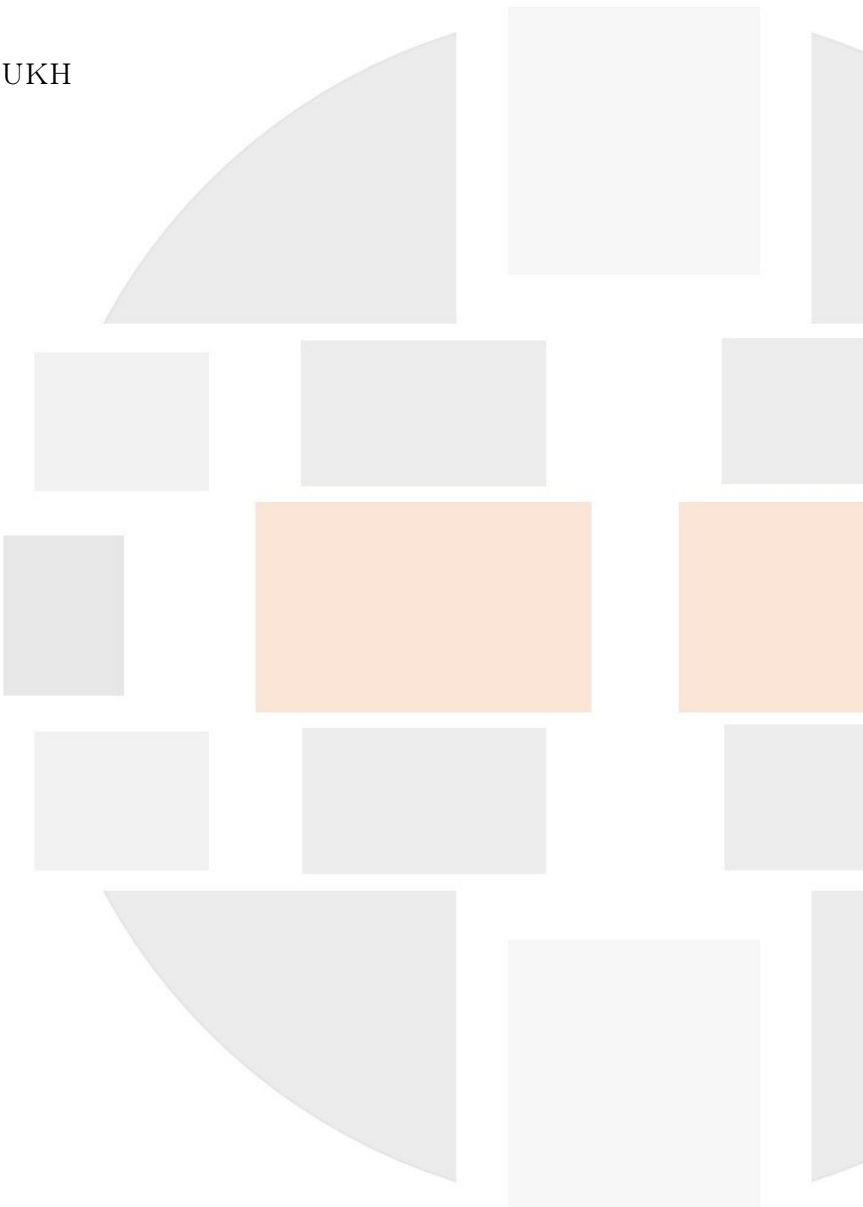
PROJECT REPORT

Deep Learning

Deep Calibration

Author:
Issame Sarroukh



Contents

1 Introduction

This report presents the results obtained after testing different neural network architectures in a problem of calibration of a stochastic volatility model. For each configuration of the network, we display a heat matrix of size maturity*times*strike expressing the error of the model according to 3 approaches: the mean relative error, the standard relative error and the maximum relative error.

2 Formulation of the problem

We are interested in the calibration of the following stochastic volatility model:

$$\begin{cases} \frac{dS_t}{S_t} = rdt + \sigma_t dW_t^1 \\ \sigma^2 = \sigma_0^2 \, e^{2\nu X_t - 2\nu^2 Var(X_t)} \\ dX_t = -\kappa X_t dt + dW_t^2 \\ dW_t^1 dW_t^2 = \rho dt \end{cases} \tag{1}$$

The model involves 4 parameters:

- σ_0 the initial volatility.
- κ the speed of return to the mean.
- ν the standard deviation of the volatility.
- ρ the correlation coefficient between the underlying and the volatility.

The objective is to set up a neural network that explains the implied volatility of a European call of strike K and maturity T based on the values of the parameters of the model (1).

To create the dataset, 5000 uniform realizations of $\theta = (\sigma_0, \kappa, \nu, \rho)$ were generated over $\mathbb{R}_+^* \times \mathbb{R}_+^* \times \mathbb{R}_+^* \times [-1, 1]$ and a range of 8 maturities $\{0.1, 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.0\}$ and 11 strikes $\{0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.1, 1.2, 1.3, 1.4, 1.5\}$ were considered. This corresponds well to a data set of size 440 000.

For each realization of the tuple $(\sigma_0, \kappa, \nu, \rho)$, we generate the model trajectories (1) according to the following discretized schemes:

$$\begin{cases} X_{t_{i+1}} = e^{-\kappa \Delta t} X_{t_i} + \sqrt{\frac{1-e^{-2\kappa\Delta t}}{2\kappa}} Z_{t_{i+1}}^1, \quad Z_{t_{i+1}}^1 \longrightarrow \mathcal{N}(0,1) \\ \sigma_{t_{i+1}} = \sigma_0 \exp\left[\nu X_{t_{i+1}} - \frac{\nu^2}{2\kappa}(1 - e^{-2\kappa t_{i+1}})\right] \\ \mathcal{E}_{t_{i+1}} = \frac{\rho}{\kappa}(1 - e^{-\kappa\Delta t}) Z_{t_{i+1}}^1 + \sqrt{1 - \left[\frac{\rho}{\kappa}(1 - e^{-\kappa\Delta t})\right]^2} Z_{t_{i+1}}^2, \quad Z_{t_{i+1}}^2 \longrightarrow \mathcal{N}(0,1) \\ S_{t_{i+1}} = S_{t_i} \exp\left[(r - \frac{\sigma_{t_i}^2}{2})\Delta t + \sigma_{t_i}\sqrt{\Delta t}\, \mathcal{E}_{t_{i+1}}\right] \end{cases} \tag{2}$$

For each maturity T, we retrieve the corresponding realizations of the underlying S_T and we compute for the different strikes K the call prices by the Monte Carlo method: $\frac{1}{M}\sum_{i=1}^{M} e^{-rT}(S_T^i - K)_+$ where M denotes the number of simulations.

Once the price is obtained, the corresponding implied volatility is calculated by inverting the Black's formula, which can be obtained by using a search algorithm for the zeros of the function.

With the dataset ready, the simulated data are divided into training and test datasets at 85% and 15% proportions respectively and normalized.

In the following section, we will focus on the development of the neural network to address the problem and we will try to test several approaches and architectures and analyze their impact.

3 Test de différentes architectures

The classical neural network model is composed of an input layer, an output layer and sub-layers connecting the input to the output, the relations between the neurons of two consecutive layers are linear with then the application of an activation function. The network optimization algorithm is based on gradient descent to estimate the neural network parameters. In the following, we examine the influence of the different parameters associated with the neural network architecture on the quality of the model in the context of calibration.

The training is performed initially on a Google Cloud virtual machine with a Tesla P100 GPU, and then on a local machine with a GTX 1070 GPU. To make the times comparable, we preferred to use the local machine.

3.1 Default architecture

The architecture used is the one found in the notebook, except that we add the early shutdown.

$\lambda = 10^{-3}$, adam optimizer, epochs=256, batch size=32, nlayers=2, neurones=2^5, elu, (early stopping patience=10 is added).

