 filters each, 2 fully connected layers with 1024 neurons each and an 8x8 kernel. The activation function used was ReLU for every neuron and the training consisted of 500 iterations using batches of 60 elements. The loss function average over 10 test batches was $(2.77 \pm 0.15) \cdot 10^{-3}$. The output on a test batch is shown in Fig.3.2.

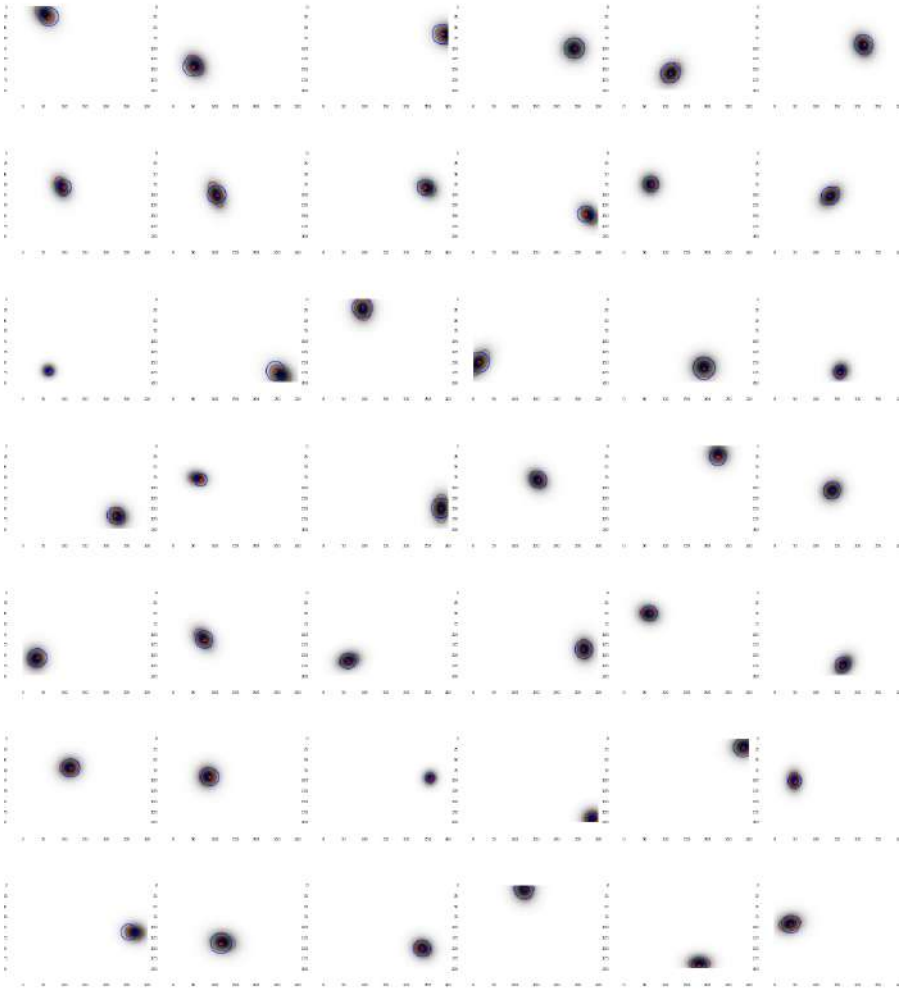

Figure 3.2: Output of the network in its original form. In red ellipses drawn using the parameters used for the images generation and in blue the circles drawn using the parameters found by the network.

3.2 Parameters optimization

Leaving the structure of the network unchanged, it was checked if the performance could be improved by changing the learning rate and implementing drop-out(Appendix B.). The results of various configurations are shown in Tab.3.2. The structure of the network was then changed while leaving the learning rate equal to $9 \cdot 10^{-5}$ and the drop-out equal to 0.1, the values with the best performance on the previous configuration. To reduce training time and compensate for the increased number of neurons *Max Pooling* was introduced. A dimensionality reduction technique, max pooling consists in taking the maximum out of a group of nearby neurons, effectively reducing the number of inputs for the following layer. A max pooling layer was added after every convolutional layer with a 2x2 pooling window, reducing the dimensions of the preceding layer by $\frac{3}{4}$ of their original value. The best results, consisting in a mean loss function of $(2.1 \pm 0.3) \cdot 10^{-3}$ after 500 training operations using batches of 60 elements, were obtained with a structure composed of 2 convolutional layers of 128 filters followed by a convolutional layer of 64 filters, 2 fully connected layers of 1024 neurons and a fully connected layer of 512 neurons. A schematic representation is given in Fig.3.4.

Varying the parameters with the new structure yielded the results in Tab.3.3, thus the parameters with the best performance in this new configuration were maintained for the rest of the discussion.

Decaying learning rate

To further improve the performance of the network, a decaying learning rate was implemented, corresponding to a non constant value of ϵ in (3.1). In particular a linearly decaying rate was used, starting with the optimal constant value found in the previous section. The results in terms of various rates of decay are shown in Tab.3.4 and Fig.3.5. The improvement of the network thus far and the output of the network with the best performance are shown on a test batch in Fig.3.6

Learning Rate (10^{-3})	**Drop-out**	**Mean Cost Function**	**Minimum Cost Function**
0.09	0	2.9	2.1
0.07	0	2.1	1.8
0.05	0	2.5	2.3
0.7	0	8.5	8.5
0.07	0.1	3.1	2.7
0.07	0.2	3.7	3

Table 3.2: Results of the variation of the network parameters

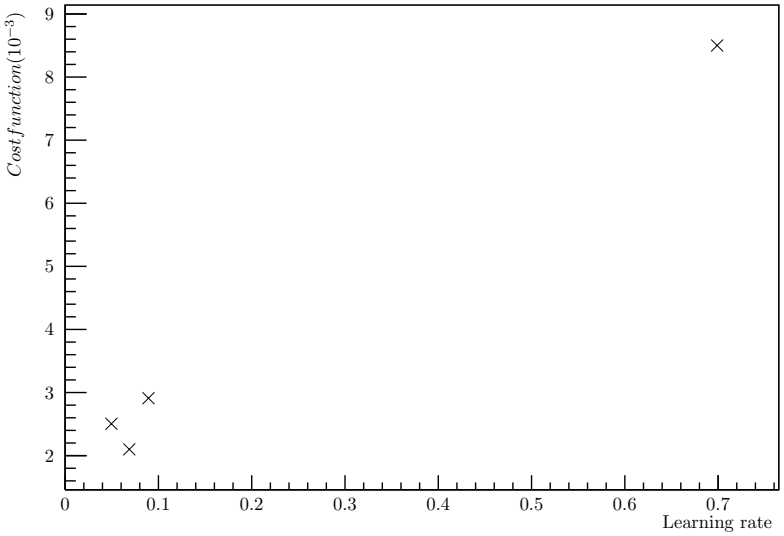

Figure 3.3: The variation of the network performance as a function of the learning rate

Kernel Dimension	**Learning Rate (10^{-3})**	**Drop-out**	**Mean Cost Function**	**Minimum Cost Function**
1	0.6	0.1	6.6	3.8
10	0.7	0.1	3.09	3.09
10	0.5	0.2	4.8	4.7
8	0.5	0.1	4.2	4
16	0.5	0.1	90.7	84.8
12	0.5	0.1	3.5	3.1

Table 3.3: Variation of the performance of the network in terms of the learning parameters in its second iteration

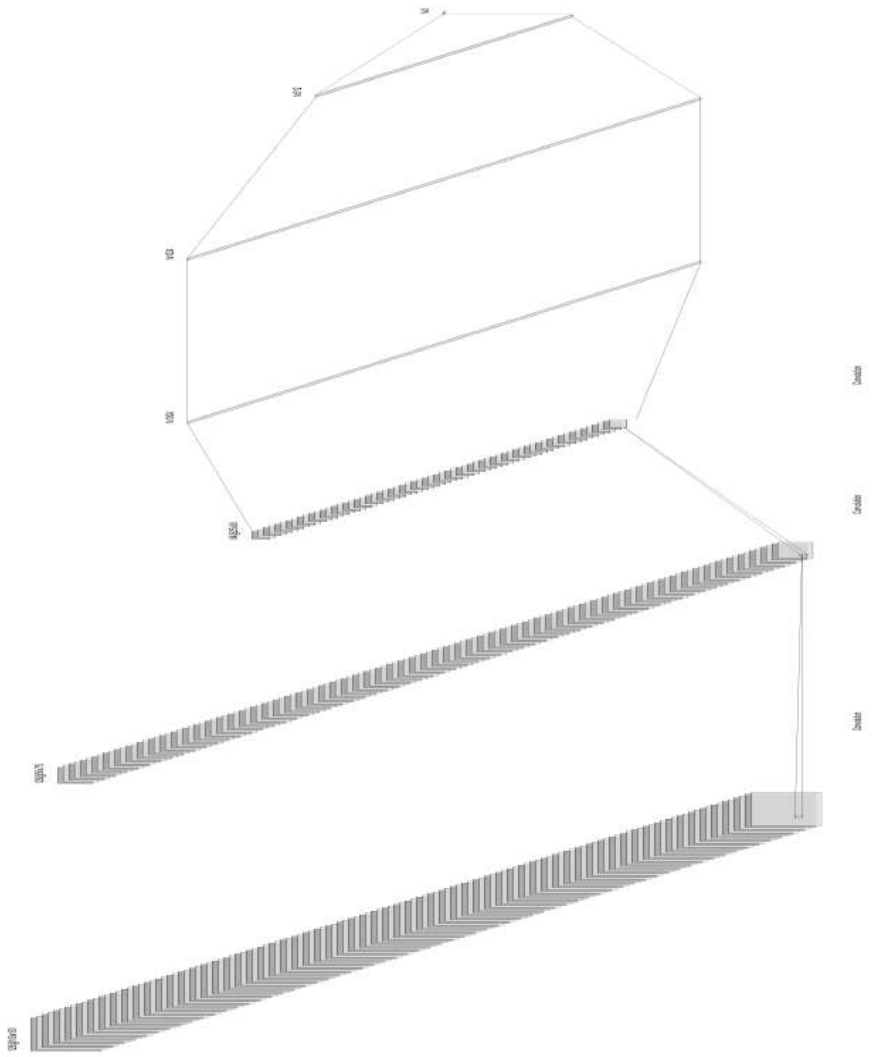

Figure 3.4: A schematic representation of the network

Fraction reached	Mean Cost Function	Minimum
1	2.1	1.04
0.1	1.5	1.23
0.04	1.3	1.1
0.001	1.4	0.95
0.004	2.1	2
0.005	1.3	1.3
0.007	1.7	0.89
0.002	0.86	0.79
0.0001	1.05	0.77

Table 3.4: Variation of the performance of the network in terms of the value of the learning rate in the last training iteration

