AC – Aprendizagem Computacional / Machine Learning

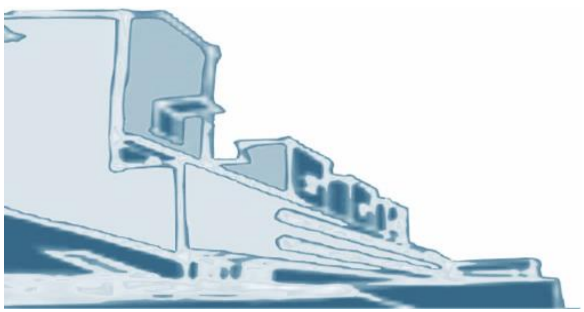
P4c – Deep CNN Models

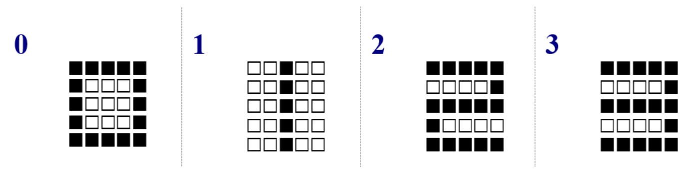
Jorge Henriques

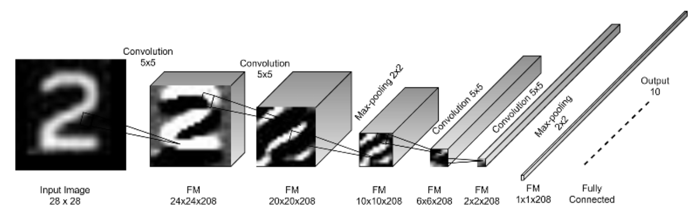
jh@dei.uc.pt

Departamento de Engenharia Informática

Faculdade de Ciências e Tecnologia







1 2 9 0

UNIVERSIDADE D COIMBRA

engenharia informática

Contents

- Deep learning Concepts

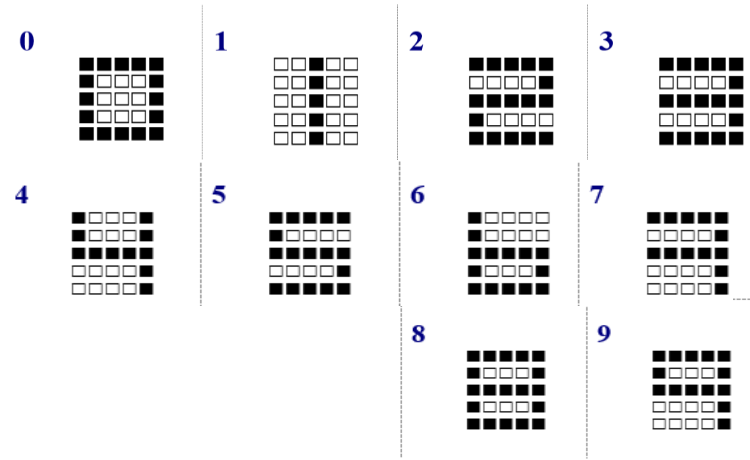
  - 1| Implement Digit recognition from scratch
    - One layer network
    - Backpropagation

  - 2| Use python/keras learning functionalities
    - Deep neural network
      - MLNN / autoencoders
    - Convolutional neural network

