CS5621 Machine Learning – Assignment

W.A.H Ranasinghe
209369L

- ★ **GIT-REPO-LINK- https://github.com/hasiniranasinghe/machine-leraning**

1.Develop a deep learning model for image classification.

The convolutional neural network (CNN) is a class of deep learning neural networks.

Here we use the tf layer from tensorflow .The tf.layers module provides a high-level API that makes it easy to construct a neural network. It provides methods that facilitate the creation of dense (fully connected) layers and convolutional layers, adding activation functions, and applying dropout regularization

CNNs contains three components:

- **Convolutional layers**, which apply a specified number of convolution filters to the image. For each subregion, the layer performs a set of mathematical operations to produce a single value in the output feature map. Convolutional layers then typically apply a ReLU activation function) to the output to introduce nonlinearities into the model.

- **Pooling layers**, which downsample the image data extracted by the convolutional layers to reduce the dimensionality of the feature map in order to decrease processing time. A commonly used pooling algorithm is max pooling, which extracts subregions of the feature map (e.g., 2x2-pixel tiles), keeps their maximum value, and discards all other values.

- **Dense (fully connected) layers**, which perform classification on the features extracted by the convolutional layers and downsampled by the pooling layers. In a dense layer, every node in the layer is connected to every node in the preceding layer.

A classic CNN architecture would look something like this:

Input ->Convolution ->ReLU ->Convolution ->ReLU ->Pooling ->ReLU ->Convolution ->ReLU ->Pooling ->Fully Connected

Building the CNN MNIST Classifier

- Keras is a deep learning library built over theano and tensorflow.
- NumPy is the fundamental package for scientific computing with Python.OpenCV is used along with matplotlib just for showing some of the results in the end.

- The core data structure of Keras is a model, a way to organize layers. The simplest type of model is the Sequential model.It is a linear stack of neural network layers for feed forward cnn
- Now we are importing core layers for our CNN network.Dense layer is actually a fully-connected layer.Dropout is a regularization technique used for reducing overfitting by randomly dropping output units in the network.Activation Function is used for introducing non-linearity and Flatten is used for converting output matrices into a vector.The second line is used for importing convolutional and pooling layers.
- keras.utils library will be useful for converting the shape of our data.
- keras.datasets import mnist-MNIST dataset is already available in keras library.So, you don't need to download it separately.

LOAD DATA

MNIST dataset consists of 28x28 size images of handwritten digits.We will load pre-shuffled data into training and testing sets.

Then We need to reshape our data to include number of channels(i.e depth).Since we have grayscale images, no. of channels is equal to 1(for RGB, it is 3).The present shape of our data is (N,H,W) where N = total number of examples or batch size while H and W refer to height and width of the image respectively.

Normalising/Feature scaling

input data for getting all our data in a similar range .It helps in faster convergence of model

For classification of digits, we need 10 classes ranging from 0-9. So, our output set must contain such labels.We can check present shape of our output and even print first 10 of them using

NETWORK CONSTRUCTION

1. add our first convolution layer with 32 kernels of size 3x3. The default stride value is (1,1).The activation function used is ReLu which is defined as max(x,0).Hence,it sets the negative elements to zero..Or Input sample must be fed in (depth,width,height) format model.add(Convolution2D(32, (3, 3), activation='relu', input_shape=(28,28,1)))
2. check the shape of our output from this layer.-(model.output_shape)
3. add more layers. need to define the shape of our input layer for only the first convolution layer.
4. add another convolution layer with 32 kernels of the same size as before with the same activation function.
5. add a pooling layer which is used for downsampling as well as making our model somewhat translation invariant.

6. Add the last layer is dropout.It is used for regularisation by randomly disabling/dropping neurons during the learning phase. Here, 0.25 refers to the probability of keeping output neurons.
7. Lastly add the Full-connected layers to reshape our output into a vector. The first FC layer contains 128 output neurons. The second Fc layer is the main classification layer with 10 output neurons for every one of the 10 digits.

FITTING THE MODEL

To fit our model to training data, we need to define the batch size and number of epochs. Setting verbose = 1 gives an insight into the training process i.e. it displays remaining time, loss and accuracy for each epoch.

Improvement

There are many aspects of the learning algorithm that can be explored for improvement.

- leverage is the learning rate,-such as evaluating the impact that smaller or larger values of the learning rate may have, as well as schedules that change the learning rate during training.
- rapidly accelerate the learning of a model and can result in large performance improvements in batch normalization.
- Batch normalization -can be used after convolutional and fully connected layers. It has the effect of changing the distribution of the output of the layer, specifically by standardizing the outputs. This has the effect of stabilizing and accelerating the learning process.

Increase in Model Depth

- changing the capacity of the feature extraction part of the model or
- changing the capacity or function of the classifier part of the model.