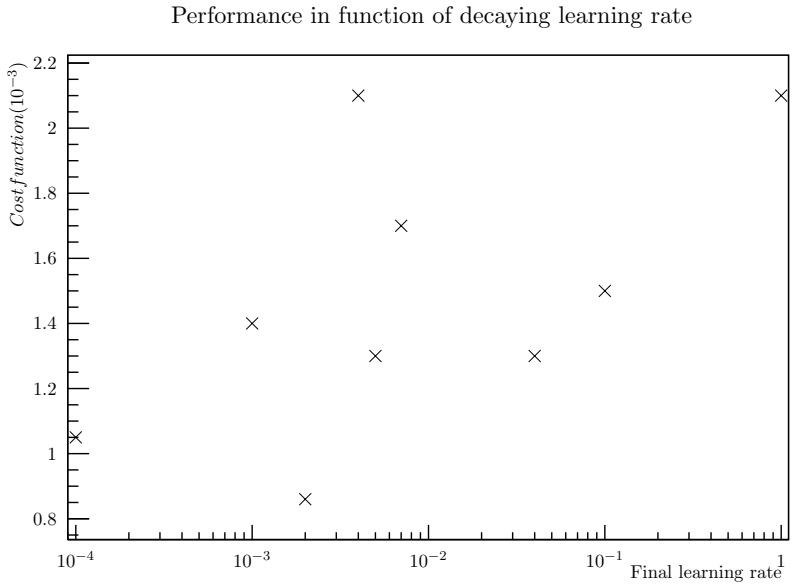

Figure 3.5: The variation of the network performance as a function of the decaying learning rate

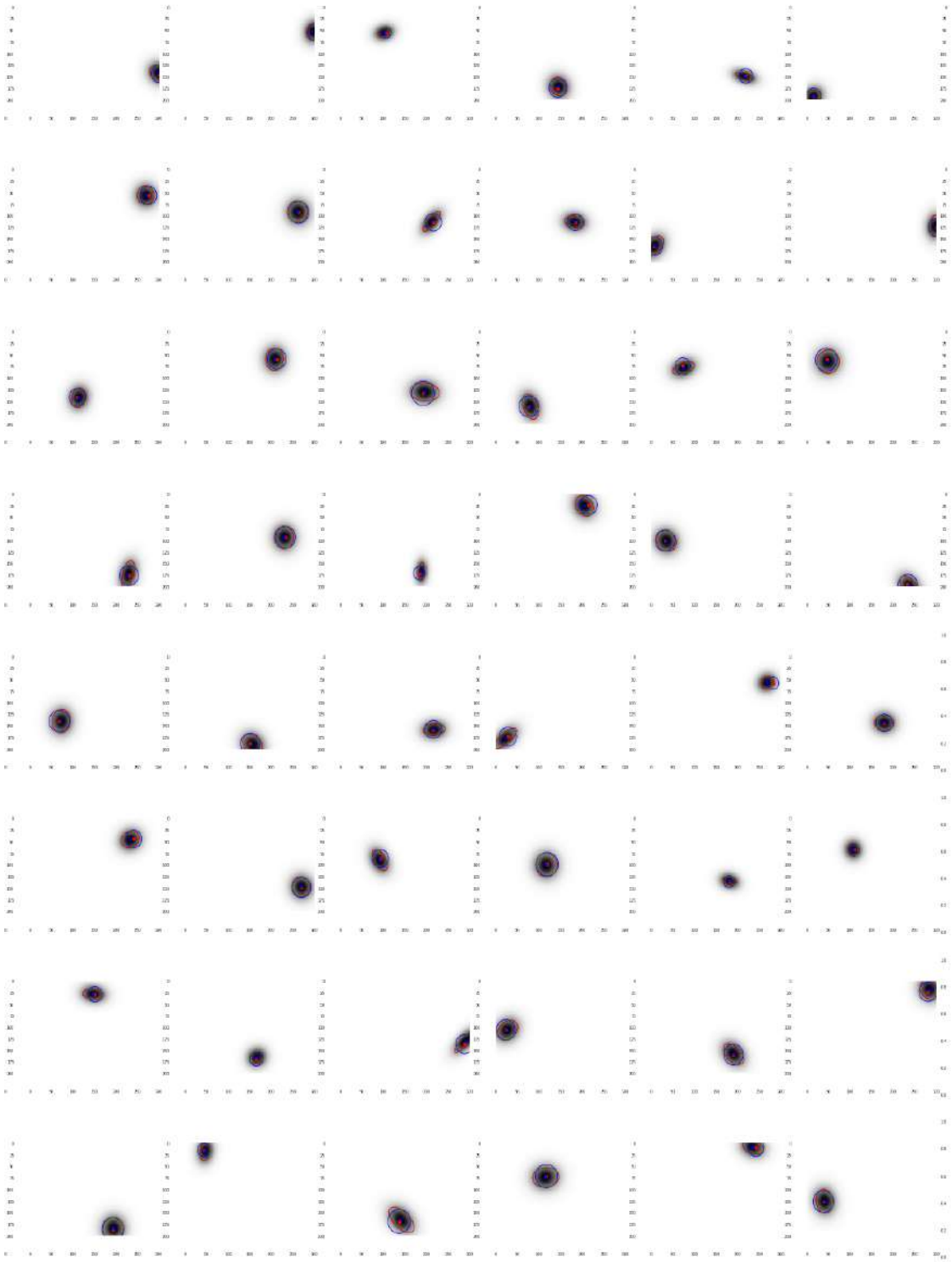

Figure 3.6: Output of the network with the best performance without determining the shape. In red ellipses drawn using the parameters used for the images generation and in blue the circles drawn using the parameters found by the network.

3.3 Shape Determination

Having obtained reasonable results in determining the centers, the same network was used to determine the σ used to generate the gaussians and their amplitude, uniformly generated in the interval [0.1;0.4]. After 1000 training iterations with batches of 60 elements each, the mean cost function on a testing set of 600 elements was $(5.8 \pm 0.55) \cdot 10^{-3}$ and the output is shown graphically in Fig. 3.7, where the true angle has been used to draw the ellipses representing the values estimated by the network to aid in the visualization of the error.

Noticing an evident rise in the error committed in the determination of the centers when compared to the previous instance, different network structures were tried with the results show in Tab.3.5 and Fig.3.8.

A further improvement has been achieved by changing the function describing the learning rate decay as shown in Tab.3.6.

Other choices of algorithms were tested:

- Mean squared error with different weights for the different variables as cost function
- Mean absolute error as cost function
- Linear activation function
- Leaky ReLU activation function
- ELU activation function
- Stochastic Gradient Descent as training algorithm
- Substitution of the Max Pooling layers with Average Pooling layers, taking the average of a group of neurons instead of the maximum

But these changes did not improve the network estimations.

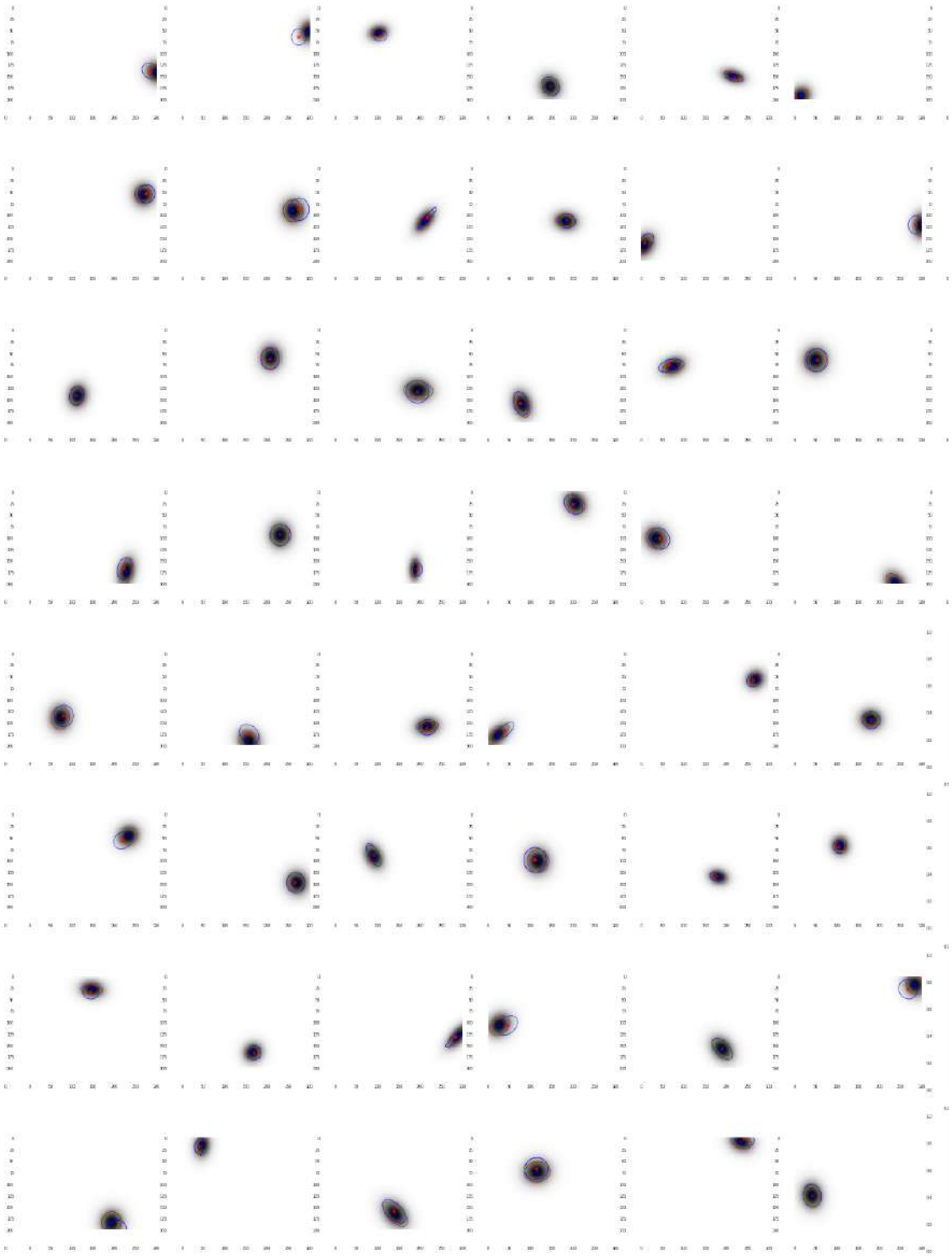

Figure 3.7: Output of the initial network estimating the shape in addition to the center. In red ellipses drawn using the parameters used for the images generation and in blue the ellipses drawn using the parameters found by the network.

