!mkdir cifar10 !curl -o cifar-10-python.tar.gz https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz (https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz) !tar -xvzf cifar-10-python.tar.gz -C cifar10

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import pygpu

%matplotlib inline
```

Попытки создать свою архитектуру, к сожалению, не дали особых результатов, так что, как и написано

в задании, попробуем обучить уже существующую сеть.

Источник: https://github.com/Lasagne/Recipes/blob/master/papers/deep_residual_learning/Deep_Residual_Learning 10.py (https://github.com/Lasagne/Recipes/blob/master/papers/deep_residual_learning/Deep_Residual_Learnin 10.py)

In [2]:

```
"""
Lasagne implementation of CIFAR-10 examples from "Deep Residual Learning for Image
Check the accompanying files for pretrained models. The 32-layer network (n=5), ach
while the 56-layer network (n=9) achieves error of 6.75%, which is roughly equivale
"""

from __future__ import print_function

import sys
import os
import time
import string
import random
import pickle

import numpy as np
import theano
import theano.tensor as T
import lasagne

# for the larger networks (n>=9), we need to adjust pythons recursion limit
sys.setrecursionlimit(10000)

# ##################### Load data from CIFAR-10 dataset #######################
# this code assumes the cifar dataset from 'https://www.cs.toronto.edu/~kriz/cifar-
# has been extracted in current working directory

def unpickle(file):
    fo = open(file, 'rb')
    dict = pickle.load(fo, encoding='bytes')
    fo.close()
    return dict

def load_data():
    xs = []
    ys = []
    for j in range(5):
      d = unpickle('cifar10/cifar-10-batches-py/data_batch_'+ str(j+1))
      x = d[b'data']
      y = d[b'labels']
      xs.append(x)
      ys.append(y)

    d = unpickle('cifar10/cifar-10-batches-py/test_batch')
    xs.append(d[b'data'])
    ys.append(d[b'labels'])

    x = np.concatenate(xs)/np.float32(255)
    y = np.concatenate(ys)
    x = np.dstack((x[:, :1024], x[:, 1024:2048], x[:, 2048:]))
    x = x.reshape((x.shape[0], 32, 32, 3)).transpose(0,3,1,2)

    # subtract per-pixel mean
    pixel_mean = np.mean(x[0:50000],axis=0)
    #pickle.dump(pixel_mean, open("cifar10-pixel_mean.pkl","wb"))
    x -= pixel_mean

    # create mirrored images
    X_train = x[0:50000,:,:,:]
    Y_train = y[0:50000]
    X_train_flip = X_train[:,:,:,::-1]
```

```
    Y_train_flip = Y_train
    X_train = np.concatenate((X_train,X_train_flip),axis=0)
    Y_train = np.concatenate((Y_train,Y_train_flip),axis=0)

    X_test = x[50000:,:,:,:]
    Y_test = y[50000:]

    return(
        lasagne.utils.floatX(X_train),
        Y_train.astype('int32'),
        lasagne.utils.floatX(X_test),
        Y_test.astype('int32'),)
```

```
Using cuDNN version 5110 on context None
Mapped name None to device cuda0: GRID K520 (0000:00:03.0)
```

In [3]:

```
from cifar import load_CIFAR10
plt.rcParams['figure.figsize'] = (10.0, 8.0)

cifar10_dir = './cifar10/cifar-10-batches-py'
X_train, y_train, X_test, y_test = load_data()
print(len(X_train))
```