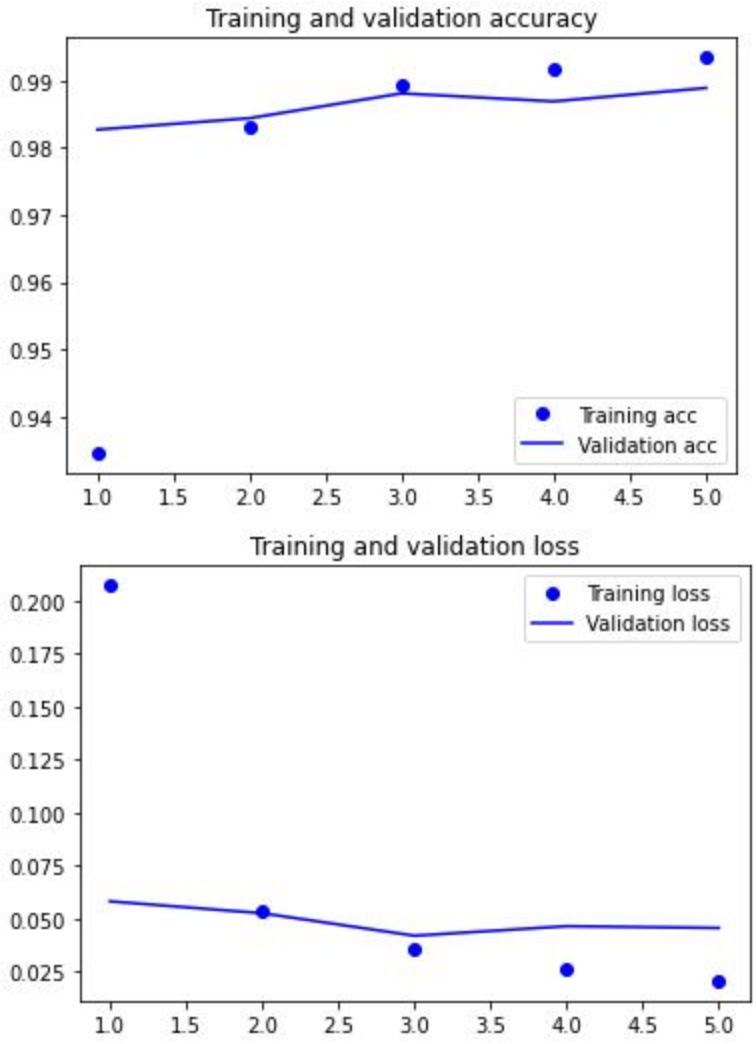
Modal accuracy

```
10000/10000 [==============================] - 3s 277us/step
Accuracy: 0.9890000224113464
Loss:  0.035324207320522695
```

Accuracy of the model at the end of each epoch.

