National University of Singapore
College of Design & Engineering – ECE

EE4400 Data Engineering and Deep Learning
Tutorial 4 – CNN Solutions

Q1. The convolution operation performs feature extraction through a filtering operation on the input values. The kernel or weights of the filter are learnt through back propagation, and different filters are created to extract different features from the input image, at low, medium and high levels in subsequent convolution layers.

Q2. Each filter acts as a receptive field and has its own set of weights/parameters. Each filter slides across the input image looking for certain features. This enables the filter to be invariant to the location of the feature, i.e. the feature can be at any location in the image. The weights/parameters related to a filter and receptive field can be viewed as being shared across the entire image.

Q3. (a) Use the (N-F)/stride+1 formula with N=9, F=5 and stride=2 to get 3x3 pixels.

(b) The input image can be padded with 1 pixel around its 4 borders. Use the (N-F+2P)/stride+1 formula to get 4x4 pixels.

(c) No. of weights: FxF+1=26. Same in both cases.

Q4. Refer to the Colab notebook with results and explanations.

To understand how to obtain the numbers of parameters at each layer, refer to the lecture slides.

EE4400 - Tutorial 4 Q4 Solution

This tutorial is on handwritten digit image classification using a convolutional neural network (CNN).

```
# To determine which version you're using:
!pip show tensorflow
```

```
Name: tensorflow
Version: 2.4.1
Summary: TensorFlow is an open source machine learning framework for everyone.
Home-page: https://www.tensorflow.org/
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: /usr/local/lib/python3.7/dist-packages
Requires: tensorboard, absl-py, numpy, gast, google-pasta, h5py, protobuf, typing-extensions, wrapt, wheel,
Required-by: fancyimpute
```

We shall use the Tensorflow package which contains the Keras sub-package to implement the CNN. The dataset is the MNIST dataset found within Keras. Import the necessary packages.

Refer to https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/

```
## baseline cnn model for mnist
from numpy import mean
from numpy import std
from matplotlib import pyplot
from sklearn.model_selection import KFold
# from tensorflow import keras
# from tensorflow.keras import layers
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.optimizers import SGD
import warnings
```

Write a function to load the training and test datasets. Reshape the inputs to be in a single channel, and convert the outputs to one-hot-encoded format.

```
## load train and test dataset
def load_dataset():
  # load dataset
  (trainX, trainY), (testX, testY) = mnist.load_data()
  # reshape dataset to have a single channel
  trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
  testX = testX.reshape((testX.shape[0], 28, 28, 1))
  # one hot encode target values
  trainY = to_categorical(trainY)
  testY = to_categorical(testY)
  return trainX, trainY, testX, testY
```

Write a function to prepare (convert integers to floats) and scale pixels (so that range is [0,1]) in the input images.

```
## scale pixels
def prep_pixels(train, test):
  # convert from integers to floats
  train_norm = train.astype('float32')
  test_norm = test.astype('float32')
  # normalize to range 0-1
  train_norm = train_norm / 255.0
  test_norm = test_norm / 255.0
  # return normalized images
  return train_norm, test_norm
```

Write a function define_model() to define the CNN model. Set up a CNN network with 16 filters each of size (3,3) followed by a 2D max pooling layer of pooling size (2,2). Add a dense layer of 50 nodes before the output with 10 nodes (i.e., 1 node per digit). Use the SGD optimizer and the categorical cross entropy loss function.

```
## define cnn model
def define_model():
  model = Sequential()
  model.add(Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))
  model.add(MaxPooling2D((2, 2)))
  model.add(Flatten())
  model.add(Dense(50, activation='relu', kernel_initializer='he_uniform'))
  model.add(Dense(10, activation='softmax'))
  # compile model
  opt = SGD(lr=0.01, momentum=0.9)
  model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
  return model
```

Write a function 'evaluate_model(dataX, dataY, n_folds=5)' to evaluate a model through n-fold cross-validation using the sklearn.model_selection function 'KFold()'. The define_model(), model.fit() and model.evaluate() functions are used. The 'evaluate_model()' function returns the 'scores' and 'histories', which are lists of accuracy values and model statistics, respectively.

```
## evaluate a model using k-fold cross-validation
def evaluate_model(dataX, dataY, n_folds=5):
  scores, histories = list(), list()
  # prepare cross validation
  kfold = KFold(n_folds, shuffle=True, random_state=1)
  # enumerate splits
  for train_ix, test_ix in kfold.split(dataX):
    # define model
    model = define_model()
    model.summary()
    # select rows for train and test
    trainX, trainY, testX, testY = dataX[train_ix], dataY[train_ix], dataX[test_ix], dataY[test_ix]
    # fit model
    history = model.fit(trainX, trainY, epochs=10, batch_size=32, validation_data=(testX, testY), verbose=0)
    # evaluate model
    _, acc = model.evaluate(testX, testY, verbose=0)
    print('> %.3f' % (acc * 100.0))
    # stores scores
    scores.append(acc)
    histories.append(history)
  return scores, histories
```