N. of convolutional layers	N. of fully connected layers	Mean cost function	Minimum
2(256),1(128)	2(2048),1(1024)	N.C.	
2(160),1(128)	2(2048),1(1024)	8.6	5.1
1(160),1(144),1(128)	2(2048),1(1024)	N.C.	
1(144),2(128)	2(2048),1(1024)	8.6	5
3(128),1(64)	2(2048),1(1024)	4.2	2.8
4(128)	2(2048),1(1024)	9.2	3.4
2(128),1(64)	2(2048),1(1024)	5.8	3.4
3(128)	2(2048),1(1024)	N.C.	
3(128)	1(4096),1(2048),1(1024)	N.C.	
3(128)	2(2048),1(1024)	N.C.	
3(128)	1(2048),2(1024)	N.C.	
3(128)	1(2048),2(1024)	6.3	
3(128)	3(1024)	5.5	3.2
2(128),1(64)	3(1024)	9.3	8.1
1(128),3(64)	3(1024)	6.5	6
3(256),1(128),1(64)	3(1024)	6.7	6.3
3(256),1(128)	3(1024)	N.C.	
2(512),1(256)	3(1024)	N.C.	
1(512),1(256),1(128)	3(1024)	6.9	6.2
2(128),1(64)	1(1024)	8.5	6.6
2(32)	1(1024)	6.9	4
2(16)	1(256)	6.3	5
2(16)	1(64)	13.1	11.7
2(16)	1(128)	9.9	8.9
2(32)	1(256)	7.5	5.3

Table 3.5: Performance of various possible structures for the network, in parentheses the number of filters/neurons per layer. N.C. used if the network failed to converge to reasonable values

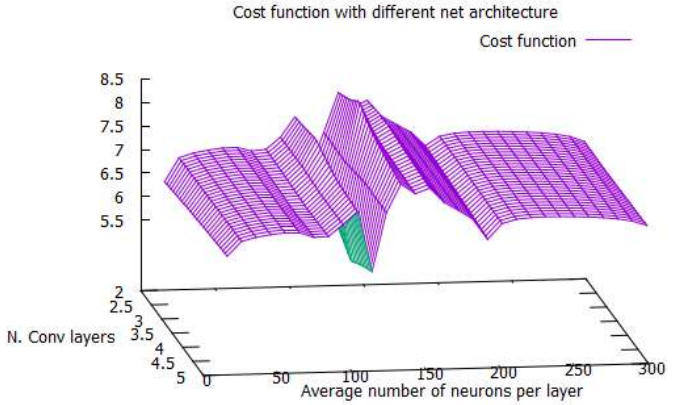

Figure 3.8: Graph showing the mean cost function as a function of the number of layers and the average number of filters per layer

Decay Function	**Mean cost function**	**Minimum**
Linear to 10^{-3} of initial value	3.8	2.8
Linear to $3 \cdot 10^{-4}$ of initial value	8.5	4.4
Linear to $5 \cdot 10^{-4}$ of initial value	4.5	2.9
Linear to 0	5.1	3.5
Cubic to 0	4.7	3.3
Sixth power to 0	5.6	3.3

Table 3.6: Performance of the network with various decay functions

Batch Normalization

Based on the observation that centering the inputs around zero with respect to the bias granted a better performance, *Batch Normalization* is a type of regularization consisting in the normalization of the output of a layer by the mean and variance of the mini-batch, thus ending up with a distribution with zero-mean and unit variance then used as input for the following layer. To prevent a reduction in the capacity of the network due to the limiting restriction of having a normalized distribution between each layer, a linear function is applied to the newly normalized output as shown in (3.2)

$$\begin{aligned} z \to \hat{z} &= \frac{z - \mathbb{E}^B[z]}{\sigma^B z} \\ \hat{z} \to z &= \alpha \hat{z} + \beta \end{aligned} \tag{3.2}$$

Where the expectation value $\mathbb{E}$ and the variance σ are estimated on the mini batch and α and β are learnable parameters. It may seem pointless to normalize a variable to have zero mean only to rescale it again to the new value β, but it must be considered that the new parameters have different learning dynamics compared to the unnormalized variables and are much easier to learn. For a more in-depth discussion see [3]. As shown in Tab.3.7 the results of the introduction of a different number of batch normalization layers did not cause a significant improvement.

Contrast Normalization

An improvement in the determination of the shape was sought by using different types of contrast normalization. In particular *Global Contrast Normalization* was implemented, consisting in rescaling the contrast of an image by removing the mean of the pixels intensity from the value of each pixel and then dividing by the standard deviation computed over all the pixels of the image. This preprocessing scheme, used to highlight the difference in bright and dark areas of a picture, did not improve performance, leaving the cost function fluctuating around the value of 20. The implementation of *Local Contrast Normalization*, consisting in removing the mean of surrounding pixels from the value of each pixel and dividing the result by their standard deviation, thus emphasizing edges and contours, slowed the training time to the point of making it unusable.

N. Convolution Layers	N. Batch-Norm layers	Mean cost function	Minimum
1(128),2(64)	3	7.0	4.4
3(64)	1	>20	>20
1(128),2(64)	1	17.4	12
3(64)	2	14.8	11.1
1(128),2(64)	2	>20	>20
1(256),1(128),1(64)	2	11.7	9.8
1(256),1(128),2(64)	2	9.1	6.5
1(256),1(128),2(64)	3	>60	>60
2(256),1(128)	3	>60	>60
1(256),2(128),2(64)	4	4.8	2.4
1(256),3(128),1(64)	4	4.4	2.7

Table 3.7: Introduction of Batch Normalization

First Model	2C(64), 1F(128), MP	2C(64),1F(128),MP	2C(64),1F(128),MP	2C(128),1C(64),1F(512),MP	2C(128),1C(64),2(512),MP	3C(128),3F(512),MP
Second Model	2C(64),1F(128)	2C(128),1F(128)	2C(128),1F(128),MP	2C(128),1C(64),1F(1024),MP	2C(128),1C(64),2F(512),MP	3C(128),3F(512),MP
Third Model	2C(64),1F(128),AP	2C(64),1F(128),AP	2C(64),1F(128),AP	2C(128),1C(64),1F(1024),AP	2C(128),1C(64),2F(512),AP	3C(128),3F(512)AP
Fourth Model	2C(64),1F(128)	3C(64),1F(128)	1C(128),2C(64),1F(128),MP	2C(128),1C(64),1F(1024),MP	2C(128),1C(64),2F(512),MP	3C(128),3F(512),MP
Fifth Model	2C(64),1F(128)	1C(128),1C(64),1F(128)	1C(256),1C(128),1F(128),MP	2C(128),1C(64),1F(1024),MP	2C(128),1C(64),2F(512),MP	3C(128),3F(512),MP
Mean Cost Function Model 1	8	12	9	10	5	6
Mean Cost Function Model 2	24	13	8	5	6	5
Mean Cost Function Model 3	7	8	9	8	8	8
Mean Cost Function Model 4	16	9	8	5	9	6
Mean Cost Function Model 5	25	13	8	10	7	7
Cost function on test set	46	9	6	7	7	7

Table 3.8: Performance of the 6 ensemble of 5 elements tested, the notation for the structure of the model is a succession of terms of the form nT(N), where n is the number of layers, T is the type of layer (C for convolution, F for fully connected) and N is the number of filters/neurons per layers. MP indicates the use of Maxpooling while AP the use of average pooling

Ensemble Learning

Noticing how the performance in the determination of the centers after the introduction of the shape variables was still not as precise as in the initial network, a type of model averaging was tested. Based on the idea that different networks will not make the same errors, different types of ensembles were used to estimate the outputs, taken as a simple average over the outputs of the single networks. In general, if the errors made by the networks are completely independent and the ensemble is composed of k elements, a reduction of $\frac{1}{\sqrt{(k)}}$ on the error can be expected, while in the case of complete correlation the error is left unchanged. In some cases even using the same structure for the different members of the ensemble may improve performance, as the difference in the randomly chosen initial parameters and mini batches used for training can cause partially independent errors. It is also interesting to note that Drop-out(Appendix A.) can be interpreted as a kind of ensemble method. A more detailed discussion on Ensemble learning can be found on [3] and [6].As shown in Tab. 3.8, none of the different configurations caused a significant improvement, suggesting a strong correlation between errors.

Specialized Networks

In the end the best performance was obtained by using 2 specialized networks, one to determine the position and another one to determine the shape. For the first network the same architecture as the one that gave the best results before introducing the shape was

N. Convolution Layers	Mean cost function	Minimum
3(128),MP	45	35
3(256),MP	44	37
2(128),1(64),MP	44	35
3(64),MP	6.7	4.7
3(64),AP	45	39
3(64)	21	19
1(256),1(128),1(64),MP	43	35
1(128),2(64),MP	9.7	6.9
2(64),1(32),MP	44	34
3(64),3BN,MP	109	78
3(64),1BN	44	37

Table 3.9: Structures tested for the shape-determining network.The notation for the structure of the model is a succession of terms of the form n(N), where n is the number of convolution layers and N is the number of filters per layer. MP indicates the use of Maxpooling while AP the use of average pooling, nBN indicates n layers of Batch Normalization layers. In all the architectures the convolution layers were followed by 3 fully connected layers of 1024 neurons each

RMS	Loss	X	Y	$\mathbf{var_X}$	$\mathbf{var_Y}$	A
Double Network	5.5 ± 1.1	2.7 ± 0.5	3.6 ± 0.9	49 ± 7	49 ± 7	0.04 ± 0.01
Single Network	5.8 ± 1.1	7 ± 1	10 ± 2	46 ± 7	47 ± 5	0.034 ± 0.005

Table 3.10: Comparison of RMS of the single variables estimation on a test sample size of 10 batches of 60 elements each

used, while for the second network multiple structures were tried out, as shown in Tab.3.9.

Training the two networks in succession resulted in a mean value of the cost function of 0.9 ± 0.2 for the position-determining network and of 7 ± 2 for the shape-determining network. In Fig.3.9 the result on a test batch in shown, where once again the inclinations of the ovals are taken from the values used for the images generation for ease of representation.

Noticing how the use of two networks did not improve the mean cost function but visibly appears to obtain a better performance, the errors on the determination of the single variables were computed and compared between the single-network architecture and the double-network one. As shown in Tab.3.10, the double network presents a strongly reduced error in the estimation of the position, compensated by an increase of the error in the determination of the σ's. Preferring this subdivision of the error, the double-network structure was used.

3.4 Angle determination

Adding the angle as an output variable to the shape-determining network caused a strong increase in the cost function, making it unable to converge to reasonable values. As shown in Tab.3.11, changing the structure of the network did not improve the performance of the system.

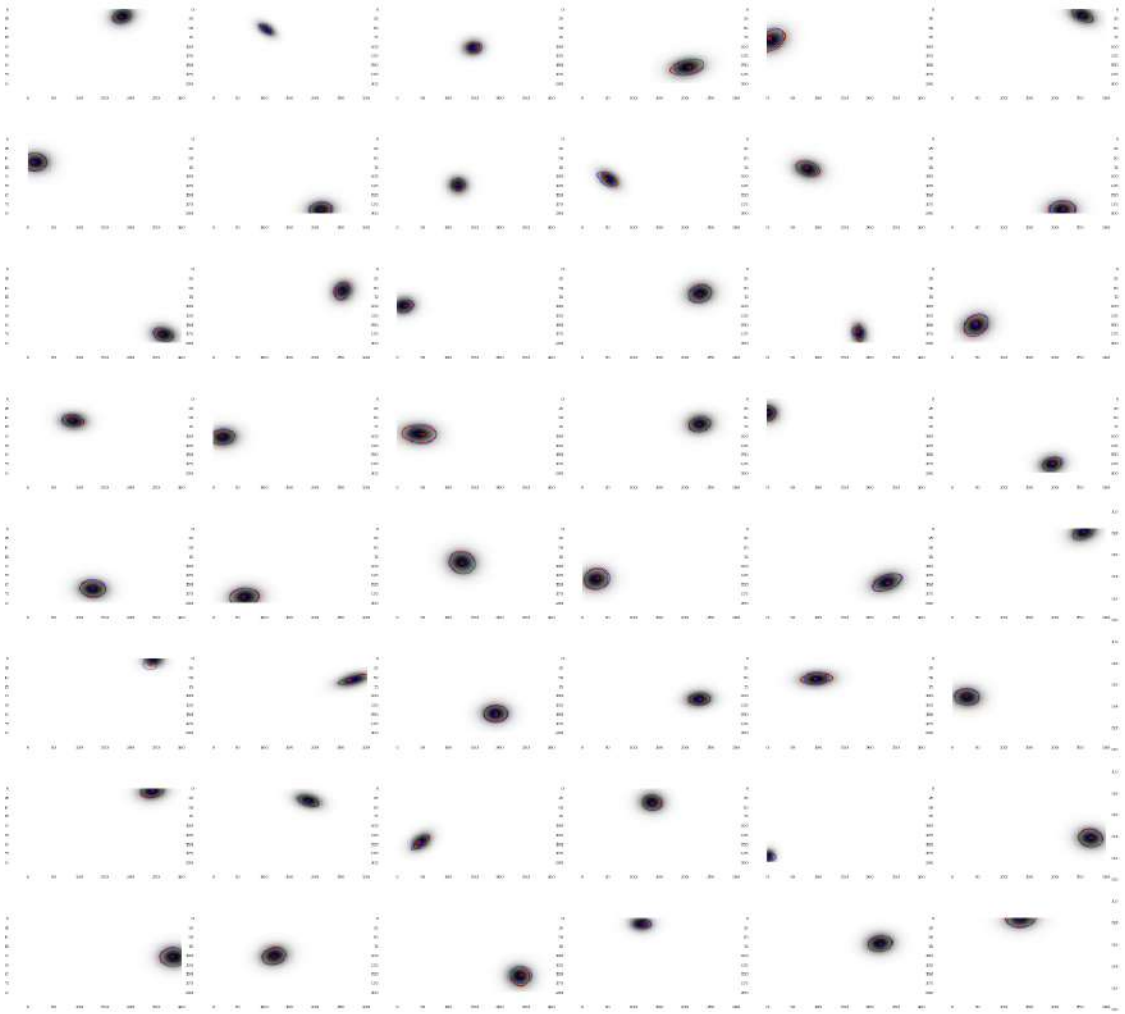

Figure 3.9: Output of the final network estimating the shape in addition to the center. In red ellipses drawn using the parameters used for the images generation and in blue the ellipses drawn using the parameters found by the network.

