Abstract

Convolutional neural network models have far-reaching applications in the field of computer vision, yet they all have a primary goal of object classification. Classifying handwritten digits is a simple example that helps to introduce key concepts of convolutional neural networks.

The Scikit Learn digits data set, a paired down version of the MNIST data set, is used to develop a fully connected neural network model with one hidden layer for digit classification. Keras, with Tensorflow as the backend, is used to generate the neural network model. While neural networks have many different hyperparameters, the focus here is on the number of nodes in the hidden layer. Therefore, all other hyperparameters are held constant.

In the end, a framework for building a convolutional neural network model is introduced, and the output of the hidden layer is explored in greater detail. Commentary is provided throughout on key observations, and the results are generalized to other digit classification exercises.

Introduction

Digit classification is often regarded as the "hello world" for convolutional neural networks. Oftentimes, convolutional neural networks are very complex models, comprised of multiple hidden layers or different architectures. While neural network models in general are regarded as "black box processes," the goal in using this small and prebuilt digits dataset is to help provide education and awareness into a neural network model. More specifically, there are four primary goals in this assignment:

1. Understand how the number of nodes impacts model accuracy
2. Understand what is happening in the hidden layer, within the broader context of a neural network architecture
3. Introduce a model-building framework
4. Draw conclusions about digit classification exercises

Literature Review

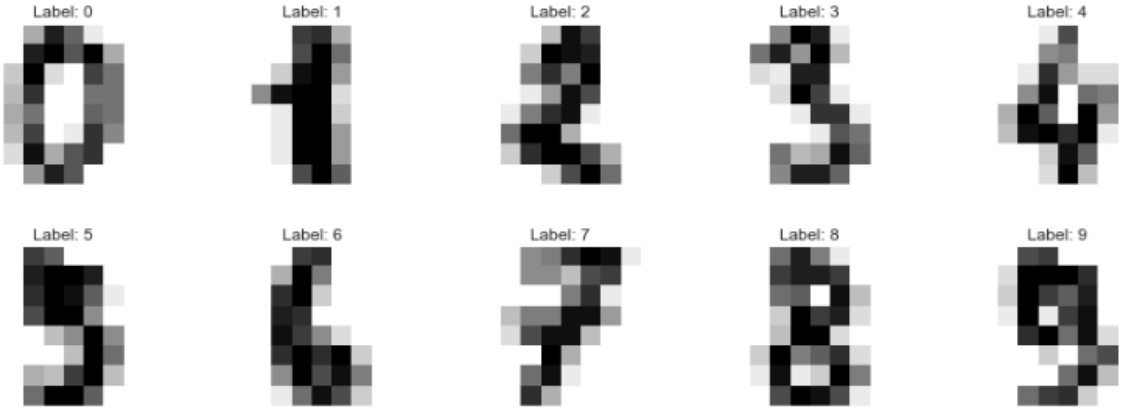
The primary source of information for this assignment was the Chollet Deep Learning with Python textbook. However, other sources were consulted to gain a deeper understanding of a fully connected neural network architecture, specifically a multilayer perceptron (MLP) which has one hidden layer. A brief summary of key findings can be found in the Appendix, but they are not comprehensive by any means.

In short, by better understanding model architecture, it is clear that the number of nodes and their weight values play a critical role in model building process. In fact, according to Chollet, the weights contain the network's knowledge (p. 58). Hence, this observation has helped to inform the focus of this assignment.

