Paris Saclay/Paris Evry University

Deep Learning

Deep Calibration

Groupe:

Ouassim Sebbar
Issame Sarroukh

Professeurs:

Sebastien Mollaret
Nicolas Brunel

Contents

1 Introduction

Ce rapport présente les résultats obtenues suite au test de différentes architectures du réseau de neurones dans une problématique de calibration d'un modèle à volatilité stochastique. Pour chaque configuration du réseau, nous affichons une matrice de chaleur de taille maturité×strike exprimant l'erreur du modèle selon 3 approches: l'erreur relative moyenne, l'erreur relative standard et l'erreur relative maximale.

2 Formulation du problème

On s'intéresse à la calibration du modèle à volatilité stochastique suivant:

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

Le modèle met en jeu 4 paramètres:

- σ_0 la volatilité initiale.
- κ la vitesse de retour à la moyenne.
- ν l'écart-type de la volatilité.
- ρ le coefficient de corrélation entre le sous-jacent et la volatilité.

L'objectif est de mettre en place un réseau de neurones qui explique à partir des valeurs des paramètres du modèle (1) la volatilité implicite d'un call européen de strike K et de maturité T.

Pour créer le jeu de données, 5000 réalisations uniformes de $\theta = (\sigma_0, \kappa, \nu, \rho)$ ont été générés sur $\mathbb{R}_+^* \times \mathbb{R}_+^* \times \mathbb{R}_+^* \times [-1, 1]$ et une plage de 8 maturités $\{0.1, 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.0\}$ et 11 strikes $\{0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.1, 1.2, 1.3, 1.4, 1.5\}$ ont été considérés. Ce qui correspond bien à un jeu de données de taille 440 000.

Pour chaque réalisation du tuple $(\sigma_0, \kappa, \nu, \rho)$, on génère les trajectoires du modèle (1) selon les schémas discrétisés suivants:

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

Pour chaque maturité T, on récupère les réalisations du sous-jacent S_T correspondantes et on calcule pour les différents strikes K les prix de call par la méthode Monte Carlo: $\frac{1}{M}\sum_{i=1}^{M} e^{-rT}(S_T^i - K)_+$ où M désigne le nombre de simulations.

Une fois le prix obtenu, on calcule la volatilité implicite correspondante par inversion de la formule de Black&Scholes qui peut être obtenue en utilisant un algorithme de recherche des zéros de la fonction.

Le jeu de données étant prêt, les données simulées sont divisées en ensembles de données d'entraînement et de test respectivement à des proportions de 85% et 15% et sont normalisés.

Dans la section suivante, nous allons nous intéresser à l'élaboration du réseau de neurones pour répondre à la problématique et nous tâcherons à tester plusieurs approches et architectures et analyser leur impact.

3 Test de différentes architectures

Le modèle classique de réseau de neurones est composé d'une couche d'entrée, d'une couche de sortie et de sous-couches reliant l'entrée à la sortie, les relations entre les neurones de deux couches consécutives sont linéaires avec ensuite l'application d'une fonction d'activation. L'algorithme d'optimisation du réseau est basé sur la descente de gradient pour estimer les paramètres du réseau neuronal. Dans ce qui suit, nous examinons l'influence des différents paramètres associés l'architecture du réseau de neurones sur la qualité du modèle dans le contexte de la calibration.

L'entraînement est effectué initialement sur une machine virtuelle Google Cloud avec un GPU Tesla P100, puis sur une machine locale avec un GPU GTX 1070. Pour que les temps soient comparables, nous avons privilégié l'utilisation de la machine locale.

3.1 Architecture par défaut

L'architecture utilisée est celle que l'on trouve dans le notebook, sauf que nous y ajoutons l'arrêt anticipé.

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

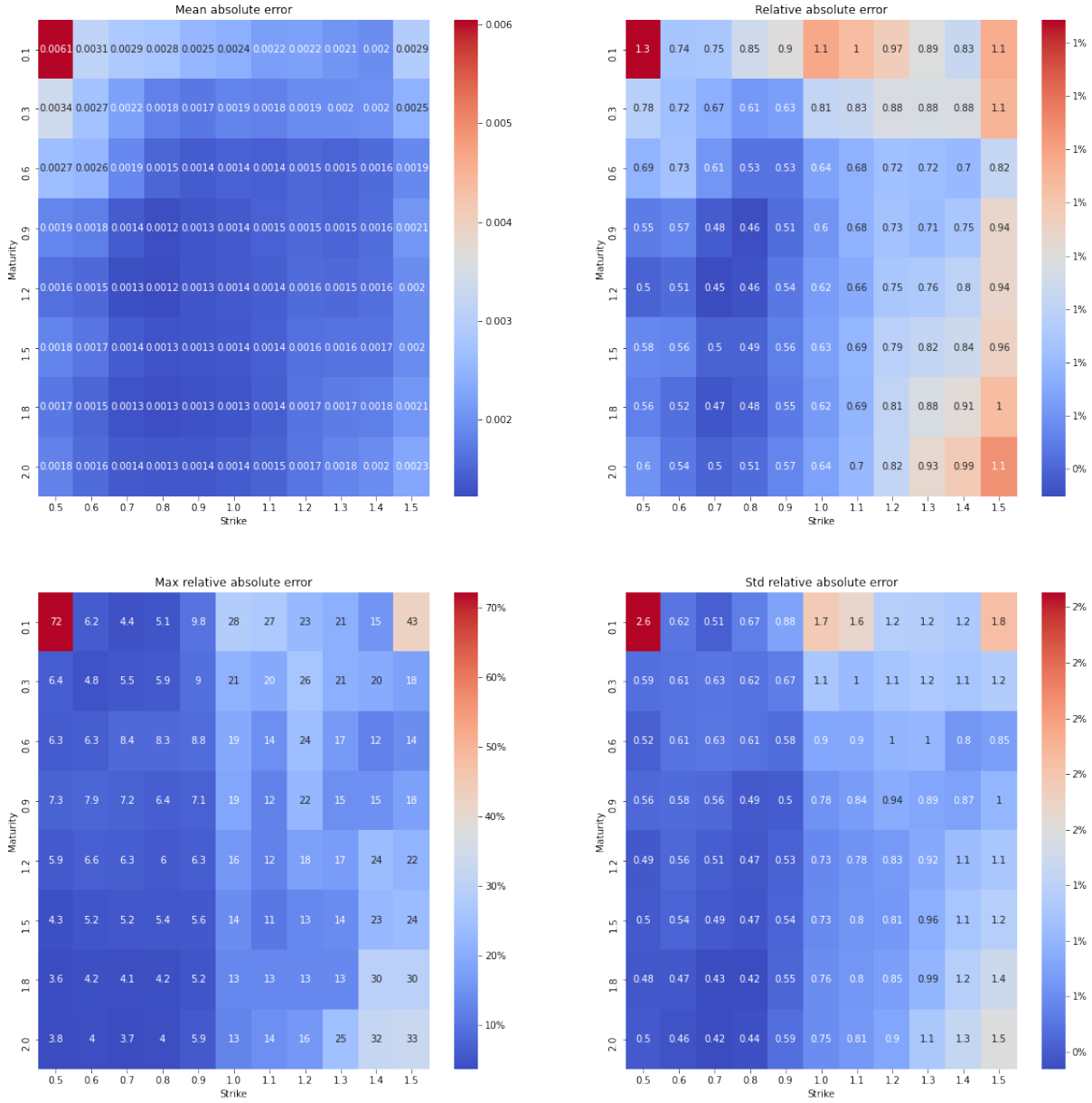

Figure 1: Heatmap pour les données train

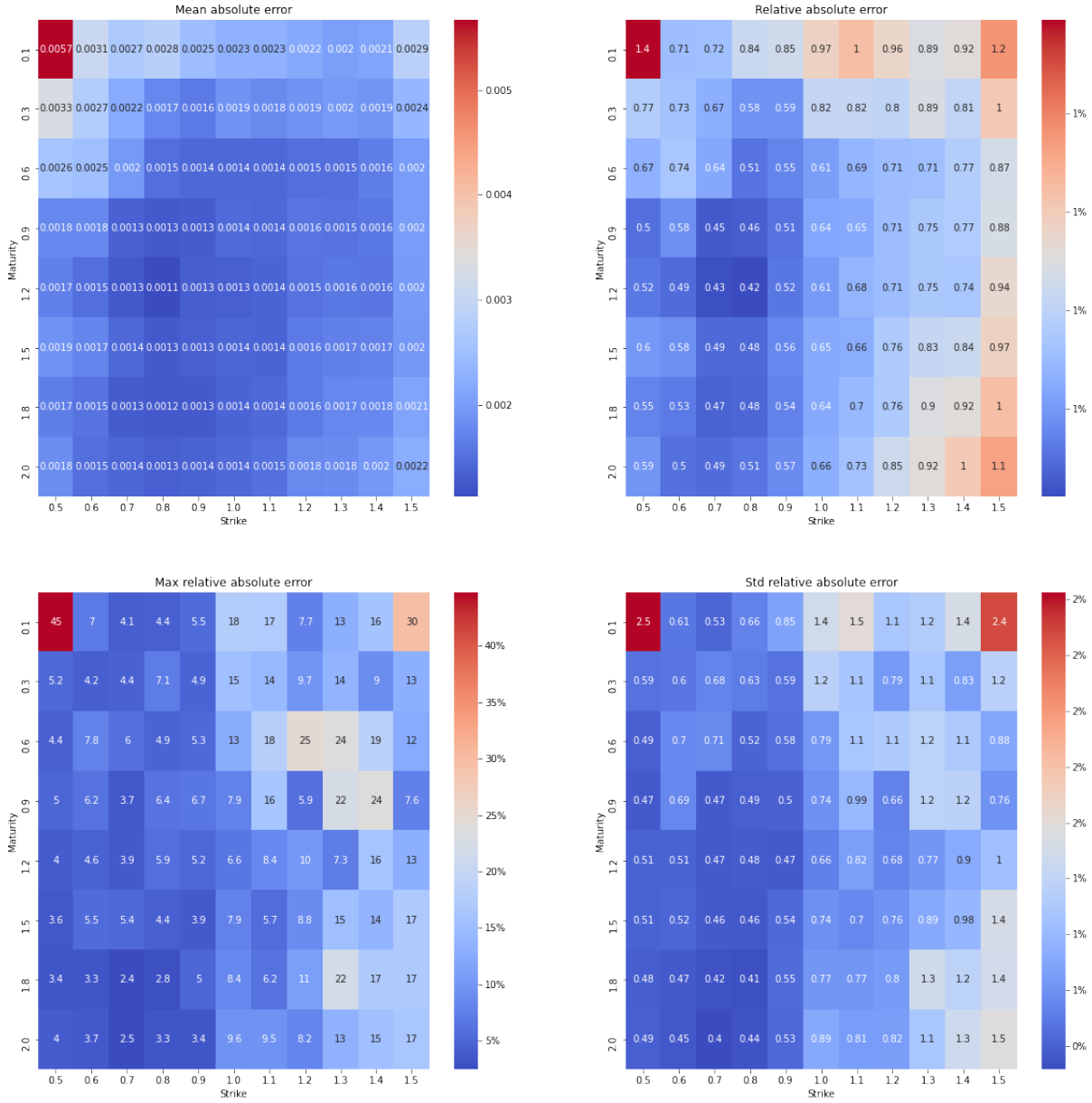

Figure 2: Heatmap pour les données test

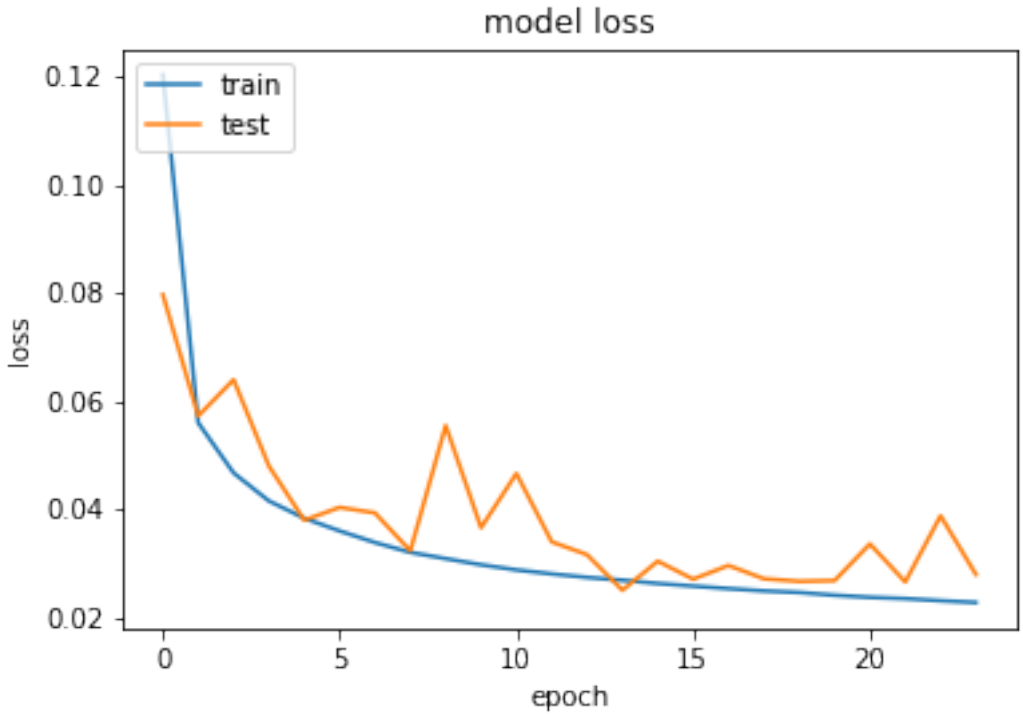

Figure 3: Erreur du modèle en fonction du nombre d'epochs

Remarquant que dans la figure 3, l'erreur de validation est instable, et que la convergence du modèle est d'environ 13min, nous allons examiner par la suite l'effet du changement des paramètres du modèle sur sa convergence et sur la qualité de la calibration.

