Documentation for Algorithms for Massive Datasets project: Turkish lira recognizer

Caccaro Sebastiano
Cavagnino Matteo
A.A.2019/2020

Contents

1 Introduction

The Objective of this project is to build a Turkish Lira banknotes image recognizer through a Convolutional Neural Network. The proposed solution, based on Tensorflow libraries, contains steps to dinamically download the dataset, preprocess the images in it and use the processed images to train a Convolutional Neural Network in recognizing and classifying them. Since the given dataset contains many images and since the request is to classify some precise details of them, it's expected to achieve good results from the proposed solution and in particular from the proposed model.

2 The Turkish Lira banknotes dataset

The chosen dataset <add ref> is originally composed of 6000 images of Turkish Lira banknotes, organized in folders grouping banknotes by their value and already splitted in training and validation set.

2.1 Preprocessing techniques applied to the dataset

In this section the following preprocessing techniques applied to the considered dataset will be discussed:

- image scaling
- training and validation sets creation

Regarding the image scaling, the given images have a size of 720x1280; to not overload the memory of the computing machine it has been reduced by 5 times resulting in a size of 140x256 per image. For the training and validation sets, they have been created starting from two text files, provided with the dataset, listing all the images that needed to be used for the training or validation phases. The training dataset array has then been processed using shuffling, batching and repeating techniques to prepare it for the learning process. The validation dataset array has instead been processed using only batching and repeating; for both the datasets, the batches size has been set to 32 images per batch.

The training dataset has also been processed using the prefetch technique, this allows later elements to be prepared while the current element is being processed. This often improves latency and throughput, at the cost of using additional memory to store prefetched elements.

2.2 Considered algorithms and their implmentation

?

3 Scalability of the proposed solution

The Scalability of this project is granted by (batching, caching, img scaling, ? ...)

4 Experiments and results

Different models have been tested during the developing process; in this section some of those will be shown and the relative results will be discussed.

4.1 Model summary

For each tested model the following data will be reported:

- The NN architecture
- Hyperparameters used
- Data on accuracy for three repeated runs
- Graph of one of the runs
- Comment on the architecture and results

In the layer tables the input layer will not be reported, as it always corresponds to resized image size (`144,256,3`).
Also note that some abbreviations are used in the Layer Config field in order to for the table to fit:

- `k` stands for `kernel size`
- `s` stands for `strides`
- `f` stands for `filters`
- `p` stands for `pool size`
- `r` stands for `rate`

4.2 Models

4.2.1 Baseline Model

Layer Type	**Layer Config**	**Activation**	**Output**	**Params**
Convolution(`Conv2d`)	`k=5, s=3, f=5`	relu	`48,86,5`	`380`
Flatten(`Flatten`)	/	relu	`20640`	`0`
Dense(`Dense`)	`u=64`	relu	`64`	`1321024`
Dense(`Dense`)	`u=6`	softmax	`64`	`390`
			TOTAL	**1,321,794**

