Deep Learning for Plasma Tomography

Francisco Matos
francisco.a.matos@tecnico.ulisboa.pt
Instituto Superior Técnico, Lisboa, Portugal

June 2016

Abstract

Tomography is a frequently used diagnostic in fusion experiments, namely to determine the shape and position of the plasma. However, plasma tomography algorithms work with only a few projections, which limits their accuracy and increases their computational complexity. Neural networks have been pointed out as a suitable alternative to carry out plasma tomography, since a trained neural network is potentially faster than traditional algorithms and capable of achieving similar accuracy. This thesis proposes to carry out tomographic reconstructions by using a deep neural network, with a topology that is the inverse of the convolutional networks that have been the subject of much research in the past few years. Applying this approach on data from a tomography diagnostic at JET, we show that the network is able to reproduce the reconstructions with high accuracy, as measured by several different metrics.

1. Introduction

Computed Tomography [21] has many practical applications, from medical imaging [22] to plasma diagnostics [10]. In nuclear fusion devices [37], tomography can be used to determine the shape and position of the plasma inside the device [1]. This can be used either for control of the fusion experiment, or for post-experimental analysis of the plasma [7].

It has been suggested that neural networks [14] can achieve tomographic reconstruction with the same or even higher accuracy [2, 31] than traditional tomographic algorithms. Neural networks are a model for machine learning which has seen tremendous progress in the past few years, in particular with the advent of *deep learning* [5]. One of the main successes of deep learning is in achieving (and even surpassing) human-level accuracy on image classification [15].

The goal of this work was to investigate how Computed Tomography could be carried out using the latest developments in neural networks and deep learning, as opposed to traditional tomographic algorithms. In particular, we measured the accuracy that a deep neural network can achieve when reconstructing the cross section of a plasma contained in a tokamak. For this purpose, data from JET, which is currently the largest fusion experiment in Europe, was used.

Section 2 of this article introduces the main topics related to the field of deep learning. Section 3 explains the principles of tomography and its applications in fusion experiments. Section 4 details a preliminary experiment that served as a proof of concept before we moved on to working on actual plasma data. Section 5 details the network we ultimately decided to used, and the experiments we performed on data from JET. Finally, section 6 details our conclusions.

2. Deep Learning

The concept of machine learning has existed since the advent of modern computers and, in general, refers to creating computer models capable of recognizing, or *learning*, patterns in a given type of input data. Once a pattern has been learned, it can be used to make predictions regarding new inputs of the same type. Mathematically, recognizing a pattern essentially means finding a function that maps specific inputs to specific outputs as correctly as possible. Over the years, several models for machine learning have been proposed; we will focus on neural networks and, more specifically, deep neural networks.

2.1. Neural Networks

Neural networks [14] are inspired by the functioning of biological brains, where each individual neuron can be active or inactive and performs a simple task; it is the composition of many millions of neurons, and the links between them, that makes the brain capable of performing complex tasks.

Similarly, an artificial neural network has many neurons (or nodes). Each node receives an input, performs a calculation on it, and outputs a result, and has several incoming and outgoing links that connect it to other nodes.

In terms of topology, neural networks are a composition of layers of nodes. Typically, there is one input layer, one output layer, and at least one "hidden" layer in between. In the input layer, each node simply receives an element of input data which is to be processed by the network. In the hidden and output layers, each node receives the outputs from nodes in the previous layer.

2.2. Convolutional Networks

Convolutional Neural Networks (CNN's) are a particular type of deep networks whose architecture is inspired by the functioning of the animal visual cortex. They are typically used for image classification tasks, in which they have had considerable success[23]. CNN's are usually composed of alternating convolutional and pooling layers, with several fully connected layers at the end.

A convolutional layer in a CNN has several *feature maps*, each one specialized in detecting a given feature in the input. A feature map (also called a filter or kernel) is essentially a group of weights shared by several nodes; filters can be much smaller than the images entering the network. Each filter slides horizontally and vertically across all the input, mapping (or more accurately, *convolving*) certain regions of the input into a particular node of the following layer. As is the case with fully connected networks, a CNN must be tuned through a training process.

Another key idea behind the architecture of CNN's is that when an image is to be classified, the relative position of its features is irrelevant compared to what features are actually present. This sort of translational invariance is usually obtained through the use of a subsampling or *pooling* layer. This type of layer essentially performs a reduction in the size of its input which eliminates the information on the spatial location of a feature detected by a previous convolutional layer.

After several successive convolutional and max pooling layers, total input size should shrink to a certain minimum, while most high-level features will have been extracted. At that point, one or more fully connected layers are used to combine those features and output a result, which may correspond, for example, to the probability that some image belongs to a certain class.

3. Plasma Tomography

In general, tomography refers to obtaining (more accurately, reconstructing) a 2D cross section of an object without having to slice it open. Though commonly associated with the medical industry due to its application in non-invasive imaging of living bodies, tomography has a wide range of applications in other fields [8].

The reconstruction of the 2D cross section of an object requires the acquisition of several of its *projections*. The cross section of the object can be reconstructed by combining multiple projections at different angles.

