**CLASSYFYING DIGIT IMAGES (MNIST)**

**Classifying Digit Images (MNIST)**

CS584 Homework 2

Edvin Beqari and Doug Williamson

GMU  October 18, 2020

This project required the implementation of three different Neural Network models using the Keras library within Python. Keras is an open-source library designed to be user-friendly, extensible, and modularto aid in rapid neural network model development and iteration. First released in 2015, Keras was integrated into the Tensorflow suite in 2017. It is currently one of the more popular neural network APIs available. 1

**Cross Validation with Grid Search**

Although this report focuses on models constructed with Keras and Tensorflow, another free library called Scikit-learn also known as “sklearn” deserves some recognition. This tool offers various machine learning algorithms for regression and classification as well as statistical and searching algorithms such as cross validation and parameter grid-search which are required features for this assignment. The library is built on top of other powerful libraries, example NumPy and SciPy similar to Tensorflow; hence, it provides good compatibility in the developing environment. The best way to maximize the use of this library is by implementing the sklearn classifier interface as shown in Table 1. All models in this assignment implement this interface.

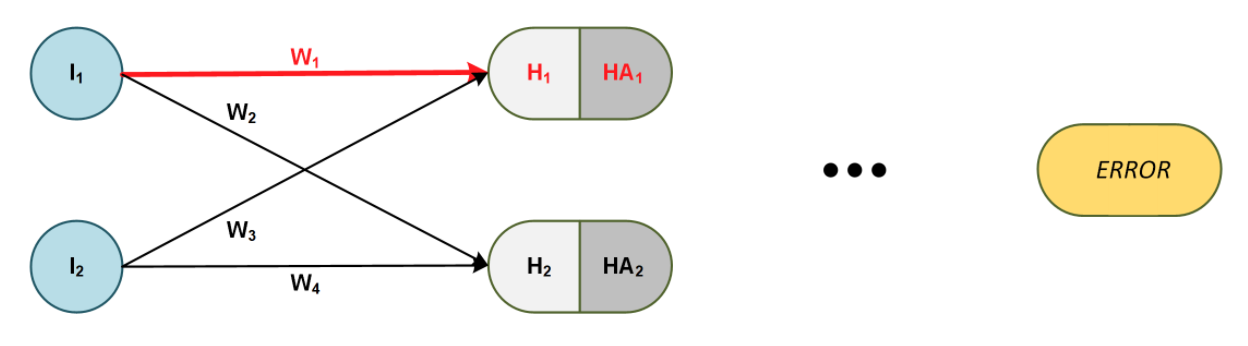
**Table 1:** Scikit-learn classifier interface

|  |
| --- |
| class SKLearnClassifierInterface():  def \_\_init\_\_(self, params):  self.\_params = params    def fit(self, x, y=None): # <-- trains the model  return self  def predict(self, x):  return(prediction) # <-- predicts from the model  def score(self, x, y=None):  return(score) # <-- scores the model from eval  def get\_params(self, deep=True):  return { "params": self.\_params, } # <-- must return a params dictionary  def set\_params(self, params): # <-- must set params same as constructor  self.\_params = params  return self |

**Model 1 – Deep Neural Network**

The base neural network model is the Deep Neural Network (DNN). This model consists of three dense layers with intermediate dropout layer to reduce overfitting and improve learning. The DNN model purpose was to compare the performance of the other Convolutional Neural Networks (CNN) models which typically are standard for computer vision and image recognition.

The DNN is a fully connected layer neural network part of the feed-forward family of graph networks. This means that each node is connected to all the nodes in the next layers. Figure 1, depicts a simplified view of a feed-forward network. The standard method to train the DNN as well CNNs is called backpropagation. One of the most interesting aspect of the this training method is that it shows clearly that these models are simply an application of chained partial derivative besides the many fancy names.