Data Preparation

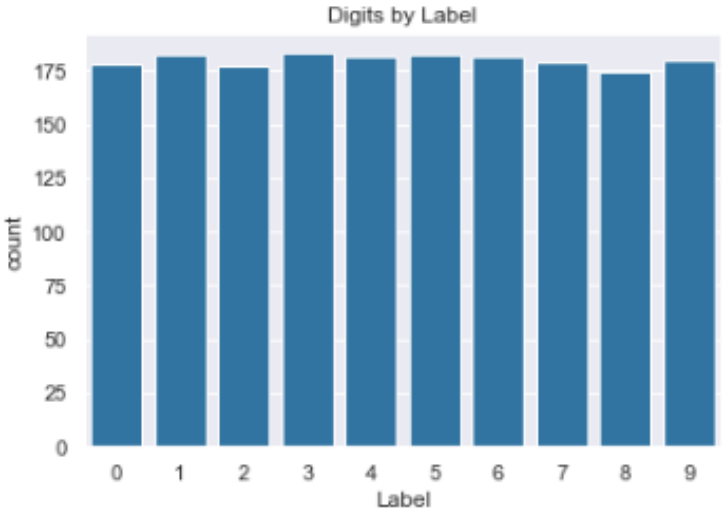
In this section, the digits data is loaded, inspected, and prepared for modeling. Based upon the digits documentation and validation steps performed, the digits data is comprised of 8x8 images that have pixels ranging from 0 to 16. Additionally, both the documentation and data validation show that there are no missing data values. It is important to validate the shape, range, and missingness of any data set, especially one used for building a neural network model.

As a next step, the first ten digit images are shown below. This step is helpful in better understanding what the data actually looks like.



The digit images, although fairly blurry, are also distinct. In validating a model, visualizing the images rendered may be a helpful exercise to consider.

Another data inspection step is to better understand how the data is split among each digit class. The plot below shows the row counts for each class.



The plot above shows that the distribution by digit class is fairly even. Had there been a less even distribution then balancing the classes may have been a data processing step to consider.

Given that data shape, range of values, missingness, and class distribution have been inspected, the data is ready to be prepared for modeling. The data preparation steps consist of:

- Reshaping the pixel data so it is of shape: (number of rows, 8x8)
- Standardizing all pixel values so that they are between [0,1], which can be accomplished by dividing by the maximum pixel value of 16
- Transforming the class labels using a one-hot-encoding mechanism
- Splitting the data into training, validation, and testing sets using a 60/20/20 split

These steps help to ensure that the neural network can readily ingest and manipulate this data during modeling.

Methods

In this section, several different models will be built using both the training and validation sets. While k-fold cross validation was considered (see code in Jupyter Notebook Appendix), it was ultimately removed from scope. By building models on the training set and validating them on the validation set, greater speed and simplicity could be achieved in meeting assignment goals. It is important to note that several functions are used to ensure code reproducibility.

Based on the goals of the assignment, the hyperparameters that will be adjusted are only the number of nodes within the hidden layer. While an initial set of node sizes was proposed, after building several models, the node sizes were adjusted to the following: 1, 2, 5, 8, 16, and 64. Thus, in total seven models will be built for this assignment. The below hyperparameters will be held constant:

- Batch size is 32.
- Epoch size is 500, but will be adjusted before generating predictions for the test set.
- Categorical crossentropy is the loss function used, as this is a natural choice given the nature of the problem (Chollet, p. 84).
- Accuracy is a key metric to measure model performance. In fact, loss and accuracy curves are used to assess/compare model performance and inform the optimal batch size.
- Rmsprop is the optimizer used, with its default learning rate.
- RELU is the activation function used in the hidden layer, as it is a common activation function. Softmax is the activation function in the output layer, because this assignment is dealing with multiclass output.
- For each model, a model summary is provided capturing the number of weights and shape of the output for the hidden and output layers.

Protecting against overfitting is essential when working with these types of models. According to Chollet, "the universal tension in machine learning is between optimization and generalization" (p. 114). Therefore, a

relatively large batch size of 500 is used as the primary mechanism to determine where a model starts to overfit the training and validation data sets.

Lastly, Chollet also states that "If your data is divided into many categories, you may cause information bottlenecks if you make the intermediate layers too small" (p. 92). Using a small number of nodes and comparing performance against a relatively larger number of nodes can validate this statement. Moreover, creating a small network is another avenue for addressing overfitting in a neural network model.

