Static gestures classification using Convolutional Neural Networks on the example of the Russian Sign Language (RSL) dactyl

Oleg Potkin
November 19, 2017

I. Definition

Introduction

Interfaces of human-computer interaction are diverse in their implementation and scope e.g. systems with console input-output, controllers with gesture control, brain-computer interfaces and others. Systems using data input based on the recognition of custom gestures have gained wide popularity since 2010 after the release of the contactless game controller Kinect from Microsoft. Gesture based controllers increase their market share and become a part of everyday life of different categories of users. So, for example, the Volkswagen car manufacturer introduced the multimedia system Golf R Touch Gesture Control to control the multimedia system of the car by gesture commands.

To translate gesture commands into a control signal, a gesture classification mechanism is needed, which can be obtained from various devices: special gloves defining joint coordinates, as well as 2D and 3D video cameras. The approach using gloves has a significant drawback – a user needs to wear a special device connected to the computer. In turn, the approach based on the concept of computer vision using video cameras is considered more natural and less expensive.

Problem Statement

Project demonstrates the core of Russian Sign Language static gesture classification system, which is based on the approach of computer vision with using convolutional neural network. The work is actual and represents a starting point for researchers in the field of gesture recognition.

Metrics

Model building in artificial neural networks refers to selecting the “optimal” network architecture, network topology, data representation, training algorithm, training parameters, and terminating criteria, such that some desired level of performance is achieved. Proper evaluation metrics, a critical aspect of any model construction, is based upon some specified NN performance measure of data that

was not used in model construction. In addition to trained NN, this performance measure is often used to evaluate the superiority of network architecture, learning algorithm, or application of a neural network.

In this project performed 3 levels of evaluation:

- On Training step. Evaluating parameters – **accuracy**, **loss**.
- On Validation step. Performs every epoch of CNN training process. Evaluating parameters – **accuracy**, **loss**.
- On Test step. Performs after the training of CNN on the data that not presented during the training process. Evaluating parameter – **accuracy**.

Accuracy is the number of correct predictions made as a ratio of all predictions made. Accuracy is the most common evaluation metric for classification problems, it is also the most misused. It is really suitable when there are an equal number of observations in each class (which is the case, see Image 2) and that all predictions and prediction errors are equally important.

Loss function is a 'categorical_crossentropy'. This function is not naturally represented as a product of the true label and the predicted value, but is convex and can be minimized using stochastic gradient descent methods. The categorical cross entropy loss is ubiquitous in modern deep neural networks.

II. Analysis

Dataset

To solve described problem with CNN approach I need a dataset. I didn't found any open dataset in this field and decided to create my own.

The dataset for learning, validating and testing a neural network consists of **around 1000 images** (1042 on the moment of writing of this paper) with a resolution of **128x128 pixels** (more parameters in table 1).

Parameter	**Value**
Format	.PNG
Width (px)	128
Height (px)	128
Color Space	RGB
Глубина	3

Table 1. Image parameters

The image is divided into **10 classes**, each of which corresponds to a strictly defined gesture (image 1):

Line 1 – Class Id.

Line 2 – Value / Letter.

Line 3 – Quantity in dataset.

Line 4 – image with gesture example.

Data samples from each class are available in the project repo.

All the category information (labels) converted to one-hot encoding that is used by the final classification. For instance, class=5 converted to a vector as [0 0 0 0 0 1 0 0 0 0] (labels start from 0 to 9). I use this vector format in the classification to calculate training error at every step of the training.

Image 1. Examples from RSL dataset (author – Oleg Potkin)

Label distribution shown on Image 2.

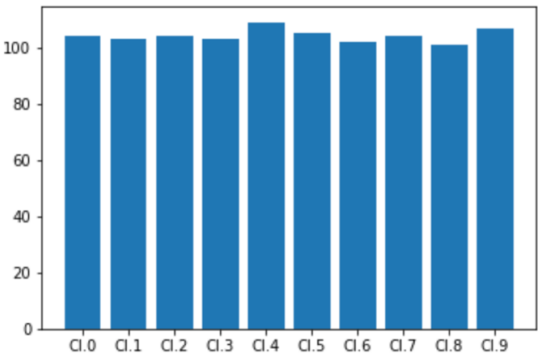

Image 2. Gesture labels distribution

Looking to the Gesture labels distribution (Image 2) we can make a conclusion that accuracy metric will be suitable in this case because there are an equal number of observations in each class.

CNN Architecture

In this part, I'm going to describe CNN architecture on abstract level. As a base for my experiment I'm going to take LeNet-5 architecture. The main disadvantage of LeNet-5 is overfitting in some cases and no built-in mechanism to avoid this. So, I'm going to improve that and add Dropout Layers. Improved LeNet-5 presented on Image 2.