  - 3| Evaluation the performance of the DNN/CNN classifier

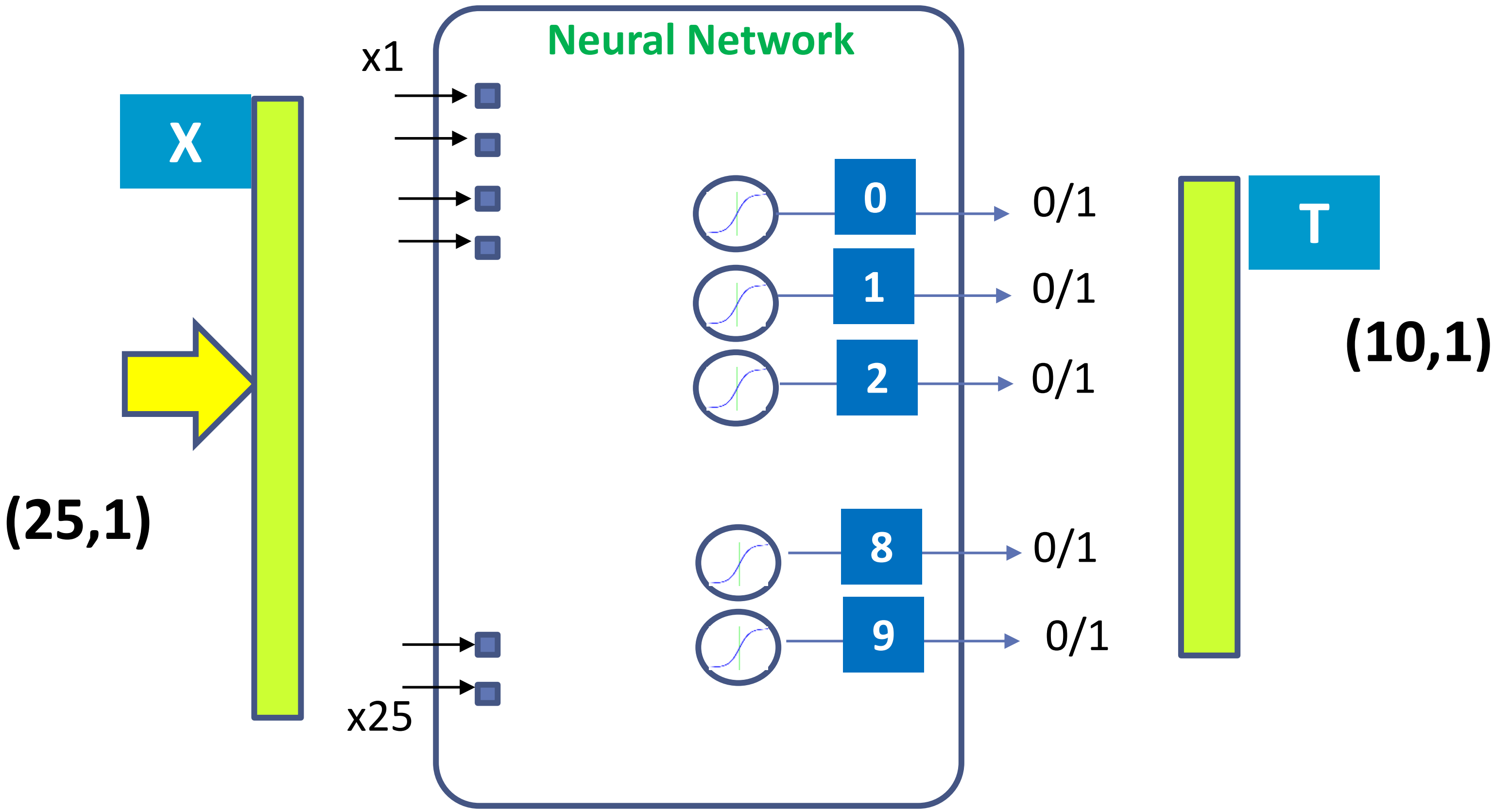
Contents

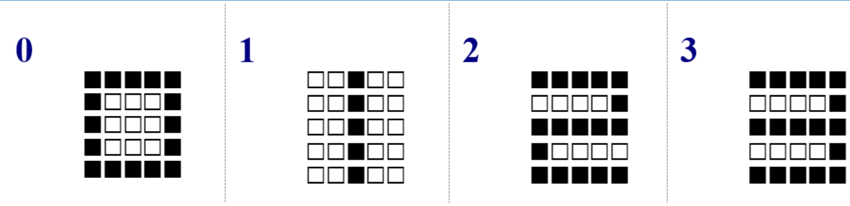
- **Datasets**
  - **1| JH dataset**
    - **digitsX.csv**
    - **digitsT:csv**
    - Digits (5,5)
    - N=100 (10 examples of each digit)



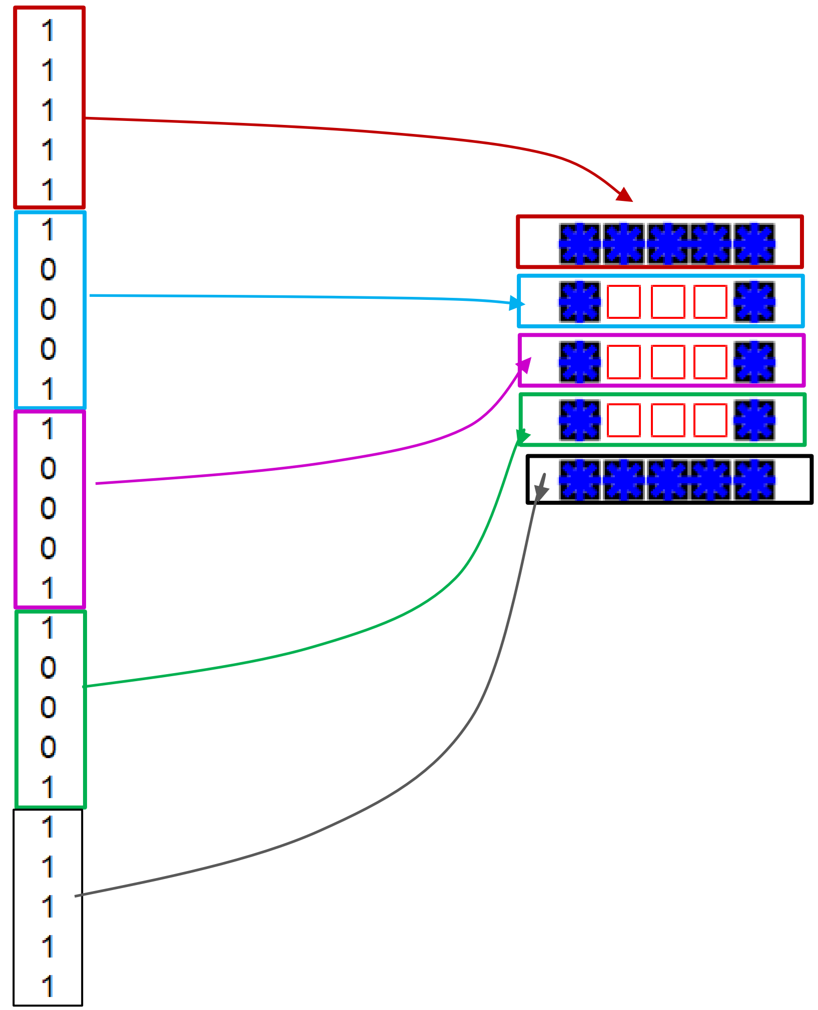
  - **2| MNIST dataset**
    - from keras.datasets import mnist
    - X = mnist.load_data()
    - Digits (28,28)
    - N=70000 (60000 train+10000 test)

- **Neural network : structure ?**
  - Inputs (**vector**) / outputs (**one node per class**)





