Advanced Topics in
Artificial Intelligence

Wrocław, 01.2018

Image classification
using python + neural network

Author: Piotr Kopka

Table of contents

1. Introduction
The goal of the project is to build an images classifier. The application uses neural network to classify given image into one of ten available categories: “airplane”, “car”, “bird”, “cat”, “deer”, “dog”, “frog”, “horse”, “ship”, “truck”.

2. Programming language and libraries
Programming language: Python 3
Libraries:
- numpy – operations on matrices
- matlplotlib – plotting information
- tkinter – graphic interface
- PIL – loading images

3. Dataset
The dataset used to train the neural network is CIFAR-10 dataset from University of Toronto. It contains 50000 colour training images of size 32 x 32 pixels divided into 10 classes. Available classes are: “airplane”, “car”, “bird”, “cat”, “deer”, “dog”, “frog”, “horse”, “ship”, “truck”. Originally the dataset is split into 5 batches. Each batch contains dictionary with 2 elements – data and labels. Data element is a numpy array of size 10000 (number of pictures in the batch) x 3072. The first 1024 entries are intensities of red colour represented as numbers from 0 to 255, the next 1024 entries are green and the last 1024 entries are blue. Labels element is a list of numbers from 0 to 9 representing one of 10 categories. Images are stored in random order.

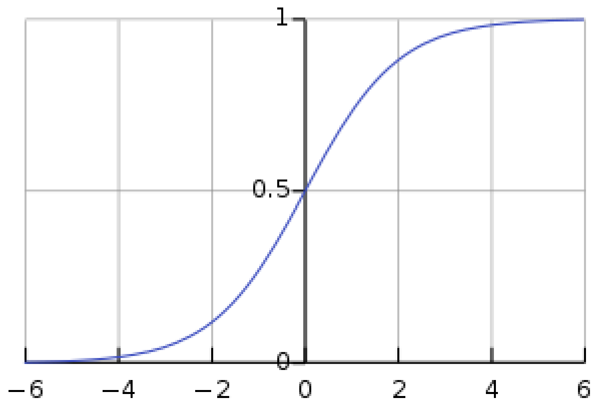
3.1. Processing the dataset
a) Writing the data and labels from dictionary to numpy arrays and normalizing values that represent colours:

```
    def _get_data_from_dict(self, name):
        return np.array(name[b'data'] / 255)

    def _get_labels_from_dict(self, name):
        return np.array(name[b'labels']
```

The normalization is done by dividing values that are in range 0-255 by 255, so the received values are in range 0-1. This is needed because sigmoid(logistic) function is used as activation function in the neural network:

$$f(x) = \frac{1}{1+e^{-x}}$$



“x” here are, in case of hidden layer, features (0 to 255) multiplied by weights (-1 to 1). If “x” values are too big, then the value of function is in almost every case very close to 1, which makes learning process not very effective.

Comparison of first 10 iterations of learning with the same neural network setup:

Training set accuracy [%] / Validation set accuracy [%]	
With normalization	Without normalization
22/21	15/15
25/24	14/14
26/26	14/15
27/26	17/16
28/27	15/16
29/28	14/14
30/29	17/16
31/29	18/18
32/30	16/15
32/30	19/19

b) Concatenating data and labels from batches to single numpy arrays:

```
def get_arrays_from_training_dataset(self):
      data_array = []
      labels_array = []

      for i in range(1,6):
          dataset = self._load_from_file('%s/data_batch_%s' %
                    (self.dataset_path, i))
          data = self._get_data_from_dict(dataset)
          labels = self._get_labels_from_dict(dataset)
          data_array.append(data)
          labels_array.append(labels)

      data_array = np.concatenate(data_array)
      labels_array = np.concatenate(labels_array)

      return data_array, labels_array
```

