Statistical Methods for Machine Learning Report

Image Classification with Artificial Neural Networks

Maurina Gabriele
Università degli Studi di Milano

May 15, 2021

Abstract

This document contains a report on the project created by Gabriele Maurina for the exam Statistical Methods for Machine Learning. The project focuses on using artificial neural networks(ANNs) to perform image classification on a dataset from Keggle.com. The dataset contains 90380 images of fruits and vegetables. The project is divided into two experiments. The first experiment trains and evaluates several models with the objective to classify images based on 10 base types, namely "apple", "banana", "cherry", "grape", "peach", "pear", "pepper", "plum", "potato" and "tomato". The second experiment trains and evaluates several models with the objective to classify images based on all 131 labels contained in the dataset. The total number of networks trained and evaluated across both experiments is 300, of which 150 are fully connected deep neural networks (DNNs) and 150 are convolutional neural networks(CNNs). The experimental results show that on average CNNs achieve a lower Zero-One loss on the testing set than DNNs. Furthermore, in the experiment with only 10 labels, CNNs and DNNs achieve similar accuracy and loss, whereas in the experiment with 131 labels, CNNs tend to outperform non-CNNs on accuracy and loss. The best networks for experiment 1 and experiment 2 score very good results, with a zero-one loss on the test set of 1% and 5% respectively.

Contents

1 Introduction

Artificial neural networks (ANNs) are machine learning models loosely inspired by biological neural networks found in human and animal brains. ANNs consist of a set of nodes, called artificial neurons, connected to each other by edges. Nodes can send and receive signals, i.e. real numbers, through their edges.

Upon receiving a signal from other nodes, a node processes said signal and can signal other nodes. Tipically a node output is a function of all its inputs. Nodes can be aggregated into layers, where usually nodes from the first layer forward signals to nodes in the next layer, which, in turn, forward signals to the next layer and the forwarding is repeated until the last layer is reached.

1.1 Deep Neural Networks

1.2 Convolutional Neural Networks

2 Experiment 1

Experiment 1 trains and evaluates ANNs on the task to classify images of fruits and vegetables of 10 base types. The goal of the experiment is to try different network types in order to find the best configuration, i.e., the configuration with the lowest zero-one loss.

2.1 Setup

This experiment is carried out on a 64 bit machine running Fedora 32, with an Intel® Core™ i7-2670QM CPU @ 2.20GHz and 6GByte of RAM. THe programming language used is Python 3.8[4]. Tensorflow2[1], with the Keras[2] interface, is used to create, train and evaluate models. Numpy[3] is used to store and manipulate data easily.

Dataset. The dataset used is available on kaggle.com/moltean/fruits. It contains 90380 images of fruits and vegetables. Experiment 1 classifies images in 10 base types, namely "apple", "banana", "cherry", "grape", "peach", "pear", "pepper", "plum", "potato" and "tomato". Only the 43513 images that represents these fruit and vegetable types are considered. The images are 100x100px photos, already cropped to fit the subject perfectly.

The script `dataset_1.py` is used to manage the dataset. This script performs 5 tasks. 1)It loads images from the dataset. 2)It labels images according to the folder they are in. If the name of the folder starts with one of the 10 labels, the label is attached to the image. Otherwise the image is discarded. 3)It resizes images according to a preferred size. 4)It generates Numpy's n-dimensional arrays containing image data and labels such that they can be easily fed to a machine learning model. The ndarray for image data is of type unsigned 8-bit integer and its shape is `(ni,is,is,3)`, where `ni` is the number of images, `is` is the image side length in pixel, and 3 is the number of channels (r,g,b). The ndarray for labels is of type unsigned 8-bit integer and its shape is `(ni,)`, where `ni` is the number of images. Arrays are created both for training and testing sets. 5)Finally, the sctipt stores said arrays in binary format, so that it will load them faster at the next iteration, without having to go through the whole process again.

The script offers a simple interface, consisting of a single function `dataset(size)`, which takes care of all the aforementioned tasks in the background and returns the dataset, with images of the preferred size, ready to be used by a machine learning model. The `dataset(size)` function is declared as follows:

```
def dataset(size):
    '''Return dataset with images of preferred size.
    If dataset already exists, it is loaded from disk,
    otherwise it is created from the image folder.'''
    files = (
        join(dataset_folder,f'x_train_{size}.npy'),
        join(dataset_folder,f'y_train_{size}.npy'),
        join(dataset_folder,f'x_test_{size}.npy'),
        join(dataset_folder,f'y_test_{size}.npy'))
    for f in files:
        if not isfile(f):
            ds = create_dataset(size)
            save_dataset(*ds,size)
            return ds
    return load_dataset(size)
```

Models. The models used differ in size, shape and type. The script `models.py` handles the creation of models according to certain parameters. The parameters supported are depth of the network, i.e., how many layers it has, width of the network, i.e., how many nodes are in each layer, type of network, i.e., DNN or CNN. Furthermore the script can handle different input and output sizes according to the needs.

