Performance Comparison of Convolutional Neural Network Optimizers for Brain Hemorrhage Classification

Josue Antonio Theodore Hattenbach Mara Hubelbank Sahil Sangwan Yinglin Wang
Department of Mathematics, Northeastern University

May 1, 2022

Abstract

The aim of this project is to train a Convolutional Neural Network (CNN) model for image classification, using Zeta Surgical's labeled CT scan image dataset. When provided with an unlabeled brain CT image, the model should provide an accurate prediction of which class the bleeding belongs to. Each image is classified as epidural, intraparenchymal, intraventricular, subarachnoid, subdural, or as belonging to multiple of the previous classes ("multi"). Moreover, the data is pre-processed to remove "flagged" samples, which are presumed to be labeled incorrectly or correspond to a corrupted image. It is additionally processed to grayscale format and down-sampled; then, it is fed to a CNN model which uses a sparse categorical cross-entropy loss function, Adam optimizer, and learning rate configured with gradient descent. As of the most recent training/testing iteration, our model has reached a testing accuracy score of 0.989; thus, it is well on its way to a well-generalized model for classification of CT brain scan data.

1 Introduction

Within the realm of computer-aided decision making, there exists a myriad of applications; in the context of computer vision, one such crucial application is the procedure of clinical triage, diagnosis, and prescription. Intracerebral hemorrhaging (ICH) is the second-most common cause of stroke (15-30%) and the most deadly [5]. For these fast-acting, high-risk injuries, doctors rely on their domain expertise to predict the correct diagnosis, necessarily taking into account the previous CT scans they've encountered and their corresponding diagnoses. Using this scan data, computer-aided decision-making in the form of image-classifying Convolutional Neural Networks (CNNs) can greatly improve the timeliness and effectiveness of care; enhancing this task is the focus of one of Boston-based Zeta Surgical's innovative medical technology initiatives.

2 Methodology

The task of this project is to build and train a CNN model to classify brain hemorrhage CT scan images into one of the classes epidural, intraparenchymal, intraventricular, subarachnoid, subdural, or as belonging to multiple of the aforementioned classes ("multi"). Additionally, we compare the performance effect of various optimizers on the final CNN model iteration.

2.1 Pre-processing

As a source dataset, we are given Zeta Surgical's labeled brain CT images of patient; feature-wise, we chose to focus on the brain-bone window depicted in each image. These images were loaded in using the CSV label files provided to us. We chose images with at least one set of outline labels, resulting in between 900 to 1300 sample images for each class. Since we do not have a CSV of labels for the intraventricular class, we chose a random sample of 1000 from all the intraventricular images, to account for the intraventricular class while avoiding over-training on it.
The images are loaded in from their files as colored images, and our first step is to convert them to grayscale. Then, we perform training, testing, and validation split using 0.80 / 0.15 / 0.05 ratios respectively. Finally, we perform downsampling on each of the sets individually using a factor of 16; the original images of dimension (512, 512) are converted to (128, 128) images.

2.2 Modeling

Our model structure consists of two convolution layers and two dense layers with a dropout layer between the two dense layers. Both take kernels of size (3, 3) and use ReLu as the activation function. Their only difference is that the first convolution layer has 32 output filters while the second layer has 64 output filters. After each convolution layer, the result is then pooled by taking the max value over (2, 2) windows.

Following the convolution section of the model, the output is flattened before being given to the neural network section. The dense layers have dimensions of 16 and then 6. The final dense layer has 6 as its dimension because there 6 total classes (5 individual types of brain hemorrhage as well as multiple types being present), so the class label values range from integers 0 to 5 inclusive. A dropout layer was added between the dense layers to avoid overfitting of the model. It has a dropout rate of 0.5, meaning that half of the input it receives are dropped.

