Nabil_ELAsri_826040_assignment3

November 26, 2021

1 Designing a CNN architecture and its formation.

The third assignment consists in the classification of numbers from 0 to 9, developing a convolutional neural network, starting from the MNIST dataset. Moreover there is a constraint in the number of parameters used: maximum 6000.

[ ]:
```
# amount some libraries for the analysis of the data and development of the␣
 ↪model
import numpy as np
import matplotlib.pyplot as plt

# DL ops
from tensorflow import keras
from tensorflow.keras import layers
```

2 Inspecting the data

During the exploration of the dataset we immediately notice that the size of the train set is 60000 records, while 10000 for the test set, each containing the image of a number from 0 to 9 in a matrix 28x28, which indicates the gray scale of each pixel of the image.

[ ]:
```
# load data
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

[ ]:
```
x_train.shape
```

[ ]:
```
(60000, 28, 28)
```

[ ]:
```
y_train.shape
```

[ ]:
```
(60000,)
```

[ ]:
```
x_test.shape
```

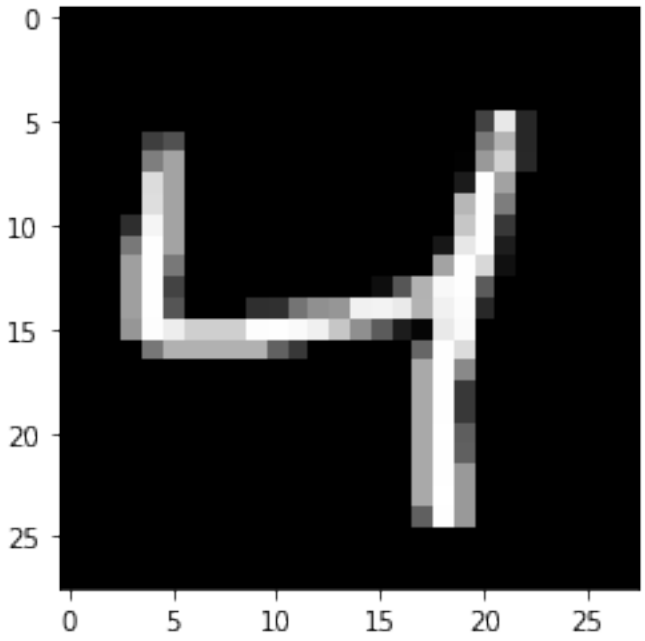
[ ]: (10000, 28, 28)

[ ]:
```
y_test.shape
```

[ ]: (10000,)

[ ]:
```
plt.imshow(x_train[2].reshape([28,28]), cmap="gray")
```

[ ]: <matplotlib.image.AxesImage at 0x7f15ec9e91d0>



3 Preparing the data

Normalization was done by simply dividing each number by 255, since the range of numbers in the above matrices is from 0 to 255. After that a reshape is done, since the convolutional layers need to know how many channels there are for each image, in this case one since they are black and white photos. Then we proceeded with the categorization of the variable to be categorized, equal to 10.

[ ]:
```
# scale images in [0,1]
x_train = x_train.astype('float')/255
x_test = x_test.astype('float')/255
# images 28x28
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
print('x_train_shape:', x_train.shape)

num_classes = 10
input_shape = (28,28,1)
```

```
#convert class vectors to binary class matrices
print('y_train_shape:', y_train.shape)
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
print('y_train_shape:', y_train.shape)
```