Naturally, it is essential to have a way to check the accuracy of tomographic algorithms. This is done by testing the algorithms on objects whose cross sections are known, and whose projections can be calculated analytically. By feeding the algorithm with those projections, it is then possible to compare the reconstructed image to the actual, known cross section of the object.

3.1. Applications in fusion diagnostics

Tomography can also be used to determine the 2D cross section of the plasma contained inside a fusion device. In contrast with medical imaging, where a body is placed in front of a radiation source, the projections of the cross section of a plasma are measured as a function of the radiation emitted by the plasma itself. In general, emitted radiation intensity depends on plasma density and temperature; therefore, it can convey information regarding plasma position [27].

Plasma emissivity can be detected by a bolometer system [17, 18]. A bolometer is a radiation measuring instrument based on temperature. Essentially, it has an absorber that captures the total radiation (the emissivity line integral) along a path, and a thermometer that measures the resulting temperature change in the absorber [20];

Proposals have long been made to use neural networks as a tool for carrying out plasma tomography [9, 25, 2, 31, 7] – that is, feed a network with the projections and expect it to output the corresponding density profile. However, previous experiments were done before the advent of deep learning. The networks used were relatively small, as were the training sets. Furthermore, they were developed with C code running on CPUs. In principle, one can expect to achieve faster and better results with optimized frameworks which generate GPU-executable code, and using larger and deeper networks, and larger training sets.

3.2. Shape and density of a fusion plasma

Interestingly, the shape of a fusion plasma resembles the shape of an ellipse. Ellipses are geometrical objects whose projections can be calculated analytically. This means that it is possible to measure the accuracy of tomographic reconstruction of plasma profiles, if we assume its shape to be elliptical [21]. The plasma density (or, more precisely, pressure) is described by the Grad-Shafranov equation [37], a well-known differential equation that describes the magneto-hydro-dynamic (MHD) equilibrium of a plasma. The simplest known solution to the Grad-Shafranov equation is Solovev's solution which involves a polynomial of fourth degree [33].

Using such a polynomial, and considering that the plasma is indeed elliptical, one gets a plasma density like

$$\rho(r) = \begin{cases} c(1 - 2r^2 + r^4) & \text{if } r \leq 1 \\ 0 & \text{if } r > 1 \end{cases}$$

where c is an arbitrary constant, and r is a measure of distance to the center of the ellipse (x_0, y_0):

$$r = \sqrt{\frac{(x - x_0)^2}{a^2} + \frac{(y - y_0)^2}{b^2}}$$

For an ellipse with the density function $\rho(r)$, it can be shown that its y- and x-projections are given by:

$$P_y(x) = \begin{cases} \frac{16}{15} bc \left(1 - \frac{(x-x_0)^2}{a^2}\right)^{\frac{5}{2}} & \text{if } \frac{(x-x_0)^2}{a^2} \leq 1 \\ 0 & \text{if } \frac{(x-x_0)^2}{a^2} > 1 \end{cases}$$

and

$$P_x(y) = \begin{cases} \frac{16}{15}ac\left(1 - \frac{(y-y_0)^2}{b^2}\right)^{\frac{5}{2}} & \text{if } \frac{(y-y_0)^2}{b^2} \leq 1 \\ 0 & \text{if } \frac{(y-y_0)^2}{b^2} > 1 \end{cases} \quad (1)$$

Thus, before applying neural networks on actual data, a preliminary experiment was carried out which consisted in generating elliptical phantoms and training a neural network to recover them from their x- and y-projections. We then evaluated the accuracy of that network in recovering previously unseen phantoms from their projections. The idea was to use this as a proof of concept before using neural networks on actual plasma data.

4. A simple experiment

The experiment presented in this section attempted to use a neural network to reconstruct phantom plasma profiles from their projections. For simplicity, we supposed that there were only two cameras, which corresponded to the top and side views of the plasma – i.e., they were positioned at the projection angles $\theta = 0$ and $\theta = 90°$ – and that the projections were obtained through parallel rays.

For this experiment, and the remaining work carried out, we used Keras[1], which is built on top of Theano [6, 3]. Both are deep learning frameworks that can generate both CPU and GPU-executable (in this case, CUDA [29]) code, and are Python-based.

To be able to use the formulae derived in section 3.2, we assumed that the plasma had an elliptical shape and was contained in a square box with normalized dimensions – that is, the coordinates of both the x and y axes were in the range $[0, 1]$. Using vertical- and horizontal-wise projections allowed us to calculate them using $P_y(x)$ and $P_x(y)$ as defined in section 3.2, respectively. Each projection had 11 equally spaced projection lines, at intervals of 0.1. Therefore, the number of inputs to the network was 22, corresponding to the total number of projection values.

The output of the network was the 2D cross section of the plasma with a certain resolution, where each output node gave the density/emmissivity ρ of the plasma at a certain position (x, y) in the normalized square box. To decide on the image resolution, we took into account the experiments carried out at the ISTTOK tokamak [7], where 3 projections, of 8 values each, were used to generate images on a 15×15 grid – roughly, doubling the number of values in each dimension. Therefore, we opted for an output resolution of 21×21, which is approximately the same proportion, and also allows us to have an evenly spaced grid at intervals of 0.05 in both dimensions. This resolution is also similar to the image sizes found in well-known datasets for deep learning, such as the MNIST database [24], where images are 28×28 pixels. Thus, there will be a total of $21 \times 21 = 441$ network outputs.