The script offers a simple interface, consisting of a generator that yields models ready to be tested. The generator is called using the function `models(input_size, output_size)`. For this experiment, the generator is tuned to return 30 different networks. The number 30 is the result of the combination of 2 types of networks (DNN and CNN), 3 depths (1,2 or 3 layers), 5 widths (multiples of 32 in the case od DNN and multiples of 8 in the case of CNN).The `models(input_size, output_size)` function is coded as follows:

```
def models(input_size,output_size):
    '''This generator returns models to test in the experiment.'''
    #dense layers, different sizes
    for i in range(1,4):
        for j in range(1,6):
            yield dense(input_size,output_size,i,j*32),i,j*32,'dense'
    #cnn, different sizes
    for i in range(1,4):
        for j in range(1,6):
            yield conv(input_size,output_size,i,j*8),i,j*8,'conv'
```

The generator uses two more support functions to create the networks. These functions are `dense(input_size, output_size,depth,size)` and `conv(input_size, output_size,depth,size)`, and they are coded as follows:

```python
def dense(input_size,output_size,depth,size):
    '''Create a dense model with specific input_size,
    output_size,depth and number of neuros.'''
    layers = [tf.keras.layers.Flatten(input_shape=(input_size,
        input_size,3))]
    for i in range(depth):
        layers.append(tf.keras.layers.Dense(size,activation='relu'))
    layers.append(tf.keras.layers.Dense(output_size))
    return tf.keras.Sequential(layers)
def conv(input_size,output_size,depth,size):
    '''Create a conv model with specific input_size,
    output_size,depth and number of neuros.'''
    layers = [tf.keras.layers.Conv2D(size,(3, 3),activation='relu',
        input_shape=(input_size,input_size,3))]
    for i in range(depth-1):
        layers += [
            tf.keras.layers.MaxPooling2D((2, 2)),
            tf.keras.layers.Conv2D(size,(3, 3),activation='relu',
                padding='same')]
    layers += [
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(size,activation='relu'),
        tf.keras.layers.Dense(output_size)]
    return tf.keras.Sequential(layers)
```

CNNs are build alternating a `Conv2D` layer with a `MaxPooling2D` layer and finishing with a `Dense` layer, this was suggested on Tensorflow's website. The activation function used is ReLU, i.e., Rectified Linear Unit, which is one of the most commonly used with ANN. In the future, the experiment could be expanded by trying different activation functions.

Execution. The exectution of the experiment is managed by the `experiment_1.py` script. The script loads training an test datasets of different sizes using the API of `dataset_1.py`. The script then loads models from `models.py`. Finally the script trains the networks on the training set and evaluates them on the test set. For each network the training history is saved in JSON format and the zero-one loss wrt. the test set is logged in a CSV file. Zero-one loss is computed by counting how many predictions on the test set are wrong and dividing the total by the number of images in the test set. Zero-one loss is a real number between 0 and 1, where 0 means a perfect set of predictions and 1 means a completely wrong set of predictions.

2.2 Results

The results collected are summarized in table1. The table contains the list of 150 ANNs trained and evaluated, with their parameters, i.e., depth, width and type, the number of epochs they were trained on and the Zero-One loss achieved