Model One - One Node

In analyzing the output from model one, the model with one node, the model summary table shows that the input-to-hidden layer contains 65 parameters, or weights. In recalling the neural network architecture, there are 64 input nodes (8x8 images) plus one bias node connected to one hidden layer node, for a total of 65 weights. The hidden layer itself has one bias neuron, which is connected to each output node, adding ten more weights to the model (for a total of 20). The model summary helps to keep track of the model architecture and is provided for background purposes only.

More importantly, the loss and accuracy curves show that there is both high loss and low accuracy, which means that this model does not perform well. The accuracy barely gets above 35%. There is some mild separation in the training and validation curves from 400 epochs onward, which is where the model starts to overfit. Furthermore, the loss and accuracy curves are not as smooth, which speaks to Chollet's point about "information bottlenecks" (p. 92).

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_311 (Dense)            (None, 1)                 65
_________________________________________________________________
dense_312 (Dense)            (None, 10)                20
=================================================================
Total params: 85
Trainable params: 85
Non-trainable params: 0
_________________________________________________________________
```

Remaining Models

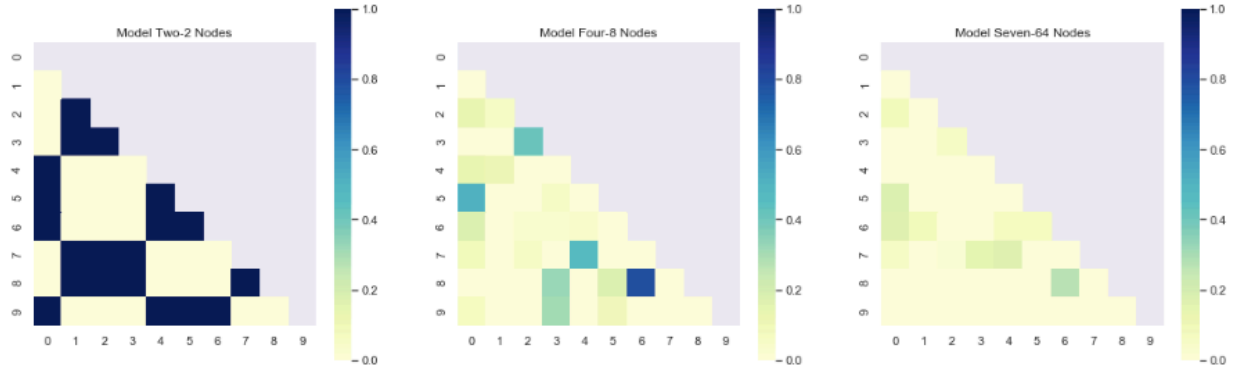
It can be observed that with each subsequent model, the performance continues to get better. However, overfitting becomes more of an issue with fewer epochs in these models. By expanding the number of nodes in the hidden layer, the model is able to learn trends more quickly within the data and thus does not

impose an "information bottleneck" (Chollet, p. 92). The accuracy curves continue to get smoother and smoother. On the basis of number of nodes alone, it is clear that just with one hidden layer, increasing the number of nodes has a significant impact on model performance. The model with 64-nodes (without overfitting) reaches an accuracy of about 95% and has a loss value of about .20.

Weights Exploration

While the output above was helpful for understanding the architecture and performance of each model, the weights of each model will be analyzed in greater detail to provide commentary on what is actually happening behind the scenes. For purposes of simplicity, model two (two nodes), model four (8 nodes) and model seven (64 nodes) are used for comparison to help confirm trends that are observed.

Only the hidden-to-output weights will be considered because these weights can be standardized and compared across models. Also, the hidden to output weights are transposed so that correlations can be calculated using the values of the weights. Otherwise, in their raw form, the correlations would be just of the nodes themselves and not provide any meaningful information. The correlation matrices of the hidden to output weights for the three models are shown below.