Train and test datasets were generated with a Python script that produced samples based on randomizing the ellipse parameters (x_0, y_0, a, b, c) subject to two restrictions: the ellipse must fit within the normalized square box, and the value of c was in the range $[0, 1]$. With these parameters we calculated, for each ellipse/phantom, the 22 values of the projections that would serve as input to the network, and the 441 values of emissivity that the network should produce as output. These are calculated based on the equations in section 3.2. Since we had the actual density values for each example in the test set, we could measure the network accuracy in recovering the density values of those phantoms. By measuring this accuracy, we could get an estimate on the effectiveness of using deep networks for recovering actual plasma cross sections.

4.1. Network topology

The main inspiration for the network used in this experiment was the *autoencoder* [16], a type of deep network that is based on the idea that data dimensionality can be reduced (encoded) into a small set of features. In essence, an autoencoder is a deep network which outputs the same image that is given as input, even though the intermediate layers of the network have far less nodes than either its input or output layers. The first part of the network can be seen as an encoder (of the image data into a small set of features), and the second part can be seen as a decoder (of those features back into the original image).

In this experiment, we used only the second part – the decoder. We regarded the projections as the encoded data, and we feed these data as input to the network. At the output, we expect to have the cross section of the object (that is, the plasma density/emmissivity at each pixel) and we regarded this as the decoded data. In other words, we regard the projections as an encoded description that needed to be "decoded" into an image of the cross section.

We opted for a network topology where each layer had twice the number of nodes of the previous layer, except for the first layer, which had 22 nodes, and the last layer, which had as many nodes as required in the output (21×21). This progression in powers of 2 was similar to the ones that we found in existing literature on the autoencoder [16], and it rendered a network with six fully connected layers.

4.2. Training parameters

With regard to network training, we used batch sizes ranging from 1 (network updates on every training example) to 100. The number of training epochs ranged from as few as 100 to as many as 200000, where larger batches were used. We generated two datasets, each with 10000 samples, and where each sample includes the projections and the emissivity values on the output

[1] https://github.com/fchollet/keras

grid. One dataset was used exclusively in the training phase, and the other in the test phase. In the training phase, we used 80% of the data for training, and 20% for validation, to be able to detect overfitting.

For weight initialization, we used Keras's built-in Glorot-uniform distribution [13]. We used both sigmoid (more specifically, hyperbolic tangent) and ReLU activation functions, and defined the network error through a mean-square error loss function. Parameter updates were done through stochastic gradient descent (SGD). We also tested different values for the network learning rate, ranging from 0.01 to 10. Learning momentum and learning decay were not used.

Training time ranged from as little as 10 minutes, in cases where larger batches were used (and with fewer epochs), to as much as 9 hours, and was conducted on a laptop with an Intel Core i3 3217U processor at 1.8 GHz (as CPU) and an NVIDIA GeForce GT 710M with 96 cores at 775 MHz (as GPU).

4.3. Results

Experimenting with different combinations of the configurations described above, we achieved good results with a training batch size of 10, and by allowing the network to train for 10000 epochs. With these values, the total network training time was 9 hours and 10 minutes. We found that a smaller batch size (for example, 1) meant that the validation loss converged to a minimum *initially* faster, but often produced large fluctuations in later epochs. It also meant that we were not taking advantage of the GPU, because greater parallelization can be achieved with larger batches [4]. Specifically, for a batch size of 1, we saw that epochs ran faster on the CPU; however, with a batch size of 10, each epoch was already running significantly faster on the GPU – on average, 3 seconds, versus approximately 8 seconds on the CPU.

For larger batch sizes (e.g. 100) the performance gain on the GPU was even higher, but the loss function converged much more slowly. In cases where we trained the network for much longer (i.e., 200000 epochs), we did not see significantly better results; in fact, we observed the occurrence of overfitting, since, at a certain point, the validation loss began to increase.

For our best results (10000 epochs with a batch size of 10), we do not believe that overfitting occurred, as we saw no indication that the validation loss was increasing. It did fluctuate from epoch to epoch, but its overall trend was one of decrease, as shown in figure 1.

Similarly, small learning rates such as 0.01 slowed down the convergence of the loss function, while large rates above 10 actually caused the loss function to diverge as epochs progressed. For that reason, we found 1.0 to be an optimal value for the learning rate, which allowed for a reasonably fast convergence (in terms of epochs) without large fluctuations.

Curiously, we found that using a network with a traditional activation function – the hyperbolic tangent – gave us better results than the ReLU, probably because the output of the hyperbolic tangent is limited to 1.0, which is also the maximum possible value of emissivity in the phantoms. The fact that we did not note the occurrence of vanishing gradients can probably be explained by the very large number of training epochs. We also found that weight initialization, in the short term, influences the loss function, but this effect becomes negligible as the network trains for more epochs.