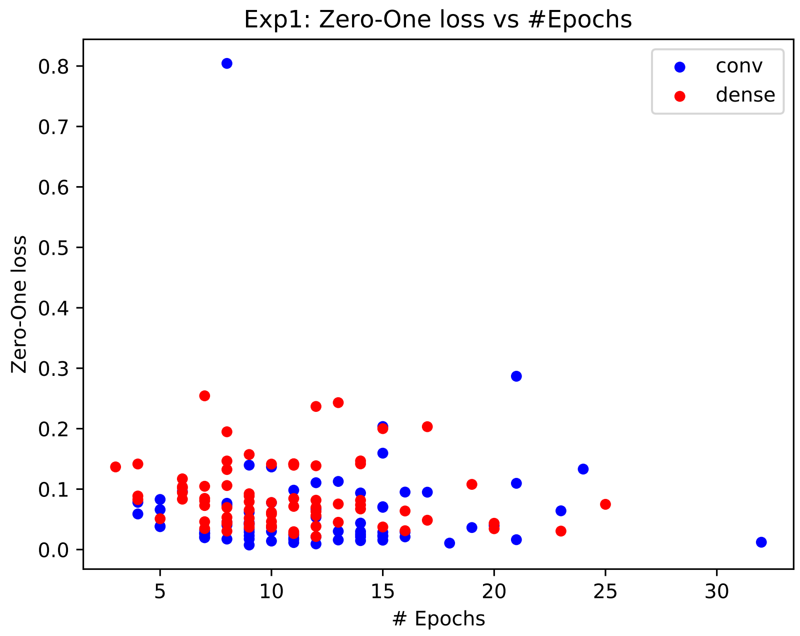

Figure 1: Zero-one loss vs epochs

by the traned ANNs on the testing set. Figure1 shows the zero-one loss and the number of epochs of each ANNs, differentiating the DNNs from the CNNs. On average the CNNs achieve a lower loss.

2.3 Reproducing experiment

3 Experiment 2

Experiment 2, similarly to experiment 1, trains and evaluates ANNs on the task to classify images of fruits and vegetables. Experiment 2, however, considers 131 labels, making the classification task more challenging. The goal of the experiment, as in experiment 1, is to try different network types in order to find the best configuration, i.e., the configuration with the lowest Zero-One loss, and to see if the best configuration for experiment 1, still holds for experiment 2.

3.1 Setup

3.2 Results

The results collected are summarized in table2.

Table 1: Experiment 1

IS	T	D	W	ZOL	#E	IS	T	D	W	ZOL	#E	IS	T	D	W	ZOL	#E
10	dense	1	32	0.07	25	20	conv	2	8	0.07	15	40	dense	3	32	0.24	13
10	dense	1	64	0.03	20	20	conv	2	16	0.03	15	40	dense	3	64	0.10	7
10	dense	1	96	0.03	23	20	conv	2	24	0.02	15	40	dense	3	96	0.09	6
10	dense	1	128	0.05	13	20	conv	2	32	0.01	10	40	dense	3	128	0.10	6
10	dense	1	160	0.08	12	20	conv	2	40	0.02	11	40	dense	3	160	0.03	8
10	dense	2	32	0.04	20	20	conv	3	8	0.11	12	40	conv	1	8	0.08	8
10	dense	2	64	0.07	8	20	conv	3	16	0.02	12	40	conv	1	16	0.11	13
10	dense	2	96	0.09	9	20	conv	3	24	0.01	9	40	conv	1	24	0.07	5
10	dense	2	128	0.06	10	20	conv	3	32	0.04	5	40	conv	1	32	0.07	12
10	dense	2	160	0.04	10	20	conv	3	40	0.02	8	40	conv	1	40	0.04	14
10	dense	3	32	0.08	11	30	dense	1	32	0.20	15	40	conv	2	8	0.16	15
10	dense	3	64	0.05	17	30	dense	1	64	0.11	19	40	conv	2	16	0.03	9
10	dense	3	96	0.03	11	30	dense	1	96	0.07	14	40	conv	2	24	0.03	9
10	dense	3	128	0.02	12	30	dense	1	128	0.08	14	40	conv	2	32	0.03	7
10	dense	3	160	0.07	9	30	dense	1	160	0.06	10	40	conv	2	40	0.02	7
10	conv	1	8	0.10	16	30	dense	2	32	0.14	12	40	conv	3	8	0.10	11
10	conv	1	16	0.04	20	30	dense	2	64	0.11	8	40	conv	3	16	0.04	9
10	conv	1	24	0.14	9	30	dense	2	96	0.05	8	40	conv	3	24	0.01	14