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/5
50000/50000 [==============================] - 43s 854us/step - loss: 0.2071 - accuracy: 0.9346 - val_loss: 0.0582 - val_accuracy: 0.9827
Epoch 2/5
50000/50000 [==============================] - 42s 836us/step - loss: 0.0538 - accuracy: 0.9830 - val_loss: 0.0526 - val_accuracy: 0.9844
Epoch 3/5
50000/50000 [==============================] - 42s 830us/step - loss: 0.0360 - accuracy: 0.9892 - val_loss: 0.0419 - val_accuracy: 0.9881
Epoch 4/5
50000/50000 [==============================] - 41s 826us/step - loss: 0.0264 - accuracy: 0.9918 - val_loss: 0.0464 - val_accuracy: 0.9869
Epoch 5/5
50000/50000 [==============================] - 41s 822us/step - loss: 0.0207 - accuracy: 0.9934 - val_loss: 0.0456 - val_accuracy: 0.9889
```

Here graphs appear to be starting to overfit the data as we get further in, but training and validation sets are pretty close to each other.



2.Add random noise to the training and test datasets and report the accuracy. You may use the following sample code to add random noise.

With the noise the modal accuracy is decreasing .

- ➔ When there is an increased noise factor then accuracy is decreasing.
- ➔ When there is an increased noise factor then loss is increasing.

Noise factor	Accuracy
0	10000/10000 [==============================] - 3s 277us/step Accuracy: 0.9890000224113464 Loss: 0.035324207320522695
0.2	10000/10000 [==============================] - 3s 281us/step Accuracy: 0.9602000117301941 Loss: 0.13461907846480609
0.4	10000/10000 [==============================] - 3s 281us/step Accuracy: 0.9412000179290771 Loss: 0.18220032302886247
0.8	10000/10000 [==============================] - 3s 273us/step Accuracy: 0.9093999862670898 Loss: 0.28015032914876936

Noise factor	Accuracy of the model at the end of each epoch.
0	Train on 50000 samples, validate on 10000 samples Epoch 1/5 50000/50000 [==============================] - 43s 854us/step - loss: 0.2071 - accuracy: 0.9346 - val_loss: 0.0582 - val_accuracy: 0.9827 Epoch 2/5 50000/50000 [==============================] - 42s 836us/step - loss: 0.0538 - accuracy: 0.9830 - val_loss: 0.0526 - val_accuracy: 0.9844 Epoch 3/5 50000/50000 [==============================] - 42s 830us/step - loss: 0.0360 - accuracy: 0.9892 - val_loss: 0.0419 - val_accuracy: 0.9881 Epoch 4/5 50000/50000 [==============================] - 41s 826us/step - loss: 0.0264 - accuracy: 0.9918 - val_loss: 0.0464 - val_accuracy: 0.9869 Epoch 5/5 50000/50000 [==============================] - 41s 822us/step - loss: 0.0207 - accuracy: 0.9934 - val_loss: 0.0456 - val_accuracy: 0.9889

0.2	Train on 50000 samples, validate on 10000 samples Epoch 1/5 50000/50000 [==============================] - 40s 809us/step - loss: 1.3825 - accuracy: 0.5249 - val_loss: 0.5609 - val_accuracy: 0.8160 Epoch 2/5 50000/50000 [==============================] - 40s 801us/step - loss: 0.4509 - accuracy: 0.8612 - val_loss: 0.4284 - val_accuracy: 0.8588 Epoch 3/5 50000/50000 [==============================] - 40s 794us/step - loss: 0.2878 - accuracy: 0.9116 - val_loss: 0.2508 - val_accuracy: 0.9222 Epoch 4/5 50000/50000 [==============================] - 43s 862us/step - loss: 0.1972 - accuracy: 0.9392 - val_loss: 0.1879 - val_accuracy: 0.9421 Epoch 5/5 50000/50000 [==============================] - 40s 804us/step - loss: 0.1473 - accuracy: 0.9542 - val_loss: 0.1400 - val_accuracy: 0.9620
0.4	Train on 50000 samples, validate on 10000 samples Epoch 1/5 50000/50000 [==============================] - 40s 805us/step - loss: 2.1676 - accuracy: 0.2002 - val_loss: 1.1744 - val_accuracy: 0.6392 Epoch 2/5 50000/50000 [==============================] - 40s 791us/step - loss: 0.7135 - accuracy: 0.7757 - val_loss: 0.5408 - val_accuracy: 0.8229 Epoch 3/5 50000/50000 [==============================] - 39s 789us/step - loss: 0.4358 - accuracy: 0.8631 - val_loss: 0.3467 - val_accuracy: 0.8937 Epoch 4/5 50000/50000 [==============================] - 40s 794us/step - loss: 0.3022 - accuracy: 0.9059 - val_loss: 0.2473 - val_accuracy: 0.9234 Epoch 5/5 50000/50000 [==============================] - 40s 796us/step - loss: 0.2301 - accuracy: 0.9275 - val_loss: 0.1898 - val_accuracy: 0.9424
0.8	Train on 50000 samples, validate on 10000 samples Epoch 1/5 50000/50000 [==============================] - 40s 798us/step - loss: 2.1767 - accuracy: 0.1958 - val_loss: 1.2837 - val_accuracy: 0.6115 Epoch 2/5 50000/50000 [==============================] - 40s 793us/step - loss: 0.7777 - accuracy: 0.7529 - val_loss: 0.5969 - val_accuracy: 0.8021 Epoch 3/5 50000/50000 [==============================] - 40s 794us/step - loss: 0.5092 - accuracy: 0.8395 - val_loss: 0.5339 - val_accuracy: 0.8283 Epoch 4/5 50000/50000 [==============================] - 40s 801us/step - loss: 0.3729 - accuracy: 0.8822 - val_loss: 0.2948 - val_accuracy: 0.9082 Epoch 5/5 50000/50000 [==============================] - 39s 790us/step - loss: 0.2781 - accuracy: 0.9125 - val_loss: 0.2769 - val_accuracy: 0.9145

3.Explain how the accuracy of the image classifier can be improved for the scenario where the dataset includes noise as in part 2 above

Noise can be added between hidden layers in the model. Given the flexibility of Keras, the noise can be added before or after the use of the activation function. It may make more sense to add it before the activation; nevertheless, both options are possible.

Conclusions

(1) Noise added during early stage of the model can be better integrated while noise added during late stage of the model tends to cause fluctuation of accuracy;

(2) Complicated neural network models can integrate and absorb noise better than simple neural network models;

(3) Sometimes adding noise can improve not only accuracy but also convergence rate. We hope that this experimental study can provide insights into future design of deep

learning neural network models and machine learning hardwares. Next generation machine learning hardwares can fully exploit the results that

References

1. https://www.pyimagesearch.com/2020/02/24/denoising-autoencoders-with-keras-tensorflow-and-deep-learning/
2. https://www.kaggle.com/c/digit-recognizer
3. https://towardsdatascience.com/convolutional-autoencoders-for-image-noise-reduction-32fce9fc1763