Param	**Value**
Batch Size	32
Optimizer	Adam
Base lr	0.001
Epochs	20

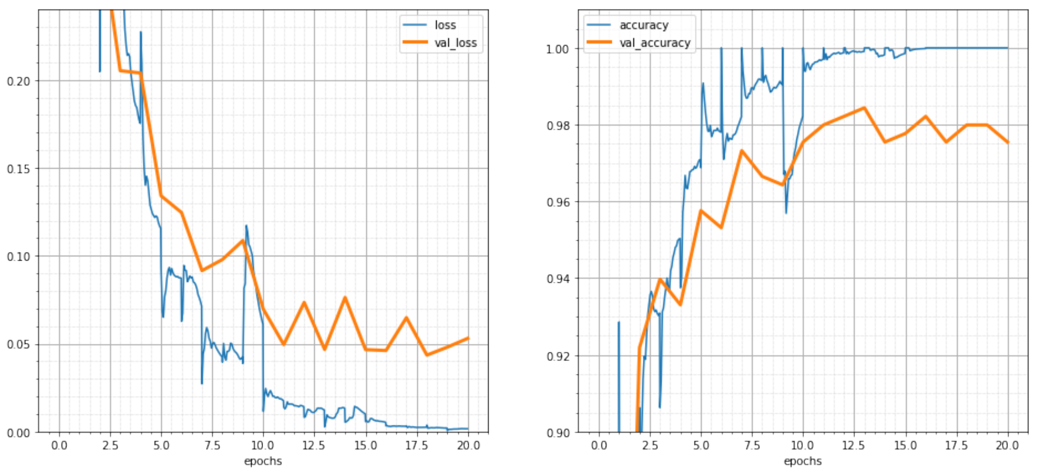


Figure 1: Graph of the first run

Run	**Loss**	**V.Loss**	**Acc.**	**V.Acc.**	**Δ Acc.**
1	0.0016	0.0530	1.0000	0.9754	0.0246
2	0.0012	0.0342	1.0000	0.9821	0.0179
3	0.0060	0.0497	0.9996	0.9866	0.0130
Avg	**0.0029**	**0.0456**	**0.9996**	**0.9814**	**0.0185**

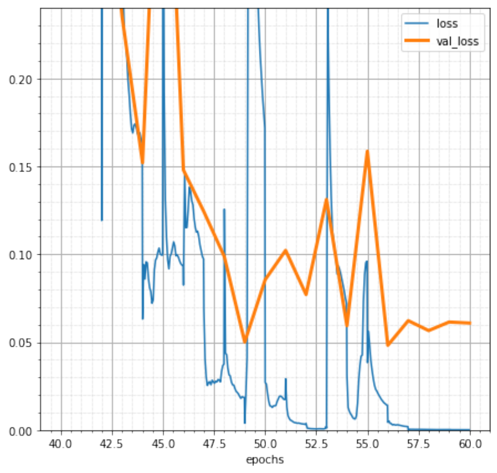
This is a very bare-bone, used to check everything is working in the net and to get a reference for the next iterations. Nevertheless, we can make a few observation:

- There is some overfitting occurring. This is due to the fact the image is only slightly reduced in the convolution, thereby creating the need for a substantial number of parameters between the flat layer and the first dense layer.
- The training basically stall after the 10th epoch, as it has already reach maximum accuracy on the train dataset.

4.2.2 Convolution Model

Layer Type	Layer Config	Activation	Output	Params
Convolution(`Conv2d`)	`k=5, s=3, f=5`	relu	`48,86,5`	`380`
Convolution(`Conv2d`)	`k=5, s=2, f=8`	relu	`24,43,8`	`1008`
Convolution(`Conv2d`)	`k=3, s=1, f=12`	relu	`24,43,12`	`876`
Convolution(`Conv2d`)	`k=3, s=1, f=15`	relu	`24,43,15`	`1635`
Convolution(`Conv2d`)	`k=3, s=1, f=18`	relu	`24,43,18`	`2448`
Flatten(`Flatten`)	/	/	`20640`	`0`
Dense(`Dense`)	`u=64`	relu	`64`	`1188928`
Dense(`Dense`)	`u=6`	softmax	`6`	`390`
			TOTAL	**1,195,665**

Param	Value
Batch Size	32
Optimizer	Adam
Base lr	0.001
Epochs	20



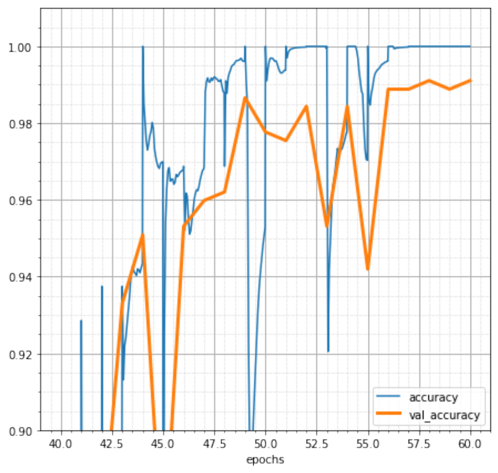


Figure 2: Graph of the third run

Run	**Loss**	**V.Loss**	**Acc.**	**V.Acc.**	**Δ Acc.**
1	1.0908e-04	0.0439	1.0000	0.9866	0.0134
2	8.7761e-05	0.0912	1.0000	0.9888	0.0112
3	1.8083e-04	0.0609	1.0000	0.9911	0.0089
Avg	**1.2589e-04**	**0.0653**	**1.0000**	**0.9889**	**0.0112**

This models is an iteration of the baseline model, achieved by adding four more convolution. The convolution have been added with the following logic in mind:

- The first two convolutions have a stride greater than 1, and that leads to a minimal reductions in the size of the tensor. This in turn should limit the number of parameters needed for training and in turn reduce overfitting.
- The layers are set in way to progressively increase the number of channel and reduce height and width.