3.2 Impact du learning rate

Dans cette partie, on cherche à quantifier l'impact du taux d'apprentissage (learning rate). On garde la même architecture précédente en variant uniquement la valeur de λ qu'on prend cette fois-ci égale à 10^{-4}. C'est-à-dire qu'on intialise un réseau de neurones avec la paramétrisation suivante:

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

Le réseau de neurones a pris un temps de 26min et 32s pour son entraînement à base des données d'apprentissage.

On peut visualiser dans la suite les résultats de prédiction de notre modèle sous forme de matrice de chaleur (heatmap) sur les données train (4) et également sur les données test (5):

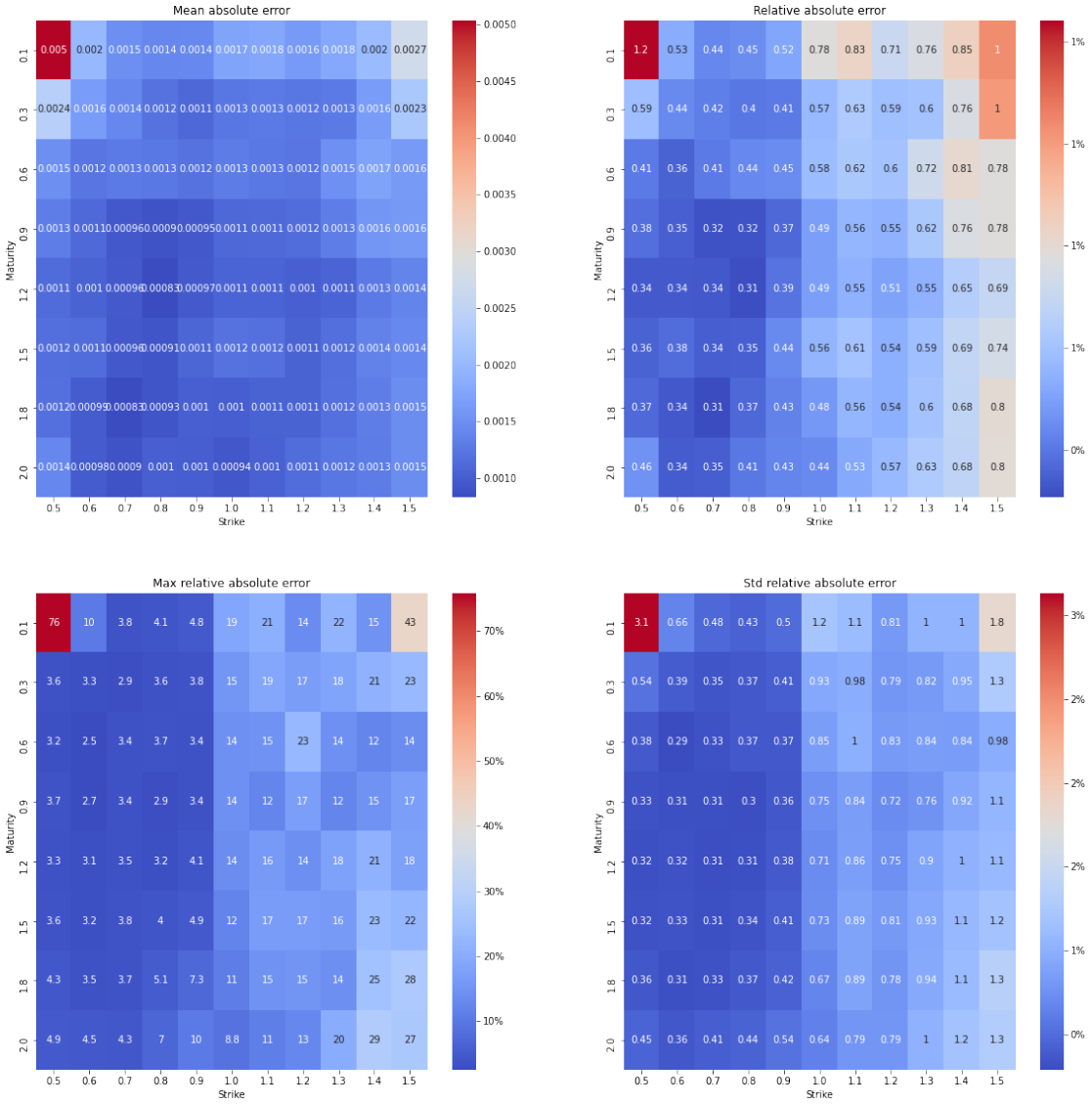

Figure 4: Heatmap pour les données train pour $\lambda = 10^{-4}$

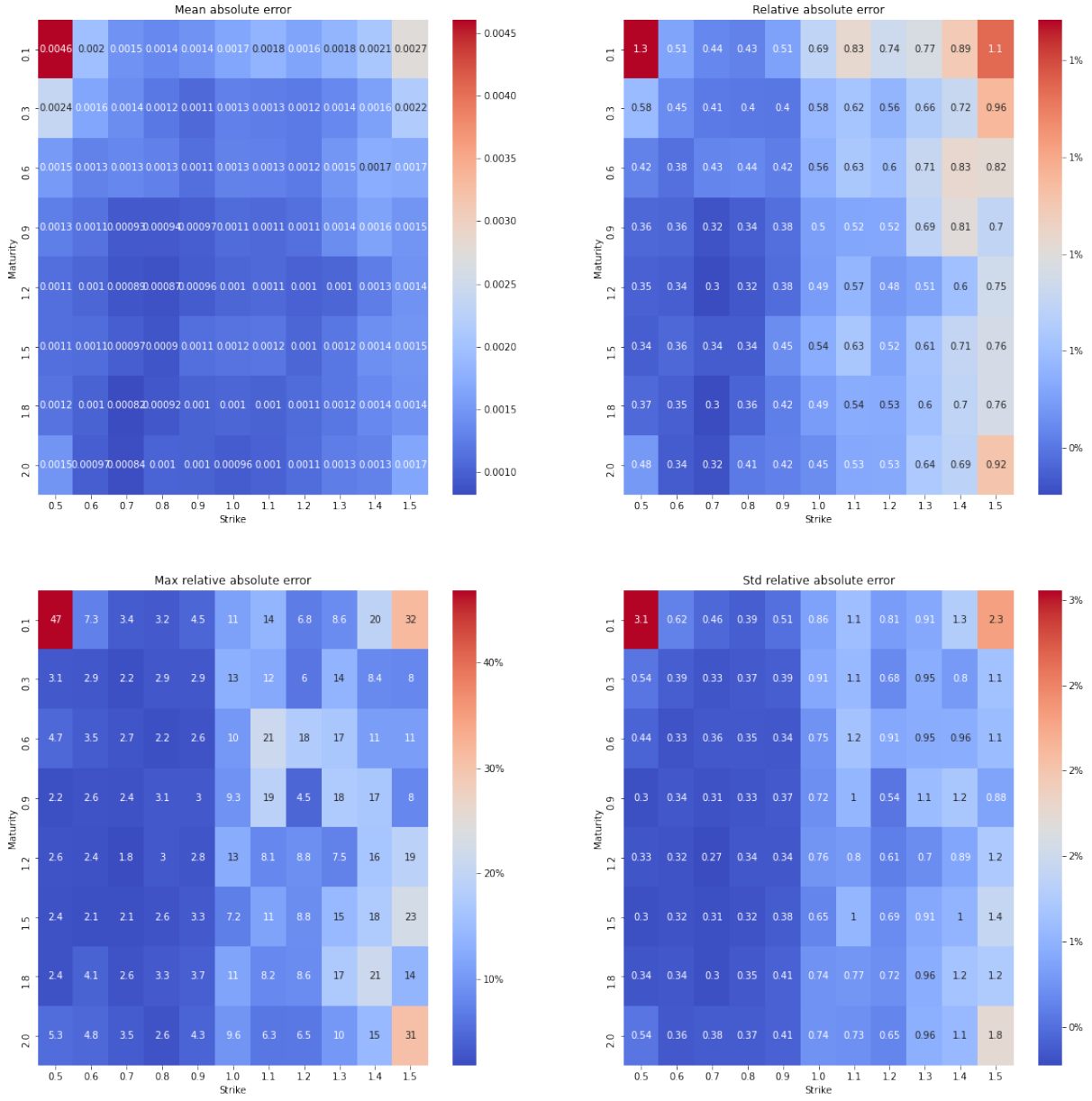

Figure 5: Heatmap pour les données test pour $\lambda = 10^{-4}$

Les figures 4 et 5 montrent que l'erreur relative moyenne (pour toutes les combinaisons de paramètres) entre les volatilités prédites par le réseau de neurones et les volatilités obtenues par la méthode Monte Carlo est nettement inférieur 1% (image de droite en haut en figures 4 et 5). L'erreur relative maximale va jusqu'à 76% pour les données train et 47% pour les données test. Pour tous types d'erreur observés, le modèle enregistre à

chaque fois un maximum d'erreur pour la maturité 0.1 et le strike 0.5 et reste performant pour le calcul des volatilités proches de la monnaie où il enregistre le moins d'erreur.

Dans la suite, on trace l'erreur du modèle comme fonction du nombre d'epochs:

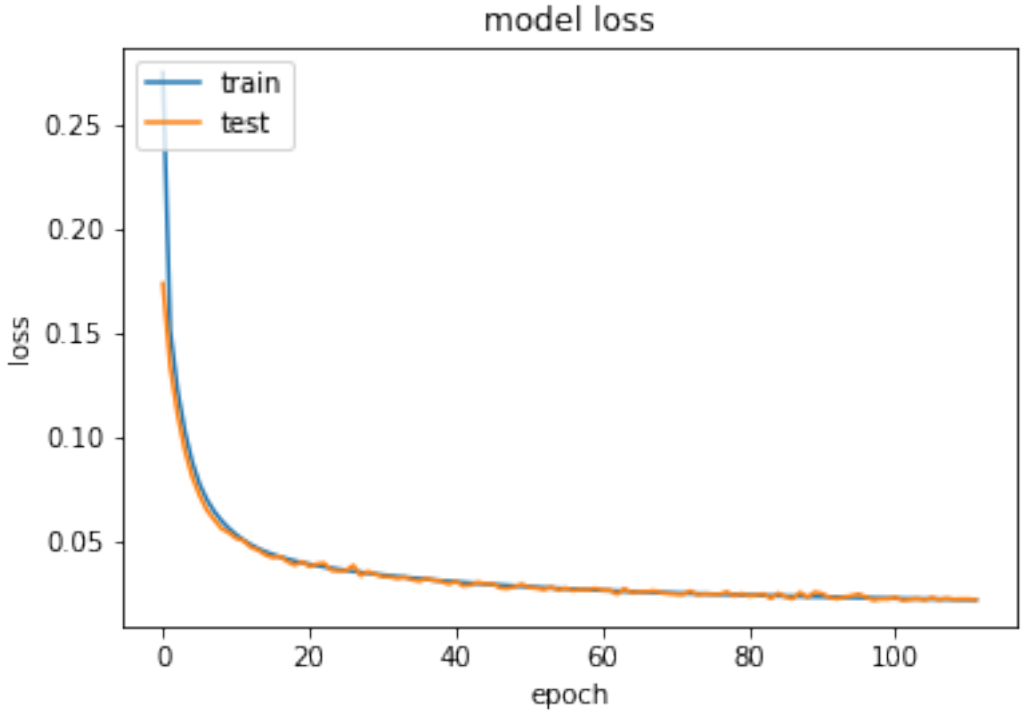

Figure 6: Erreur du modèle en fonction du nombre d'epochs

On voit bien que l'erreur du réseau de neurones reste invariante entre les données train et les données test, ce qui prouve que le modèle généralise bien. Comparé à la fonction erreur de la section précédente, on remarque une grande stabilité de plus. Ceci est dû certainement au fait de la diminution du taux d'apprentissage dans la configuration actuelle qui a permis au modèle de gagner en stabilité et convergence mais en contrepartie a allongé considérablement le processus d'entraînement.

