Project Machine Learning

Author: Leonardo Pajer (k-12031)

In Data Science field, every day new machine-learning algorithms are studied and created. My aim in this project is to demonstrate that sometimes less sophisticated algorithms can reach similar level of accuracy (if not better) and are trainable in less time. This to bring back the focus on the dataset-dependence that every algorithm present with every different dataset.

Within my project I will analyse an image-classification problem. The core of my project will be the optimization of my model's hyperparameters via grid-search. To do this I will wrap Keras models in scikit-learn and I will choose the most convenient parameters activation function, neuron numbers, kernel initializer and dropout ratio. The main objective of my project is to show how much models performances vary across nonspecific state-of-the-art algorithms and simple algorithms suited for our specific dataset.

Dataset: Intel Image Classification

This is a dataset about images of natural scenarios. This Data contains around 25k images of size 150x150 distributed under 6 categories. {'buildings' -> 0, 'forest' -> 1, 'glacier' -> 2, 'mountain' -> 3, 'sea' -> 4, 'street' -> 5}**.** As already mentioned, the shape of the images is 150x150 pixel in RGB format. This means that the shape passed to the input layer will be (150,150,3)**.**[1]

Model selection[2]:

My model is a basic convolutional neural network.

> A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of a series of convolutional layers that convolve with a multiplication or other dot product. The activation function is commonly a RELU layer and is subsequently followed by additional convolutions such as pooling layers, fully connected layers and normalization layers, referred to as hidden layers because their inputs and outputs are masked by the activation function and final convolution[3].

I will compare it with a state-of-the-art ResNet convolutional network using the library keras with a model built using Tensorflow beckend. A residual neural network (ResNet) is an artificial neural network (ANN) of a kind that builds on constructs known from pyramidal cells in the cerebral cortex. Residual neural networks do this by utilizing skip connections, or shortcuts to jump over some layers[4].

[1] Link: 3

[2] The architecture of my models is in the appendix otherwise the plot of the ResNet architecture would occupy the next 5 pages.

[3] Source: https://en.wikipedia.org/wiki/ Convolutional_neural_network

[4] Source: https://en.wikipedia.org/wiki/Residual_neural_network

List of optimized parameters:

The parameter that I firstly decided to optimize are the following: Batch size, epochs, activation function, optimization algorithm, weight initialisation, learning rate, dropout regularisation, neurons in hidden layers.

However, I naively did not consider the time required to each step of my optimization. As result, once I tried to execute the first cross validation for the first parameters [batch_size, nr_epochs] I immediately understood that such a process for all the possible values of the parameters that I chose required me one week with my laptop (cpu : Intel Core i7 8th gen, gpu : Nvidia GTX-1080Ti). I consequently decided to run a less accurate cross validation on a subset of the whole train dataset with a reduced number of parameters. Once saw that over 13 epochs the model was overfitting, e.i. the validation score was not increasing while the score of the training set was, I decided to keep the next-model's epochs fixed at 13, with batch size fixed at 100. I ran the same grid search cross validation to test the best activation function, kernel initializer, the dropout rate and finally the number of neurons in the first layer.

Firstly, the result of the activation function's grid search returned an unexpected output: the best one was the sigmoid, immediately followed by softmax, while I was expecting relu instead since it is generally the most reliable and adopted one. In my final model I preferred softmax since in the various try the softmax was always the best one.

```
Best: 0.664054 using {'activation': 'sigmoid'}
0.661917 (0.019319) with: {'activation': 'softmax'}
0.259708 (0.076212) with: {'activation': 'relu'}
0.642679 (0.041871) with: {'activation': 'tanh'}
0.664054 (0.018651) with: {'activation': 'sigmoid'}
0.297114 (0.007033) with: {'activation': 'linear'}
```

Secondly, the kernel initializer has been tested within the list ['uniform', 'normal', 'zero']. The output of this cross validation is the following:

```
Best: 0.668685 using {'init_mode': 'uniform'}
0.668685 (0.017473) with: {'init_mode': 'uniform'}
0.654792 (0.018414) with: {'init_mode': 'normal'}
0.183826 (0.004931) with: {'init_mode': 'zero'},
```

where init_mod is the kernel initializer. As we can see the best result is obtained in the case of a uniform initializer. Additionally, we can see the accuracy rate slightly increasing.