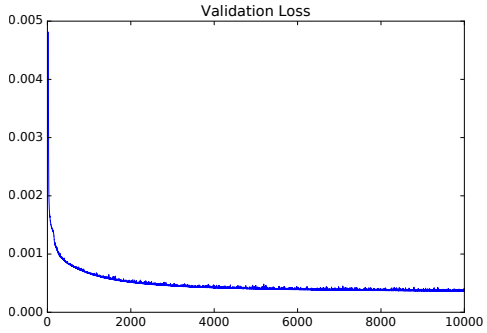

Figure 1: Validation loss per training epoch. At epoch 1, validation loss had a value of 0.00481; at epoch 10.000, its value was 0.00036.

After the network was trained, to obtain a measure of its accuracy we calculated the root mean square error (RSME) on the test set, which yielded a value of 1.87%. This means that, on average, the error of each pixel in the reconstruction is below 2%, which makes most reconstructions virtually indistinguishable from the original phantoms.

The good results we obtained in this experiment led us to believe that neural networks were, indeed, a good tool for tomographic reconstruction. Thus, we then decided to apply the same approach on data from JET.

5. Experiments on JET Data

The data from JET was collected over a period of several years – from September 2011 to December 2015 – and consisted of a total of 18357 samples, where each sample contains two projections and the image reconstructed from those projections using traditional tomographic methods.

Each data sample corresponded to an instant in time of a certain *pulse*, which is a plasma discharge in the JET tokamak. Pulses were identified by a number in the range $[80176, 89152]$, where larger values are more recent.

5.1. The KB5 diagnostic

KB5 is the bolometer system installed at JET, which measures radiation emitted by the plasma inside the tokamak [17, 18]. It is composed of two subsystems (cameras), KB5H and KB5V, whose lines of sight are shown in figure 2. KB5H and HB5V both have a total of 24 active channels, that is, they both measure radi-

ation along 24 lines of sight, though KB5V also has 8 reserve channels [26].

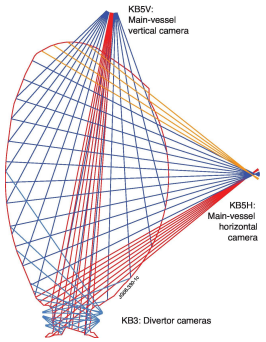

Figure 2: Bolometer system installed at JET (Source: EUROfusion Figures Database)

With this configuration, we initially expected our network to have an input array of 48 values, corresponding to the 24 views of each camera. However, the actual number of bolometer measurements per sample in the data turned out to be 52. The 4 extra values per sample can be explained based on the fact that there are 8 reserve channels of KB5V which are typically used for calibration purposes [26].

Another obvious difference between the data and the phantoms we used for our experiment was the size of the plasma profile reconstructions: we had generated images of 21×21 pixels, while the actual plasma profiles in the JET data had 115×196 pixels. It quickly became apparent that using a fully connected decoder would be unwise due to the size of the required network and the corresponding number of links.

5.2. Data preprocessing

Since the data was collected over several years, hardware and components of the JET tokamak may sometimes have been changed or upgraded. Thus, even ignoring anomalous situations, plasma measurements and image reconstructions could have different values and behavior as a result of changes in equipment.

For this reason, we decided to calculate several statistics – average, minimum, maximum, and median – over the samples in each pulse. Though by itself this was a somewhat coarse overview of the data, it nonetheless allowed us to check if any discernible patterns existed in samples from different times. In particular, we wished to verify if recent pulses differed significantly from the older ones. This was important to know before the training process because, if the data did indeed show great divergence over time, our network would attempt to fit very different samples, which would probably degrade its capacity for accurate reconstruction. If that were the case, it might make more sense to conduct separate training processes for different time frames.

To further simplify the analysis, we grouped pulses in their corresponding campaigns. A campaign is essentially a time frame along which pulses were carried out. This grouping is useful because equipment changes mostly occur in the time between two successive campaigns; therefore, large variation in data values, if it existed, was expectable mainly in transitions between campaigns.

In carrying out this analysis, we saw that our initial assumption – that data values potentially varied significantly over time – was correct. Regardless of this fact, some patterns were clearly distinct over time, in particular within groups of consecutive campaigns. Specifically, we noticed the existence of two campaign groupings – C29+C30b+C30c and C31+C32+C33 – containing data that showed overall similar trends.

Ultimately, we concluded that four main sets of data existed which might be useful to conduct separate training processes. Those sets were respectively composed by campaigns:

- C29 – 3238 projection/image samples;
- C30b + C30c – 2845 projection/image samples;
- C31 + C32 – 4162 projection/image samples;
- C33 – 4044 projection/image samples.

We grouped C30b + C30c and C31 + C32 because those campaigns were carried out at very close dates, and also so that the number of total data samples did not vary significantly from one set to another.

In each of these 4 sets, bolometer measurements and image pixel intensities have overall similar values, yet are still sufficiently dispersed so as not to be too similar – if they were, we would potentially be training a network too prone to overfitting, with no capacity for generalization. We judged the data in the other campaigns as being either too anomalous or as having too little variation and too few samples to be of use for training.

