Midterm2 Assignment 3

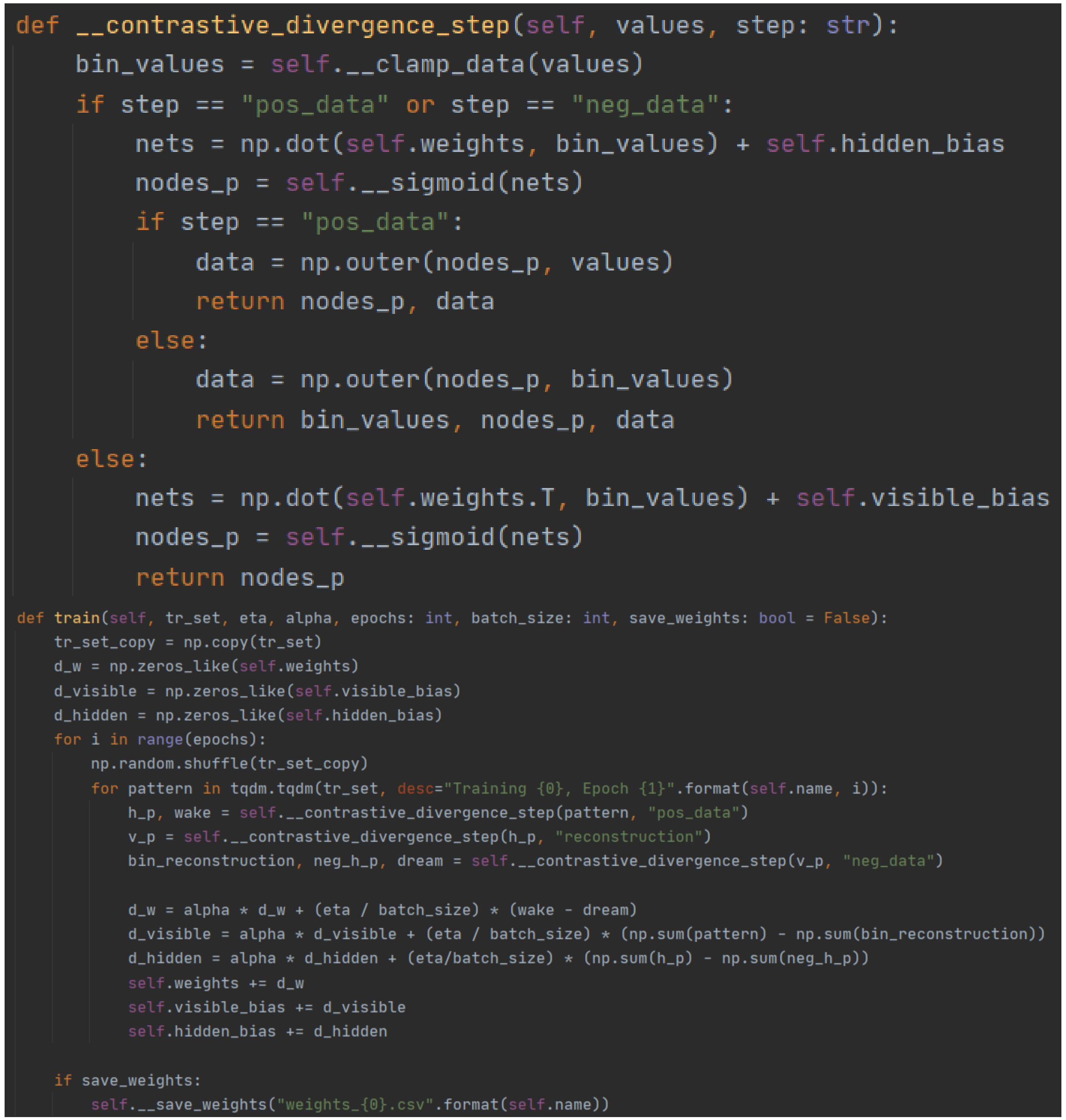
Intelligent Systems for Pattern Recognition