- **Inputs**
  - **Image (matrix) ➔ vector**
  - **(5x5) ➔ (25,1)**



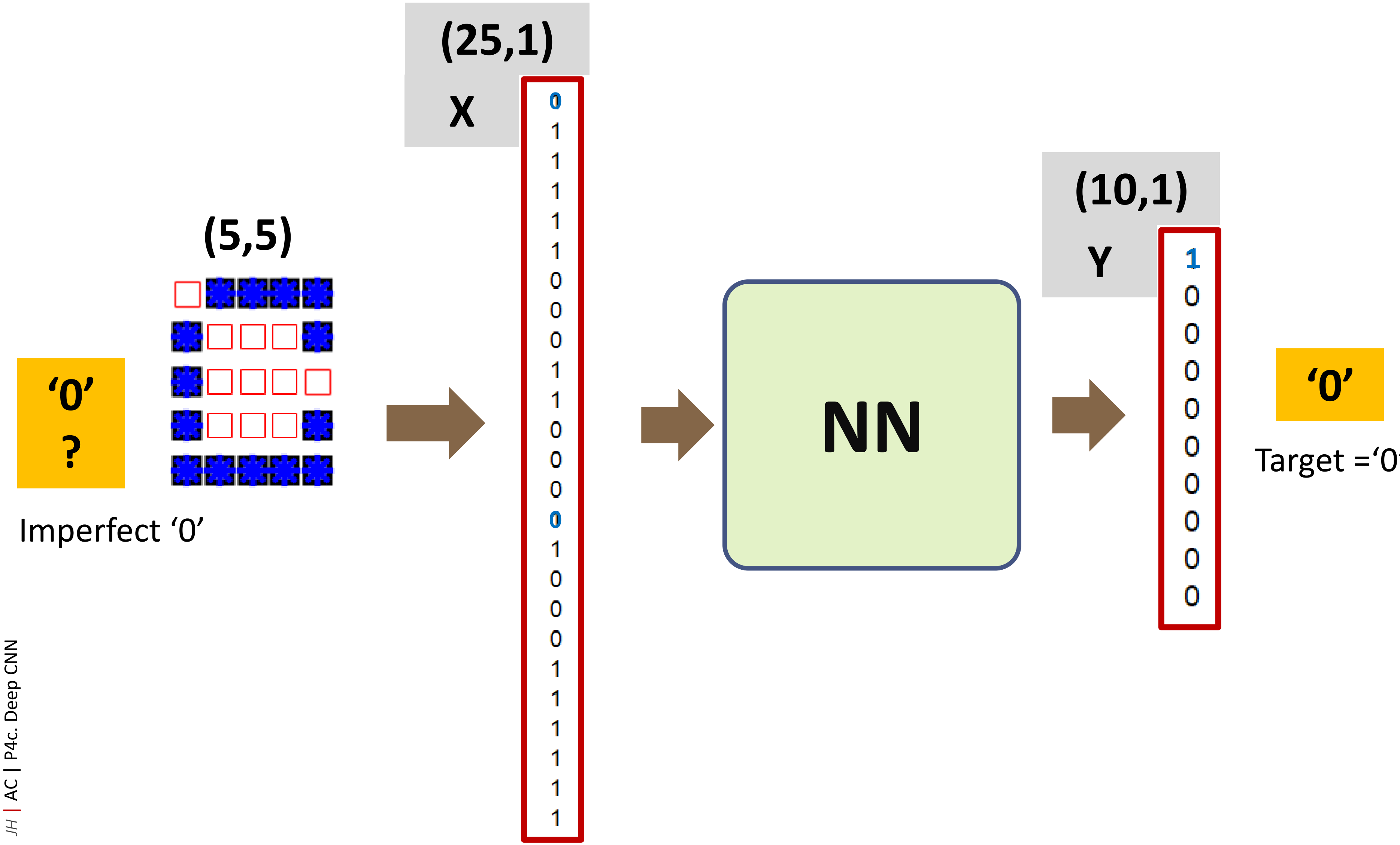
  - X=[:,1]
  - (25,1)

Target

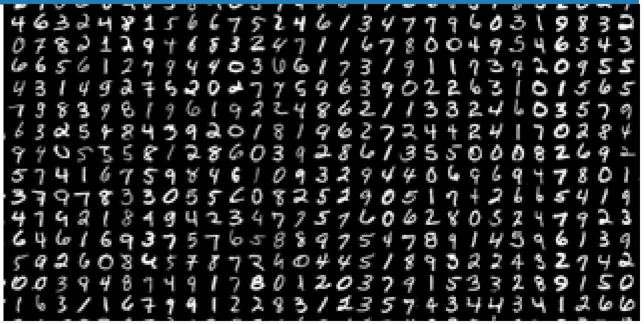
- **One node per class: (10,1)**

Digit	0	1	2	3	4	5	6	7	8	9
Output	1	0	0	0	0	0	0	0	0	0
	0	1	0	0	0	0	0	0	0	0
	0	0	1	0	0	0	0	0	0	0
	0	0	0	1	0	0	0	0	0	0
	0	0	0	0	1	0	0	0	0	0
	0	0	0	0	0	1	0	0	0	0
	0	0	0	0	0	0	1	0	0	0
	0	0	0	0	0	0	0	1	0	0
	0	0	0	0	0	0	0	0	1	0
	0	0	0	0	0	0	0	0	0	1

- **One example**



(25,1)
X
0
1
1
1
1
1
0
0
0
1
1
0
0
0
0
1
0
0
0
1
1
1
1
1
1
(5,5)
'0'
?
Imperfect '0'
NN
(10,1)
Y
1
0
0
0
0
0
0
0
0
0
'0'
Target ='0'



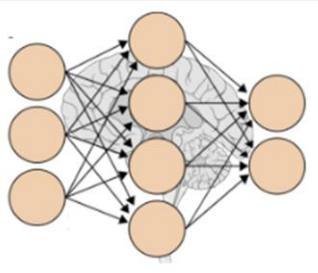
- **MNIST dataset**

  - Total of 70,000 examples of handwritten digits
  - Divided into two subsets:
    - Training set: 60,000 examples
    - Test set: 10,000 examples
  - Each image is grayscale [0 .. 255] image of size 28×28 pixels (784 total features for flat/vector input).
  - Labeled with one of 10 classes (digits 0.. 9).

Contents

- **Task 1: One-layer / MLNN (hidden layer1)**
  - JH dataset

- **Implement a simple NN**
  - Adaline ?
  - One hidden layer
  - MLNN (one hidden layer) ?



Last Class

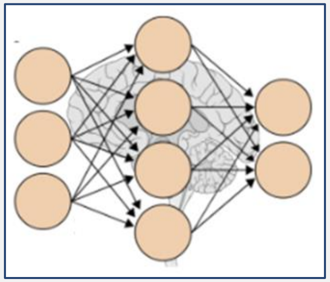
- SE, SP, F1score

  Computed for each digit

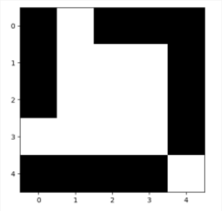
  - For example, with respect to digit '2'
  - T   =[2 2 7 3 4 5 6 7 0 9 9 2 0]
  - T1   =[1 1 0 0 0 0 0 0 0 0 0 1 0]
  - Ynet =[0 1 0 0 1 0 0 0 1 0 0 0 1]
  -        FN TP TN TN FP TN . . .