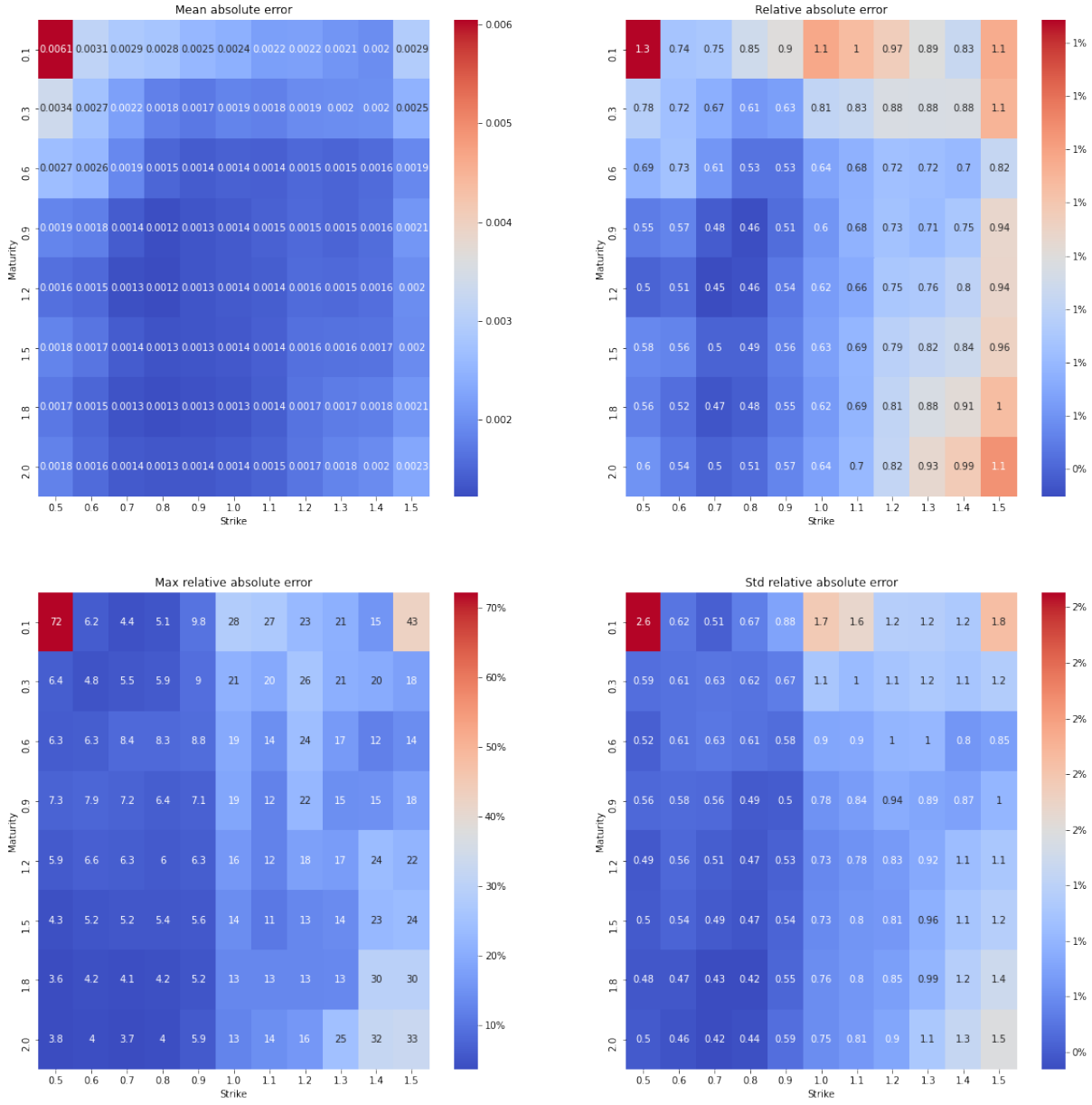


Figure 1: Heatmap on training data

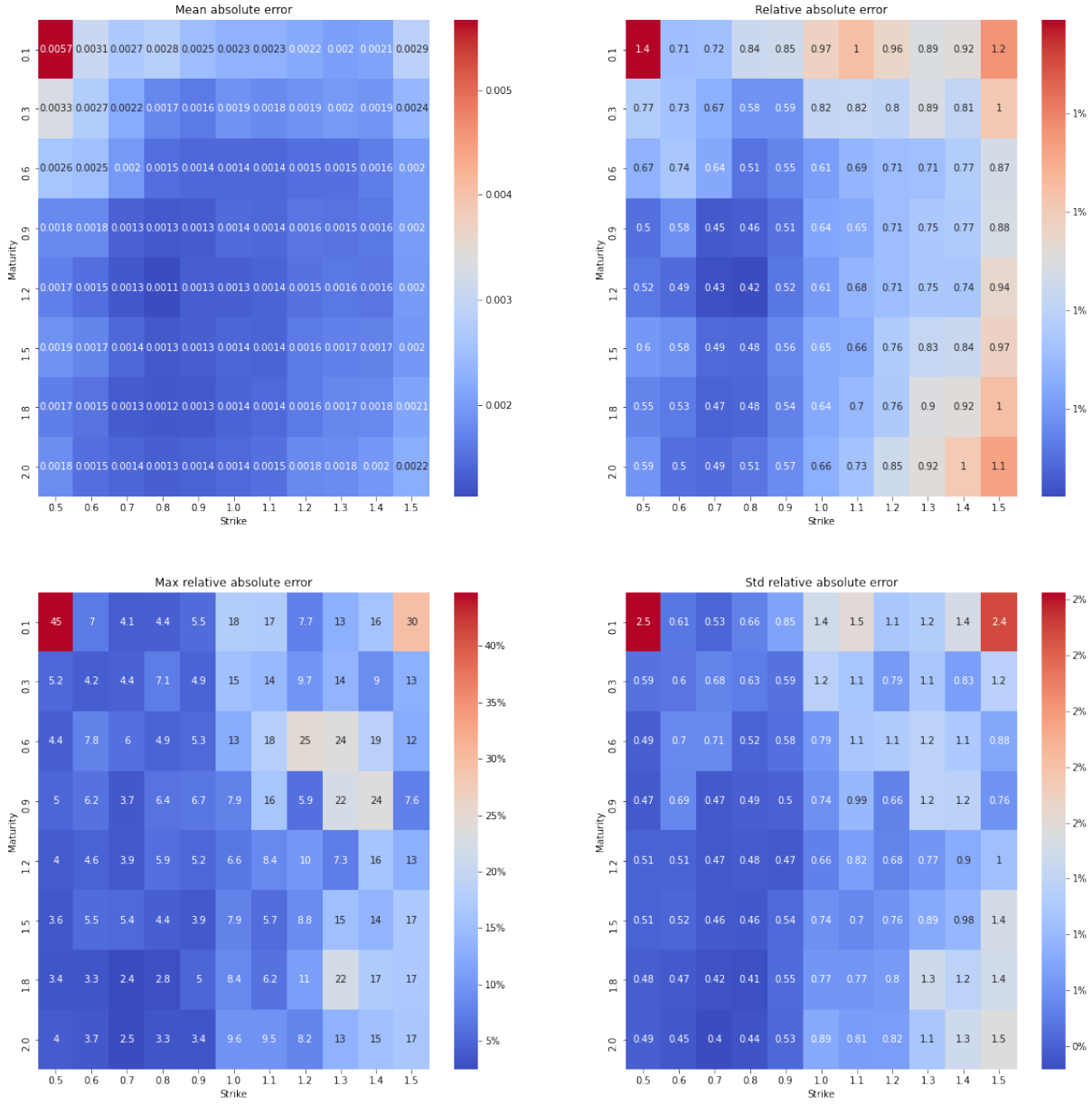


Figure 2: Heatmap on testing$_data$

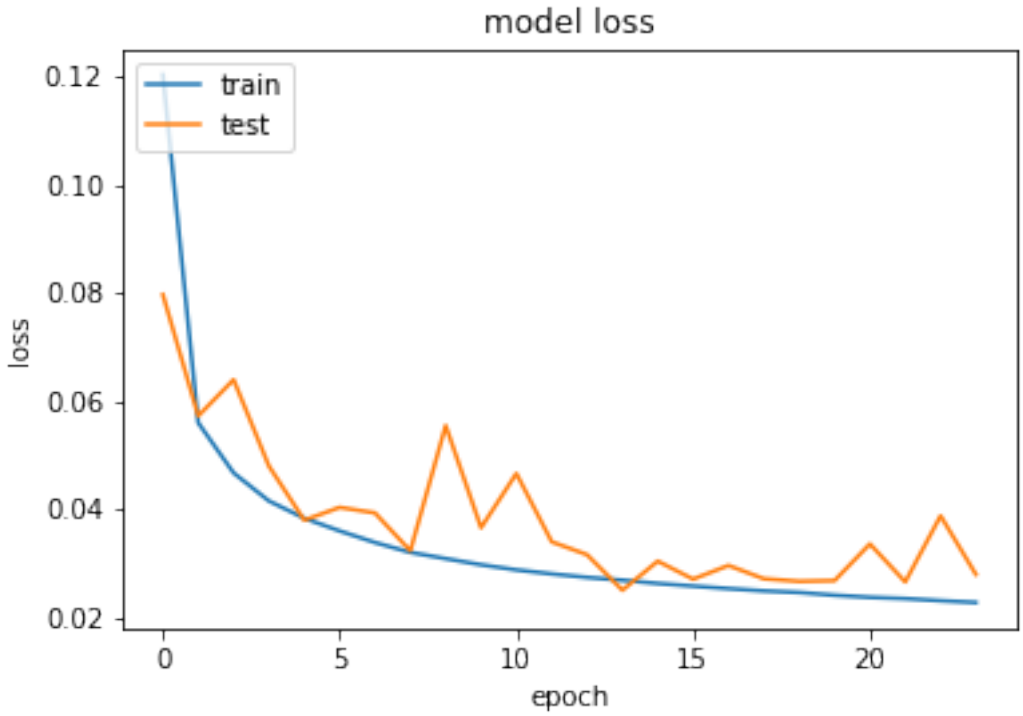


Figure 3: Model error as a function of the number of epochs

Noting that in the figure, the validation error is unstable, and that the convergence of the model is about 13min, we will examine next the effect of changing the model parameters on its convergence and on the quality of the calibration.

3.2 Impact of learning rate

In this part, we try to quantify the impact of the learning rate. We keep the same architecture as before, only varying the value of λ which we take this time equal to 10^{-4}. That is to say that we initialize a neural network with the following parameterization:

$\lambda = 10^{-4}$, adam optimizer, epochs=256, batch size=32, nlayers=2, neurones=2^5, elu, early stopping patience=10.

The neural network took a time of 26min and 32s for its training based on the training data.

We can visualize in the following the prediction results of our model in the form of a heatmap on the train data (4) and also on the test data (5):

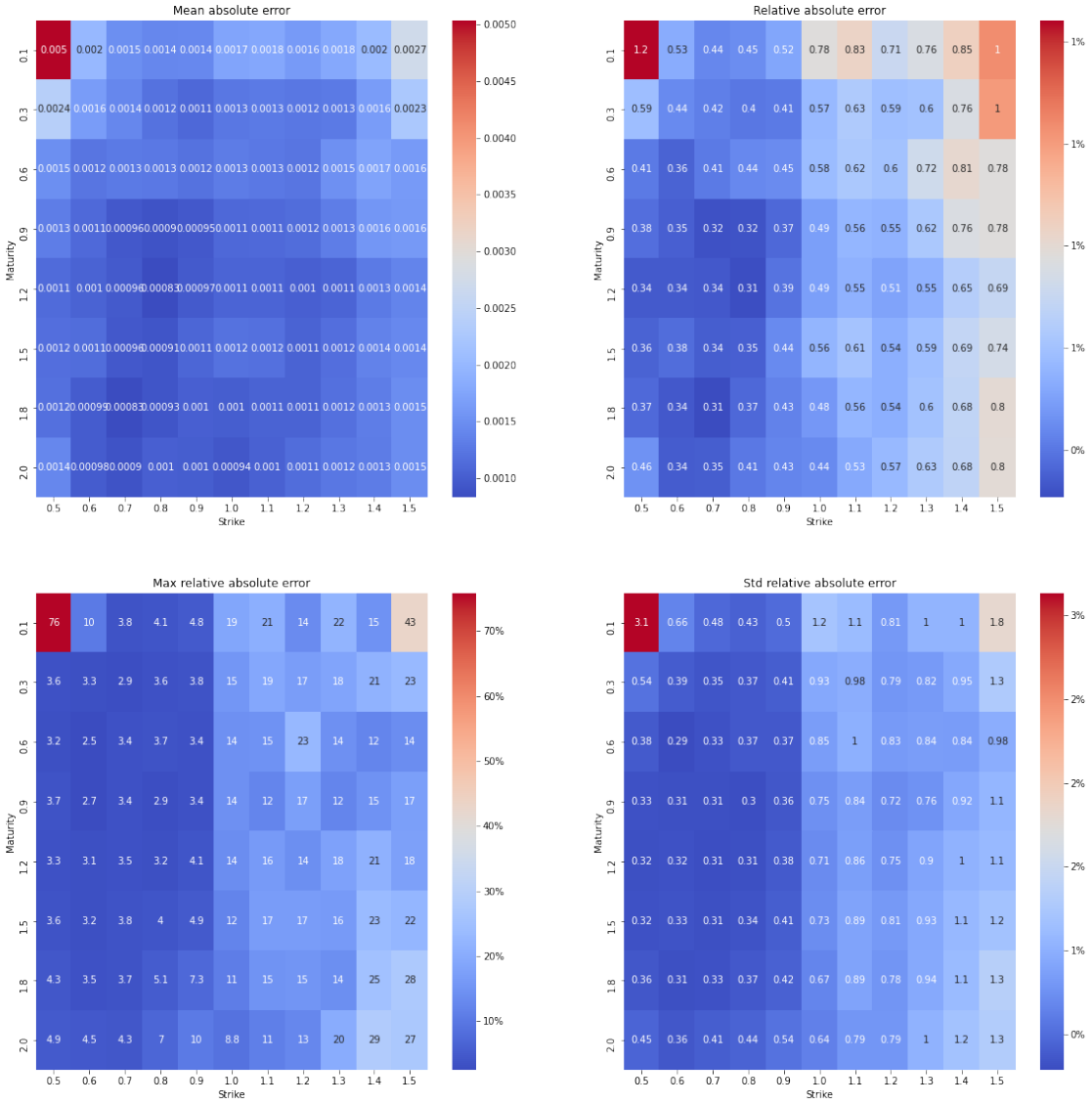


Figure 4: Heatmap for train data for $\lambda = 10^{-4}$

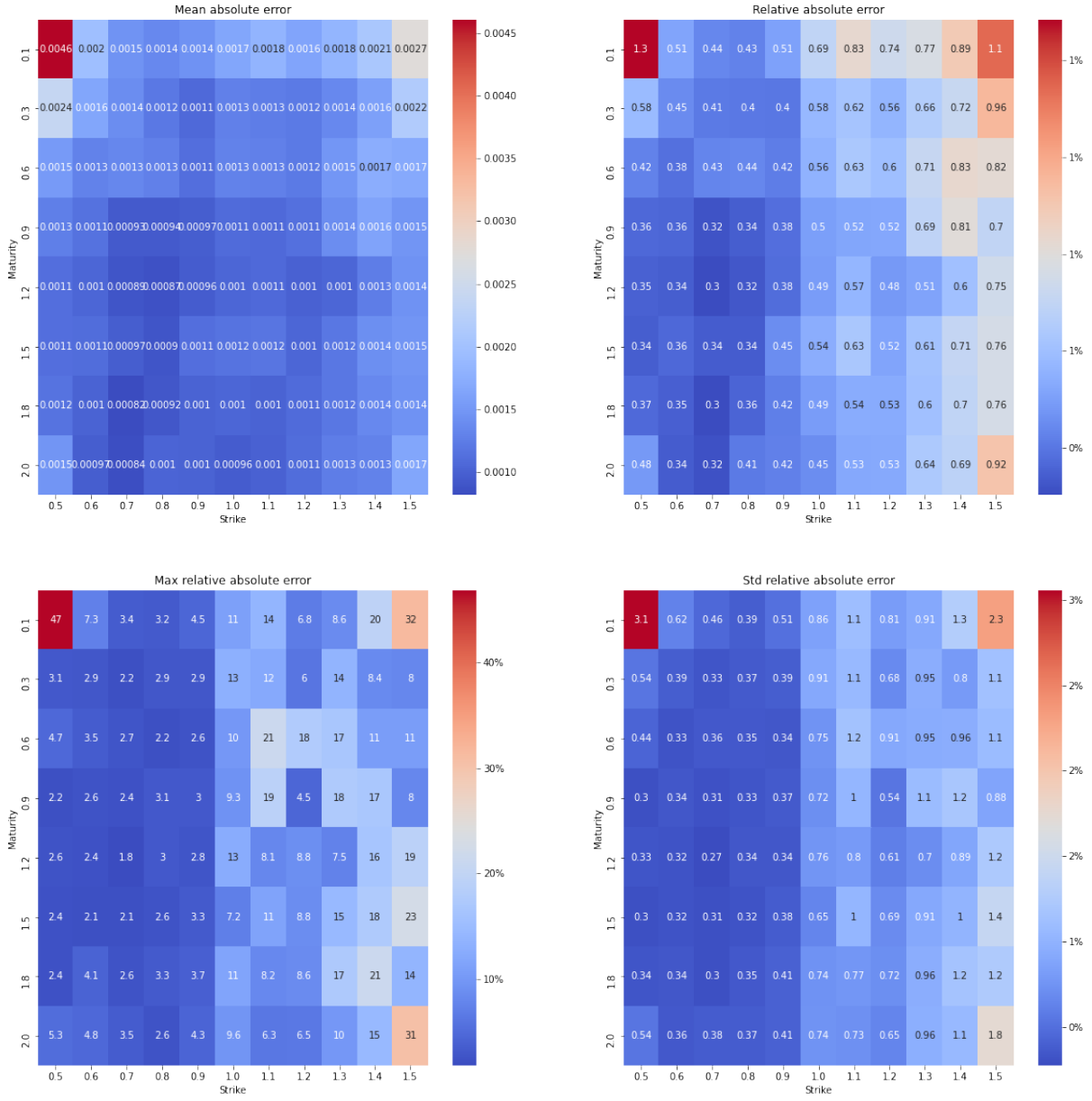


Figure 5: Heatmap for test data for $\lambda = 10^{-4}$

The figures 4 and 5 show that the average relative error (for all parameter combinations) between the volatilities predicted by the neural network and the volatilities obtained by the Monte Carlo method is significantly lower than 1 (top right image in figures 4 and 5). The maximum relative error is up to 76% for the train data and 47% for the test data. For all types of error observed, the model records a maximum error each time for

the maturity 0.1 and the strike 0.5 and remains efficient for the calculation of volatilities close to the money where it records the least error.