The projections we possessed each had, individually, a total of 52 dimensions/bolometer measurements. Even though we were now filtering data samples by campaigns, the individual network inputs could have noise, or their 52 dimensions might be correlated. In particular, if the input dimensions were correlated, the network might have had difficulty in extracting information from its inputs to generate their corresponding emissivity profiles. Thus, our intuition was that we could use some form of pre-processing step that allowed us to decorrelate the network input dimensions. This should ease the task of discovering the most relevant information to reconstruct the emissivity profiles, while using with the same computing power.

We decided to achieve this by using Principal Components Analysis (PCA), because of its common usage as preprocessing technique in deep learning tasks [34, 28]. Using PCA was also in line with the work done by

Ronchi et al. [31], who used the technique in experiments with neural networks for tomographic reconstruction. We performed PCA by using Python's scikit-learn library[2].

Apart from decorrelating the data, PCA also allowed us to see that, in the dataset composed by campaigns C30b + C30C, 6 of the 52 input data dimensions had no variance whatsoever, which probably means that the corresponding KB5 channels were not being used. This phenomenon also occurred in the other 3 campaign sets, though not on as many channels; in those cases, only 2 input dimensions had no variance. We therefore decided to remove these 2 dimensions from all the samples in the network input data, and use only the remaining 50, in all campaign sets.

5.3. The Up-convolutional Network

Section 2 described convolutional neural networks and their typical uses – image recognition and classification. The chapter also explained one of the main benefits of CNN's: that the number of total network links does not grow significantly even with larger inputs.

As we saw, our existing plasma profile reconstructions are 115×196 images. The ideal network to output results that large would be one where the number of parameters could be kept small, as in a CNN. Usually, a CNN receives as input an image, and outputs some encoding of that image, which in most cases is a probability of the image belonging to a certain class.

On the other hand, the data we would be processing (the projections measured by the JET bolometer system) could be considered an encoding of the plasma profile, as we have seen in section 4. Thus, our problem was the opposite of the one solved by a CNN: we required a network topology that did not increase in size regardless of the output dimensions, and with the ability to reconstruct an image given an encoding of that image.

For this reason, we used the opposite of a CNN – an *up-convolutional neural network (uCNN)* – where the typical progression of convolutional, pooling and fully connected layers is turned around, as done by Dosovitskiy et al [12]. The key in the functioning of that network is the "up-convolution" operation, which can be thought of as the opposite of the convolution + pooling operations typical in CNN's.

Intuitively, an up-convolution, having essentially the opposite effect of a convolution + pooling operation, would also be built on top of the inverses of those operations: that is, unpooling and deconvolution. In practice, deconvolution and convolution are symmetrical [38, 30] and therefore, an up-convolution can be done by an unpooling followed by a regular convolution [12, 32]. For this reason, we ultimately opted to carry out deconvolution by using the already-existing convolution operation implemented by the Keras framework.

[2] http://scikit-learn.org/stable/modules/classes.html

The same way a pooling operation will cut down on its input's size, an unpooling operation will have the opposite effect – given an input, it will output a result larger in width and height. Our work uses the unpooling operation currently implemented by the Keras framework (UpSampling2D, as named in the Keras API)[3]. This particular implementation of unpooling merely copies the input to all locations of the output region.

The topology of our up-convolutional network is thus a sequence of fully connected layers and up-convolutions. Inputs are a 1D array containing the JET bolometer signals of one data sample. Those inputs are fed to two FC layers with 7500 nodes each. This number was chosen because our experiments showed that larger FC layers yielded significantly better results than smaller ones – which leads us to believe that a major part of the network learning process occurs in these layers. We did not use even more FC layers because we were operating at almost the limit of available GPU memory. In any case, an experiment was done where we pushed the number of FC layers nodes to 9000 (roughly the limit at which no more memory was available), and this yielded essentially the same results.

The output of the second FC layer is reshaped from a sequence of 7500 values to a $(20 \times 15 \times 25)$ matrix, where each dimension, respectively, corresponds to depth, width, and height. That matrix can already be thought of as a rough "draft" of the final network output, but one where each of the 20 samples along the depth dimension is a particular representation (channel) emphasizing certain features of the final image. It will then be up to the up-convolutions to focus, zoom-in, smoothen and combine those channels to generate the final output, which will be a 2D plasma profile. All layers in the network use ReLU activation functions.

The reshaped data is fed to three sequential up-convolutions, each consisting of one (2×2) un-pooling operation and *two* 5×5 convolutions with 30 feature maps (the double convolution is based on the work of Dosovitskiy et al. [12], which concluded that this generated better results than a single convolution). Similarly to what we observed in the FC layers, increasing the number of feature maps in the up-convolutions, for example, to 50, did not yield better results. Both convolutions at each level have a stride of 1 (i.e., they slide trough all their inputs). One final, single convolution with only 1 feature map combines the features coming from the lower-level layers, and gives the network output. All convolutions are padded to preserve input size. The unpooling/upsampling operations are responsible for the actual zooming in on the FC layer output; the convolutional layers can be thought of as performing a search for a particular feature in the unpooling operation output, and performing a smoothing of the image.

