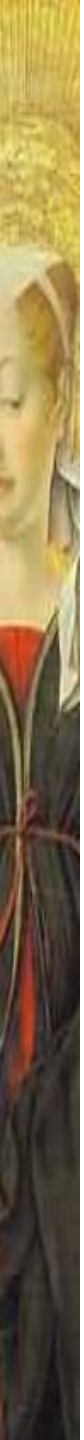
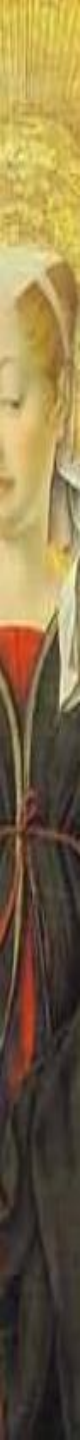


Topics in Machine Learning

CS212 Topics in Computing 2
Second Half
Lecture 4

Term 2 2019

Dr. Dmitri Roussinov
dmitri.roussinov@strath.ac.uk



Review: Perceptron and its Training Algorithm

Prediction:

a = 0

For i from 1 to length(W) do:

a += W[i] * X[i]

Y = 1 if a >= 0, 0 otherwise

Training:

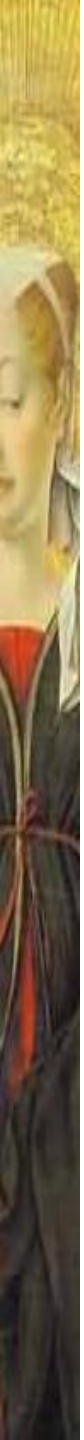
While error exists:

For all training examples do:

error = label - prediction

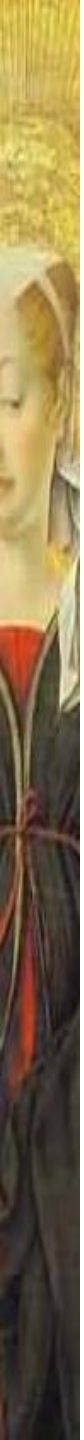
For i from 1 to length(W) do:

W[i] += learning_rate * error * X[i]



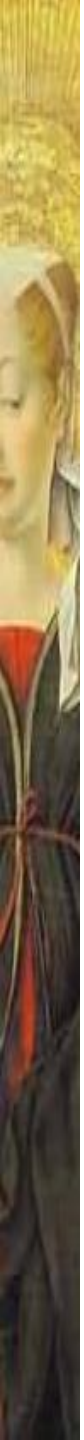
Extra Reading Links On Perceptrons

- (not to be tested)
- https://towardsdatascience.com/perceptron-the-artificial-neuron-4d8c70d5cc8d
- http://www.inf.fu-berlin.de/inst/ag-ki/rojas_home/documents/1996/NeuralNetworks/K3.pdf
- http://www.cs.cmu.edu/~roni/10601-slides/ch4-x4.pdf



Learned So far

- How a perceptron can make predictions
- What types of data it can learn
  - E.g. XOR vs AND
- Actual learning algorithm

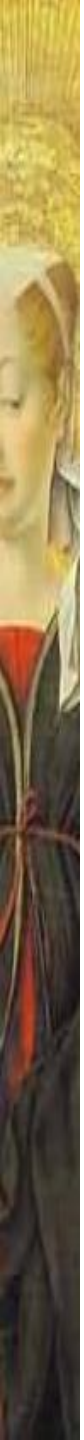


Plan for Today/Next

- Go over how to combine
  - your trainable perceptron from Lab 2
  - with image handing part

  to implement our first computer-vision application that can read hand-written digits

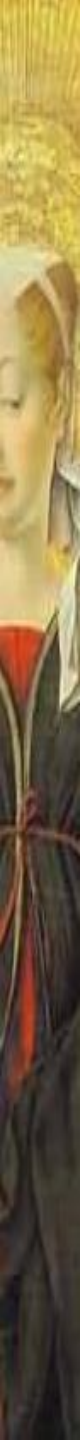
    Still only with possible 2 outcomes
- (Possibly) start multi-class perceptron
  - So next week we can handle any digits (10 possible classes)



Today's Lab Tasks

- Create a trainable perceptron that can understand hand-written digits
- …at least somewhat

