Session 2
Discussion Forum 1
Training a Neural Network in KERAS

Organization of DISCUSION FORUMs

The discussion forum are exercises/challenges that will test practically the information shared in On-line classes. You must write a program and execute it in a notebook format. At the end of the week you must submit this notebook. The submission deadline date will be Saturday before the on-line class.

The notebook must contain the responses to some questions. To respond them, use a cell in the notebook with the answers.

During the week (Monday to Friday) you can post in the forum sharing your questions, your thoughts and your accomplishments to the group. I will try to answer all your posts trying to guide you to solve the exercise and any other question that you may raise.

Learning Objectives

The objective of this assignment is to create and run a first program using neural networks. This program would be made using KERAS.

This assignment will allow you to fine tune your environments (either local of COLAB) with all the installed packages to make KERAS-Tensorflow to work.

Tensorflow can be a bit tricky as it may have some conflicts with other packages, so that is very important that you prepare a place to work out all the examples in the course.

Files for this Practice

There are some notebook examples that can be obtained from the course github REPO.

```
    github.com/castorgit/DL-Course
```

The sample notebook for this practice is:

```
    020_MNIST_MLP_Keras.ipynb
```

Some notes about Deep Learning environments

This Practice is to refresh how to create ANN to solve Deep Learning problems using KERAS, and to fine tune your environment to accomodate the Keras-Tensorflow packages and make sure they work together with yoour gymnasium ones.

There are two major tools for neural networks. Tensorflow-Keras and Pytorch. We will focus on KERAS in this course, however both tools are thriving and you can find many application in one or the other.

PyTorch is an open-source deep learning framework developed by Facebook's AI Research lab (FAIR). It has gained immense popularity in the machine learning community due to its dynamic computational graph, ease of use, and flexibility. PyTorch provides a comprehensive platform for building and training deep learning models, making it an excellent choice for both research and production. Keras has been developed by Google.

There are several resources for Deep Learning, I can recommend [Tra19], for Pytorch you can find a good introduction in [SAV20]. If you are interested in the best introduction to Neural Networks / Deep Learning look at this book, from François Chollet, A senior Google engineer and the author of Keras [Cho17], be sure you are reading the 2nd edition which is reviewed and more recent.

Some background on Keras

Keras is a high-level neural network library that was first developed by François Chollet in 2015. It was initially designed to serve as a user-friendly interface for building deep learning models, sitting on top of lower-level machine learning frameworks like TensorFlow, Theano, or Microsoft Cognitive Toolkit (CNTK). Over time, it became tightly integrated into TensorFlow and is now its official high-level API.

Keras is known for its simplicity and modularity, making it ideal for beginners and experts alike. It supports building neural networks with layers, offering flexibility to design both simple and complex architectures. Key features include support for convolutional neural networks (CNNs), recurrent neural networks (RNNs), and hybrid models.

Keras provides built-in tools for data preprocessing, visualization, and evaluation. It also supports distributed training and is compatible with CPUs, GPUs, and TPUs. With extensive community support and a wealth of pre-trained models, Keras is a go-to library for deep learning tasks in academia and industry.

MNIST Dataset Background

The MNIST dataset is a collection of handwritten digits widely used for training and testing machine learning models, particularly in the field of image processing and classification. It consists of 70,000 grayscale images, each representing a single digit from 0 to 9.

The MNIST dataset serves as a benchmark for evaluating machine learning algorithms, especially for image classification tasks. It has been extensively used to train and test various models, ranging from simple classifiers to complex convolutional neural networks.

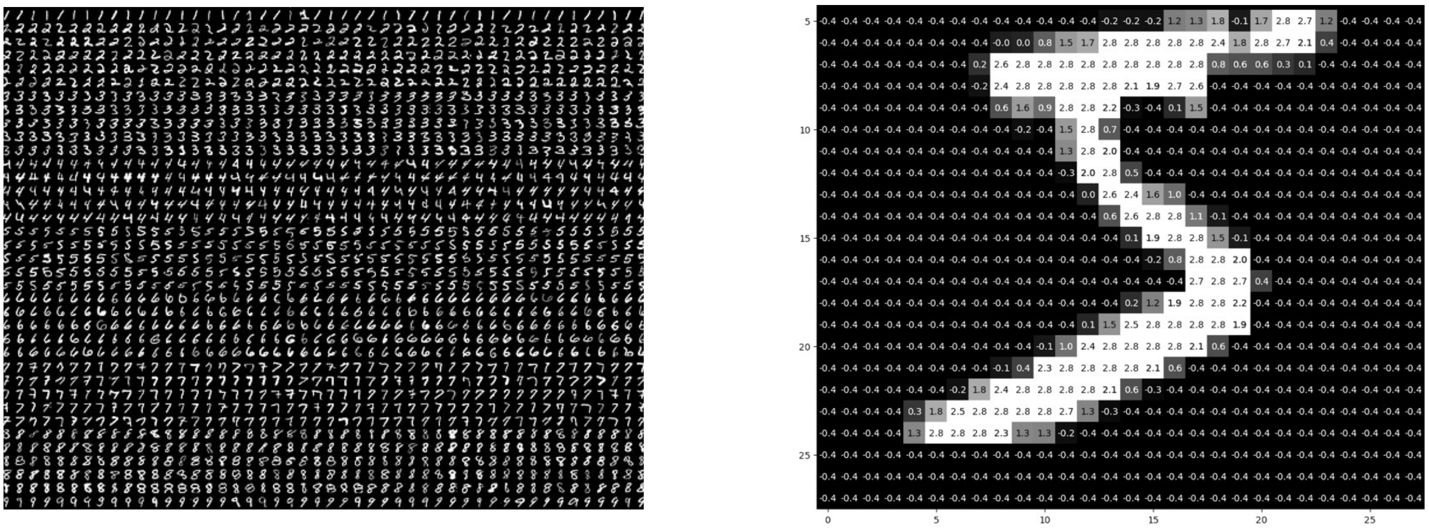
The MNIST dataset is a fundamental resource in the machine learning community due to its simplicity, accessibility, and effectiveness in benchmarking algorithms for handwritten digit recognition.

