Joanna Sroka 246756

Introduction

The goal of this exercise was to implement a model that would allow classification of clothes photographs. Image classification is one of the most fundamental problems in ML. Given a set of images that are all labeled with a single category (digits 0...9), model should predict these categories for a novel set of test images and measure the accuracy of the predictions. There are a variety of challenges associated with this task, e.g. different points of view, image deformation, etc.

There are many different techniques and models to solve the problem of image classification. In this project, I will discuss different classification models and compare them. I will begin with simple classic machine learning algorithms: K-nearest neighbors. For a more advanced model, I will implement a Convolutional Neural Network.
It will show that KNN achieves worse classification accuracy than CNN. The implemented approaches are evaluated by the accuracy of the predictions on test photos.

Methods

K Nearest Neighbours (KNN)
KNN classifier can work directly on images without feature extraction. Therefore, I didn't use extract features in any way. This technique can be described as discriminative modeling. It's purpose is to model conditional probability distribution. K-Nearest Neighbors is a non-parametric classification algorithm. The basic idea behind it is simple. Given a image to classify, find k images in the train set that are "closest", that is the most similar to the test image. Assign the most frequent among k labels corresponding to neighbours to the test vector or image. I chose the same parameter k (k = 5) that was used in the Benchmark. Above this number I didn't get a better result.

To calculate the distance between two objects is used some closeness metric.
I used the optimized Euclidean distance method, also called L2 norm, without any loops:

```
def euclidean_distance(X, X_train): #obliczanie odleglosci euklidesowej
    return -2 * np.dot(X, X_train.T) + np.sum(X_train**2, axis=1) + np.sum(X**2, axis=1)[:, np.newaxis]
```

In order to compute the distance matrix efficiently I needed to vectorize the operation. By vectorizing I mean expressing the operations done on each pair of elements of matrices as an operation done on whole matrices.

