#DLUPC

Fine-tuning of the Pooling Layer in a Convolutional Network

Team 3 - Telecommunications Engineering Students
Oriol Barbany, Andreu Bosch, Juanjo Nieto, Sergi Sánchez
{oriol.barbany, andreu.bosch, juan.jose.nieto, sergi.sanchez.deutsch}@alu-etsetb.upc.edu

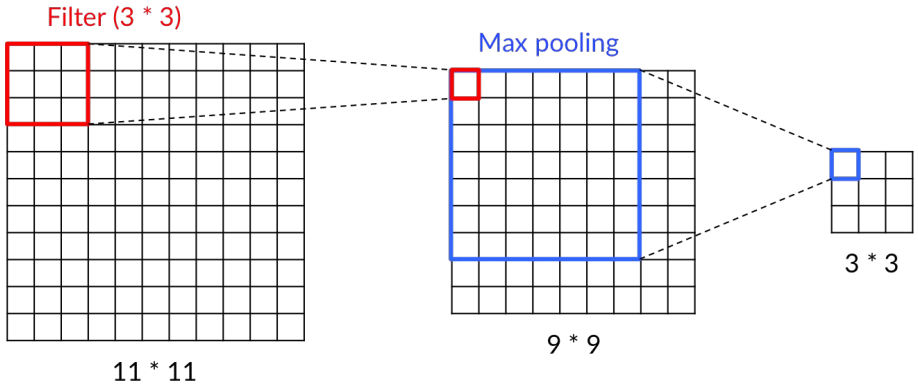
UPC

Outline

1. Pooling Layer
2. Pooling in Keras
3. Problem Statement
4. Results
5. Conclusions
6. References

Pooling Layer [1]

- Makes the **representations smaller** and more manageable for later layers
- Useful to get invariance to small local changes



*Example of a Max Pooling Layer with Strides 2 for each dimension extracted from [2]. Note that convolutional layer don't include padding so there is a dimensionality reduction before pooling.

Pooling in Keras

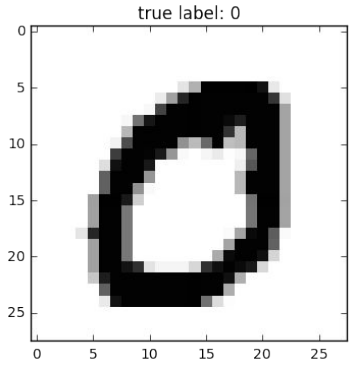
Keras offers different pooling layers [3] based on averaging or maximum values depending on the input shape:

- **Average Pooling** {1D,**2D**,3D}
- **Max Pooling** {1D,**2D**,3D}
- Global Average Pooling {1D,2D}
- Global Max Pooling {1D,2D}

Adding a Pooling Layer to the Model (e.g. Max Pooling 2D):

- `model.add(MaxPooling2D(pool_size=(2, 2)))`

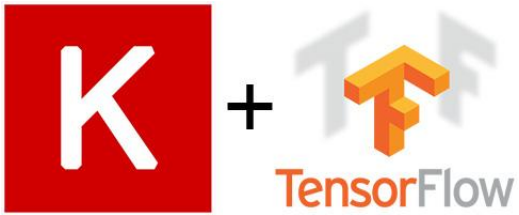
Problem Statement



Classification of handwritten digits belonging to **MNIST database**:

- Grayscale images of size 28x28 pixels
- Digits from 0 to 9 (Multiclass classification problem)
- Train and test databases (we split test to have a **validation dataset** of 20% the whole training)

Solution based on a **Convolutional Neural Network** architecture programmed through Keras library running on top of TensorFlow



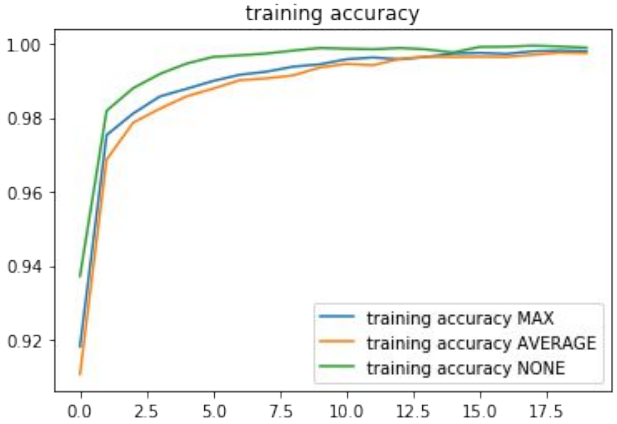
Results

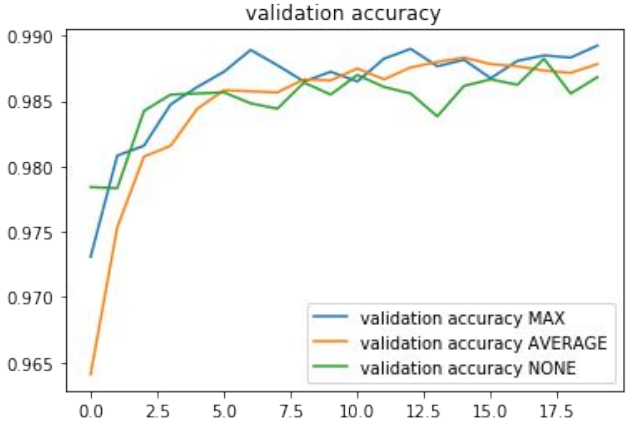
Given the nature of the problem, we studied the performance with all the possible Keras pooling layers:

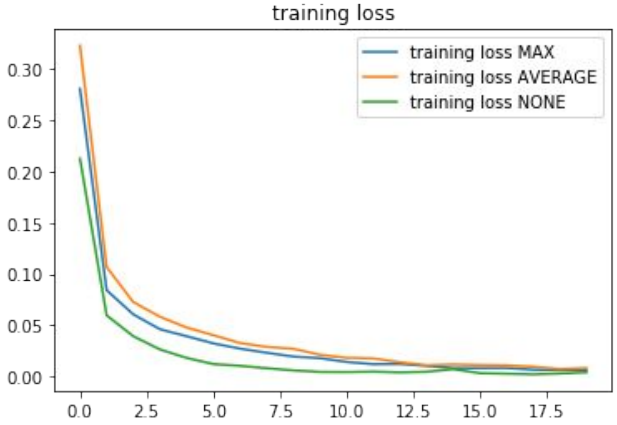
- Max Pooling 2D
- Average Pooling 2D
- No Pooling (fully connected layers)

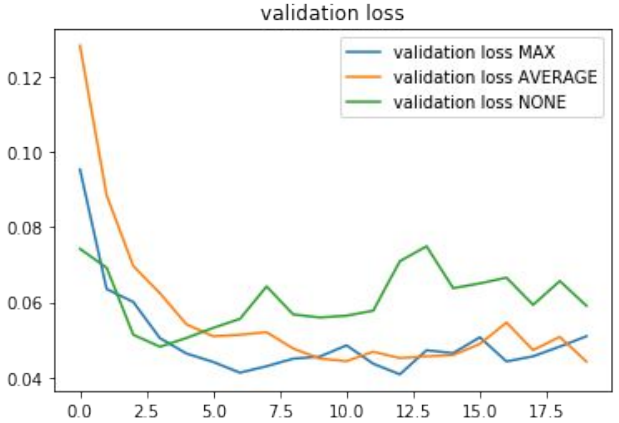
The criteria that we used to evaluate was not only based on **error metrics** (error loss and accuracy) but also on the **run-time** of both when training the model and once it is trained, and the **memory footprint**.

Results


training accuracy
training accuracy MAX
training accuracy AVERAGE
training accuracy NONE


validation accuracy
validation accuracy MAX
validation accuracy AVERAGE
validation accuracy NONE


training loss
training loss MAX
training loss AVERAGE
training loss NONE


validation loss
validation loss MAX
validation loss AVERAGE
validation loss NONE

Results

Baseline error pooling: 1.19%

Baseline error max: 1.19%

Baseline error without pooling: 1.10%

Based on accuracy and loss results over the training dataset, a **Fully Connected Layer** (no Pooling) **works better**

However, **validation error is lower with Averaging and Max Pooling Layers**

This seems a case of **overfitting** due to the increase in the number of input parameters for the next layer when no pooling layer is added (e.g. in the case of strides 2, number of parameters when a pooling layer is added drops by a factor 4)

Nevertheless, baseline error on test dataset is slightly smaller **without Pooling**

Results

The **run-time** depends on the number of computations and its complexity. The best training time was attributed to the **Average Pooling** case while prediction was way faster with **No Pooling**:

- With a fully connected scheme, no extra operations are needed as there is no pooling layer
- However, the lack of dimensionality reduction provided by a pooling layer increases the size of the convolution whose computational complexity is much higher than average and maximum operations

Average pooling: 58.84s / 0.72s

Maximum pooling: 58.22s / 0.71s

Without pooling: 77.24s / **0.70s**

Results

The **memory footprint** depends on the number of parameters of the model and it is estimated using [4]

In the case of having a pooling layer, the number of parameters was determined to drop to a quarter than without pooling in the case of Strides set to 2.

The obtained results show numbers that are very close to the theoretical ones (1.80x0.25=0.45)

Average pooling: 0.46 GB

Maximum pooling: 0.46 GB

Without pooling: 1.80 GB

Conclusions

Importance of tuning the model taking into account the **performance with the validation dataset**:

- Early stopping to prevent overfitting (in our case this is not enough)
- Measure the overall performance in validation to evaluate different classifiers

Importance of dividing the whole database into train, validation and test to evaluate its performance with unseen samples (avoid overfitting and the pumped up results obtained with hyperparameters tuned with the validation dataset)

Sometimes **more parameters doesn't mean better performance** (trade-off)

Conclusions

Best results (taking into account all the previous evaluations) were found with **Max Pooling Layer** but this depends on the nature of each problem and should be tuned in every case

Pooling Layers are specially suitable for low memory devices

Once the model is trained runtime is not a constraint, but if we want to fit a model it is much smaller with Pooling Layer and can be decisive in some cases

References

[1] Verónica Villaplana. Convolutional Neural Networks. Introduction to Deep Learning Course

[2] Yosuke Saito. Easy Implementation of Convolution and Pooling Layer in Deep Learning

[3] Keras Documentation. Pooling Layers

[4] Fabricio Pereira. Algorithm to estimate memory usage of a Keras model

Thanks for your attention

Q&A