In the following, we plot the model error as a function of the number of epochs:

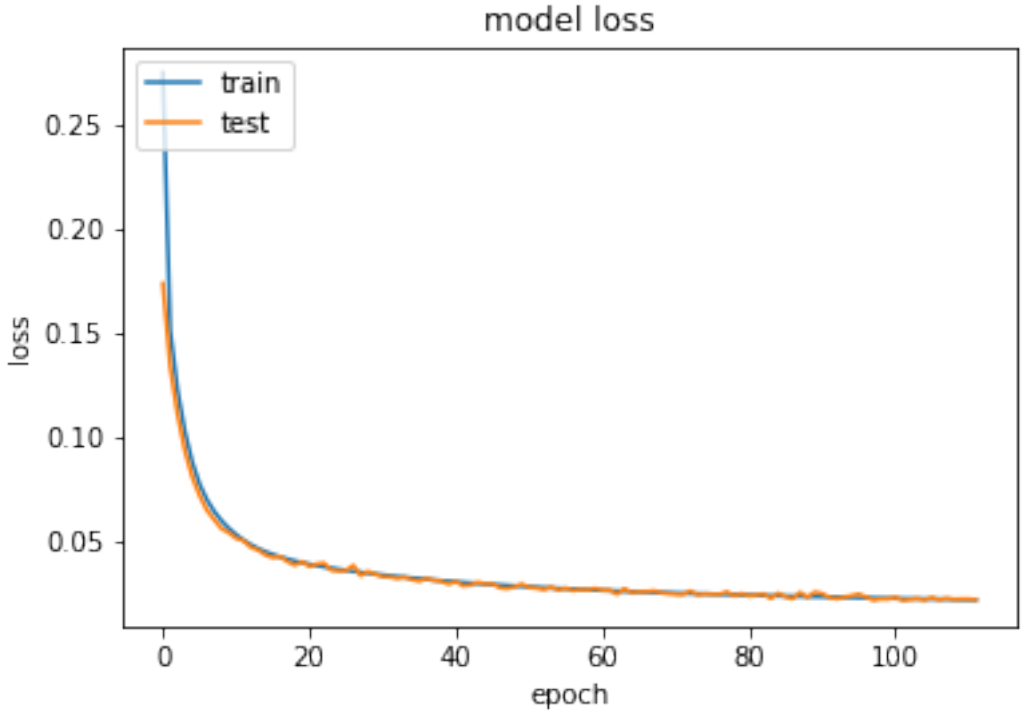


Figure 6: Erreur du modèle en fonction du nombre d'epochs

We can see that the error of the neural network remains invariant between the train data and the test data, which proves that the model generalizes well. Compared to the error function of the previous section, we notice a great stability. This is certainly due to the fact that the training rate has been reduced in the current configuration, which has allowed the model to gain in stability and convergence but in return has considerably lengthened the training process.

3.3 Impact of Batch size

In this part, we study the influence of the Batch size by adopting the following configuration of the neural network:

$\lambda = 10^{-4}$, adam optimizer, epochs=256, batch size=128, nlayers=2, neurones=2^5, elu, early stopping patience=10.

The neural network took a time of 17min and 22s for its training based on the training data.

We can visualize in the following the prediction results of our model in the form of a heatmap on the train data (7) and also on the test data (8):

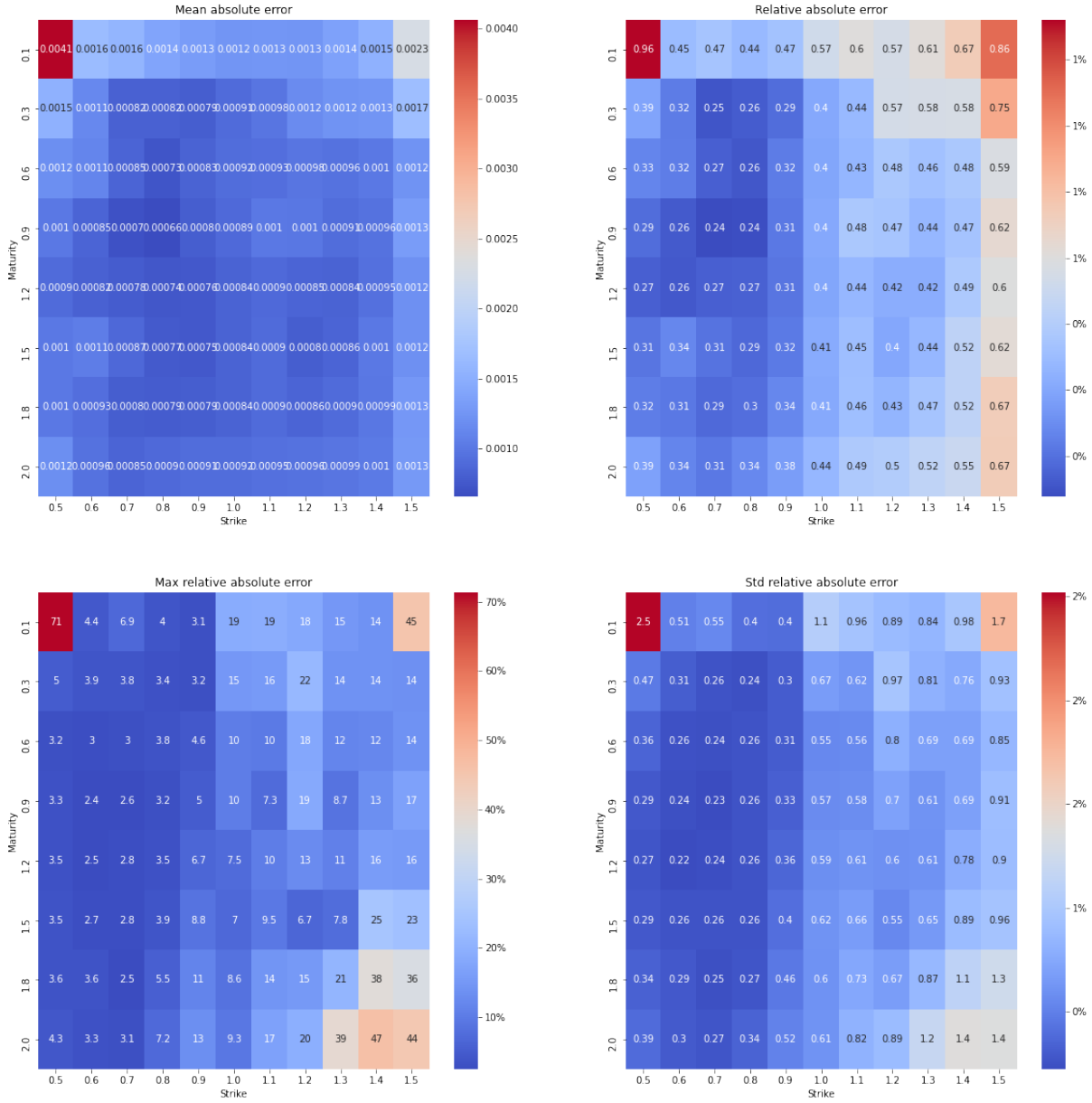


Figure 7: Heatmap for train data forr batch size = 128

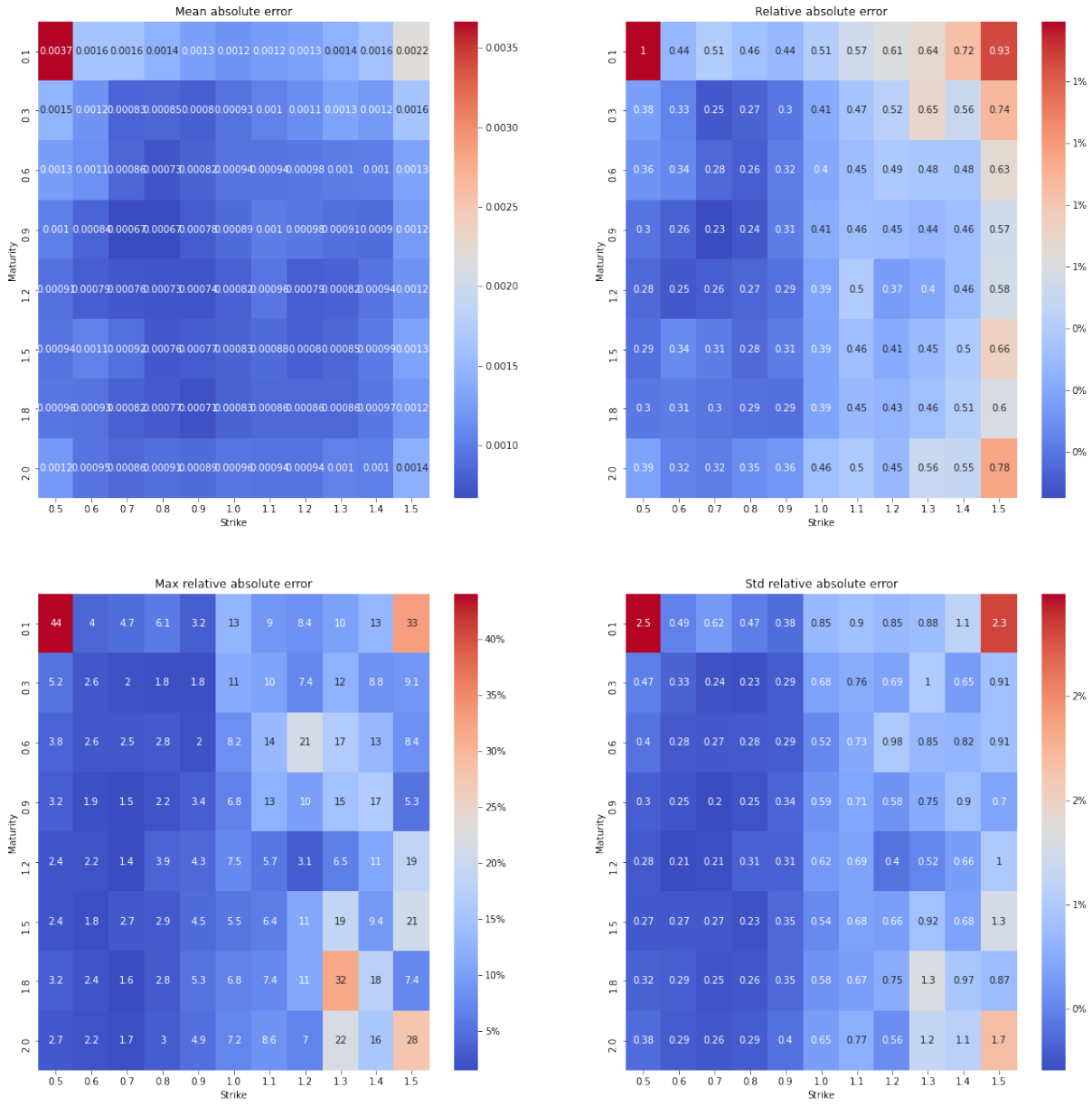


Figure 8: Heatmap for test data for batch size = 128

The average relative error (for all parameter combinations) from the figures 7 and 8 is well below 1%. While the standard relative error reaches its maximum of 2.5% for both train and test data again for maturity 0.1 and strike 0.5 (bottom right image in figures 7 and 8). The maximum relative error goes up to 71% for the train data and 44% for the test data.