Master Degree in Computer Science, AI Curriculum

A.Y. 2020/2021

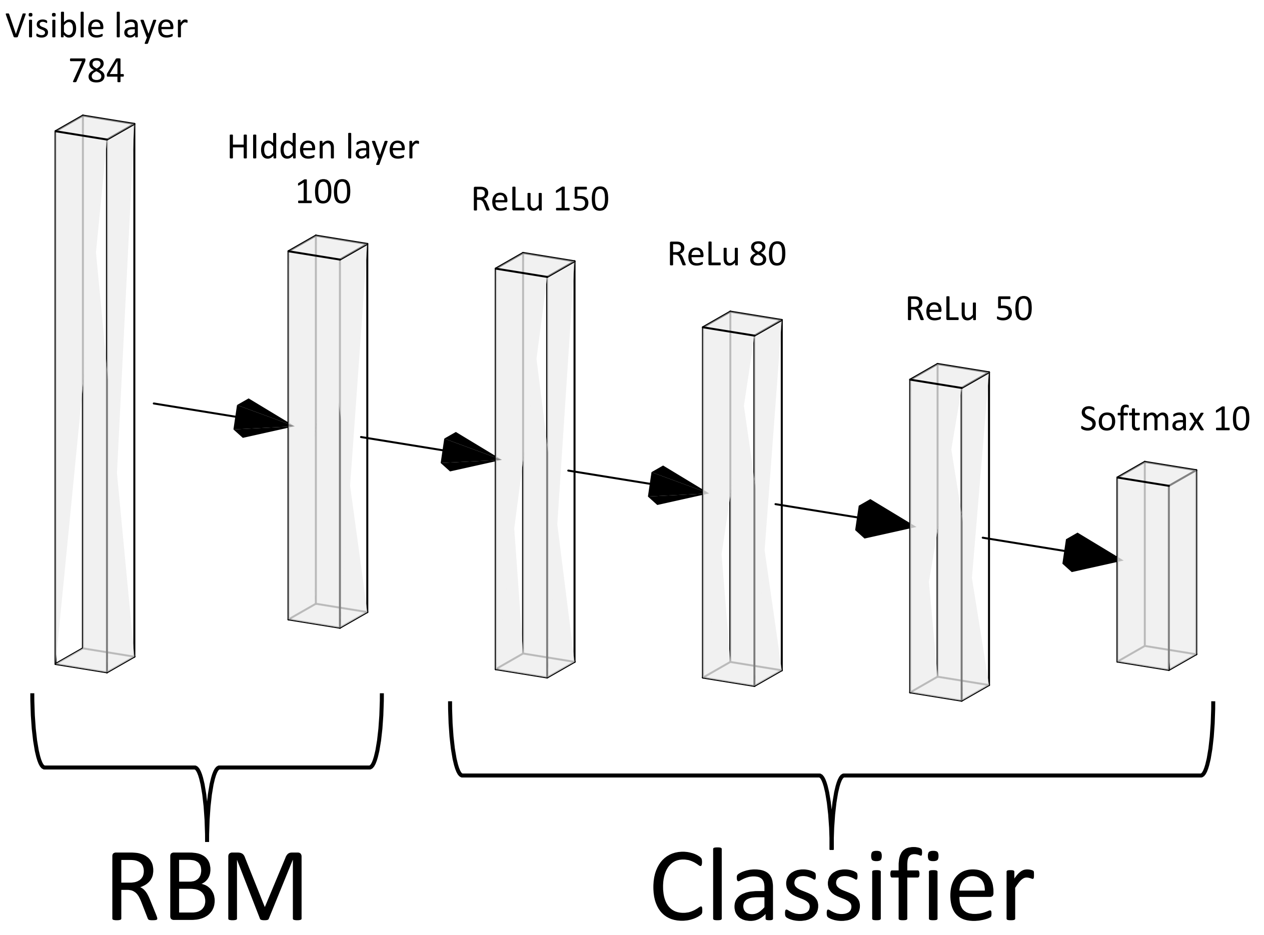
Alessandro Ristori

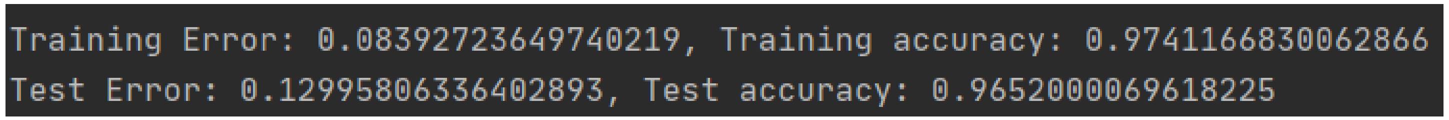
Main RBM Code

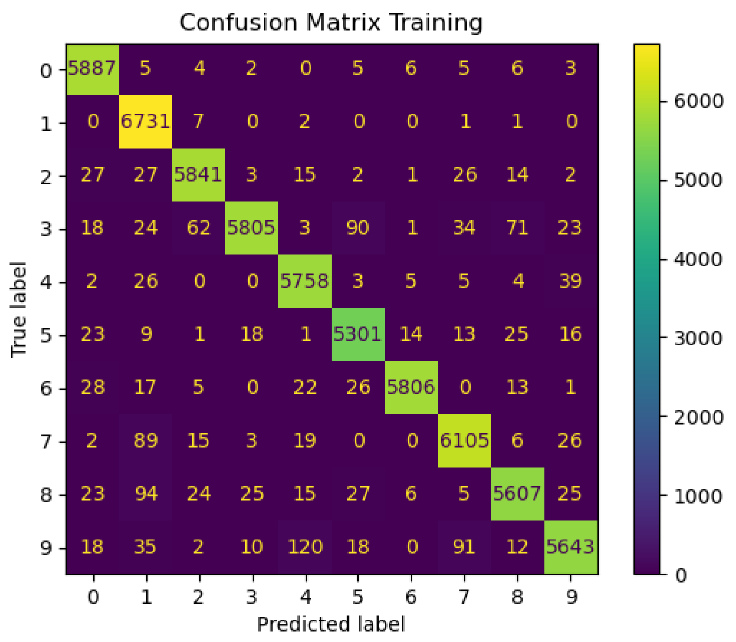


```python
def __contrastive_divergence_step(self, values, step: str):
    bin_values = self.__clamp_data(values)
    if step == "pos_data" or step == "neg_data":
        nets = np.dot(self.weights, bin_values) + self.hidden_bias
        nodes_p = self.__sigmoid(nets)
        if step == "pos_data":
            data = np.outer(nodes_p, values)
            return nodes_p, data
        else:
            data = np.outer(nodes_p, bin_values)
            return bin_values, nodes_p, data
    else:
        nets = np.dot(self.weights.T, bin_values) + self.visible_bias
        nodes_p = self.__sigmoid(nets)
        return nodes_p
def train(self, tr_set, eta, alpha, epochs: int, batch_size: int, save_weights: bool = False):
    tr_set_copy = np.copy(tr_set)
    d_w = np.zeros_like(self.weights)
    d_visible = np.zeros_like(self.visible_bias)
    d_hidden = np.zeros_like(self.hidden_bias)