```
100000
```

First of all -- Checking Questions

Вопрос 1: Чем отличаются современные сверточные сети от сетей 5 летней давности?

5 лет назад вычислительные мощности были сильно ниже нынешних, в связи с чем обучение сверточных нейросетей было намного более длительным процессом. Современные технологии позволили увеличить сверточные сети и обучать их за адекватное время.

Вопрос 2: Какие неприятности могут возникнуть во время обучения современных нейросетей?

Если обучить нейросеть на выборке с каким-то доминирующим классом, то она будет видеть этот класс практически везде (на лекции было забавное видео с миром, состоящим из собак).

Вопрос 3: У вас есть очень маленький датасет из 100 картинок, но вы очень хотите использовать нейросеть, какие неприятности вас ждут и как их решить?

Во-первых, недообучение. Данных явно не хватит, чтобы обучить сеть. Можем использовать предобученную сеть.

Вопрос 4: У вас есть очень маленький датасет из 100 картинок, классификация, но вы очень хотите использовать нейросеть, какие неприятности вас ждут и как их решить? что делать если первый вариант решения не заработает?

<Ответ>

Вопрос 5: Как сделать стайл трансфер для музыки? oO

Возможно, стоит попробовать что-то типа этого: получить спектрограмму звука → пропустить ее через предобученную сеть → восстановить по спектрограмме звук.

Соберите нейронку:

- Many times x (Conv+Pool)
- Many small convolutions like 3x3
- Batch Norm
- Residual Connection
- Data Augmentation
- Learning rate Schedule
- ...

Для вдохновения

- http://torch.ch/blog/2015/07/30/cifar.html (http://torch.ch/blog/2015/07/30/cifar.html)
- http://www.robots.ox.ac.uk/~vgg/research/very_deep/ (http://www.robots.ox.ac.uk/~vgg/research/very_deep/)
- https://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf (https://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf)
- https://github.com/szagoruyko/wide-residual-networks (https://github.com/szagoruyko/wide-residual-networks)

Самое интересное

- Для сдачи задания нужно набрать на точность тесте > **92.5**% (это займет много времени, торопитесь :))
- Для получения бонусных баллов > **95.0**%
- Будет очень хорошо если вы придумаете свою архитектуру или сможете обучить что-то из вышеперечисленного :)
- А для обучения всего этого добра вам будет куда удобнее использовать GPU на Amazon
  - Инструкция https://github.com/persiyanov/ml-mipt/tree/master/amazon-howto (https://github.com/persiyanov/ml-mipt/tree/master/amazon-howto)
  - Вам помогут tmux, CuDNN, ssh tunnel, nvidia-smi, ...
  - Wish you get fun :)

In [4]:

```
import lasagne
from theano import tensor as T
from lasagne.nonlinearities import *

input_X = T.tensor4("X")
target_y = T.vector("target Y integer",dtype='int32')
```

In [5]:

```
from lasagne.layers import Conv2DLayer as ConvLayer
#from lasagne.layers.dnn import Conv2DDNNLayer as ConvLayer
from lasagne.layers import ElemwiseSumLayer
from lasagne.layers import InputLayer
from lasagne.layers import DenseLayer
from lasagne.layers import GlobalPoolLayer
from lasagne.layers import PadLayer
from lasagne.layers import ExpressionLayer
from lasagne.layers import NonlinearityLayer
from lasagne.nonlinearities import softmax, rectify
from lasagne.layers import batch_norm

def build_cnn(input_var=None, n=5):

    # create a residual learning building block with two stacked 3x3 convlayers as
    def residual_block(l, increase_dim=False, projection=False):
        input_num_filters = l.output_shape[1]
        if increase_dim:
            first_stride = (2,2)
            out_num_filters = input_num_filters*2
        else:
            first_stride = (1,1)
            out_num_filters = input_num_filters

        stack_1 = batch_norm(ConvLayer(l, num_filters=out_num_filters, filter_size=
        stack_2 = batch_norm(ConvLayer(stack_1, num_filters=out_num_filters, filter

        # add shortcut connections
        if increase_dim:
            if projection:
                # projection shortcut, as option B in paper
                projection = batch_norm(ConvLayer(l, num_filters=out_num_filters, f
                block = NonlinearityLayer(ElemwiseSumLayer([stack_2, projection]),n
            else:
                # identity shortcut, as option A in paper
                identity = ExpressionLayer(l, lambda X: X[:, :, ::2, ::2], lambda s
                padding = PadLayer(identity, [out_num_filters//4,0,0], batch_ndim=1
                block = NonlinearityLayer(ElemwiseSumLayer([stack_2, padding]),nonl
        else:
            block = NonlinearityLayer(ElemwiseSumLayer([stack_2, l]),nonlinearity=r

        return block

    # Building the network
    l_in = InputLayer(shape=(None, 3, 32, 32), input_var=input_var)

    # first layer, output is 16 x 32 x 32
    l = batch_norm(ConvLayer(l_in, num_filters=16, filter_size=(3,3), stride=(1,1),

    # first stack of residual blocks, output is 16 x 32 x 32
    for _ in range(n):
        l = residual_block(l)

    # second stack of residual blocks, output is 32 x 16 x 16
    l = residual_block(l, increase_dim=True)
    for _ in range(1,n):
        l = residual_block(l)