N. Convolution Layers	**Loss**	**Minimum**
3(64)	76	62
3(128)	74	66
3(256)	76	61
1(256),1(128),1(64)	75	61

Table 3.11: Structure of the shape and angle determining network.The notation for the structure of the model is a succession of terms of the form n(N), where n is the number of convolution layers and N is the number of filters per layers. In all the architectures the convolution layers were followed by 3 fully connected layers of 1024 neurons each

N. Convolution Layers	Loss	Minimum
3(128)	23	18
3(64)	44	39
1(128),2(64)	44	39
2(128),1(64)	43	40
1(256),2(128)	20	14
3(256)	43	38
2(256),1(128)	42	38
1(256),1(128),1(64)	36	30
1(256),2(128),1(64)	44	38

Table 3.12: Angle determining network structure for ovaloid shapes.The notation for the structure of the model is a succession of terms of the form n(N), where n is the number of convolution layers and N is the number of filters per layers. In all the architectures the convolution layers were followed by 3 fully connected layers of 1024 neurons each

Associating the inability of the network to converge to the distribution used for the generation of the σ's, making most shapes similar to circles, for which an angle of rotation is not defined, new parameters were chosen for the images generation. At first the interval used for the σ's generation was expanded, going from $[0;\frac{\pi}{2}]$ to $[0;\pi]$, but this lead to no major improvement. Training the second network on a dataset composed solely of gaussians for which the difference between σ_X^2 and σ_Y^2 was greater than 80 on the [120;400] interval, thus resembling ellipses more than circles, lead to reasonable results, as shown in Tab.3.12.

Training the shape-determining network only on gaussians with a strong parameter requirement did not cause a loss of generalization, given how a completely wrong estimation of the angle used to generate a gaussian resembling a circle is completely acceptable. In Fig.3.10 the optimal results obtained with the complete network are shown, corresponding to a mean cost function of 0.7 ± 0.15 for the determination of the centers and 36 ± 13 for the determination of the shape.

3.5 Noise

Having optimized the network for the determination of all the required variables, white noise was included in the images generation. Using a uniformly distributed noise in the interval [0;0.07], equivalent to an intensity of up to 70% of the amplitude of the gaussians, caused a great increase in the cost function and the RMS of the second network, while leaving the errors of the first one virtually unchanged. Noticing a very significant increase in the error of the angle, a three-network structure was implemented. Using a network for the the estimation of the centers, one for the estimation of the amplitude and the σ and a third one for the inclination of gaussian granted the best performance, shown in Tab.3.13 and Fig.3.11. The different configurations for the third network all resulted in similar values for the loss function, thus the system with the fewer number of neurons was chosen to maximize efficiency.

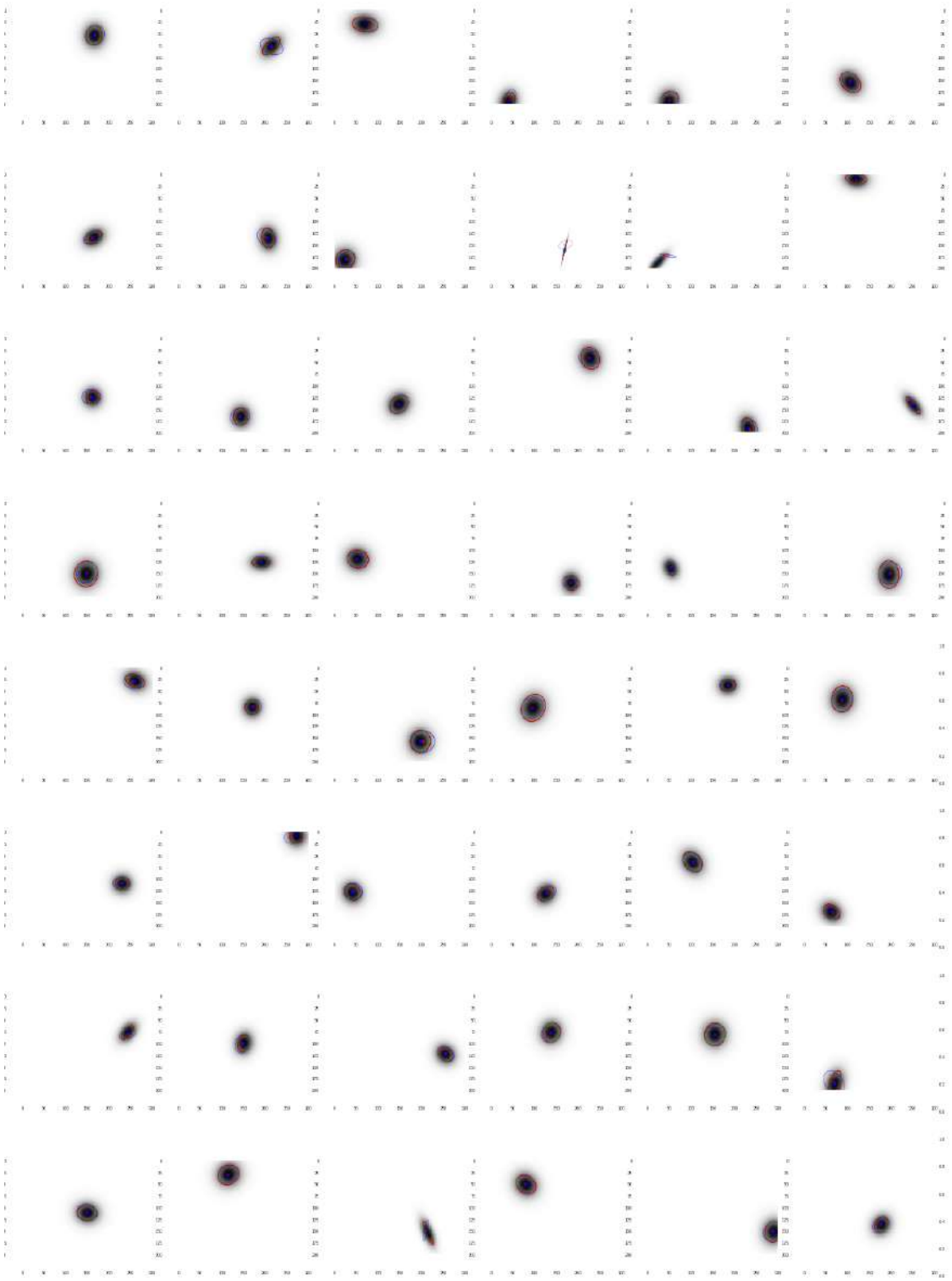

Figure 3.10: Output of the final network estimating all the required variables. In red ellipses drawn using the parameters used for the images generation and in blue the ellipses drawn using the parameters found by the network.

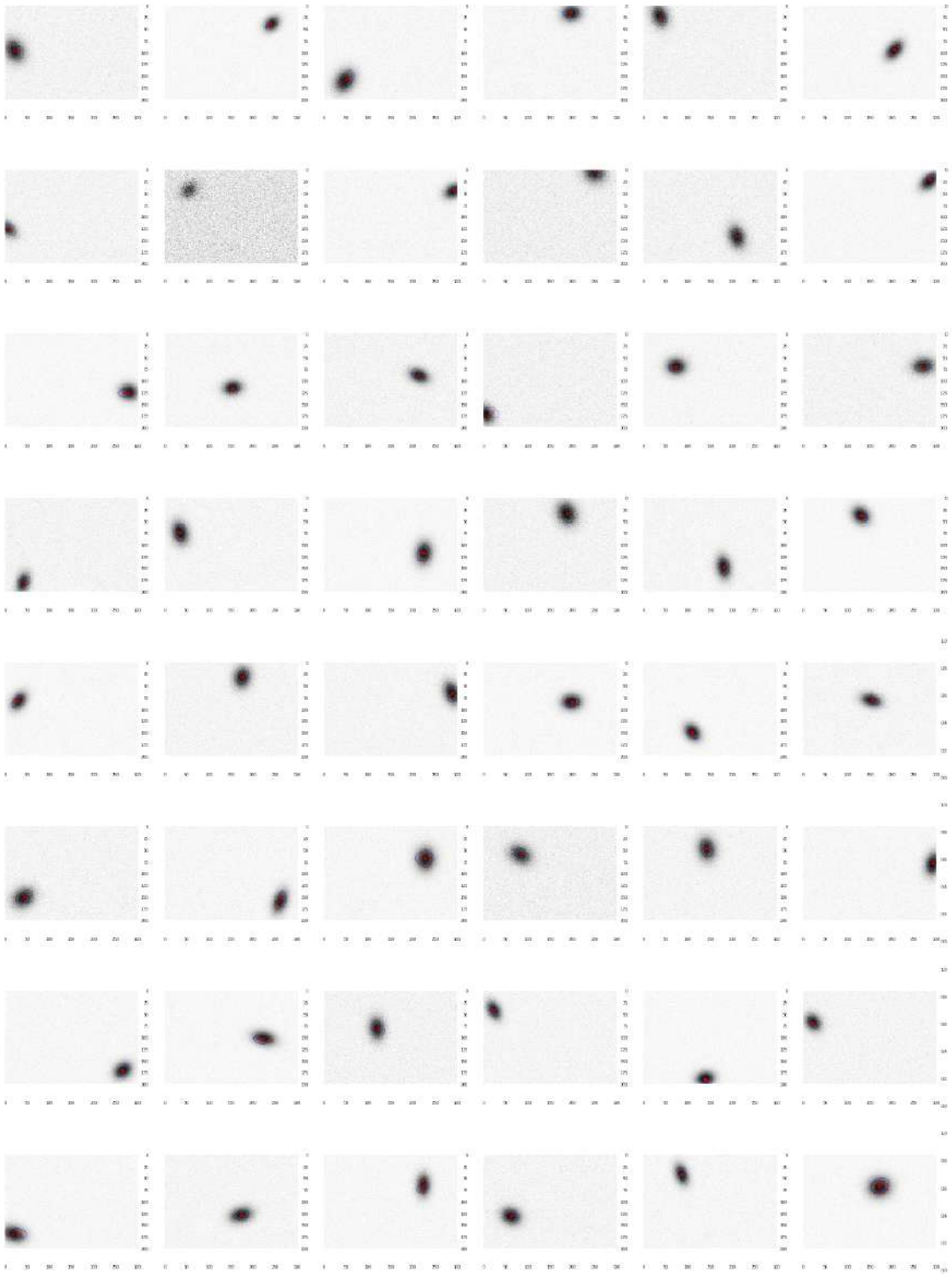

Figure 3.11: Output of the network with background white noise. In red ellipses drawn using the parameters used for the images generation and in blue the ellipses drawn using the parameters found by the network.