    for i in range(epochs):
        np.random.shuffle(tr_set_copy)
        for pattern in tqdm.tqdm(tr_set, desc="Training {0}, Epoch {1}".format(self.name, i)):
            h_p, wake = self.__contrastive_divergence_step(pattern, "pos_data")
            v_p = self.__contrastive_divergence_step(h_p, "reconstruction")
            bin_reconstruction, neg_h_p, dream = self.__contrastive_divergence_step(v_p, "neg_data")

            d_w = alpha * d_w + (eta / batch_size) * (wake - dream)
            d_visible = alpha * d_visible + (eta / batch_size) * (np.sum(pattern) - np.sum(bin_reconstruction))
            d_hidden = alpha * d_hidden + (eta/batch_size) * (np.sum(h_p) - np.sum(neg_h_p))
            self.weights += d_w
            self.visible_bias += d_visible
            self.hidden_bias += d_hidden

    if save_weights:
        self.__save_weights("weights_{0}.csv".format(self.name))
```

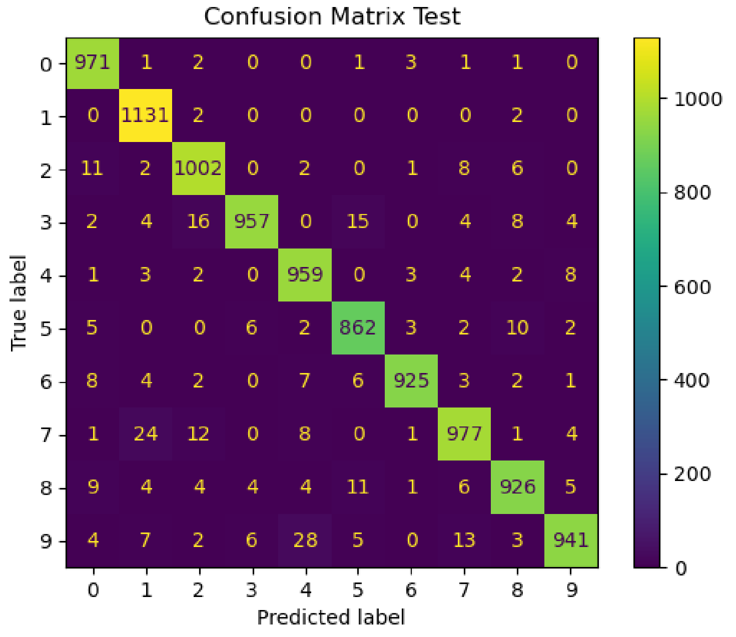
Model and confusion matrices




```
Training Error: 0.08392723649740219, Training accuracy: 0.9741166830062866
Test Error: 0.12995806336402893, Test accuracy: 0.9652000069618225
```