```
x_train_shape: (60000, 28, 28, 1)
y_train_shape: (60000,)
y_train_shape: (60000, 10)
```

4 Building the network

NEURAL NETWORK: - 75 epochs - 2 convolutional layers, with respectively 10 and 4 filters, size 3x3, no padding - 2 pooling layers (MaxPooling2D) of size 2x2, "same" padding - 1 dense layer of 35 neurons - 2 dropout layers - activation function layers: ReLU - activation function output layer: Softmax - optimizer: RMSprop - batch size: 512 - loss function: categorical_crossentropy

The number of epochs was chosen based on when the accuracy of the train set and the test set reached convergence, avoiding overfitting and/or underfitting Respecting the limit imposed on the parameters, choices were made on the number of neurons and the size of the filters in the convolutional layers. The activation function ReLU is a fairly standard choice and was the most performing while for the activation function of the output layer the Softmax was a choice forced by the fact that it was a multiclass classification. The optimizer used for this model was RMSprop, an algorithm that is widely used in the literature when developing convolutional neural networks and that was in fact the most performant. To avoid overfitting, two dropout layers were introduced.

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

     layers.Conv2D(10, kernel_size=(3,3), activation='relu', padding='valid'),
     layers.MaxPooling2D(pool_size=(2,2), padding='same'),
     layers.Dropout(0.1),

     layers.Conv2D(4, kernel_size=(3,3), activation='relu', padding='valid'),
     layers.MaxPooling2D(pool_size=(2,2), padding='same'),

     layers.Flatten(),
     layers.Dense(35, activation='relu'),
     layers.Dropout(0.1),

     layers.Dense(num_classes, activation='softmax')
     ]
)

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 10)        100

 max_pooling2d (MaxPooling2D  (None, 13, 13, 10)       0
 )

 dropout (Dropout)           (None, 13, 13, 10)        0

 conv2d_1 (Conv2D)           (None, 11, 11, 4)         364

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 4)          0
 2D)

 flatten (Flatten)           (None, 144)               0

 dense (Dense)               (None, 35)                5075

 dropout_1 (Dropout)         (None, 35)                0

 dense_1 (Dense)             (None, 10)                360

=================================================================
Total params: 5,899
Trainable params: 5,899
Non-trainable params: 0
_________________________________________________________________
```

1. INPUT
2. CONVOLUTIONAL, SHAPE 26x26x10, PARAMETERS [(1 * 3 * 3 * 10)+10] = 100
3. POOLING, SHAPE 13x13x10, PARAMETERS 0
4. DROPOUT, SHAPE 13x13x10, PARAMETRS 0
5. CONVOLUTIONAL, SHAPE 11x11x4, PARAMETERS [(10 * 3 * 3 * 4)+4] = 364
6. POOLING, SHAPE 6x6x4, PARAMETERS 0
7. FLATTERN
8. DENSE, SHAPE 35, PARAMETERS (144 * 35)+35 = 5075
9. DROPOUT, SHAPE 35, PARAMETERS 0
10. OUTPUT, SHAPE 10, PARAMETERS (35 * 10)+10 = 360

