IBM Introduction to Machine Learning

Deep Learning and Reinforcement Learning

Final Project – Esteban Carboni

Main objectives of the analysis

The model will be focused on Transfer Learning. The main objective of the analysis is to develop some model by keeping the early layers of a pre-trained network and re-training the later layers for our specific application. Fine tuning must be performed using data that is like the pre-trained network aiming to achieve the better accuracy score and using memory resources as efficiently as possible.

Data description

The dataset contains 60.000 grayscale images of 10 fashion categories, along with a test set of 10.000 images. Each image is 28 pixels wide and 28 pixels high. Each category is labeled with a number from 0 to 9. Classes are detailed below:

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

```
np.unique(y_train,
          return_counts=True)
```

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8),
 array([6000, 6000, 6000, 6000, 6000, 6000, 6000, 6000, 6000, 6000],
       dtype=int64))
```

```
np.unique(y_test,
          return_counts=True)
```

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8),
 array([1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000],
       dtype=int64))
```

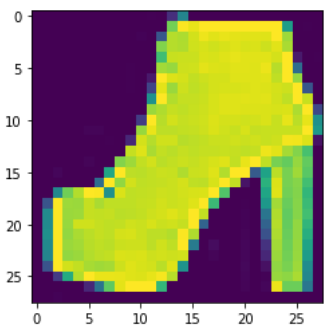
```
x_train[555].shape
```

```
(28, 28)
```

We clearly notice that we have the same number of observations (6000 for the train set and 1000 for the test set) for each of our 10 categories. Also, we can observe the shape of a random image (32x32). Here are some examples of the images and their corresponding label, an ankle boot (9) and a t-shirt (0):

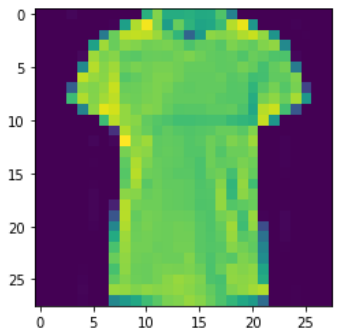
```
print(y_train[44])
plt.imshow(x_train[44])
```

```
9

<matplotlib.image.AxesImage at 0x1d43580d130>
```




```
print(y_train[66])
plt.imshow(x_train[66])
```

```
0

<matplotlib.image.AxesImage at 0x1d4358b1970>
```



Model deployment

First, we set the main parameters and to simplify things, we write a function to include all the training steps. As input, the function takes a model, training set, test set, and the number of classes. Inside the model object will be the state about which layers we are freezing and which we are training. Then, the data is shuffled and split between train and test sets. Next step is to create two datasets: one including half of the classes (T-shirt/top, Trouser, Pullover, Dress and Coat) and another one including the other half (Sandal, Shirt, Sneaker, Bag and Ankle boot). After that, we define the "feature" and “classification” layers. The feature layers are the early layers that we expect will "transfer" to a new problem. We will freeze these layers during the fine-tuning process. The classification layers are the later layers that predict the specific classes from the features learned by the feature layers. This is the part of the model that needs to be re-trained for a new problem. Finally, we create out model by combining the two sets of layers. Here is a summary of our model:

```
# parameters
batch_size = 128
num_classes = 5
epochs = 5
img_rows, img_cols = 28, 28
filters = 32
pool_size = 2
kernel_size = 3
```

```
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 (MaxPooling2D) (None, 12, 12, 32)        0
_________________________________________________________________
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: 600,165
Trainable params: 600,165
Non-trainable params: 0
```

Now, we train our model on the second half of the classes (Sandal, Shirt, Sneaker, Bag and Ankle boot). We notice that accuracy tends to go up and probably can continue to improve.

```
x_train shape: (30000, 28, 28, 1)
30000 train samples
5000 test samples
Epoch 1/5
235/235 [==============================] - 30s 130ms/step - loss: 1.5948 - accuracy: 0.2493 - val_loss: 1.5705 - val_accuracy:
0.3732
Epoch 2/5
235/235 [==============================] - 30s 127ms/step - loss: 1.5608 - accuracy: 0.3275 - val_loss: 1.5316 - val_accuracy:
0.5008
Epoch 3/5
235/235 [==============================] - 30s 127ms/step - loss: 1.5246 - accuracy: 0.3951 - val_loss: 1.4897 - val_accuracy:
0.5526
Epoch 4/5
235/235 [==============================] - 30s 127ms/step - loss: 1.4843 - accuracy: 0.4565 - val_loss: 1.4410 - val_accuracy:
0.5892
Epoch 5/5
235/235 [==============================] - 30s 126ms/step - loss: 1.4332 - accuracy: 0.5191 - val_loss: 1.3826 - val_accuracy:
0.6614
Training time: 0:02:31.040372
Test score: 1.3825602531433105
Test accuracy: 0.6614000201225281
```

We freeze only the feature layers. A lot of the training time is spent "back-propagating" the gradients back to the first layer. Therefore, if we only need to compute the gradients back a small number of layers, the training time is much quicker per iteration. This is in addition to the savings gained by being able to train on a smaller data set.

```
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 (MaxPooling2D) (None, 12, 12, 32)        0
_________________________________________________________________
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: 600,165
Trainable params: 590,597
Non-trainable params: 9,568
```