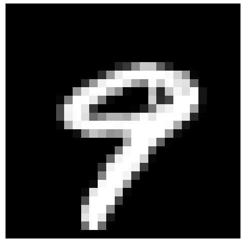


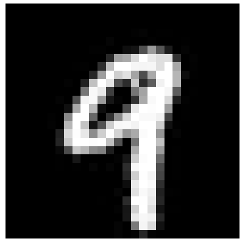
	0	1	2	3	4	5	6	7	8	9
0	5887	5	4	2	0	5	6	5	6	3
1	0	6731	7	0	2	0	0	1	1	0
2	27	27	5841	3	15	2	1	26	14	2
3	18	24	62	5805	3	90	1	34	71	23
4	2	26	0	0	5758	3	5	5	4	39
5	23	9	1	18	1	5301	14	13	25	16
6	28	17	5	0	22	26	5806	0	13	1
7	2	89	15	3	19	0	0	6105	6	26
8	23	94	24	25	15	27	6	5	5607	25
9	18	35	2	10	120	18	0	91	12	5643

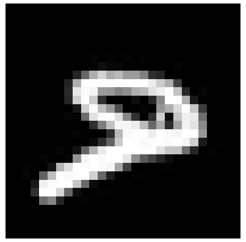


	0	1	2	3	4	5	6	7	8	9
0	971	1	2	0	0	1	3	1	1	0
1	0	1131	2	0	0	0	0	0	2	0
2	11	2	1002	0	2	0	1	8	6	0
3	2	4	16	957	0	15	0	4	8	4
4	1	3	2	0	959	0	3	4	2	8
5	5	0	0	6	2	862	3	2	10	2
6	8	4	2	0	7	6	925	3	2	1
7	1	24	12	0	8	0	1	977	1	4
8	9	4	4	4	4	11	1	6	926	5
9	4	7	2	6	28	5	0	13	3	941

What if we rotate the digits by a few radiants?

