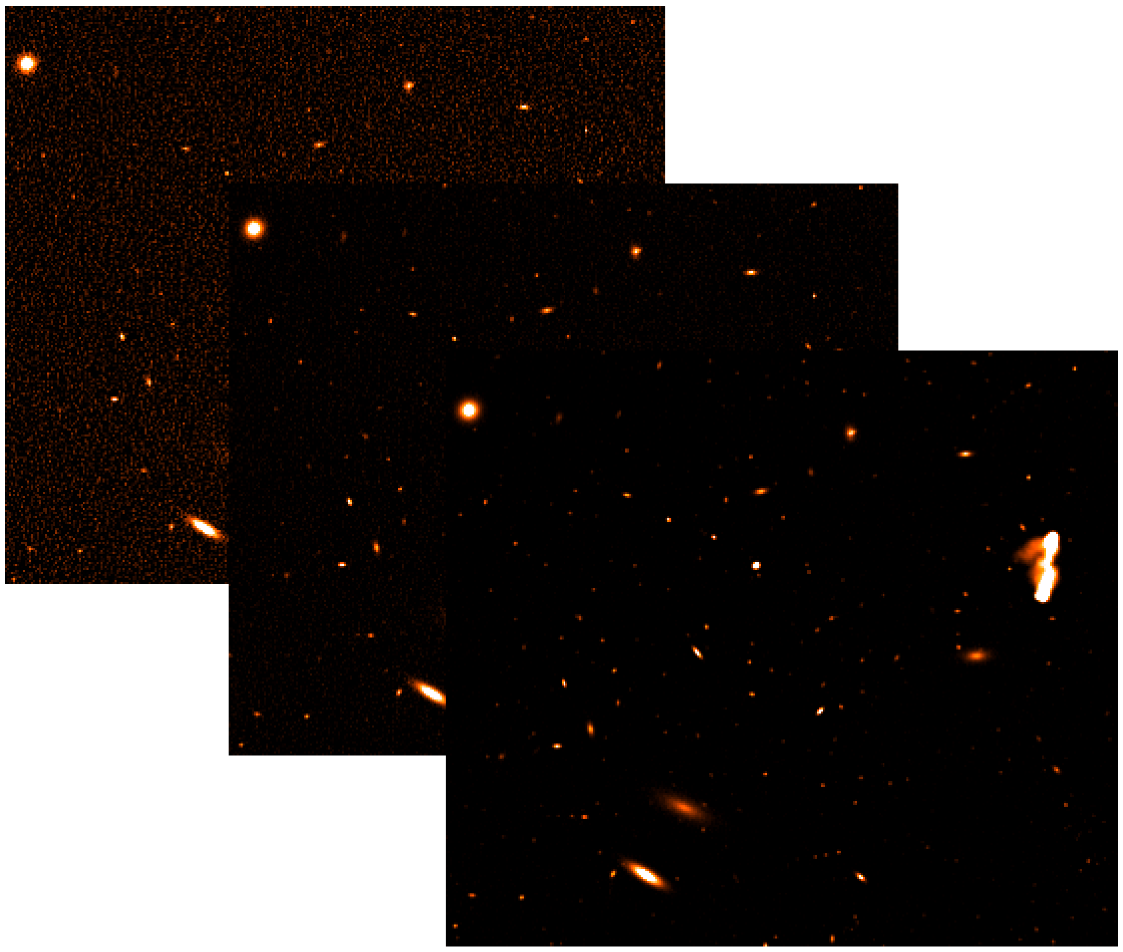
Square Kilometer Array Data Challenge

Astronomic source localization and classification



Carlo Longhi
carlo.longhi@studio.unibo.it

Antonio Damiano
antonio.damiano3@studio.unibo.it

Contents

1 - Introduction

The Square Kilometer Array (SKA) Data Challenge consists of processing high-resolution space images in order to identify astronomic sources and their properties.

The challenge is to undertake source finding, source property characterization and source population identification from simulated SKA data. For this purpose, we have designed two Convolutional Neural Networks.
The first model we designed is tasked with the simple localization of astronomic sources.
The second model, more complex than the first one has the additional task of classifying each source according to one of the three possible categories.

1.1 - Convolutional Neural Networks (CNN)

Convolutional Neural Networks are deep neural networks designed for processing structured arrays of data such as images, whose architecture is inspired by the organization of the human visual cortex [1]. CNNs are one the most common deep learning frameworks. Their main characteristic is the use of convolutional layers which are composed of a set of learnable filters. Filters are slid across the input to compute a convolution at every spatial position and pass the result to the following layer, ultimately achieving a hierarchical extraction of features. Because filters are typically smaller than the input and because of the use of shared weights, the number of parameters is significantly lower than in fully-connected layers. The reduction of parameters in CNNs helps to avoid the vanishing gradient problem, where the addition of layers can cause the gradient to decrease to zero, a problem most commonly encountered in fully connected neural networks. The filters also ensure parameter sharing, which enforces translational invariance, defined as when the network produces the same response if the input is translated horizontally or vertically.

