Deep Learning for Image Classification

1 Background

Deep learning first started to receive widespread attention and interest when Krizhevsky, Sutskever, and Hinton won the 2012 ImageNet challenge using a convolutional neural network they called AlexNet. AlexNet achieved error rates 10% lower than other approaches. In 2013, most competitors switched to deep learning models. By 2015, competitors were able to beat the performance of humans. Deep learning continues to outperform previous approaches across a wide range of complex data types including images, text, speech, and genomic data.

In this project, you are going to implement and compare two deep learning models for an image classification problem using the Keras tensorflow API. You are using a data set of 28x28 pixel grayscale images of 10 types of written characters from classical Japanese works. Your resulting models should correctly identify the appropriate class for as many character samples as possible. Once you've implemented, tuned, and evaluated the models, you will reflect on what you learned in a lab report.

2 Instructions

1. **Read the following paper, available in the readings section of canvas**: Tarin Clanuwat, Kitamoto, A., Yamamoto, K., Bober-Irizar, M., Lamb, A., and Ha, D. 2018. "Deep Learning for Classical Japanese Literature." 32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

2. **Download, Load, and Visualize the Data.**

   - Download the following four files using the code in the download_data notebook:

     kmnist-train-imgs.npz
     kmnist-train-labels.npz
     kmnist-test-imgs.npz
     kmnist-test-labels.npz

   - Load the data.
   - Plot 3 examples from 4 of the classes.
   - Reshape the images into 4D tensors of shape $(N, 28, 28, 1)$. The first dimension (N) represents the number of images. Each image is 28x28 pixels. Since the images are grayscale, they only have 1 channel.

- The tensors represent the 256 possible gray colors using integer values (e.g., 0, 1, 2, ..., 255). Convert the tensors to 32-bit floats and divide by 255 to scale the pixels to the range of $[0, 1]$.

3. **Implement a Multi-Layer Perceptron / Fully Connected Feedforward Neural Network as a baseline.**

   - Parameterize your network. Is this a binary or multiclass classification problem? Review the probabilistic loss functions available in Keras and determine which loss function is most appropriate. Review and pick one of the optimizers available in Keras.
   - Implement a neural network using the Keras Sequential Model API with:

     1 input layer
     flatten layer
     2 dense layers of 8 nodes each (reLU)
     1 dense layer of 10 nodes (why?) and a softmax activation function

   - Use the compile() method to build the model with your chosen loss function, optimizer, and metric. Use accuracy for the metric.
   - Use the fit() method to train the model. Start with 10 epochs.
   - Optimize the validation accuracy of the model by trying different choices for the number of training epochs, the number of hidden dense layers (e.g. 0, 1, 2, 3, etc.), the number of neurons in the first two dense layers, and the optimizer.[1] Record the results your experimentation and include them in your lab report.
   - Finally, evaluate your model on the test set. Call the predict() method on your testing set's features. This will return a 2D Numpy array of estimated probabilities. Scikit learn offers several metrics that work on multi-class problems. In this case, use the accuracy_score() and confusion_matrix() functions.

4. **Implement a (small) Convolutional Neural Network.**

   - Parameterize your CNN. Review the convolutional layer and max pooling layer options in the Keras API documentation. In particular, look at the input shapes and determine what type of convolutional and max pooling layers are most appropriate for the dimensionality of your data.
   - Implement a neural network using the Keras Sequential API with:

[1] The optimizer can have a significant impact on the training set (number of epochs) and not all optimizers are best for all networks. Before you change the architecture, you should try the different optimizers. Find the optimizer which converges most quickly for your network. Next, change the number of epochs until you find a value at which the validation accuracies plateau. Lastly, evaluate the number of hidden dense layers and number of nodes in each layer.

1 input layer
2 convolutional layers (reLU
1 local or global max pooling layer with a 2x2 pool size
1 flatten layer (if not using a global max pooling layer)
2 dense layers (reLU) – use optimized parameter choices from the MLP
1 dense layer with 10 nodes (1 for each output class, w/ softmax)

As before, have the model report accuracy as one of the metrics.

- Output a summary of the model architecture using the summary() method.
- Optimize the validation accuracy of the model by trying different choices of:
  The optimizer Number of epochs The number of kernels and kernel sizes (e.g., sizes of 3, 5, 7, etc. steps) With: no max pooling, max pooling, and global max pooling (but not flatten layer)
  Record the results your experimentation and include them in your lab report.
- Evaluate your model's predictions as you did for the MLP.

5. **Lab Report** – write up your analysis in a 2-page report. Your report should address:

   - Background on the data set. What is the MNIST digits data set? Why was a new data set needed to replace it? Why did the authors focus on characters from classical Japanese? What did they hope to achieve?
   - Include the summaries of your optimized DNN and CNN models in the report. Explain how the the number of parameters (variables) for each layer was calculated for each layer based on input sizes. Explain how changes in the hyper-parameters (e.g., number of nodes / kernels) changed the number of parameters (variables).
   - What were the impact of the various parameters on model performance? Which changes increased, decreased, or did not change accuracy? What changes were the most beneficial? (Think of what advice you might give to someone who wanted to do a similar project in terms of using their time most efficiently.)
   - Which model (DNN or CNN) performed better? Why do you think this is?
   - Interpret the confusion matrices. Were the errors balanced across the classes for both models? Were some classes harder to correctly predict than others? If not, why do you think this is? (Consider visualizing some examples of misclassified characters alongside examples from the correct and erroneously predicted classes.)
   - Think about the similarities and differences between project 2 (classical ML) and this project. How are classical ML and deep learning similar and different based on your experience? Which approach do you like most?

Submit your lab report as a PDF and your code as a notebook file in canvas.