In the following, we visualize the evolution of the model error as a function of the number of epochs:

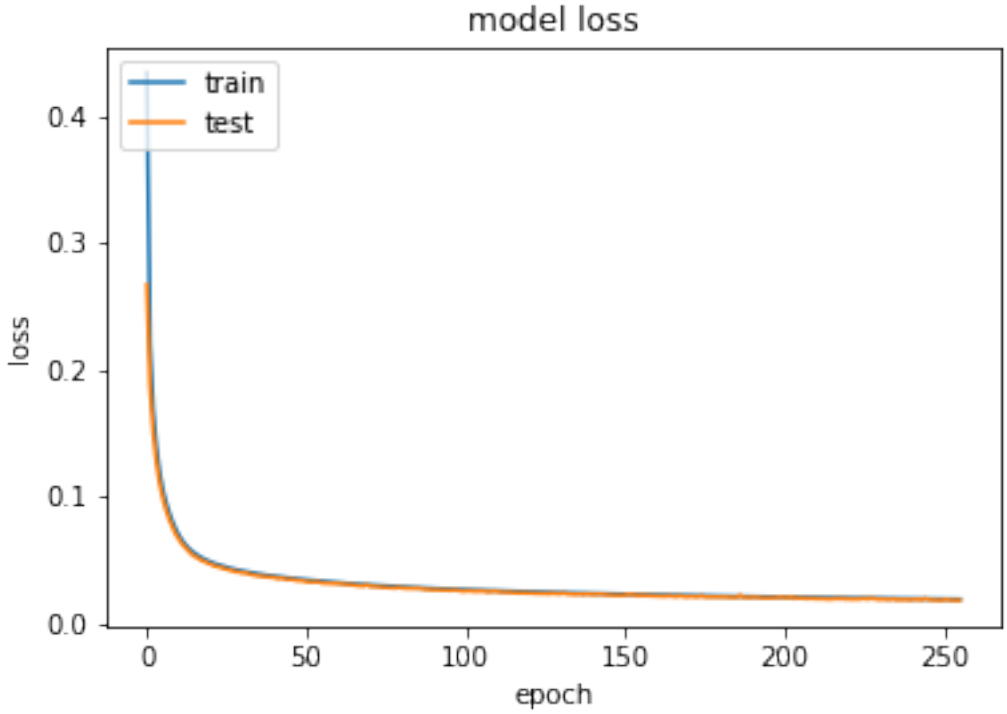


Figure 9: Evolution of the model error as a function of the number of epochs

We see that the model error is stable on both train and test data. A large value of the batch size increases the accuracy of the model estimation and reduces the learning time (17min and 22s) by ensuring a faster convergence.

3.4 Impact of the number of neurons

In this section, we adopt the same parameterization as before, this time lowering the number of neurons to 16. The configuration of the model is as follows:

$\lambda = 10^{-4}$, adam optimizer, epochs=252, batch size=128, nlayers=2, neurones=2^4, elu, early stopping patience=10.

The neural network took a time of 15min and 59s for its training based on the training data.

We can visualize in the following the prediction results of our model in the form of heatmap on the train data (10) and also on the test data (11):

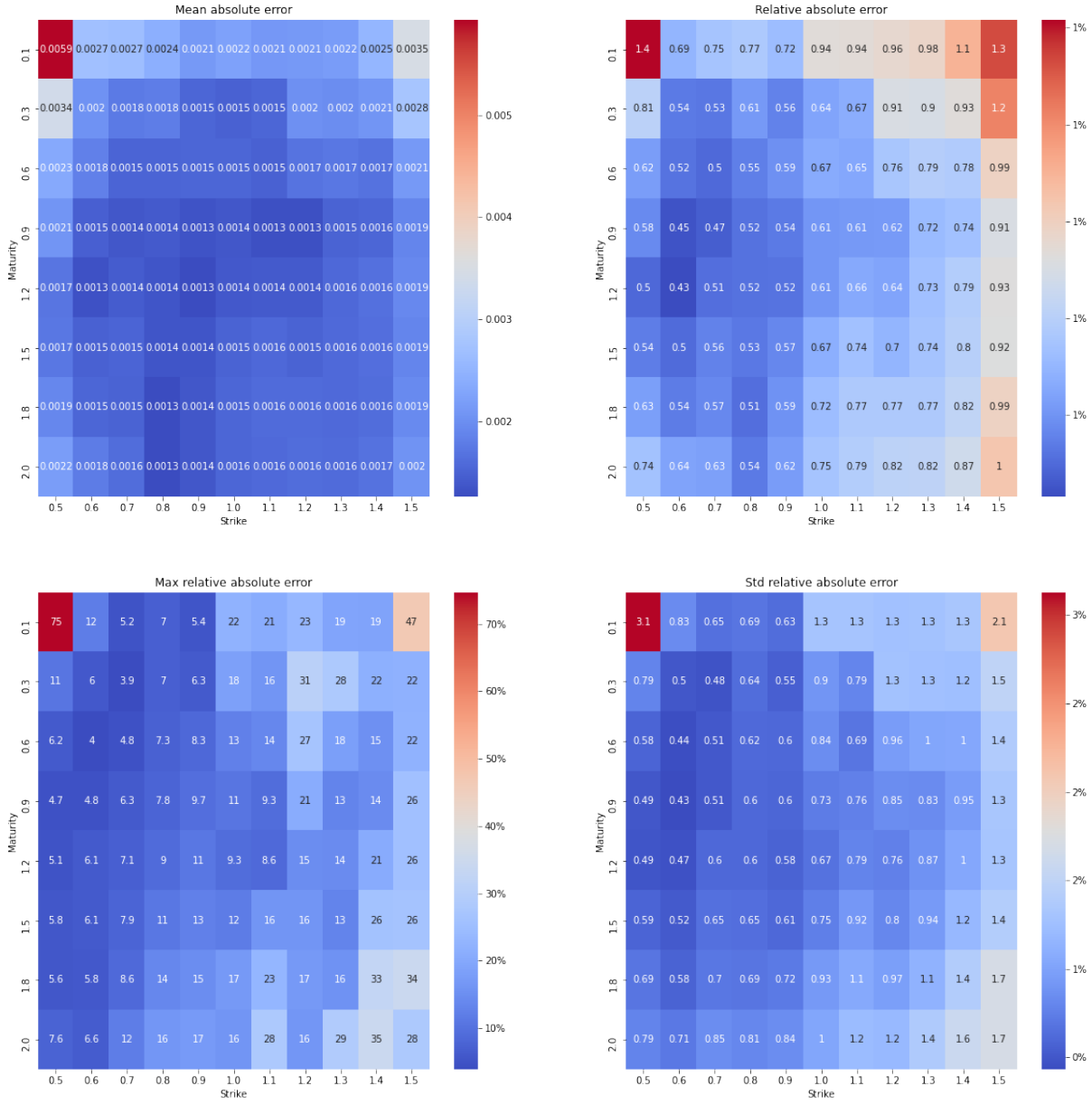


Figure 10: Heatmap for train data for a number of neurons = 16

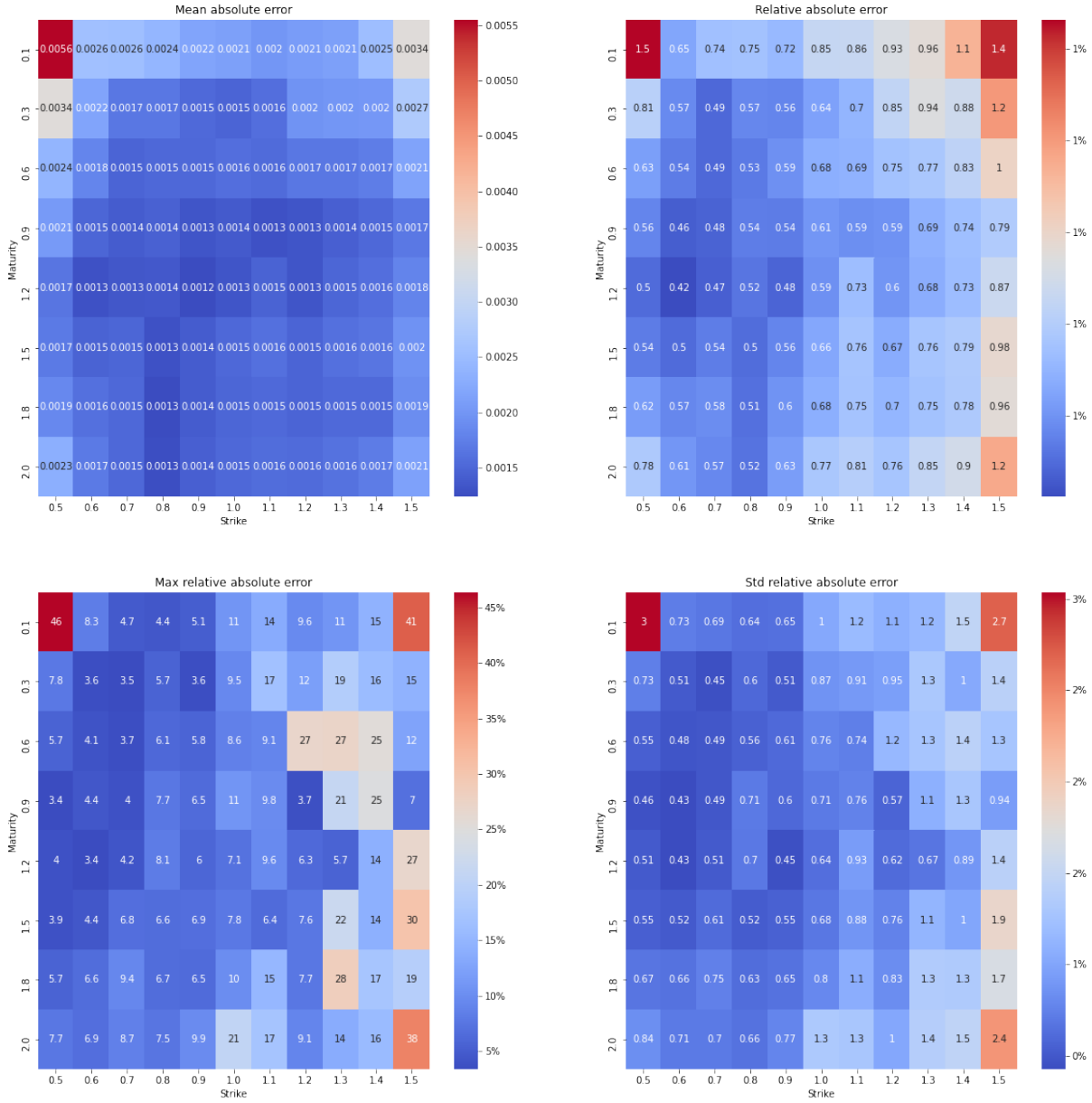


Figure 11: Heatmap for test data for a number of neurons = 16

The average relative error from figures 10 and 11 is for any strike and maturity less than 1.5%. While the maximum relative error is up to 75% for the train data and 46% for the test data (bottom left image in figures 10 and 11). As for the relative standard error does not exceed 3% for the train and test samples.

In the following, we increase the number of neurons to 64. The configuration of the model is as follows:

$\lambda = 10^{-4}$, adam optimizer, epochs=252, batch size=128, nlayers=2, neurones=2^6, elu, early stopping patience=10.