TOTAL = 5899

[ ]:
```
batch_size = 512
epochs = 75

model.compile(loss='categorical_crossentropy', optimizer='RMSprop',␣
 ↪metrics=['accuracy'])
```

```
network_history = model.fit(x_train, y_train, batch_size=batch_size,␣
↪epochs=epochs, validation_split=0.2)
```

```
Epoch 1/75
94/94 [==============================] - 15s 19ms/step - loss: 1.2745 -
accuracy: 0.5991 - val_loss: 0.5128 - val_accuracy: 0.8597
Epoch 2/75
94/94 [==============================] - 1s 13ms/step - loss: 0.5156 - accuracy:
0.8400 - val_loss: 0.3089 - val_accuracy: 0.9114
Epoch 3/75
94/94 [==============================] - 1s 12ms/step - loss: 0.3629 - accuracy:
0.8889 - val_loss: 0.2318 - val_accuracy: 0.9320
Epoch 4/75
94/94 [==============================] - 1s 12ms/step - loss: 0.2859 - accuracy:
0.9126 - val_loss: 0.2047 - val_accuracy: 0.9374
Epoch 5/75
94/94 [==============================] - 1s 12ms/step - loss: 0.2391 - accuracy:
0.9265 - val_loss: 0.1687 - val_accuracy: 0.9490
Epoch 6/75
94/94 [==============================] - 1s 12ms/step - loss: 0.2078 - accuracy:
0.9357 - val_loss: 0.1476 - val_accuracy: 0.9554
Epoch 7/75
94/94 [==============================] - 1s 12ms/step - loss: 0.1876 - accuracy:
0.9427 - val_loss: 0.1310 - val_accuracy: 0.9588
Epoch 8/75
94/94 [==============================] - 1s 12ms/step - loss: 0.1702 - accuracy:
0.9473 - val_loss: 0.1169 - val_accuracy: 0.9651
Epoch 9/75
94/94 [==============================] - 1s 12ms/step - loss: 0.1585 - accuracy:
0.9515 - val_loss: 0.1147 - val_accuracy: 0.9662
Epoch 10/75
94/94 [==============================] - 1s 13ms/step - loss: 0.1508 - accuracy:
0.9528 - val_loss: 0.1075 - val_accuracy: 0.9672
Epoch 11/75
94/94 [==============================] - 1s 12ms/step - loss: 0.1409 - accuracy:
0.9563 - val_loss: 0.1057 - val_accuracy: 0.9674
Epoch 12/75
94/94 [==============================] - 1s 12ms/step - loss: 0.1339 - accuracy:
0.9585 - val_loss: 0.0964 - val_accuracy: 0.9716
Epoch 13/75
94/94 [==============================] - 1s 12ms/step - loss: 0.1273 - accuracy:
0.9599 - val_loss: 0.0956 - val_accuracy: 0.9717
Epoch 14/75
94/94 [==============================] - 1s 12ms/step - loss: 0.1227 - accuracy:
0.9620 - val_loss: 0.0921 - val_accuracy: 0.9722
Epoch 15/75
94/94 [==============================] - 1s 13ms/step - loss: 0.1190 - accuracy:
```

```
0.9624 - val_loss: 0.0922 - val_accuracy: 0.9722
Epoch 16/75
94/94 [==============================] - 1s 12ms/step - loss: 0.1135 - accuracy:
0.9638 - val_loss: 0.0853 - val_accuracy: 0.9743
Epoch 17/75
94/94 [==============================] - 1s 13ms/step - loss: 0.1123 - accuracy:
0.9647 - val_loss: 0.0815 - val_accuracy: 0.9756
Epoch 18/75
94/94 [==============================] - 1s 12ms/step - loss: 0.1062 - accuracy:
0.9671 - val_loss: 0.0814 - val_accuracy: 0.9758
Epoch 19/75
94/94 [==============================] - 1s 12ms/step - loss: 0.1062 - accuracy:
0.9668 - val_loss: 0.0775 - val_accuracy: 0.9763
Epoch 20/75
94/94 [==============================] - 1s 13ms/step - loss: 0.1004 - accuracy:
0.9681 - val_loss: 0.0783 - val_accuracy: 0.9766
Epoch 21/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0992 - accuracy:
0.9687 - val_loss: 0.0751 - val_accuracy: 0.9773
Epoch 22/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0984 - accuracy:
0.9684 - val_loss: 0.0754 - val_accuracy: 0.9768
Epoch 23/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0968 - accuracy:
0.9696 - val_loss: 0.0716 - val_accuracy: 0.9783
Epoch 24/75
94/94 [==============================] - 1s 13ms/step - loss: 0.0936 - accuracy:
0.9704 - val_loss: 0.0737 - val_accuracy: 0.9787
Epoch 25/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0918 - accuracy:
0.9708 - val_loss: 0.0750 - val_accuracy: 0.9783
Epoch 26/75
94/94 [==============================] - 1s 13ms/step - loss: 0.0902 - accuracy:
0.9721 - val_loss: 0.0680 - val_accuracy: 0.9798
Epoch 27/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0887 - accuracy:
0.9706 - val_loss: 0.0656 - val_accuracy: 0.9809
Epoch 28/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0874 - accuracy:
0.9719 - val_loss: 0.0678 - val_accuracy: 0.9793
Epoch 29/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0859 - accuracy:
0.9728 - val_loss: 0.0651 - val_accuracy: 0.9817
Epoch 30/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0848 - accuracy:
0.9731 - val_loss: 0.0773 - val_accuracy: 0.9777
Epoch 31/75
94/94 [==============================] - 1s 13ms/step - loss: 0.0843 - accuracy:
```

```
0.9742 - val_loss: 0.0729 - val_accuracy: 0.9792
Epoch 32/75
94/94 [==============================] - 1s 13ms/step - loss: 0.0834 - accuracy:
0.9738 - val_loss: 0.0672 - val_accuracy: 0.9806
Epoch 33/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0794 - accuracy:
0.9751 - val_loss: 0.0646 - val_accuracy: 0.9812
Epoch 34/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0815 - accuracy:
0.9746 - val_loss: 0.0658 - val_accuracy: 0.9808
Epoch 35/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0801 - accuracy:
0.9745 - val_loss: 0.0623 - val_accuracy: 0.9819
Epoch 36/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0756 - accuracy:
0.9764 - val_loss: 0.0661 - val_accuracy: 0.9812
Epoch 37/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0772 - accuracy:
0.9754 - val_loss: 0.0607 - val_accuracy: 0.9828
Epoch 38/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0747 - accuracy:
0.9760 - val_loss: 0.0634 - val_accuracy: 0.9826
Epoch 39/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0744 - accuracy:
0.9758 - val_loss: 0.0599 - val_accuracy: 0.9825
Epoch 40/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0725 - accuracy:
0.9766 - val_loss: 0.0587 - val_accuracy: 0.9829
Epoch 41/75
94/94 [==============================] - 1s 13ms/step - loss: 0.0715 - accuracy:
0.9773 - val_loss: 0.0622 - val_accuracy: 0.9825
Epoch 42/75
94/94 [==============================] - 1s 13ms/step - loss: 0.0716 - accuracy:
0.9773 - val_loss: 0.0590 - val_accuracy: 0.9827
Epoch 43/75
94/94 [==============================] - 1s 13ms/step - loss: 0.0714 - accuracy:
0.9777 - val_loss: 0.0579 - val_accuracy: 0.9829
Epoch 44/75
94/94 [==============================] - 1s 13ms/step - loss: 0.0681 - accuracy:
0.9781 - val_loss: 0.0602 - val_accuracy: 0.9830
Epoch 45/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0702 - accuracy:
0.9776 - val_loss: 0.0564 - val_accuracy: 0.9836
Epoch 46/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0671 - accuracy:
0.9786 - val_loss: 0.0592 - val_accuracy: 0.9832
Epoch 47/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0675 - accuracy:
```

```
0.9783 - val_loss: 0.0557 - val_accuracy: 0.9845
Epoch 48/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0659 - accuracy:
0.9786 - val_loss: 0.0572 - val_accuracy: 0.9843
Epoch 49/75
94/94 [==============================] - 1s 13ms/step - loss: 0.0655 - accuracy:
0.9789 - val_loss: 0.0588 - val_accuracy: 0.9837
Epoch 50/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0647 - accuracy:
0.9791 - val_loss: 0.0578 - val_accuracy: 0.9828
Epoch 51/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0646 - accuracy:
0.9791 - val_loss: 0.0551 - val_accuracy: 0.9843
Epoch 52/75
94/94 [==============================] - 1s 13ms/step - loss: 0.0648 - accuracy:
0.9799 - val_loss: 0.0543 - val_accuracy: 0.9850
Epoch 53/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0632 - accuracy:
0.9790 - val_loss: 0.0545 - val_accuracy: 0.9840
Epoch 54/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0618 - accuracy:
0.9797 - val_loss: 0.0570 - val_accuracy: 0.9835
Epoch 55/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0621 - accuracy:
0.9801 - val_loss: 0.0580 - val_accuracy: 0.9833
Epoch 56/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0614 - accuracy:
0.9801 - val_loss: 0.0567 - val_accuracy: 0.9824
Epoch 57/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0631 - accuracy:
0.9798 - val_loss: 0.0544 - val_accuracy: 0.9847
Epoch 58/75
94/94 [==============================] - 1s 13ms/step - loss: 0.0600 - accuracy:
0.9802 - val_loss: 0.0555 - val_accuracy: 0.9837
Epoch 59/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0607 - accuracy:
0.9802 - val_loss: 0.0516 - val_accuracy: 0.9851
Epoch 60/75
94/94 [==============================] - 1s 13ms/step - loss: 0.0582 - accuracy:
0.9811 - val_loss: 0.0535 - val_accuracy: 0.9852
Epoch 61/75
94/94 [==============================] - 1s 13ms/step - loss: 0.0581 - accuracy:
0.9811 - val_loss: 0.0516 - val_accuracy: 0.9846
Epoch 62/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0571 - accuracy:
0.9816 - val_loss: 0.0546 - val_accuracy: 0.9857
Epoch 63/75
94/94 [==============================] - 1s 13ms/step - loss: 0.0578 - accuracy:
```

