Relazione Assignment 3

Gianmaria Balducci

4 Dicembre 2020

Data Preparation

Goal: The assignment consists in the design of a CNN architecture and its training. The CNN has to be designed with the aim of reaching the maximum possible accuracy on the test set, with the hard constraint of a maximum of 7K learnable parameters.
Dataset: Input dataset: MNIST of handwritten digits. This dataset has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.
Preparation: Mnist digits range from 0 to 9, so this is a multiclass classification problem with 10 classes, and the input image is (28 x 28).
First operation is perform a grayscale normalization to reduce the effect of illumination's differences in order to facilitate the train.
Then i reshape all data to 28x28x1 3D matrices for manage the input of the CNN.
Before building the model i checked if there are classes in the label that are unbalanced, but all the classes are quite balanced.
In order to facilitate the classification, label in train set and test set is converted from class vector to binary class matrix with 10 features. This will allow me to use 10 units as output for the last fully-convolutional layer.

Model

The structure of the model is:

```
model = keras.Sequential (
    [
      keras.Input(shape =  input_shape),

      layers.Conv2D(8, kernel_size=(3,3), activation='relu'),
      layers.Conv2D(8, kernel_size=(3,3), activation='relu'),
      layers.MaxPool2D(pool_size=(2,2)),

      layers.Conv2D(16, kernel_size=(3,3), activation='relu'),
      layers.Conv2D(16, kernel_size=(3,3), activation='relu'),
      layers.MaxPool2D(pool_size=(2,2)),

      layers.Flatten(),
      layers.Dropout(0.3),
      layers.Dense(num_classes, activation='softmax'),
    ]
)
model.summary()
```

Figura 1: Neural Network

The input shape of this neural network is 28x28x1 (the original shape of image) that are width, height and one dimension for grayscale. In order to keep the number of parameters under 7k, the model has 2 convolution layers with a bank of 8 filters for each layer, each filter dimension is (3x3x1) with the last dimension that must be one, equal to input image. After this two layers there is a pooling layer with max pooling of 2x2.
Then there are other two convolutional layers with bank of 16 filters for each layer, again each filter dimension is (3x3). There is another pooling layer that is equal to the previous

one (2x2) before flatten layer that flattens dimensions of our data from (4,4,16) to (1, 256). After this i've applied Dropout regularization that keep out 30% of the neurons, with Dropout the model obtain the lowest loss. Finally there is output layer with 10 units that is our target dimension. Activaction function is relu and for the output function i choose *softmax* that compute the probability of categories. I've tried also the sigmoid with worse results.
I've tried different architectures for this model, in particular i built a simpler model than the one described that is faster in training stage but reaches worse performance on the test set. Moreover i built a more complicated model that reaches more or less the same performances on the training set and test set but is slower in the training. Increasing the pool size in pooling layers decrease drastically the number of parameters and therefore also the accuracy. I've tried also to introduce batch normalization in each hidden layer but the performances decreased dramatically (on the test set i had an accuracy around 50-60%). The introduction of another dense layer do not increase model's performance in training and test sets. In the table below are shown the parameters count for each layer as we done in practical lesson.

Input	**Layer**	**Output**	**Prameters**
(28,28,1)	C1: (3x3x1x8)	(26,26,8)	80 = 3x3x1x8+8
(26,26,8)	C2: (3x3x8x8)	(24,24,8)	584
(24,24,8)	p1: max(2x2)	(12,12,8)	0
(12,12,8)	C3: (3x3x8x16)	(10,10,16)	1168
(10,10,16)	C4: (3x3x16x16)	(8,8,16)	2320
(8,8,16)	p2: max(2x2)	(4,4,16)	0
(4,4,16)	Flatten	(1x 256)	0
(1x256)	D1: 10	1x10	2570
			Tot = 6722

Training

For the training i used early stopping regularization which monitors validation loss and stop the training after 5 epochs that do not improve. In the image below are shown the choice of training's hyper-parameters

```
batch_size = 128
epochs = 150

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

network_history = model.fit(x_train, y_train, batch_size=batch_size, callbacks=early_stopping,  epochs = epochs, validation_split=0.2)
```

Figura 2: hyper-parameters

Loss function: I choose *categorical_ crossentropy* because is particular suitable for categorical classification tasks , i've tried *mse, mae* and *sparse categorical cross_ entropy* but with worse results.
Optimizer and learning rate: The choice fell on adam with the default learning rate (**0.001**). Increasing this learning rate speeds up the training phase but take my model to worse performances in terms of accuracy and loss in both training and test set; and

loss decreases less. Decreasing learning rate the model reaches more or less the same performances but after many more epochs so i keep the default value. I've tried to use different optimizer such *SGD* with different learning rate but adam work better in this case.

Epochs: Number of epochs are 150 but the training is stopped by early stopping around 25/35 epochs beacuse the validation loss doesn't improve.