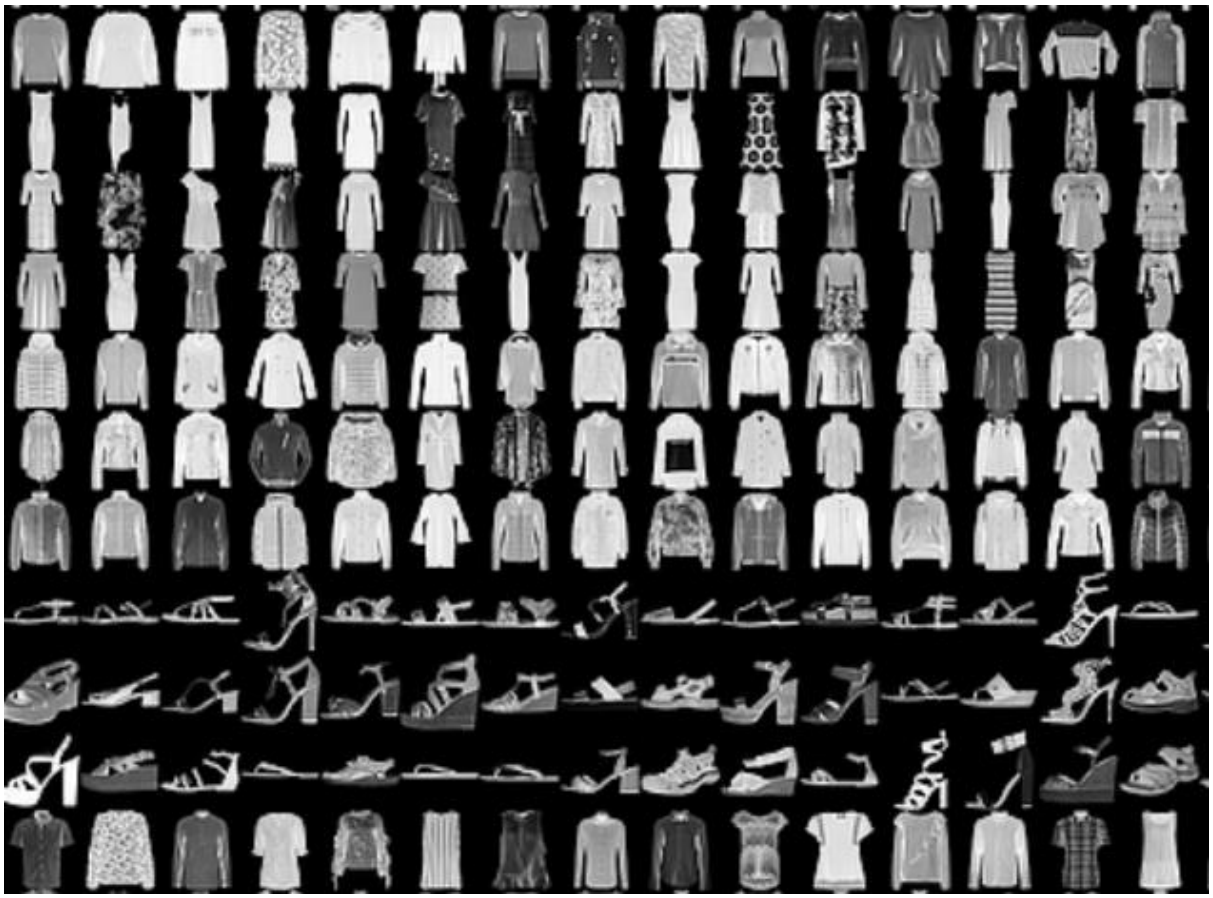
```
Test accuracy: 0.8577
```

The final average accuracy of the model is **85.77%.** According to the Benchmarks result, the same classifier can achieve the result of **86.0%** accuracy. This small difference may have been caused by differences in implementation, e.g. implementation of different closeness metric or which class the algorithm has chosen if both had had the same probability.

Fashion Mnist

This is a dataset of Zalando’s article images.

Sample pictures (before adding distortion):



It consists of a training set (60,000 examples) and a test set (10,000 examples). Each image is a 28 x 28 array with values ranging from 0 to 255. Each example is associated with a label indicating one of 10 classes:

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Each example of training and test is assigned to one of the above labels.

Import Fashion Mnist dataset with Tensorflow/Keras:
Fashion-MNIST dataset is downloaded from Keras dataset.

```
fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

Convolutional Neural Networks (CNN):

I've created a simple CNN. I chose this classificator because it's the most popular neural network model used for image classification task. I built my model using the Keras framework.

The final average accuracy of the model is **92,32%.**

I trained the model in 50 epochs, compiled it with categorical_crossentropy loss function and Adam optimizer:

```
model.compile(loss=keras.losses.sparse_categorical_crossentropy,
              optimizer='adam',
              metrics=['accuracy'])
```

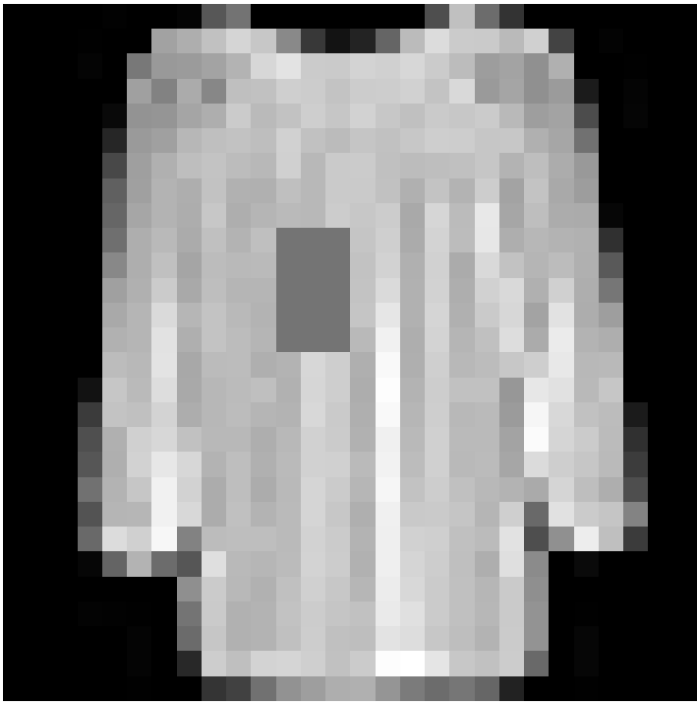
```
model.fit(train_images, train_labels, epochs=50)
```

Cutout/Random erasing
I also used Cutout/Random erasing implementation written by **yu4u** (https://github.com/yu4u/cutout-random-erasing). This is a kind of image preprocessing methods for convolutional neural networks. It tries to regularize model by using training images with randomly changed or removed parts. I've decided that 15% of training photos will be modified using this method. Function eraser() changes single input image.
This method contains several parameters that can be customized, e.g. the probability of changing images or the proportions of the changed image relative to the initial image:

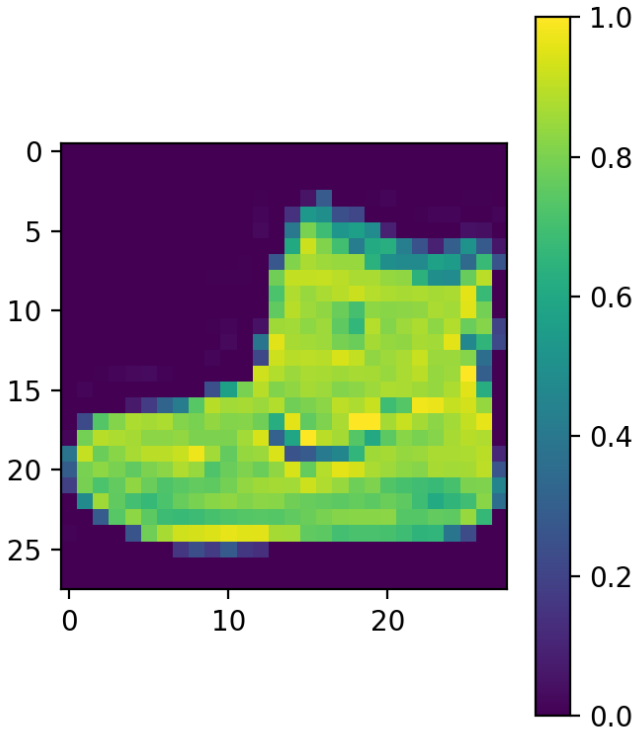
```
#cutout/random erasing
def get_random_eraser(p=0.5, s_l=0.02, s_h=0.4, r_1=0.3, r_2=1/0.3, v_l=0, v_h=255, pixel_level=False):
    def eraser(input_img):
```

- `p` : the probability that random erasing is performed
- `s_l`, `s_h` : minimum / maximum proportion of erased area against input image
- `r_1`, `r_2` : minimum / maximum aspect ratio of erased area
- `v_l`, `v_h` : minimum / maximum value for erased area
- `pixel_level` : pixel-level randomization for erased area

Sample photo after using the eraser() function:



This is one of the images from the training dataset:



The data should be preprocessed before training the network. I scaled pixel color values to range from 0 to 1:

```
train_images = train_images / 255.0

test_images = test_images / 255.0
```

I also reshaped input images, as keras.layers.Conv2D requires that input image is 3 dimensional and those dimensions are (rows, columns, channels) for channels_last data_format. My images have only one color channel:

```
train_images = train_images.reshape((-1,28,28,1))
test_images = test_images.reshape((-1,28,28,1))
```

I used the Sequential model to create a simple CNN consisting of repeating structure: convolution layer, followed by a pooling layer then a dropout layer.

I defined the input data shape in the first layer. The last layer is a dense layer with softmax activation which classifies the data as one of 10 labels.

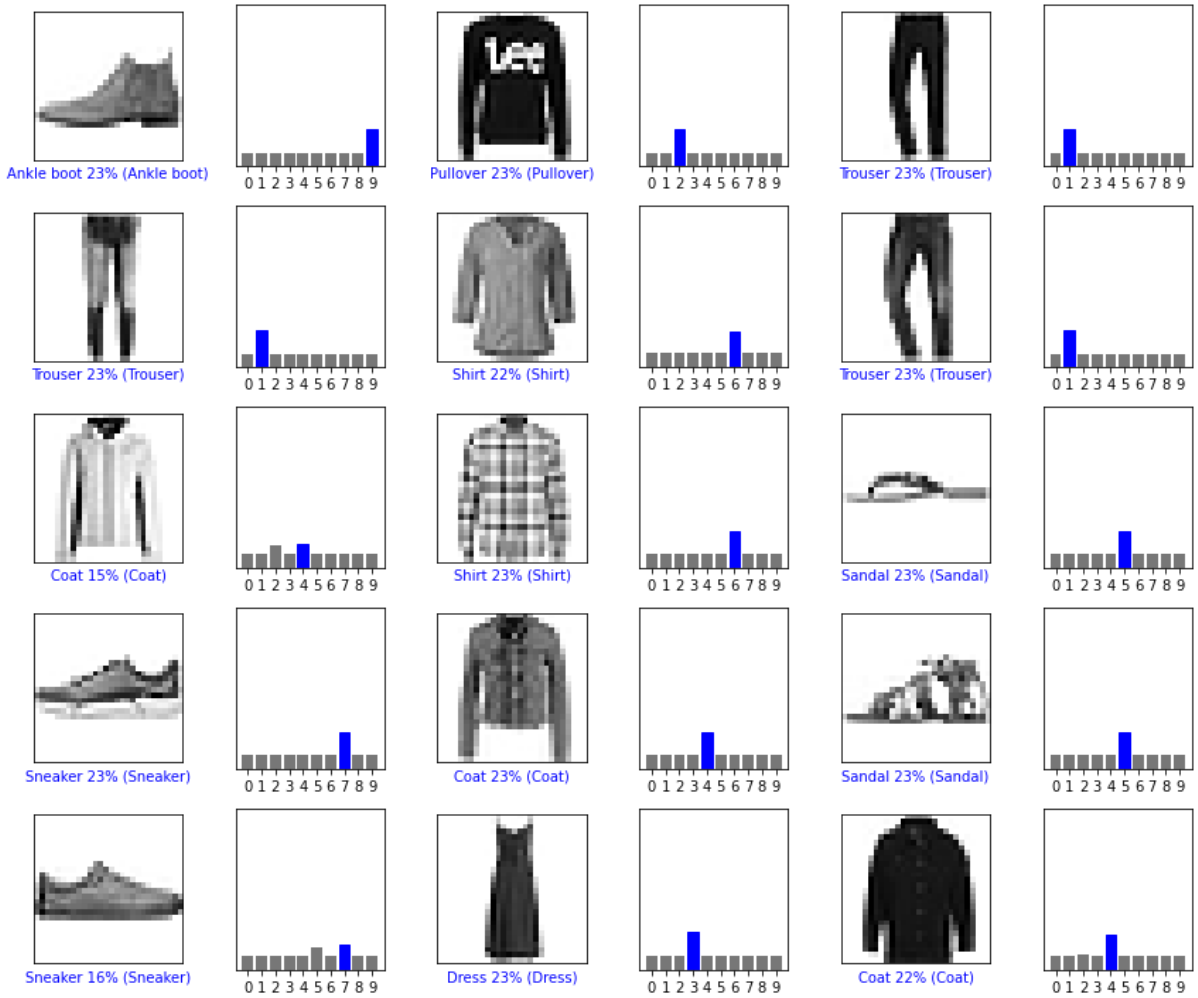
```
model = keras.Sequential()
# Conv + Maxpooling
model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=(4,4), padding='same',
 activation='relu', input_shape=(28,28, 1)))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2,2)))
# Dropout
model.add(tf.keras.layers.Dropout(0.1))
# Conv + Maxpooling
model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=(4,4), padding='same', activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2,2)))
# Dropout
model.add(tf.keras.layers.Dropout(0.3))
# Flatting 3D feature to 1D feature vector
model.add(tf.keras.layers.Flatten(input_shape=(28,28, 1)))
# Fully connected Layer
model.add(tf.keras.layers.Dense(256, activation='relu'))
# Dropout
model.add(tf.keras.layers.Dropout(0.5))
# Fully Connected Layer
model.add(tf.keras.layers.Dense(10, activation='softmax'))
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 28, 28, 64)        1088
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
dropout_6 (Dropout)          (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 14, 14, 32)        32800
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 7, 7, 32)          0
_________________________________________________________________
dropout_7 (Dropout)          (None, 7, 7, 32)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 1568)              0
_________________________________________________________________
dense_4 (Dense)              (None, 256)               401664
_________________________________________________________________
dropout_8 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                2570
=================================================================
Total params: 438,122
Trainable params: 438,122
Non-trainable params: 0
```

