Exercise 3 in Computer Vision: Deep Learning
Discussion on 11/11/2019 10.00 am – 11.00 am (NB 3/57)

In preparation of this exercise please make sure that the OpenCV image processing library is installed. You can do this via

```
pip install opencv-python
```

We will work with the MNIST dataset that is covered in more detail in one of the future lectures. In short, it contains binary images of handwritten digits, thus, has classes from 0 to 9. The attached library `DLCVDataset` provides an interface for loading and preprocessing MNIST. The dataset will be downloaded from the internet on first execution. Do not work with the entire dataset until you consider your code bug free.

Extract Histogram-of-Oriented-Gradients features from every image. You can use the module `DLCVDatasets` from earlier exercises in order to do that. Train a linear classifier to classify the digit images. An example for a gradient descent variant for doing this has been uploaded to MOODLE for the previous exercise. Try different parameter combinations (cell size, block size, histogram bins) automatically and evaluate which choice of parameters works best. What is the error rate when using the best parameter combination?

Let's examine how well the HOG feature extraction separates the different digit classes. Perform a k-means clustering on the HOG vectors corresponding to the best choice of feature extraction parameters. Use $k = 10$ for this (or adapt k to the number of digit classes that you work with if you do not use all of them). You end up with k cluster centers. Show the images corresponding to some of the HOG vectors that have minimum Euclidean distance to the cluster centers. Does the HOG feature extraction provide a useful clustering of the MNIST dataset?

Tips:

Tensorflow has different APIs to let you solve this exercise. The **historical API** was covered in the first exercise. The interface is available under `tensorflow.compat.v1`.

For reasons that will become clearer in the course of the lectures you should use `tf.nn.sparse_softmax_cross_entropy_with_logits` as your loss function. The function receives two parameters: `logits` will get the output of the neural network and internally normalize it, `labels` just receives the placeholder that holds the class indices for each training example (i. e. `y_train` or `y_test` from `DLCVDataset.get_dataset`)

`tf.matmul` — creates a matrix multiplication operation in tensorflow
`tf.argmax` — creates an operation that yields the index of the maximum element along a given axis
`tf.cast` — creates an operation that changes the data type of a tensor
`tf.reduce_sum` — creates an operation that sums the elements of a tensor along an axis and removes said axis from the result
`tf.reduce_mean` — creates an operation that computes the average of the elements of a tensor along an axis and removes said axis from the result

Alternatively, you can use the beginner-friendlier **Keras Functional API**. Within this API a linear classifier is known as a *dense layer*. See

`https://www.tensorflow.org/guide/keras/functional`

for a gentle introduction. You can use `keras.optimizers.SGD` as an optimizer.

The following code snippet defines a function to extract HOG features from an image:

```
def compute_hog(cell_size, block_size, nbins, imgs_gray):
    """
    Wrapper for the OpenCV interface for HOG features.

    :param cell_size  number of pixels in a square cell in x and y direction (e.g. (4,4), (8,8))
    :param block_size number of cells in a block in x and y direction (e.g., (1,1), (1,2))
    :param nbins      number of bins in a orientation histogram in x and y direction (e.g. 6, 9, 12)
    :param imgs_gray  images with which to perform HOG feature extraction (dimensions (nr, width, he
    :return array of shape H x imgs_gray.shape[0] where H is the size of the resulting HOG feature v
```

```
        (depends on parameters)
    """
    hog = cv2.HOGDescriptor(_winSize=(imgs_gray.shape[2] // cell_size[1] * cell_size[1],
                                      imgs_gray.shape[1] // cell_size[0] * cell_size[0]),
                            _blockSize=(block_size[1] * cell_size[1],
                                        block_size[0] * cell_size[0]),
                            _blockStride=(cell_size[1], cell_size[0]),
                            _cellSize=(cell_size[1], cell_size[0]),
                            _nbins=nbins)
    # winSize is the size of the image cropped to a multiple of the cell size

    hog_example = hog.compute(np.squeeze(imgs_gray[0, :, :]).astype(np.uint8)).flatten().astype(np.f

    hog_feats = np.zeros([imgs_gray.shape[0], hog_example.shape[0]])

    for img_idx in range(imgs_gray.shape[0]):
        hog_image = hog.compute(np.squeeze(imgs_gray[img_idx, :, :]).astype(np.uint8)).flatten().ast
        hog_feats[img_idx, :] = hog_image

    return hog_feats
```