10	conv	1	32	0.02	16	30	dense	2	128	0.06	12	40	conv	3	32	0.02	21
10	conv	1	40	0.02	14	30	dense	2	160	0.08	4	40	conv	3	40	0.01	18
10	conv	2	8	0.06	23	30	dense	3	32	0.07	14	50	dense	1	32	0.25	7
10	conv	2	16	0.07	8	30	dense	3	64	0.10	6	50	dense	1	64	0.14	11
10	conv	2	24	0.03	7	30	dense	3	96	0.07	7	50	dense	1	96	0.14	11
10	conv	2	32	0.02	9	30	dense	3	128	0.05	9	50	dense	1	128	0.15	8
10	conv	2	40	0.02	11	30	dense	3	160	0.09	4	50	dense	1	160	0.08	13
10	conv	3	8	0.11	21	30	conv	1	8	0.29	21	50	dense	2	32	0.19	8
10	conv	3	16	0.06	9	30	conv	1	16	0.04	9	50	dense	2	64	0.12	6
10	conv	3	24	0.10	6	30	conv	1	24	0.08	4	50	dense	2	96	0.04	9
10	conv	3	32	0.02	13	30	conv	1	32	0.03	10	50	dense	2	128	0.05	8
10	conv	3	40	0.01	11	30	conv	1	40	0.06	4	50	dense	2	160	0.14	3
20	dense	1	32	0.20	17	30	conv	2	8	0.14	10	50	dense	3	32	0.08	7
20	dense	1	64	0.07	11	30	conv	2	16	0.04	9	50	dense	3	64	0.07	12
20	dense	1	96	0.03	16	30	conv	2	24	0.02	7	50	dense	3	96	0.09	9
20	dense	1	128	0.04	12	30	conv	2	32	0.03	14	50	dense	3	128	0.05	5
20	dense	1	160	0.04	15	30	conv	2	40	0.02	7	50	dense	3	160	0.03	7
20	dense	2	32	0.08	9	30	conv	3	8	0.20	15	50	conv	1	8	0.80	8
20	dense	2	64	0.08	7	30	conv	3	16	0.04	8	50	conv	1	16	0.09	14
20	dense	2	96	0.06	12	30	conv	3	24	0.03	7	50	conv	1	24	0.08	5
20	dense	2	128	0.05	10	30	conv	3	32	0.02	9	50	conv	1	32	0.04	10
20	dense	2	160	0.03	11	30	conv	3	40	0.01	12	50	conv	1	40	0.04	19
20	dense	3	32	0.06	16	40	dense	1	32	0.14	11	50	conv	2	8	0.07	15
20	dense	3	64	0.04	10	40	dense	1	64	0.13	8	50	conv	2	16	0.05	12
20	dense	3	96	0.08	10	40	dense	1	96	0.14	14	50	conv	2	24	0.03	7
20	dense	3	128	0.04	8	40	dense	1	128	0.15	14	50	conv	2	32	0.02	15
20	dense	3	160	0.05	7	40	dense	1	160	0.16	9	50	conv	2	40	0.04	5
20	conv	1	8	0.13	24	40	dense	2	32	0.24	12	50	conv	3	8	0.08	10
20	conv	1	16	0.09	17	40	dense	2	64	0.14	10	50	conv	3	16	0.03	13
20	conv	1	24	0.05	8	40	dense	2	96	0.08	6	50	conv	3	24	0.03	16
20	conv	1	32	0.03	14	40	dense	2	128	0.14	4	50	conv	3	32	0.01	32
20	conv	1	40	0.02	11	40	dense	2	160	0.04	9	50	conv	3	40	0.02	15