2 - The Data

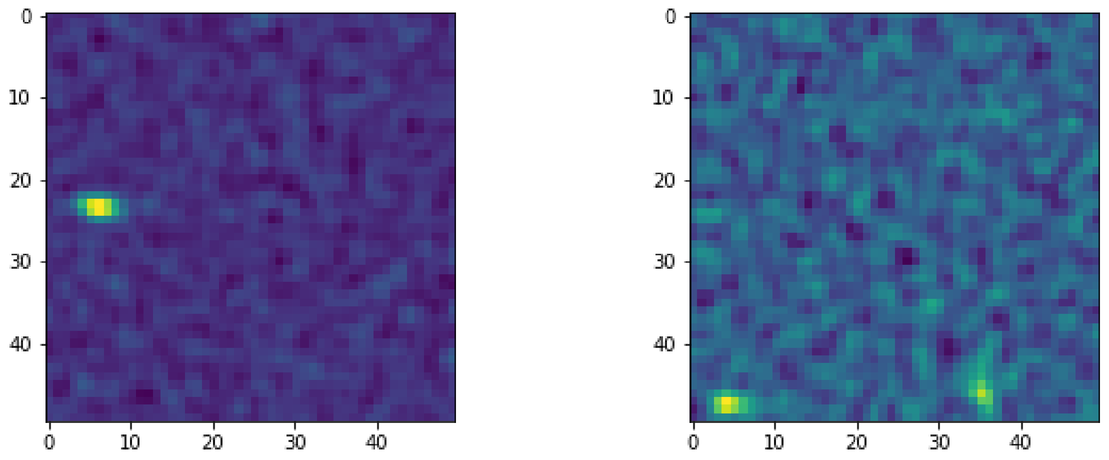
The dataset provided by SKA consists of 9 image files in FITS format of dimensions 32768x32768 pixels [2]. Each file is a simulated image in one of three frequencies: 560 Hz, 1400 Hz, 9200 Hz. For each frequency, three telescope integration depths are provided: 8, 100 and 1000 hours. The nine maps are a plausible realization of the radio sky at those frequencies.
A training set is also provided for each frequency band, with the data of the sources contained in an area of 4000x4000 pixels in the original image. For each source, the information provided in the training consists of:

- Source ID
- Right ascension of the source core
- Declination of the source core
- Right ascension of the source centroid
- Declination of the source centroid
- Integrated flux density
- Integrated flux density of core/total
- Major axis dimension
- Minor axis dimension
- PA
- Size
- Class
- Selection
- Pixel x coordinate of the centroid
- Pixel y coordinate of the centroid

2.1 - Data Preprocessing

The first step in data preprocessing has been to calculate the signal-to-noise ratio of the sources. For this purpose, we divided the image files in square cutouts of side length equal to 50 pixels. For each of these cutouts, the mean squared error was calculated. The next step was to discard those sources whose integrated flux density is lower than the error of the corresponding cutout [3].
In the tested training sets, the remaining sources were roughly a tenth of the original number.

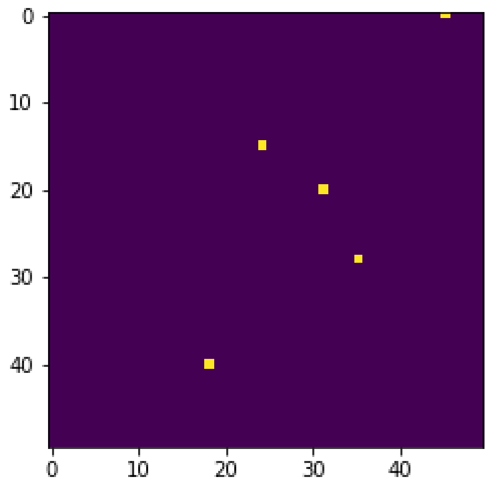
After noise filtering, the images underwent another preprocessing step: normalization.
To each cutout was subtracted its mean value and it was divided by its standard deviation. This step was necessary to counteract the effect of loss of significance due to the small values of the image. Normalizing the data also helps to speed up the training process and improves the accuracy of the model.



Two examples of image cutouts after the normalization

3 - Source localization

Our approach to the localization problem has been to use a convolutional neural network that, fed with image cutouts, outputs a predicted location map where each pixel's value represents the probability of that pixel to be the location of a source in the original image.
For this reason, from the source location data, we generated a map of locations where each pixel corresponds to a source and matched each cutout with its corresponding map.

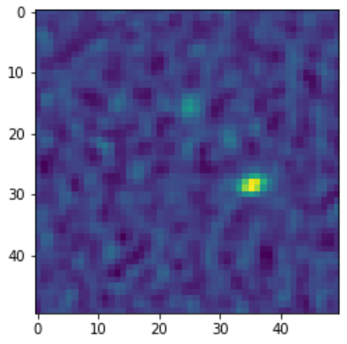


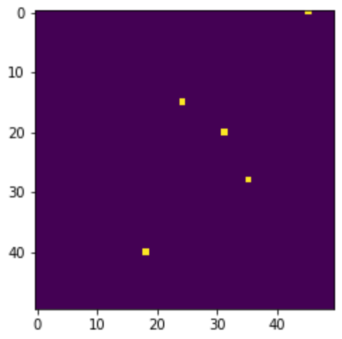
Example of a real location map with data from the training set

We found the optimal edge size of the cutouts to be 50 pixels. Using bigger cutouts produced significantly fewer images to feed to the network, which resulted in a drop in performance. On the other hand, smaller cutouts produced worse results caused by misclassification problems for sources located near the edges.

3.1 - The model

Our model is a simple Convolutional Neural Network consisting of three convolutional layers and one fully-connected layer. The purpose of the convolutional layers is to extract features from the images that are then integrated in the final layer to output a prediction for the location of the source.