**Figure 1:** Simplified view of a feed-forward network emphasizing the weight w1

**Table 2:** Neural Network parameter definitions

|  |  |
| --- | --- |
|  | Let represent the total error from of all the digit classes (0-9) |
|  | Node input for the deep neural network |
|  | Hidden node for the neural network |
|  | Activated hidden node |
|  | Connection weight between an input-hidden, hidden-hidden, or hidden-output |
|  | Small constant used in gradient decent to reduce the derivative value |
|  | Batch size, i.e.: 128 means that the weights are updated every 128 consecutive pixels |

|  |  |
| --- | --- |
|  | (1) |

Figure 1, omits any possible deep hidden layers and output layer but it is understood that there may be any number of layers in between. Also, a network may be as wide as specified for each layer. During the forward step, the input information is passed though the nodes where it is added and activated to the value of other input nodes. Backpropagation flows in the opposite direction by calculating the error contribution of each output node for each connection weight in the network. For example, *Eq. 1*, shows the application of chain rule to find the error derivative with respect to the first weight of the network. This equation also corresponds with Figure 1 above, where *w1* is highlighted in red.

The next step in backpropagation, after all gradients are calculated, is the weight update. The base algorithm to perform this step is called stochastic gradient descent. The algorithm features a constant eta () which ensures small steps towards the error minimum. Equation 2 below, captures the basic equation for the algorithm including mini-batch. The later feature averages the gradient for  training samples and updates the weight only for the mini-batch size during the training loop.

The gradient descent in *Eq. 2* it is still used mostly by academics and experimental networks; however, most working models including the models trained in this paper use versions of the algorithm that converge much faster using adaptive learning rates with velocity and momentum.

|  |  |
| --- | --- |
|  | (2) |

For each model in this assignment, the team performed cross validation on the selected parameter to determine the optimal value. For the first DNN model, the following hyper-parameters were tuned with cross-validation and grid-search:

**Table 3:** Hyperparameter search

|  |  |  |
| --- | --- | --- |
| **Parameters** | **Range** | **Best Params** |
| *Epochs* | [5, 10, 15, 20] | 20 |
| *Batch Size* | [26, 32, 64, 128] | 26 |
| *Hidden Nodes* | [[128, 64, 32], [128, 32, 64], [64, 128, 32]] | [128, 64, 32] |
| *Activation Function* | [0.2, 0.5] | 0.2 |

This model achieved an evaluation accuracy of ***0.9824***, for the test data provided with the assignment. Although this is a fair score, it is underperforming by almost one percent from the benchmark models for MNIST dataset.

|  |  |
| --- | --- |
| **Figure 2:** DNN training and evaluation (using test data) model accuracy | **Figure 3:** DNN training and evaluation (using test data) model accuracy |

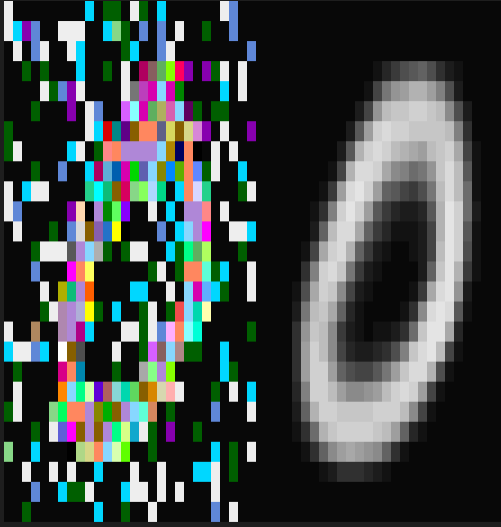
The training graphs, in *Figure 2*, and *Figure 3*, show that the model started to show high bias. The training for this model can be very well stopped at 10 epochs – nonetheless, it is a good illustration of overfitting and is left in the report.

|  |  |
| --- | --- |
| **Figure 4:** Bar chart of DNN model evaluation scores | **Figure 5:** Confusion matrix for DNN multi-class classification |

Additional model evaluation measures such as precision, recall, and Fi Score are shown in *Figure 4*. An import measure is the F1 Score where a perfect model has an F1 Score equal to one. The bar chart therefore shows the digit classes where the model comes up short.

Equivalently important, the confusion matrix in Figure 5, illustrates the amount of training samples that were misclassified for each class, and the number of samples correctly classified shown in the diagonal cells.

Per assignment instructions, all models were also trained with noisy data where sigma () was set to 25. An example sample is shown below in *Figure 6*. No significant improvement was noticed in the model performance; hence, the training results for the noisy sets are omitted from the report.