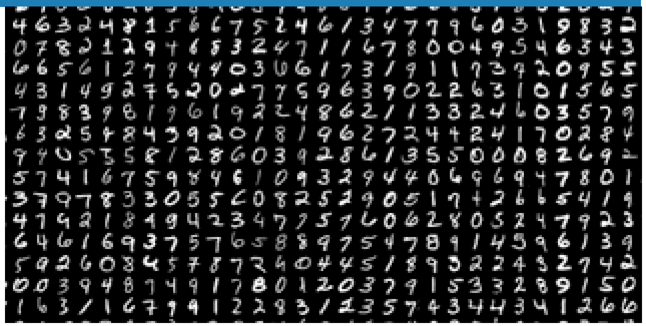
Task 1: One-layer or MLNN (hidden layer)

```
#------------------------- DATA  X(25,100)  T(10,100)
filename ='digitsX.csv'
df       = read_csv(filename)
X        = df.values
filename ='digitsT.csv'
df       = read_csv(filename)
T        = df.values
N        = X.shape[1]
```



```
#------------------------- Image visualization
dig = X[:,36]      # dimension (25,1)
plt.figure(1)
plt.imshow( dig.reshape( (5,5)), cmap=plt.cm.gray_r)
plt.show()
```

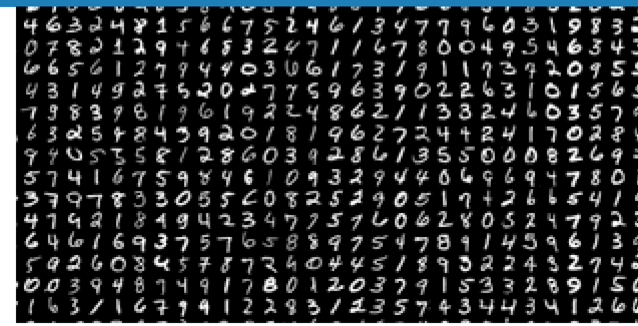




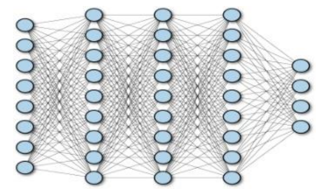
- **Task 2: Deep NN**
  - **MNIST dataset**

  - Total of 70,000 examples of handwritten digits
  - Divided into two subsets:
    - Training set: 60,000 examples
    - Test set: 10,000 examples

  - Each image is grayscale image of size 28×28 pixels (784 total features for flat input).
  - Labeled with one of 10 classes (digits 0–9).

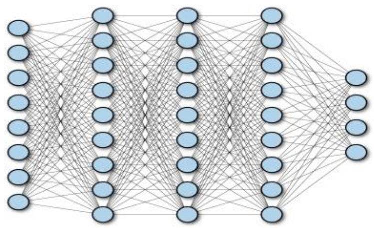


- **Task 2: Implement a Deep NN**



- 2.1 | MLNN – “high” number of layers
- 2.2 | Encoders - decoders
- 2.3 | CNN - Convolutional neural networks

▪Task 2.1| Deep NN - MLNN

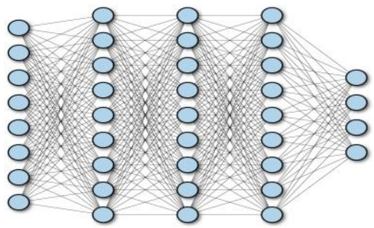


```
from keras.datasets import mnist
#------------------------- load MNIST  [0-225] gray scale
#----    X = mnist.load_data()

(Xtrain, Ttrain), (Xtest, Ttest) = mnist.load_data()

>  Xtrain.shape
> (60000, 28, 28)
```

```
#------------------------- Reshape
Xtrain = Xtrain.reshape(Xtrain.shape[0], num_pixels).astype('float32')
Xtest  =  Xtest.reshape( Xtest.shape[0], num_pixels).astype('float32’)

> Xtrain.shape
> (60000, 784)
```

```
#------------------------- Normalization [0.155] -> [0..1]  (gray->bw)
Xtrain = Xtrain / 255
```

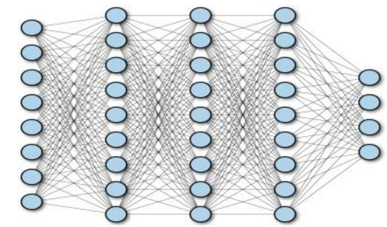
Task 2.1| Deep NN - MLNN



```
#------------------------  Target [6000,1]
(Xtrain, Ttrain), (Xtest, Ttest) = mnist.load_data()

>  Ttrain.shape
>  (60000,)        vector
```

```
from tensorflow.keras.utils import to_categorical
#------------------------ Target [6000.10]  - output - ten neurons
Ttrain = to_categorical(Ttrain)

> Ttrain.shape
> (60000,10)       matrix
```

Task 2.1| Deep NN - MLNN



```
from keras.models import Sequential
from keras.layers import Dense
#------------------------- Define and Train  [6000.10]
Numclasses=10

model = Sequential()
model.add( Dense(19, activation='relu’) )
model.add( Dense(5, activation='relu’) )
model.add( Dense(4, activation='relu’) )
model.add( Dense(3, activation='relu’) )
model.add( Dense(num_classes,  activation='softmax’) )

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

model.fit( Xtrain, Ttrain, epochs=12, batch_size=500,verbose=2)

Ytrain  = model.predict(Xtrain).round()
Ytest   = model.predict(Xtest).round()
```

- Softmax layer as the output layer

- For *classification* problems it is advantageous to use the activation function *softmax*, at the output layer