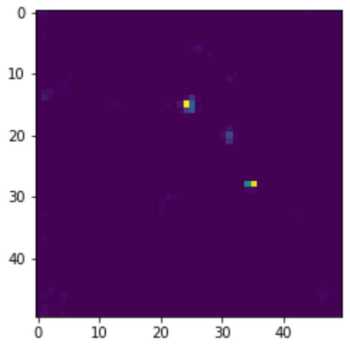
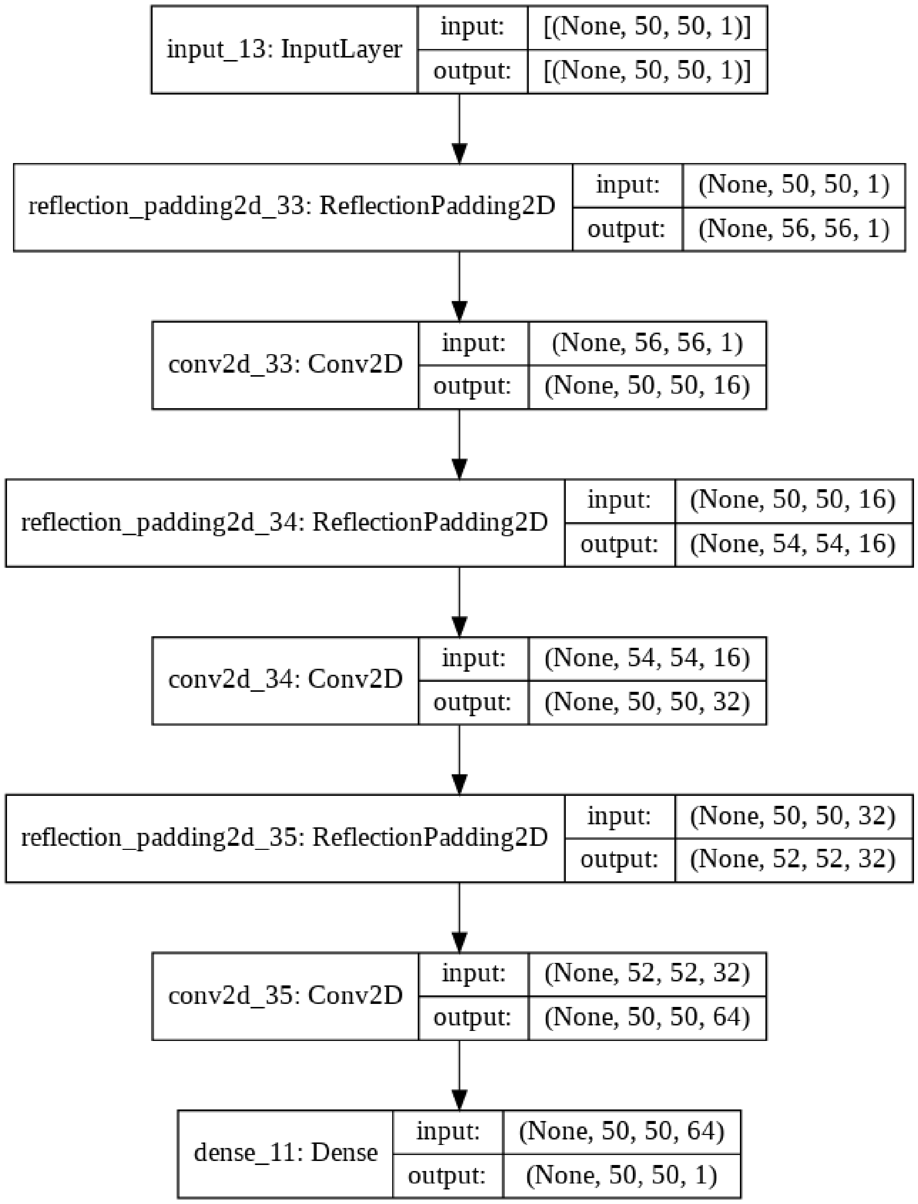


Figure 1: one of the image cutouts from the original fits file
Figure 2: the corresponding real source location map
Figure 3: the model's predicted source location map

We have experimented with an even lower number of filters, using only two convolutional layers and reducing the size of the filters but the best results were produces training the following model:



Our choice of a smaller model is determined by the type of features we are looking for in the images. The astronomic objects present in the images are small and do not present complex structures. For this reason, we can reduce the size of the convolutional filters and their number in each layer as we do not need to detect complex features. This strategy also helps in keeping the number of parameters low and helps to keep the training process faster.

3.2 - Padding

In a convolutional layer, the choice of the filters determines the size of the output [1].
Along each axis, the dimension of the input is determined by the formula:
$((W + P - K) / S) + 1$

Where W is the dimension of the input, P is the padding, K is the filter size and S is the stride.
To avoid information loss, we set the stride for each of our filters to 1. For this reason, the size of our output is only determined by the size of the filter. Since we want to output an image of the same size as the input image, we use padding before every convolutional layer.
Applying zero padding produces unwanted localization effects along the borders because our images are composed of positive and negative values. For this reason, we have implemented a custom padding layer that applies reflective padding before each convolution.

3.3 - Pooling

Pooling layers are used to downsample feature maps by summarizing the presence of features [1]. The most common types of pooling are max and average pooling.
We experimented with pooling layers between the convolutional layers. Because of the change in dimensions of the architecture of the layer due to pooling, an upsampling layer had to be inserted before the output.
In general, the application of pooling layers resulted in lower metrics compared to the original model, probably due to the loss in localization precision.