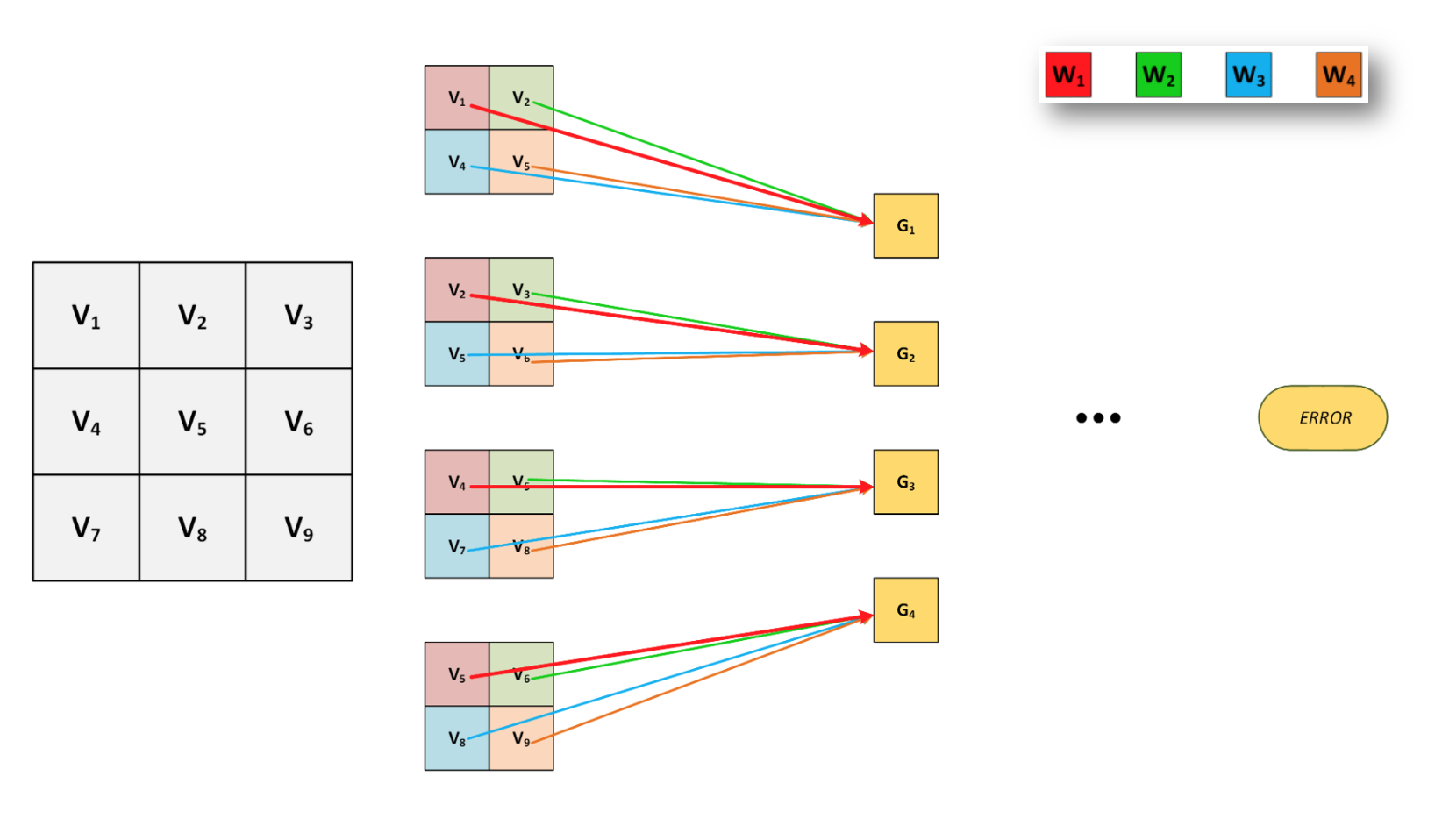
**Figure** **6:** Noisy training data example digit

**Model 2 – Convolutional Neural Network**

For the second model, we selected to implement a pure convolutional neural network. In a CNN model3, an image is converted into a digital matrix, often at the pixel level, and then a filter or kernel is applied via matrix multiplication to calculate the next layer in the network as shown in *Figure 7*. The filter is applied as a sliding window which is passed over every element of the image in strides. The result is the image pixels are aggregated at each level reducing the overall matrix size while highlighting certain features of the original image. This process can be used to detect edges and curves within the image which makes it very effective at handwriting analysis. In other words, the convolutional network is self-engineering features to improve the image classification.