The results show the model is slightly less overfitting than the baseline. Both models reached 1.0000 accuracy on the training set, so this translate in an increased validation accuracy.

4.2.3 Convolution and Pooling Model

Layer Type	**Layer Config**	**Activation**	**Output**	**Params**
Convolution(`Conv2d`)	`k=5, s=1, f=5`	relu	`144,256,5`	`380`
MaxPooling(`MaxPooling2D`)	`p=2x2`	/	`72,128,8`	`0`
Convolution(`Conv2d`)	`k=5, s=1, f=8`	relu	`72,128,8`	`1008`
MaxPooling(`MaxPooling2D`)	`p=2x2`	/	`36,64,12`	`0`
Convolution(`Conv2d`)	`k=3, s=1, f=12`	relu	`36,64,12`	`876`
MaxPooling(`MaxPooling2D`)	`p=2x2`	/	`18,32,15`	`0`
Convolution(`Conv2d`)	`k=3, s=1, f=15`	relu	`18,32,15`	`1635`
MaxPooling(`MaxPooling2D`)	`p=2x2`	/	`9,16,18`	`0`
Convolution(`Conv2d`)	`k=3, s=1, f=18`	relu	`9,16,18`	`2448`
Flatten(`Flatten`)	/	/	`2592`	`0`
Dense(`Dense`)	`u=64`	relu	`64`	`165952`
Dense(`Dense`)	`u=6`	softmax	`6`	`390`
			TOTAL	**172,689**

Param	**Value**
Batch Size	32
Optimizer	Adam
Base lr	0.001
Epochs	20

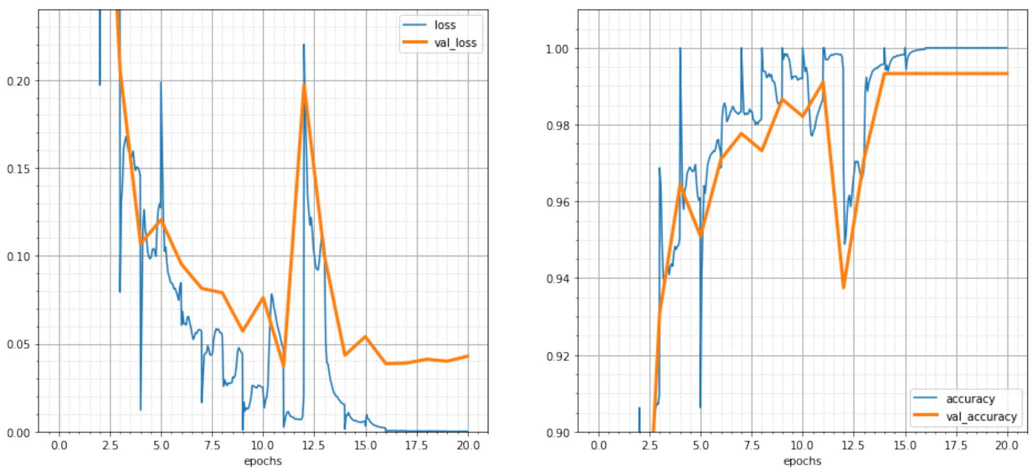


Figure 3: Graph of the first run

Run	**Loss**	**V.Loss**	**Acc.**	**V.Acc.**	**Δ Acc.**
1	1.8558e-04	0.0429	1.0000	0.9933	0.0067
2	6.4439e-04	0.0461	1.0000	0.9866	0.0134
3	1.2224e-04	0.0133	1.0000	0.9933	0.0067
Avg	**3.1740e-04**	**0.0341**	**1.0000**	**0.9911**	**0.0893**

This model applies the use of the Conv-Pool technique in order to convolve and reduce the data at every step. Since the reduction is performed by various MaxPooling layers, the first two convolution have now a stride equal to 1. Pooling helps by reducing the tensor dimension while retaining most of the information. Moreover, the model has a number of parameter which an order of magnitude lower than the previous model. Therefore the model has now little to no overfitting, and has gained a good amount of validation accuracy.