3.3 Impact du Batch size

Dans cette partie, on étudie l'influence du Batch size en adoptant la configuration suivante du réseau de neurones:

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

Le réseau de neurones a pris un temps de 17min et 22s pour son entraînement à base des données d'apprentissage.

On peut visualiser dans la suite les résultats de prédiction de notre modèle sous forme de matrice de chaleur (heatmap) sur les données train (7) et également sur les données test (8):

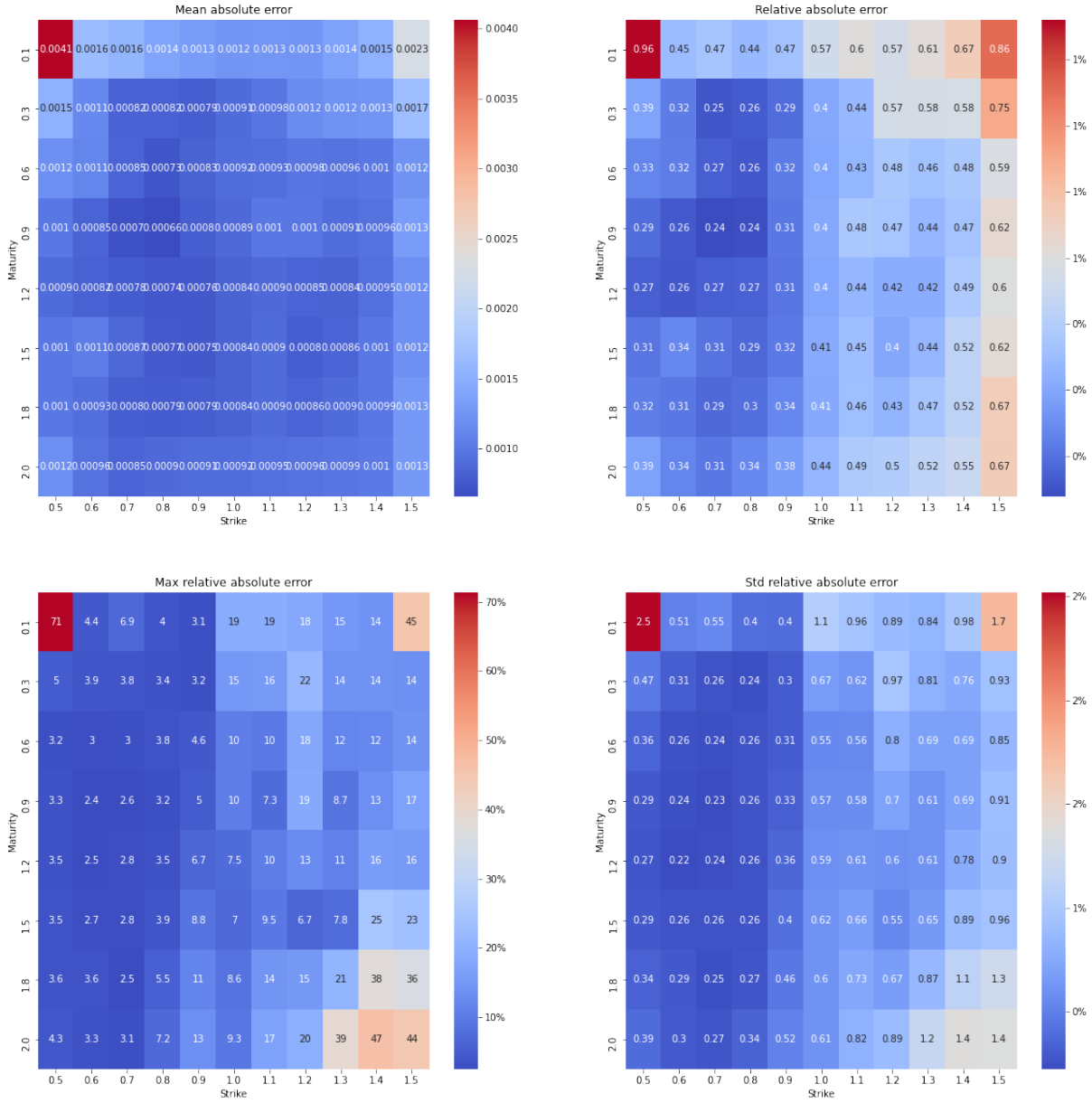

Figure 7: Heatmap pour les données train pour batch size = 128

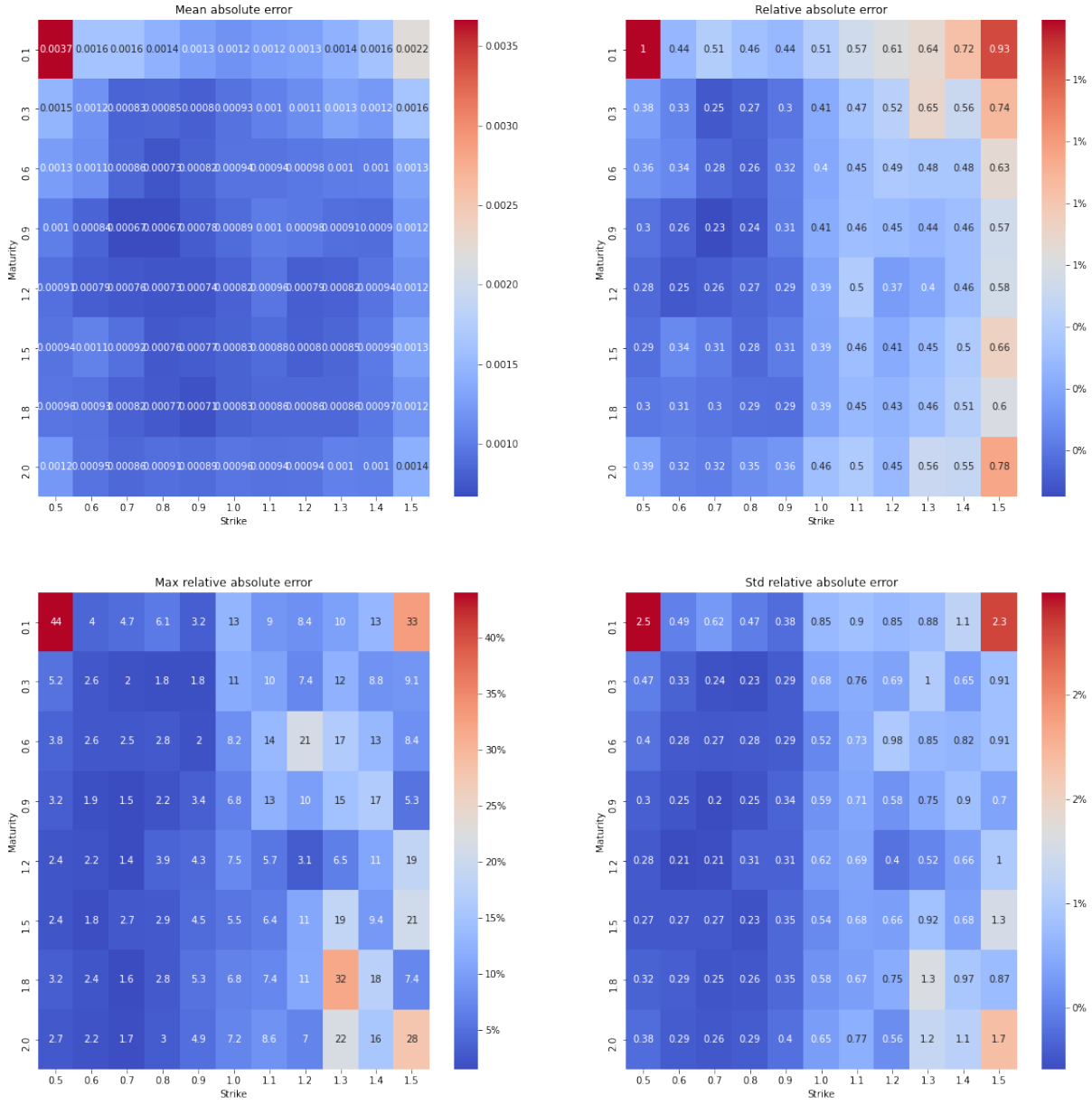

Figure 8: Heatmap pour les données test pour batch size = 128

L'erreur relative moyenne (pour toutes les combinaisons de paramètres) d'après les figures 7 et 8 est nettement inférieur 1%. Tandis que l'erreur relative standard atteint son maximun de 2.5% à la fois pour les données train et test encore une fois pour la maturité 0.1 et le strike 0.5 (image de droite en bas en figures 7 et 8). L'erreur relative maximale va jusqu'à 71% pour les données train et 44% pour les données test.

Dans la suite, on visualise l'évolution de l'erreur du modèle comme fonction du nombre d'epochs:

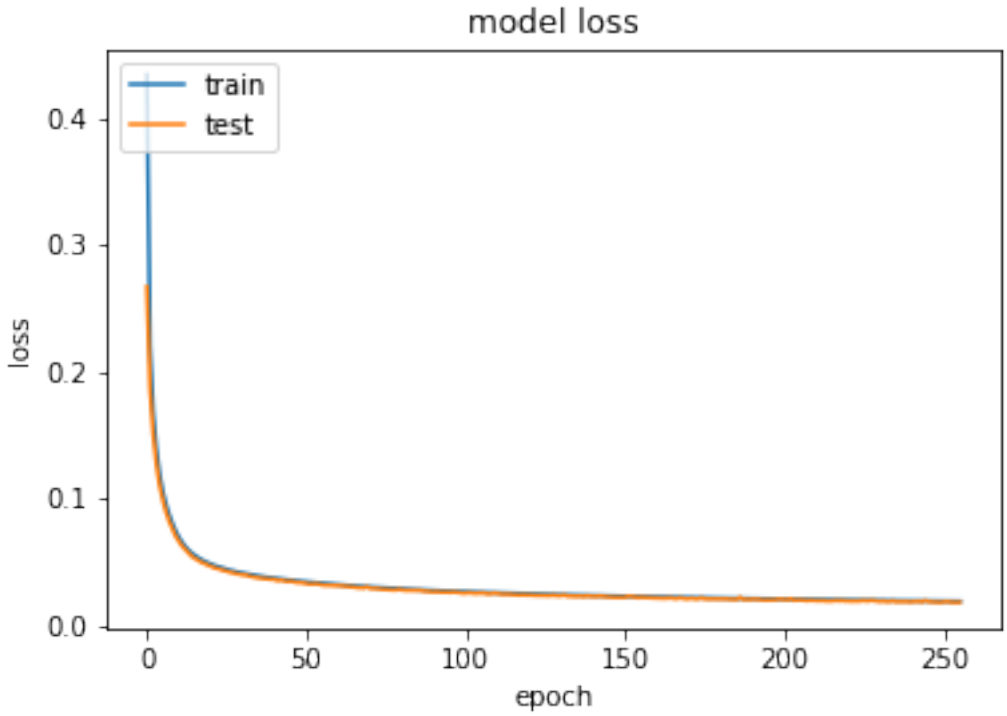

Figure 9: Evolution de l'erreur du modèle en fonction du nombres d'epochs

On voit que l'erreur du modèle est stable à la fois sur les données train et les données test. Une grande valeur du batch size augmente la précision de l'estimation du modèle et réduit le temps d'apprentissage (17min et 22s) en assurant une convergence plus rapide.