- **Number of Samples:** The dataset contains 60,000 training images and 10,000 test images.

- **Image Dimensions:** Each image is 28 pixels in height and 28 pixels in width, resulting in a total of 784 pixels per image.
- **Classes:** The dataset is balanced, with an equal distribution of digits from 0 to 9, making it suitable for multiclass classification tasks.

Example Images

Figure 1 shows examples of handwritten digits from the MNIST dataset.



(a) Some MNIST numbers

(b) An MNIST Single digit representation

Figure 1: The MNIST dataset

1 Developing a Deep Learning Classifier

To develop a deep learning image classification algorithm we need a set of labeled images, in this case MNIST which contains thousands of digital digit images. (A good book to use as an introductory book to deep learning is this one [Tra19], but to follow the examples in pytorch I recomend this one [SAV20], which has the examples in torch)

The first step consists of Preprocessing the data by normalizing pixel values and splitting it into training, validation, and test sets.

Then we define the model architecture suitable for our data structure, in our case the input layer will be 784 (or 28x28) and the output layer will be 10 (as we are recognizing 10 digits). Each architecture will ahve a specific set of layers like convolutional, pooling, and fully connected layers, along with activation functions such as ReLU.

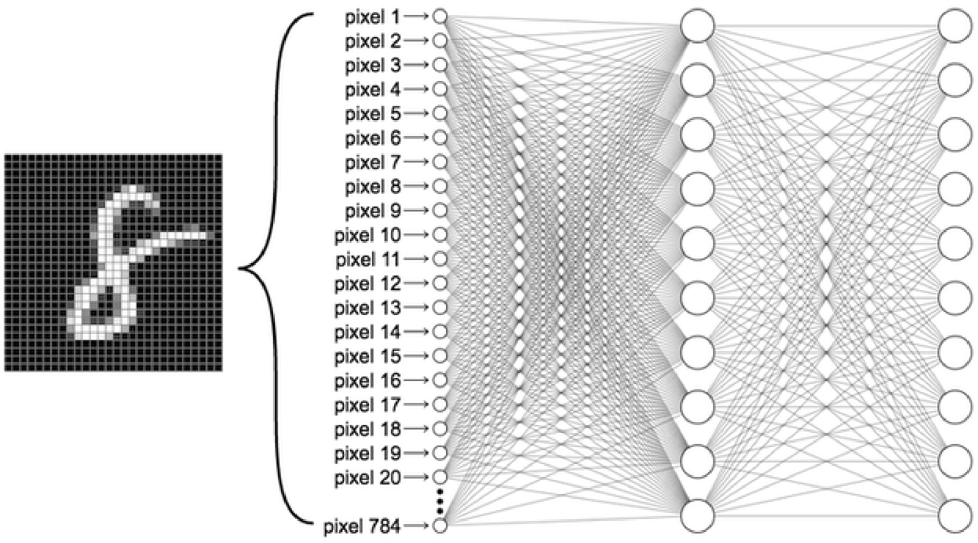


Figure 2: An MLP network for MNIST with hidden layer of 10 neurons

After defining the network we must train it with the training dataset. For this reason we must initialize the model parameters and choose a loss function, commonly cross-entropy for classification tasks. Select an optimizer like Adam or SGD to update the model weights during training.

Train the model by passing the training data through the network, computing the loss, and updating the weights iteratively. Use the validation set to tune hyperparameters and avoid overfitting, implementing techniques like dropout or data augmentation if necessary.

After training, evaluate the model performance on the test set to gauge its accuracy and generalization capability. Visualize results using confusion matrices and ROC curves. If performance is unsatisfactory, consider refining the model architecture, adding more data, or employing transfer learning.

A Multilayer perceptron for the MNIST will have the structure of figure 2

2 Activities for the Assignment

2.1 Generate a Multilayer Perceptron (MLP) for MNIST

You have in the example a MLP classifier. Run the example and obtain results. You may try increasing the hidden_layer size.

