MINI-PROJECT 3 REPORT
COMP 551, APPLIED MACHINE LEARNING
Modified MNIST

By
Carl Perreault-Lafleur (260745153),
Rivka Mitchell (260668534),
Khoren Ponsin (260733452)

Fall 2019

School of Computer Science

Abstract

The goal of this project is to implement a model that predicts the highest value digit in an image containing three randomly placed, randomly rotated MNIST digits in a gray-scale image. The models were used to participate in a Kaggle Competition titled “Modified MNIST". The goal of this competition was to achieve the top prediction accuracy score on a provided test dataset. Through a variety experiments used to tune hyperparameters and determine the top performing neural network architecture, we arrived at a model which achieved a test accuracy score of 96.033%. We also tried different techniques such as building a larger dataset or preprocessing our data by splitting each picture into three single digits. Unfortunately, those methods did not lead to a better accuracy.

1 Introduction

Image analysis is a major research area of study in Machine Learning. For this project, we were given 50000 images obtained from a Modified MNIST data set. Each image contains three digits, and the goal is to output the digit in the image with the highest numeric value. One could be tempted to think that the best approach to that prediction problem is to first split each image into three single digits and then predict each digit independently. It turns out, as it often does in deep learning, that this intuitive approach does not give better results than a brute force model on data that did not undergo a lot of preprocessing.

The preprocessing of the data consisted of removing the background of each image so that the digits were more distinguishable. As it is very common to use Convolutional Neural Network (CNN) models when dealing with image analysis tasks, our first model was a 4-layer CNN. We spent a lot of time tuning the learning rate and the batch size of that model. Whereas we could quickly draw conclusions on which batch size to use to increase our accuracy and reduce our running time, it was harder for the learning rate. In the end, the tuning of that 4-layer CNN did not generate much better results. This led us to implement another neural network: one based on the structure of the VGG-16 neural network. On the contrary to our previous neural network, we added batch normalization and this increased the accuracy significantly. Our highest score on Kaggle, of 96.033% was achieved with VGG-16 along with batch normalization.

In parallel to tuning the 4-layer CNN and VGG-16 models, we attempted the intuitive preprocessing approach of splitting each image into three single digits. Using that preprocessing along with the 4-layers CNN, the overall prediction of the biggest digit in each picture was worse, even though the single digit classifier worked very well on the validation set. We also tried to build a larger data set using randomly placed MNIST digits but once again, it did not yield to a higher accuracy on the test set. Since this technique required some extra computational cost, we did not investigate it further.

2 Dataset and setup

The data we worked with consisted of pictures containing 3 rotated digits on different funky backgrounds. More precisely, we possessed the gray-scale matrices of every picture, each with value being an integer between 0 and 255, where 0 represents a black pixel, 255 represents a white pixel, and any value between those bounds is a gray pixel with black and white intensities proportional to its value. The given training set consists of 50000 examples, while the test set consists of 10000 examples. From the training set, we created our own validation set by randomly selecting 5000 pictures. Hence, in the rest of the project, the training set will refer to the set of 45000 training examples, and the validation set the other 5000 examples.

In terms of resources, we quickly understood that our computers were not powerful enough to allow us to experiment as we would like. Thus, we used the Google Platform Compute Engine. We chose a virtual machine that is specialized in heavy computations rather than having big memory.

3 The Preprocessing

As explained above, the raw data that we were given was composed of three digits (recognizable by their gray scale value of 255), on different funky backgrounds. Our first idea was to find a way to remove those backgrounds. However, since deep learning models generally do not need feature engineering, we set the gray scale intensity of all the pixels with gray scale intensity strictly less than N to 0. This threshold $N \in [0, 255]$ is now a parameter that we can tune, and we will be able to know if we obtain better results with $N = 0$ (namely without touching the image). By inspection, we found that our first 4-layer CNN (detailed below) was giving the best results with $N = 250$.