    # third stack of residual blocks, output is 64 x 8 x 8
    l = residual_block(l, increase_dim=True)
    for _ in range(1,n):
```

```
        l = residual_block(l)

    # average pooling
    l = GlobalPoolLayer(l)

    # fully connected layer
    network = DenseLayer(
            l, num_units=10,
            W=lasagne.init.HeNormal(),
            nonlinearity=softmax)

    return network
```

In [6]:

```
net = build_cnn(input_X, 9)
```

In [7]:

```
y_predicted = lasagne.layers.get_output(net)
all_weights = lasagne.layers.get_all_params(net, trainable=True)
print (all_weights)
```

```
[W, beta, gamma, W, beta, gamma, W, beta, gamma, W, beta, gamma, W, b
eta, gamma, W, beta, gamma, W, beta, gamma, W, beta, gamma, W, beta,
 gamma, W, beta, gamma, W, beta, gamma, W, beta, gamma, W, beta, gamm
a, W, beta, gamma, W, beta, gamma, W, beta, gamma, W, beta, gamma, W,
 beta, gamma, W, beta, gamma, W, beta, gamma, W, beta, gamma, W, bet
a, gamma, W, beta, gamma, W, beta, gamma, W, beta, gamma, W, beta, ga
mma, W, beta, gamma, W, beta, gamma, W, beta, gamma, W, beta, gamma,
 W, beta, gamma, W, beta, gamma, W, beta, gamma, W, beta, gamma, W, b
eta, gamma, W, beta, gamma, W, beta, gamma, W, beta, gamma, W, beta,
 gamma, W, beta, gamma, W, beta, gamma, W, beta, gamma, W, beta, gamm
a, W, beta, gamma, W, beta, gamma, W, beta, gamma, W, beta, gamma, W,
 beta, gamma, W, beta, gamma, W, beta, gamma, W, beta, gamma, W, bet
a, gamma, W, beta, gamma, W, beta, gamma, W, beta, gamma, W, b]
```

In [8]:

```
penalty = 0.0001 * lasagne.regularization.regularize_layer_params(lasagne.layers.ge
loss = lasagne.objectives.categorical_crossentropy(y_predicted, target_y).mean()
loss = loss + penalty
accuracy = lasagne.objectives.categorical_accuracy(y_predicted, target_y).mean()
```

In [9]:

```
updates = lasagne.updates.momentum(loss, all_weights, learning_rate=0.1, momentum=0
```

In [10]:

```
train_fun = theano.function([input_X, target_y], [loss, accuracy], updates=updates,
accuracy_fun = theano.function([input_X, target_y], accuracy, allow_input_downcast=
```

Вот и всё, пошли её учить

In [11]:

```
def iterate_minibatches(inputs, targets, batchsize, shuffle=False, augment=False):
    assert len(inputs) == len(targets)
    if shuffle:
        indices = np.arange(len(inputs))
        np.random.shuffle(indices)
    for start_idx in range(0, len(inputs) - batchsize + 1, batchsize):
        if shuffle:
            excerpt = indices[start_idx:start_idx + batchsize]
        else:
            excerpt = slice(start_idx, start_idx + batchsize)
        if augment:
            # as in paper :
            # pad feature arrays with 4 pixels on each side
            # and do random cropping of 32x32
            padded = np.pad(inputs[excerpt],((0,0),(0,0),(4,4),(4,4)),mode='constan
            random_cropped = np.zeros(inputs[excerpt].shape, dtype=np.float32)
            crops = np.random.random_integers(0,high=8,size=(batchsize,2))
            for r in range(batchsize):
                random_cropped[r,:,:,:] = padded[r,:,crops[r,0]:(crops[r,0]+32),cro
            inp_exc = random_cropped
        else:
            inp_exc = inputs[excerpt]

        yield inp_exc, targets[excerpt]
```

In [12]:

```
f = open("network.txt", "w")
```

In [ ]:

```
learning_rate = 0.1
```

В результате ошибки в if-е, уменьшающем learning_rate, я была вынуждена остановить обучение

и продолжить после исправления. В результате потерялась часть истории обучения на 1-41 эпохах.

До дедлайна 1,5 часа - перезапустить с нуля не успеваю. Поставлю пересчитываться и залью отдельным документом

отчет по accuracy с нуля. Надеюсь, это не слишком критично.

$\Downarrow$

Процесс обучения

In [ ]:

```
import time

num_epochs = 80 #количество проходов по данным
batch_size = 128 #размер мини-батча



for epoch in range(0, num_epochs):
    # In each epoch, we do a full pass over the training data:
    train_err = 0
    train_acc = 0
    train_batches = 0
    start_time = time.time()
    for batch in iterate_minibatches(X_train, y_train, batch_size, True, True):
        inputs, targets = batch
        train_err_batch, train_acc_batch= train_fun(inputs, targets)
        train_err += train_err_batch
        train_acc += train_acc_batch
        train_batches += 1
    # And a full pass over the validation data:
    val_acc = 0
    val_batches = 0
    for batch in iterate_minibatches(X_test, y_test, batch_size):
        inputs, targets = batch
        val_acc += accuracy_fun(inputs, targets)
        val_batches += 1

    # Then we print the results for this epoch:
    f.write("Epoch {} of {} took {:.3f}s".format(epoch + 1, num_epochs, time.time()
    "  training loss (in-iteration):\t\t{:.6f}".format(train_err / train_batches) +
    "  train accuracy:\t\t{:.2f} %".format(train_acc / train_batches * 100) + '\n'
    "  validation accuracy:\t\t{:.2f} %".format(val_acc / val_batches * 100) + '\n'
    )
    print("Epoch {} of {} took {:.3f}s".format(epoch, num_epochs, time.time() - sta
    print("  training loss (in-iteration):\t\t{:.6f}".format(train_err / train_batc
    print("  train accuracy:\t\t{:.2f} %".format(train_acc / train_batches * 100))
    print("  validation accuracy:\t\t{:.2f} %".format(val_acc / val_batches * 100))
    # adjust learning rate as in paper
    # 32k and 48k iterations should be roughly equivalent to 41 and 61 epochs

    if (epoch+1) == 43 or (epoch+1) == 52:
        print("Declining learning rate in 10 times...")
        learning_rate *= 0.1
        updates = lasagne.updates.momentum(loss, all_weights, learning_rate=learnin
        train_fun = theano.function([input_X, target_y], [loss, accuracy], updates=
```

```
/home/ec2-user/anaconda3/lib/python3.6/site-packages/ipykernel/__main
__.py:17: DeprecationWarning: This function is deprecated. Please cal
l randint(0, 8 + 1) instead
```

```
Epoch 0 of 80 took 378.519s
  training loss (in-iteration):         2.808084
  train accuracy:               24.55 %
  validation accuracy:          43.90 %
Epoch 1 of 80 took 378.480s
  training loss (in-iteration):         1.837737
  train accuracy:               54.52 %
  validation accuracy:          66.93 %
Epoch 2 of 80 took 378.501s
  training loss (in-iteration):         1.330402
  train accuracy:               70.09 %
  validation accuracy:          74.49 %
Epoch 3 of 80 took 378.541s
  training loss (in-iteration):         1.088764
  train accuracy:               76.40 %
  validation accuracy:          78.10 %
Epoch 4 of 80 took 378.547s
  training loss (in-iteration):         0.944863
  train accuracy:               79.62 %
  validation accuracy:          80.45 %
Epoch 5 of 80 took 378.598s
  training loss (in-iteration):         0.849066
  train accuracy:               81.88 %
  validation accuracy:          80.55 %
Epoch 6 of 80 took 378.569s
  training loss (in-iteration):         0.786436
  train accuracy:               83.19 %
  validation accuracy:          82.93 %
Epoch 7 of 80 took 378.629s
  training loss (in-iteration):         0.741913
  train accuracy:               84.32 %
  validation accuracy:          82.60 %
Epoch 8 of 80 took 378.637s
  training loss (in-iteration):         0.704478
  train accuracy:               85.25 %
  validation accuracy:          83.81 %
Epoch 9 of 80 took 378.660s
  training loss (in-iteration):         0.678399
  train accuracy:               85.97 %
  validation accuracy:          84.66 %
Epoch 10 of 80 took 378.711s
  training loss (in-iteration):         0.657041
  train accuracy:               86.59 %
  validation accuracy:          85.12 %
Epoch 11 of 80 took 378.690s
  training loss (in-iteration):         0.643267
  train accuracy:               87.17 %
  validation accuracy:          84.87 %
Epoch 12 of 80 took 378.663s
  training loss (in-iteration):         0.634448
  train accuracy:               87.28 %
  validation accuracy:          85.38 %
Epoch 13 of 80 took 378.716s
  training loss (in-iteration):         0.624054
  train accuracy:               87.79 %
  validation accuracy:          84.68 %
Epoch 14 of 80 took 378.694s
  training loss (in-iteration):         0.610324
  train accuracy:               88.24 %
  validation accuracy:          86.18 %
```