From the correlation matrices, the following can be observed:

- In the two-node model, the digits 0,4,5,6,9 are strongly correlated with one another as is the digits 1,2,3,7,8. This observation suggests that the pixels within these images are different.
- In the eight-node model, the pattern becomes more distinct. Here, the digits 6 and 8 have the largest correlation, which may be one cause for misclassification. The strong correlation between 0/5 and 4/7 suggest that these numbers have a similar shape and are only distinguished by a few pixels.
- Surprisingly, the 64-node model does not have any strong results. All the correlations are fairly weak, if not nonexistent.

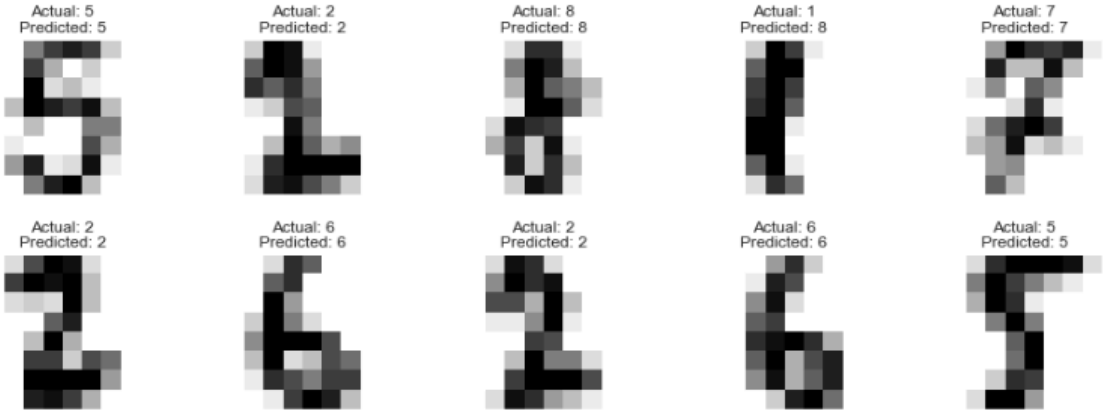
While this weight analysis is not comprehensive, it is revealing. Models with a fewer number of nodes have more greater correlation among weight values from the hidden-to-output layer than do models with a larger number of nodes. This result helps to illustrate that the weights are learning different patterns from the pixels

and responding to certain attributes in order to classify the digits appropriately. As a model increases the number of nodes, this information is more spread out. With a larger number of nodes, each node is capturing a different/unique piece of information, contributing to less correlation among the weights.

Results

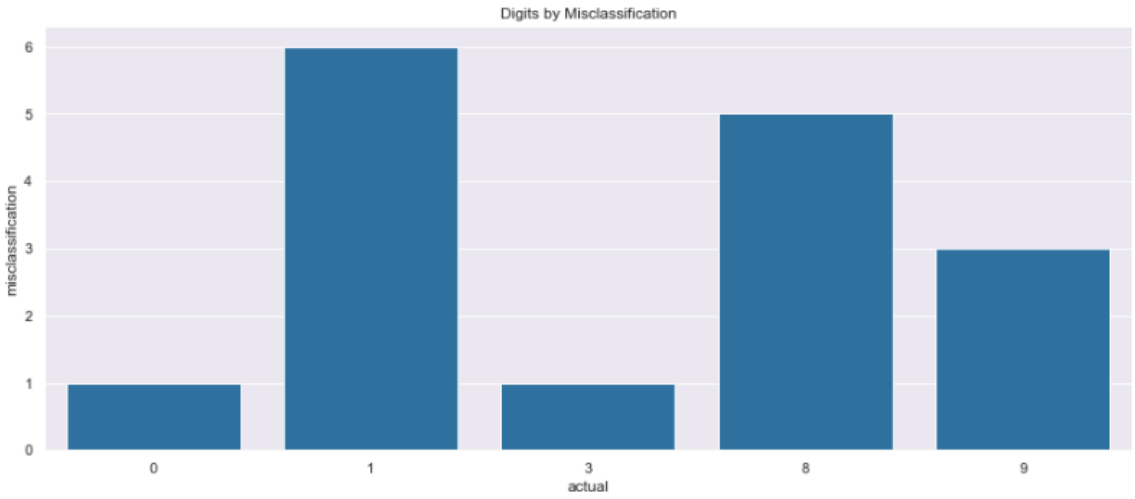
Before commentary can be provided on what was learned, a final step in the modeling process is to evaluate and select the final model. Given that several models began overfitting after a certain number of epochs, all models, except models one and two, are refit using a smaller number of epochs and then evaluated on the test set. The model with 64 nodes is selected as the final model as it has the highest overall accuracy on the test set, at 95.5%.

For a digit classification problem, a helpful exercise is to examine the output of the final model. Below is a sample of the first ten digits from the test set, along with their actual and predicted values.



From these first ten digits, this model misclassifies one value, which is expected given the model's overall accuracy of 95.5%. More importantly, the misclassification occurs for a value that is supposed to be a one but is predicted to be an eight. Eight is also included in this sampling, and it is clear that the images of ones and eights are similar to one another, which is not expected.

As a final step, the misclassifications are examined to help discern future improvements for this model.