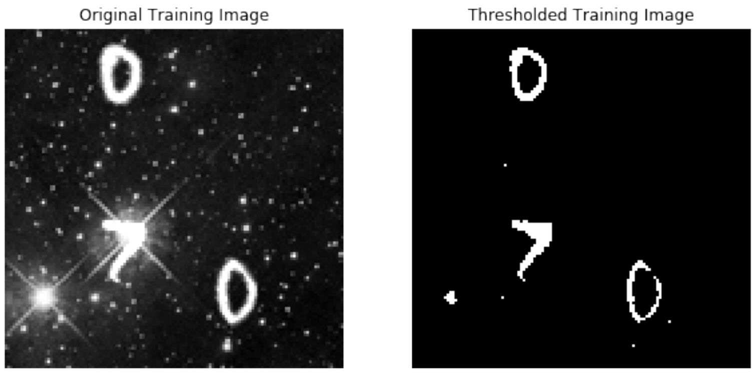

Figure 1: An example of the tresholding idea with $N = 250$.

4 Different models trained

The first model we implemented was a 4-layer Convolutional Neural Network (CNN). We spent the most time on improving this model. When we saw that we reached a plateau with this model, namely we were not able to improve the best accuracy anymore, we tried another model, the popular Visual Geometry Group 16 (VGG-16). In this section, we provide details on the work we did to improve these two models.

For each of the models, we used the cross entropy loss function as our criterion function, and the Adam optimization function. These choices were based on in class suggestions. While training the models, we looked at the evolution of both the losses obtained by the model for the training and validation set, and the accuracy of the predictions of the model on the validation set during training. In order to select the best model and avoid overfitting we implemented our own stopping criterion. The criterion is as follows: if the accuracy on the validation set increases by less than $\varepsilon \in \mathbb{R}^+$ for $m \in \mathbb{N}$ epochs in a row then the training stops. We then keep the model that has been trained at the epoch that yielded the best accuracy. If the early criterion does not occur, we simply choose the model that has been trained at the epoch which yielded the best accuracy.

Also, since we could not train any model by feeding the entire training set because of memory issues, we had to train it using batches of the training set. That is, we train iteratively $M \in \mathbb{N}$ training examples at a time. This parameter M is the batch size.

4.1 Convolutional Neural Network

As a first try, we implemented a simple 4-layer CNN. The architecture of the model is detailed here:

Hidden Layers 1 to 4
2D-Convolution
Dropout
ReLU
2D-Max Pooling
Output layer
Linear Transformation

Table 1: Architecture of the 4-layers CNN

We had to choose a learning rate for this CNN, which we did by tuning this parameter.

A. Tuning the Learning Rate

The most critical parameter for a CNN is the learning rate. A learning rate that is too high can lead to a plateau in terms of accuracy, while a learning rate that is too low might reach optimal accuracy too slowly. Therefore, we fixed the batch size to 256, and varied the learning rate. We collected the accuracy after the stopping criterion was reached. Note that we chose the maximal number of epochs to be sufficiently large so that the validation set accuracies would converge for each different learning rate. We obtained the graph in Figure 2.
As expected, the best accuracy was reached with one of the smallest learning rates. Hence, for our 4-layers CNN, we will use the learning rate of $8 \cdot 10^{-5}$ from now on. However, we must admit that the behaviour of the curve was not expected. We expected a decreasing curve since theory indicates that lower learning rates lead to higher the accuracy. Nonetheless,

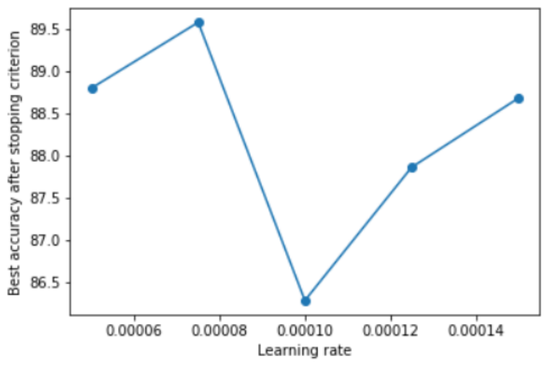

Figure 2: Tuning of the learning rate parameter, for a batch size of 256

reaching the convergence points of the smaller learning rates takes much more time than for the bigger learning rates. For example, it took 260 epochs for models using the learning rate $8 \cdot 10^{-5}$ to converge, but at most 80 for those with larger learning rates to converge. Hence, the running time dimension is non-negligible in the choice of the learning rate. This is why in the next part we experimented with different values of the batch size to see if this parameter was correlated to the time the model needs to converge in terms of validation accuracy.