	Loss	X	Y	$\mathbf{var_X}$	$\mathbf{var_Y}$	θ	A
RMS	45.8	2.76	4.3	35.2	25.2	0.42	0,02

Table 3.13: RMS for the various variables with the optimal architecture in a noisy environment

3.6 2-Gaussians generalization

Having optimized the performance for the determination of the parameters of images with a single gaussian, the ability of the network to adapt to images with either one or two gaussians was tested. As mentioned in (2.1) the number of output nodes and labels used for each image must be kept constant, thus in order for the network to return the characteristics of two different shapes, the number of outputs had to be doubled. However, given the necessity to process not only images with two gaussians but also ones with a single gaussian, a means to determine if a set of parameters in output was referring to an actual shape or was just a set of "ghost" values referring to an unexisting gaussian was needed. Two different datasets were generated, each with a 50% probability to create an image with a single or with two gaussians but differing in the mechanics used to identify the number of shapes in an image. For the first dataset(*single-bit* dataset in the following) a single output variable was added for each gaussian, having a value of 1 if the gaussian was generated and 0 otherwise, and in the case of non-existence all the corresponding labels used for training were set to the value of 0. In this simple approach the value of the newly created output variable could be interpreted as the *degree of belief* of the network in the existence of the corresponding shape. For the second dataset the number of variables was doubled for each gaussian and for each desired parameter 2 output nodes were used. The first trained to determine the normalized value used in the generation and the second one to determine its complementary, while keeping both values equal to zero if the corresponding gaussian was not generated. The idea behind the second dataset was that the zeroes corresponding to non-existing shapes, still used during training, could "bring down" the estimation of the network and the use of the complementary value could help offset this error. Another design problem in the generation algorithm was the ambiguity caused by the order in which the gaussians were generated. The training procedure of the network depends on the positions of the labels and during each training step a correspondence between the position of the label and the feature of the image associated to its value is expected. Considering a naive approach where 2 gaussians are generated and the corresponding labels are stored in the order in which the generation took place, this ambiguity becomes evident. Just by looking at the generated image it is impossible to determine which shape was generated first, thus to a single image two different "correct" answers could correspond, differing only in the order of the subset of values representing each gaussian. The criterion chosen to remove this ambiguity was to register the labels in descending order of the x-coordinate, resulting in a label for each image with the following pattern:

$$[X_{max}, Y_{X_{max}}, var^x_{X_{max}}, var^y_{X_{max}}, \theta_{X_{max}}, A_{X_{max}}, p_{X_{max}}, X_{min}, Y_{X_{min}}, var^x_{X_{min}}, var^y_{X_{min}}, \theta_{X_{min}}, A_{X_{min}}, p_{X_{min}}]$$

Only the position-determining network was used to determine the best performance and for the first dataset both an all-linear output configuration and one using the soft-max unit to determine the p's were tested, with the results shown in Fig.3.12.

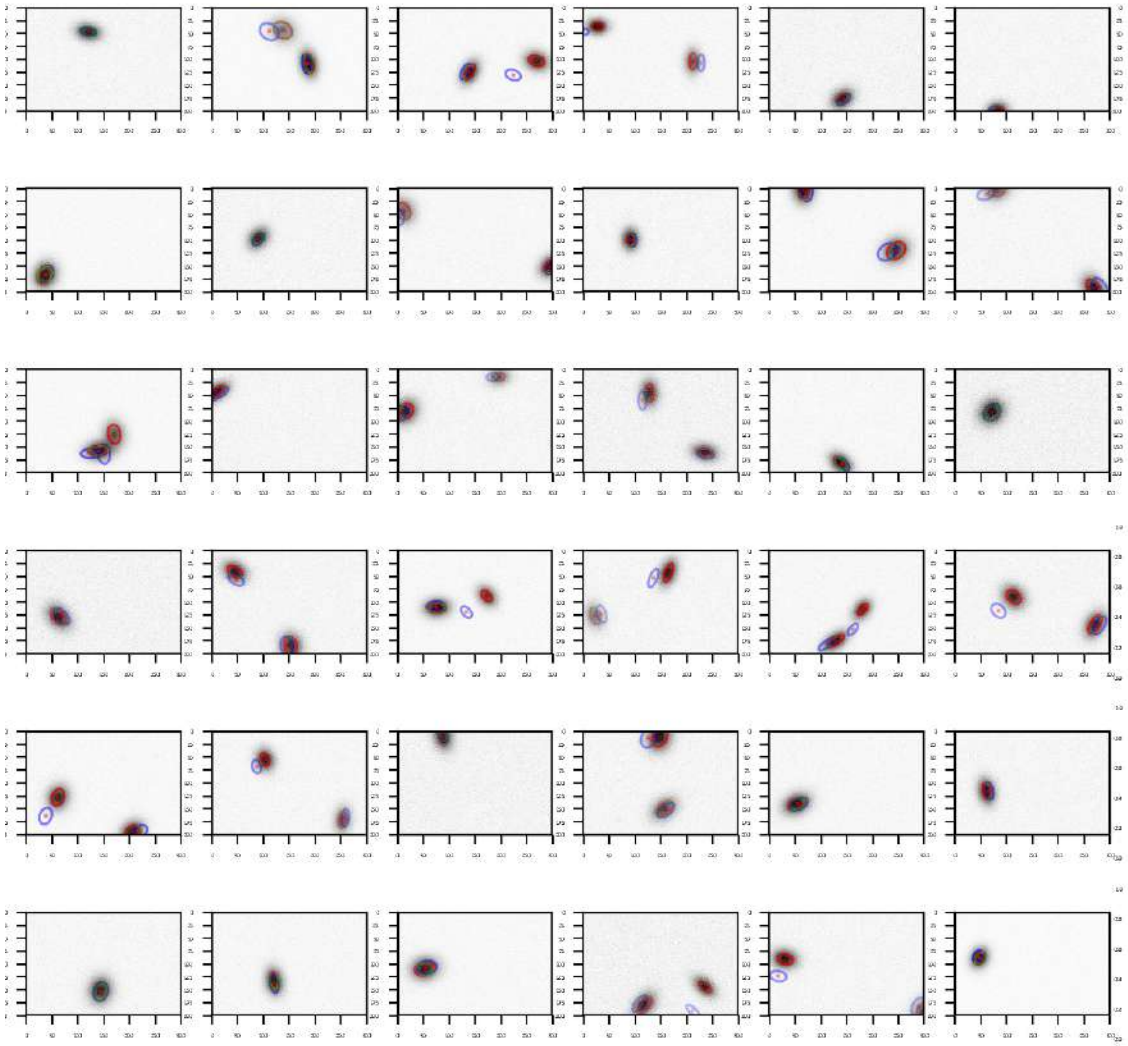

Figure 3.12: Output of the network with 1 or 2 gaussians. In red ellipses drawn using the parameters used for the images generation and in blue the circles drawn using the position found by the network.

	$\mathbf{X_{max}}$	$\mathbf{Y_{max}}$[a]	$\mathbf{X_{min}}$	$\mathbf{Y_{min}}$
Single-bit	4.1	9.1	17.9	17.5
Swapping	6.6	7.5	7.7	9.9
Only X	4.2	/	4.3	/
Only Y	/	5.9	/	7

[a]Referring to the maximum value of Y or the Y corresponding to the gaussian with maximum X depending on the dataset, as explained above

Table 3.14: Comparison of the RMS with the different dataset structures using data taken from batches of 60 elements

	$\mathbf{Var_x^1}$	$\mathbf{Var_y^1}$	θ^1	$\mathbf{A^1}$	$\mathbf{p^1}$	$\mathbf{Var_x^2}$	$\mathbf{Var_y^2}$	θ^2	$\mathbf{A^2}$	$\mathbf{p^2}$
Labels	13.4	10.8	0.14	0.020	0.006	13.8	10.4	0.14	0.017	0.013
Network estimation	31.4	24.0	0.55	0.051	0.015	40.6	21.1	0.45	0.042	0.056

Table 3.15: Comparison of the RMS when using the correct values of the positions or the the estimations of the network as the center for the cropping operation. The apex 1 was used to refer to the gaussian with the greater x

The performance of the 3 approaches tested was similar and the resulting loss functions compatible but still insufficient. A final generation paradigm (*Swapping dataset* in the following) was tested by building upon the single-bit dataset. Starting from the same ordering scheme as above a second transformation was applied to the set of labels, consisting in ordering all the remaining variables in descending order and following each variable with a new output variable, its *swapping index*, used to re-order the variables in output from the network and re-associating each variable to the corresponding gaussian. Noticing an improvement but still deeming the accuracy reached as unsatisfactory, the network was trained to recognize only one of the coordinates for each shape in the search of any asymmetries or clues to improve its performance. The RMS's for the single-bit dataset, the swapping dataset and the tests with single variables in output are shown in Tab.3.14

The training of the shape-determining network showed even worse results, being unable to converge altogether.

Close-up network

To tackle this problem and improve the accuracy of the resulting position a new approach was taken. The rough estimation of the position given by the first network was used as the center of a rectangle of sides equal to value of a parameter, the *crop-size*, times the respective dimensions of the image. The second network was then trained on these cropped images, each containing a single gaussian. At first, as a proof of principle and as a benchmark for comparing errors, the correct values of the positions taken from the labels were used to crop the images, then the output of the first network was used to test the viability of this approach. As shown in Tab. 3.15 the error is about three times larger in the second case. However, as it could be expected, by using the cropped images the RMS's are comparable to the best case with a single gaussian discussed in 3.5.