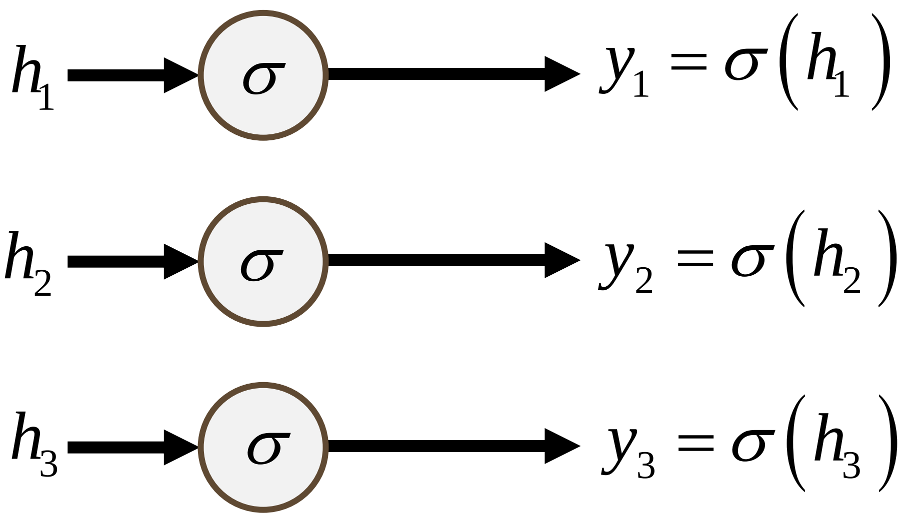
- *Softmax* normalizes the output and **mimic a probability** vector for each output.

- Softmax function accounts not only for the weighted sum of the inputs, but also for the inputs to the other output nodes

$$f(x_i) = \frac{e^{x_i}}{\sum_{i=1}^{n} e^{x_i}}$$

- Softmax layer as the output layer

Output Layer



In general, the output of network can be any value.

May not be easy to interpret

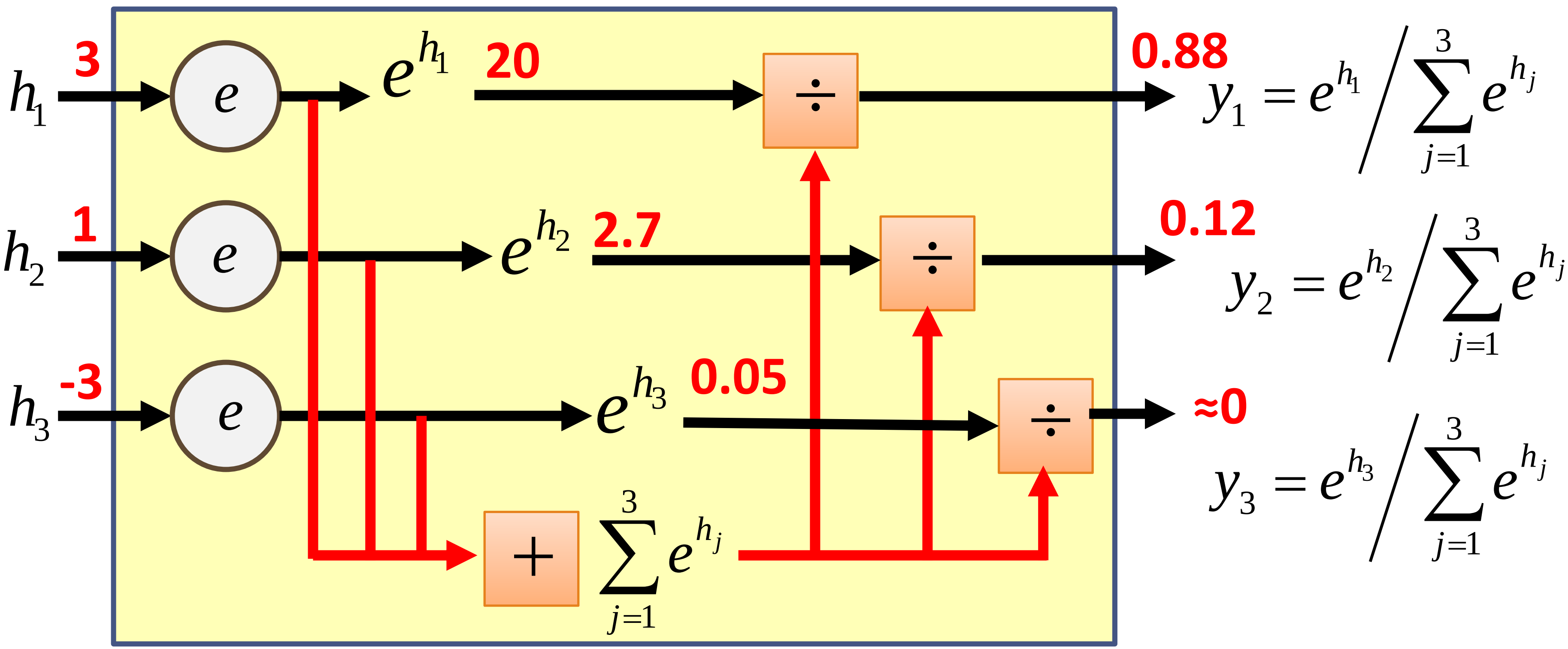
$$s(y_i) = \frac{e^{y_i}}{\sum_N e^{y_i}}$$

