Activity 2.2 - Transfer Learning

Objective(s):

This activity aims to introduce how to apply transfer learning

Intended Learning Outcomes (ILOs):

- Demonstrate how to build and train neural network
- Demonstrate how to apply transfer learning in neural network

Resources:

- Jupyter Notebook
- CIFAR-10 dataset

Procedures

Load the necessary libraries

```
from __future__ import print_function

import datetime
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
```

Set the parameters

```
now = datetime.datetime.now
batch_size = 128
num_classes = 5
epochs = 5
img_rows, img_cols = 28, 28
filters = 32
pool_size = 2
kernel_size = 3
```

Set how the input data is loaded

```
if K.image_data_format() == 'channels_first':
    input_shape = (1, img_rows, img_cols)
else:
    input_shape = (img_rows, img_cols, 1)
```

- Write a function to include all the training steps.
- Use the model, training set, test set and number of classes as function parameters

```
def train_model(model, train, test, num_classes):
    x_train = train[0].reshape((train[0].shape[0],) + input_shape)
    x_test = test[0].reshape((test[0].shape[0],) + input_shape)
    x_train = x_train.astype('float32')
    x_test = x_test.astype('float32')
    x_train /= 255
    x_test /= 255
    print('x_train shape:', x_train.shape)
    print(x_train.shape[0], 'train samples')
    print(x_test.shape[0], 'test samples')

    # convert class vectors to binary class matrices
    y_train = keras.utils.to_categorical(train[1], num_classes)
    y_test = keras.utils.to_categorical(test[1], num_classes)

    model.compile(loss='categorical_crossentropy',
                  optimizer='adadelta',
                  metrics=['accuracy'])

    t = now()
    model.fit(x_train, y_train,
              batch_size=batch_size,
              epochs=epochs,
              verbose=1,
              validation_data=(x_test, y_test))
    print('Training time: %s' % (now() - t))

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

Shuffle and split the data between train and test sets

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

Create two datasets

- one with digits below 5
- one with 5 and above

```
x_train_lt5 = x_train[y_train < 5]
y_train_lt5 = y_train[y_train < 5]
x_test_lt5 = x_test[y_test < 5]
y_test_lt5 = y_test[y_test < 5]

x_train_gte5 = x_train[y_train >= 5]
y_train_gte5 = y_train[y_train >= 5] - 5
x_test_gte5 = x_test[y_test >= 5]
y_test_gte5 = y_test[y_test >= 5] - 5
```

- Define the feature layers that will used for transfer learning
- Freeze these layers during fine-tuning process

```
feature_layers = [
    Conv2D(filters, kernel_size,
           padding='valid',
           input_shape=input_shape),
    Activation('relu'),
    Conv2D(filters, kernel_size),
    Activation('relu'),
    MaxPooling2D(pool_size=pool_size),
    Dropout(0.25),
    Flatten(),
]
```

Define the classification layers

```
classification_layers = [
    Dense(128),
    Activation('relu'),
    Dropout(0.5),
    Dense(num_classes),
    Activation('softmax')
]
```

Create a model by combining the feature layers and classification layers

```
model = Sequential(feature_layers + classification_layers)
```

Check the model summary

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320       
                                                                 
 activation (Activation)     (None, 26, 26, 32)        0         
                                                                 
 conv2d_1 (Conv2D)           (None, 24, 24, 32)        9248      
                                                                 
 activation_1 (Activation)   (None, 24, 24, 32)        0         
                                                                 
 max_pooling2d (MaxPooling2  (None, 12, 12, 32)        0         
 D)                                                              
                                                                 
 dropout (Dropout)           (None, 12, 12, 32)        0         
                                                                 
 flatten (Flatten)           (None, 4608)              0         
                                                                 
 dense (Dense)               (None, 128)               589952    
                                                                 
 activation_2 (Activation)   (None, 128)               0         
                                                                 
 dropout_1 (Dropout)         (None, 128)               0         
                                                                 
 dense_1 (Dense)             (None, 5)                 645       
                                                                 
 activation_3 (Activation)   (None, 5)                 0         
                                                                 
=================================================================
Total params: 600165 (2.29 MB)
Trainable params: 600165 (2.29 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Train the model on the digits 5,6,7,8,9

```
train_model(model,
            (x_train_gte5, y_train_gte5),
            (x_test_gte5, y_test_gte5), num_classes)
```