Write a function to plot the learning curves of cross entropy loss and classification accuracy, using 'histories' as input.

```
## plot diagnostic learning curves
def summarize_diagnostics(histories):
  for i in range(len(histories)):
    # plot loss
    pyplot.subplot(2, 1, 1)
```

```
    pyp       p   ( ,  ,  )
    pyplot.title('Cross Entropy Loss')
    pyplot.plot(histories[i].history['loss'], color='blue', label='train')
    pyplot.plot(histories[i].history['val_loss'], color='orange', label='test')
    # plot accuracy
    pyplot.subplot(2, 1, 2)
    pyplot.title('Classification Accuracy')
    pyplot.plot(histories[i].history['accuracy'], color='blue', label='train')
    pyplot.plot(histories[i].history['val_accuracy'], color='orange', label='test')
  pyplot.show()
```

Write a function to summarize the model performance using a box plot of the accuracy values, using 'scores' as input.

```
## summarize model performance
def summarize_performance(scores):
  # print summary
  print('Accuracy: mean=%.3f std=%.3f, n=%d' % (mean(scores)*100, std(scores)*100, len(scores)))
  # box and whisker plots of results
  pyplot.boxplot(scores)
  pyplot.show()
```

Write the main processing steps that calls the functions defined above to do the following steps: (i) load the dataset, (ii) prepare pixel data, (iii) evaluate model, (iv) plot learning curves, and (v) summarize estimated performance. Note: the evaluate_model() function may take 5 to 10 minutes to run since the network and data are fairly large, and a lot of processing is required.

```
# suppress warnings
warnings.filterwarnings("ignore",category=UserWarning)

## main processing

# load dataset
trainX, trainY, testX, testY = load_dataset()
```

```
# prepare pixel data
trainX, testX = prep_pixels(trainX, testX)
```

```
print(trainX.shape)
print(trainY.shape)
print(testX.shape)
print(testY.shape)
```

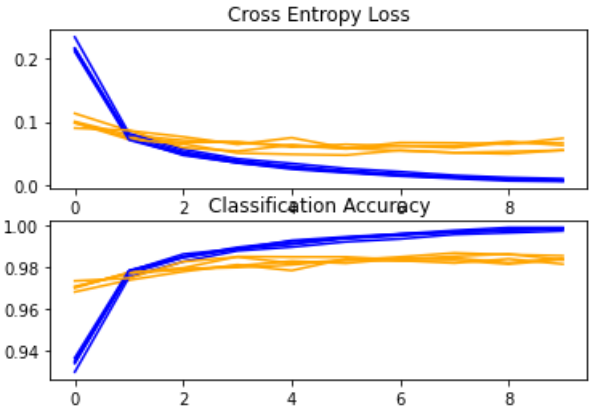
```
(60000, 28, 28, 1)
(60000, 10)
(10000, 28, 28, 1)
(10000, 10)
```

```
# evaluate model
scores, histories = evaluate_model(trainX, trainY)
print(scores)
print(histories)
```

```
Model: "sequential_17"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_27 (Conv2D)           (None, 26, 26, 16)        160
_________________________________________________________________
max_pooling2d_27 (MaxPooling (None, 13, 13, 16)        0
_________________________________________________________________
flatten_17 (Flatten)         (None, 2704)              0
_________________________________________________________________
dense_34 (Dense)             (None, 50)                135250
_________________________________________________________________
dense_35 (Dense)             (None, 10)                510
```

```
=================================================================
Total params: 135,920
Trainable params: 135,920
Non-trainable params: 0
_________________________________________________________________
> 98.383
Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_28 (Conv2D)           (None, 26, 26, 16)        160
_________________________________________________________________
max_pooling2d_28 (MaxPooling (None, 13, 13, 16)        0
_________________________________________________________________
flatten_18 (Flatten)         (None, 2704)              0
_________________________________________________________________
dense_36 (Dense)             (None, 50)                135250
_________________________________________________________________
dense_37 (Dense)             (None, 10)                510
=================================================================
Total params: 135,920
Trainable params: 135,920
Non-trainable params: 0
_________________________________________________________________
> 98.117
Model: "sequential_19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_29 (Conv2D)           (None, 26, 26, 16)        160
_________________________________________________________________
max_pooling2d_29 (MaxPooling (None, 13, 13, 16)        0
_________________________________________________________________
flatten_19 (Flatten)         (None, 2704)              0
_________________________________________________________________
dense_38 (Dense)             (None, 50)                135250
_________________________________________________________________
dense_39 (Dense)             (None, 10)                510
=================================================================
Total params: 135,920
Trainable params: 135,920
Non-trainable params: 0
_________________________________________________________________
> 98.333
Model: "sequential_20"
```

```
# learning curves
summarize_diagnostics(histories)
```




```
# summarize estimated performance
summarize_performance(scores)
```

```
Accuracy: mean=98.333 std=0.129, n=5
```