B. Tuning the Batch Size

The second hyperpameter we tuned was the batch size. We tried 6 different batch sizes: 16, 32, 64, 128, 256 and 512, by fixing the learning rate to $8 \cdot 10^{-5}$ as discussed above. It turned out that the smaller the batch size we used, the faster the model converged (in terms of accuracy on the validation set). This is observable in Figure 7 in the Appendix: for each model, we plotted the accuracies on the validation set at each epoch. We also observed that large batch sizes gave lower accuracies. This is not surprising since it is well known that "when using large batch sizes there is a persistent degradation in generalization performance" [1]. This is also discussed in [2]. In Figure 7 and Table 2, the "Chosen Model" corresponds to the epoch that satisfied the early stopping criterion or the epoch that gave the highest accuracy (if the early stopping criterion was never satisfied).

Batch Size	**16**	**32**	**64**	**128**	**256**	**512**
Running time per epoch (min)	4.53	5.13	4.84	4.5	4.52	4.25
Chosen model accuracy on validation set	0.9251	0.9224	0.9140	0.8952	0.8880	0.8556

Table 2: The highest accuracy and the running time with different batch size.

From the above table, one can see that the highest accuracy on the validation set was reached with a batch size of 16. However, from the graph of Figure 7 in the Appendix, one can see that the early stopping criterion was satisfied much earlier than the 100th epoch (at the 60th epoch more precisely). However, by submitting on Kaggle, we only got a score or 0.90, which showed us that a small batch size has the propensity of not generalizing well. Since we did not want to take the risk of overfitting, we did not use batch size 16 for the VGG-16 model. As mentioned earlier, the larger the batch size, the slower the convergence of the model and the harder it is to generalize. This led us to not use batch sizes 128, 256 and 512 for the VGG-16 model. We were then left with batch size 32 and 64. Since we witnessed that a small batch size led to overfitting (what happened with our batch size 16 model), we decided to pick a batch size of 64 for further experiments (such as trying different activation and loss functions). Unfortunately those experiments did not improve the accuracy. This lead us to conclude that we reached a plateau on this 4-layers CNN. We therefore decided to implement a new model.

4.2 VGG-16

After performing multiple tests, it became apparent that our 4-layer CNN model was not able to achieve accuracy scores above 90% on the test data. With this in mind, we decided to try another form of neural network. More specifically, we implemented an architecture inspired by the VGG-16 neural network. We began by training a model using the exact VGG-16 neural network. Unfortunately, the validation accuracy score of this model reached a plateau at 27%. As indicated by [4], batch normalization is said to improve the accuracy performance of VVG-16. In fact, we found that adding batch normalization at the end of each layer solved this problem and allowed us to create a model which reached 96.56% accuracy on the validation set. Also, by inspection (by default of tuning the learning rate which takes too long), we noticed that taking a learning rate of $1 \cdot 10^{-4}$ performed well.