The prediction results of our heatmap model on the train data and also on the test data can be seen in figures (12) and (13) respectively:

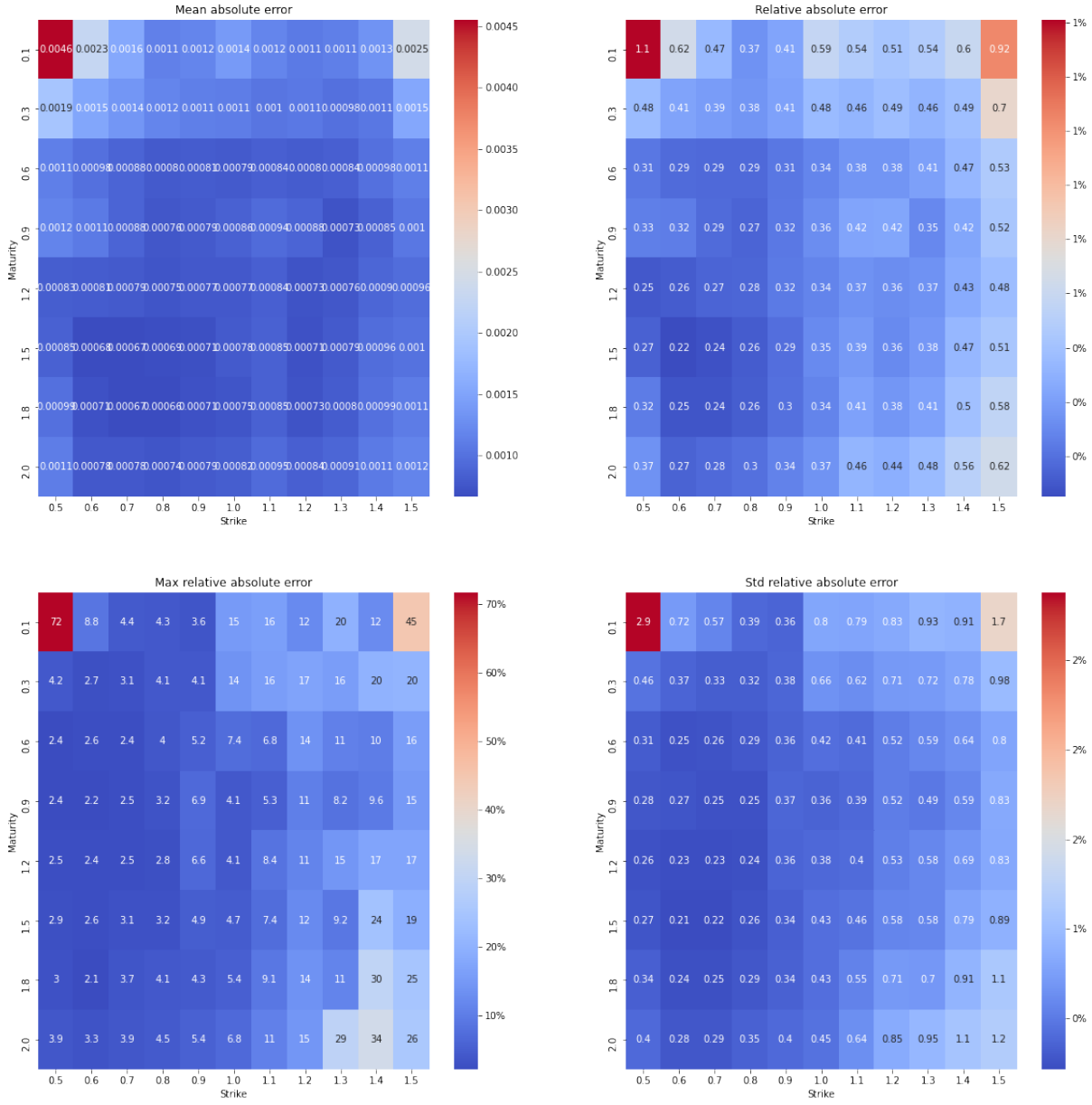


Figure 12: Heatmap for train data for a number of neurons = 64

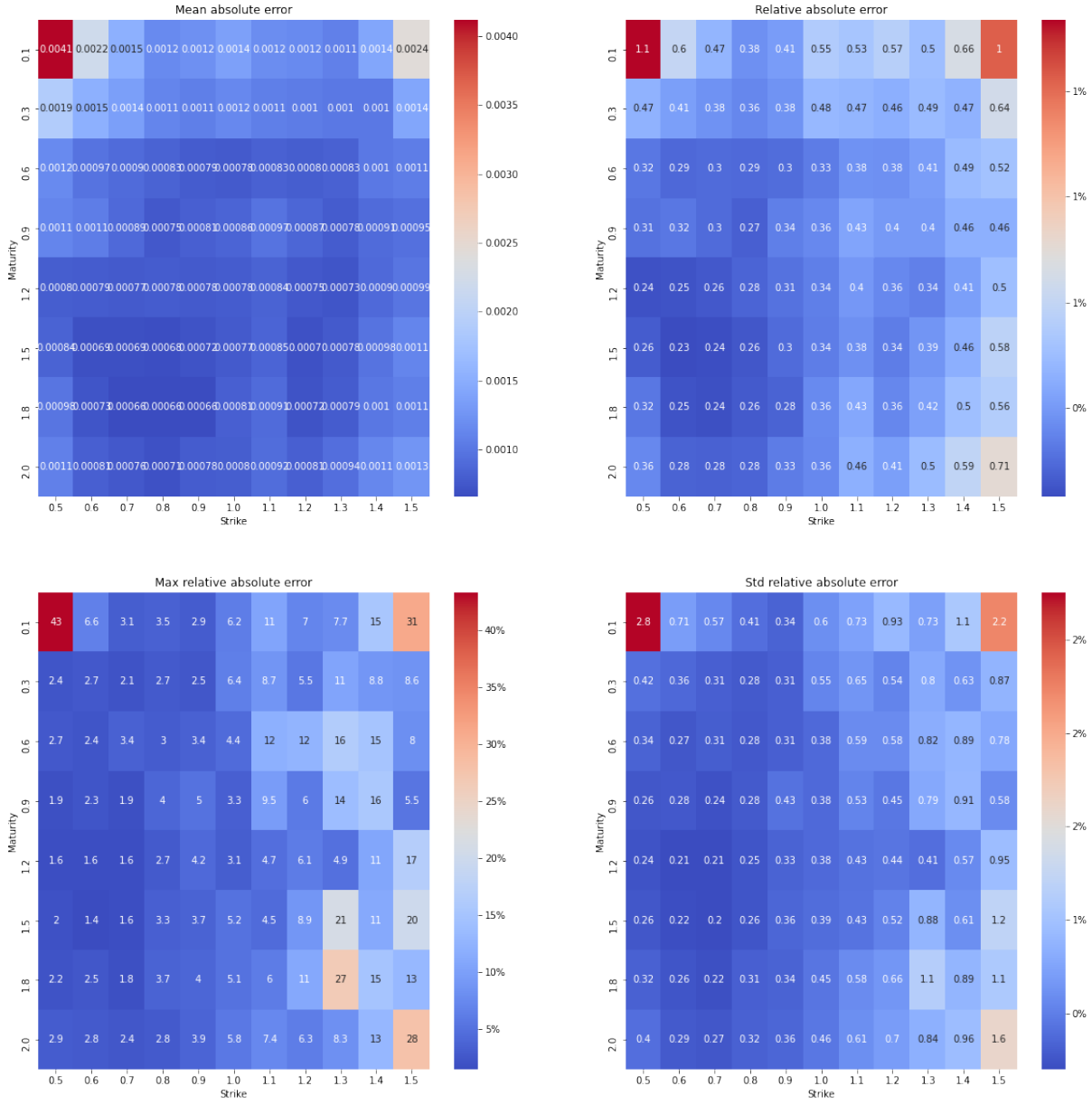


Figure 13: Heatmap for test data for a number of neurons = 64

The average relative error according to figures 12 and 13 is for any strike and maturity lower than 1.1% (right image on top in figures 12 and 13. While the maximum relative error goes up to 2% for the train data and 43% for the test data, reached in the maturity 0.1 and strike 0.5. As for the standard relative error does not exceed 2.9% for the train and test samples.

The following figures show the evolution of the error as a function of the number of epochs for the 2 cases:

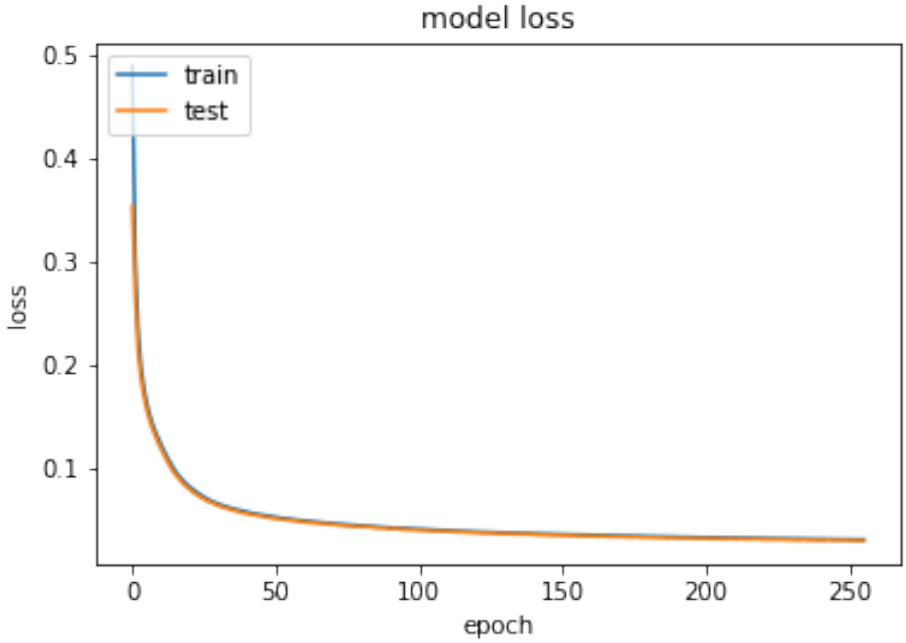


Figure 14: Evolution of the model error as a function of the number of epochs for a number of neurons = 16

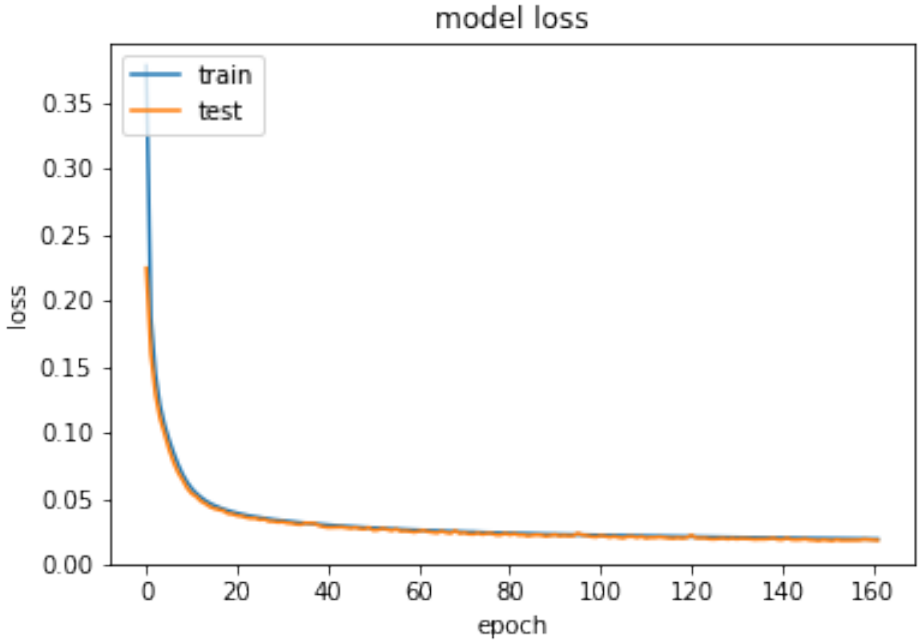


Figure 15: Evolution of the model error as a function of the number of epochs for a number of neurons = 64

From the two previous figures, we can deduce that the number of neurons has a significant impact on the prediction quality of the model. The more we increase the number of neurons, the more the accuracy of the model increases with respect to the estimation of the training data but with a risk of over-fitting and loss of generalization of the model. In this case, the learning phase of the model takes a considerable amount of time, whereas for a much smaller number of neurons, this time is not too important, but we lose in terms of the accuracy of the model (see 14).

3.5 Impact of the number of layers

The chosen parameters are those giving the best optimization, with a training rate $\lambda = 10^{-4}$, the adam optimizer, and a number of epochs equal to 256, and a batch size=128, the number of layers has been increased and equal to 3, the number of neuron is fixed to 2^5, the strategy of early stop is always used, with a stop of training after 10 steps of non-improvement of the model, the estimated time of the training is 15min 51s with the elu activation function.

The number of layers allows to add an additional parameterization to the neural network, thus allowing to approximate more precisely the volatility we are trying to estimate, it should be noted that this generally has an impact on the learning time, and can induce problems like overfitting.