3.4 Impact du Nombre du neurones

Dans cette section, on adopte la même paramétrisation précédente en faisant baisser cette fois-ci le nombre de neurones à 16. La configuration du modèle est donc la suivante:

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

Le réseau de neurones a pris un temps de 15min et 59s pour son entraînement à base des données d'apprentissage.

On peut visualiser dans la suite les résultats de prédiction de notre modèle sous forme de matrice de chaleur (heatmap) sur les données train (10) et également sur les données test (11):

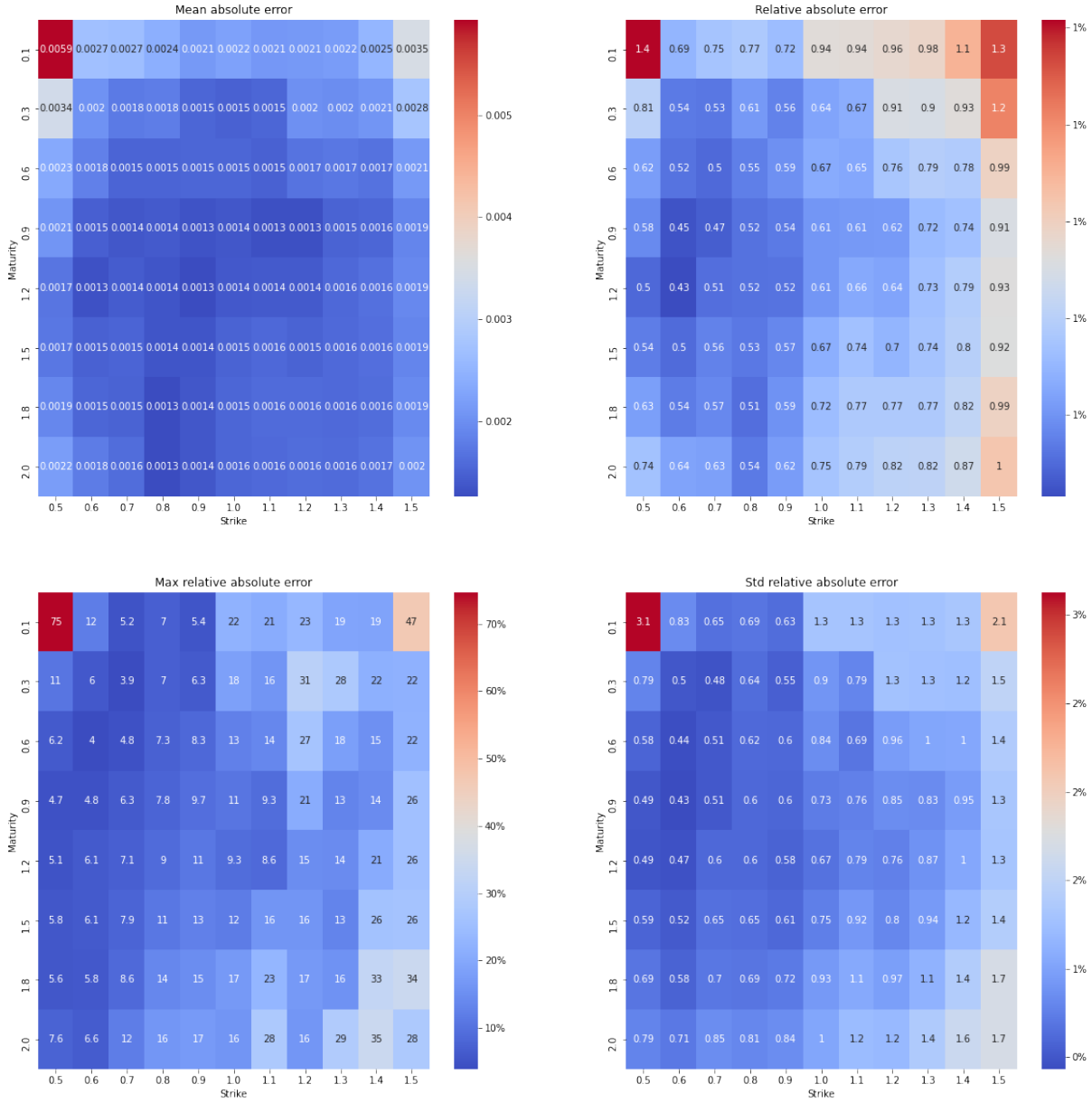

Figure 10: Heatmap pour les données train pour un nombre de neurones = 16

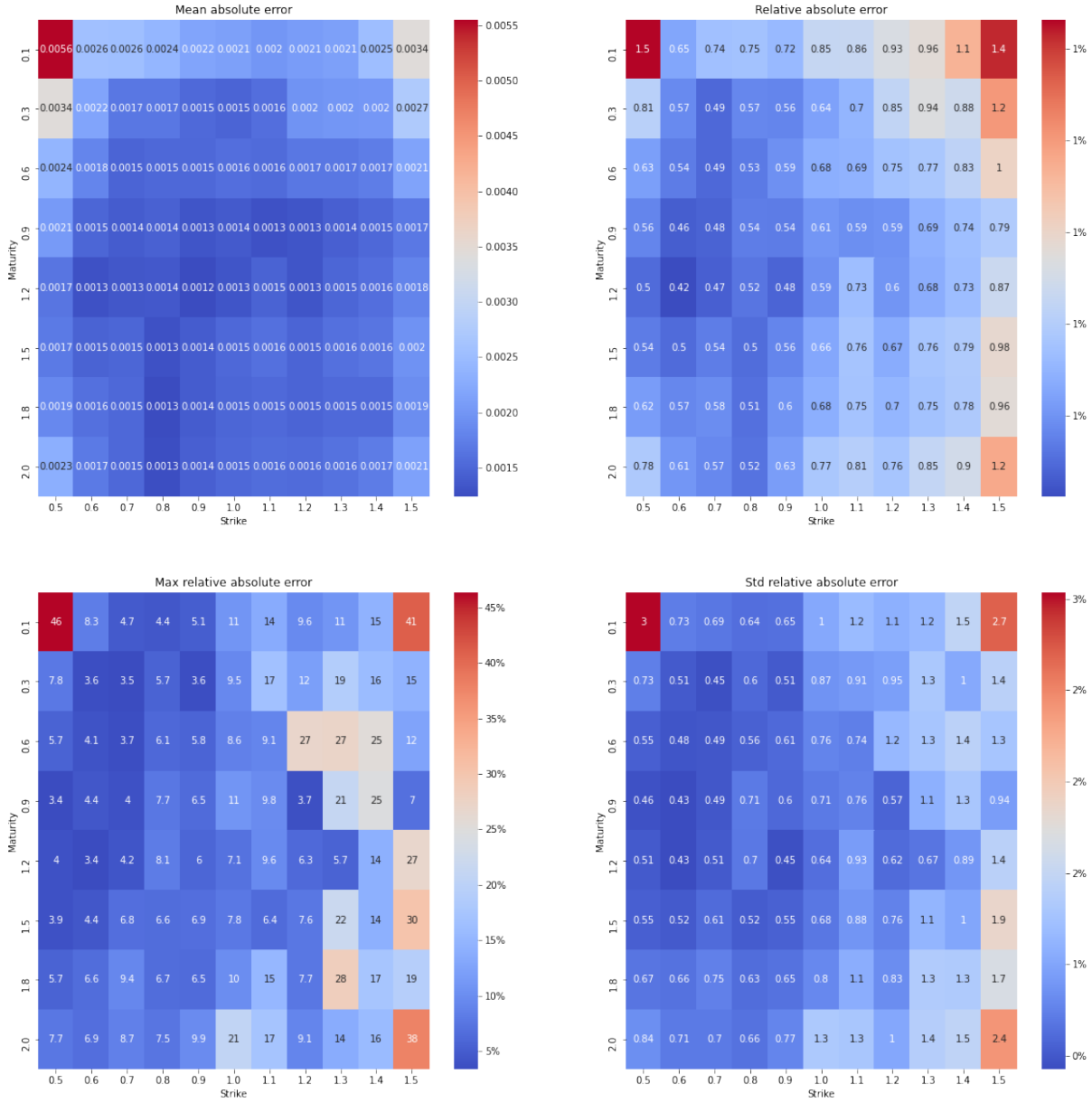

Figure 11: Heatmap pour les données test pour un nombre de neurones = 16

L'erreur relative moyenne d'après les figures 10 et 11 est pour tout strike et maturité inférieur à 1.5%. Tandis que l'erreur relative maximale va jusqu'à 75% pour les données train et 46% pour les données test(image de gauche en bas en figures 10 et 11). Quant à l'erreur relative standard ne dépasse pas 3% pour les échantillons train et test.

Dans la suite, on augmente le nombre de neurones à 64. La configuration du modèle est donc la suivante:

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

Les résultats de prédiction de notre modèle sous forme de matrice de chaleur (heatmap) sur les données train et également sur les données test sont visibles respectivement sur les figures (12) et (13):