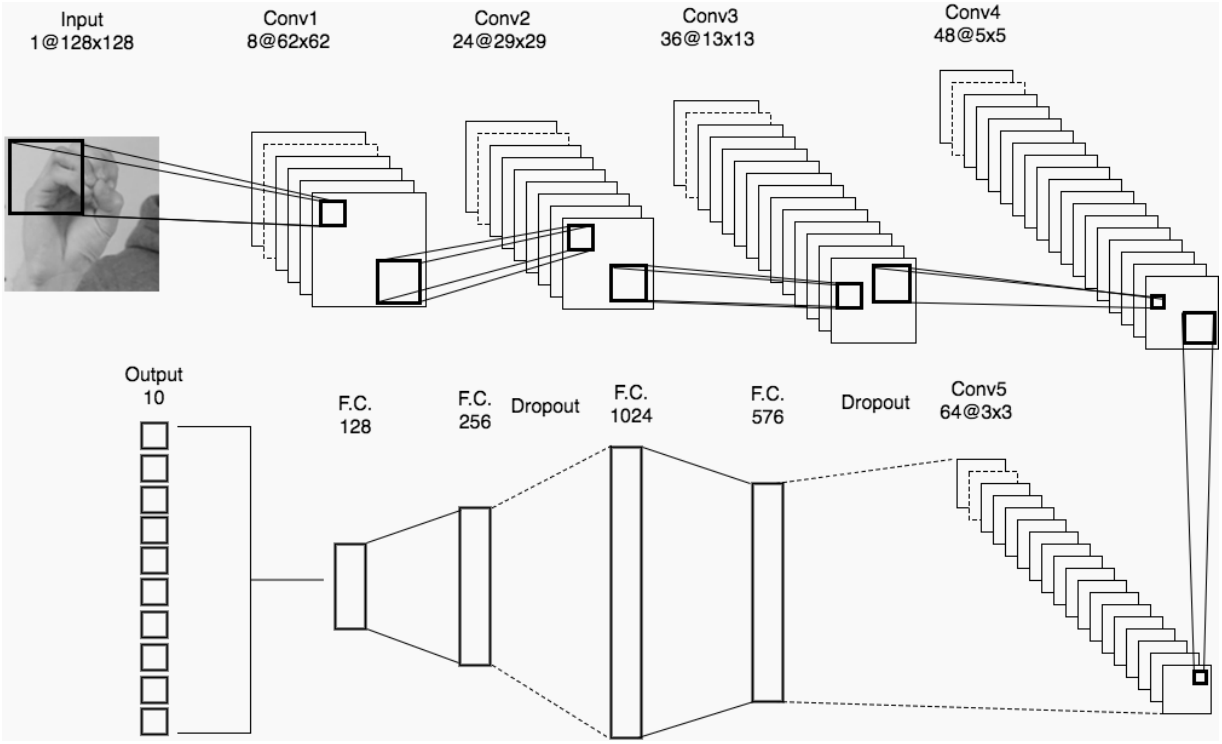

Image 3. CNN for RSL classification.

III. Methodology

Data Preprocessing

Before sending data to the classifier, it is necessary to perform image transformations that will help to get rid of minor characteristics for the classifier, which in turn will increase productivity when classifying the classifier. This process is called preprocessing.

Data preprocessing includes 2 steps:

- Convert RGB to YCrCb color space
- Threshold color components to perform hand segmentation

In the work presented, the RGB color space was transformed into YCbCr for skin segmentation and binarization based on threshold values was applied (Image 3).