MNIST task



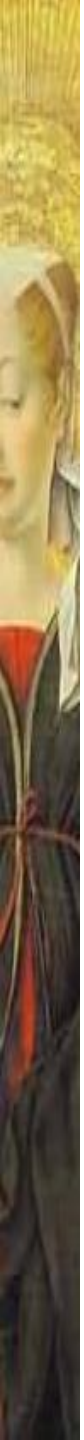
High Level Plan to Test Any Machine Learning Model

- Read training set
- Read testing set
- For certain number of epochs do:

  For each data point d in the training set do:

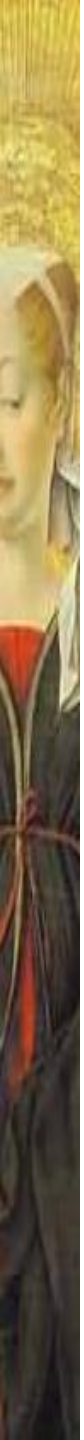
    Train the model using d as example

  Test the performance on the training and testing sets



Skeleton of Java Implementation

```
public static void main(String[] args) throws Exception  {



    Image[] data_train = Read("E:\\212-labs\\train.txt");
    Image[] data_test = Read("E:\\212-labs\\test.txt");

    for (int e = 0; e<100; e++) {

        for(int j = 0;j<data_train.length; j++)
            Train(data_train[j]);

        System.out.print("epoch:");
        System.out.print(e);
        System.out.print(" train:");
        PrintAccuracy(data_train);
        System.out.print(" test:");
        PrintAccuracy(data_test);
        System.out.println("");
    }
}
```



To Note

- We test on data that is different from training data to simulate real-life situations
- We print accuracies both on training and testing sets
- You can vary number of epochs

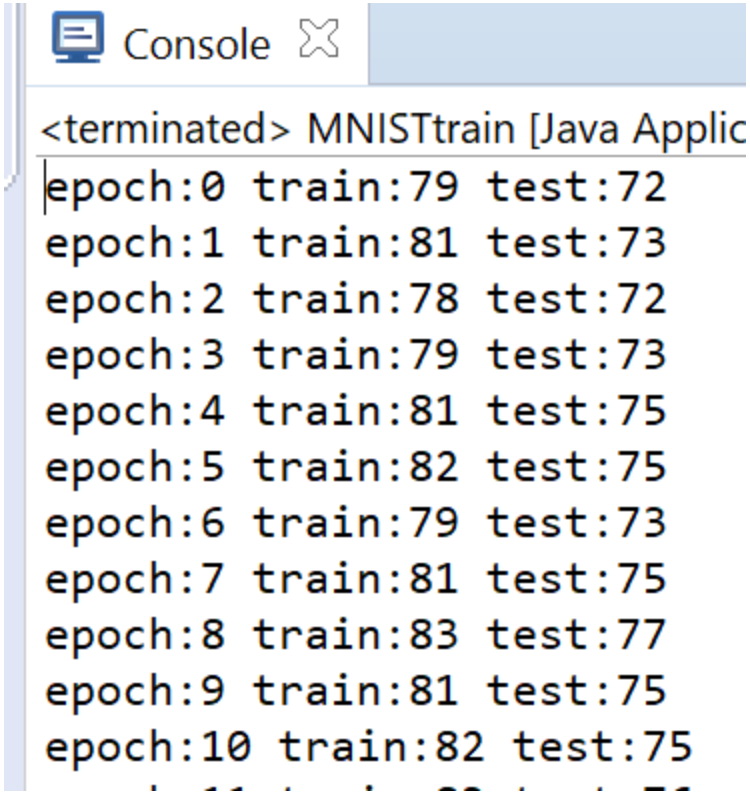
How to Measure the Accuracy

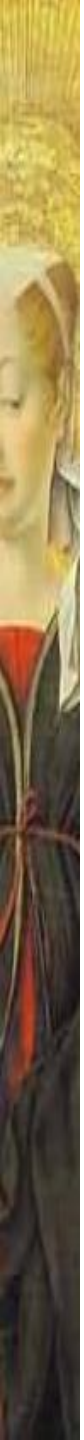
```
public static void PrintAccuracy(Image data[]) {
    float err = 0;
    for(int j = 0;j<data.length; j++) {
        int p = output(data[j].pixels);
        if (p != data[j].label)
            err += 1;
    }
    System.out.print(100 - (int)(100*err/data.length));
}
```

To note:

- We report the proportion of correct predictions
- For this, we count the number of errors
- Rounding to integers done by operator *(int)*

The output


Console
<terminated> MNISTtrain [Java Applic
epoch:0 train:79 test:72
epoch:1 train:81 test:73
epoch:2 train:78 test:72
epoch:3 train:79 test:73
epoch:4 train:81 test:75
epoch:5 train:82 test:75
epoch:6 train:79 test:73
epoch:7 train:81 test:75
epoch:8 train:83 test:77
epoch:9 train:81 test:75
epoch:10 train:82 test:75



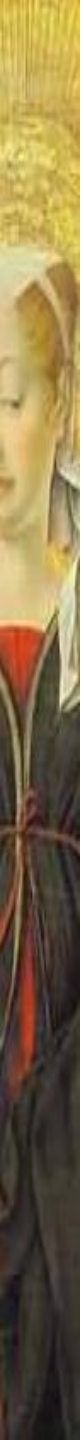
What is our performance?

- We will simply take the best accuracy reported on the test set
- In real applications, a third set called a validation set is used to decide when to stop and what parameters and algorithms to use
  - e.g. learning rate, using bias or not, number of epochs etc.
- Validation set should not overlap with neither training nor testing sets

Recall Our Image Class

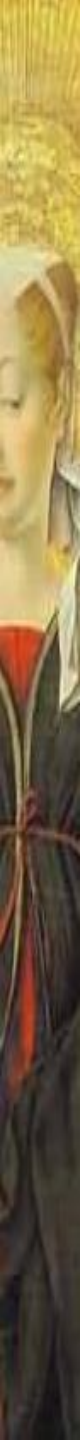
```
static int  image_size = 28 * 28;

static class Image {
    double[] pixels; // 28x28 = 784 numbers in flat array
    int label;

    Image() {
        pixels = new double[image_size];
    }
    void set(int pixel, int value) {
        pixels[pixel] = value;
    }
    void set_label(int value) {
        label = value;
    }
}
```

- Important: we will definitely need a flat array of 784 pixels!



So prediction function should look like this:

```
public class MNISTtrain {

    public static void check(boolean e) {
        if (!e){
            System.out.println("unexpected result");
            System.exit(0);
        }
    }

    public static int output(double X[]) {

        double a = 0;
        check(X.length == W.length);
        for(int i = 0; i < X.length; i++)
            a += W[i]*X[i];
        if (a >= 0)
            return 1;
        else
            return 0;

    }
```



