Developing a deep learning model for image classification

We will develop a **Convolutional Neural Network** Model for MNIST handwritten digit classification problem. We used Keras library with a Tensorflow backend for building the model

Name: R.M.Shashikala Ranathunga

Github repo: https://github.com/shashikalaranathunga/Deep-Learning

Q1)

Training-test dataset descriptors

- This dataset has the training set of 60,000 examples and test set of 10,000 examples and that images are indeed square with 28×28 pixels and will download the dataset from Keras itself.

- reshaped x_train and x_test because CNN accepts only 4-D vector.
- pixel values for each image in the dataset are in the range between black and white, or 0 and 255.So, Scale these values to a range of 0 to 1 before feeding them to the convolutional neural network model.This is done first converting the data type from integers to floats then dividing the pixel values by the maximum value

Model design

- Here we define CNN as sequence of layers and used have used sequential model.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1600)              0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               204928
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
=================================================================
Total params: 225,034
Trainable params: 225,034
Non-trainable params: 0
```

- convolutional layer

  Convolutional layer takes in a channel of dimension 1 since the images are grayscaled. The kernel size is chosen to be of size 3x3 with stride of 1. This will extract 26 feature maps using 32 kernelsThe **output dimension** at this layer will be **26x 26 x 32**

- max-pooling layer

  Next used pooling to Reduces the dimensionality of each feature map but retains the most important information.Here used maxpooling. Here we applied RelU activation to it followed by a max-pooling layer with kernel size of 2 .This down-samples the feature maps to dimension of **13x 13 x 32**. The purpose of **ReLU** is to introduce non-linearity in CNN, since most of the real-world data would be non-linear.

- Similarly output from the third layer is 11x11x64 because the second layer was convolutional with 64 filters with kernel size 3x3.

  In this way convolution layers and pooling layers can be added.

- Flattening

  Next, transforms the format of the images from a two-dimensional array to a one-dimensional array.It allows the output to be processed by standard fully connected layers.

- classification(Fully connected layer)

  fully connected layer with 128 neurons and rectifier activation function.

- apply dropout regularization as a solution for overfitting in deep learning

- output layer has 10 neurons for the 10 classes and a softmax activation function to output probability-like predictions for each class.

b) Here are accuracy of the model at the end of the epoch

```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x=x_train,y=y_train, epochs=10)
```

```
Epoch 1/10
60000/60000 [==============================] - 32s 536us/sample - loss: 0.2126 - acc: 0.9362
Epoch 2/10
60000/60000 [==============================] - 14s 236us/sample - loss: 0.0829 - acc: 0.9756
Epoch 3/10
60000/60000 [==============================] - 14s 237us/sample - loss: 0.0586 - acc: 0.9822
Epoch 4/10
60000/60000 [==============================] - 14s 237us/sample - loss: 0.0466 - acc: 0.9860
Epoch 5/10
60000/60000 [==============================] - 15s 252us/sample - loss: 0.0399 - acc: 0.9880
Epoch 6/10
60000/60000 [==============================] - 14s 237us/sample - loss: 0.0347 - acc: 0.9892
Epoch 7/10
60000/60000 [==============================] - 15s 252us/sample - loss: 0.0297 - acc: 0.9909
Epoch 8/10
60000/60000 [==============================] - 14s 236us/sample - loss: 0.0247 - acc: 0.9921 - loss: 0.0225 - acc: 0.9 - ET
A: 8s - loss: 0.0225 - acc: 0 - ETA: 7s - loss: - ETA: 6s - loss: 0.0245 - acc: - ETA: 6
Epoch 9/10
60000/60000 [==============================] - 14s 237us/sample - loss: 0.0234 - acc: 0.9921
Epoch 10/10
60000/60000 [==============================] - 14s 238us/sample - loss: 0.0218 - acc: 0.9929 - loss:
```

```
<tensorflow.python.keras.callbacks.History at 0x2512fba77b8>
```

```
score = model.evaluate(x_test, y_test, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 0.02885303852265261
Test accuracy: 0.9915
```

Q2)

accuracy degraded after adding noise to the training and test dataset.