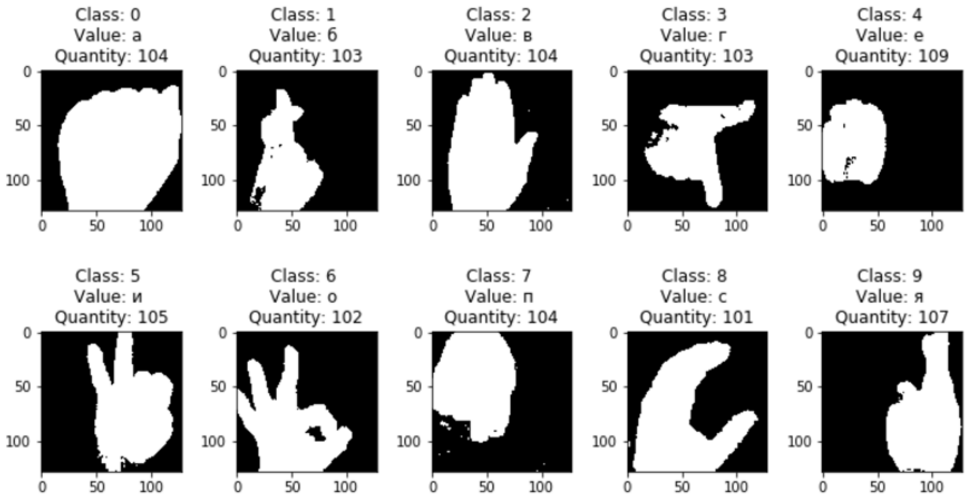

Image 4. Samples of images after preliminary processing. Parameters: the number of the class, the correspondence of the gesture to the letter of the Russian alphabet, the number of images of the class.

The final stage of preliminary processing is normalization. The pixel values are in the range from 1 to 255, but for correct operation of the neural network it is necessary to input values from 0 to 1.

In order to minimize the possibility of retraining classifier, the data set is divided into 3 parts:

- **Training set** contains 80% of the data set (666 images).
- **Test set**, contains 20% of the data set (209 images). Using for evaluation of the model. These samples never used during training and cross-validation processes.
- **Cross-validation set**, contains 20% of the training sample (167 images).