The structure of our model with two convolutional layers is similar to that of two of the models tested in a study of classification techniques for diagnosing brain tumors from MRI images. Since both our topics deal with medical diagnosis of the brain and there is a degree of similarity between CT and MRI images, we used this paper as a reference. [4]

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_10 (Conv2D)           (None, 126, 126, 32)      320
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 63, 63, 32)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 61, 61, 64)        18496
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 30, 30, 64)        0
_________________________________________________________________
flatten_5 (Flatten)          (None, 57600)             0
_________________________________________________________________
dense_10 (Dense)             (None, 16)                921616
_________________________________________________________________
dropout_5 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_11 (Dense)             (None, 6)                 102
=================================================================
Total params: 940,534
Trainable params: 940,534
Non-trainable params: 0
_________________________________________________________________
```

2.3 Optimizers

When building a convolutional neural network, optimizers play a vital role in speeding up the process of updating the weights. In this paper we tested the performance of our CNN model for image classification using seven out of the nine Tensorflow Keras Optimizers Classes. The utilized classes are AdaDelta, AdaGrad, Adam, AdaMax, NAdam, RMSprop, and SGD (Stochastic Gradient Descent).

2.3.1 SGD

SGD is the father of all other optimizers. In fact, all the other optimizers are nothing but the advancements done to the basic idea of Stochastic Gradient Descent. The S in SGD refers to stochastic, which essentially means “randomness”. SGD randomly picks a single point at each step of the iteration and uses that to estimate the rest of the points. Although SGD is an enhanced version of gradient descent which works way faster, when it comes to industry, SGD isn’t used much.

2.3.2 AdaGrad