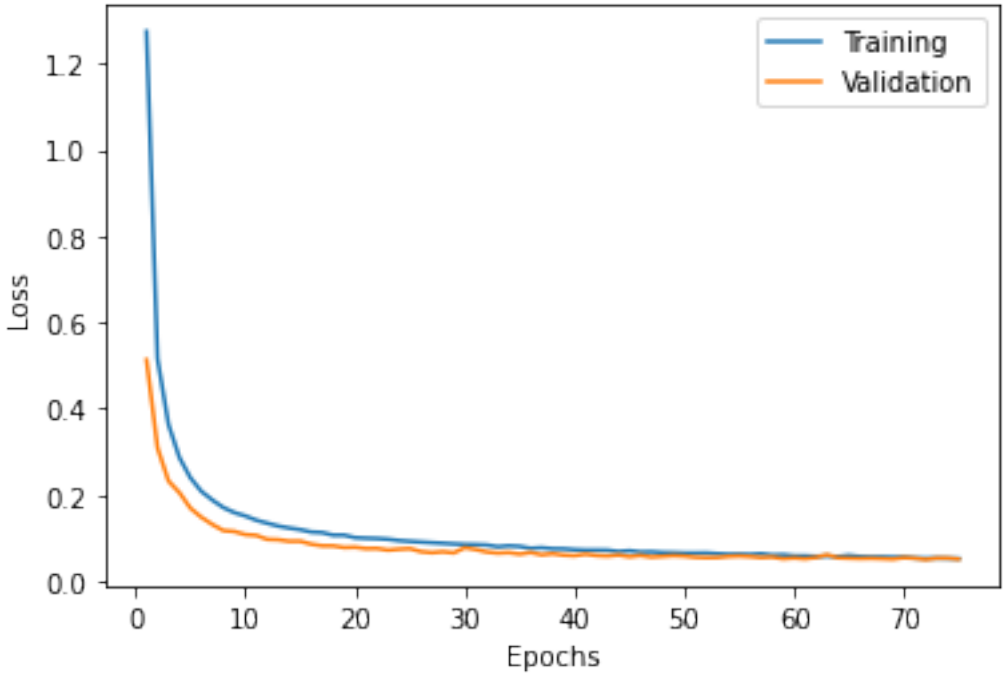
```
0.9816 - val_loss: 0.0607 - val_accuracy: 0.9829
Epoch 64/75
94/94 [==============================] - 1s 13ms/step - loss: 0.0566 - accuracy:
0.9821 - val_loss: 0.0538 - val_accuracy: 0.9847
Epoch 65/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0593 - accuracy:
0.9808 - val_loss: 0.0529 - val_accuracy: 0.9852
Epoch 66/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0565 - accuracy:
0.9818 - val_loss: 0.0521 - val_accuracy: 0.9845
Epoch 67/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0557 - accuracy:
0.9819 - val_loss: 0.0526 - val_accuracy: 0.9859
Epoch 68/75
94/94 [==============================] - 1s 13ms/step - loss: 0.0557 - accuracy:
0.9818 - val_loss: 0.0518 - val_accuracy: 0.9864
Epoch 69/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0550 - accuracy:
0.9821 - val_loss: 0.0507 - val_accuracy: 0.9858
Epoch 70/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0553 - accuracy:
0.9825 - val_loss: 0.0541 - val_accuracy: 0.9845
Epoch 71/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0531 - accuracy:
0.9829 - val_loss: 0.0528 - val_accuracy: 0.9849
Epoch 72/75
94/94 [==============================] - 1s 13ms/step - loss: 0.0526 - accuracy:
0.9824 - val_loss: 0.0497 - val_accuracy: 0.9868
Epoch 73/75
94/94 [==============================] - 1s 12ms/step - loss: 0.0530 - accuracy:
0.9822 - val_loss: 0.0530 - val_accuracy: 0.9855
Epoch 74/75
94/94 [==============================] - 1s 13ms/step - loss: 0.0524 - accuracy:
0.9832 - val_loss: 0.0531 - val_accuracy: 0.9859
Epoch 75/75
94/94 [==============================] - 1s 13ms/step - loss: 0.0519 - accuracy:
0.9828 - val_loss: 0.0506 - val_accuracy: 0.9864
```