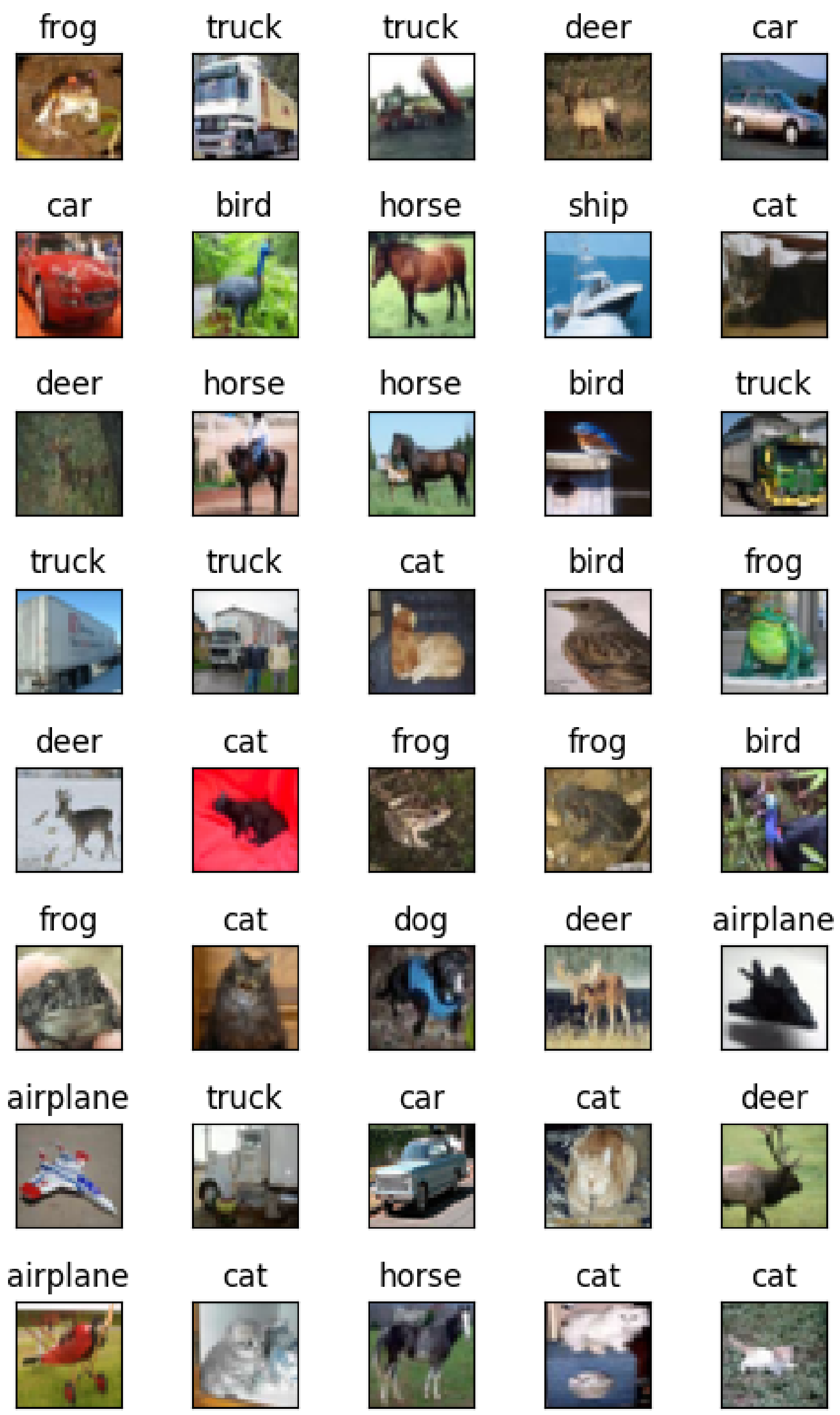
The result are data array of size 50000 x 3072 and labels array of size 50000 x 1.

c) Splitting dataset into training and validation sets:

```
def split_dataset_into_train_valid(self, dataset, training_percent):
    split_number = int(training_percent / 100 * 50000)
    training_dataset = dataset[:split_number]
    validation_dataset = dataset[split_number:]

    return training_dataset, validation_dataset
```

3.2. Sample images from the dataset



4. Input image

Application supports images in jpeg format.

An image should contain object that belongs to one of the following categories: “airplane”, “car”, “bird”, “cat”, “deer”, “dog”, “frog”, “horse”, “ship”, “truck”. Otherwise the result will be random.

4.1. Input image processing

a) Image is opened using PIL library

```
    image = Image.open(name)
```

b) Image is resized to size 32 x 32, then it is written to numpy array. Later the array representing the image is transposed and flattened to convert the image to the same format as training images.

```
    image = image.resize((32, 32))
    image = np.asarray(image)
    image = image.transpose([2, 0, 1])
    image = image.flatten() / 255
```

5. Neural network

5.1. Architecture

The neural network is a feedforward neural network trained using error backpropagation algorithm. It consists of 3 fully connected layers. Input layer has 3072 units, because it is the number of features from images (32 pixels x 32 pixels x 3 colours). Number of hidden units is adjusted during training. Number of output units is 10, because there are 10 categories of images.

5.2. Learning algorithm

a) Encoding labels into one-hot representation:

```
    def _encode_labels_onehot(self, labels):
        onehot_encoded_labels = np.zeros((labels.shape[0], self.output_units))

        for index in range(labels.shape[0]):
            value = labels[index]
            onehot_encoded_labels[index, value] = 1

        return onehot_encoded_labels
```

In the beginning labels are represented as numbers from 0 to 9. One-hot representation means that, instead of 1 number, there is a vector with 10 values, where 9 of them are zeros and 1, which represents the proper category is one. This is needed for neural network, predicting and calculating error wouldn’t be possible without this representation.

b) Initializing weights:

```
    def _initialize_weights(self):
        self.bias_input_hidden = np.zeros(self.hidden_units)
        self.weights_input_hidden = np.random.uniform(-1, 1, (self.input_units,
                                        self.hidden_units))
        self.bias_hidden_output = np.zeros(self.output_units)
        self.weights_hidden_output = np.random.uniform(-1, 1,
                                        (self.hidden_units, self.output_units))
```

There are several approaches to weights initialization. The simplest possible is to initialize all weights with zeros, although it is not recommended because there is no asymmetry between neurons if their weights are the same. They all calculate the same gradient during backpropagation and perform exactly the same parameter update. The simplest approach that works is to initialize weights with small random numbers, close to zero. This way, every neuron is unique in the beginning. Biases can be initialized with zeros because the symmetry is broken with random weights initialization. Here weights are initialized with random numbers between -1 and 1 and biases are initialized with zeros.