```
Test accuracy: 0.9232000112533569
```

The final average accuracy of the model is **92.32%**. In the Benchmark this classifier achieved accuracy of **91,6%**. This small difference may have been caused by some differences in implementation, e.g. thanks to Cutout/Random erasing.

Correct prediction labels are blue and incorrect prediction labels are red. The number gives the percentage (out of 100) for the predicted label:



Sources:

https://www.youtube.com/watch?v=RJudqel8DVA&feature=youtu.be

https://www.youtube.com/watch?v=FiNgII1wRNk

https://medium.com/@souravdey/l2-distance-matrix-vectorization-trick-26aa3247ac6c

https://medium.com/datadriveninvestor/k-nearest-neighbor-classification-with-python-numpy-tensorflow-on-fashion-mnist-dataset-d8361187c09c

https://www.machinecurve.com/index.php/2019/10/06/how-to-use-sparse-categorical-crossentropy-in-keras/

https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-fashion-mnist-clothing-classification/

https://www.tensorflow.org/tutorials/keras/save_and_load

Tensorflow Tutorials which I used to write the model:

https://www.pyimagesearch.com/2019/02/11/fashion-mnist-with-keras-and-deep-learning/

https://www.youtube.com/watch?v=yFrc2SCdW7w&list=PLMz1vLpcJgGCPONANPZymB1l6W9PzLzy5

https://www.youtube.com/watch?v=lOZGYzTn9Z8

https://www.tensorflow.org/tutorials/keras/classification

https://blog.tensorflow.org/2018/04/fashion-mnist-with-tfkeras.html

https://github.com/cmasch/zalando-fashion-mnist/blob/master/Simple_Convolutional_Neural_Network_Fashion-MNIST.ipynb

Cutout/Random erasing:

https://github.com/yu4u/cutout-random-erasing

Results

Classifier	My implementation	Benchmark
KNeighborsClassifier	0.858	0.860
2 Conv+pooling	0.923	0.916

Usage

Import Fashion Mnist dataset with Tensorflow/Keras:
Fashion-MNIST dataset is downloaded from Keras dataset.

```
fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

I installed the library: Tensorflow. I used these libraries:

```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
import numpy as np
import os

from tensorflow.keras.models import save_model, load_model
```

I also installed pyyaml h5py to save/load model.

I saved my model in HDF5 standard:

```
model.save('my_model.h5')
```

You can reload saved model:
new_model = tf.keras.models.load_model('my_model.h5')

Or just run the program. Each method is in a separate, single file.
I tested both programs directly in Google colab (TensorFlow with GPU). Thanks to high performance of machine computing the model it didn't take very long. I can highly recommend it.