y_i

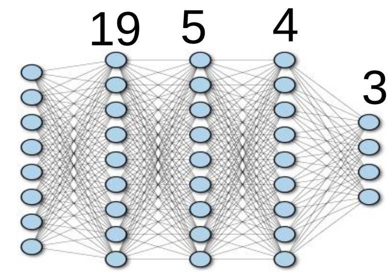
$S(y_i)$

- Softmax layer as the output layer

Softmax Layer



Task 2.1: MLNN



- N_EPOCHS=2

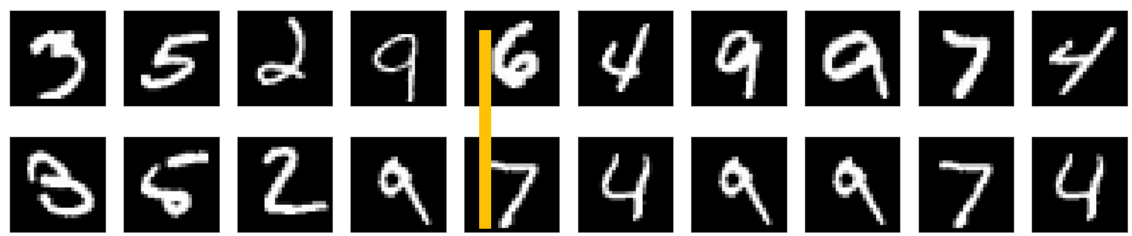
Test

Target

- N_EPOCHS=12

Test

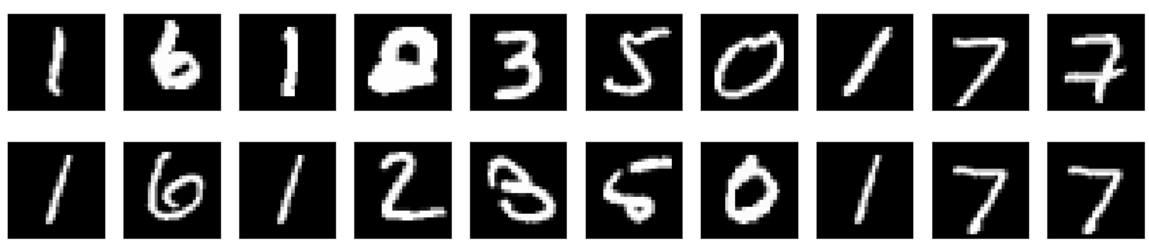
Target



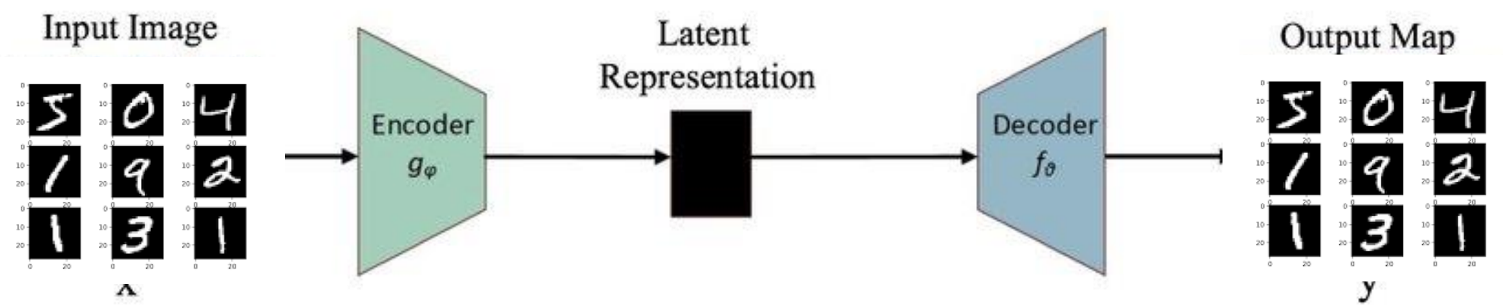
- N_EPOCHS=32

Test

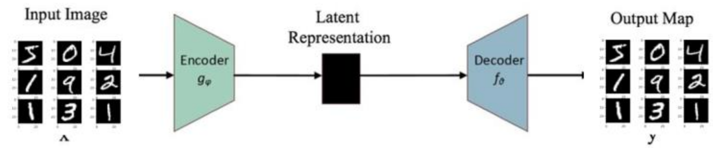
Target



- **Task 2.2: Autoencoders**

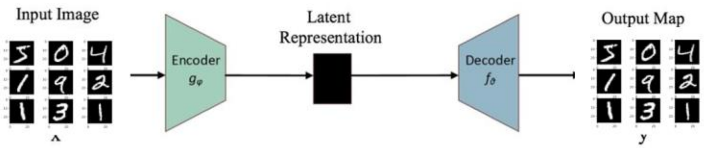


Task 2.2: Autoencoders




```
(Xtrain, Ttrain), (Xtest, Ttest) = mnist.load_data()

#---------------------------------- normalization, reshape
X_train = X_train.astype('float32') / 255.
X_test  = X_test.astype('float32') / 255.
X_train = np.reshape(X_train, (len(X_train), 28, 28, 1))
X_test  = np.reshape(X_test,  (len( X_test), 28, 28, 1))
```

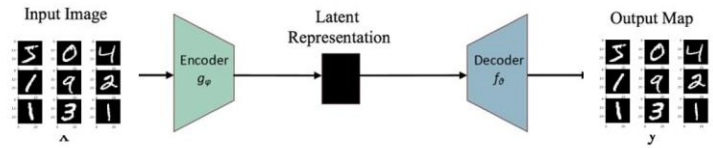
Task 2.2: Autoencoders



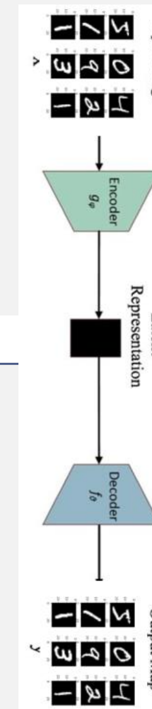

```
#------------------------- ENCODER
input_image = Input(shape =(28, 28, 1))

x = Conv2D(16, (3, 3), activation ='relu', padding ='same')(input_image)
x = MaxPooling2D((2, 2), padding ='same')(x)

x = Conv2D(8, (3, 3), activation ='relu', padding ='same')(x)
x = MaxPooling2D((2, 2), padding ='same')(x)

x = Conv2D(8, (3, 3), activation ='relu', padding ='same')(x)
encoded_layer = MaxPooling2D((2, 2), padding ='same')(x)
```

- Conv2D(16, (3, 3) -----
  - Creates a 2D convolutional layer with **16 filters** (or kernels).
  - Each filter has a kernel size of 3x3 (**mask**).
  - activation='**relu**':
  - padding='**same**': Ensures the output has the same spatial dimensions (height and width) as the input by adding zero-padding around the input