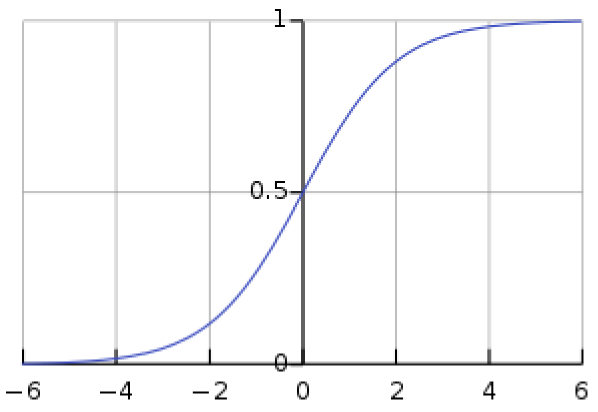
c) Forward propagation:

```
    def _calculate_sigmoid(self, z):
        return 1 / (1 + np.exp(-z))
```

Sigmoid(logistic) function is used as activation function. The role of activation function is to limit the output signal to a finite value, in case of logistic function it is between 0 and 1, which can be treated as probability of occurrence of a feature. The function should be non-linear, because only then neural network can approximate any function. The function should be also differentiable to make learning by backpropagation possible, because it uses derivative of activation function. Logistic function is one of the most popular, because it is non-linear, it gives values between 0 and 1, which can be interpreted as probability and it has simple derivative.

$$f(x) = \frac{1}{1+e^{-x}}$$

$$f'(x) = f(x)(1 - f(x))$$



```
    def _propagate_forward(self, features):
        input_for_hidden_layer = features.dot(self.weights_input_hidden) +
                                 self.bias_input_hidden
        hidden_layer_output = self._calculate_sigmoid(input_for_hidden_layer)

        input_for_output_layer =
                           hidden_layer_output.dot(self.weights_hidden_output) +
                           self.bias_hidden_output
        network_output = self._calculate_sigmoid(input_for_output_layer)

        return hidden_layer_output, network_output
```

First the feature matrix, which is the input layer, is multiplied by weights connecting input layer and hidden layer and biases are added. Then activation of hidden layer is calculated. The same steps are performed between hidden layer and output layer. The result is matrix that has 10 values that are probabilities of every class for each training example.

d) Backpropagation:

```
    def _propagate_back(self, network_output, hidden_layer_output, features,
                        onehot_encoded_labels):
        error_output = network_output - onehot_encoded_labels
        error_hidden = error_output.dot(self.weights_hidden_output.T) *
                       (hidden_layer_output * (1 - hidden_layer_output))

        delta_weights_input_hidden = (features.T).dot(error_hidden)
        delta_bias_input_hidden = (np.ones(features.shape[0])).dot(error_hidden)

        delta_weights_hidden_output = (hidden_layer_output.T).dot(error_output)
        delta_bias_hidden_output =
                       (np.ones(hidden_layer_output.shape[0])).dot(error_output)

        return delta_weights_input_hidden, delta_bias_input_hidden,
               delta_weights_hidden_output, delta_bias_hidden_output
```

Backpropagation is a method to train neural network by calculating gradient with respect to weights. Output of the neural network is compared to real values and then the network is updated by adjusting weights to came closer to real values. Weights are adjusted backwards from output to input.
In the beginning error in output layer is calculated, that is difference between network output and real values. Then it is possible to calculate error in previous layer. Previously calculated error in output layer is multiplied by weights connecting hidden layer with output layer and then by derivative of activation function. Having calculated errors in outputs of hidden and output layer, it is possible to obtain gradient of weights by multiplying output of previous layer by error of the output of the given layer. This gradient tells whether error is increasing or decreasing with increasing or decreasing given weight. Later, weights are changed according to that gradient.

e) Regularization:

```
def _regularize(self, delta_weights_input_hidden,
                delta_weights_hidden_output):
    delta_weights_input_hidden = delta_weights_input_hidden +
                   self.regularization_parameter * self.weights_input_hidden
    delta_weights_hidden_output = delta_weights_hidden_output +
                  self.regularization_parameter * self.weights_hidden_output

    return delta_weights_input_hidden, delta_weights_hidden_output
```

The purpose of regularization is to prevent from overfitting. Overfitting means that the neural networks works good with training data, but has poor accuracy with data, that it hasn't seen before. Here, to deltas calculated in previous step there is added regularization term, which is a coefficient multiplied by current weights. It's like adding a penalty term, so it is possible to control the model complexity.

f) Updating weights:

```
def _update_weights(self, delta_weights_input_hidden,
                    delta_bias_input_hidden, delta_weights_hidden_output,
                    delta_bias_hidden_output):
    self.weights_input_hidden -= self.learning_rate *
                                 delta_weights_input_hidden
    self.bias_input_hidden -= self.learning_rate * delta_bias_input_hidden

    self.weights_hidden_output -= self.learning_rate *
                                  delta_weights_hidden_output
    self.bias_hidden_output -= self.learning_rate * delta_bias_hidden_output
```

Updating weights is done by subtracting weight's gradient multiplied by learning rate from current weight. Learning rate is a coefficient that influences speed of learning. If this speed is too big, then training is less accurate, if it is too small, then learning can take a very long time.

5.3. Predicting

```
def predict_labels(self, features):
    network_output = self._propagate_forward(features)[1]
    predicted_labels = np.argmax(network_output, axis = 1)

    return predicted_labels
```

The image is fed into neural network and forward propagation step is performed. The output is a vector of 10 values, where each value is a probability of each class. The class with highest probability is returned.

6. Training the neural network

Parameters that are possible to manipulate in order to make learning process more efficient are:

a) Number of neurons in hidden layer – if it is too small, then the network can experience underfitting, if it is too big, then it may have the opposite problem – overfitting. Overfitting means that the neural networks works good with training data, but has poor accuracy with data, that it hasn't seen before. Underfitting means that the network can't fit training examples.

b) Learning rate – it determines how fast the network is learning. If it is too big, then learning algorithm can overshoot the optimal solution, if it is too small, then learning can take a very long time.

c) Regularization coefficient – the purpose of regularization is to prevent from overfitting. If the coefficient is too small, then the network can experience overfitting, if it is too big, then it may have the opposite problem – underfitting.

d) Number of iterations – the network should be trained until its accuracy stops improving.

e) Number of minibatches – instead of using all training examples at once to calculate gradient and update weights, the dataset is divided into parts and only a part of dataset is used at once. It improves speed of learning and also its accuracy. This number specifies into how many parts the training set is divided.