4.2.4 Convolution and Pooling Model with Dropout

Layer Type	Layer Config	Activation	Output	Params
Convolution(`Conv2d`)	`k=5, s=1, f=5`	relu	`144,256,5`	`380`
MaxPooling(`MaxPooling2D`)	`p=2x2`	/	`72,128,8`	`0`
Convolution(`Conv2d`)	`k=5, s=1, f=8`	relu	`72,128,8`	`1008`
MaxPooling(`MaxPooling2D`)	`p=2x2`	/	`36,64,12`	`0`
Convolution(`Conv2d`)	`k=3, s=1, f=12`	relu	`36,64,12`	`876`
MaxPooling(`MaxPooling2D`)	`p=2x2`	/	`18,32,15`	`0`
Convolution(`Conv2d`)	`k=3, s=1, f=15`	relu	`18,32,15`	`1635`
MaxPooling(`MaxPooling2D`)	`p=2x2`	/	`9,16,18`	`0`
Convolution(`Conv2d`)	`k=3, s=1, f=18`	relu	`9,16,18`	`2448`
Dropout(`Dropout`)	`r=0.75`	/	`9,16,18`	`0`
Flatten(`Flatten`)	/	/	`2592`	`0`
Dense(`Dense`)	`u=64`	relu	`64`	`165952`
Dropout(`Dropout`)	`r=0.75`	/	`64`	`0`
Dense(`Dense`)	`u=6`	softmax	`6`	`390`

Param	Value
Batch Size	32
Optimizer	Adam
Base lr	0.001
Epochs	20

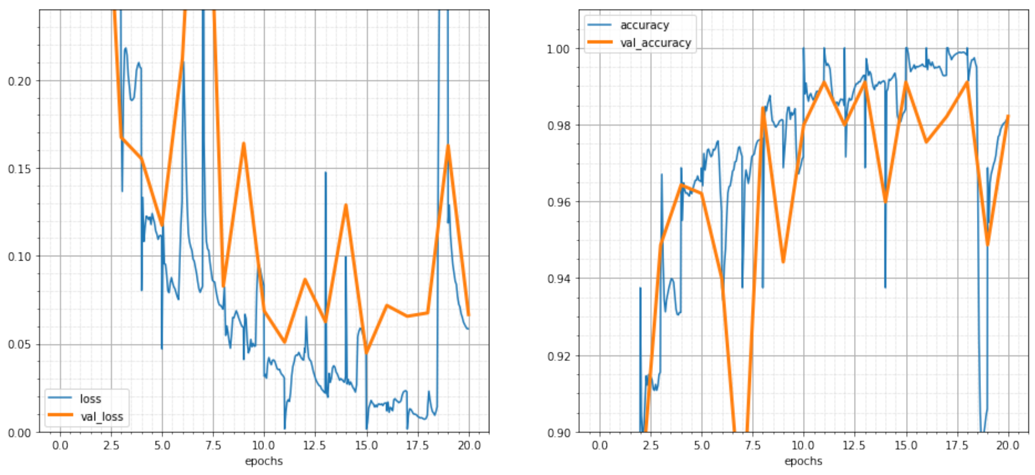


Figure 4: Graph of the first run

Run	Loss	V.Loss	Acc.	V.Acc.	Δ Acc.
1	0.0582	0.0666	0.9817	0.9821	-0.0004
2	0.0063	0.0269	0.9987	0.9933	0.0054
3	0.1172	0.0615	0.9654	0.9866	-0.0212
Avg	**0.0606**	**0.0517**	**0.9820**	**0.9873**	**-0.0054**