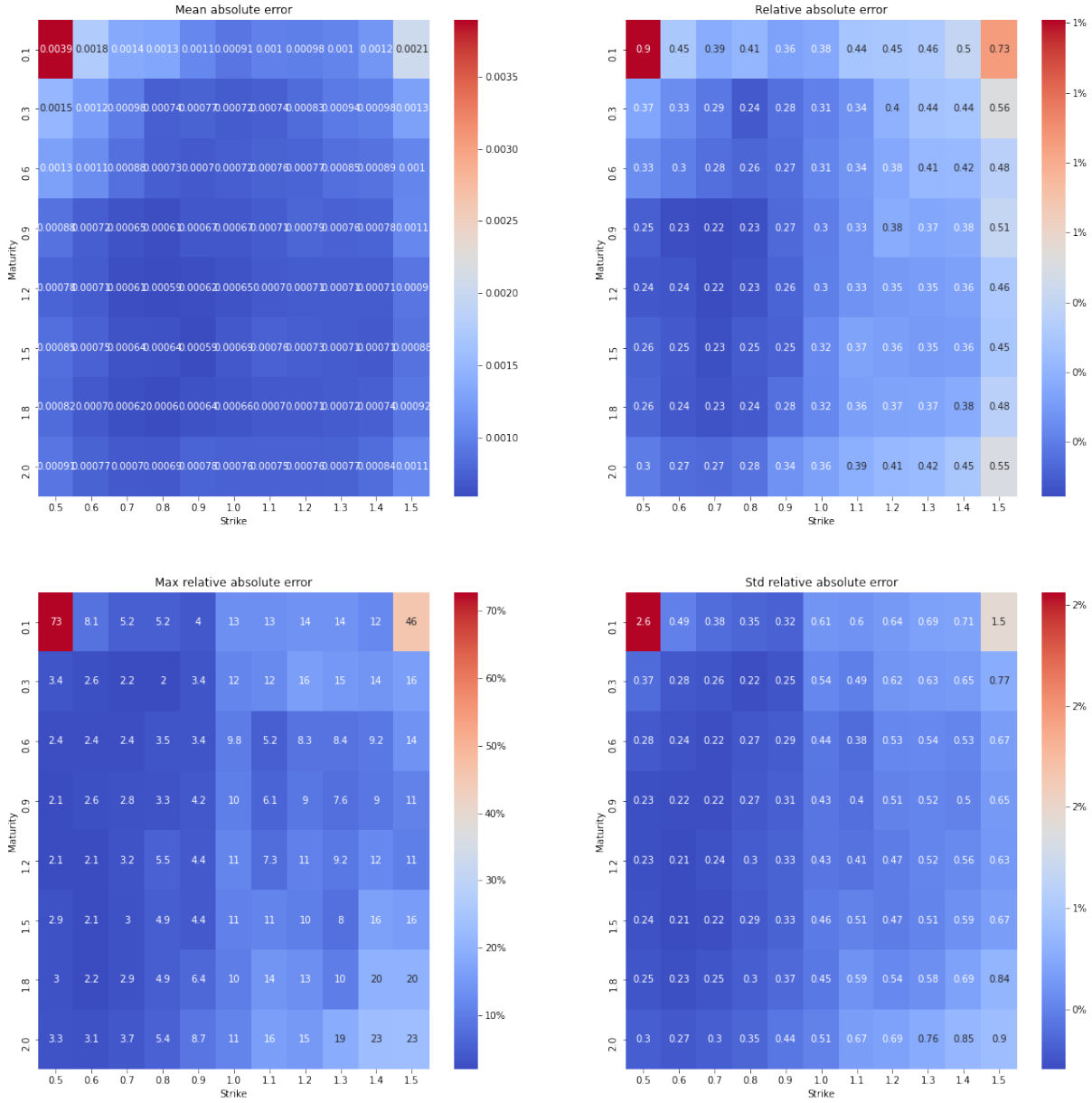


Figure 16: The different errors estimated on the training data for fixed strikes and maturities

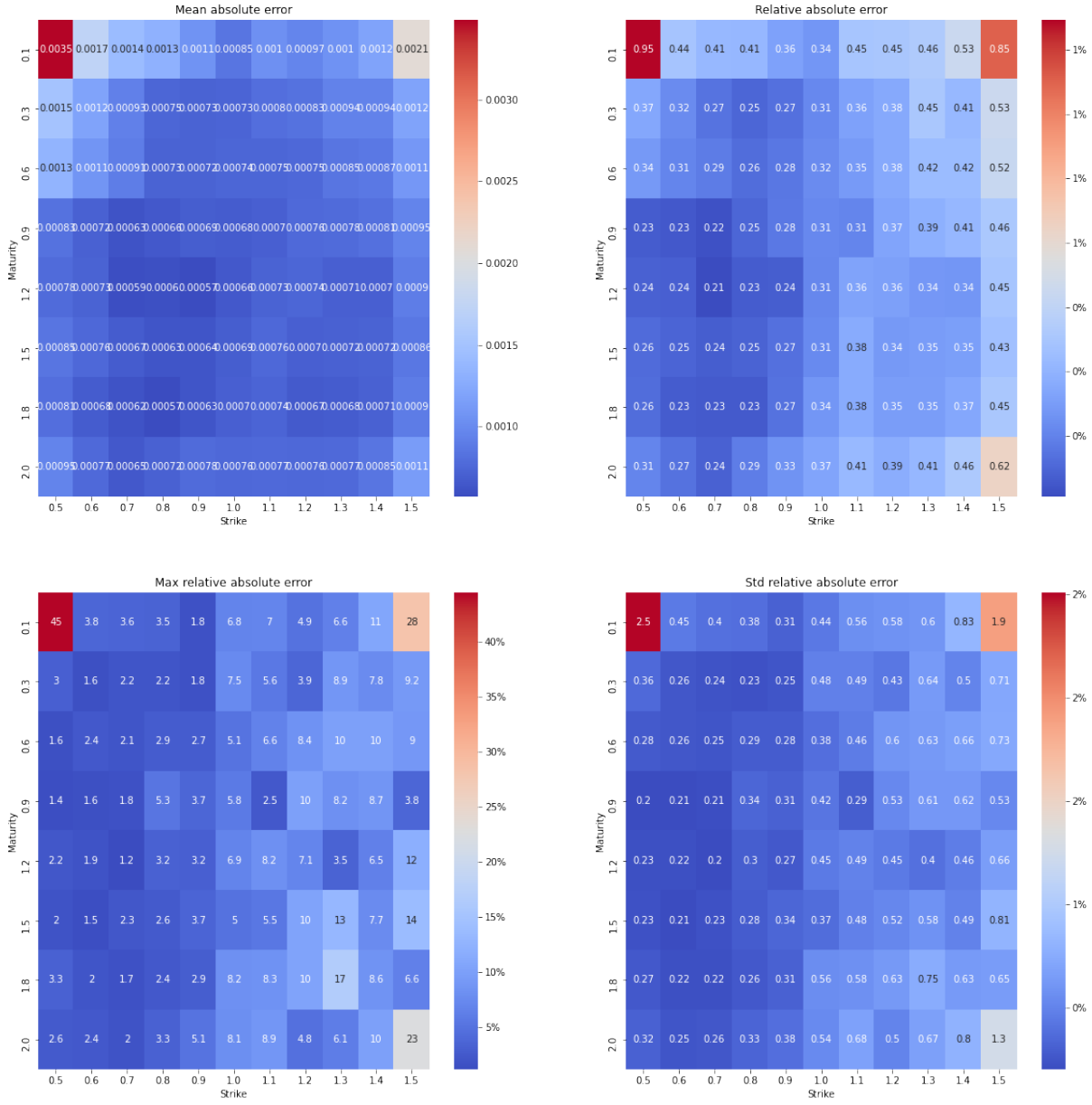


Figure 17: The different errors estimated on the test data for fixed strikes and maturities

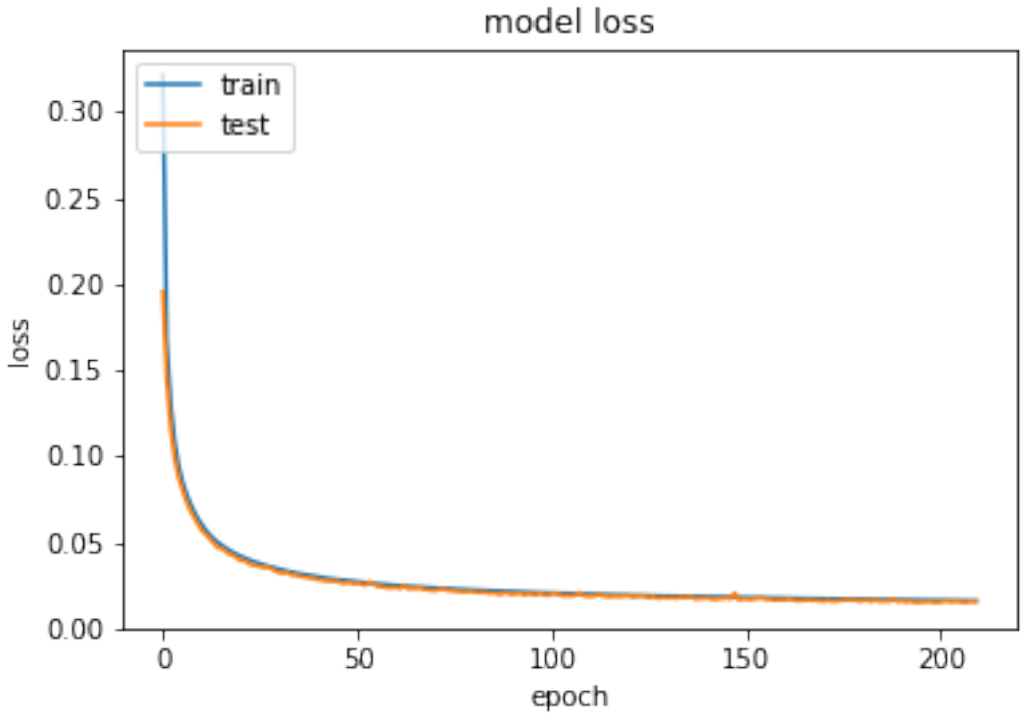


Figure 18: Comparison of losses as training progresses

The results are comparable to the previous ones, except that the training requires more steps to reach the previous score, i.e. more training time, the loss function shows some visible instabilities, this implies that increasing the number of layers does not lead to a better model, It is worth noting that when decreasing the number of layers (layers means sub-layers, i.e. a layer besides the input and output), the model, in the case of a single layer, is unable to reproduce the curve with good accuracy, and fails to capture the complexity of the implicit volatility function, indeed, the loss is 0. 05 in terms of RMSE.

3.6 Impact of the optimizer

There are different types of optimizers, these play on the way the variables are updated at each training step of the model, among the most known: Adam, RMSprop and SGD, we give the result in the case of the use of RMSprop in the model optimization.

$\lambda = 10^{-4}$, RMSprop optimizer, epochs=252, batch size=128, nlayers=2, neurones=2^4, activation=elu, early stopping patience=10

The training time of the model is $13min2s$, several parameters are tested for λ in the case of the RMSprop, we choose $\lambda = 10^{-4}$.