With this progression of up-convolutions, the final output shape of the network is 120×200. However,

[3] http://keras.io/layers/convolutional/

the existing tomographic reconstructions in the JET data had a size of 115×196. This meant that the network would not be able to train with the existing reconstructions because their shapes did not match with the network output. To solve this problem, we decided to pad all images used in network training with the additional columns and rows entirely filled with zeros, thus equalizing their shape with that of the output. We did not judge this as being problematic because the first and last rows and columns of the existing images were, themselves, already filled with zeros.

5.4. Training and results

We had, before training the network, decided to use 4 data sets corresponding to different JET campaign time frames, and carry out separate training processes for each of them. In all those cases, we used 90% of the available data for training/validation, and 10% for testing. In the training/validation phase, we further subdivided the data into 90% for the training set, and 10% for the validation set, which was used to monitor the training progress. Thus, we had a total of 4 training, validation and test sets, corresponding to different JET campaign time frames.

We decided to use simple stochastic gradient descent as our network loss optimizer. This decision was due to the fact that by using SGD, we could directly tune the network learning rate – as opposed to other more advanced optimizers, such as Adagrad or Adadelta, that automatically change it. Tuning the learning rate turned out to be an important factor. Specifically, with the scaling of the data we had carried out before training, we obtained good results with a small learning rate – 0.001. Larger learning rates meant that overfitting quickly became a problem, with large validation loss fluctuations from epoch to epoch. Large fluctuations made it difficult to judge whether the training process was progressing well. They also meant that we could achieve a final result that was very good, but only because of a coincidence – that is, in the final epoch, some fluctuation might have occurred that rendered a very good result, but had no effective meaning in the long-term trend of the training process. Also for these reasons, learning rate momentum and decay were not used.

Weight initialization used the Glorot uniform distribution implemented by Keras. We used a batch size of 10, i.e., network parameters were updated based on the error value for batches of 10 training samples. 4 training processes were carried out, each one with one of the 4 data sets we had previously chosen.

The loss function we chose was the mean absolute error (MAE), also implemented by Keras. The choice of this loss function meant that the network would attempt to carry out a pixel-by-pixel optimization on the absolute error of its outputs. We chose it because of its common usage as an optimizer.

We ran the network on an NVIDIA GeForce GTX 480 GPU with 480 CUDA cores at 1401 MHz. Training took approximately 48 hours for each of the 4 sets of campaigns.

In each of the 4 training cases the overall trend of the validation loss was to decrease (along with the training loss) until around epoch 1500, when it started to behave asymptotically. Because of this, we have no reason to believe that overfitting occurred until that time, and therefore, the 4 training processes were carried out for 1500 epochs.

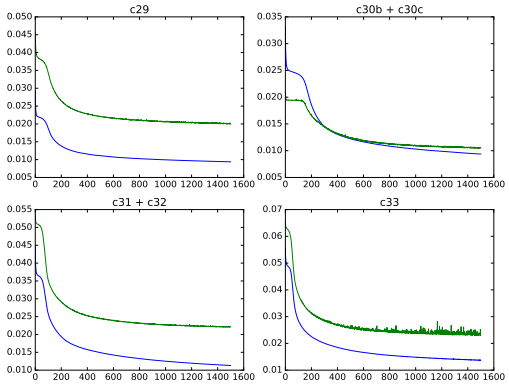

Figure 3: Loss (blue) and validation loss (green) progression in all training processes. Y-axis: MAE values. X-axis: training epoch.

An alternative to MAE would have been to train the network using Keras's mean squared error (MSE) loss function. However, we did not use it, because MSE greatly compounds large errors relative to small ones and consequently may make the training process more difficult, with greater loss fluctuations from epoch to epoch. Similarly, it generates loss and validation loss values that, for experimental purposes, are harder to interpret than MAE values.

In any case, both MAE and MSE are potentially poor predictors of signal fidelity [35]. Specifically, the MAE is computed as the average over the absolute errors of all pixels of a network output. This means that it is possible for an output as a whole to have a relatively good average loss value, when in practice, what is happening is that some of its pixels are perfectly adjusted to their true values, while others have potentially large deviations. This is also true for other loss functions such as mean square error (MSE).

Thus, we restricted the use of MAE to the training process – that is, we only calculated it on the training and validation sets, to monitor the training progress. Measurement of the actual, final image quality (post-training, on the test set of each data grouping) was done by using three other metrics. Those were the Normalized Root Mean Squared Error (NRMSE), Peak Signal-to-Noise Ratio (PSNR) [19] and Structural Similarity

(SSIM) [36]. Each metric compared the already existing reconstructions (in the JET data) with the ones generated by the network. In all cases, we performed a scaling of the images, squeezing all pixel values to the range $[1; 1]$.