Unlike SGD, where the learning rate is hard-coded while initializing and it remained constant throughout the execution, AdaGrad uses a technique called adaptive learning rate. The idea is that the learning rate keeps changing according to certain conditions of the weights. Moreover, AdaGrad also uses the concept of momentum to reach the minima faster. It's known that AdaGrad works best for sparse data.

2.3.3 RMSProp

RMSProp (Root Mean Square Propagation) is essentially an advanced version of AdaGrad which was developed while keeping in mind the weaknesses of AdaGrad. When using AdaGrad, the learning rates are reduced a lot after going through several batches. This introduces the issue of a very slow convergence by the optimizer. RMSProp tackles this issue by exponentially reducing the learning rates, so they don't get very low, making the process of updating weights very slow. Just like AdaGrad, RMSProp also uses the concept of momentum to reach the minima more effectively. It's recommended to implement RMSProp when dealing with RNNs (Recurrent Neural Networks) but it also performs well in usual scenarios with a similar perfomance to AdaDelta, and it's only surpassed by the Adam optimizer in some of the scenarios.

2.3.4 AdaDelta

AdaDelta is another improvement of AdaGrad, where delta refers to the difference between the current weight and the newly updated weight. AdaDelta removes the adaptive learning rate technique and replaces it with an exponential moving average of squared deltas. This change solves the issue of AdaGrad's slow convergence but it also makes this optimizer computationally very expensive. Moreover, AdaDelta doesn't require you to manually set a global learning rate since you're not dealing with learning rates at all. Consequently, AdaDelta works well when you're not sure about what learning rate to set since it doesn't require any.

2.3.5 Adam

The Adam (Adaptive Movement Estimation) optimizer is one of the most used optimizers since it was built while keeping in mind the weaknesses of the aforementioned optimizers. This optimizer is great when it comes to computational efficiency, taking very little memory and making it the ideal option for large data sets. The Adam optimizer also takes advantage of the previous learning rates by using the previous gradients to update the current weights. Moreover, the Adam optimizer has greater momentum