6.1 Tests with various parameters

a) variable - number of hidden units:

learning rate – 0,0002
regularization parameter – 0
minibatches – 100

Training set accuracy [%] / Validation set accuracy [%]				
Iteration \ Hidden units	100	300	**500**	700
1	19/19	20/20	**23/22**	23/23
5	25/25	27/26	**28/27**	29/27
10	28/28	30/29	**31/30**	30/29
15	30/29	33/31	**34/32**	31/30
20	32/31	34/32	**35/33**	34/31

b) variable - number of minibatches:

hidden units - 500
learning rate – 0,0002
regularization parameter – 0

Training set accuracy [%] / Validation set accuracy [%]					
Iteration \ Minibatches	50	100	200	**400**	600
1	21/21	22/22	21/20	**22/22**	22/21
5	23/23	27/26	28/27	**29/28**	28/27
10	29/28	30/28	31/30	**32/30**	32/30
15	29/29	33/31	33/32	**34/32**	34/32
20	30/29	33/31	34/33	**35/33**	35/33

c) variable - regularization parameter:

hidden units - 500
learning rate – 0,0002
minibatches - 400

Training set accuracy [%] / Validation set accuracy [%]			
Iteration \ Regularization parameter	0	**0,1**	0,2
1	22/22	**22/21**	22/21
5	29/28	**28/27**	28/27
10	32/30	**31/30**	33/31
15	34/32	**34/34**	35/33
20	35/33	**35/34**	37/35

6.2. Choice of parameters

The neural network was tested with various parameters by performing 20 iterations. Results were similar, so choosing which parameters are right wasn't obvious. Number of hidden units, number of minibatches and regularization parameter were chosen according to tests above. Learning rate was tuned to such number that accuracy is constantly improving without noises. Number of iterations was tuned to such number that further learning brings no improvements in accuracy.

Chosen parameters:

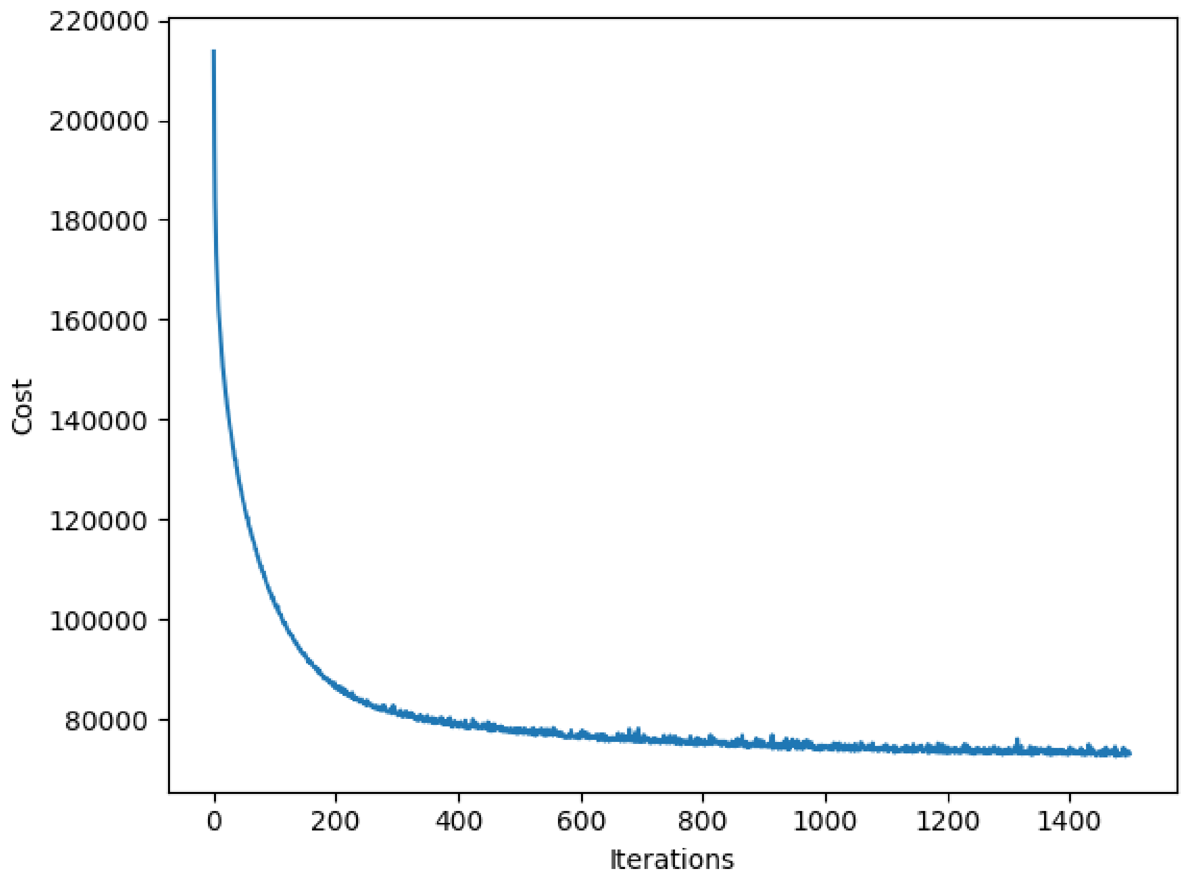
Hidden units	500
Learning rate	0,0002
Regularization parameter	0,1
Iterations	1500
Minibatches	400