Task 2.2: Autoencoders

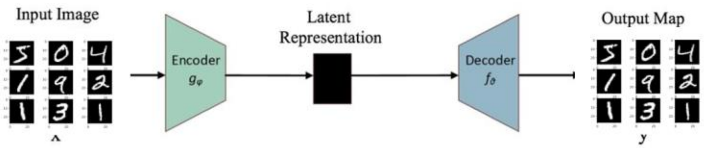



```
#------------------------- ENCODER
input_image = Input(shape =(28, 28, 1))

x = Conv2D(16, (3, 3), activation ='relu', padding ='same')(input_image)
x = MaxPooling2D((2, 2), padding ='same')(x)

x = Conv2D(8, (3, 3), activation ='relu', padding ='same')(x)
x = MaxPooling2D((2, 2), padding ='same')(x)

x = Conv2D(8, (3, 3), activation ='relu', padding ='same')(x)
encoded_layer = MaxPooling2D((2, 2), padding ='same')(x)
```




```
#------------------------- DECODER
x = Conv2D(8, (3, 3), activation ='relu', padding ='same')(encoded_layer)
x = UpSampling2D((2, 2))(x)
x = Conv2D(8, (3, 3), activation ='relu', padding ='same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(16, (3, 3), activation ='relu')(x)
x = UpSampling2D((2, 2))(x)

decoded_layer = Conv2D(1, (3, 3), activation ='sigmoid', padding ='same')(x) )
```

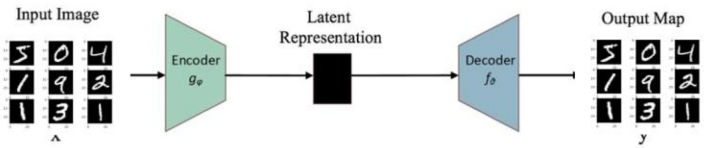
Task 2.2: Autoencoders



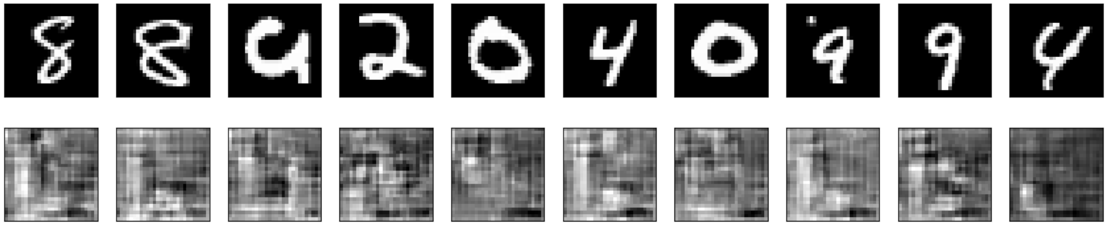

```
#------------------------- BUILT
autoencoder = Model(input_image, decoded_layer)

autoencoder.compile(optimizer ='adam', loss ='binary_crossentropy')


#------------------------- TRAIN
autoencoder.fit(X_train, X_train, epochs = 2, batch_size = 256,
            shuffle = True,
            validation_data =(X_test, X_test))


#------------------------- VISUALIZATION
visualize(autoencoder, X_test)
```

shuffle = True:
The training data is shuffled at the beginning of each epoch to improve generalization and avoid overfitting