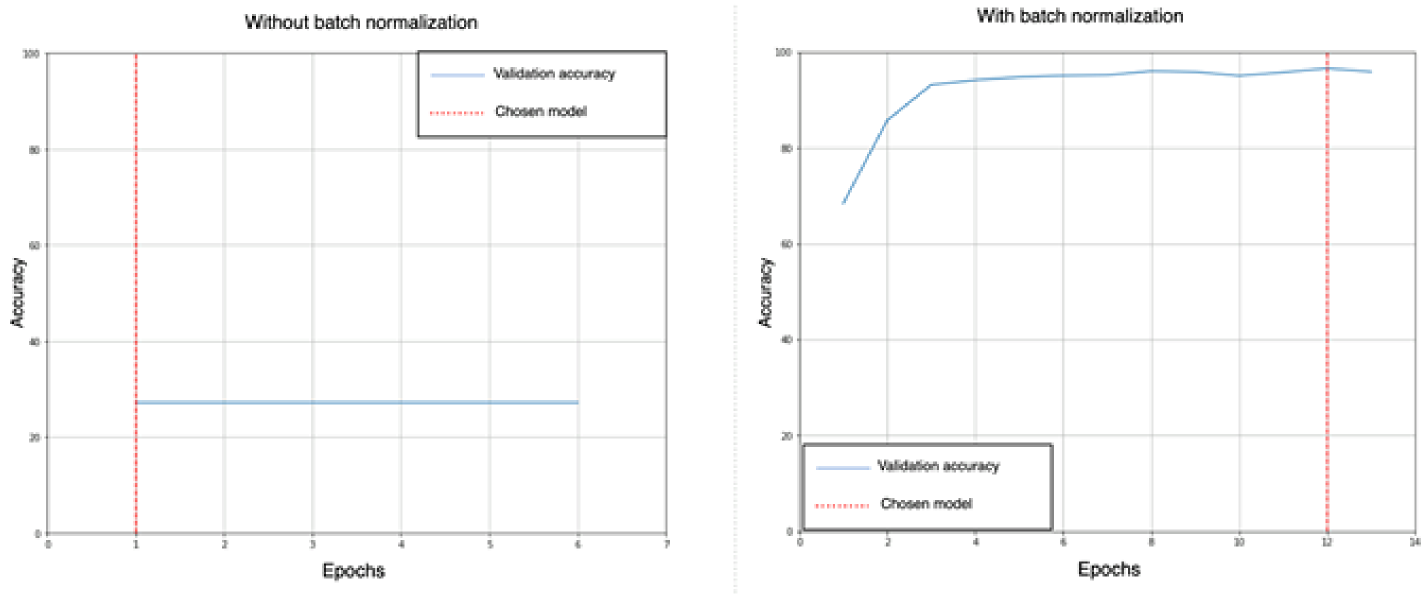

Figure 3: Influence of batch normalization on VGG-16 model validation accuracy scores

5 Other attempted approaches

5.1 Intuitive Approach

Our first approach was to attempt to predict the value of each digit present in a test digit. Given an image from the test dataset, we began by applying our preprocessing techniques. We then extracted the portions of the image containing a digit and resized these new images to be of the same size as the MNIST training data, as shown in Figure 5. Since the handwritten digits are not always fully connected, this process resulted in a slight loss of information. Approximately 2% of the test images were not properly split. Through this technique, we had transformed the test data such that each entry was now composed of three images, one for each digit present in the original entry.

We then imported the traditional MNIST dataset, [3], and applied a rotation to each image by a random degree between 0 and 360, as seen in Figure 4. This transformed data was then split into a training set of 60000 images, and a validation set of 10000 images. It was then used to train our 4-layer CNN, resulting in a model that achieved an accuracy score of 94.8% on the validation set.

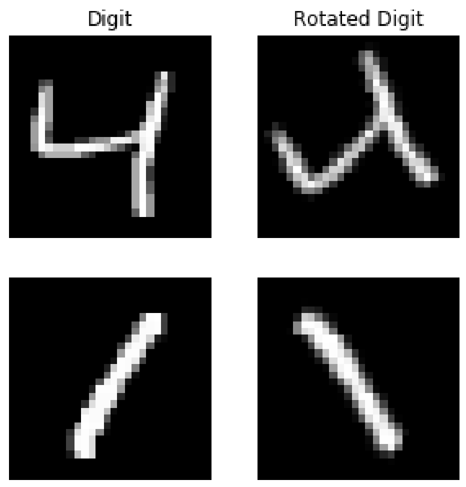

Figure 4: Transformations of MNIST data

For each entry in our transformed test dataset, we used our model to predict the value of each image in an entry. This resulted in three predictions per test image; one for each digit present. Taking the maximum of the three predictions gives the form of prediction that we desired for the project task. Unfortunately, this technique did not prove fruitful, and the test accuracy of this model was 22%.

5.2 Building a Larger Dataset

Another approach we took was to double the size of our training data. We did this so that our model would train on a larger variety of images, in the hopes of achieving a model with lower bias and lower variance. We did this through the following process.