while going towards a minima, while reducing the irregularities in its path. As mentioned before, Adam doesn't need a lot of memory and it's very efficient when it comes to computational power. Lastly, Adam doesn't require a lot of tweaking with the parameters since even with minimal tuning, it works well enough. Hence, the Adam optimizer is the best among the adaptive optimizers in most of the cases.

2.3.6 AdaMax

As the name suggests, AdaMax is an adaptation of the Adam optimizer; it's a variant of Adam based on the infinity norm (max). This optimizer is often considered superior to Adam, especially in models with embeddings.

2.3.7 NAdam

The Nesterov-accelerated Adam, or the NAdam is another adaptation of Adam to add the Nesterov momentum, which is an improved type of momentum. This is an extension to momentum where the update is performed using the gradient of the projected update to the parameter rather than the actual current variable value. In essence, Nesterov momentum is a 'smarter' version of momentum and helps reach the minima while keeping the future steps in mind. In conclusion, NAdam is Adam plus Nesterov Momentum.

2.4 Evaluation

Due to our aim of comparing optimizers for this particular use case, we made a model for each of the seven common optimizers. AdaDelta, AdaGrad, Adam, AdaMax, NAdam, and RMSprop use the implementation from TensorFlow while SGD is implemented by us so we can manually find the best learning rate. The models all share the same structure and only differ in the optimizer they are compiled with. They are trained using the same set of training and validation data with 3 epochs and batch sizes of 512. Similarly, they are all evaluated using the same testing set. We were limited by the computing capacity of our machines with how many epochs we could feasibly run. Ideally, we would want to aim for 5 - 10 epochs during training with early stopping implemented.

3 Results

We first plot the loss of each model over each training epoch, to determine trends in model loss minimization over each iteration (Figures 1 and 2). We subsequently plot the accuracy of each model over each training epoch (Figure 3). The accuracy graphs indicate how much progress is made over each epoch, and a comparison of the rates would allow us to determine which model learned the fastest. Finally we plot training accuracy with validation accuracy for each model (Figures 4 and 5). These graphs demonstrate the translation of training success into success on validation data, with mixed results. (View Appendices)