The plot above helps to show that ones and eights are the more common misclassified digits. Furthermore, the table below showcases which predictions are made for each misclassification. For instance, it can be observed that if an eight is misclassified, the majority of the time the prediction shows a one. This information could be used to help augment the training set with additional examples for certain classes.

	actual	predicted	misclassification
0	0	4	1
1	1	5	1
2	1	8	2
3	1	9	3
4	3	2	1
5	8	1	4
6	8	7	1
7	9	4	1
8	9	7	1
9	9	8	1

In summary, this assignment developed seven convolutional neural network models using a single hidden layer, focusing on how the number of nodes within the hidden layer affected model performance. It was observed that model accuracy improved as the number of nodes increased. However, overfitting was a problem that had to be addressed by reducing the number of epochs.

More importantly, some initial analysis was performed on the relationship between the weights of each digit within the hidden-to-output layer. The results suggest that as the number of nodes increase, there is less correlation between the weight values for each digit. In short, it appears that the weights share more information when there are less nodes, supporting Chollet's statement that the weights are the network's knowledge (p. 58).

Conclusions

Given the results observed above, this exercise has proved to be informative for future digit classification models.

First, building a model for handwritten digits can be challenging because of the variability in the data. More time should be spent up front inspecting each image and understanding the range of values. For instance, plotting the average pixel depth for each pixel for each digit could help point out standard data patterns and thus help hone in on atypical images that should be analyzed further.

While an MLP model, a neural network with a single hidden layer, achieved a fairly strong performance, it would be worthwhile to consider a deeper architecture with more hidden layers. By stacking

multiple layers together, the network will be able to learn different patterns from the data and thus achieve a higher accuracy. Statistical power is very important for this type of problem.

Additionally, k-fold cross validation should be used to build any digit classification model, as this step is most helpful in selecting and validating the appropriate hyperparameters. Because the focus of this assignment was just on the nodes, k-fold cross validation was omitted. However, considering other values for hyperparameters is needed to ensure that the optimal model is chosen. As part of this process, a strategy for addressing overfitting will be needed. Building in appropriate time for model training is key.