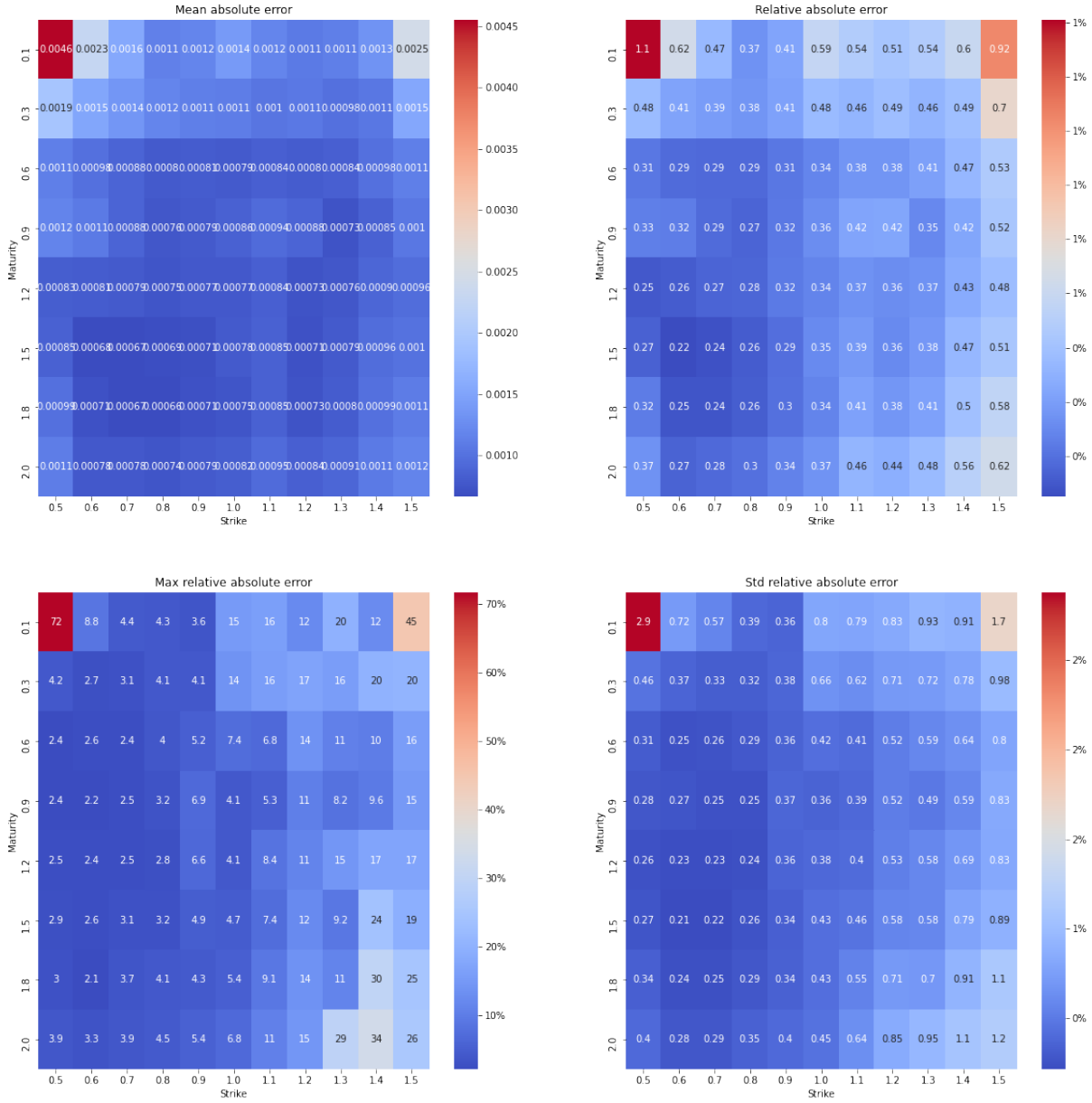

Figure 12: Heatmap pour les données train pour un nombre de neurones = 64

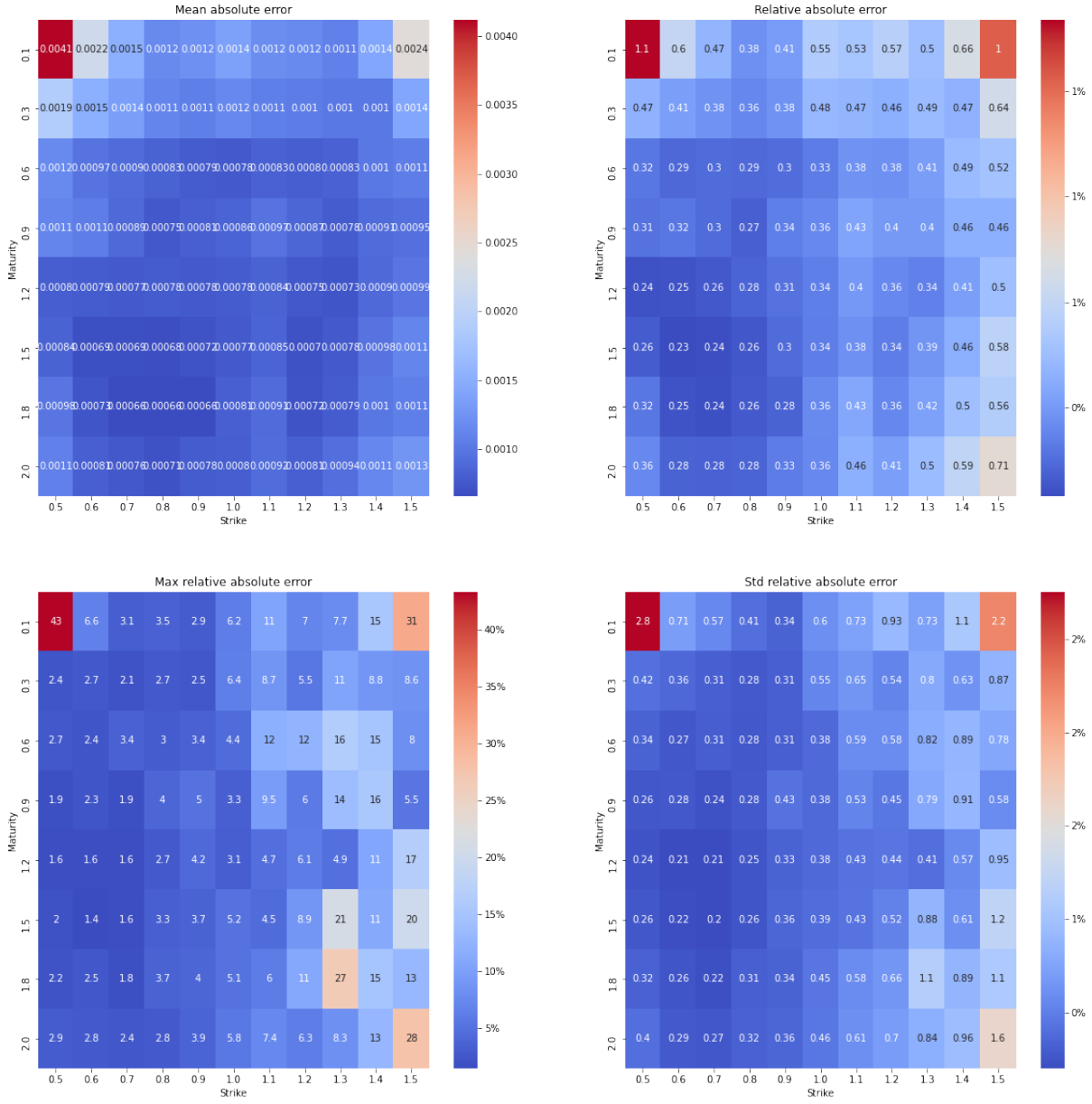

Figure 13: Heatmap pour les données test pour un nombre de neurones = 64

L'erreur relative moyenne d'après les figures 12 et 13 est pour tout strike et maturité inférieur à 1.1% (image de droite en haut en figures 12 et 13. Tandis que l'erreur relative maximale va jusqu'à 2% pour les données train et 43% pour les données test, atteinte dans la maturité 0.1 et strike 0.5. Quant à l'erreur relative standard ne dépasse pas 2.9% pour les échantillons train et test.

Les figures suivantes montre l'évolution de l'erreur en fonction du nombre d'epochs pour les 2 cas:

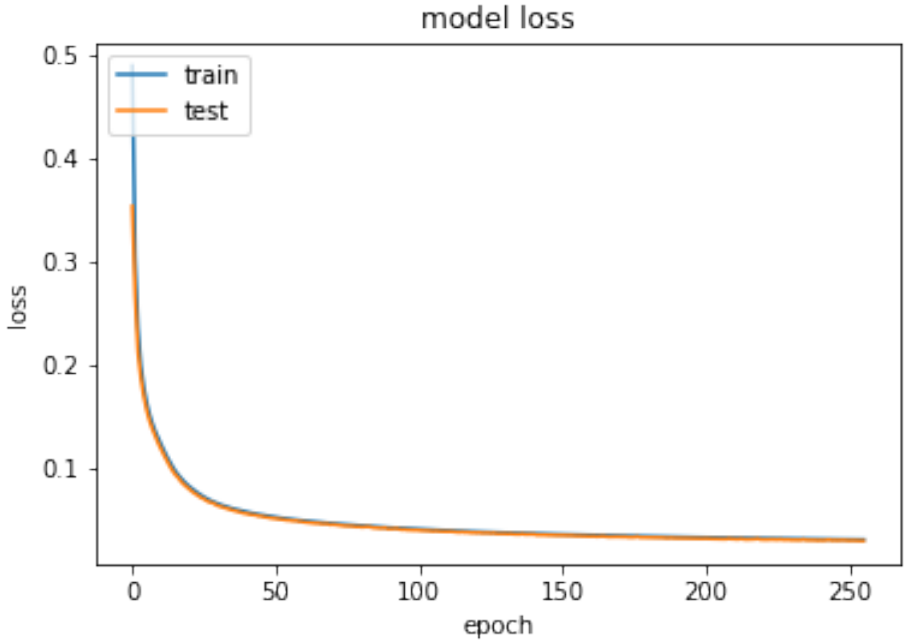

Figure 14: Evolution de l'erreur du modèle en fonction du nombres d'epochs pour un nombre de neurones = 16

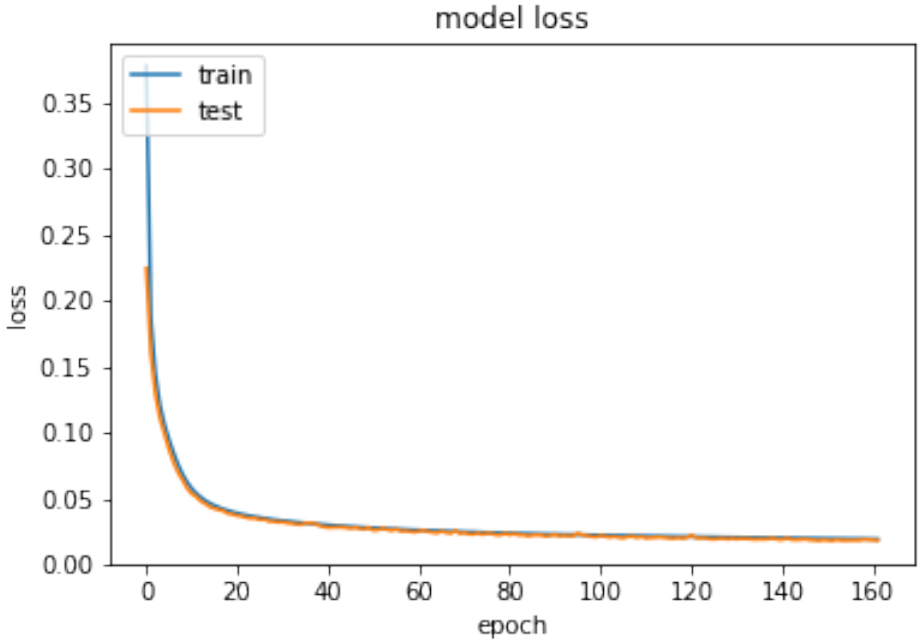

Figure 15: Evolution de l'erreur du modèle en fonction du nombres d'epochs pour un nombre de neurones = 64

D'après les deux figure précédentes, on peut déduire que le nombre de neurones a un impact signifiant sur la qualité de prédiction du modèle. Plus on augmente le nombre de neurones, plus la précision du modèle croît quant à l'estimation des données d'entraînement mais avec un risque d'over-fitting et perte de la généralisation du modèle. Dans ce cas, la phase d'apprentissage du modèle prend un temps considérable tandis que pour un nombre de neurones beaucoup moins, ce temps n'est pas trop important cependant on perd au niveau de la précision du modèle (voir 14).

3.5 Impact du nombre des layers

Les paramètres choisis sont ceux donnant la meilleur optimisation, avec un taux d'entraînement $\lambda = 10^{-4}$, l'optimizer adam optimizer, et un nombre d'epochs égale 256, et un batch size=128, le nombre de layers a été augmenté et égale à 3, le nombre de neurone est fixé à 2^5, la stratégie d'arrêt anticipé est toujours utilisée, avec un arrêt d'entraînement après 10 étapes de non amélioration du modèle, le temps estimé de l'entraînement est de 15min 51s avec la fonction d'activation elu.

Le nombre de couches (layers) permet d'ajouter un paramétrage en plus au réseau de neurones, donc permet globalement d'approcher plus précisément la volatilité qu'on essaie d'estimer, il est à noter que cela à généralement un impacte sur le temps d'apprentissage, et peut induire des problèmes comme l'overfitting.

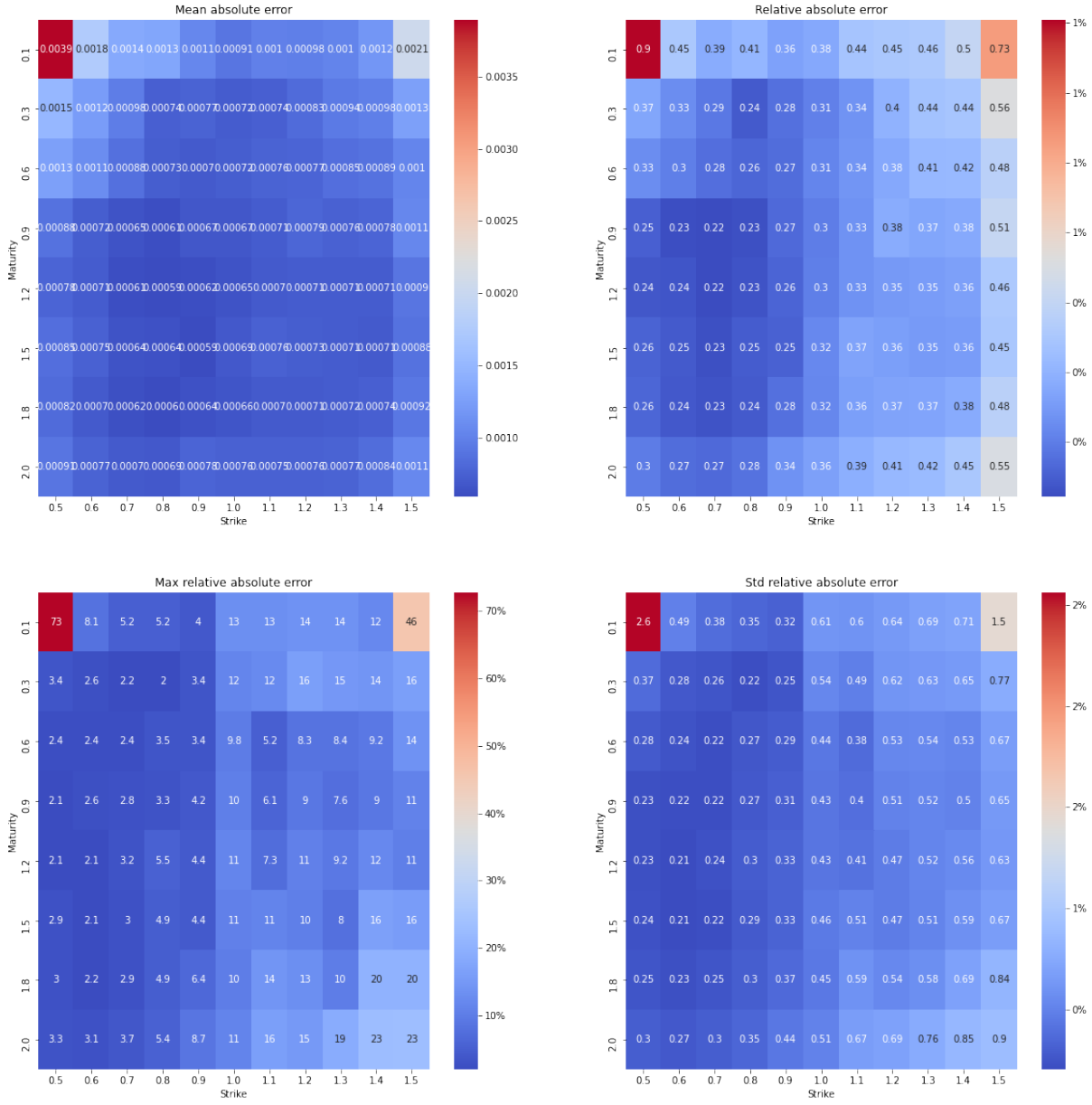

Figure 16: Les différentes erreurs estimées sur les données d'entraînement pour des strikes et des maturités fixes