Thirdly, I tested the dropout rate. Dropout rate is defined to randomly, with a probability p, to impede some neurons to pass their output to the next layer. This, since deep neural networks are very good in reach high accuracy in the training test, is used to avoid overfitting. The result of the cross validation is the following:

```
Best: 0.683292 using {'dropout_rate': 0.3}
0.659067 (0.009028) with: {'dropout_rate': 0.2}
0.683292 (0.005457) with: {'dropout_rate': 0.3}
0.666548 (0.004869) with: {'dropout_rate': 0.4}
```

As above, the accuracy is increased. In this case the best dropout rate is 0.3.

Lastly, I tested the number of the neurons present in the first layer within the list [32, 64, 128]. Results of the validation are:

```
Best: 0.674385 using {'neurons': 128}
0.637691 (0.018298) with: {'neurons': 32}
0.659067 (0.004753) with: {'neurons': 64}
0.674385 (0.012229) with: {'neurons': 128}
```

In this case the accuracy did not improve. This is probably since already the number of the neurons in the first layer was set at 128 and this time the training resulted in a slightly worse accuracy. Finally I fitted the same dataset in a ResNet20.

Results:

Once trained my model and optimized its parameters, I fitted the last model in the whole train dataset, and it achieved an accuracy of 85%. The accuracy registered in the test dataset showed that the model is quite consistent.

```
loss: 0.5182
accuracy: 0.8153
```

In fact the accuracy is around 82%.

The ResNet20, even if not optimized, showed surprising results:

```
Test loss: 1.1212866408229512
Test accuracy: 0.7288920555965817
```

The accuracy value would be higher since the final model used was the one generated in the 10^{th} epoch of fitting. For probably randomic reasons, during the 10^{th} epoch the validation accuracy resulted to be way higher than all the other epochs. Moreover, this is the only peak of accuracy in a more generally linear growth of accuracy over epochs, so I tend to exclude the presence of overfitting. Another relevant matter is the training accuracy: the 15^{th} and final epoch showed a training accuracy of *circa* 96%, which is high.

Conclusions:

As I described in the introduction, my aim was to show the dataset dependency that algorithms are obligated to cope with. To do this I assumed that a tuned algorithm, even if simple in terms of complexity, can reach similar accuracy rate of state-of-the-art algorithms. In my case I was working in the image-recognition field and as “simple algorithm” I chose a Convolutional Neural Network, whilst as state-of-the-art algorithm I chose ResNet20, that is a more refined Residual Neural Network. The results of my project display that both algorithms, in first place, did not perform as I expected in the evaluation process. Besides, ResNet outscored the CNN for 15% in the fitting process. In the second hand, the CNN model performed significantly better, i.e. 10% better accuracy

ratio. Finally, considering also the time per sample of the two models required to fit them (<70ms for CNN and circa 130ms for ResNet), at least in my case, a simpler and tuned algorithm demonstrated to be two times quicker and the accuracy ratio registered a sensible difference in CNN's favour. Nonetheless this is only a simple comparison and I am aware that is not 'fair' for what concerns the ResNet's side. With more powerful calculators a more precise and scientific research project can lead to interesting results.

Appendix

1) Convolutional Neural Network architecture

```
Model: "sequential_36"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_72 (Conv2D)           (None, 148, 148, 128)     3584
_________________________________________________________________
max_pooling2d_71 (MaxPooling (None, 74, 74, 128)       0
_________________________________________________________________
conv2d_73 (Conv2D)           (None, 72, 72, 32)        36896
_________________________________________________________________
max_pooling2d_72 (MaxPooling (None, 36, 36, 32)        0
_________________________________________________________________
flatten_35 (Flatten)         (None, 41472)             0
_________________________________________________________________
dropout_35 (Dropout)         (None, 41472)             0
_________________________________________________________________
dense_70 (Dense)             (None, 50)                2073650
_________________________________________________________________
dense_71 (Dense)             (None, 6)                 306
=================================================================
Total params: 2,114,436
Trainable params: 2,114,436
Non-trainable params: 0
```