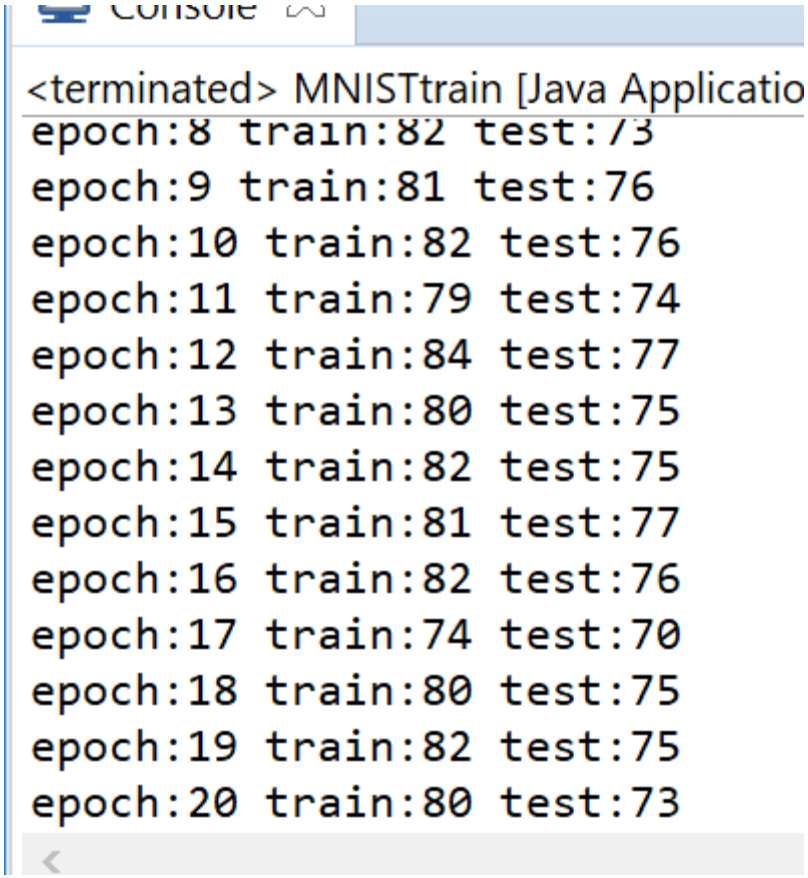
Binarizing Classes

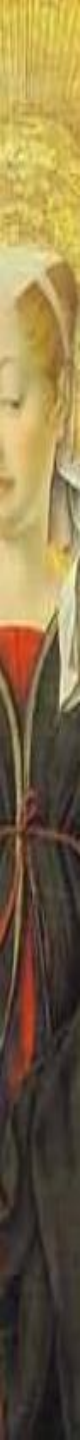
- So far we only allowed perceptron to predict two possible outcomes (0 or 1)
- But we have 10 possible digits: 0,1,…9
- What do we do?
- We train it to tell between two groups of the digits {0,1,2,3,4} and {5,6,7,8,9} !
- Next lecture we will look at how we can handle more than 2 classes

So, the small change to reading images:

```
for(int i = 0; i < number_of_images; i++) {
  t[i] =  new Image();
  for(int row = 0; row < 28; row ++) {

      st = br.readLine();
      String[] line_parts = st.split("[\t ]+"
      int label = Integer.parseInt(line_parts
      for(int pixel = 0; pixel < 28; pixel++)
          int value = Integer.parseInt((line_
          t[i].set(row * 28 + pixel, value);
      }
      if(label >= 5)
          t[i].set_label(1);
      else
          t[i].set_label(0);

  }
}
```

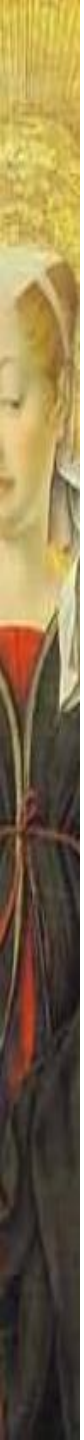
Possible Results





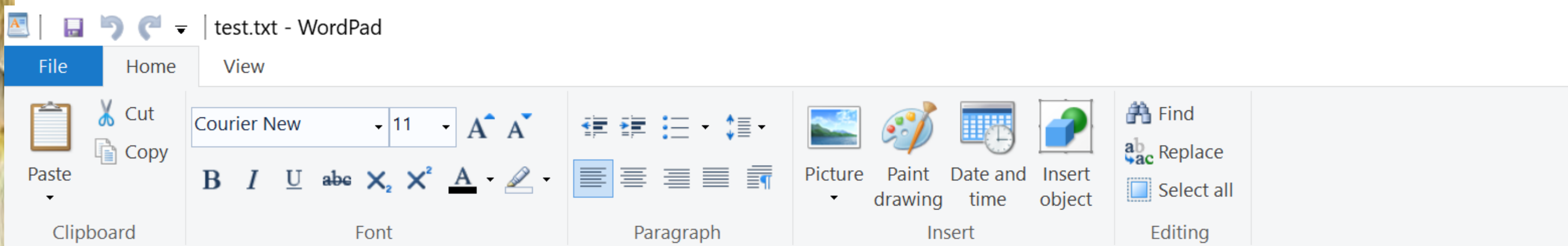
Note:

- Better than random
- Train normally better than test
- Results may depend on various parameters and configurations
  - learning rate, number of epochs, use of bias, etc.
- To get better performance we will use multi-class perceptron (next lecture)