Furthermore, it would be a worthwhile exercise to render images from the model themselves. That way, predicted and actual images for each digit class could be compared to one another. This exercise, along with some of the graphical output shown above, can help to reveal where a model might not be performing as well. Data augmentation may be a helpful next step in improving model accuracy, as this process introduces more relevant data for training. Another avenue may be to consider a more specialized model that helps to focus on smaller areas of an image that are used to separate digit classes (that are more similar to one another).

All in all, it is clear that a digit classification exercise requires thoughtful inspection of the data and the model results, since the goal is to develop a highly accurate model.

Sources

Chollet, F. (2018). Deep Learning with Python. Shelter Island: Manning.

How to Visualize a Deep Learning Neural Network Model in Keras. (2017, September 26). Retrieved from https://machinelearningmastery.com/visualize-deep-learning-neural-network-model-keras/

Ujjwalkarn. (2016, August 10). A Quick Introduction to Neural Networks. Retrieved from https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/

Appendix

Below is a diagram that details the MLP architecture for a problem with two classes/output values (Ujjwalkarn).

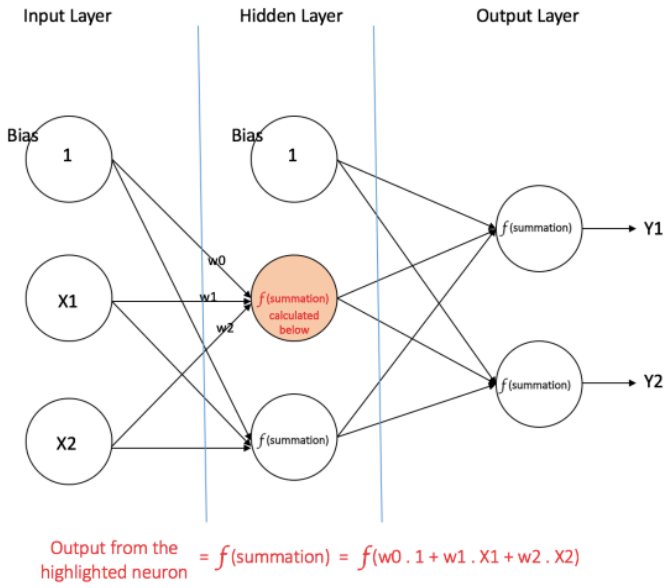
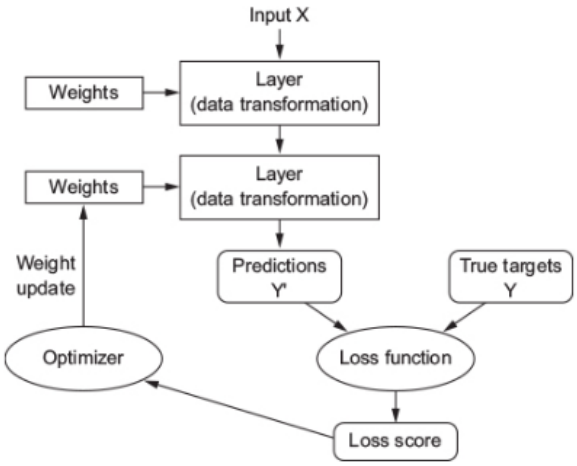


Figure 4: a multi layer perceptron having one hidden layer

From the diagram above, we can better understand the following:

- Fully connected means that each neuron is connected to the neuron in the next layer.
- The weights are integral to the model, as each neuron has a weight associated to it, which is then summed together in the next layer using an activation function.
- Bias nodes are present in the input and hidden layers and are used to help manipulate output to their desired value, or range.

From the diagram below (Chollet p.58), we can better understand the role of the loss function and optimizer, as this diagram shows at a high level the backpropagation algorithm.



At first pass, a fully connected convolutional neural network model uses input values and weights to obtain an output value that is then transformed (via an activation function) for the subsequent layer. The input values in this context represent each individual pixel within an image. Once the model obtains a set of predictions, it then compares the quality of those predictions against the actual values.

The weights of the neural network model, including the input-to-hidden weights and hidden-to-output weights, are then adjusted through this backpropagation algorithm, starting with the top most layer. The backpropagation algorithm chains together the derivatives at each node to better understand a given weight's contribution to the total loss value. This information helps to inform how the weights are adjusted, since the broader goal of the model is to reduce the error via the loss function. For greater explanation on this process, please consult the Chollet textbook.