Implement an MLP using KERAS for the MNIST dataset use the notebook and read carefully the comments:

- Design the architecture (change hidden_layer size)
- Train the model modify the number of epochs for training
- Evaluate the model on a validation set
- Report accuracy and discuss any challenges encountered

Try to obtain better results by increasing the size of the network and neurons.

To increase the number of neurons in the hidden_layer you must modify the variable **hidden_size**

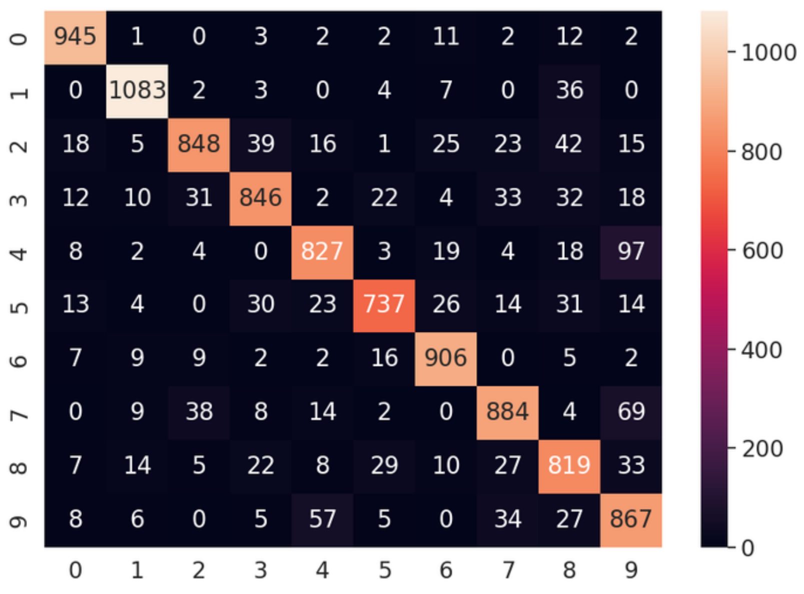
The input must have 784 neurons which is the size of the input image (28x28), then each layer must be Dense with a size established by the parameter **hidden_size**. The Output layer must have 10 neurons (the number of classes to classify the digits).

```
hidden_size = 10

inputs = Input(shape= (784,))
x = Dense(hidden_size)(inputs)
x = ReLU()(x)
x = Dense(10)(x)
output = Softmax()(x)

model = Model(inputs=inputs, outputs=output)
```

This is the logic, you just must follow it and increase the number of layers, until you get to the maximum accuracy. As you are performing a classification you could insert a confusion matrix of the 10 classes to see if the classifier is working well with all of them (like this one 3)



	0	1	2	3	4	5	6	7	8	9
0	945	1	0	3	2	2	11	2	12	2
1	0	1083	2	3	0	4	7	0	36	0
2	18	5	848	39	16	1	25	23	42	15
3	12	10	31	846	2	22	4	33	32	18
4	8	2	4	0	827	3	19	4	18	97
5	13	4	0	30	23	737	26	14	31	14
6	7	9	9	2	2	16	906	0	5	2
7	0	9	38	8	14	2	0	884	4	69
8	7	14	5	22	8	29	10	27	819	33
9	8	6	0	5	57	5	0	34	27	867

Figure 3: Example of a confussion matrix for the MNIST exercise

HINT: The maximum accuracy you can obtain with this problem is around 98% don't over-engineer the network to get a good accuracy, one or two hidden layers will do. In this first example, as we have not optimized anything the result is worse (below 90% possibly). Remember that the objective of this assignment is to run the example in your environment (or Google COLAB). In a future assignment we will work on optimizing the network and getting close to the mentioned 98%.

One small point regarding input tensor size: In this example we are using a 1 dimensional matrix (vector) of 784 positions. This comes from flattening the 2 dimensional images of 28x28 (result = 784).

3 Questions to Answer

Answer the questions inside the notebook in a cell. You can do that in Jupyter Notebooks by creating a Cell of type 'Markdown'. There are some formatting codes for Markdown that you can learn as well, a valuable skill if you write jupyter notebooks to share.

```
https://www.markdownguide.org/cheat-sheet/
```

1. How long have you trained the network?
2. What accuracy do you obtain with this program?
3. What do you see analyzing the confusion matrix?
4. Do you think the program is overfitting by looking at the Loss/Accuracy plots?

4 Deliverables and submission

Submit the following files with content

- A Notebook with the MLP program

References

[Cho17] François Chollet. *Deep Learning with Python*. 2nd ed. Manning, Nov. 2017. ISBN: 9781617294433.

[Tra19] Andrew Trask. *Grokking Deep Learning*. 1st. USA: Manning Publications Co., 2019. ISBN: 1617293709.

[SAV20] Eli Stevens, Luca Antiga, and Thomas Viehmann. *Deep Learning with Pytorch*. Manning, 2020. ISBN: 9781633438859.