The main advantage of using NRMSE over MAE or even MSE is that NRMSE is normalized over each input; therefore, we can compare the error of two output samples even if they have significantly different values (and indeed, the data preprocessing stage of our work had shown us that output image pixel values could have significant variation). We used PSNR because it is commonly used in image quality assessment. PSNR is logarithmic and is measured in decibels, which means that even a small increase in its value can indicate a significant increase in signal quality. PSNR has no maximum value *per se*; the intuition behind it is merely that the larger it is, the better.

Another reason for using those two metrics is that we did not want to restrict ourselves to the use of SSIM, whose inner working is significantly different. Unlike other error metrics, SSIM attempts to measure structural changes over the entire image, rather than performing a point-by-point error measurement. SSIM's output values are in the interval $[-1; 1]$, where an SSIM value of -1 means two images are completely different, and 1 means they are identical.

To calculate all these metrics on our output data, we used used python's scikit-image library [4].

	SSIM		
	worst	mean	best
C29	0.899	0.980	0.999
C30b + C30c	0.922	0.964	0.985
C31 + C32	0.953	0.990	0.998
C33	0.895	0.986	0.998

Table 1: Final values of SSIM for each training set.

	NRMSE		
	worst	mean	best
C29	0.083	0.032	0.009
C30b + C30c	0.075	0.043	0.023
C31 + C32	0.071	0.037	0.033
C33	0.096	0.029	0.021

Table 2: Final values of NRMSE for each training set.

Tables 1, 2 and 3 show, for each metric, the minimum, mean and maximum value over each test set. All image data was scaled to values in the interval $[-1; 1]$ before the metrics were calculated. The overall worst values for SSIM (0.895), NRMSE (0.096) and PSNR (27.208) all occurred in images generated from data in campaign C33. The best values for SSIM are all very close to 1, which indicates that most of the reconstructions carried out by the network were, according to that metric, very good reproductions of the originals. The PSNR values were similar to those expectable in image compression tasks, which usually generate a PSNR between 30 and 50. The NRMSE gives, overall, similarly good values.

	PSNR		
	worst	mean	best
C29	28.026	37.355	47.181
C30b + C30c	29.074	34.494	39.502
C31 + C32	29.451	35.099	36.093
C33	27.208	37.858	39.898

Table 3: Final values of PSNR for each training set.

Though this was not the main focus of our work, we also took a rough estimate of the time it took the trained network to reconstruct individual images. In all the 4 trained networks, this was around 10ms.

6. Conclusions

We used an up-convolutional neural network to generate tomographic reconstructions of plasma emissivity profiles based on plasma projection data. We divided our data into several sets corresponding to different timeframes of JET campaigns. In all of thoses cases, our overall results were good, with the overall worst values being 0.895 for SSIM, 0.096 for NRMSE and 27.208 for PSNR. Since all of these occurred in the same campaign (C33), one can conclude that there probably was some inherent aspect of the data from that time that rendered training more difficult. The results also showed that while the network training process itself may take some time, post-training image reconstruction is quick. This is in line with the results from previous experiments with neural networks for tomographic reconstruction.

We trained the network on projection/emissivity data samples, where the emissivity profiles were obtained through traditional tomographic methods. This means that the accuracy of our network is ultimately limited by the quality of the existing reconstructions. For this reason, in future work, one way to increase the accuracy of our results is to make sure the reconstructions obtained through traditional tomography are more accurate than the ones currently existing. Similarly, we believe that the existing reconstructions are generally very similar to each other. Better results (or at least, better capacity for network generalization) might be achieved by training with images with greater variability.

[4] http://scikit-image.org/docs/dev/api/skimage.measure.html

Our network was optimized using Mean Absolute Error because of its common usage in deep learning experiments. While this optimizer allowed us to achieve good results, it is not necessarily the best one, because it performs a pixel-by-pixel optimization. This means that outputs may simultaneously have pixels with good and poor values. The values of the three additional metrics we used lead us to believe that this phenomenon, if it exists, is somewhat restricted. Nevertheless, one possible improvement to our results would be to develop some network training optimizer based on, for instance, SSIM. Another (probably more complex) possibility would be an optimizer akin to the work done by Dosovitskiy and Brox in [11], which uses a loss function that computes image distances (i.e., error) using image features. One additional way to check the accuracy of our results would be to perform a cross-validation with some other diagnostic existing at JET.

We did not further increase network size due to lack of computing power. In any case, our experiments indicate that there are no significant benefits in increasing the network size.

All in all, our conclusion is that, using this hardware and network topology, the best improvements to be made would be to use better training sets or, perhaps more importantly, using alternative (more appropriate) loss functions than MAE. Nevertheless, we believe our results were satisfactory.

References

[1] M. Anton, H. Weisen, M. J. Dutch, W. von der Linden, F. Buhlmann, R. Chavan, B. Marletaz, P. Marmillod, and P. Paris. X-ray tomography on the TCV tokamak. *Plasma Physics and Controlled Fusion*, 38(11):1849, 1996.

[2] O. Barana, A. Murari, P. Franz, L. C. Ingesson, and G. Manduchi. Neural networks for real time determination of radiated power in JET. *Review of Scientific Instruments*, 73(5):2038–043, 2002.

[3] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley, and Y. Bengio. Theano: new features and speed improvements. *arXiv preprint arXiv:1211.5590*, 2012.