Consider the partition of the training data $C_0, ..., C_9$, where each C_i corresponds to the set of training samples in class i. Then for each $i \in \{0, ..., 9\}$ we used [5] to create $|C_i|$ new images containing three randomly placed MNIST digits on a black

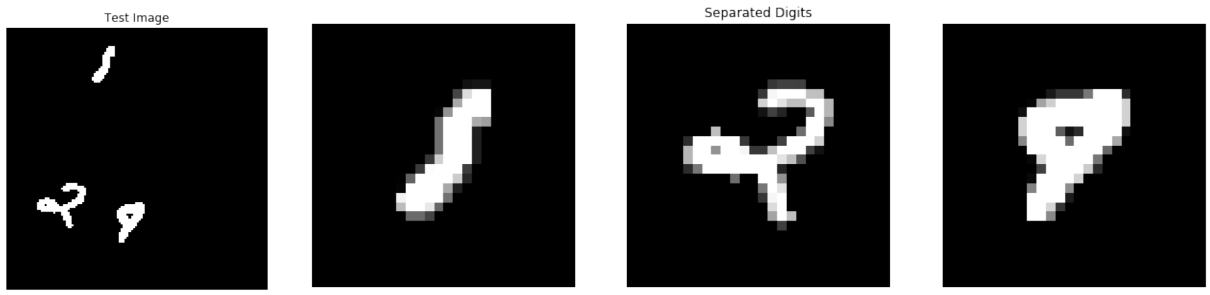

Figure 5: Separation of digits in test image

background, with the highest digit present in the image being i. Since [5] does not create images with rotated digits, we transformed each newly created image by rotating each digit present by a random angle between 0 and 360 degrees. These transformed newly created images as well as the preprocessed previously provided training data were then used to train our 4-layer CNN model.

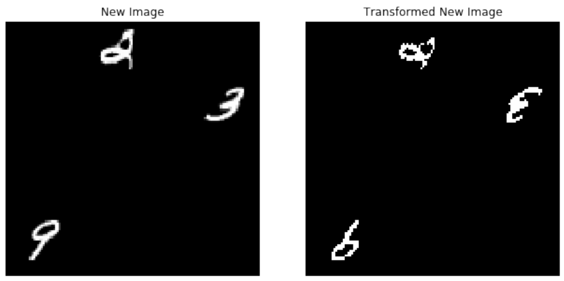

Figure 6: New image created using [5] and its transformation

After training this model we achieved an accuracy score of 92% on the doubled training data. While the resulting test accuracy of this model was 89%, it did not prove to be an improvement in accuracy from the same model trained on the originally provided training data alone. By training on the original training dataset and the doubled dataset, we garnered models with almost the exact same test accuracies. For this reason, we abandoned this technique as its test accuracy score was not worth its extra computational cost.

6 Final Results

In the end, despite our efforts to help our models by giving it single digits to predict the value on instead of the original picture, or building a bigger dataset, the method that gave us our best result was based on VGG-16. After adding the batch normalization, as discussed in part **4.2**, we managed to reach 96.033% on the test set, our current best score on Kaggle.

7 Discussion and Conclusion

There are several directions one could take to attempt to form a better prediction model. Seeing as our modified VGG-16 achieved the best test accuracy score, it is likely that by performing parameter tuning on this model, one would be able to achieve a better model. It would be advisable to perform experiments to determine the optimal values for the batch size and the learning rate.

Further, another well known model for image classification is the VGG-19 model. Another direction to take would be to determine whether this model outperforms our modified VGG-16 model, and determine the optimal hyperparameters and normalization techniques to apply for this model.

Lastly, it would be interesting to investigate whether ensemble methods would lead to improvements in model accuracy. By this we mean, once a best model is determined, to train multiple versions of this model on the training data. Using these multiple models, we would form predictions with each model on the test data. Then, we would take the majority vote from these models for each test image. It is likely that should we train multiple high performing models, the probability that the majority of them will misclassify a test image is very low. Therefore by taking a majority vote, we would hope to see an improvement in test accuracy score.

In conclusion, while our modified VGG-16 model granted us with relatively successful test accuracy score, there are still many more experiments we could conduct, and changes we could make in order to improve this result. Also, we can say that against our intuitions, but as it is well-known in deep learning, the preprocessing was not the key to improve the accuracy. We also understood the importance of computational power, as time constraints were a major obstacle we encountered when training our models.

8 Statement of Contributions

Understanding the Google Compute Engine, and designing the neural networks was completed by Carl Perreault-Lafleur. Hyperparameter tuning was completed by Khoren Ponsin. Other attempted approaches were done by Rivka Mitchell.

References

[1] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks, 2017.

[2] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima, 2016.

[3] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[4] Dmytro Mishkin, Nikolay Sergievskiy, and Jiri Matas. Systematic evaluation of CNN advances on the imagenet. *CoRR*, abs/1606.02228, 2016.