2) Residual Neural Network: ResNet20

```
__________________________________________________________________________________________________
_____
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
=========
input_1 (InputLayer)            (None, 150, 150, 3)  0
__________________________________________________________________________________________________
_________
conv2d_1 (Conv2D)               (None, 150, 150, 16) 448         input_1[0][0]
__________________________________________________________________________________________________
_________
batch_normalization_1 (BatchNor (None, 150, 150, 16) 64          conv2d_1[0][0]
__________________________________________________________________________________________________
_________
activation_1 (Activation)       (None, 150, 150, 16) 0           batch_normalization_1[0]
[0]
__________________________________________________________________________________________________
_________
conv2d_2 (Conv2D)               (None, 150, 150, 16) 2320        activation_1[0][0]
__________________________________________________________________________________________________
_________
batch_normalization_2 (BatchNor (None, 150, 150, 16) 64          conv2d_2[0][0]
__________________________________________________________________________________________________
_________
```

```
activation_2 (Activation)       (None, 150, 150, 16) 0           batch_normalization_2[0]
[0]
__________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 150, 150, 16) 2320        activation_2[0][0]
__________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 150, 150, 16) 64          conv2d_3[0][0]
__________________________________________________________________________________________
add_1 (Add)                     (None, 150, 150, 16) 0           activation_1[0][0]
                                                                 batch_normalization_3[0]
[0]
__________________________________________________________________________________________
activation_3 (Activation)       (None, 150, 150, 16) 0           add_1[0][0]
__________________________________________________________________________________________
conv2d_4 (Conv2D)               (None, 150, 150, 16) 2320        activation_3[0][0]
__________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 150, 150, 16) 64          conv2d_4[0][0]
__________________________________________________________________________________________
activation_4 (Activation)       (None, 150, 150, 16) 0           batch_normalization_4[0]
[0]
__________________________________________________________________________________________
conv2d_5 (Conv2D)               (None, 150, 150, 16) 2320        activation_4[0][0]
__________________________________________________________________________________________
batch_normalization_5 (BatchNor (None, 150, 150, 16) 64          conv2d_5[0][0]
__________________________________________________________________________________________
add_2 (Add)                     (None, 150, 150, 16) 0           activation_3[0][0]
                                                                 batch_normalization_5[0]
[0]
__________________________________________________________________________________________
activation_5 (Activation)       (None, 150, 150, 16) 0           add_2[0][0]
__________________________________________________________________________________________
conv2d_6 (Conv2D)               (None, 150, 150, 16) 2320        activation_5[0][0]
__________________________________________________________________________________________
batch_normalization_6 (BatchNor (None, 150, 150, 16) 64          conv2d_6[0][0]
__________________________________________________________________________________________
activation_6 (Activation)       (None, 150, 150, 16) 0           batch_normalization_6[0]
[0]
__________________________________________________________________________________________
conv2d_7 (Conv2D)               (None, 150, 150, 16) 2320        activation_6[0][0]
__________________________________________________________________________________________
batch_normalization_7 (BatchNor (None, 150, 150, 16) 64          conv2d_7[0][0]
__________________________________________________________________________________________
add_3 (Add)                     (None, 150, 150, 16) 0           activation_5[0][0]
                                                                 batch_normalization_7[0]
[0]
__________________________________________________________________________________________
activation_7 (Activation)       (None, 150, 150, 16) 0           add_3[0][0]
__________________________________________________________________________________________
conv2d_8 (Conv2D)               (None, 75, 75, 32)   4640        activation_7[0][0]
__________________________________________________________________________________________
batch_normalization_8 (BatchNor (None, 75, 75, 32)   128         conv2d_8[0][0]
```