[4] Y. Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade*, pages 437–478. Springer, 2012.

[5] Y. Bengio, I. J. Goodfellow, and A. Courville. *Deep Learning*. MIT Press, 2015. (in preparation).

[6] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: A cpu and gpu math compiler in python. In *Proc. 9th Python in Science Conf*, pages 1–7, 2010.

[7] P. J. Carvalho. *Tomography Algorithms for Real-Time Control in ISTTOK*. PhD thesis, IST, April 2010.

[8] L. D. Chiffre, S. Carmignato, J.-P. Kruth, R. Schmitt, and A. Weckenmann. Industrial applications of computed tomography. {*CIRP*} *Annals - Manufacturing Technology*, 63(2):655–677, 2014.

[9] G. Demeter. Tomography using neural networks. *Review of scientific instruments*, 68(3):1438–1443, 1997.

[10] N. Denisova. Plasma diagnostics using computed tomography method. *IEEE Transactions on Plasma Science*, 37(4):502–512, April 2009.

[11] A. Dosovitskiy and T. Brox. Generating images with perceptual similarity metrics based on deep networks. *arXiv preprint arXiv:1602.02644*, 2016.

[12] A. Dosovitskiy, J. T. Springenberg, M. Tatarchenko, and T. Brox. Learning to generate chairs, tables and cars with convolutional networks. *arXiv:1411.5928*, 2015.

[13] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *JMLR Workshop and Conference Proceedings*, pages 249–256, 2010.

[14] S. Haykin. *Neural Networks and Learning Machines*. Prentice Hall, 3rd edition, 2008.

[15] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. *CoRR*, abs/1502.01852, 2015.

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

[17] A. Huber, K. McCormick, P. Andrew, P. Beaumont, S. Dalley, J. Fink, J. Fuchs, K. Fullard, W. Fundamenski, L. Ingesson, et al. Upgraded bolometer system on JET for improved radiation measurements. *Fusion Engineering and Design*, 82(5):1327–1334, 2007.

[18] A. Huber, K. McCormick, P. Andrew, M. de Baar, P. Beaumont, S. Dalley, J. Fink, J. Fuchs, K. Fullard, W. Fundamenski, et al. Improved radiation measurements on jet–first results from an upgraded bolometer system. *Journal of nuclear materials*, 363:365–370, 2007.

[19] Q. Huynh-Thu and M. Ghanbari. Scope of validity of PSNR in image/video quality assessment. *Electronics Letters*, 44(13):800–801, June 2008.

[20] L. Ingesson, B. Alper, B. Peterson, and J.-C. Vallet. Tomography diagnostics: Bolometry and soft-x-ray detection. *Fusion Science and Technology*, 53(2):528–576, 2008.

[21] A. C. Kak and M. Slaney. *Principles of Computerized Tomographic Imaging*. IEEE Press, 1988.

[22] W. A. Kalender. X-ray computed tomography. *Physics in Medicine and Biology*, 51(13):R29, 2006.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[24] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[25] X. F. Ma, M. Fukuhara, and T. Takeda. Neural network ct image reconstruction method for small amount of projection data. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 449(1):366–377, 2000.

[26] K. McCormick, A. Huber, C. Ingesson, F. Mast, J. Fink, W. Zeidner, A. Guigon, and S. Sanders. New bolometry cameras for the jet enhanced performance phase. *Fusion engineering and design*, 74(1):679–683, 2005.

[27] J. Mlynar, V. Weinzettl, G. Bonheure, A. Murari, J.-E. Contributors, et al. Inversion techniques in the soft-x-ray tomography of fusion plasmas: toward real-time applications. *Fusion Science and Technology*, 58(3):733–741, 2010.

[28] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng. Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 689–696, 2011.

[29] J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable parallel programming with CUDA. *ACM Queue*, 6(2):40–53, March/April 2008.

[30] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1520–1528, 2015.

[31] E. Ronchi, S. Conroy, E. A. Sundan, G. Ericsson, M. G. Johnson, C. Hellesen, H. Sjastrand, and M. Weiszflog. Neural networks based neutron emissivity tomography at JET with real-time capabilities. *Nuclear Instruments and Methods in Physics Research A*, 613(2):295–303, 2010.

[32] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015*, pages 234–241. Springer, 2015.

[33] S. Sesnic, D. Poljak, and E. Sliskovic. A review of some analytical solutions to the Grad-Shafranov equation. In *22nd International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 24–27, September 2014.

[34] Y. Sun, X. Wang, and X. Tang. Deep learning face representation from predicting 10,000 classes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1891–1898, 2014.

[35] Z. Wang and A. C. Bovik. Mean squared error: love it or leave it? a new look at signal fidelity measures. *Signal Processing Magazine, IEEE*, 26(1):98–117, 2009.

[36] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *Image Processing, IEEE Transactions on*, 13(4):600–612, 2004.

[37] J. Wesson. *Tokamaks*. Oxford University Press, 4th edition, 2011.

[38] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus. Deconvolutional networks. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2528–2535. IEEE, 2010.