3.1 Comparison of optimizer methods

The trends in accuracy and loss reflect the expected behavior of each optimizer. Consider the loss and accuracy over epochs of the model trained with the naive SGD optimizer. Compared to Adam, AdaDelta, AdaGrad and AdaMax optimized models, this model began with considerably higher loss on the first epoch, then minimized it over the next 2 epochs (Figure 1). It did not approach the minimum losses of the other optimizers, nor did it minimize as quickly (Figure). This can be attributed to the fact that SGD has one global learning rate for all parameters, while the other models have learning rates tailored to each parameter. Furthermore, SGD does not have momentum, i.e. it does not add a discounted factor of the previous gradient to the current gradient, which means it will take more epochs to converge on local minima of the loss function. Adam, AdaDelta, AdaGrad, and AdaMax are designed to resolve these issues.

The differing trends between these optimizers with momenta can also be explained by their implementation. AdaGrad is designed to perform updates on the learning rate of each parameter, at every step. The update formula divides the learning rate by the squared sums of past gradients. As these accumulate, the learning rate approaches 0, and the model ceases to learn. AdaDelta builds on the AdaGrad optimizer by restricting the window of past gradients to some fixed size, thus dampening the rapid decrease in the learning rate. Our results for the AdaDelta and AdaGrad models (accuracy vs epoch) confirm this. The AdaGrad model initially improved in accuracy from the first to the second epoch, but decreased on the third (Figure 3). This may be attributable to its learning rates having decreased too quickly. AdaDelta experienced a decline in accuracy as well, but it was less severe than that of AdaGrad, because of the restricted window over which the squared past gradients could be used to divide (and decrease) the learning rate.

4 Conclusion