Benchmark

I'm using LeNet-5 CNN as a benchmark model for this project. Parameters of this model presented on Scheme 1.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lambda_1 (Lambda)            (None, 128, 128, 1)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 126, 126, 6)       60
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 42, 42, 6)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 40, 40, 16)        880
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 13, 16)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2704)              0
_________________________________________________________________
dense_1 (Dense)              (None, 120)               324600
_________________________________________________________________
dense_2 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_3 (Dense)              (None, 10)                850
=================================================================
Total params: 336,554.0
Trainable params: 336,554.0
Non-trainable params: 0.0
```

Scheme 1. Benchmark model architecture.

I performed training and test of benchmark model and got follow results:

- Test loss: 0.513365882435
- Test accuracy: 0.8708133977

Model going to overfit, because during the training process it showing very good results: loss 0.0681 and accuracy 0.9985.

Implementation

For this project I'll use Python 3.x, NumPy, Sci-Kit, OpenCV, Tensorflow and Keras.

Better Known as CNN (Conv Nets) are one of the premier, state of art, Artificial Neural Network design architecture, which helps in Image Based Classifications. The Basic Principle behind the working of CNN is the idea of Convolution, which produces filtered Feature Maps stacked over each other.

In order to recognize parts of an image, we would ideally like to segment that image into meaningful parts. If we consider an image as a matrix, each subset of that image might represent an object (a pattern). Let's assume that every neuron in the input layer takes a separate pixel.

There would be two approaches to this - bottom-up (combining pixels into meaningful patterns) and top-down (splitting the whole image into meaningful parts).

How does the "not fully connected" part work here?

Simple, since if we connect every neuron in the input layer to every neuron in the next layer, we would consider the whole image in each following neuron. Therefore, this would be the top-down approach - you take everything and eventually null some connections to isolate certain parts.

While, if you would connect a subset of input pixels to the next layer (for example, a subset could be a 3x3 pixel square), you take the bottom-up approach, connecting parts of images into meaningful patterns, resulting in a model that generalizes better.

The neural network for my project consists of the following layers (Scheme 2)

Layer (type)	Output Shape	Param #	Connected to
lambda_1 (Lambda)	(None, 128, 128, 1)	0	lambda_input_1[0][0]
convolution2d_1 (Convolution2D)	(None, 62, 62, 8)	208	lambda_1[0][0]
convolution2d_2 (Convolution2D)	(None, 29, 29, 24)	4824	convolution2d_1[0][0]
convolution2d_3 (Convolution2D)	(None, 13, 13, 36)	21636	convolution2d_2[0][0]
convolution2d_4 (Convolution2D)	(None, 5, 5, 48)	43248	convolution2d_3[0][0]
convolution2d_5 (Convolution2D)	(None, 3, 3, 64)	27712	convolution2d_4[0][0]
dropout_1 (Dropout)	(None, 3, 3, 64)	0	convolution2d_5[0][0]
flatten_1 (Flatten)	(None, 576)	0	dropout_1[0][0]
dense_1 (Dense)	(None, 1024)	590848	flatten_1[0][0]
dropout_2 (Dropout)	(None, 1024)	0	dense_1[0][0]
dense_2 (Dense)	(None, 256)	262400	dropout_2[0][0]
dense_3 (Dense)	(None, 128)	32896	dense_2[0][0]
dense_4 (Dense)	(None, 10)	1290	dense_3[0][0]

Total params: 985,062
Trainable params: 985,062
Non-trainable params: 0

Scheme 2: CNN layers and parameters

- Input layer. The size of the layer corresponds to the size of the image after preliminary processing 1x128x128 elements.
- Convolution layer (Conv1). Output size 8x62x62. Convolution window size 5x5. Activation function - ReLU (Rectified Linear Unit or rectifier). ReLU has the

following formula f (x) = max (0, x) and realizes a simple threshold transition at zero. Compared to the sigmoid activation function, ReLU increases learning speed and classifier performance.

- Convolution layer (Conv2). Output size 24x29x29. Convolution window size 5x5. Activation function - ReLU.
- Convolution layer (Conv3). Output size 36x13x13. Convolution window size 5x5. Activation function - ReLU.
- Convolution layer (Conv4). Output size 48x5x5. Convolution window size 5x5. Activation function - ReLU.
- Convolution layer (Conv5). The output size is 64x3x3. The size of the convolution window is 3x3. The activation function is ReLU.
- Dropout 1 layer, p = 0.25.
- Fully-connected layer 1. Size 576.
- Fully- connected layer 2. The size is 1024.
- Dropout 2 layer, p = 0.25.
- Fully- connected layer 3. Size 256.
- Fully- connected layer 4. Size 128.
- Output layer. Size 10. Activation function - softmax.

Convolution, in general, is an operation of filtering the input data. In image recognition context, 2D filters are used to process data at different layer to generate the features. These filters are pre-defined and their coefficients are computed during training.

Training is done through **backpropagation algorithm** with **gradient descent**.

Each neuron at the output of the Convolution layer will be connected to all other neurons after weighted properly, Similar to Convolution layer, weight of these taps in **Fully Connected layer** is found though backpropagation algorithm.

Dropout layer. The layer is constructed so that each neuron can fall out of this layer with probability p, therefore, other neurons remain in the layer with probability q =1-p. The dropped neurons are not included in the classifier training, that is, at each new epoch the neural network is partially changed. This approach effectively solves the problem of retraining classifier.

Refinement

I’ve started from LeNet-5 architecture and performed follow steps to improve result:

- Adjusted hyperparameters (described below)

- Applied dropout Technique
- Add image preprocessing to reduce number of unimportant details (e.g. color information from RGB space)

After tuning and improvements I stopped with follow hyperparameters on Neural Network:

- Learning rate = 0.001;
- batch size = 128;
- number of epochs = 20;
- loss = categorical_crossentropy (see Metrics);
- metrics = accuracy (see Metrics)

IV. Results

Training results

The validation set helped determine if the model was over or under fitting (Image 4)

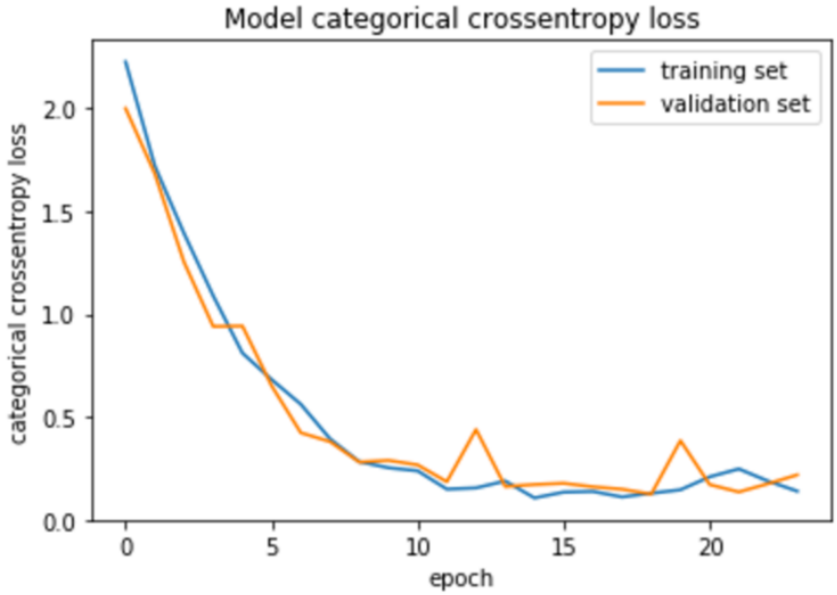

Image 5. Training process parameters.

And, finally, test set results:

- Test loss: 0.238594918445
- Test accuracy: 0.913875598657

Results are significantly better than it was on benchmark model (Test loss in case of LeNet-5 was 0.513365882435 and Test accuracy was 0.8708133977)

Experiment

An aim of experiment to verify that the model generalizes well to unseen data. In addition to the use of a test sample, an experimental sample (images of hands made under conditions very different from those in which the main data set was recorded (Image 6)) was developed and used.

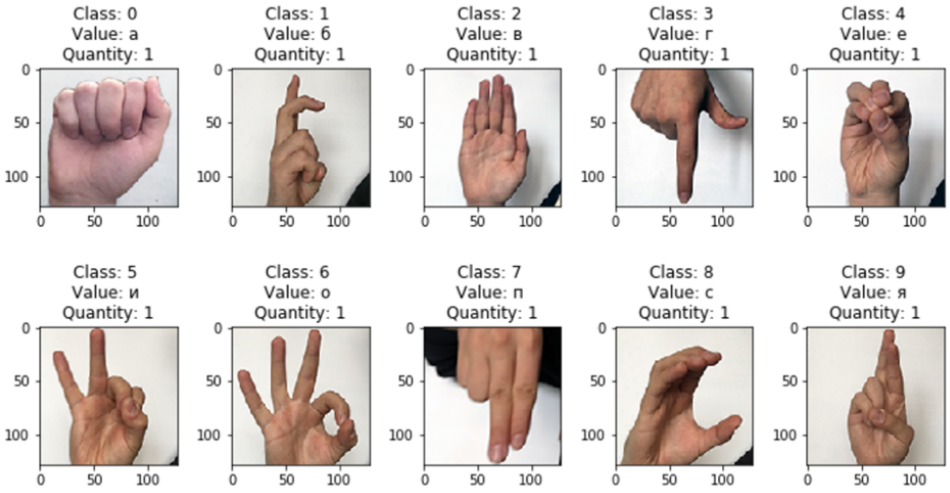

Image 6. Experimental images

The probability distribution for 10 experimental examples:

```
[[ 1.     0.     0.     0.     0.     0.     0.     0.     0.     0.   ]
 [ 0.     0.998  0.     0.     0.     0.     0.     0.     0.     0.001]
 [ 0.     0.002  0.998  0.     0.     0.     0.     0.     0.     0.   ]
 [ 0.     0.     0.     0.     0.     0.     0.     1.     0.     0.   ]
 [ 0.     0.     0.195  0.     0.021  0.     0.     0.783  0.     0.   ]
 [ 0.     0.     0.     0.     0.     0.94   0.06   0.     0.     0.   ]
 [ 0.     0.     0.     0.     0.     0.     1.     0.     0.     0.   ]
 [ 0.     0.     0.     0.     0.     0.     0.     1.     0.     0.   ]
 [ 0.     0.     0.     0.     0.     0.     0.     0.     1.     0.   ]
 [ 0.     0.003  0.     0.     0.     0.     0.     0.     0.     0.997]]
```

- Class 0 (А): True, Probability = 1
- Class 1 (Б): True, Probability = 0.998
- Class 2 (В): True, Probability = 0.998
- Class 3 (Г): False, Probability = 0 (Detected Class 7 with Probability = 1). The detection is wrong because this gesture looks similar with Class 7.
- Class 4 (Е): False, Probability = 0.021 (Detected Class 7 with Probability = 0.783). Model made a mistake because it probably found similarities in Class 5 and Class 7.
- Class 5 (И): True, Probability = 0.94
- Class 6 (О): True, Probability = 1

- Class 7 (П): True, Probability = 1
- Class 8 (С): True, Probability = 1
- Class 9 (Я): True, Probability = 0.997

V. Conclusion

As a result of the research, a set of data was developed, consisting of more than a thousand elements belonging to ten classes - dactylem of Russian Gesture Language. Algorithms and procedures for preliminary data processing in Python 3.5 are developed.

A classifier based on a convolutional neural network using the Keras and TensorFlow framework was developed and developed, demonstrating the accuracy of classification in 91.4% on the test data set and 80% on the experimental data set.

The presented accuracy results are not enough for the introduction of this system into production, but it can become a basis for further research in this direction.

Difficulties during the project implementation.

First and most difficult thing is overfitting. To avoid this, I used different techniques:

- Used dropout layers
- Adjusted hyper parameters
- Data preprocessing

Another difficult thing is a data collection. Of course, the project like this required much more data and augmentation could help at this point. But it will be useful to attract some crowdsourcing resources.

In some case there were misclassifications: Class 3 and Class 4. These classes require some additional training examples because the model has difficulties with finding of differences in these classes now.

Future improvement

The main points of final results improvement:

- Use much more data (to think about crowdsourcing data collection)
- Use data augmentation techniques
- Use advanced preprocessing techniques (more experiments with color spaces, hog features etc.)