**Table 4:** Convolutional neural networks parameters

|  |  |
| --- | --- |
|  | Let be a pixel in each MNIST image i.e.: one of 28x28 total pixels |
|  | Then is a function of the weighted connections and pixel values. This function is analogous to activated hidden nodes in the regular feed-forward |



**Figure 7:** Simplified view of a convolutional neural network emphasizing the shared weights w1-w4

Convolutional neural networks are trained the same way with backpropagation as the fully connected layer. However, the noticeable distinction here is that CNNs share the connection weights in the resulting hidden nodes ; hence, the word convolution. The partial derivative for an example weight i.e.: *w1* is shown below in *Equation 3*. Note that convolutional layers may and do include activation functions omitted in the example for simplicity.

|  |  |
| --- | --- |
|  | (3) |

As stated earlier in the report, CNNs have been typically outperformed other models in image recognition at least until recently with the transformer models. Kaggle results show that CNNs can achieve score up to 0.***9975*** with the MNIST dataset. Model 2 in this assignment achieved a best evaluation score of ***0.9942*** which is short of the **0.996** team goal to beat Kevin Brensikes’s team***.* Not to be mistaken**,this score was achieved in our third and final model.

|  |  |
| --- | --- |
| **Figure 8:** CNN training and evaluation (using test data) model accuracy | **Figure 9:** CNN training and evaluation (using test data) model accuracy |

|  |  |
| --- | --- |
| **Figure 10:** Bar chart of CNN model evaluation scores | **Figure 11:** Confusion matrix for CNN multi-class classification |

**Table 5:** CNN hyperparameter search

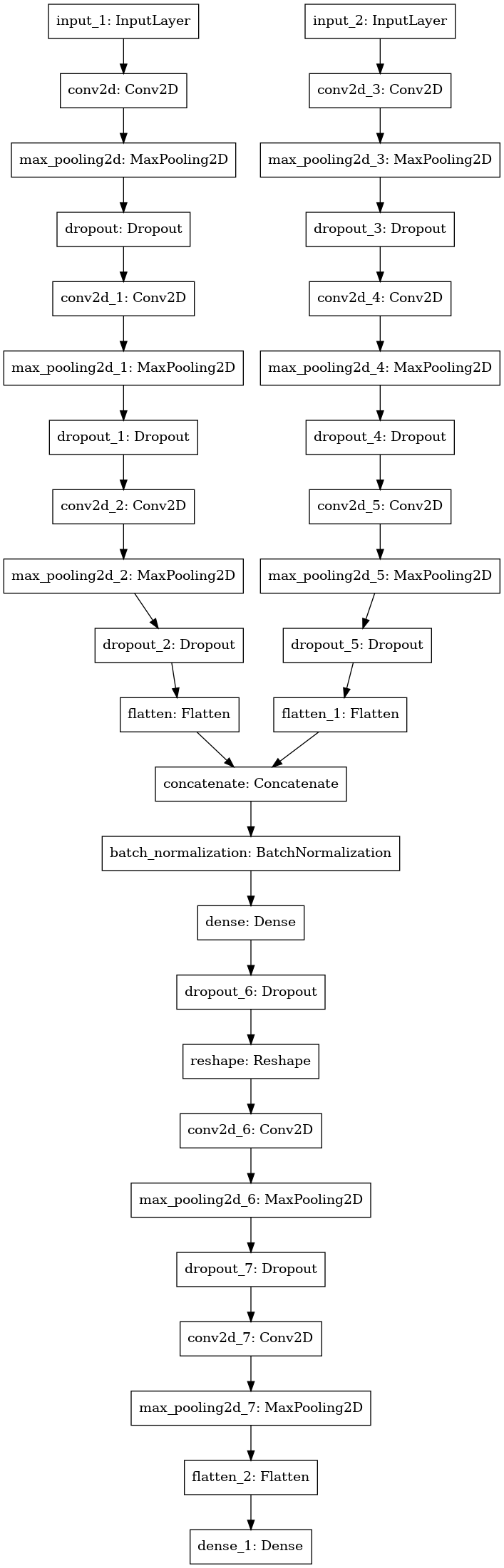
|  |  |  |
| --- | --- | --- |
| **Parameters** | **Range** | **Best Params** |
| *Epochs* | [5, 10, 15, 20] | 20 |
| *Batch Size* | [26, 32, 64, 128] | 128 |
| *Filters* | [[32, 64], [64, 32], [128, 64], [64, 128]] | [64, 128] |
| *Kernel* | [3, 4, 5] | 4 |
| *Pool* | [2, 3], | 2 |
| *Strategy* | ['pool', 'norm'] | 'pool' |
| *Dropout* | [0.2, 0.5] | 0.5 |