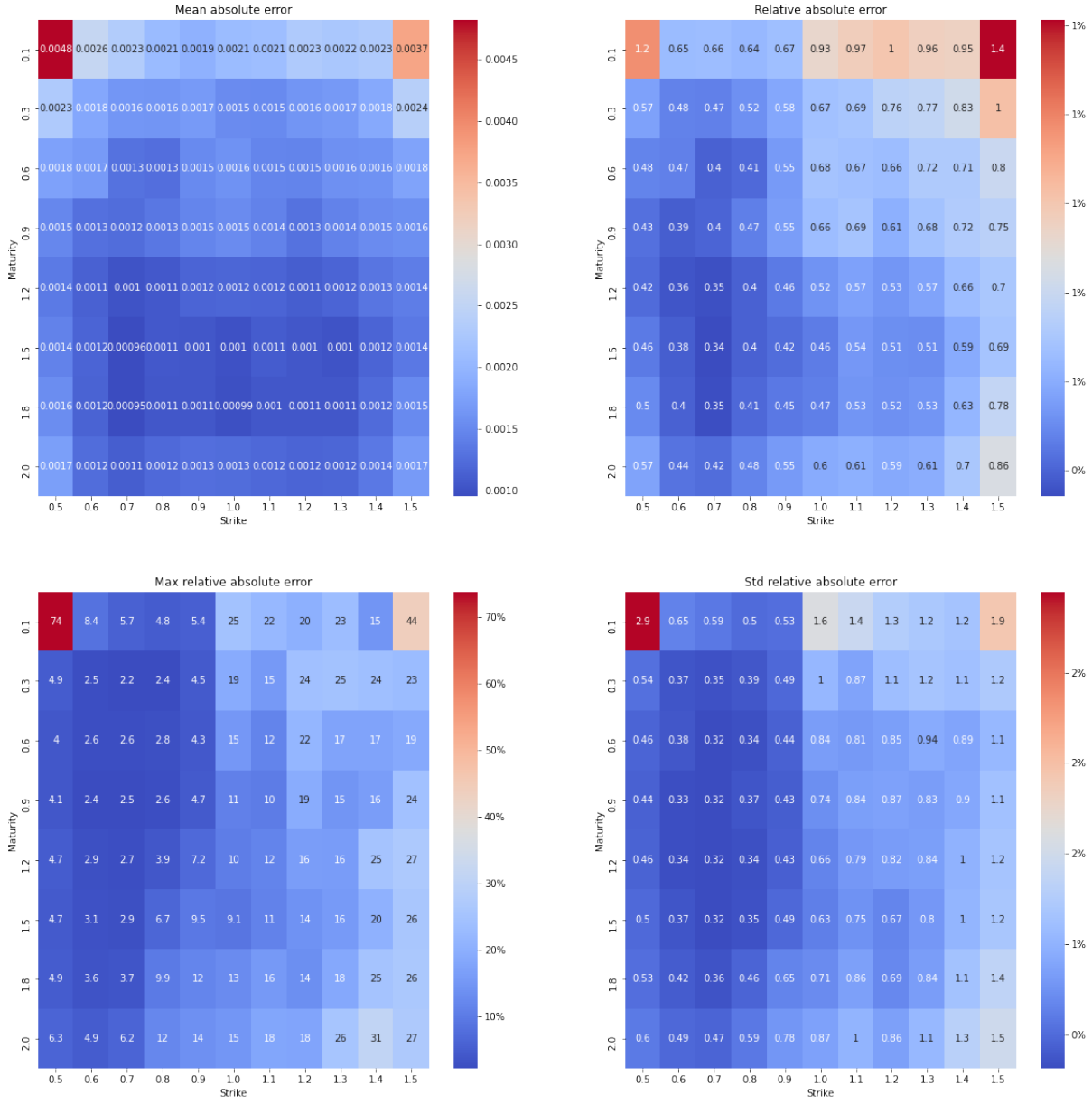


Figure 19: Heatmap for train data

on test data

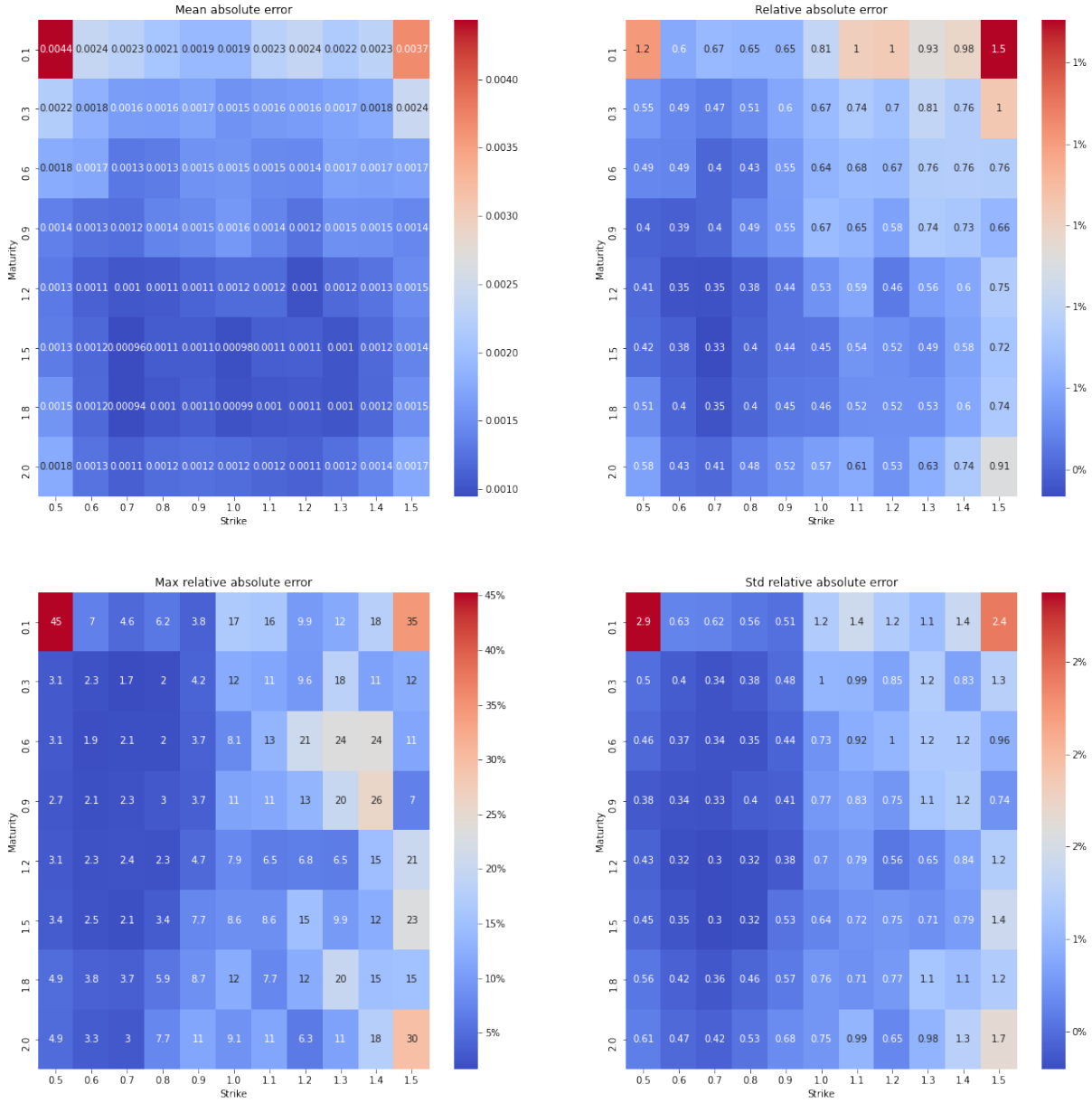


Figure 20: Heatmap for test data

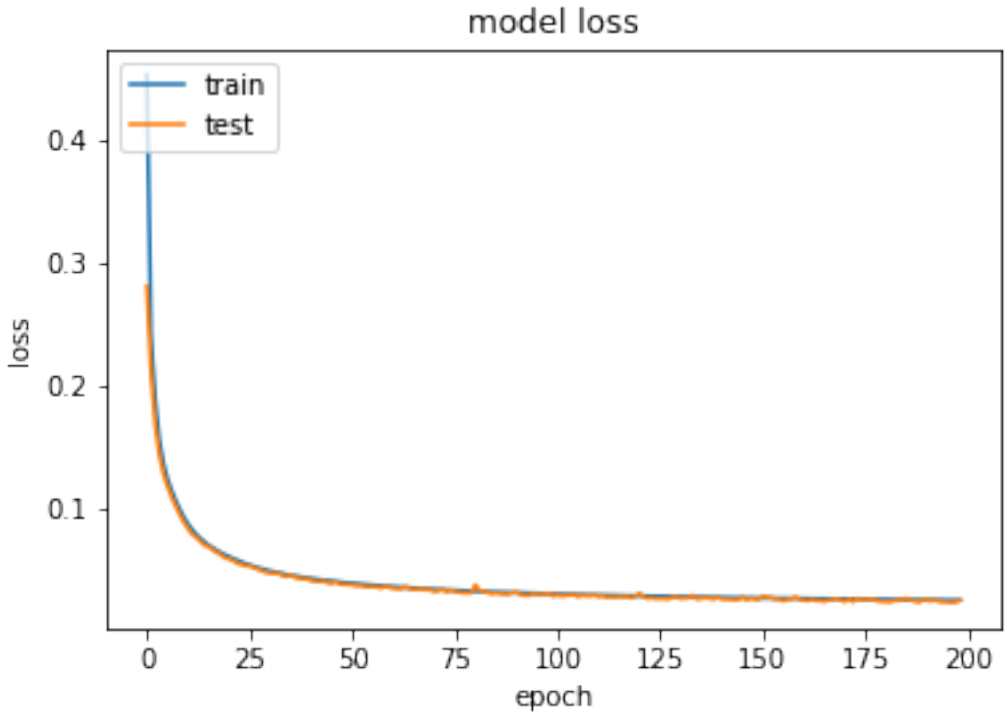


Figure 21: Model error as a function of the number of epochs

On the test data, it is clear that the model provides less accurate results compared to Adam's optimizer, in particular on the figure the relative average error is particularly large at low maturity, the loss curve shows more instability than in the case of Adams' optimization, therefore, we deduce that the latter is a better choice in the present case.

3.7 Impact of the activation function

There are several activation functions that can be chosen, the most used of these methods are: relu, sigmoid (adapted for classification problems), elu. In our case, we choose the impact of using the activation function relu on the quality of the model. It is generally preferable to use the relu function in calibration problems.

The architecture used in the optimization by changing the activation function are the following:
$\lambda = 10^{-4}$, optimizer=Adam, epochs=256, batch size=128, nlayers=2, neurones=2^5, early stopping patience=10.

The optimization time for this model is 13min 55s.