3.2 - Loss Function

The chosen loss function for our solution is the Binary Cross-Entropy. Binary Cross-Entropy is the default loss function for binary classification problems. The output of our model is the probability that a certain pixel of the input image corresponds to the position of an astronomic source.

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

3.3 - Optimizer

In our localization model, we have used the Adam optimizer with the following parameters:

- ➢ Learning rate = 0.0001
- ➢ Beta1 = 0.9
- ➢ Beta2 = 0.999
- ➢ Epsilon = 1e-7

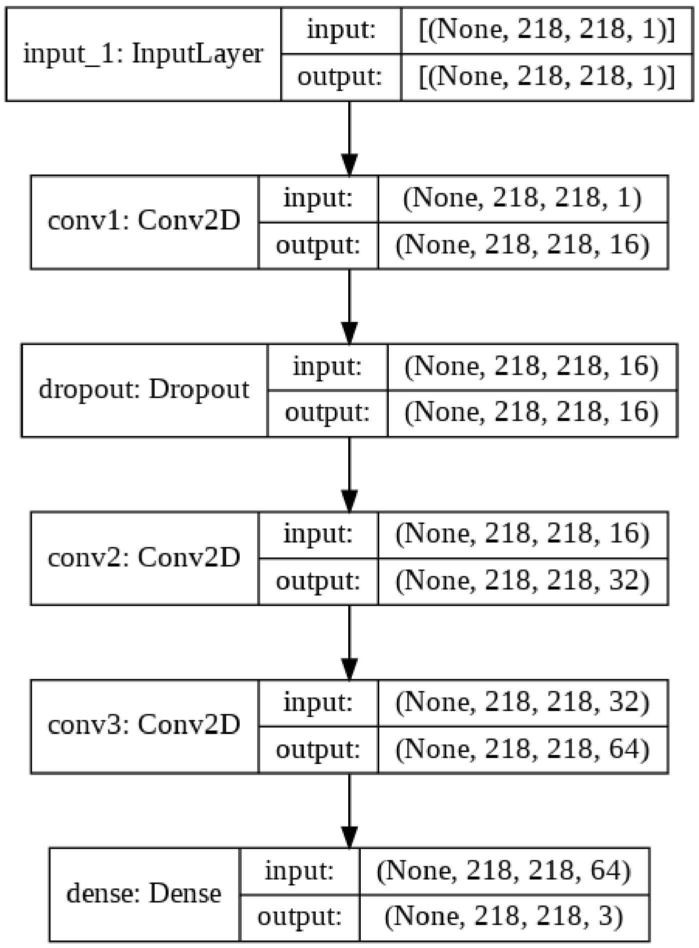
3.4 - Data Augmentation for source localization

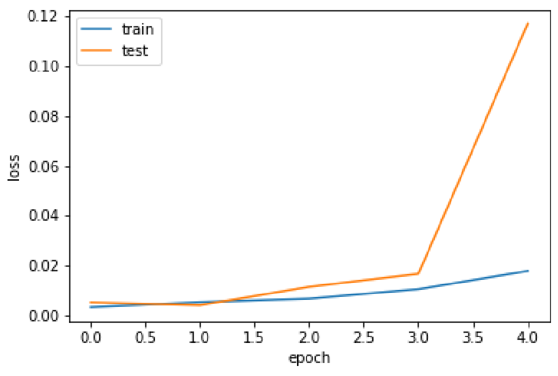
We experimented with data augmentation by flipping the images vertically and horizontally. Data augmentation helps to increase the number of training data and therefore to improve the training process, yielding slightly better prediction results.

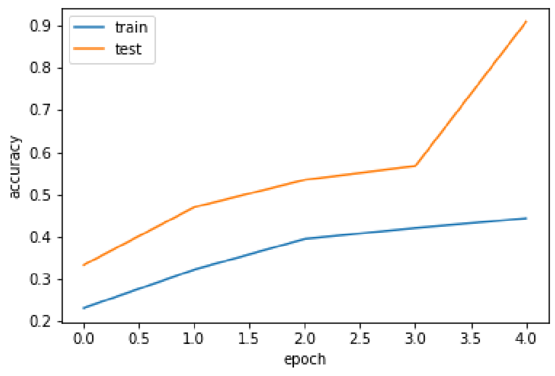
4 - Source classification

One of the purposes of the challenge was source population identification. There are three possible categories in our data for astronomic sources: SFG, AGN-steep and AGN-flat [2].
To represent the class data we chose the one-hot encoding: instead of a single image we opted for a 3-channel image where each channel represents one of the three classes provided for the astronomical sources.

As for localization, for the classification task we opted for a convolutional neural network. We initially adapted the model previously used for localization to the new task. The main difference was on the chosen loss function, categorical cross-entropy instead of a binary cross-entropy.
The only change to the model is the dimensionality of the output fully-connected layer, now set to 3, to represent the one-hot encoding adopted for the task.
The final difference with the original model is the activation function of the fully-connected layer, which previously used the sigmoid activation function, and now employs the softmax function.


input_1: InputLayer
input:
[(None, 218, 218, 1)]
output:
[(None, 218, 218, 1)]
conv1: Conv2D
input:
(None, 218, 218, 1)
output:
(None, 218, 218, 16)
dropout: Dropout
input:
(None, 218, 218, 16)
output:
(None, 218, 218, 16)
conv2: Conv2D
input:
(None, 218, 218, 16)
output:
(None, 218, 218, 32)
conv3: Conv2D
input:
(None, 218, 218, 32)
output:
(None, 218, 218, 64)
dense: Dense
input:
(None, 218, 218, 64)
output:
(None, 218, 218, 3)