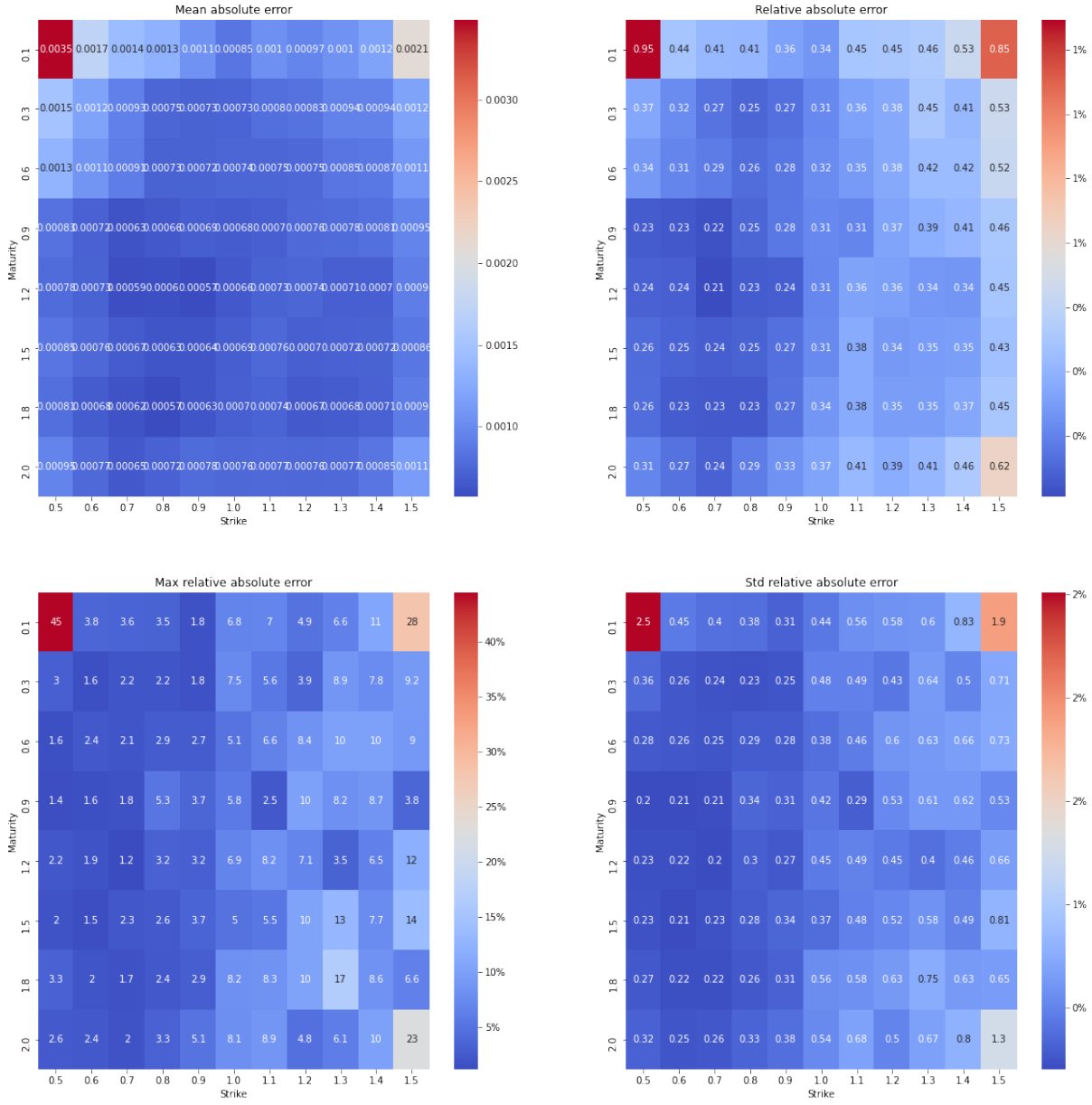

Figure 17: Les différentes erreurs estimées sur les données de test pour des strikes et des maturités fixes

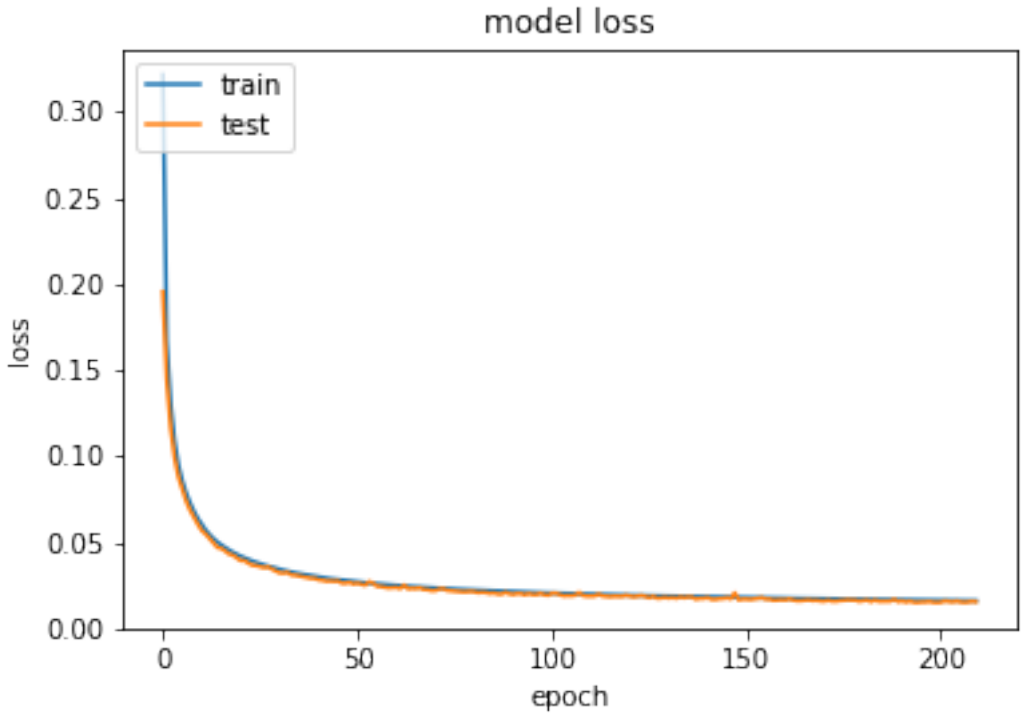

Figure 18: Comparaison des pertes au fur et à mesure de l'entraînement

Les résultats sont comparables aux précédentes, sauf que l'entraînement demande plus d'étapes pour arriver au score précédente soit plus de temps d'entraînement, la fonction perte affiche quelques instabilités visibles, cela implique que l'augmentation du nombre des couches ne conduit pas à un meilleur modèle, il est à noter que lors de la diminution du nombre de couches (couches veut dire sous-couches soit une couche outre que l'entrée et la sortie), le modèle, dans le cas d'une seule couche, est incapable de reproduire la courbe avec une bonne précision, et ne réussit pas à capturer la complexité de la fonction volatilité implicite, en effet, la perte est de 0.05 en terme de RMSE.

3.6 Impact de l'optimizer

Il y a différents types d'optimizers, ces derniers jouent sur la manière dont les variables sont mises à jour à chaque étape d'entraînement du modèle, parmi les plus connus: Adam, RMSprop et SGD, on donne le résultat dans le cas de l'utilisation de RMSprop dans l'optimisation du modèle.

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

Le temps d'entrainememnt du modèle est $13min2s$, plusieurs paramètres sont testés pour λ dans le cas du RMSprop, on choisit $\lambda = 10^{-4}$.