We can observe above the differences between the number of total params, trainable params, and non-trainable params.

```
x_train shape: (30000, 28, 28, 1)
30000 train samples
5000 test samples
Epoch 1/5
235/235 [==============================] - 10s 42ms/step - loss: 1.6580 - accuracy: 0.2209 - val_loss: 1.6189 - val_accuracy:
0.2194
Epoch 2/5
235/235 [==============================] - 10s 42ms/step - loss: 1.5983 - accuracy: 0.2666 - val_loss: 1.5576 - val_accuracy:
0.3822
Epoch 3/5
235/235 [==============================] - 10s 43ms/step - loss: 1.5500 - accuracy: 0.3302 - val_loss: 1.5089 - val_accuracy:
0.4036
Epoch 4/5
235/235 [==============================] - 10s 42ms/step - loss: 1.5068 - accuracy: 0.3966 - val_loss: 1.4667 - val_accuracy:
0.4656
Epoch 5/5
235/235 [==============================] - 10s 43ms/step - loss: 1.4695 - accuracy: 0.4433 - val_loss: 1.4285 - val_accuracy:
0.5592
Training time: 0:00:50.535936
Test score: 1.4285324811935425
Test accuracy: 0.5591999888420105
```

Note that even though nearly all (590K/600K) of the parameters were trainable, the training time per epoch was still much reduced. This is because the unfrozen part of the network was very shallow, making backpropagation faster.

Now we will make the opposite training process: train on the first half of the classes and finetune only the last layers on the second half of the classes.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
activation_4 (Activation)    (None, 26, 26, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 24, 24, 32)        9248
_________________________________________________________________
activation_5 (Activation)    (None, 24, 24, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 32)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 12, 12, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4608)              0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               589952
_________________________________________________________________
activation_6 (Activation)    (None, 128)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 5)                 645
_________________________________________________________________
activation_7 (Activation)    (None, 5)                 0
=================================================================
Total params: 600,165
Trainable params: 600,165
Non-trainable params: 0
```

```
Epoch 1/5
235/235 [==============================] - 31s 131ms/step - loss: 1.6136 - accuracy: 0.2199 - val_loss: 1.5894 - val_accurac
y: 0.3260
Epoch 2/5
235/235 [==============================] - 30s 127ms/step - loss: 1.5830 - accuracy: 0.3025 - val_loss: 1.5564 - val_accurac
y: 0.3774
Epoch 3/5
235/235 [==============================] - 31s 130ms/step - loss: 1.5511 - accuracy: 0.3670 - val_loss: 1.5219 - val_accurac
y: 0.4422
Epoch 4/5
235/235 [==============================] - 30s 129ms/step - loss: 1.5205 - accuracy: 0.4161 - val_loss: 1.4850 - val_accurac
y: 0.5602
Epoch 5/5
235/235 [==============================] - 30s 129ms/step - loss: 1.4830 - accuracy: 0.4540 - val_loss: 1.4441 - val_accurac
y: 0.6258
Training time: 0:02:32.908443
Test score: 1.4440964460372925
Test accuracy: 0.6258000135421753
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 26, 26, 32)	320
activation_4 (Activation)	(None, 26, 26, 32)	0
conv2d_3 (Conv2D)	(None, 24, 24, 32)	9248
activation_5 (Activation)	(None, 24, 24, 32)	0
max_pooling2d_1 (MaxPooling2	(None, 12, 12, 32)	0
dropout_2 (Dropout)	(None, 12, 12, 32)	0
flatten_1 (Flatten)	(None, 4608)	0
dense_2 (Dense)	(None, 128)	589952
activation_6 (Activation)	(None, 128)	0
dropout_3 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 5)	645
activation_7 (Activation)	(None, 5)	0

Total params: 600,165
Trainable params: 590,597
Non-trainable params: 9,568

```
x_train shape: (30000, 28, 28, 1)
30000 train samples
5000 test samples
Epoch 1/5
235/235 [==============================] - 10s 43ms/step - loss: 1.6277 - accuracy: 0.2085 - val_loss: 1.5997 - val_accuracy:
0.2334
Epoch 2/5
235/235 [==============================] - 10s 42ms/step - loss: 1.5797 - accuracy: 0.2723 - val_loss: 1.5473 - val_accuracy:
0.3328
Epoch 3/5
235/235 [==============================] - 10s 42ms/step - loss: 1.5322 - accuracy: 0.3455 - val_loss: 1.4947 - val_accuracy:
0.4668
Epoch 4/5
235/235 [==============================] - 10s 41ms/step - loss: 1.4871 - accuracy: 0.4222 - val_loss: 1.4434 - val_accuracy:
0.6250
Epoch 5/5
235/235 [==============================] - 10s 42ms/step - loss: 1.4407 - accuracy: 0.4983 - val_loss: 1.3953 - val_accuracy:
0.6988
Training time: 0:00:49.857927
Test score: 1.3953053951263428
Test accuracy: 0.6988000273704529
```

Key findings

We were able to reduce the training time, despite not having achieved the same accuracy results that we had before. Each epoch is moving a lot faster and getting continuous improvement on accuracy score. This was one of our main objectives and it is significant since we could add extra epochs and get improved accuracy while doing it in less time than just running it from the beginning.

Suggestions for next steps

Models should be revisited and consider where to fine tune and how deep to fine tune for achieving better and growing accuracy results. This could be done by trying to hold different parts constant and changing parameters. In addition, the time advantage gained in training time can be exploited. Python notebook code can be found on GitHub: IBM-Introduction-to-Machine-Learning/Deep Learning and Reinforcement Learning Final Project.ipynb at master · estebancarboni/IBM-Introduction-to-Machine-Learning (github.com)