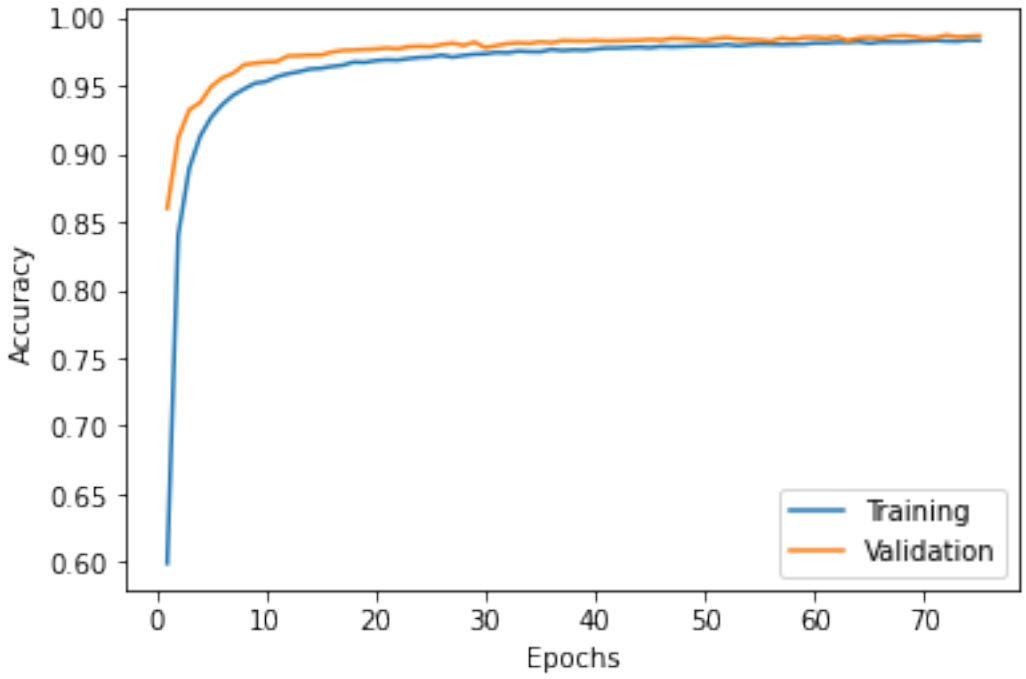
5 Analyze and comment the training results