4.2.5 Batch Normalization Model

Layer Type	Layer Config	Activation	Output	Params
Convolution(`Conv2d`)	`k=5, s=1, f=5`	/	`144,256,5`	`375`
MaxPooling(`MaxPooling2D`)	`p=2x2`	/	`72,128,8`	0
Batch Norm.(`BatchN.`)	/	/	`72,128,8`	15
	`Relu Activation`			
Convolution(`Conv2d`)	`k=5, s=1, f=8`	/	`72,128,8`	`1000`
MaxPooling(`MaxPooling2D`)	`p=2x2`	/	`36,64,12`	0
Batch Norm.(`BatchN.`)	/	/	`36,64,12`	24
	`Relu Activation`			
Convolution(`Conv2d`)	`k=3, s=1, f=12`	/	`36,64,12`	`864`
MaxPooling(`MaxPooling2D`)	`p=2x2`	/	`18,32,15`	0
Batch Norm.(`BatchN.`)	/	/	`18,32,15`	36
	`Relu Activation`			
Convolution(`Conv2d`)	`k=3, s=1, f=15`	/	`18,32,15`	`1620`
MaxPooling(`MaxPooling2D`)	`p=2x2`	/	`9,16,18`	0
Batch Norm.(`BatchN.`)	/	/	`9,16,18`	45
	`Relu Activation`			
Dropout(`Dropout`)	`r=0.06`	/	`9,16,18`	0
Convolution(`Conv2d`)	`k=3, s=1, f=18`	/	`9,16,18`	`2430`
Flatten(`Flatten`)	/	/	`2592`	0
Batch Norm.(`BatchN.`)	/	/	`2592`	7776
	`Relu Activation`			
Dense(`Dense`)	`u=64`	/	`64`	`165888`
Batch Norm.(`BatchN.`)	/	/	`64`	192
	`Relu Activation`			
Dropout(`Dropout`)	`r=0.06`	/	`64`	0
Dense(`Dense`)	`u=6`	softmax	`6`	`390`
			TOTAL	**180,655**

Param	**Value**
Batch Size	32
Optimizer	Adam
Base lr	0.00005
Epochs	50

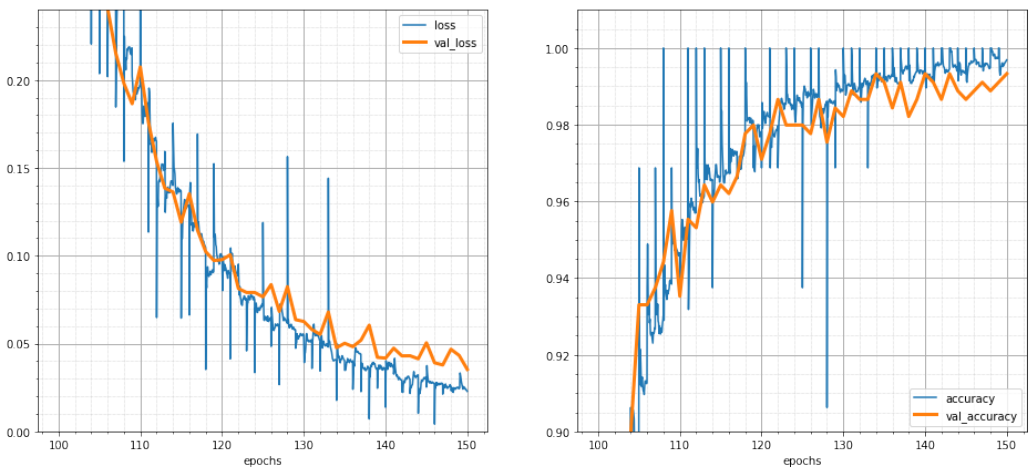


Figure 5: Graph of the third run

Run	**Loss**	**V.Loss**	**Acc.**	**V.Acc.**	**Δ Acc.**
1	0.0276	0.0439	0.9962	0.9933	0.0029
2	0.0255	0.0510	0.9962	0.9888	0.0074
3	0.0227	0.0353	0.9969	0.9933	0.0036
Avg	**0.0253**	**0.0434**	**0.9964**	**0.9918**	**0.0046**

5 Conclusions

As seen in the previous sections, after a few experiments, this project has been succesful in classifying the images of Turkish Lira banknotes with high accuracy and low data loss. (altro?) aggiungere migliori risultati ottenuti -

Esempio di citazione[1]

References

[1] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1993.

[2] Albert Einstein. *Zur Elektrodynamik bewegter Körper*. (German) [*On the electrodynamics of moving bodies*]. Annalen der Physik, 322(10):891–921, 1905.

[3] Knuth: Computers and Typesetting,
`http://www-cs-faculty.stanford.edu/~uno/abcde.html`