```
x_train shape: (29404, 28, 28, 1)
29404 train samples
4861 test samples
Epoch 1/5
230/230 [==============================] - 32s 134ms/step - loss: 1.6115 - accuracy: 0.2078 - val_loss: 1.5917 - val_accuracy: 0.2845
Epoch 2/5
230/230 [==============================] - 26s 114ms/step - loss: 1.5889 - accuracy: 0.2589 - val_loss: 1.5687 - val_accuracy: 0.3843
Epoch 3/5
230/230 [==============================] - 28s 120ms/step - loss: 1.5667 - accuracy: 0.3237 - val_loss: 1.5446 - val_accuracy: 0.5131
Epoch 4/5
230/230 [==============================] - 26s 115ms/step - loss: 1.5451 - accuracy: 0.3868 - val_loss: 1.5187 - val_accuracy: 0.6573
Epoch 5/5
230/230 [==============================] - 26s 114ms/step - loss: 1.5195 - accuracy: 0.4549 - val_loss: 1.4898 - val_accuracy: 0.7338
Training time: 0:02:22.660789
Test score: 1.48978853225708
Test accuracy: 0.7337996363639832
```

Freeze only the feature layers

```
for l in feature_layers:
    l.trainable = False
```

Check again the summary and observe the parameters from the previous model

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 activation (Activation)     (None, 26, 26, 32)        0

 conv2d_1 (Conv2D)           (None, 24, 24, 32)        9248

 activation_1 (Activation)   (None, 24, 24, 32)        0

 max_pooling2d (MaxPooling2  (None, 12, 12, 32)        0
 D)

 dropout (Dropout)           (None, 12, 12, 32)        0

 flatten (Flatten)           (None, 4608)              0

 dense (Dense)               (None, 128)               589952

 activation_2 (Activation)   (None, 128)               0

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 5)                 645

 activation_3 (Activation)   (None, 5)                 0

=================================================================
Total params: 600165 (2.29 MB)
Trainable params: 590597 (2.25 MB)
Non-trainable params: 9568 (37.38 KB)
_________________________________________________________________
```

Train again the model using the 0 to 4 digits

```
train_model(model,
            (x_train_lt5, y_train_lt5),
            (x_test_lt5, y_test_lt5), num_classes)
```

```
x_train shape: (30596, 28, 28, 1)
30596 train samples
5139 test samples
Epoch 1/5
240/240 [==============================] - 11s 43ms/step - loss: 1.5762 - accuracy: 0.2983 - val_loss: 1.5490 - val_accuracy: 0.4678
Epoch 2/5
240/240 [==============================] - 10s 42ms/step - loss: 1.5416 - accuracy: 0.3760 - val_loss: 1.5131 - val_accuracy: 0.5386
Epoch 3/5
240/240 [==============================] - 10s 42ms/step - loss: 1.5089 - accuracy: 0.4409 - val_loss: 1.4783 - val_accuracy: 0.5908
Epoch 4/5
240/240 [==============================] - 10s 42ms/step - loss: 1.4762 - accuracy: 0.4921 - val_loss: 1.4442 - val_accuracy: 0.6375
Epoch 5/5
240/240 [==============================] - 10s 41ms/step - loss: 1.4445 - accuracy: 0.5401 - val_loss: 1.4100 - val_accuracy: 0.6896
Training time: 0:01:22.452439
Test score: 1.410048007965088
Test accuracy: 0.6896283030509949
```