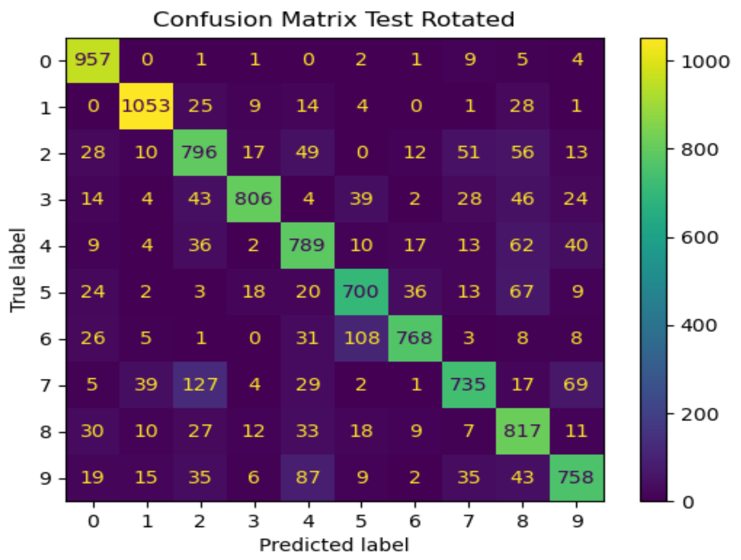

Original digit


∏/6 rotation


-∏/6 rotation

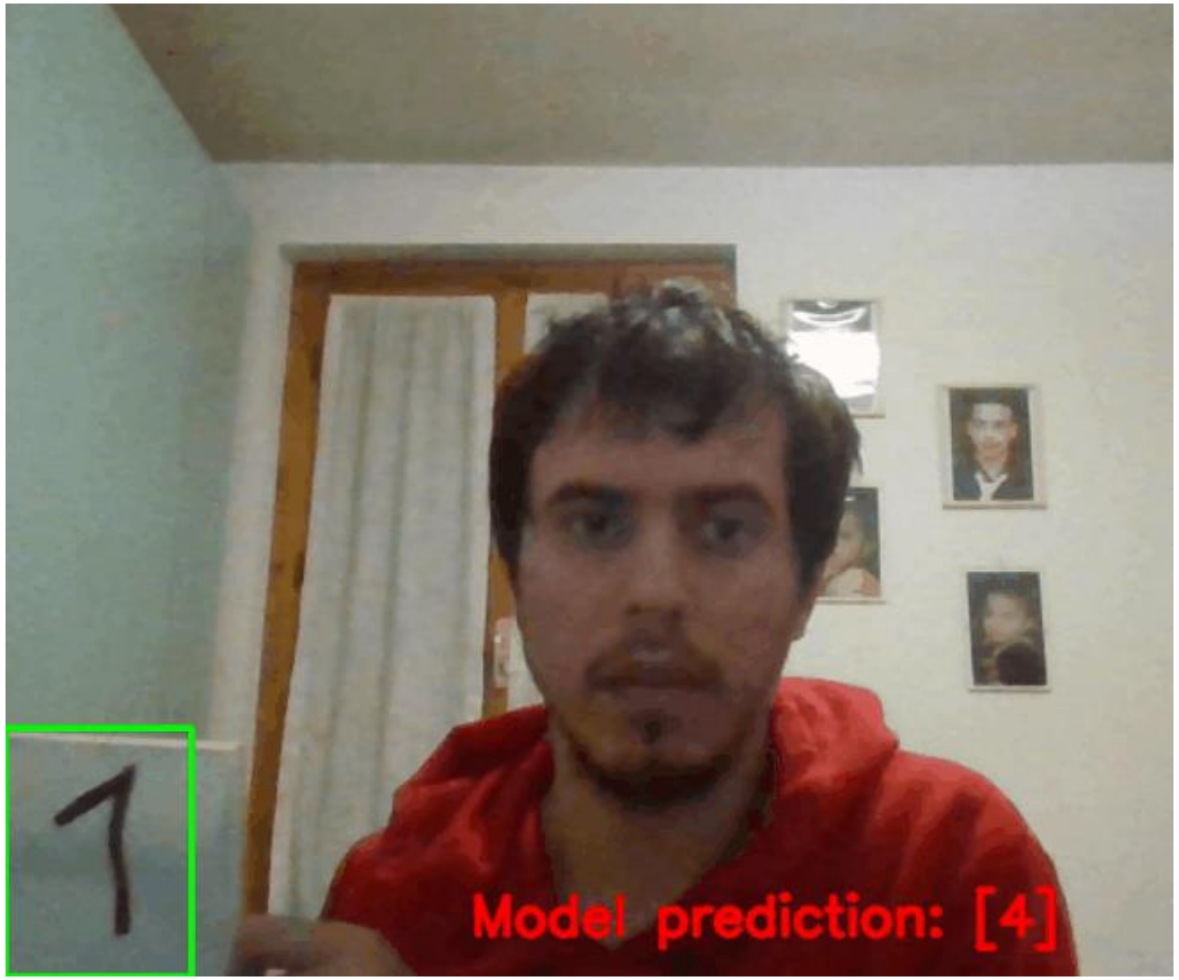
The model was tested on new data derived from the original test set: each pattern/data was randomly rotated over an interval [-∏/6, ∏/6].

It seemed that the model was somewhat sensitive towards the rotations (as expected), even the most insignificant ones; so i tried testing it in real time with opencv.


Confusion Matrix Test Rotated

True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	957	0	1	1	0	2	1	9	5	4
1	0	1053	25	9	14	4	0	1	28	1
2	28	10	796	17	49	0	12	51	56	13
3	14	4	43	806	4	39	2	28	46	24
4	9	4	36	2	789	10	17	13	62	40
5	24	2	3	18	20	700	36	13	67	9
6	26	5	1	0	31	108	768	3	8	8
7	5	39	127	4	29	2	1	735	17	69
8	30	10	27	12	33	18	9	7	817	11
9	19	15	35	6	87	9	2	35	43	758

1000
800
600
400
200
0



Final considerations