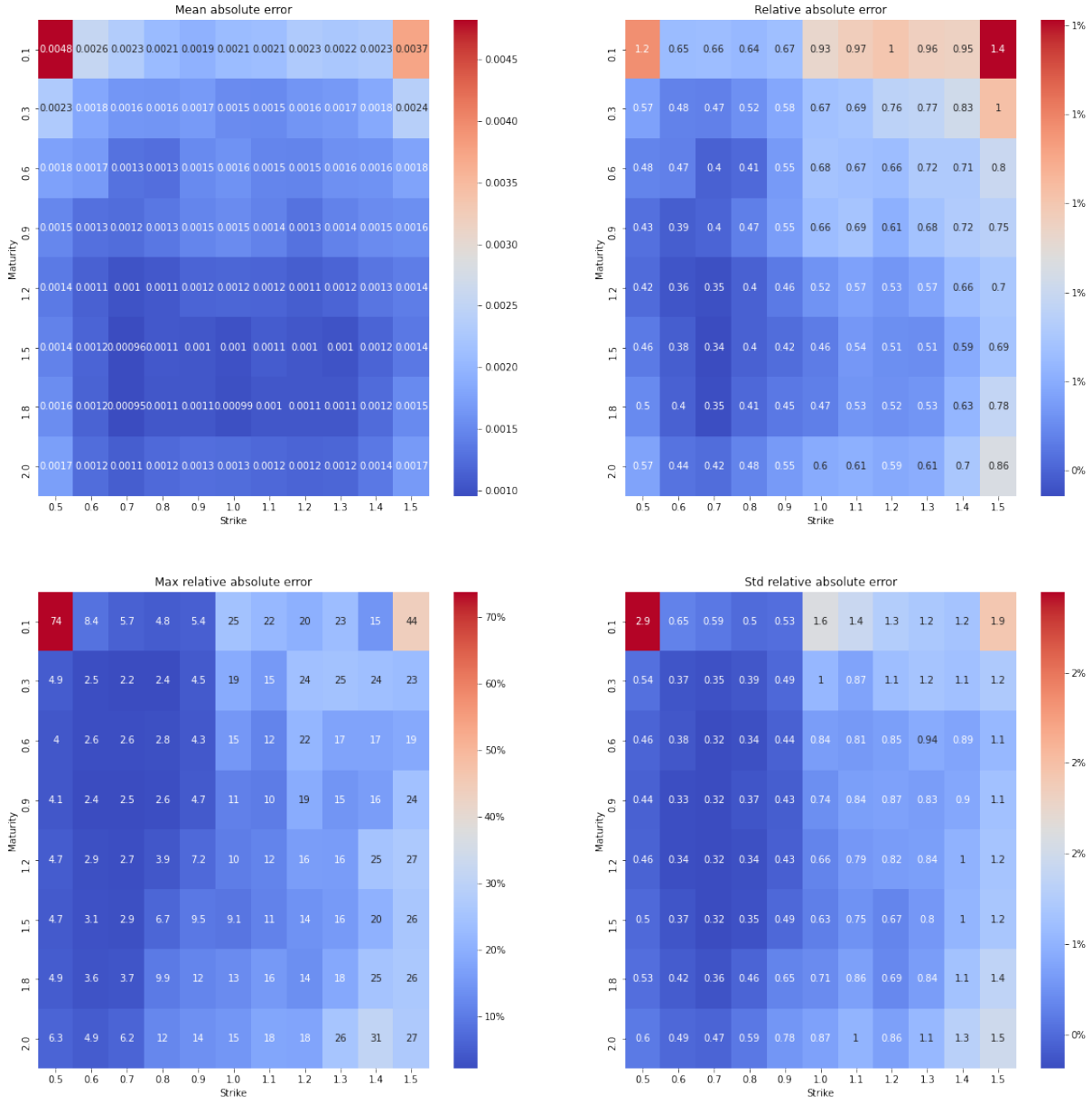

Figure 19: Heatmap pour les données train

sur les données de test

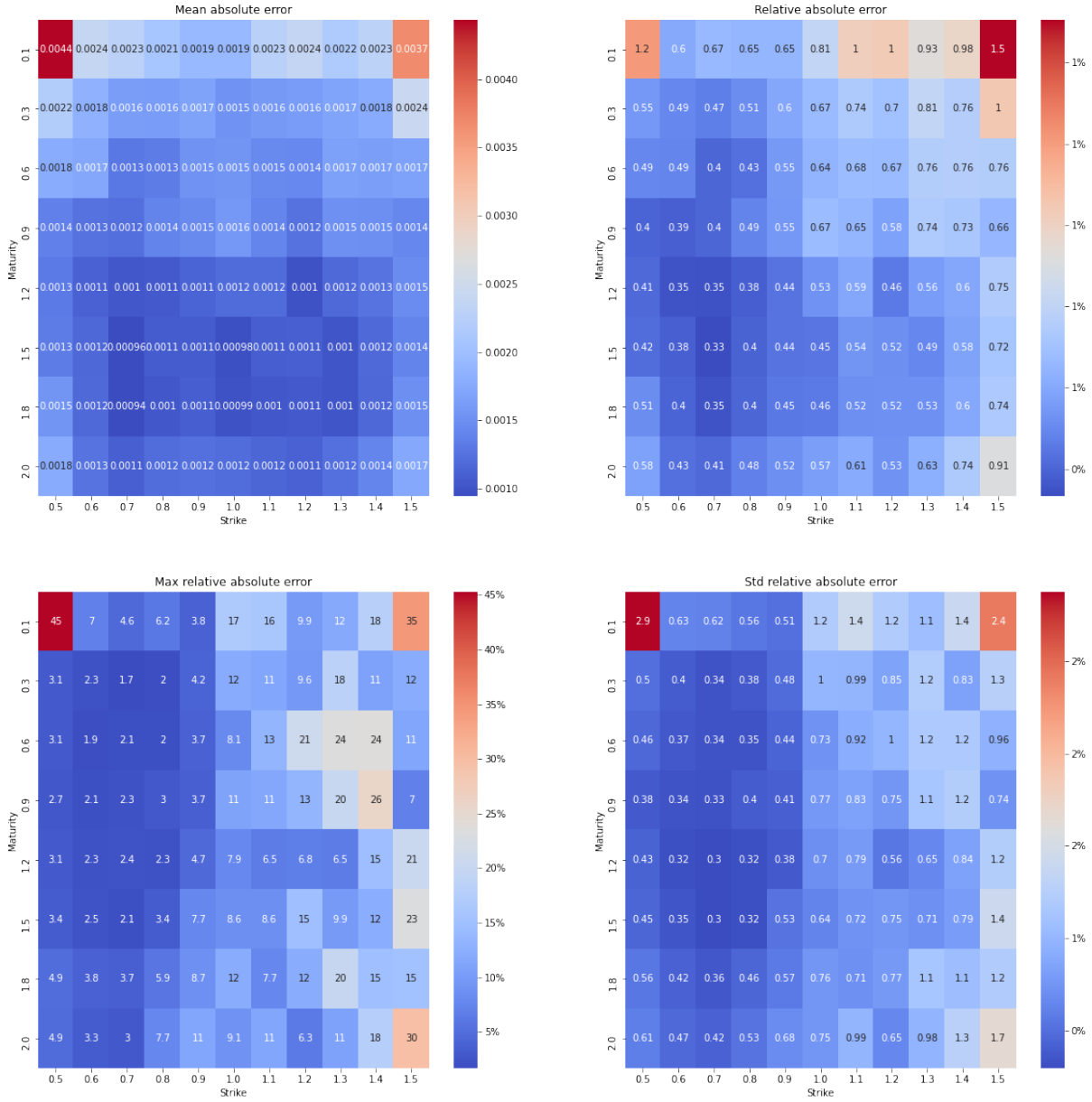

Figure 20: Heatmap pour les données test

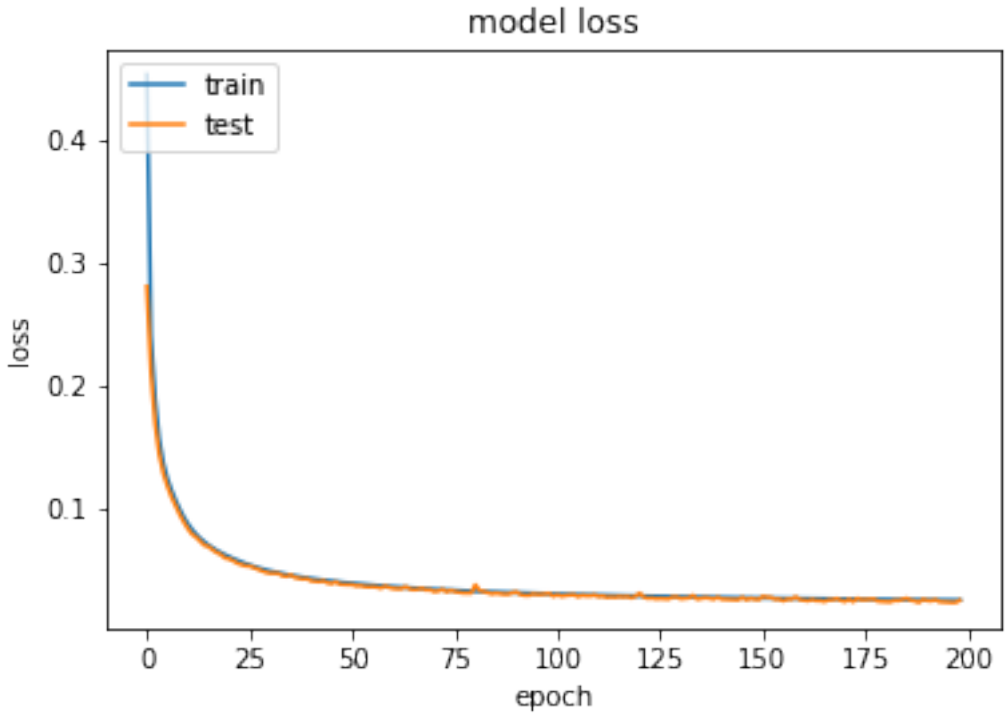

Figure 21: Erreur du modèle en fonction du nombre d'epochs

Sur les données de test, il est clair que le modèle fournit des résultats moins précis par rapport à l'optimiseur d'Adam, en particulier sur la figure 20 l'erreur moyenne relative est particulièrement importante à faible maturité, la courbe de perte présente plus d'instabilité que dans le cas de l'optimisation par Adams, par conséquent, nous déduisons que ce dernier est un meilleur choix dans le présent cas.

3.7 Impact de la fonction d'activation

Il y a plusieurs fonctions d'activations qui peuvent être choisis, les plus utilisés de ces méthodes sont: relu, sigmoid (adapté pour des problèmes de classifications), elu. Dans notre cas, on choisit l'impacte de l'utilisation de la fonction d'activation relu sur la qualité du modèle. Il est généralement préférable d'utiliser dans les problèmes de calibration la fonction relu.

L'architecture utilisée dans l'optimisation en changeant la fonction d'activation sont les suivantes:
$\lambda = 10^{-4}$, optimizer=Adam, epochs=256, batch size=128, nlayers=2, neurones=2^5, early stopping patience=10.

Le temps d'optimisation de ce modèle est de 13min 55s.