IS is image size in pixel;

T is the type of model, can be ”dense” or ”conv”;

D is the depth of the model;

W is the width of the model;

ZOL is the Zero-One loss achieved on the test set;

#E is the number of epochs the model was trained for.

Table 2: Experiment 2

IS	**T**	**D**	**W**	**ZOL**	**#E**	**IS**	**T**	**D**	**W**	**ZOL**	**#E**	**IS**	**T**	**D**	**W**	**ZOL**	**#E**
10	dense	1	32	0.07	28	20	conv	2	8	0.12	17	40	dense	3	32	0.38	10
10	dense	1	64	0.08	22	20	conv	2	16	0.09	11	40	dense	3	64	0.18	6
10	dense	1	96	0.08	17	20	conv	2	24	0.14	7	40	dense	3	96	0.10	8
10	dense	1	128	0.08	14	20	conv	2	32	0.05	12	40	dense	3	128	0.12	5
10	dense	1	160	0.07	11	20	conv	2	40	0.09	5	40	dense	3	160	0.16	7
10	dense	2	32	0.09	16	20	conv	3	8	0.24	12	40	conv	1	8	0.44	12
10	dense	2	64	0.09	12	20	conv	3	16	0.08	11	40	conv	1	16	0.07	10
10	dense	2	96	0.07	9	20	conv	3	24	0.10	7	40	conv	1	24	0.99	7
10	dense	2	128	0.09	10	20	conv	3	32	0.12	5	40	conv	1	32	0.08	10
10	dense	2	160	0.07	9	20	conv	3	40	0.06	7	40	conv	1	40	0.14	12
10	dense	3	32	0.10	16	30	dense	1	32	0.95	22	40	conv	2	8	0.13	13
10	dense	3	64	0.11	8	30	dense	1	64	0.37	26	40	conv	2	16	0.19	14
10	dense	3	96	0.08	11	30	dense	1	96	0.30	25	40	conv	2	24	0.12	8
10	dense	3	128	0.12	7	30	dense	1	128	0.13	17	40	conv	2	32	0.08	5
10	dense	3	160	0.13	7	30	dense	1	160	0.13	12	40	conv	2	40	0.04	8
10	conv	1	8	0.30	19	30	dense	2	32	0.34	14	40	conv	3	8	0.13	13
10	conv	1	16	0.11	20	30	dense	2	64	0.16	10	40	conv	3	16	0.22	13
10	conv	1	24	0.11	10	30	dense	2	96	0.08	8	40	conv	3	24	0.07	11
10	conv	1	32	0.06	17	30	dense	2	128	0.11	6	40	conv	3	32	0.07	8
10	conv	1	40	0.07	10	30	dense	2	160	0.11	5	40	conv	3	40	0.05	5
10	conv	2	8	0.19	16	30	dense	3	32	0.55	17	50	dense	1	32	0.99	6
10	conv	2	16	0.12	16	30	dense	3	64	0.14	11	50	dense	1	64	0.99	4
10	conv	2	24	0.09	8	30	dense	3	96	0.12	7	50	dense	1	96	0.53	18
10	conv	2	32	0.10	8	30	dense	3	128	0.08	13	50	dense	1	128	0.99	4
10	conv	2	40	0.10	6	30	dense	3	160	0.13	5	50	dense	1	160	0.53	16
10	conv	3	8	0.21	21	30	conv	1	8	0.17	15	50	dense	2	32	0.99	4
10	conv	3	16	0.11	12	30	conv	1	16	0.68	18	50	dense	2	64	0.91	14
10	conv	3	24	0.09	15	30	conv	1	24	0.67	18	50	dense	2	96	0.89	13
10	conv	3	32	0.14	7	30	conv	1	32	0.27	21	50	dense	2	128	0.31	15
10	conv	3	40	0.05	12	30	conv	1	40	0.14	14	50	dense	2	160	0.16	5
20	dense	1	32	0.52	23	30	conv	2	8	0.21	14	50	dense	3	32	0.90	14
20	dense	1	64	0.31	28	30	conv	2	16	0.10	9	50	dense	3	64	0.56	12
20	dense	1	96	0.12	22	30	conv	2	24	0.04	9	50	dense	3	96	0.16	7
20	dense	1	128	0.09	14	30	conv	2	32	0.07	6	50	dense	3	128	0.14	5
20	dense	1	160	0.09	12	30	conv	2	40	0.07	8	50	dense	3	160	0.15	4
20	dense	2	32	0.31	13	30	conv	3	8	0.40	30	50	conv	1	8	0.99	6
20	dense	2	64	0.08	8	30	conv	3	16	0.13	8	50	conv	1	16	0.39	13
20	dense	2	96	0.09	11	30	conv	3	24	0.08	7	50	conv	1	24	0.41	12
20	dense	2	128	0.10	10	30	conv	3	32	0.09	5	50	conv	1	32	0.21	17
20	dense	2	160	0.14	6	30	conv	3	40	0.06	9	50	conv	1	40	0.15	8
20	dense	3	32	0.13	16	40	dense	1	32	0.94	31	50	conv	2	8	0.39	23
20	dense	3	64	0.11	9	40	dense	1	64	0.81	20	50	conv	2	16	0.27	15
20	dense	3	96	0.12	4	40	dense	1	96	0.30	27	50	conv	2	24	0.22	17
20	dense	3	128	0.09	8	40	dense	1	128	0.41	13	50	conv	2	32	0.11	6
20	dense	3	160	0.11	4	40	dense	1	160	0.24	13	50	conv	2	40	0.06	9
20	conv	1	8	0.80	28	40	dense	2	32	0.99	10	50	conv	3	8	0.49	14
20	conv	1	16	0.37	28	40	dense	2	64	0.27	14	50	conv	3	16	0.10	6
20	conv	1	24	0.10	10	40	dense	2	96	0.11	9	50	conv	3	24	0.07	5
20	conv	1	32	0.11	7	40	dense	2	128	0.11	12	50	conv	3	32	0.07	5
20	conv	1	40	0.07	6	40	dense	2	160	0.18	4	50	conv	3	40	0.05	6

IS is the image size in pixel;

T is the type of model, can be ”dense” or ”conv”;

D is the depth of the model;

W is the width of the model;

ZOL is the Zero-One loss achieved on the test set;

#E is the number of epochs the model was trained for.

3.3 Reproducing experiment

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] François Chollet et al. Keras. `https://keras.io`, 2015.

[3] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[4] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.