[5] Shao-Hua Sun. Multi-digit mnist for few-shot learning, 2019.

9 Appendix

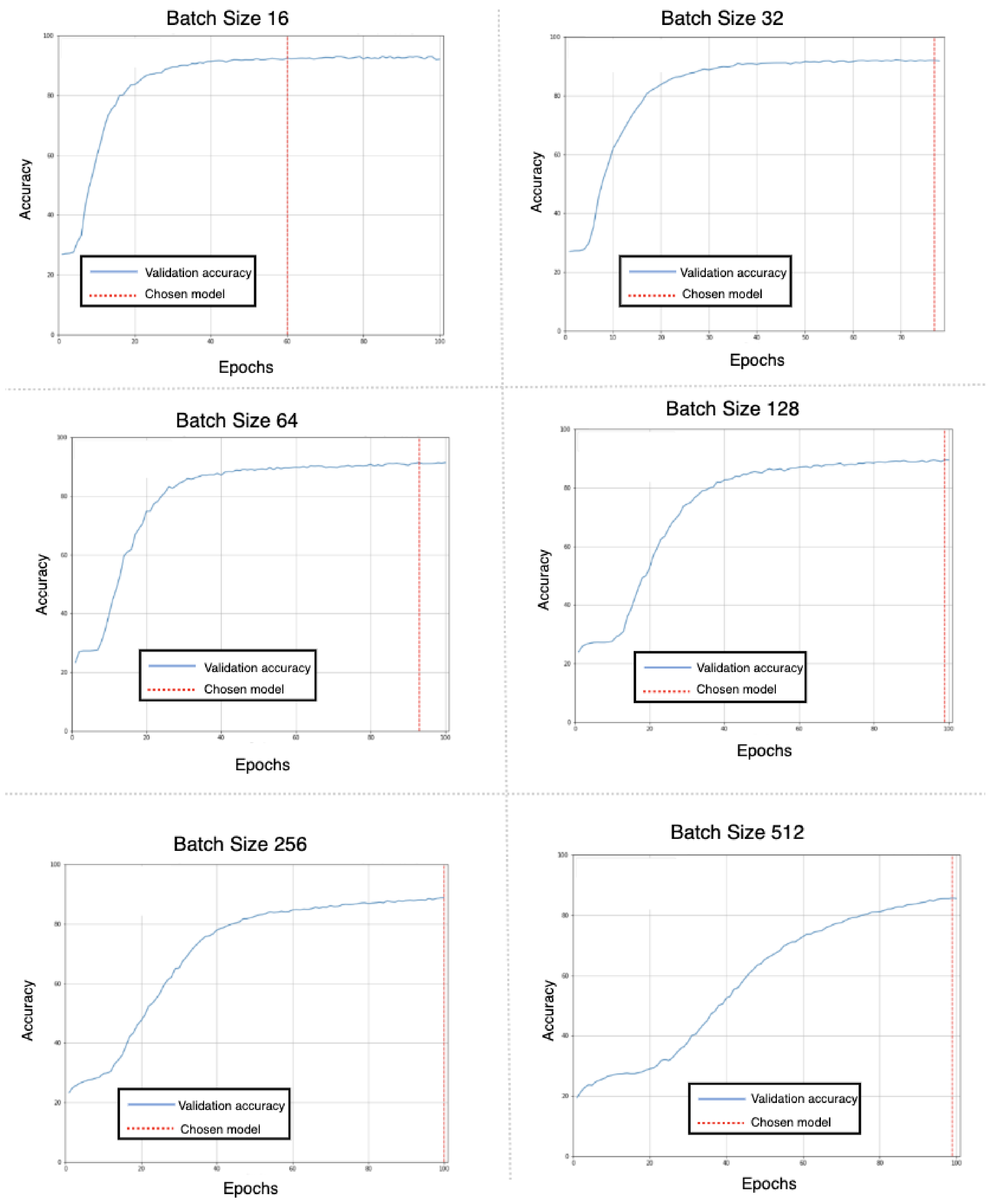

Figure 7: Plots of the accuracies on the validation set at each epoch for different batch sizes. The learning rate used was 0.00008. Note that we ran each model on 100 epochs except for batch size 32, for which we ran 80 epochs only.