There does not appear to be any indication of underfitting/overfitting. The two curves have similar trends and the loss function is relatively small.

[ ]:
```python
x_plot = list(range(1,epochs+1))

def plot_history(network_history):
    plt.figure()
```

```
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.plot(x_plot, network_history.history['loss'])
    plt.plot(x_plot, network_history.history['val_loss'])
    plt.legend(['Training', 'Validation'])

    plt.figure()
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.plot(x_plot, network_history.history['accuracy'])
    plt.plot(x_plot, network_history.history['val_accuracy'])
    plt.legend(['Training', 'Validation'], loc='lower right')
    plt.show()

plot_history(network_history)
```





6 Model evaluations and conclusions

The performance of the model is very good even using few parameters, the ratio between model complexity and results, using this type of network for this problem, is really appreciable.

[ ]:
```
# evaluate on test
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
print('Train loss:', network_history.history['loss'][-1])
print('Train accuracy:', network_history.history['accuracy'][-1])
```

```
Test loss: 0.046000394970178604
Test accuracy: 0.986299991607666
Train loss: 0.05190221220254898
Train accuracy: 0.9828125238418579
```

[ ]:
```
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('Nabil_ELAsri_826040_assignment3.ipynb')
```

```
--2021-11-26 17:40:39--  https://raw.githubusercontent.com/brpy/colab-
pdf/master/colab_pdf.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)...
```

```
185.199.109.133, 185.199.108.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.109.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1864 (1.8K) [text/plain]
Saving to: colab_pdf.py

colab_pdf.py        100%[===================>]   1.82K  --.-KB/s    in 0s

2021-11-26 17:40:40 (18.2 MB/s) - colab_pdf.py saved [1864/1864]

Mounted at /content/drive/

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

Extracting templates from packages: 100%
```