0.12
0.10
0.08
0.06
0.04
0.02
0.00
loss
train
test
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
epoch


0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
accuracy
train
test
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
epoch

As can be seen in the presented graphs, the results of this first test were not accurate. Also, the loss value was not satisfactory, with an increasing trend that could be interpreted with an overfitting problem.
To improve the results and solve the overfitting there were different techniques that we could choose. In particular to improve the accuracy of our CNN we gradually tested different techniques. There are three main approaches that we considered:

1. Training with more data using Data Augmentation
2. Adding Dropout layers
3. Using BatchNormalization

4.1 - Data Augmentation for source classification

Training deep learning neural network models on more data can result in more skillful models, and the augmentation techniques can create variations of the images that can improve the ability of the fit models to generalize what they have learned to new images.
Data augmentation is a technique to artificially create new training data from existing training data. This is done by applying domain-specific techniques to examples from the training data that create new and different training examples.

Image data augmentation is perhaps the most well-known type of data augmentation and involves creating transformed versions of images in the training dataset that belong to the same class as the original image. Transforms include a range of operations from the field of image manipulation, such as shifts, flips, zooms, and much more.

Convolutional Neural Networks can learn features that are invariant to transformations in the image. Nevertheless, augmentation can further aid in this transform invariant approach to learning.
Image data augmentation is typically only applied to the training dataset, and not to the validation or test dataset. Taking advantage of the methods offered by the Keras library, we applied resizing, rescaling, rotation and flipping.

4.2 - Dropout

A single model can be used to simulate having a large number of different network architectures by randomly dropping out nodes during training [1]. This technique is called dropout and offers a

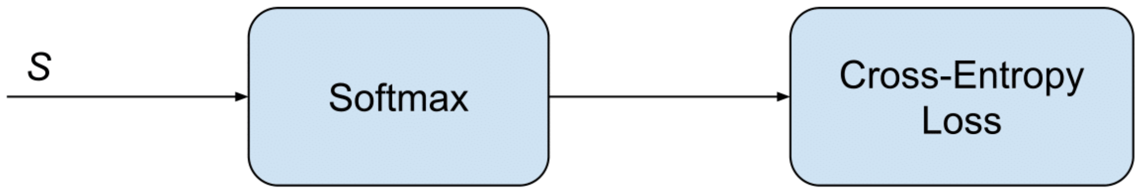
very computationally cheap and remarkably effective regularization method to reduce overfitting and improve generalization error in deep neural networks of all kinds.
In synthesis, dropout is a regularization method that approximates training a large number of neural networks with different architectures, in parallel.
During training, some number of layer outputs are randomly ignored or “dropped out.” This has the effect of making the layer behave as if it is composed of a different number of nodes. Ultimately, each update to a layer during training is performed with a different “view” of the configured layer.

4.3 - Batch Normalization

Batch normalization is a technique that allows every layer of the network to perform the learning process more independently as it normalizes the output of the previous layers. This has the effect of speeding up the training process by making it possible to use higher learning rates. It also reduces overfitting because it has a regularization effect.

4.4 - Categorical cross-entropy

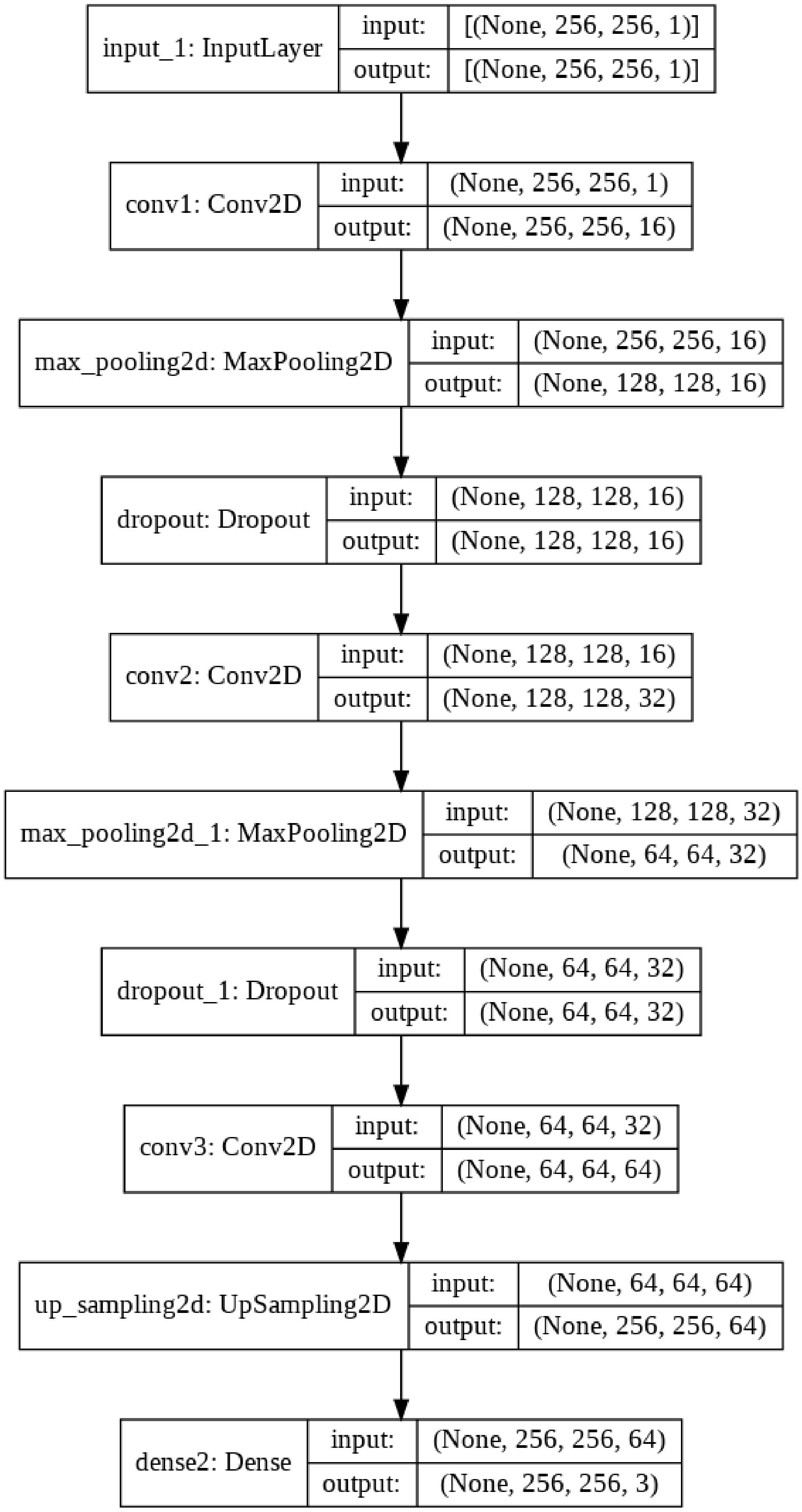
Categorical Cross-entropy is a loss function that is used in multi-class classification tasks. It is a Softmax activation combined with a Cross-Entropy loss. We used categorical cross-entropy to train our CNN to output a probability over the 3 classes for each image. In the specific case of Multi-Class classification, the labels are one-hot, so only the positive class keeps its term in the loss.