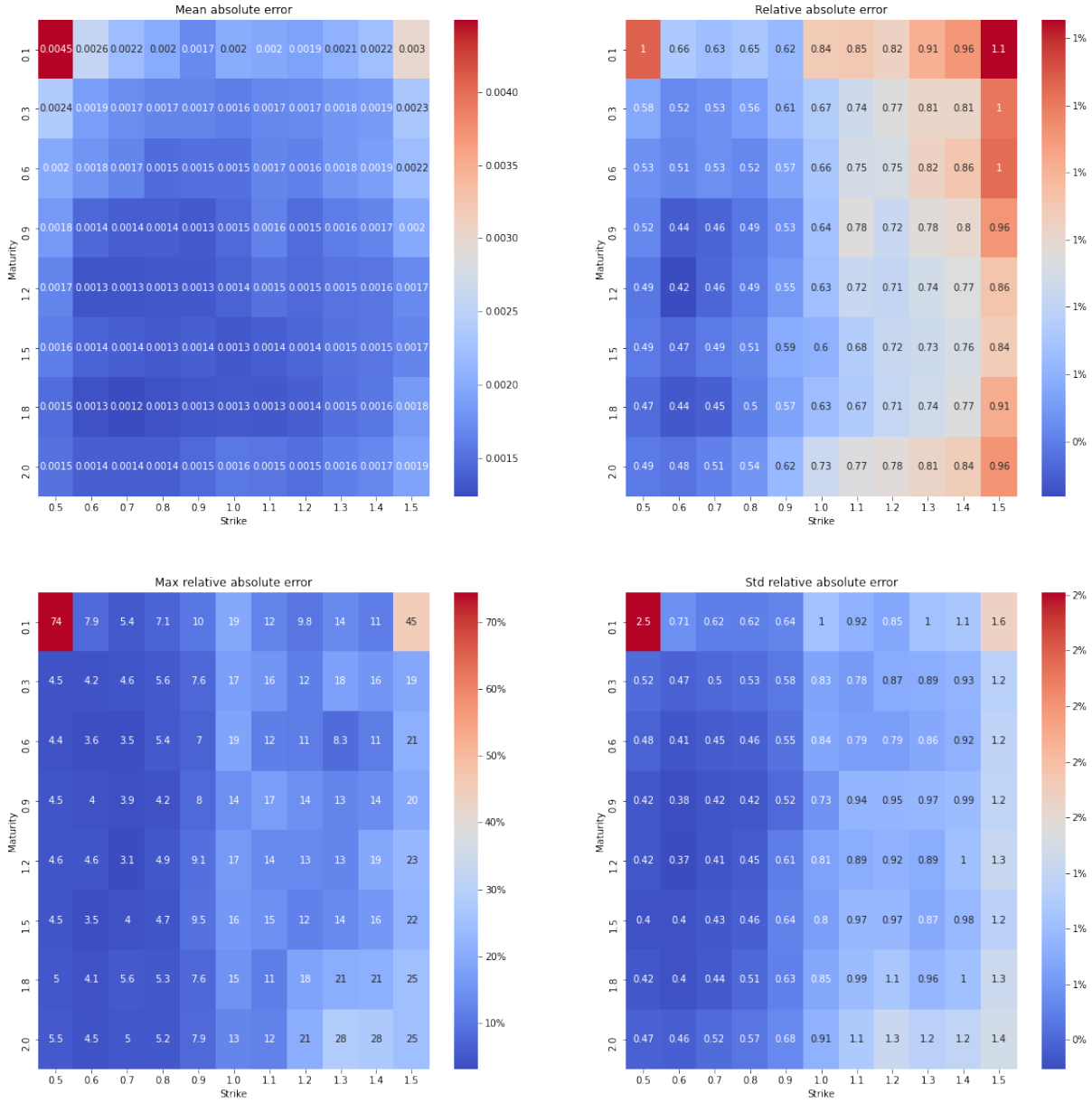

Figure 22: Heatmap pour les données train

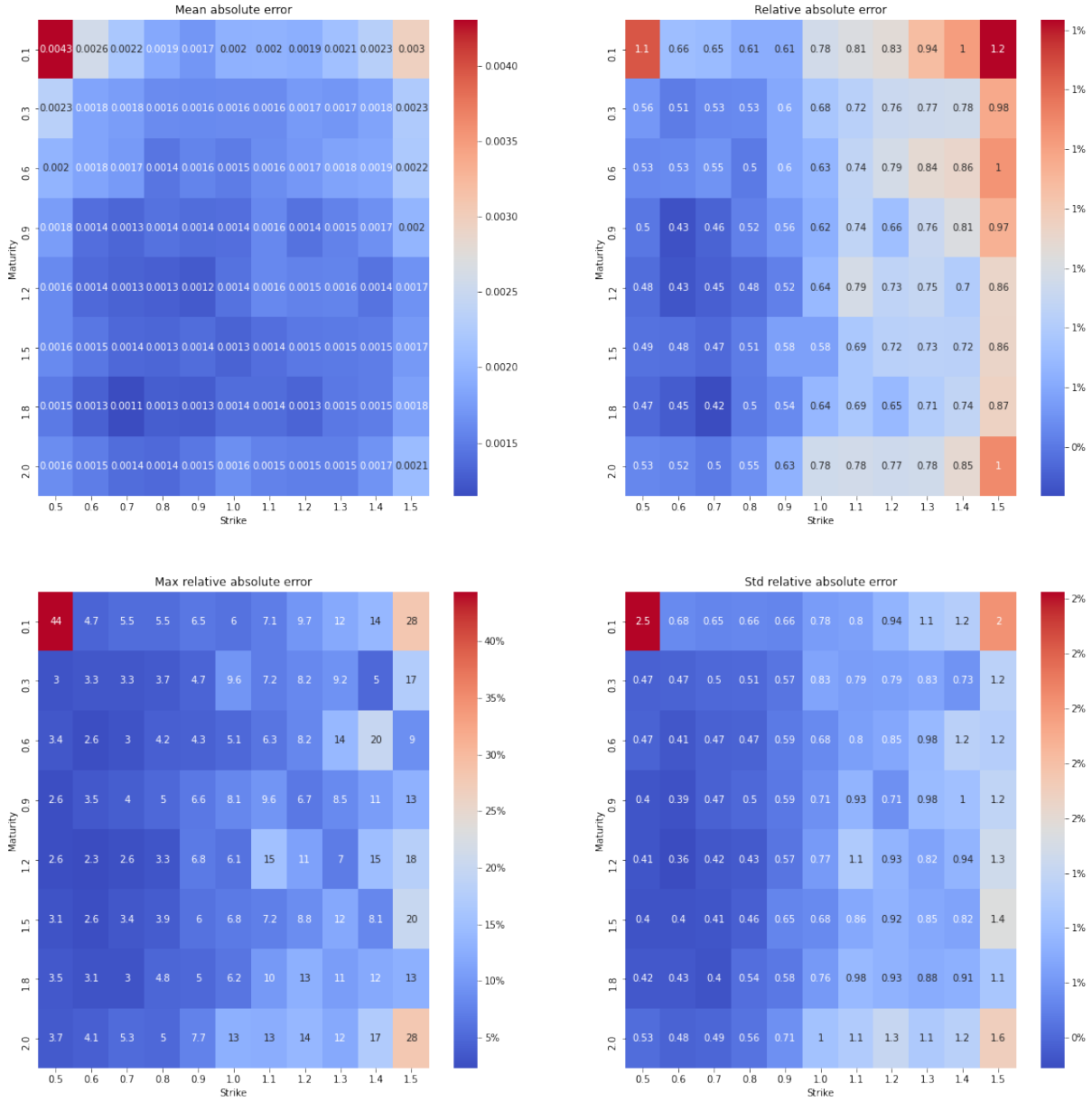

Figure 23: Heatmap pour les données test

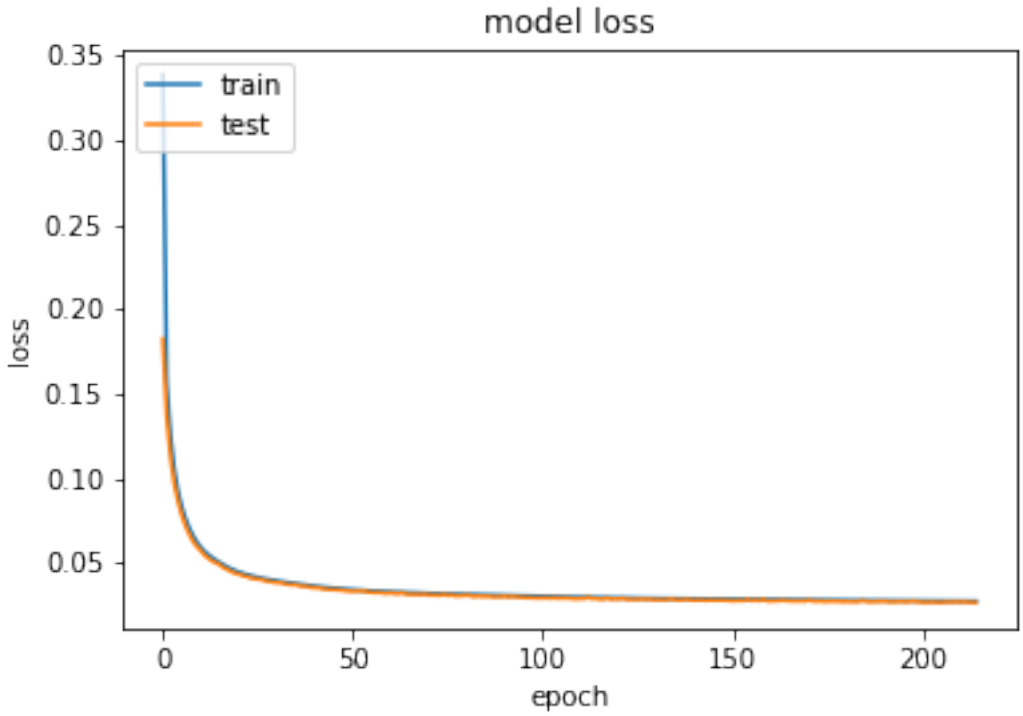

Figure 24: Erreur du modèle en fonction du nombre d'epochs

Comme prévu, pour des paramètres identiques, l'utilisation de la fonction d'activation elu conduit à des résultats bien plus stables comparés au cas de l'utilisation de la fonction d'activation relu, ce qui est clairement observé dans la figure 23, l'erreur en particulier l'erreur moyenne est bien plus prononcée.

3.8 Impact du Batch normalization et du Drop Out

La méthode de normalisation par lot est généralement destinée à accélérer la convergence du modèle de Deep Learning, mais son application introduit dans ce cas précis des instabilités dans les modèles dont la couche est paramétrée par défaut, comme illustré ci-dessous. Ceci est également observé pour l'application de la méthode Drop Out en amont de la dernière couche.

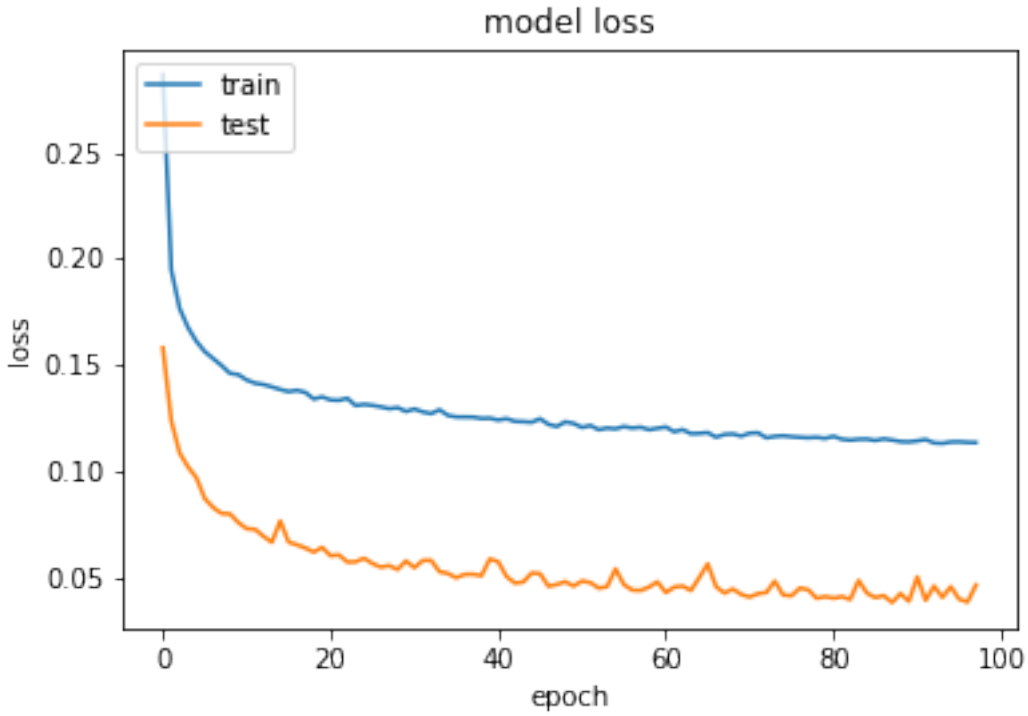

Figure 25: Erreur du modèle en fonction du nombre d'epochs Batch Normalisation

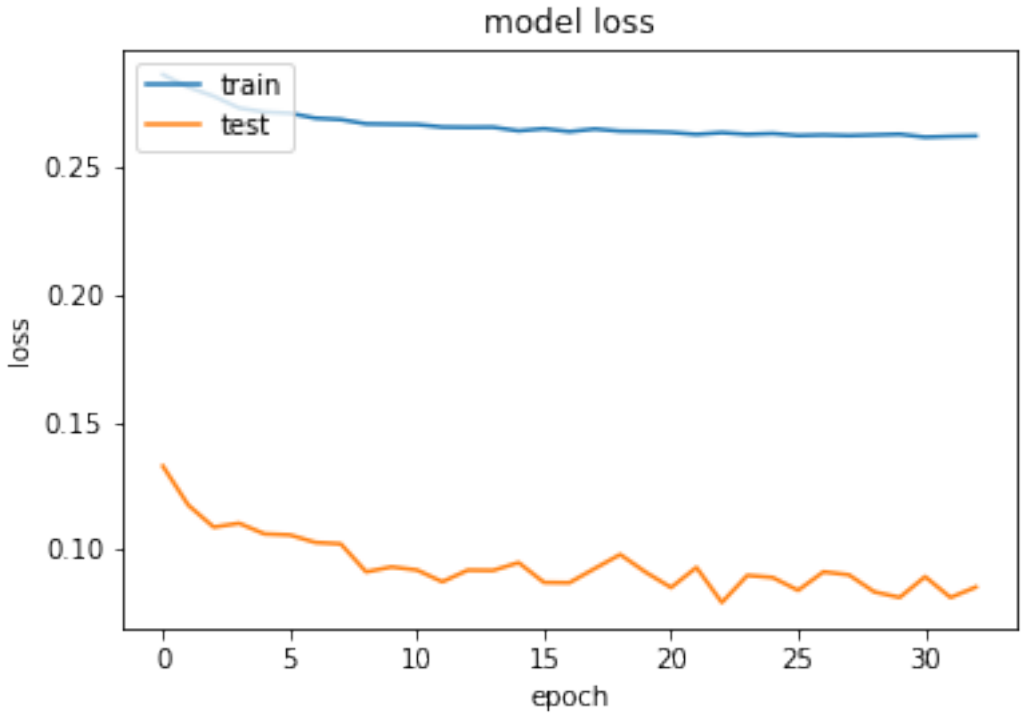

Figure 26: Erreur du modèle en fonction du nombre d'epoch Early stopping

L'erreur d'apprentissage est supérieure à l'erreur de validation, ce qui n'est pas habituel dans le cadre des applications de Deep Learning, il est par conséquent souhaitable de

renoncer à ces deux méthodes et de privilégier l'optimisation en générant davantage de données.

Nous n'avons pas regardé l'impact du nombre d'epochs sur la qualité de la calibration car nous avons opté pour l'arrêt anticipé, en effet, le nombre d'epochs est le nombre de passage sur l'ensemble des données d'entraînement, plus cette valeur est grande plus le modèle sera optimisé, en revanche, une grande valeur peut conduire à un surentraînement du modèle, nous laissons le choix de ce paramètre à la fonction "callback" Early Stopping.

4 L'architecture retenue

L'architecture retenue est celle avec les paramètres suivantes: $Adams(10^{-4})$, epochs=256, batch size=128, $nlayers = 2$, $n_{neurones} = 2^5$,

5 Conclusion

La méthodologie adoptée pour régler les paramètres s'avère problématique, car un biais est introduit lors de l'amélioration du score du modèle sur des données de test, ainsi le modèle peut ne pas bien se généraliser lorsqu'il est testé sur de nouvelles données, une optimisation de cette approche serait de faire une validation croisée permettant de réduire le biais introduit lors du choix des paramètres sur une base de validation. Une autre amélioration similaire à celle introduite dans l'article de BDSE est de générer les données à chaque étape du training, ce qui optimisera le rendement des modèles choisis.