Batch size: for this parameters i went to trial and error and better results are reached by 128. The **metric** for evaluate the training is accuracy. I have not created a variable for validation set but in the function of training i set up validation data to 20% of the training. Below there are the plots of the accuracy and the loss of training and validation sets. After the training i have analyzed the results for understand if my model work properly. And as you can see training loss decrease not too fast and this mean that the model has a good learning rate. Validation loss is a little less than training loss and this is a good sign. The accuracy of the validation set is a little more than accuracy of training and this indicate that the model can generalize well with this type of training.

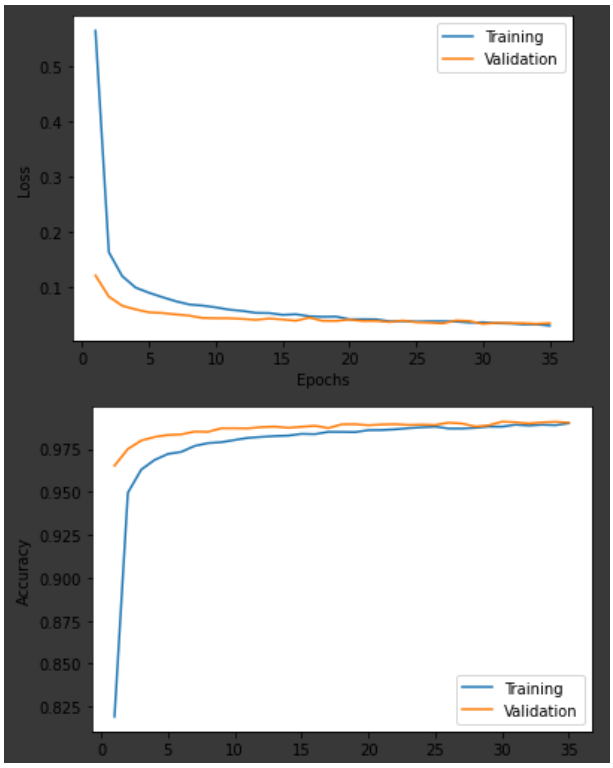

Figura 3: Training and Validation Loss and Accuracy

Classification Performance

Training

Also with the hard constraint of a maximum of 7K learnable parameters the model can reaches 99% of accuracy on the training set and very low loss. These results are also obtained thanks to the very good quality dataset, however the modifications of hyperparameters and the architecture of the network can affect a lot the results.
Classification Performance on the training set:

```
Train loss 0.01787150464951992
Train accuracy 0.9948333501815796
```

Figura 4: Loss and Accuracy on Training set

	precision	recall	f1-score	support
0	1.00	1.00	1.00	5923
1	1.00	1.00	1.00	6742
2	0.99	1.00	0.99	5958
3	1.00	0.99	1.00	6131
4	1.00	1.00	1.00	5842
5	0.99	0.99	0.99	5421
6	1.00	0.99	1.00	5918
7	0.99	1.00	0.99	6265
8	0.99	0.99	0.99	5851
9	0.99	0.99	0.99	5949
micro avg	1.00	0.99	0.99	60000
macro avg	1.00	0.99	0.99	60000
weighted avg	1.00	0.99	0.99	60000
samples avg	0.99	0.99	0.99	60000

Figura 5: Classification Report Training set

As we see the precision on training set in some categories is 1, and also the recall and f1-score, therefore false positive are very low and true positive are very high.

Test

Test set is downloaded directly from keras so i dont' need to split training set. In order to evaluate the model after the training it need to predict the correct labels based on test data and compare them with the true labels of the testset. This process is for evaluate if after training the model is capable to generalize well with unseen data. This model with **6722** parameters obtain around **99%** of accuracy on test set and is a satisfying result. Therefore on the testset model obtain very good performances and the loss is low. Below there is classification report on testset.

```
Test loss 4.779223442077637
Test accuracy 0.9909999966621399
```

Figura 6: Loss and Accuracy on Test set

	precision	recall	f1-score	support
0	0.99	0.99	0.99	980
1	1.00	0.99	0.99	1135
2	0.99	0.99	0.99	1032
3	0.99	0.99	0.99	1010
4	0.99	0.99	0.99	982
5	0.99	0.99	0.99	892
6	1.00	0.99	0.99	958
7	0.99	0.99	0.99	1028
8	0.98	0.99	0.99	974
9	0.99	0.99	0.99	1009
micro avg	0.99	0.99	0.99	10000
macro avg	0.99	0.99	0.99	10000
weighted avg	0.99	0.99	0.99	10000
samples avg	0.99	0.99	0.99	10000

Figura 7: Classification Report Test set

As we see the model can obtain 99% of precision in all the categories of digits, the lower precision is on the category 8 because number eight is more difficult to recognize (is similar to 9), the true difference can be seen with very few learnable parameters where the category 8 has the worse results on precision, recall and f1-score. Other categories and also 8 category have precison recall and f1-score of 99%. This mean that on the test set there are very low false positive and very high true positive. Therefore model obtain very satisfactory results.