Using the correct values of the position, the cropped image will always have a single gaussian

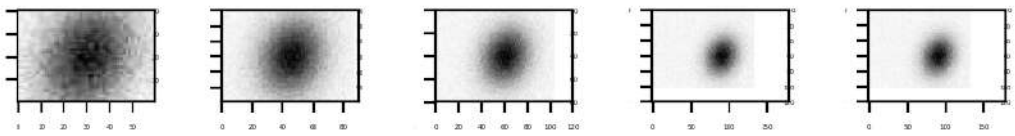

Figure 3.13: Examples of cropped images with different values of the crop-size parameter. From left to right the values used are: 0.1,0.15,0.2,0.25,0.3

Crop Size	**0.1**	**0.15**	**0.2**	**0.25**	**0.3**
$\mathbf{x_1}$	3.01	1.99	1.91	1.99	2.19
$\mathbf{y_1}$	2.45	1.98	1.80	44.99[a]	25.67
$\mathbf{var_{x_1}}$	39.7	38.1	40.9	34.9	33.02
$\mathbf{var_{y_1}}$	23.5	23.1	27.9	27.4	27.48
θ_1	0.59	0.47	0.58	0.58	0.53
$\mathbf{A_1}$	0.053	0.045	0.043	0.049	0.051
$\mathbf{p_1}$	0.016	0.0075	0.0046	0.0065	0.0087
$\mathbf{x_2}$	3.55	2.05	3.29	13.33	9.58
$\mathbf{y_2}$	6.48	2.92	4.19	45.47	27.48
$\mathbf{var_{x_2}}$	32.6	23.1	30.15	23.06	22.02
$\mathbf{var_{y_2}}$	18.5	12.7	19.38	20.92	21.28
θ_2	0.32	0.28	0.39	0.47	0.47
$\mathbf{A_2}$	0.04	0.028	0.026	0.061	0.066
$\mathbf{p_2}$	0.028	0.021	0.017	0.020	0.050

[a]A typical example of a batch where for one of the images the guess of the network is completely off, leading to a very high RMS even if the errors on the rest of the batch are comparable to the ones obtained with different crop sizes

Table 3.16: Comparison of the RMS when using different values for the crop-size parameter

in the center of the image. Using the network output, instead, results in the image of a gaussian that is shifted from the center by the same amount as the error committed by the network in the determination of the position. The second network was then trained to also output this offset, using the difference between the true value of the position and the estimation of the first network as the label for these new variables. By adding these offsets to the initial estimation, an adjusted value for the position was obtained. In Tab. 3.16 the RMS for different values of the crop-size parameter are shown, while in Fig. 3.13 an example of cropped images is given. It can be seen that the performance for values of the *crop size* parameter greater than 0.15 are comparable. Combining the obtained results with the notion that using a bigger value for the parameter would correspond to a larger cropped area and thus to a higher probability of finding a second gaussian in the image, a crop-size equal to 0.15 was used in the following. Using the 2 networks in such a manner, errors comparable to the case with a single gaussian were obtained, except for the angle, in some cases still showing a noticeable error. In Fig. 3.14 a visual representation of the results on a test batch is given. Noticing how the angle in output from the network would sometimes lead to a shape perpendicular to the gaussian, the generation parameters for the angle were rolled back, removing the restriction $\sigma_x > \sigma_y$ and using an angle in $[0; \frac{\pi}{2}]$.

3.7 Performance Evaluation

The performance of the final iteration of the network was estimated using 5 batches of 60 elements each. In Tab. 3.17 the RMS for the different variables are shown, taken as the

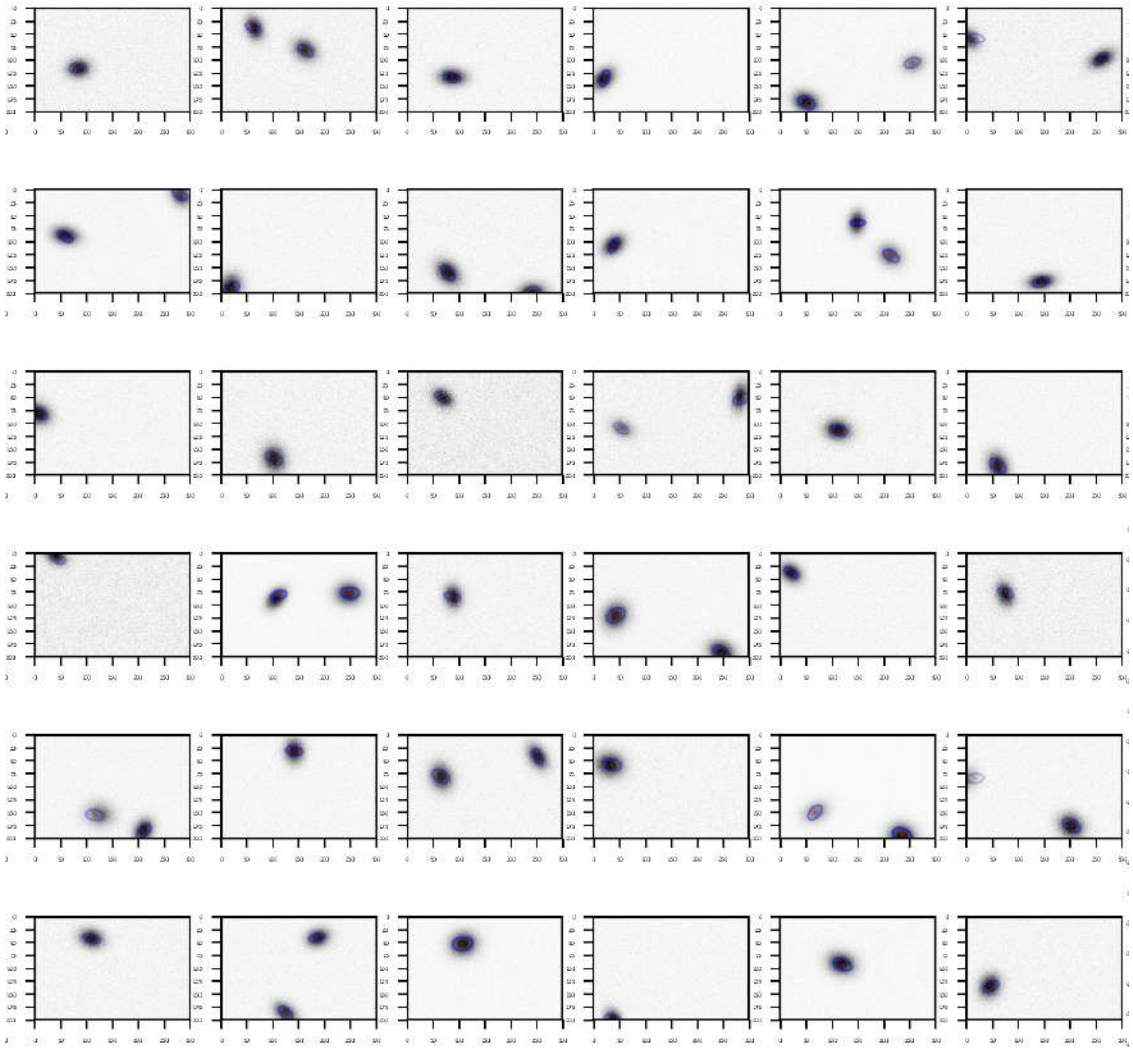

Figure 3.14: Output of the network on a test batch

Variables	$\mathbf{x_1}$	$\mathbf{y_1}$	$\mathbf{Var_x^1}$	$\mathbf{Var_y^1}$	θ^1	$\mathbf{A^1}$	$\mathbf{p^1}$
RMS	3 ± 1	20 ± 15	64 ± 4	54 ± 5	0.44 ± 0.02	0.06 ± 0.02	0.018 ± 0.009
Variables	$\mathbf{x_2}$	$\mathbf{y_2}$	$\mathbf{Var_x^2}$	$\mathbf{Var_y^2}$	θ^2	$\mathbf{A^2}$	$\mathbf{p^2}$
RMS	8 ± 5	21 ± 14	48 ± 6	43 ± 8	0.32 ± 0.04	0.05 ± 0.01	0.05 ± 0.03

Table 3.17: Evaluation of the network RMS. The apex 1 was used to refer to the gaussian with the greater x

Max error	**x**	**y**	$\mathbf{Var_x}$	$\mathbf{Var_y}$	θ	**A**	**p**
0.01	0.81 ± 0.06	0.82 ± 0.03	0.52 ± 0.03	0.48 ± 0.06	0.48 ± 0.06	0.6 ± 0.2	0.83 ± 0.04
0.05	0.99 ± 0.01	0.97 ± 0.01	0.62 ± 0.07	0.66 ± 0.04	0.59 ± 0.04	0.92 ± 0.07	0.997 ± 0.005
0.1	0.995 ± 0.006	0.993 ± 0.006	0.72 ± 0.01	0.79 ± 0.03	0.68 ± 0.06	0.99 ± 0.01	1
0.15	0.998 ± 0.005	0.989 ± 0.007	0.83 ± 0.02	0.88 ± 0.04	0.82 ± 0.05	0.996 ± 0.006	1
0.2	1	0.995 ± 0.006	0.90 ± 0.02	0.96 ± 0.02	0.92 ± 0.03	1	1
0.25	1	0.995 ± 0.006	0.96 ± 0.02	0.97 ± 0.02	0.998 ± 0.005	1	0.998 ± 0.005

Table 3.18: The percentage of times the estimation of the network had an error smaller than *max error*%. Data taken from 5 batches of 60 images each.

mean over the results obtained with each batch.

The high variance, in particular regarding the position, is due to a behavior typical of neural networks where the error is not distributed normally. In practice, apart from a typical normal distribution, sometimes the network will "misfire" and present a very high error, leading to a higher RMS on the batch even if all the other gaussians are recognized with reasonable precision. To account for this effect the RMS was accompanied by the data shown in Tab. 3.18 and Fig. 3.15, showing the percentage of gaussians where the error committed for each variable independently is smaller the the parameter *max error* on the x axis (as a percentage of the maximum value the respective variable can take). The results on the determination of the position is very promising, with over 50% of the values being determined with an error smaller than 1% and over 85% with an error smaller than 5%. A greater error is associated to the determination of the variances and the angle but still showing a maximum error of 20% in over the 90% of the test cases. Lastly, in Fig. 3.16, an example on the results obtained by the network on a batch size is given.

In Fig.3.17 the errors on a test batch are shown by plotting the values estimated by the network against the labels. It can be seen how the estimated position is very close to the real values and the p's, when rounded to the nearest value, match the labels perfectly. A trend present for all the variables consists in a greater error when the labels are closer to the edge of the generation interval. While this can be easily understood in the case of the position, where a value near the edge of the picture would equal a gaussian that is only partially visible, it is supposed that in the case of the variances this is due to the lower number of examples with "extreme" values seen during training. It appears the network tends to "crush" the estimation towards the center of the generation interval, leading to an overestimation of the variances closer to the upper edge and an underestimation of variances closer to lower one. While it is supposed that the performance on the border values could be improved by sampling from a different probability distribution, leading to a greater number of values at the edge of the interval, this could also degrade the performance in the rest of the cases. It was decided to keep the generation paradigm as it