```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x=x_train_noisy,y=y_train, epochs=10)
```

```
Epoch 1/10
60000/60000 [==============================] - 47s 784us/sample - loss: 1.1881 - acc: 0.5958
Epoch 2/10
60000/60000 [==============================] - 15s 248us/sample - loss: 0.8893 - acc: 0.7054
Epoch 3/10
60000/60000 [==============================] - 15s 255us/sample - loss: 0.8227 - acc: 0.7297 - loss:
Epoch 4/10
60000/60000 [==============================] - 14s 238us/sample - loss: 0.7904 - acc: 0.7412 - lo - ETA: 6s -
Epoch 5/10
60000/60000 [==============================] - 15s 243us/sample - loss: 0.7584 - acc: 0.7496
Epoch 6/10
60000/60000 [==============================] - 14s 238us/sample - loss: 0.7357 - acc: 0.7575
Epoch 7/10
60000/60000 [==============================] - 15s 247us/sample - loss: 0.7150 - acc: 0.7646
Epoch 8/10
60000/60000 [==============================] - 15s 242us/sample - loss: 0.7003 - acc: 0.7693
Epoch 9/10
60000/60000 [==============================] - 15s 245us/sample - loss: 0.6884 - acc: 0.7713 - loss: 0.6890 - ac
Epoch 10/10
60000/60000 [==============================] - 15s 248us/sample - loss: 0.6780 - acc: 0.7734

<tensorflow.python.keras.callbacks.History at 0x232f3d80240>
```

```
score = model.evaluate(x_test_noisy, y_test, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 0.5927135842323303
Test accuracy: 0.8065
```

Q3)

Accuracy of the image classifiers can be improved in scenarios such as dataset set with noises using **autoencoders.**

use convolutional neural networks (convnets) as encoders and decoders. The encoder will consist in a stack of `Conv2D` and `MaxPooling2D` layers (max pooling being used for spatial down-sampling), while the decoder will consist in a stack of `Conv2D` and `UpSampling2D` layers.

When noisy digits fed to the network, and bottom, the digits are reconstructed by the network.

The model converges to a loss of 0.15 after 100 epochs

New model is trained using noise reduced decoded images for the classification and it's accuarcies can be increased .

```
Epoch 1/10
282/282 [==============================] - 9s 31ms/step - loss: 0.8308 - accuracy: 0.7282 - val_loss: 0.4024 - val_accurac
y: 0.8770
Epoch 2/10
282/282 [==============================] - 9s 30ms/step - loss: 0.4318 - accuracy: 0.8610 - val_loss: 0.3350 - val_accurac
y: 0.8950
Epoch 3/10
282/282 [==============================] - 8s 29ms/step - loss: 0.3709 - accuracy: 0.8831 - val_loss: 0.3172 - val_accurac
y: 0.9060
Epoch 4/10
282/282 [==============================] - 8s 29ms/step - loss: 0.3244 - accuracy: 0.8954 - val_loss: 0.2888 - val_accurac
y: 0.9140
Epoch 5/10
282/282 [==============================] - 8s 29ms/step - loss: 0.2905 - accuracy: 0.9059 - val_loss: 0.2896 - val_accurac
y: 0.9060
Epoch 6/10
282/282 [==============================] - 8s 29ms/step - loss: 0.2703 - accuracy: 0.9124 - val_loss: 0.2979 - val_accurac
y: 0.9080
Epoch 7/10
282/282 [==============================] - 8s 29ms/step - loss: 0.2488 - accuracy: 0.9197 - val_loss: 0.2814 - val_accurac
y: 0.9130
Epoch 8/10
282/282 [==============================] - 8s 29ms/step - loss: 0.2357 - accuracy: 0.9212 - val_loss: 0.2671 - val_accurac
y: 0.9160
Epoch 9/10
282/282 [==============================] - 8s 29ms/step - loss: 0.2213 - accuracy: 0.9249 - val_loss: 0.2604 - val_accurac
y: 0.9210
Epoch 10/10
282/282 [==============================] - 8s 29ms/step - loss: 0.2031 - accuracy: 0.9334 - val_loss: 0.2914 - val_accurac
y: 0.9190
```