$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \qquad CE = -\sum_i^C t_i log(f(s)_i)$$

4.5 - Updated model

We applied these techniques gradually, first on the dataset composed of images of 560Mhz.
The architecture of the network goes as follow:



input_1: InputLayer
input: [(None, 256, 256, 1)]
output: [(None, 256, 256, 1)]
conv1: Conv2D
input: (None, 256, 256, 1)
output: (None, 256, 256, 16)
max_pooling2d: MaxPooling2D
input: (None, 256, 256, 16)
output: (None, 128, 128, 16)
dropout: Dropout
input: (None, 128, 128, 16)
output: (None, 128, 128, 16)
conv2: Conv2D
input: (None, 128, 128, 16)
output: (None, 128, 128, 32)
max_pooling2d_1: MaxPooling2D
input: (None, 128, 128, 32)
output: (None, 64, 64, 32)
dropout_1: Dropout
input: (None, 64, 64, 32)
output: (None, 64, 64, 32)
conv3: Conv2D
input: (None, 64, 64, 32)
output: (None, 64, 64, 64)
up_sampling2d: UpSampling2D
input: (None, 64, 64, 64)
output: (None, 256, 256, 64)
dense2: Dense
input: (None, 256, 256, 64)
output: (None, 256, 256, 3)

5 - Results

To calculate the metrics and evaluate our results we give some definitions:

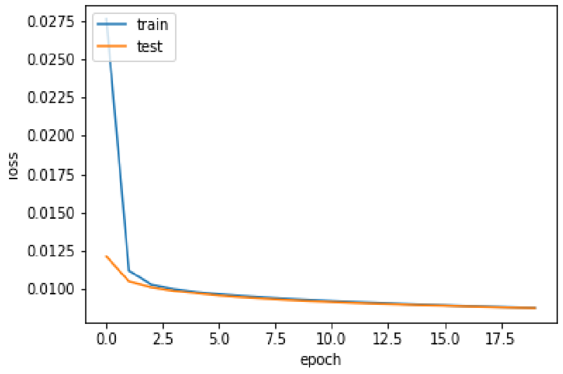
- True Positives: sum of pixels with a value greater than the threshold in the predicted solution map that are less than three pixels away from the location of a source in the true solution map.
- False Positives: sum of pixels with a value greater than the threshold in the predicted solution map that are more than three pixels away from the location of a source in the true solution map.
- False Negatives: sum of pixels with a value lower than the threshold in the predicted solution map that are less than three pixels away from the location of a source in the true solution map

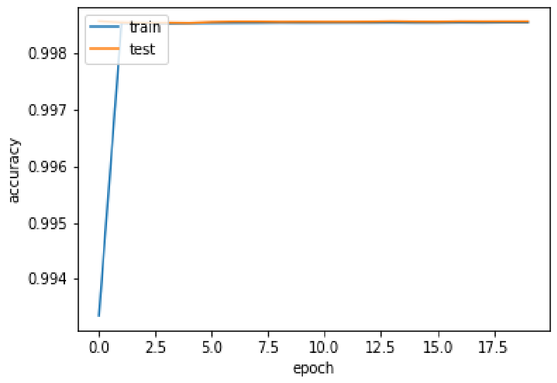
We focus on the precision and recall metrics and use the F1 score metric to combine them:

- ➢ Precision: $TruePositives\ /\ (TruePositives\ +\ FalsePositives)$
- ➢ Recall: $TruePositives\ /\ (TruePositives\ +\ FalseNegatives)$
- ➢ F1 score: $(2\ x\ Precision\ x\ Recall)\ /\ (Precision\ +\ Recall)$

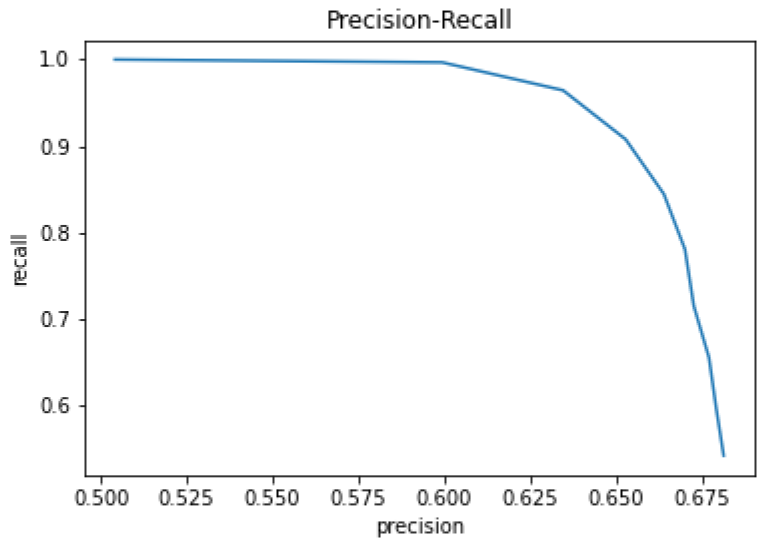
5.1 - Localization Results

Using the model described above, after 20 epochs the training process produces the following results for loss and accuracy on the 560 MHz and 8 hours integration data:





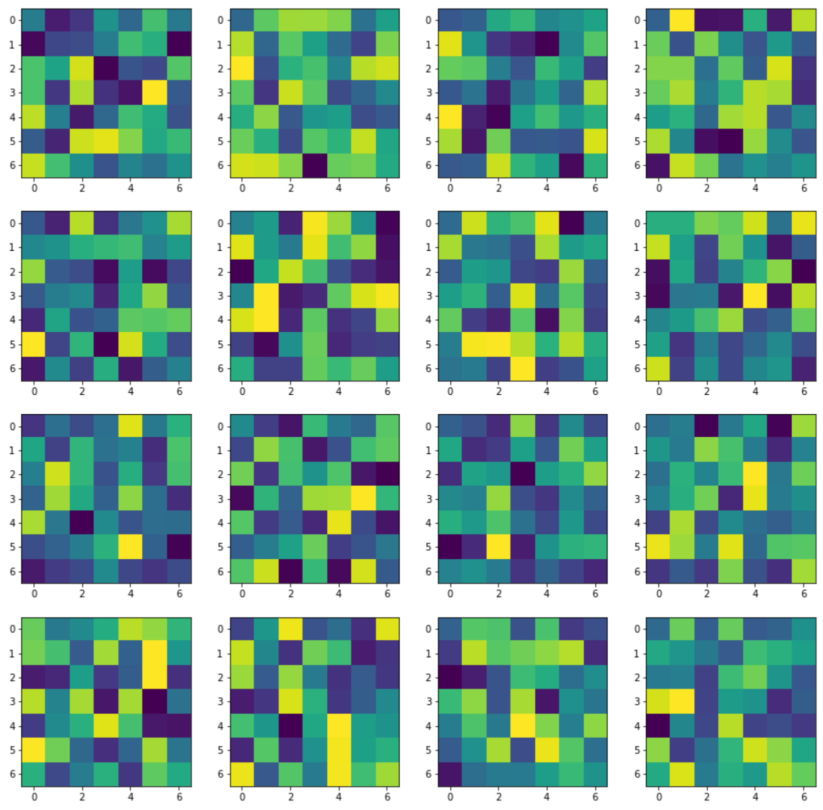
After the training, we can predict the values of the test set and evaluate the results obtained in this way with the metrics defined previously. Using different thresholds for the prediction maps we obtain different values for the precision and recall metrics.