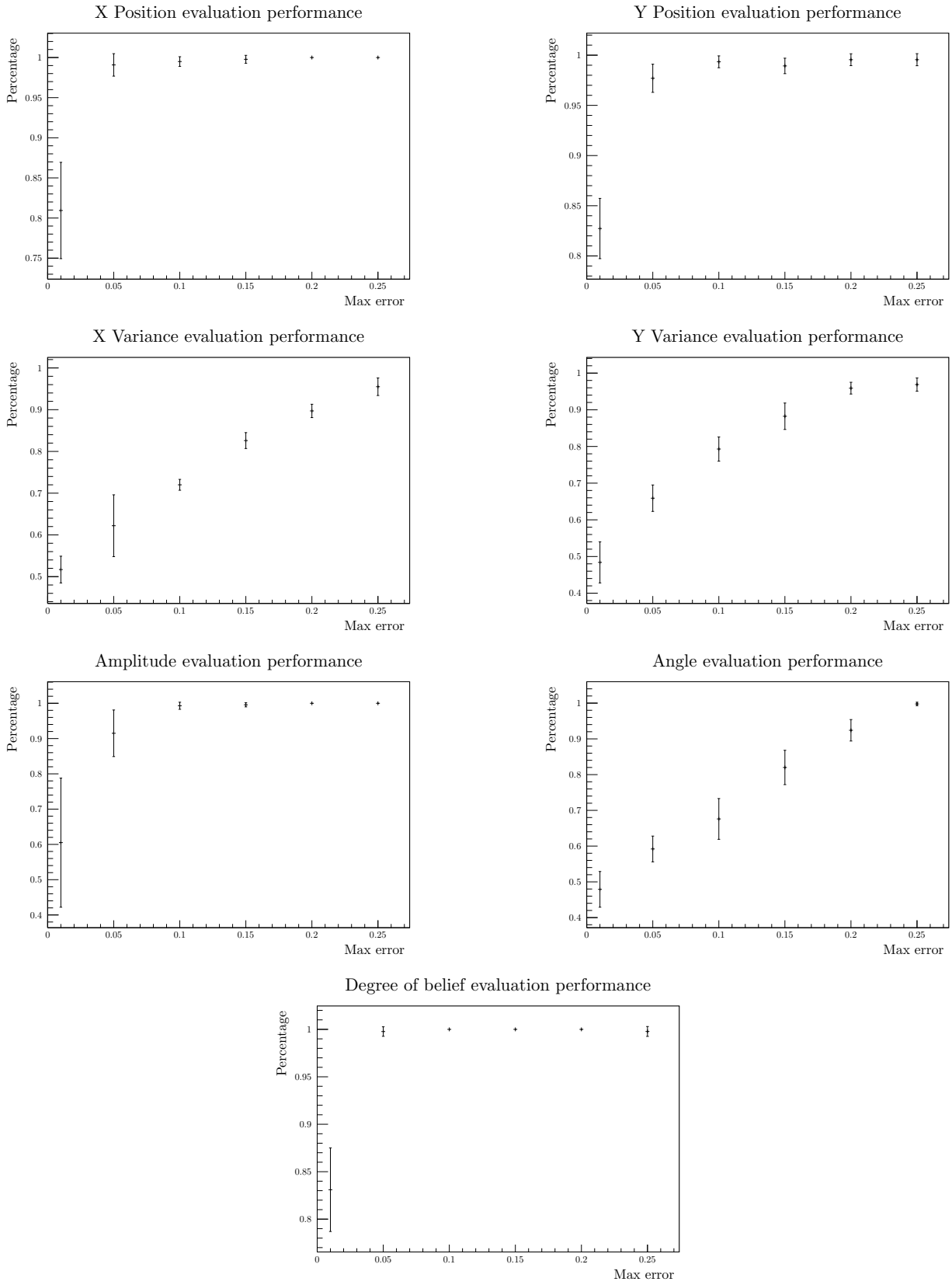

Figure 3.15: The percentage of times the network estimation had an error smaller than *max error*%. Data taken from 5 batches of 60 images each.

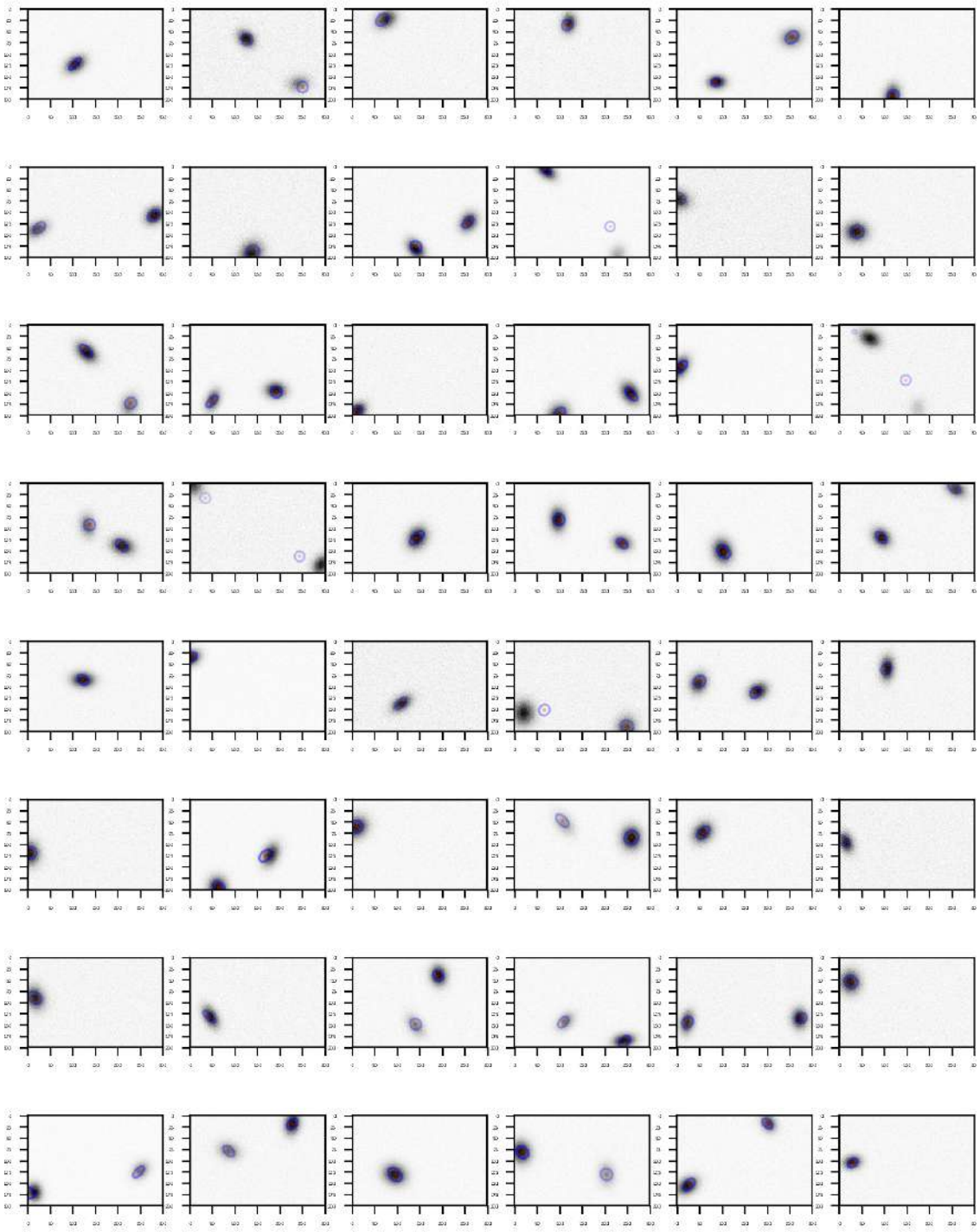

Figure 3.16: Output of the network on a test batch

is, given the regularity of the problem the system was developed for. In fact, in its practical application to STRIKE it is deemed unlikely the system will encounter gaussians with a great difference in the variances, corresponding to very "flattened" ellipses.

3.8 Full image scaling

The ability to distinguish between pictures with a number of gaussians ranging from zero to two was a fundamental milestone in the development of the network, ensuring the system could perform reasonably in the case of two gaussians too close to each other to be separated by simply cropping the image. Scaling the network directly to work with the tens of gaussians expected in a single thermographic image soon proved unfeasible and was seen as undesirable due to the rigidness such a system would imply, for example requiring always images with the same size and a maximum number of gaussians. To overcome these limitations and allow the application of the network to the full scale images coming from the camera, a kind of "filter" system was implemented. In a manner analogous to the way the kernel of the convolutional layers is applied to the image via convolution, the network was applied only to a rectangle of 200x300 pixels of the image at a time. By moving the "active" rectangle on the image and putting together the various information gathered by the single applications of the network, a comprehensive study of the image can be obtained. In particular the input image was divided in partially over-lapping rectangles of dimensions equals to the ones of the images used for training and shifted by a number of pixels equal to the parameters $strideX$ and $strideY$ in the x and y direction respectively. The output of each application was then taken in consideration in the final estimation only if the degree of belief in the existence of the respective gaussian was greater than a parameter called p_{limit}. The position estimated in the single application of the network was in the 200x300 interval, as taught during the training procedure, so the actual positions with respect to the full image was reconstructed by adding the position of the rectangle to the estimation of the network. The values of the stride parameters were chosen to be smaller than the rectangle dimensions. This decision was taken to decrease the probability of having a single wrong estimation of the gaussians parameters due to the presence of more than two gaussians in the active rectangle. In this case, in fact, the output of the network regarding the position was completely wrong, taking a sort of mean of the various gaussians present but with a corresponding low value of p. This choice of strides also resulted in usually more than one estimation of the parameters of a single gaussian, taken from different views. To account for this effect, the presence of multiple gaussians with centers distant less than a parameter called d_{limit} was assumed due to multiple estimation of a single gaussian and as such the mean of the parameters obtained in the different estimations was taken. In the starting tests of this configuration it was clear how the range of configurations the network had to handle was greater than the one met during training, leading to poor results. This was caused by the choice of generation parameters used for the gaussians, proven to be too restrictive. In the training dataset the centers of the gaussians were generated uniformly in the interval given by the image dimensions, leading to either images with only background or images with at least half of the gaussians visible and leaving out the possibility of only the tails of some gaussians showing, a case often encountered in the full size application. The generation algorithm was expanded to include these possibilities, in particular the interval used for the generation of the centers was expanded by 100 pixels in each direction but the respective labels were all set to 0 if one of the coordinates ended up out of the bounds

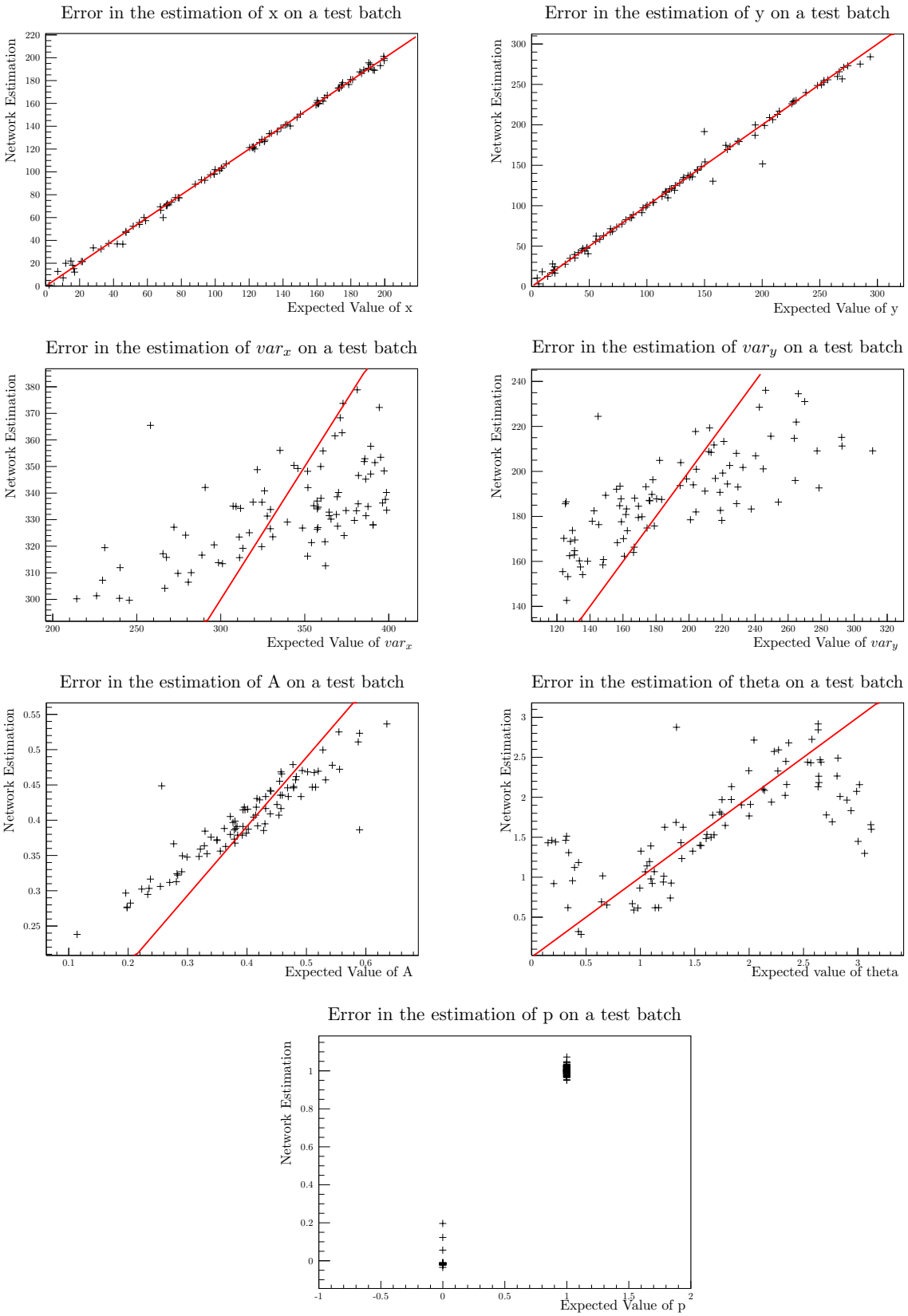

Figure 3.17: The errors on the network estimation on a test batch seen as the distance from the bisector of the quadrant shown in red.