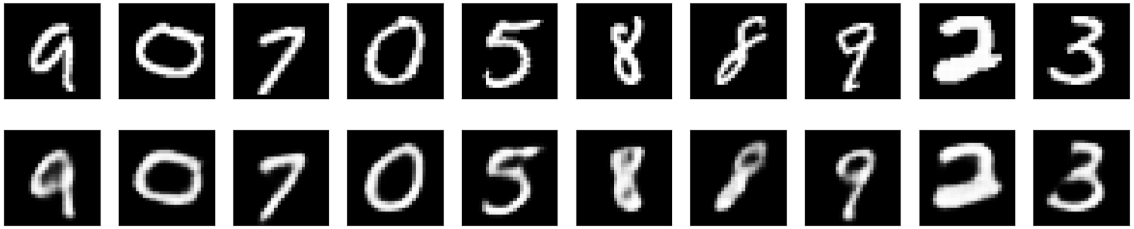
Task 2.2: Autoencoders



- N_EPOCHS=2

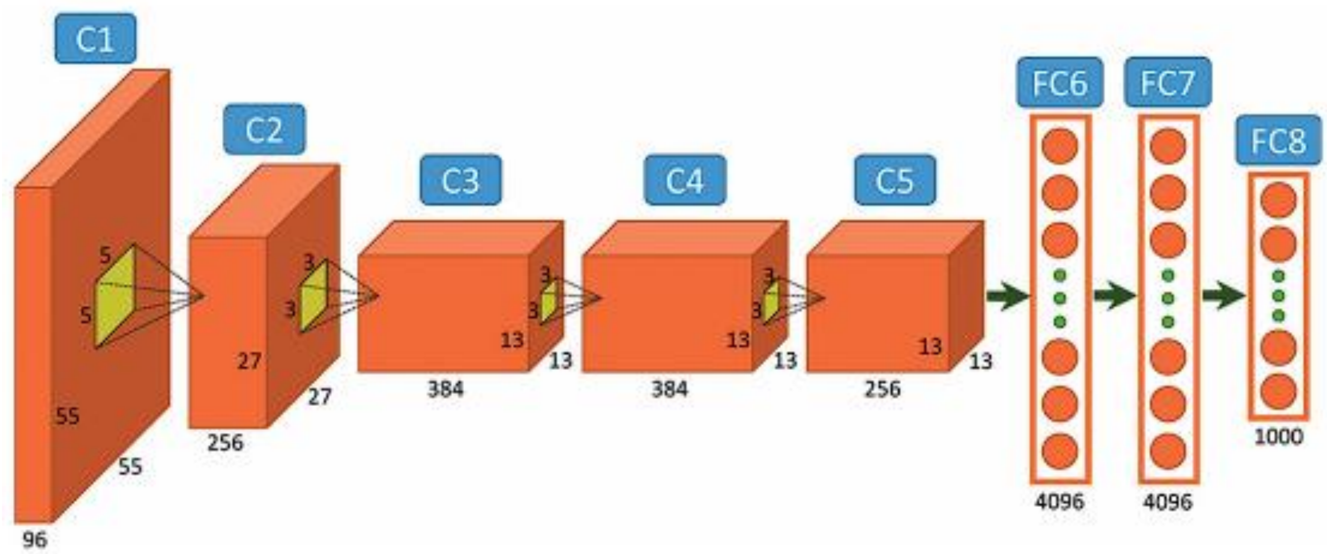


- N_EPOCHS=20

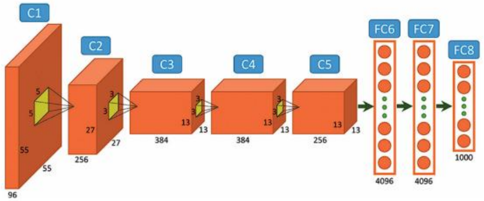


Task 2.3: **CNN**

- Convolutional neural network



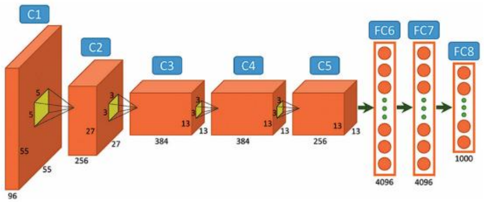
Task 2.3: CNN




```
#--------------------------- RESHAPE
 if k.image_data_format() == 'channels_first':   #theano
        Xtrain = Xtrain.reshape(Xtrain.shape[0], 1, 28,28)
        Xtest = Xtest.reshape(Xtest.shape[0], 1, 28,28)
        inpx = (1, 28,28)
    else:
        Xtrain = Xtrain.reshape(Xtrain.shape[0], 28, 28, 1)
        Xtest = Xtest.reshape(Xtest.shape[0], 28, 28, 1)
        inpx = (28, 28, 1)
```

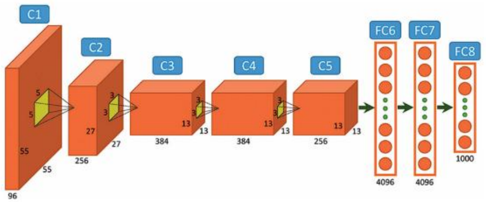
- Define dimension of the image
  - **k.image_data_format() == 'channels_first'**
  - **False in mnist**
  - Check how the image data is formatted in the current environment.

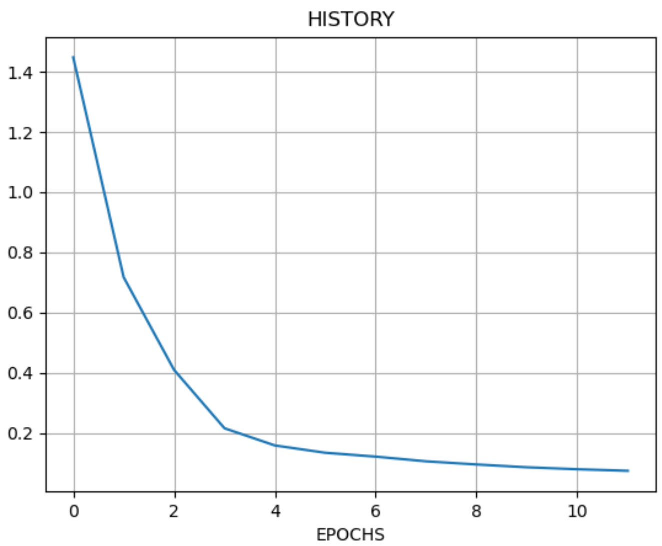
Task 2.3: CNN



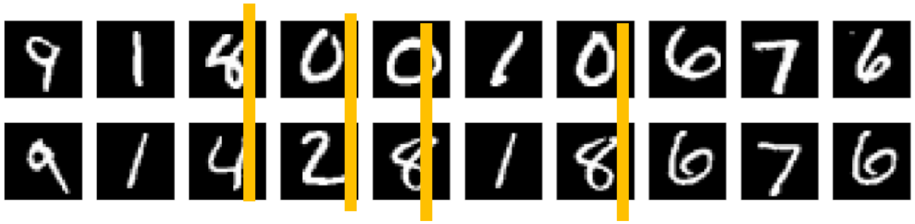

```
model = Sequential()

#--------------------------------------------------------  CNN 1
model.add( Conv2D(64, (3, 3), activation='relu') )
model.add( MaxPooling2D(pool_size=(2, 2)))

#--------------------------------------------------------  CNN 2
model.add( Conv2D(32, (3, 3), activation='relu') )
model.add( MaxPooling2D(pool_size=(2, 2)))

#--------------------------------------------------------  VECTOR output
model.add(Flatten())

#--------------------------------------------------------  MLNN
model.add(Dense(14, activation=‘relu’))
model.add(Dense( 6, activation=‘relu'))
model.add(Dense(num_classes, activation='softmax'))

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

history = model.fit(Xtrain, Ttrain, epochs=12, batch_size=500)

print(history.history['loss’])
```

Task 2.3: CNN





EPOCHS=2



EPOCHS=22

```
train  = model.predict(Xtrain).round()
Ytest  = model.predict(Xtest).round()
scores = model.evaluate(Xtest, Ttest, verbose=0)

print("Baseline Error: %.2f%%" % (100-scores[1]*100))
```

- **Conv2D(64, (3, 3), activation='relu')**
  - Performs 2D convolution,
  - Specifies the number of filters (kernels) = 64
  - Mask : Indicates the filter size or kernel size, which is 3x3.
  - Activation: 'relu'

- MaxPooling2D(pool_size=(2, 2))
  - Downsampling the spatial dimensions of feature maps.
  - In this case, a 2x2 reducing the spatial dimensions by a factor of 2

- Flatten()
  - Takes a multi-dimensional tensor (e.g., 2D, 3D, or higher) and flattens it into a single 1D vector.
  - Required before feeding the output of a convolutional or pooling layer into a fully connected (dense) layer.

Contents

4| conclusions

- Deep learning / CNN
- Concepts
  - 1| Implement Digit recognition from scratch
  - One layer network ? One hidden layer ‘
- 2| Use python/keras learning functionalities
  - MLL
  - Autoencoders
  - Convolutional neural network
- 3| Evaluation the performance of the DNN/CNN classifier