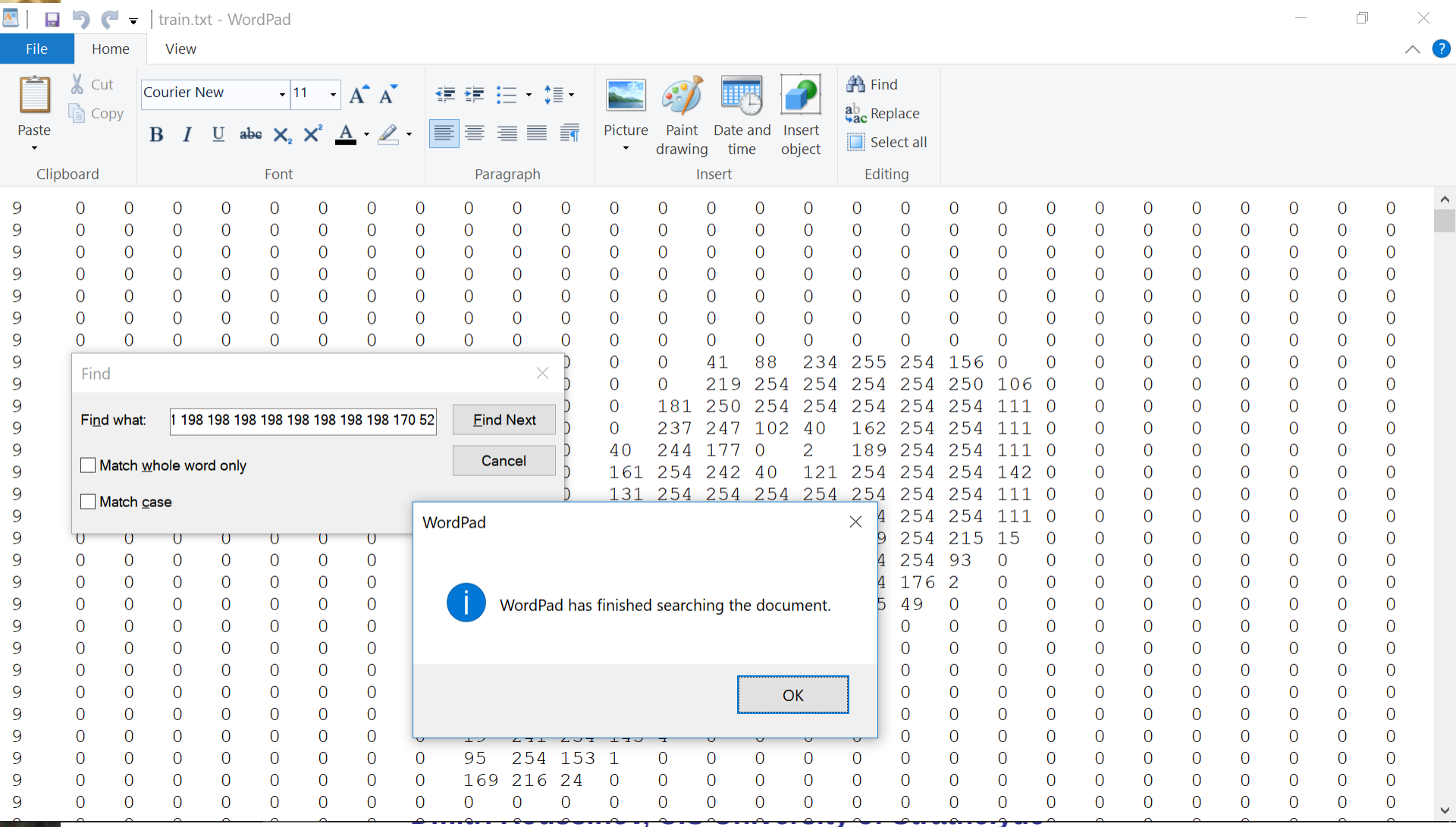
How to verify train/test don’t overlap

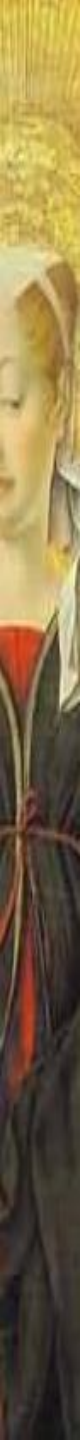
Select some “non trivial” line in test



7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	84	185	159	151	60	36	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	222	254	254	254	254	241	198	198	198	198	198	198	198	198	170	52	0
7	0	0	0	0	0	0	67	114	72	114	163	227	254	225	254	254	254	250	229	254	254	140	0
7	0	0	0	0	0	0	0	0	0	0	0	17	66	14	67	67	67	59	21	236	254	106	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	83	253	209	18	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	22	233	255	83	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	129	254	238	44	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	59	249	254	62	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	133	254	187	5	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	205	248	58	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	126	254	182	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	75	251	240	57	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	19	221	254	166	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	3	203	254	219	35	0	0	0	0	0	0

Search for it in train





What to check

- If there is not such line, we are fine
- Can check a few more randomly selected sequences
- If we find it, verify that the entire image is still not the same