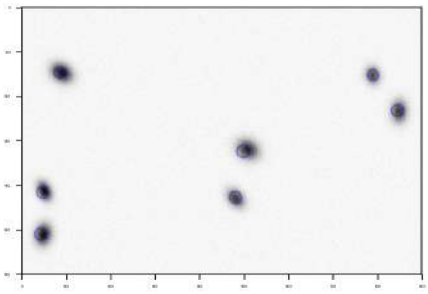

Figure 3.18: Result of the network on a full size image

of the image. This decision was based on the expectation that if the center is outside the active field of vision of the network during that application, a following one will have a better view and will be able to determine the gaussians parameters more accurately. Given that the typical linear dimension of a gaussian are smaller than 100 pixels, sometimes the gaussian would not be visible in the image. To prevent this from happening too often and to ensure a reasonable number of images used for training showing two gaussians, a 30% probability of having an image with two gaussians generated inside the bounds was added while the uniform generation in the expanded interval was used in the remaining 70% of the cases. In Fig.3.19 the single applications of the network to the crops of a 600x900 image with $strideX$ and $strideY$ equal to a forth of the dimensions of the training image, p_{limit} = 0.95 and d_{limit} = 17 pixels is shown, while the full image output is given in Fig.3.18.

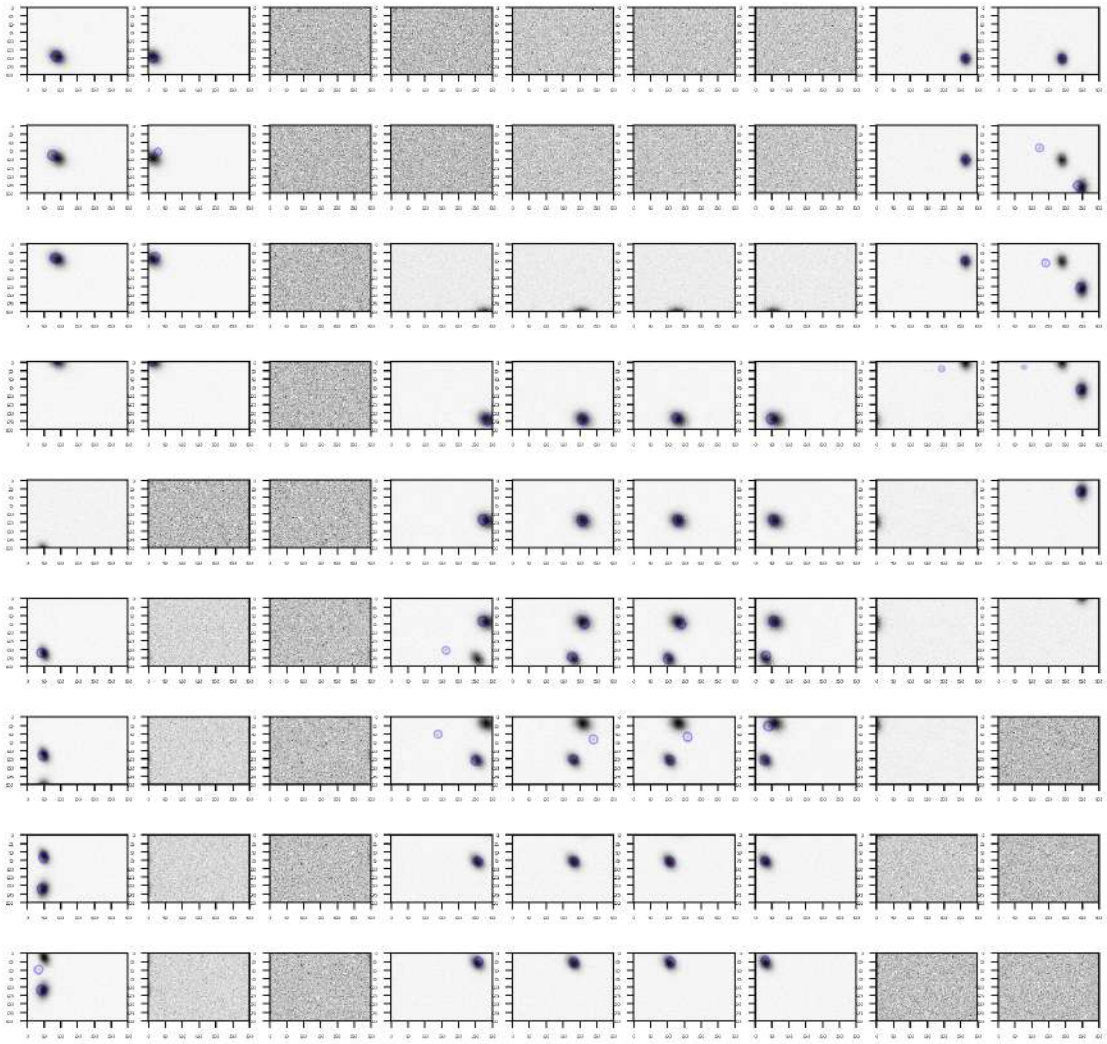

Figure 3.19: Application of the network to the cropped image

Chapter 4

Conclusions

During the thesis work a convolutional neural network was developed and optimized to retrieve the parameters of the beamlets striking the diagnostic calorimeter in the SPIDER experiment. The performance reached is deemed reasonable, even considering the increased amount of "misfires" in the images with two gaussians, given that the increment in the error in images with a single gaussian when training the network on a dataset with a mixed number of gaussians is very small. In practice, as mentioned in the description of the experiment (Chapter 1.2), there are 80 beamlets striking each tile of the calorimeter at once. However the complete image can be divided in many sub-images to be analyzed independently. The possibility of using the network as a kind of "filter" remains available and the proof of principle shown in the last paragraph demonstrates that this could be achieved. This is a particularly approachable solution in this case, due to the very regular arrangement of the beamlets, while still maintaining the flexibility of being able to recognize at least 2 gaussians if the two happened to be too close to be easily divided in different images with a grid search. It is expected that the neural network developed will be a useful tool in the study of the characteristics of the particle beams generated by SPIDER.

Appendix

Appendix A: Convolution

Convolution is a mathematical operation defined between two continuous functions as [1]

$$(f * g)(x) = \int f(y)g(x-y)dy$$

usually f is referred to as the *input*, g as the *kernel* and $(f * g)$ as the *feature map*. For numerical calculations this can be discretized as

$$(f * g)(x) = \sum_{y=-\infty}^{\infty} f(y)g(x-y)$$

and by assuming the functions have non-zero values only for the discrete set of points whose values are stored in the PC's memory, it can be reduced to a sum with a finite number of terms. This can be easily generalized to the multi-dimensional case in the form

$$(f * g)(x,q) = \sum_{y=-\infty}^{\infty} \sum_{z=-\infty}^{\infty} f(y,z)g(x-y,q-z)$$

In CNN the Kernel is usually much smaller than the input, leading to a matrix where most of the entries are zero and whose multiplication by the input is equivalent to applying a small matrix to only a sub-set of neurons. Useful properties of the convolution operation include:

- Commutativity

$$(f * g)(x) = \int f(y)g(x-y)dy = (g * f)(x) = \int g(y)f(x-y)dy$$

- $\mathcal{F}(f * g) = \mathcal{F}(f)\mathcal{F}(g)$
- $\mathcal{F}^{-1}(f * g) = \mathcal{F}^{-1}(f)\mathcal{F}^{-1}(g)$

Where $\mathcal{F}$ and $\mathcal{F}^{-1}$ denote the fourier transform and the anti-transform respectively.

[1] There are different conventions for the normalization of the resulting function, in some instances a $\frac{1}{\sqrt{2\pi}}$ factor is included

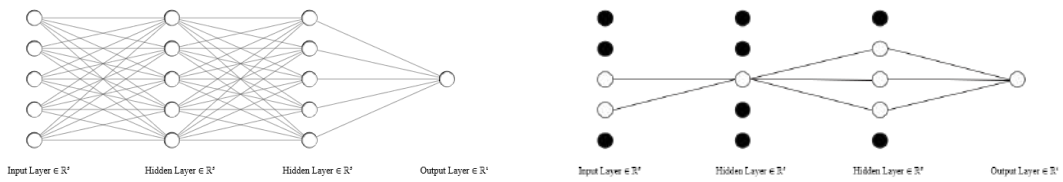

Figure 1: State of the network without(left) and with (right) drop-out. Black circles indicate shut down neurons during one of the training operations

Appendix B: Drop-out

A regularization method consisting in setting a percentage (*drop-out rate*) of "shutting down" each neuron inividually during each iteration of the training procedure, thus training an ensamble of some of the possibile sub-nets of the original network. The idea, based on biology, is based on increasing the effectiveness of the network by increasing the robustness of the single neurons, that must be able to work even if some of the other neurons to which they are connected are shut down. In linear networks, such as the one used in this instance, a neuron can be effectively subtracted from the network by multiplying its ouput value by 0, thus drop-out can be implemented by applying a randomly generated mask of 0's and 1's to the output of each layer, where 0 is picked with a probability equal to the drop-out rate and 1 with a probability equal to its complementary. Furthermore drop-out is computationally cheap, requiring only *O(n)* computation per example per update. At test time the entire network is used, using a scaled-down value of the weigths to compensate, giving an output that works as a mean of the results of the single sub-nets. A schematic representation is given in Fig.1 and additional information can be found on [9] and [3].

Bibliography

[1] Leon Bottou. *Stochastic Gradient Descent Tricks, present in "Neural Networks, Tricks of the Trade, Reloaded"*, volume 7700 of *Lecture Notes in Computer Science (LNCS)*. Springer, January 2012.

[2] Rita S. Delogu, Augusto Montisci, Antonio Pimazzoni, Gianluigi Serianni, and Giuliana Sias. Neural network based prediction of heat flux profiles on strike. *Fusion Engineering and Design*, 2019. `https://doi.org/10.1016/j.fusengdes.2019.03.178`.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org(27/06/2019)`.

[4] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[5] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[6] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexand re G. R. Day, Clint Richardson, Charles K. Fisher, and David J. Schwab. A high-bias, low-variance introduction to Machine Learning for physicists. *arXiv e-prints*, page arXiv:1803.08823, Mar 2018.

[7] ITER Organization. Technical photos. `http://www.iter.org/(27/06/2019)`.

[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nat*, 323:533–536, oct 1986.

[9] 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.