We found the best optimizer is the Adam optimizer with 0.98 testing accuracy. SGD is close, but its loss is slightly larger across multiple runs. There are many areas of improvement. The observed testing and validation accuracy is significantly higher than the training accuracy for all models. This may suggest that the models are underfit, but we would have to confirm this by running more epochs which is limited by our present hardware constraints. Additionally, we would like to include CT images of healthy brains too so that our model will be able to diagnose the presence of brain hemorrhages rather than only classify the hemorrhages.

5 Acknowledgement

We would like to thank Professor Wang for giving us the inspiration, background knowledge, and domain proficiency necessary for this project. Moreover, we would like to thank Zeta Surgical for providing the expansive pre-labeled data set, and for informing us of this critical application of matrix methods in data analysis and machine learning.

6 Appendices

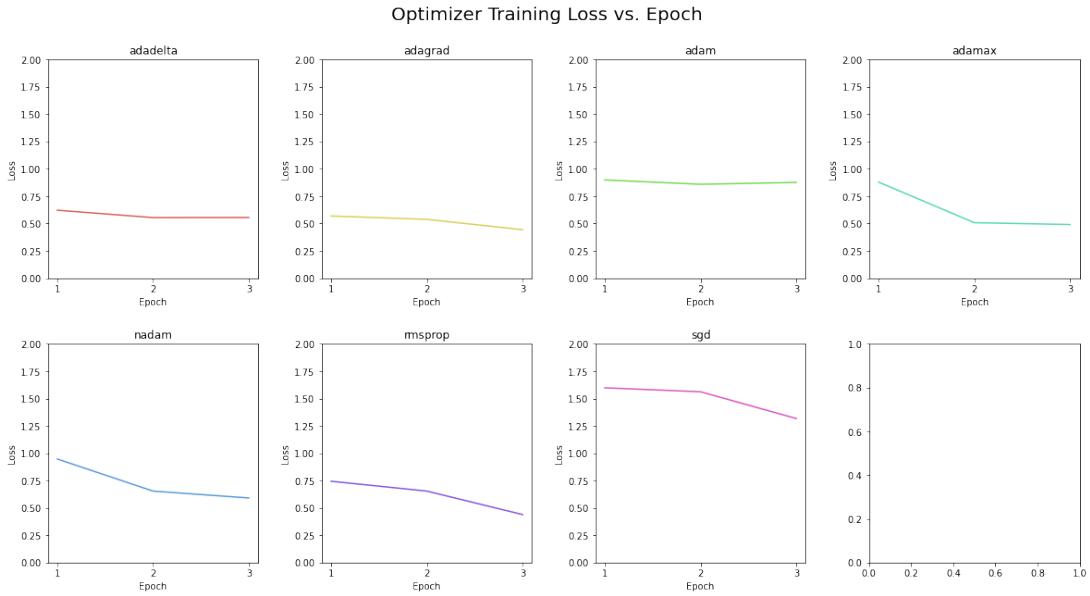

Figure 1: Loss vs. Epoch by Optimizer

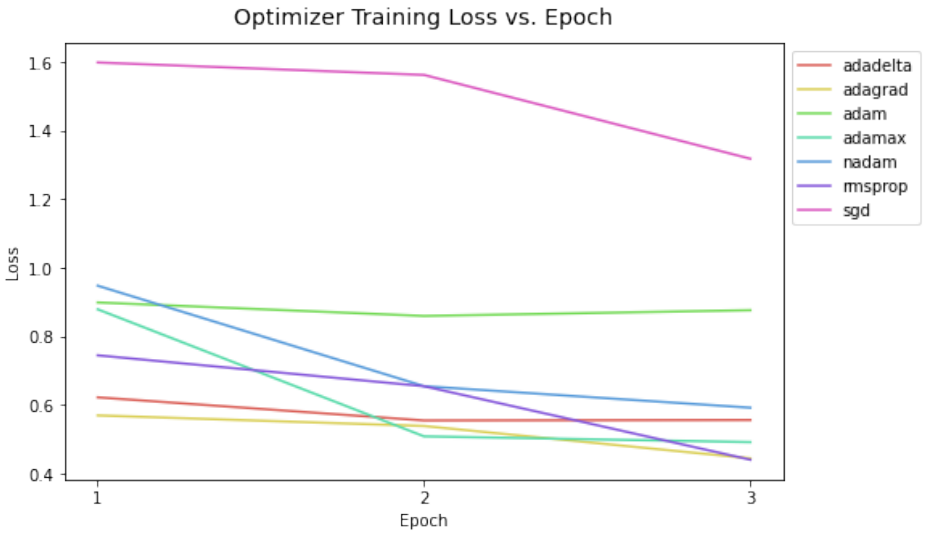

Figure 2: Training Loss vs. Epoch, All Optimizers

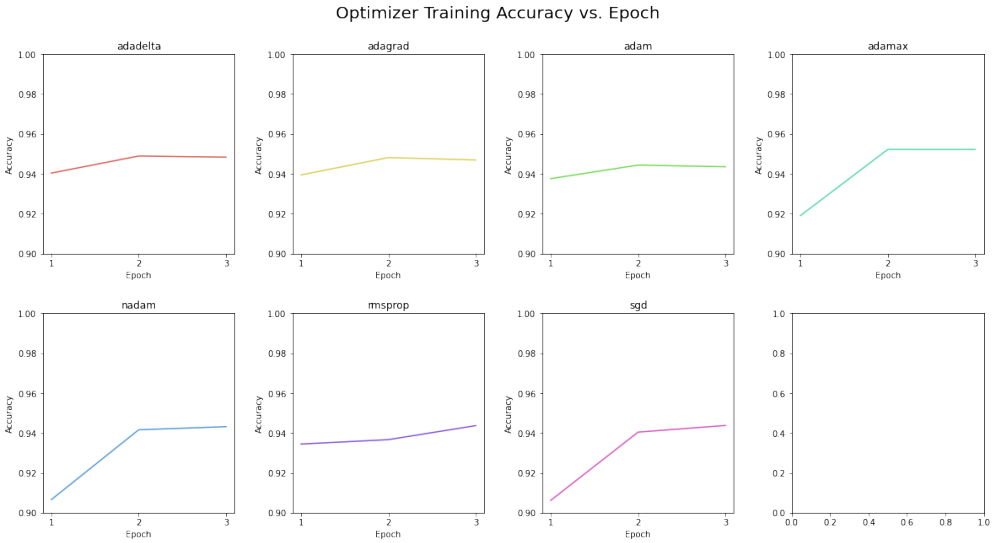

Figure 3: Optimizer Training Accuracy over Epochs

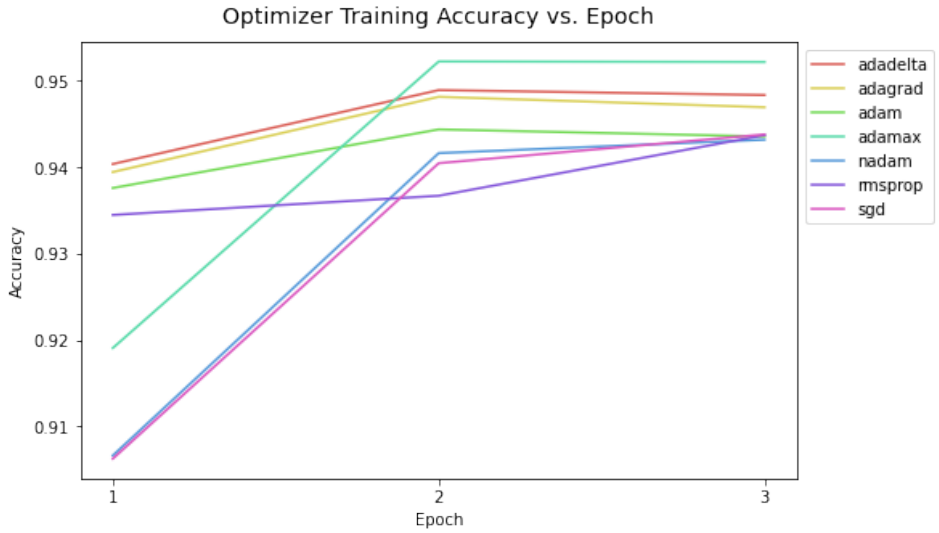

Figure 4: Training Accuracy vs. Epoch, All Optimizers

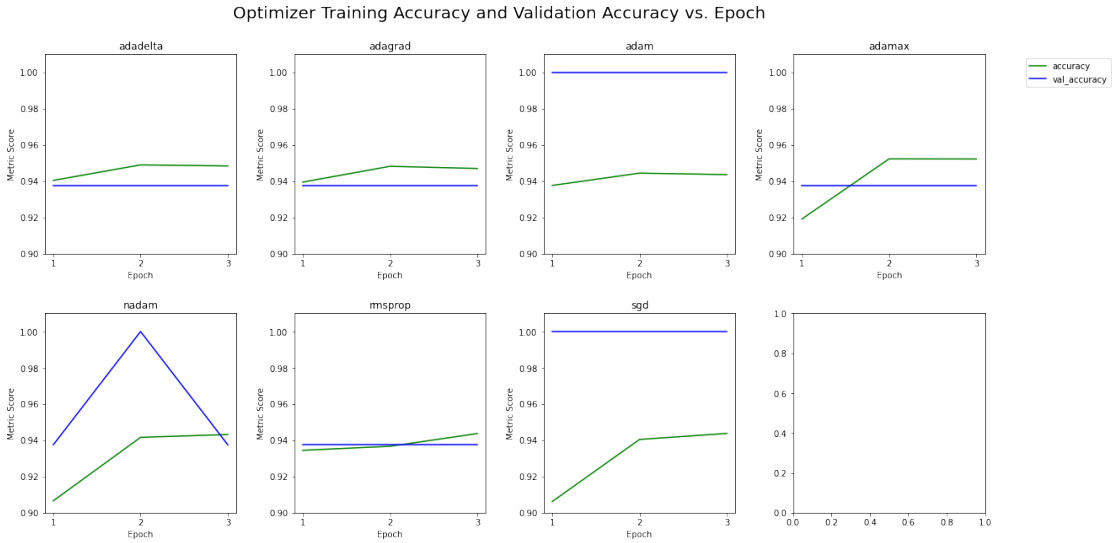

Figure 5: Optimizer Training Accuracy and Validation Accuracy over Epochs

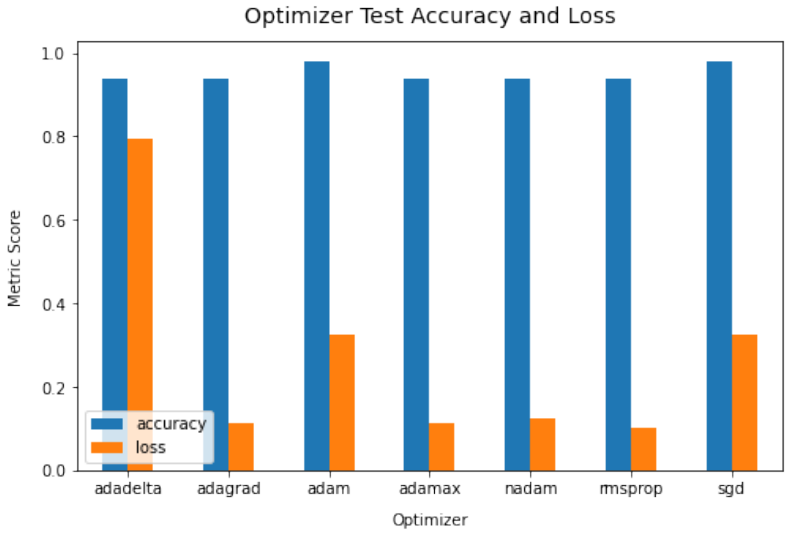

Figure 6: Optimizer Accuracy and Loss on Test Data

References

[1] https://analyticsarora.com/complete-glossary-of-keras-optimizers-and-when-to-use-them-with-code/#How-Do-Optimizers-Work

[2] https://analyticsindiamag.com/guide-to-tensorflow-keras-optimizers/

[3] Ruder, Sebastian. "An Overview of Gradient Descent Optimization Algorithms." Arxiv, 15 June 2017, https://arxiv.org/pdf/1609.04747.pdf.

[4] Wahlang, I.; Maji, A.K.; Saha, G.; Chakrabarti, P.; Jasinski, M.; Leonowicz, Z.; Jasinska, E. Brain Magnetic Resonance Imaging Classification Using Deep Learning Architectures with Gender and Age. Sensors 2022, 22, 1766. https://doi.org/10.3390/s22051766.

[5] Neel, Patel T, and Scott D Simon. "Intracerebral Hemorrhage." AANS, https://www.aans.org/en/Patients/Neurosurgical-Conditions-and-Treatments/Intracerebral-Hemorrhage.