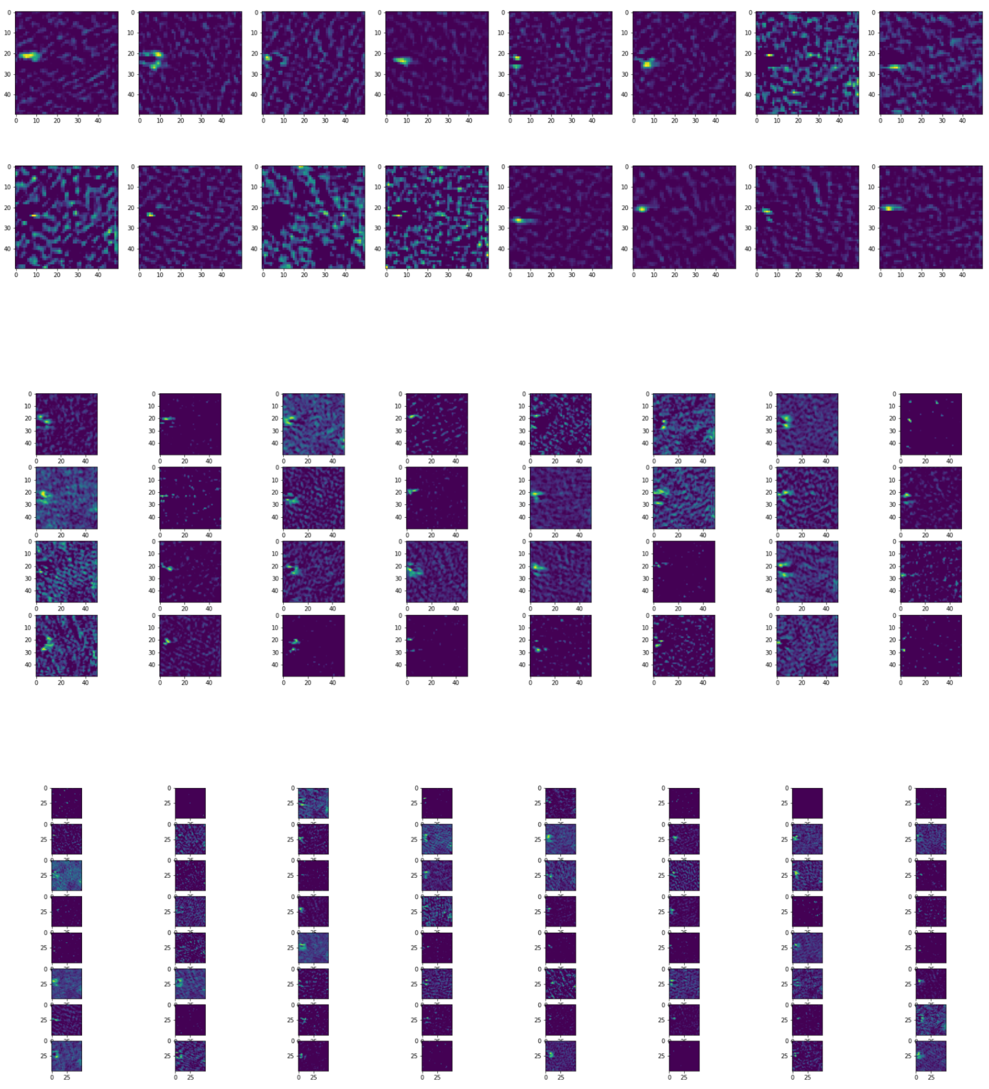
The Precision-Recall curve describes how the two metrics change according to the chosen threshold

The best results are obtained with a threshold of *0.04:*

	Precision	*Recall*	*F1 score*
Training Set	*0.617411*	*0.939898*	*0.745264*
Validation Set	*0.584396*	*0.935133*	*0.719286*
Test Set	*0.583919*	*0.938231*	*0.719839*



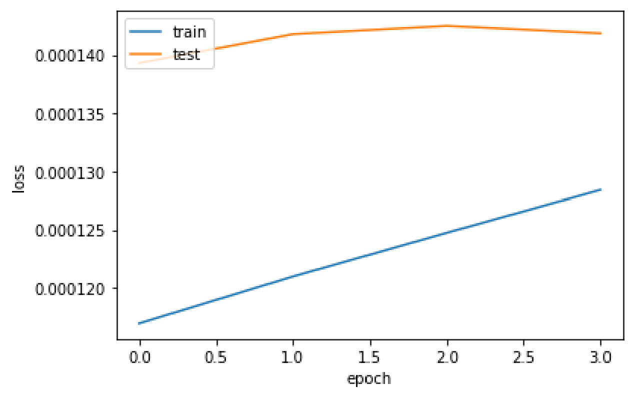
Example of the 7x7 filters learned by the first convolutional layer

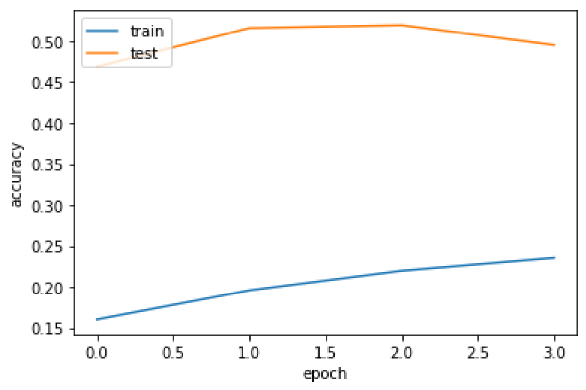


Final feature maps of the model for the three convolutional layers

5.2 - Classification Results

Even after having experimented with different techniques to improve the performance of our classification model, the best results obtained for loss and accuracy are not at the level of those of the localization task. After training for 10 epochs on the 560 MHz dataset the model produces the following statistics:





With the other datasets (1400Mhz and 9200Mhz) the results are even less accurate.

6 - Conclusions

Our main focus has been on the localization task, where we achieved the best results. Our model yields good results on the majority of the image files provided in the challenge and succeeds in identifying a good number of astronomic sources producing an accurate source localization map. The model performs well on all the frequencies and telescope integrations provided for the challenge.

The model built for the classification tasks does not achieve the same results and fails to produce quality predictions. This is due to the nature of the task approached which is much more complex than the localization and would require the deployment of more complicated models.

References

[1] Ian Goodfellow, Yoshua Bengio, Aaron Courville, *Deep Learning,* *www.deeplearningbook.org*

*[*2] Bonaldi et al., *Square Kilometer Array Science Data Challenge 1: analysis and results,* *https://arxiv.org/abs/2009.13346*

[3] Brightness in Radio Astronomy http://physics.wku.edu/~gibson/radio/brightness.html