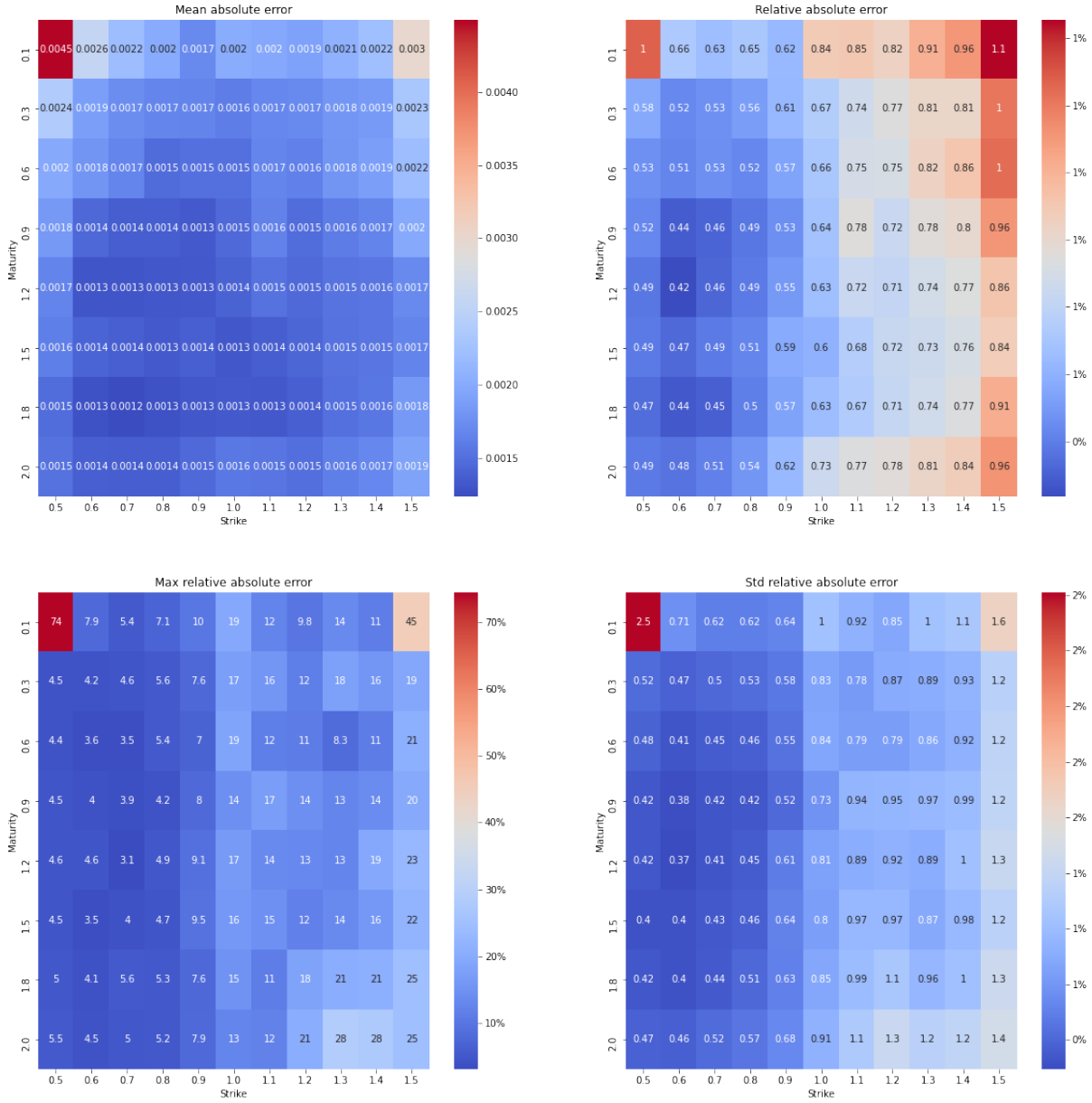


Figure 22: Heatmap for train data

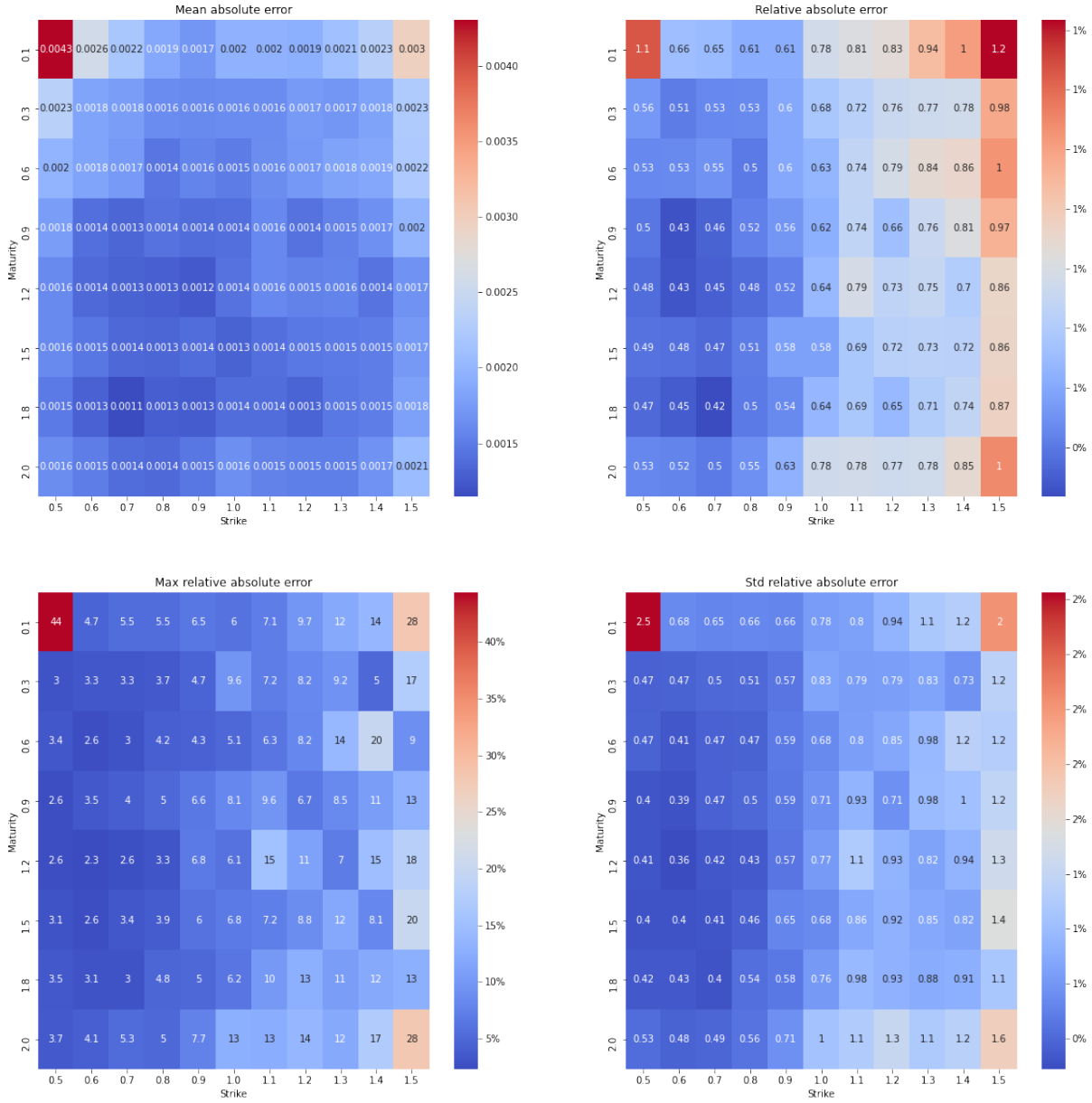


Figure 23: Heatmap for test data

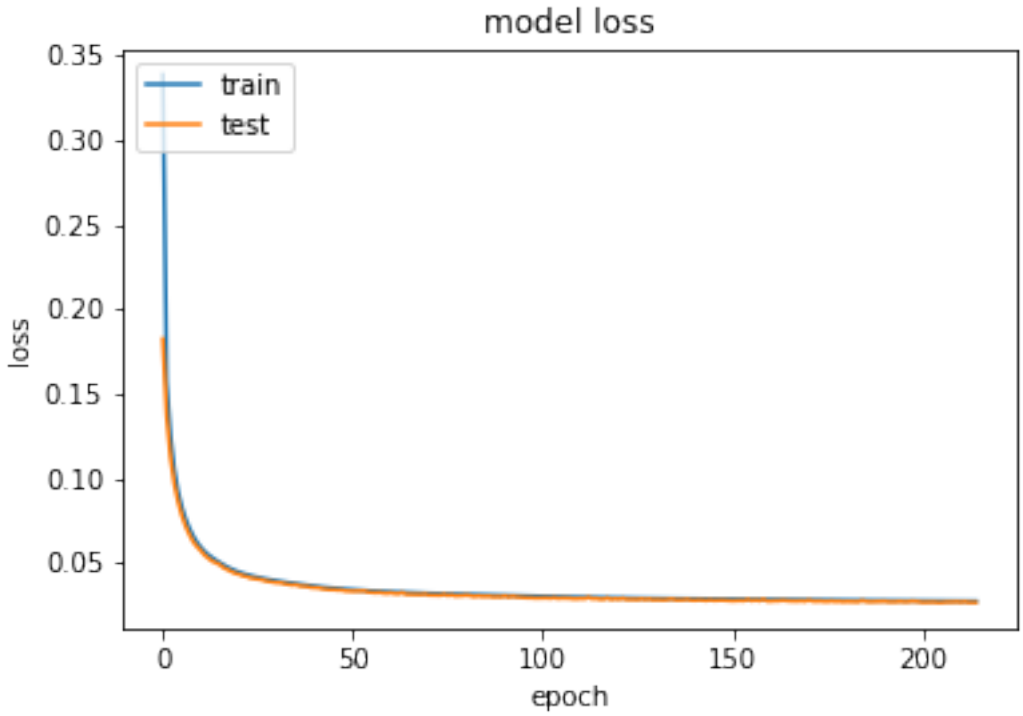


Figure 24: Model error as a function of the number of epochs

As expected, for identical parameters, the use of the elu activation function leads to much more stable results compared to the case of using the relu activation function, which is clearly observed in figure 23, the error in particular the average error is much more pronounced.

3.8 Impact of Batch normalization and Drop Out

The batch normalization method is generally intended to speed up the convergence of the Deep Learning model, but its application in this case introduces instabilities in models with default layer settings, as illustrated below. This is also observed for the application of the Drop Out method in front of the last layer.

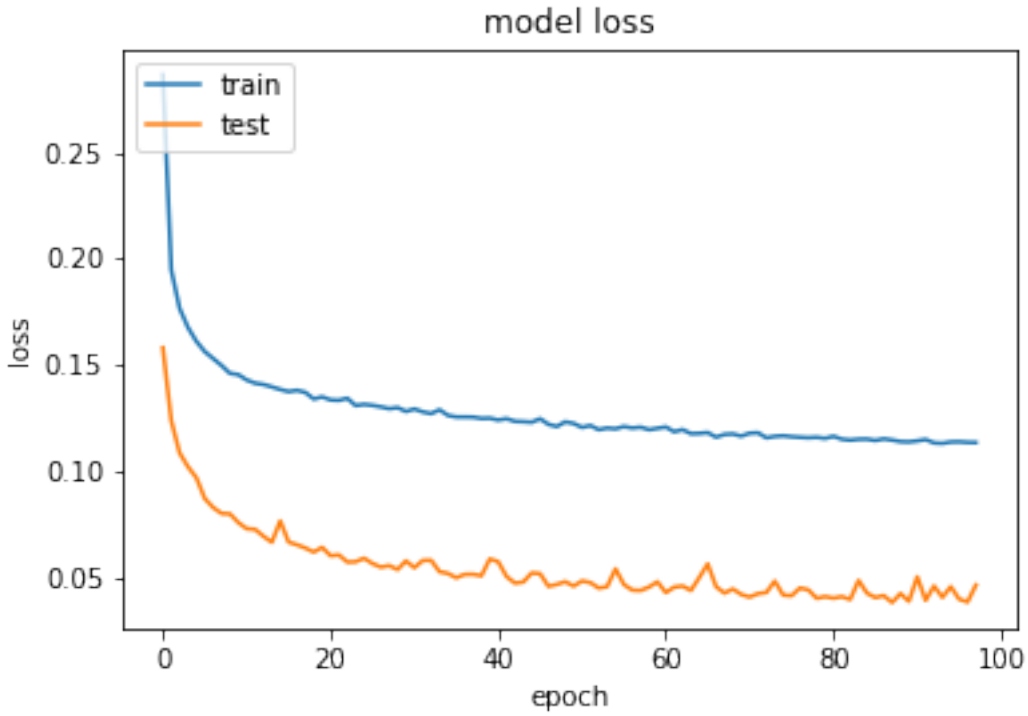


Figure 25: Model error as a function of the number of Batch epochs Normalization

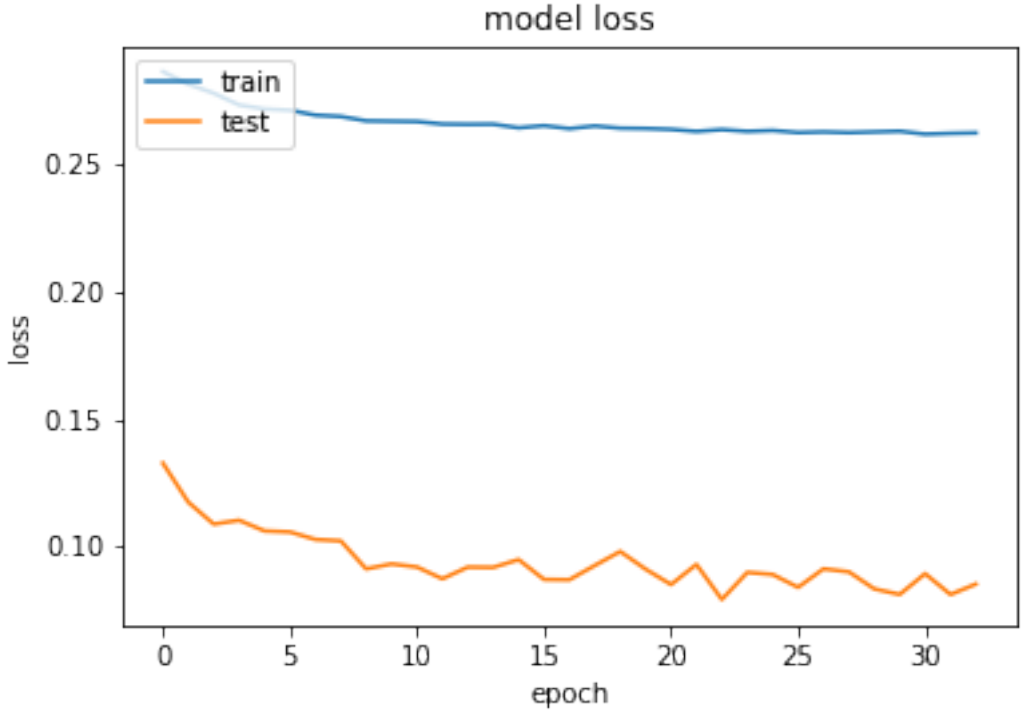


Figure 26: Model error as a function of the number of Early stopping epochs

The learning error is higher than the validation error, which is not usual in Deep

Learning applications, so it is desirable to forgo both methods and focus on optimization by generating more data.

We did not look at the impact of the number of epochs on the quality of the calibration because we opted for early stopping, in fact, the number of epochs is the number of runs on the set of training data, the larger this value is the more the model will be optimized, on the other hand, a large value can lead to an overtraining of the model, we leave the choice of this parameter to the function "callback" Early Stopping.

4 The chosen architecture

The chosen architecture is the one with the following parameters: $Adams(10^{-4})$, epochs=256, batch size=128, $nlayers = 2$, $n_{neurones} = 2^5$,

5 Conclusion

The methodology adopted to tune the parameters proves to be problematic, as a bias is introduced when improving the model score on test data, thus the model may not generalize well when tested on new data, an optimization of this approach would be to do a cross validation allowing to reduce the bias introduced when choosing the parameters on a validation basis. Another improvement similar to the one introduced in the BDSE paper is to generate the data at each step of the training, which will optimize the performance of the chosen models.