6.3. Effects of training

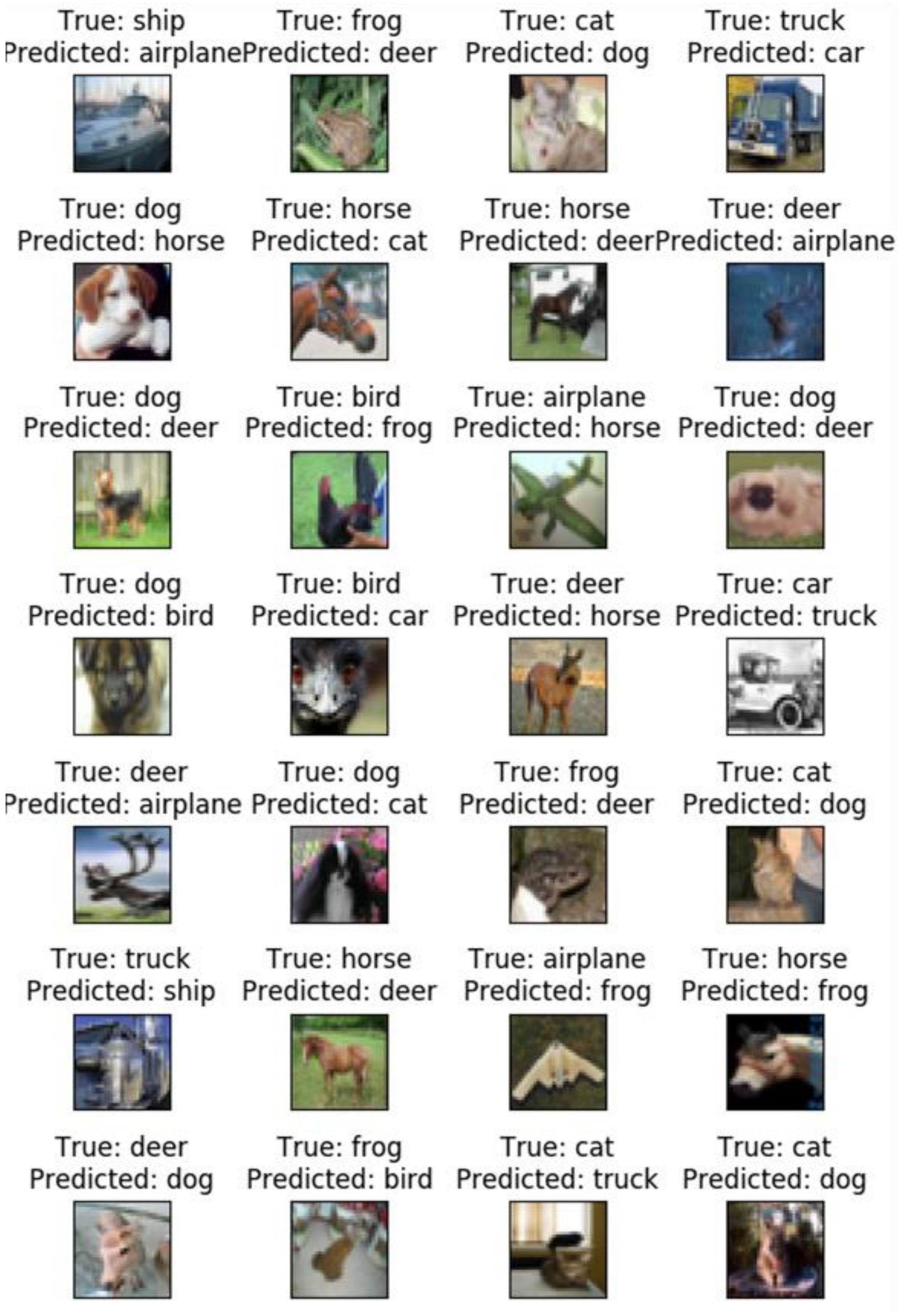
Neural network accuracy:

Training set accuracy	64 %
Test set accuracy	53 %

Gradient convergence:

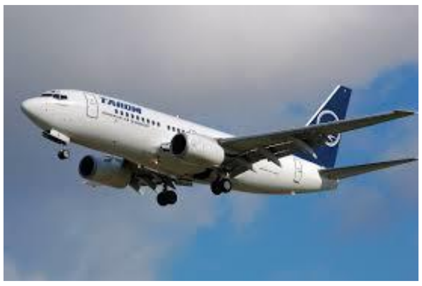
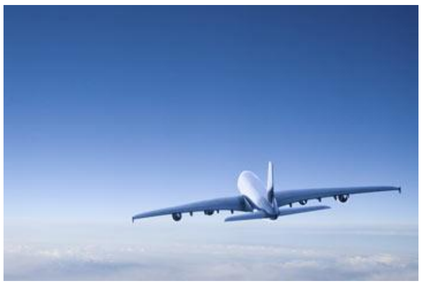
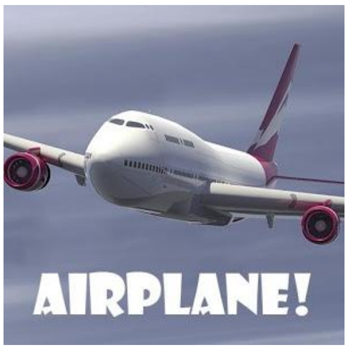
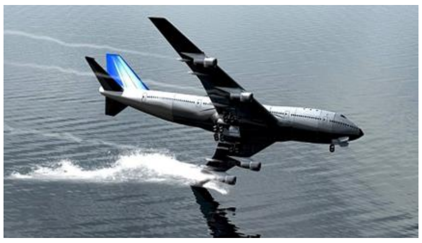
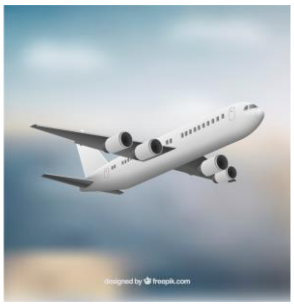
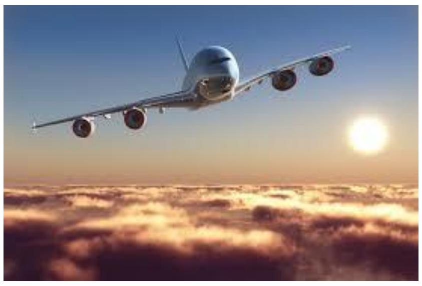
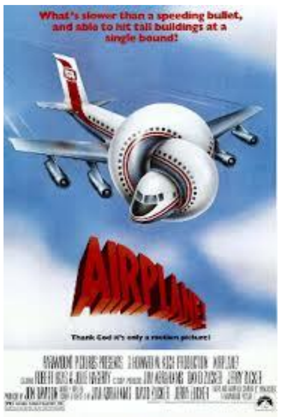
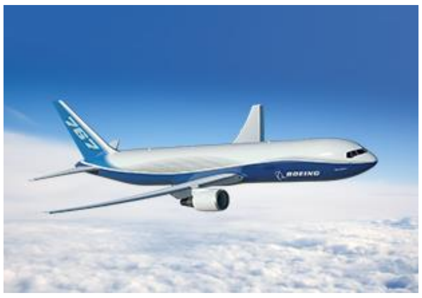
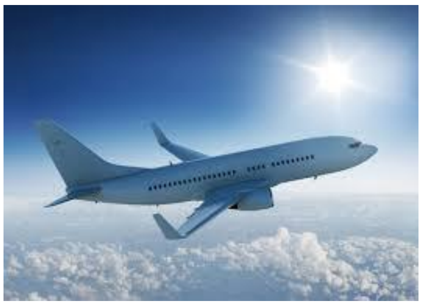
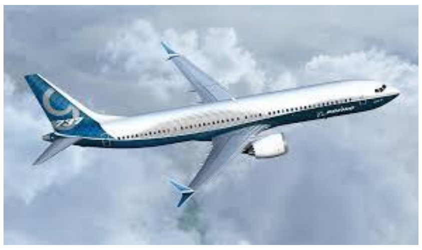


Sample misclassified images from test set:



7. Tests of the application
Tests were performed on 10 first images from google images from each category.

<u>a) Airplane – 8/10 correct:</u>

		
Airplane	Airplane	Airplane
		
Bird	Airplane	Ship
 What's slower than a speeding bullet, and able to hit tall buildings at a single bound! AIRPLANE! Thank God it's only a motion picture!		
Airplane	Airplane	Airplane
		
Airplane		

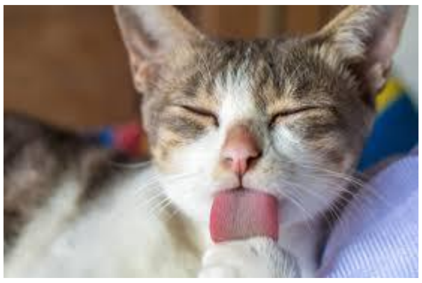
b) Bird – 5/10 correct:

	Kingfisher	
Cat	Bird	Deer
Bird	Bird	Deer
Dog	Frog	Bird
Bird		

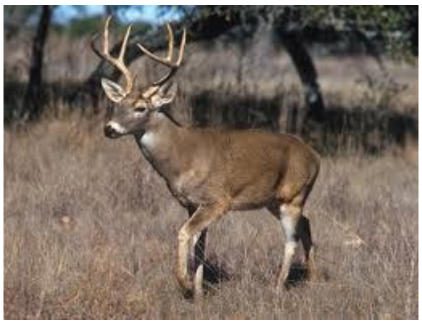
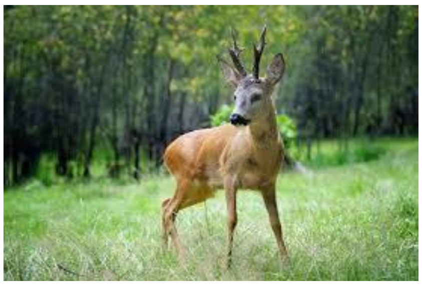
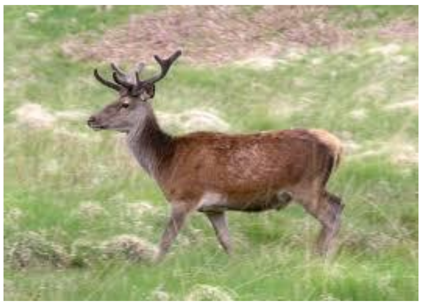
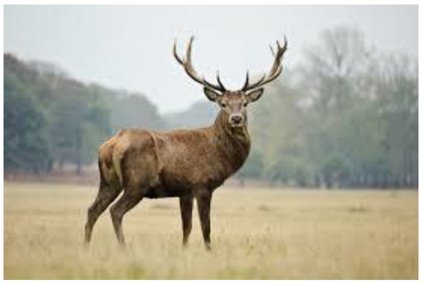
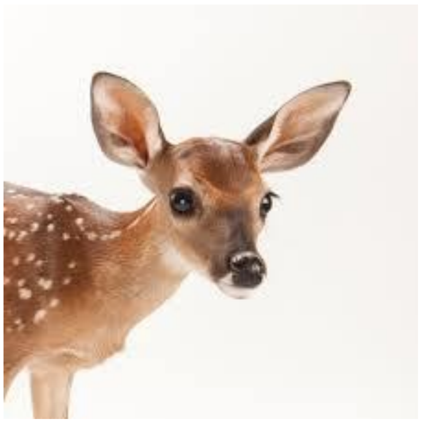
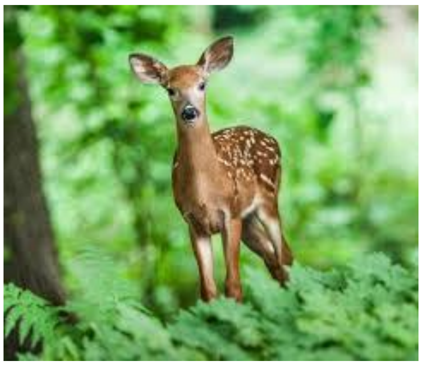
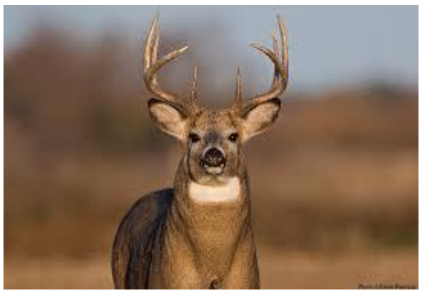
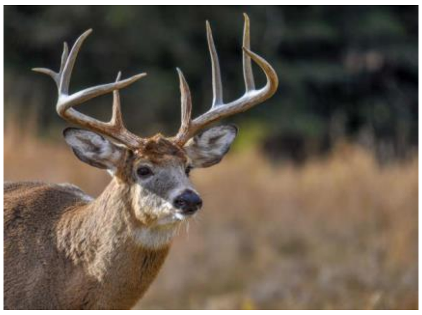
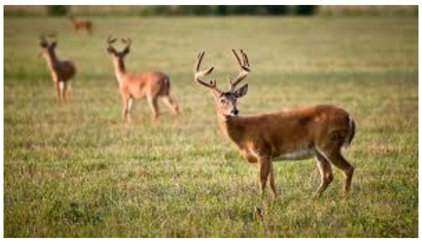
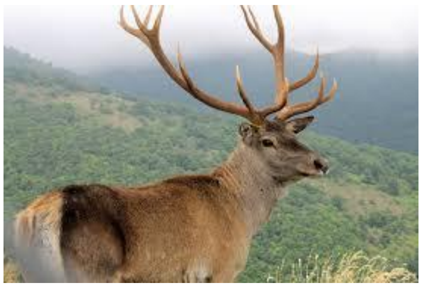
c) Car – 10/10 correct:

Car	Car	Car
Car	Car	Car
Car	Car	Car
Car		

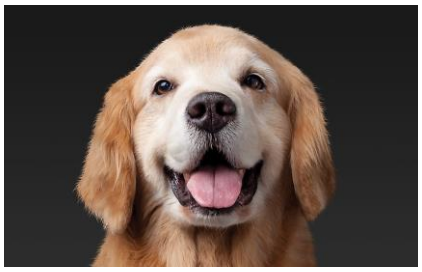
d) Cat – 2/10 correct:

		
Cat	Horse	Bird
Cat	Dog	Dog
Ship	Airplane	Dog
Airplane		

e) Deer – 7/10 correct:

		
Deer	Deer	Deer
		
Airplane	Cat	Deer
		
Horse	Deer	Deer
		
Deer		

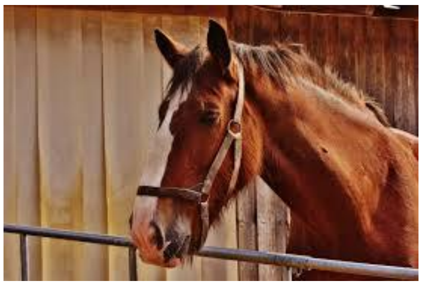
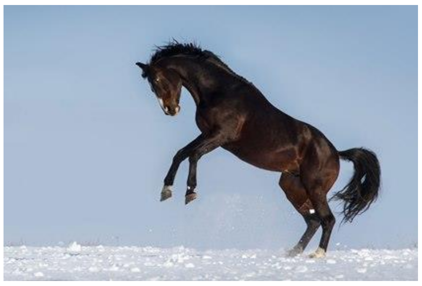
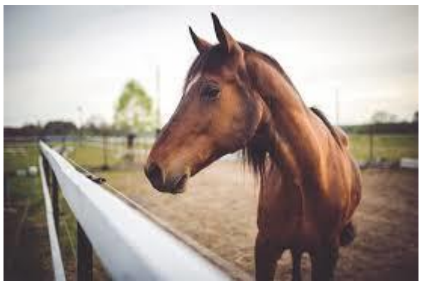
f) Dog – 4/10 correct:

		
Dog	Dog	Dog
Dog	Bird	Bird
Frog	Airplane	Airplane
Airplane		

g) Frog – 10/10 correct:

Frog	Frog	Frog
Frog	Frog	Frog
Frog	Frog	Frog
Frog		

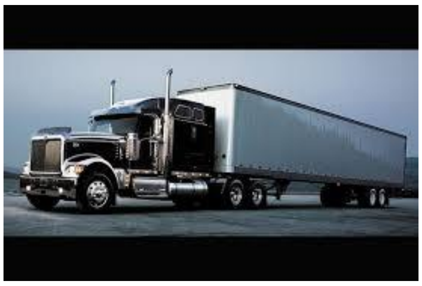
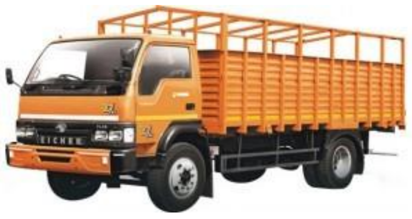
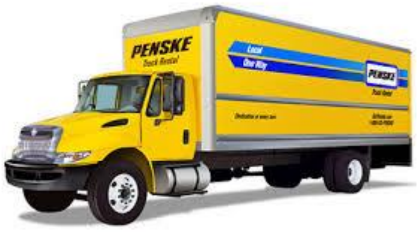
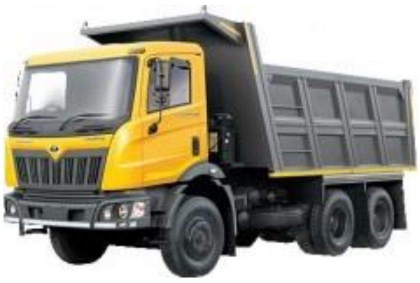
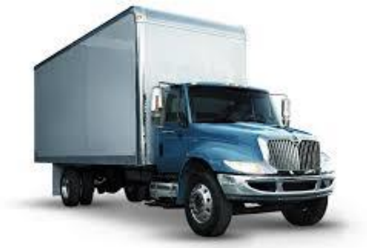
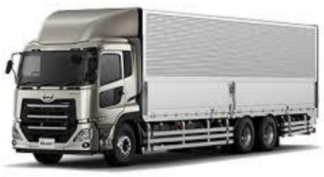
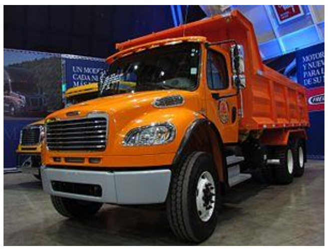
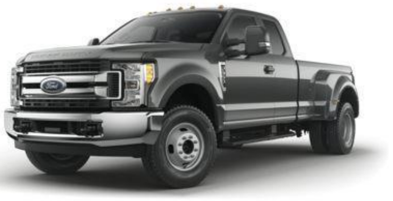
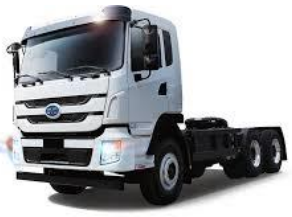
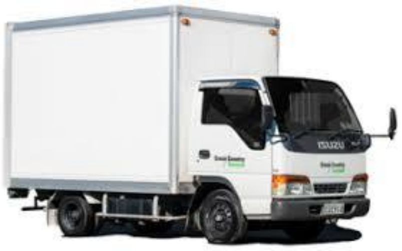
h) Horse – 7/10 correct:

Horse	Horse	Horse
Horse	Horse	Horse
		
Cat	Airplane	Horse
		
Dog		

<u>i) Ship – 10/10 correct:</u>

Ship	Ship	Ship
Ship	Ship	Ship
Ship	Ship	Ship
Ship		

j) Truck – 6/10 correct:

		
Airplane	Truck	Truck
		
Truck	Car	Truck
		
Frog	Truck	Car
		
Truck		

Accuracy of the neural network										
Image \ Category	Airplane	Bird	Car	Cat	Deer	Dog	Frog	Horse	Ship	Truck
1	+	-	+	+	+	+	+	+	+	-
2	+	+	+	-	+	+	+	+	+	+
3	+	-	+	-	+	+	+	+	+	+
4	-	+	+	+	-	+	+	+	+	+
5	+	+	+	-	-	-	+	+	+	-
6	-	-	+	-	+	-	+	+	+	+
7	+	-	+	-	-	-	+	-	+	-
8	+	-	+	-	+	-	+	-	+	+
9	+	+	+	-	+	-	+	-	+	+
10	+	+	+	-	+	-	+	-	+	+
Accuracy [%]	**80**	**50**	**100**	**20**	**70**	**40**	**100**	**60**	**100**	**60**
Overall accuracy: 68 %										

7.1. Conclusion

Application accuracy on images from internet is similar to that achieved on training set. Some categories have significantly better accuracy than others. Cars, ships and frogs were recognized correctly in every case, while recognizing cats doesn’t work at all, results are random. All test images presented only one thing in typical pose. Providing other images results in random outcome. It is a consequence of training set, which contains only images like that. Other thing that affects accuracy is quality of training set. Training images are small and poor quality, some of them are difficult to recognize for human. Also neural network that was applied is quite simple, probably using deep convolutional neural network would result in higher accuracy. Still, taking into account simple implementation of the neural network and poor quality of training data, the accuracy is satisfactory.

8. Sources

- Nielsen M., *Neural Networks and Deep Learning,* http://neuralnetworksanddeeplearning.com/
- https://www.cs.toronto.edu/~kriz/cifar.html
- Machine Learning course by Andrew Ng, Stanford University,
  https://www.coursera.org/learn/machine-learning
- http://scikit-learn.org/stable/documentation.html
- http://www.numpy.org/
- https://en.wikibooks.org/wiki/Artificial_Neural_Networks
- Convolutional Neural Networks for Visual Recognition course, Stanford University,
  http://cs231n.github.io/
- Raschka S., *Python Machine Learning*, Packt Publishing, Birmingham 2015