**Model 3 – Two Branch Convolutional Neural Network**

Model 3 is a combination of convolutional layers and dense layers. The model also features helper layers such as normalization, dropout, pooling, and concatenation. One of the major differences of this model is that it significantly increases the number of filters by concatenating two branches of CNN layers**.** Each branchcontainsfilters of differentkernels(windows). The model is actually inspired by variational autoencoders although does not compress the information as such models; however, the dense layers are placed between convolutional layers rather than at the end where they are typically found in the literature. This was constructed in the hope that the model will re-create its own interpretation of the training images before is classifies them in the later convolutional layer. The model architecture is shown in detail in *Figure 12****,*** in the next page. The tradeoff between this model and the previous CNN is that by increasing the size of the network we decrease bias but the model may not perform as well in other datasets. However, the team was satisfied with the model performance for this specific task since it achieved the target performance goal of **0.996** on the test set included in the assignment. Due to the size of this network, this model was trained heuristically rather than using cross-validation with grid-search.

|  |  |
| --- | --- |
| **Figure 1:** CNN training and evaluation (using test data) model accuracy | **Figure 2:** CNN training and evaluation (using test data) model accuracy |

|  |  |
| --- | --- |
| **Figure 1:** Bar chart of CNN model evaluation scores | **Figure 2:** Confusion matrix for CNN multi-class classification |

****

**Figure 12:** Two branch CNN architecture

**Table 6:** Two branch CNN params

|  |  |  |  |
| --- | --- | --- | --- |
| *Epochs* | 60 | *Batch Size* | 128 |

**Conclusion**

This assignment covered a lot of ground in the context of deep learning algorithms. Keras and Tensorflow are pretty standard libraries in many real-world deployed models. This exercise also created the right conditions to learn the APIs of other important libraries such as Scikit-Learn. This library is heavily used by machine learning practitioners to structure their algorithms and take advantage of the transformers, classifiers, and pipelines. Three models were implemented and described in the report with varying architectures and performance. The best model achieved an outstanding score of 0.996 which is considered a good score in online competitions.

**How to Save a Model**

|  |
| --- |
| In a terminal, type the following command:  python3 pa2PreBest.py --training\_x MNISTXtrain1.npy --training\_y MNISTytrain1.npy --outModel two\_branch\_model  **Note:** This command will train the model with the training data in the params, and save the model in the same location as the .py file in the **SavedModel** format which is the Tensorflow recommended format.  **Filenames:**  m1.h5 ~ dnn\_model  m2.h5 ~ cnn\_model  m3.h5 ~ two\_branch\_model  **The best model** (two\_branch\_model) **is renamed:**  pa2PreBest.py |

**How to Load a Pre-trained Model**

|  |
| --- |
| from tensorflow import keras  model = keras.models.load\_model('<path/to/location>/PA2BestModel.h5)  **Important:** Per Tensorflow documentation, the use of the **old .H5** format is discouraged; hence, the models were saved in the recommended format **SavedModel** format but the filename was still named “PA2BestModel.h5” per assignment requirements. |

**References:**

1. [https://en.wikipedia.org/wiki/ Keras](https://en.wikipedia.org/wiki/BERT_(language_model))
2. <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>
3. <https://people.minesparis.psl.eu/fabien.moutarde/ES_MachineLearning/TP_convNets/convnet-notebook.html>