Supplementary Activity

Now write code to reverse this training process. That is, you will train on the digits 0-4, and then finetune only the last layers on the digits 5-9.

```
from __future__ import print_function

import datetime
import keras
from keras.datasets import mnist
from keras.models import Sequential
```

```
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

now = datetime.datetime.now

batch_size = 128
num_classes = 5
epochs = 5

img_rows, img_cols = 28, 28
filters = 32
pool_size = 2
kernel_size = 3

if K.image_data_format() == 'channels_first':
    input_shape = (1, img_rows, img_cols)
else:
    input_shape = (img_rows, img_cols, 1)


def train_model(model, train, test, num_classes):
    x_train = train[0].reshape((train[0].shape[0],) + input_shape)
    x_test = test[0].reshape((test[0].shape[0],) + input_shape)
    x_train = x_train.astype('float32')
    x_test = x_test.astype('float32')
    x_train /= 255
    x_test /= 255
    print('x_train shape:', x_train.shape)
    print(x_train.shape[0], 'train samples')
    print(x_test.shape[0], 'test samples')

    y_train = keras.utils.to_categorical(train[1], num_classes)
    y_test = keras.utils.to_categorical(test[1], num_classes)

    model.compile(loss='categorical_crossentropy',
                  optimizer='adadelta',
                  metrics=['accuracy'])

    t = now()
    model.fit(x_train, y_train,
              batch_size=batch_size,
              epochs=epochs,
              verbose=1,
              validation_data=(x_test, y_test))
    print('Training time: %s' % (now() - t))
    score = model.evaluate(x_test, y_test, verbose=0)
    print('Test score:', score[0])
    print('Test accuracy:', score[1])


(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train_lt5 = x_train[y_train < 5]
y_train_lt5 = y_train[y_train < 5]
x_test_lt5 = x_test[y_test < 5]
y_test_lt5 = y_test[y_test < 5]

x_train_gte5 = x_train[y_train >= 5]
y_train_gte5 = y_train[y_train >= 5] - 5
x_test_gte5 = x_test[y_test >= 5]
y_test_gte5 = y_test[y_test >= 5] - 5

feature_layers = [
    Conv2D(filters, kernel_size,
           padding='valid',
           input_shape=input_shape),
    Activation('relu'),
    Conv2D(filters, kernel_size),
    Activation('relu'),
    MaxPooling2D(pool_size=pool_size),
    Dropout(0.25),
    Flatten(),
]

classification_layers = [
    Dense(128),
    Activation('relu'),
```

```
    Dropout(0.5),
    Dense(num_classes),
    Activation('softmax')
]

model = Sequential(feature_layers + classification_layers)


# train on the digits 0-4
train_model(model,
            (x_train_lt5, y_train_lt5),
            (x_test_lt5, y_test_lt5), num_classes)
```

```
x_train shape: (30596, 28, 28, 1)
30596 train samples
5139 test samples
Epoch 1/5
240/240 [==============================] - 28s 114ms/step - loss: 1.6164 - accuracy: 0.1962 - val_loss: 1.5845 - val_accuracy: 0.2839
Epoch 2/5
240/240 [==============================] - 28s 118ms/step - loss: 1.5725 - accuracy: 0.2881 - val_loss: 1.5355 - val_accuracy: 0.5090
Epoch 3/5
240/240 [==============================] - 29s 121ms/step - loss: 1.5261 - accuracy: 0.3868 - val_loss: 1.4823 - val_accuracy: 0.6525
Epoch 4/5
240/240 [==============================] - 28s 116ms/step - loss: 1.4745 - accuracy: 0.4798 - val_loss: 1.4220 - val_accuracy: 0.7702
Epoch 5/5
240/240 [==============================] - 27s 114ms/step - loss: 1.4184 - accuracy: 0.5577 - val_loss: 1.3527 - val_accuracy: 0.8276
Training time: 0:02:22.642510
Test score: 1.352695345878601
Test accuracy: 0.8275929093360901
```

```
for l in feature_layers:
    l.trainable = False


# finetune only the last layers on the digits 5-9.
train_model(model,
            (x_train_gte5, y_train_gte5),
            (x_test_gte5, y_test_gte5), num_classes)
```

```
x_train shape: (29404, 28, 28, 1)
29404 train samples
4861 test samples
Epoch 1/5
230/230 [==============================] - 10s 43ms/step - loss: 1.5987 - accuracy: 0.2745 - val_loss: 1.5602 - val_accuracy: 0.3691
Epoch 2/5
230/230 [==============================] - 10s 42ms/step - loss: 1.5618 - accuracy: 0.3098 - val_loss: 1.5194 - val_accuracy: 0.4127
Epoch 3/5
230/230 [==============================] - 10s 44ms/step - loss: 1.5235 - accuracy: 0.3628 - val_loss: 1.4796 - val_accuracy: 0.4867
Epoch 4/5
230/230 [==============================] - 10s 45ms/step - loss: 1.4872 - accuracy: 0.4243 - val_loss: 1.4399 - val_accuracy: 0.5894
Epoch 5/5
230/230 [==============================] - 10s 44ms/step - loss: 1.4507 - accuracy: 0.4803 - val_loss: 1.3999 - val_accuracy: 0.6727
Training time: 0:00:50.646911
Test score: 1.3998538255691528
Test accuracy: 0.6727010607719421
```

Conclusion

Through separating different parts of the dataset into different numbers I was able to identify how to raise accuracy on certain parts and improve on their accuracy based on their shape and size.