activation_8 (Activation)	(None, 75, 75, 32)	0	batch_normalization_8[0][0]
conv2d_9 (Conv2D)	(None, 75, 75, 32)	9248	activation_8[0][0]
conv2d_10 (Conv2D)	(None, 75, 75, 32)	544	activation_7[0][0]
batch_normalization_9 (BatchNor	(None, 75, 75, 32)	128	conv2d_9[0][0]
add_4 (Add)	(None, 75, 75, 32)	0	conv2d_10[0][0] batch_normalization_9[0][0]
activation_9 (Activation)	(None, 75, 75, 32)	0	add_4[0][0]
conv2d_11 (Conv2D)	(None, 75, 75, 32)	9248	activation_9[0][0]
batch_normalization_10 (BatchNo	(None, 75, 75, 32)	128	conv2d_11[0][0]
activation_10 (Activation)	(None, 75, 75, 32)	0	batch_normalization_10[0][0]
conv2d_12 (Conv2D)	(None, 75, 75, 32)	9248	activation_10[0][0]
batch_normalization_11 (BatchNo	(None, 75, 75, 32)	128	conv2d_12[0][0]
add_5 (Add)	(None, 75, 75, 32)	0	activation_9[0][0] batch_normalization_11[0][0]
activation_11 (Activation)	(None, 75, 75, 32)	0	add_5[0][0]
conv2d_13 (Conv2D)	(None, 75, 75, 32)	9248	activation_11[0][0]
batch_normalization_12 (BatchNo	(None, 75, 75, 32)	128	conv2d_13[0][0]
activation_12 (Activation)	(None, 75, 75, 32)	0	batch_normalization_12[0][0]
conv2d_14 (Conv2D)	(None, 75, 75, 32)	9248	activation_12[0][0]
batch_normalization_13 (BatchNo	(None, 75, 75, 32)	128	conv2d_14[0][0]
add_6 (Add)	(None, 75, 75, 32)	0	activation_11[0][0] batch_normalization_13[0][0]
activation_13 (Activation)	(None, 75, 75, 32)	0	add_6[0][0]

conv2d_15 (Conv2D)	(None, 38, 38, 64)	18496	activation_13[0][0]
batch_normalization_14 (BatchNo	(None, 38, 38, 64)	256	conv2d_15[0][0]
activation_14 (Activation)	(None, 38, 38, 64)	0	batch_normalization_14[0][0]
conv2d_16 (Conv2D)	(None, 38, 38, 64)	36928	activation_14[0][0]
conv2d_17 (Conv2D)	(None, 38, 38, 64)	2112	activation_13[0][0]
batch_normalization_15 (BatchNo	(None, 38, 38, 64)	256	conv2d_16[0][0]
add_7 (Add)	(None, 38, 38, 64)	0	conv2d_17[0][0] batch_normalization_15[0][0]
activation_15 (Activation)	(None, 38, 38, 64)	0	add_7[0][0]
conv2d_18 (Conv2D)	(None, 38, 38, 64)	36928	activation_15[0][0]
batch_normalization_16 (BatchNo	(None, 38, 38, 64)	256	conv2d_18[0][0]
activation_16 (Activation)	(None, 38, 38, 64)	0	batch_normalization_16[0][0]
conv2d_19 (Conv2D)	(None, 38, 38, 64)	36928	activation_16[0][0]
batch_normalization_17 (BatchNo	(None, 38, 38, 64)	256	conv2d_19[0][0]
add_8 (Add)	(None, 38, 38, 64)	0	activation_15[0][0] batch_normalization_17[0][0]
activation_17 (Activation)	(None, 38, 38, 64)	0	add_8[0][0]
conv2d_20 (Conv2D)	(None, 38, 38, 64)	36928	activation_17[0][0]
batch_normalization_18 (BatchNo	(None, 38, 38, 64)	256	conv2d_20[0][0]
activation_18 (Activation)	(None, 38, 38, 64)	0	batch_normalization_18[0][0]
conv2d_21 (Conv2D)	(None, 38, 38, 64)	36928	activation_18[0][0]
batch_normalization_19 (BatchNo	(None, 38, 38, 64)	256	conv2d_21[0][0]
add_9 (Add)	(None, 38, 38, 64)	0	activation_17[0][0]

```
                                                                 batch_normalization_19[0
][0]
__________________________________________________________________________________________
_________
activation_19 (Activation)      (None, 38, 38, 64)   0           add_9[0][0]
__________________________________________________________________________________________
_________
average_pooling2d_1 (AveragePoo (None, 4, 4, 64)     0           activation_19[0][0]
__________________________________________________________________________________________
_________
flatten_1 (Flatten)             (None, 1024)         0           average_pooling2d_1[0][0
]
__________________________________________________________________________________________
_________
dense_1 (Dense)                 (None, 10)           10250       flatten_1[0][0]
==========================================================================================
=========
Total params: 284,042
Trainable params: 282,666
Non-trainable params: 1,376
__________________________________________________________________________________________
_________
ResNet20v1
Not using data augmentation.
```