```
Epoch 15 of 80 took 378.694s
  training loss (in-iteration):         0.609201
  train accuracy:                88.35 %
  validation accuracy:           85.81 %
Epoch 16 of 80 took 378.701s
  training loss (in-iteration):         0.602784
  train accuracy:                88.70 %
  validation accuracy:           87.42 %
Epoch 17 of 80 took 378.686s
  training loss (in-iteration):         0.598464
  train accuracy:                88.84 %
  validation accuracy:           87.02 %
Epoch 18 of 80 took 378.742s
  training loss (in-iteration):         0.587752
  train accuracy:                89.38 %
  validation accuracy:           86.41 %
Epoch 19 of 80 took 378.762s
  training loss (in-iteration):         0.585491
  train accuracy:                89.36 %
  validation accuracy:           86.62 %
Epoch 20 of 80 took 378.724s
  training loss (in-iteration):         0.579152
  train accuracy:                89.81 %
  validation accuracy:           86.82 %
Epoch 21 of 80 took 378.714s
  training loss (in-iteration):         0.581053
  train accuracy:                89.77 %
  validation accuracy:           87.24 %
Epoch 22 of 80 took 378.729s
  training loss (in-iteration):         0.575041
  train accuracy:                89.89 %
  validation accuracy:           87.08 %
Epoch 23 of 80 took 378.777s
  training loss (in-iteration):         0.575473
  train accuracy:                89.97 %
  validation accuracy:           87.45 %
Epoch 24 of 80 took 378.759s
  training loss (in-iteration):         0.573153
  train accuracy:                90.19 %
  validation accuracy:           86.68 %
Epoch 25 of 80 took 378.759s
  training loss (in-iteration):         0.569565
  train accuracy:                90.30 %
  validation accuracy:           86.98 %
Epoch 26 of 80 took 378.747s
  training loss (in-iteration):         0.566468
  train accuracy:                90.38 %
  validation accuracy:           87.75 %
Epoch 27 of 80 took 378.765s
  training loss (in-iteration):         0.566764
  train accuracy:                90.52 %
  validation accuracy:           87.43 %
Epoch 28 of 80 took 378.752s
  training loss (in-iteration):         0.561814
  train accuracy:                90.71 %
  validation accuracy:           87.33 %
Epoch 29 of 80 took 378.745s
  training loss (in-iteration):         0.566068
  train accuracy:                90.58 %
  validation accuracy:           88.06 %
Epoch 30 of 80 took 378.757s
```

```
  training loss (in-iteration):         0.558696
  train accuracy:               90.86 %
  validation accuracy:          87.18 %
Epoch 31 of 80 took 378.781s
  training loss (in-iteration):         0.559322
  train accuracy:               90.94 %
  validation accuracy:          87.59 %
Epoch 32 of 80 took 378.787s
  training loss (in-iteration):         0.557029
  train accuracy:               90.98 %
  validation accuracy:          88.42 %
Epoch 33 of 80 took 378.785s
  training loss (in-iteration):         0.556903
  train accuracy:               91.08 %
  validation accuracy:          87.92 %
Epoch 34 of 80 took 378.831s
  training loss (in-iteration):         0.555281
  train accuracy:               91.13 %
  validation accuracy:          88.37 %
Epoch 35 of 80 took 378.847s
  training loss (in-iteration):         0.554841
  train accuracy:               91.17 %
  validation accuracy:          87.34 %
```

In [16]:

```
import time

num_epochs = 80 #количество проходов по данным
batch_size = 128 #размер мини-батча



for epoch in range(35, num_epochs):
    # In each epoch, we do a full pass over the training data:
    train_err = 0
    train_acc = 0
    train_batches = 0
    start_time = time.time()
    for batch in iterate_minibatches(X_train, y_train, batch_size, True, True):
        inputs, targets = batch
        train_err_batch, train_acc_batch= train_fun(inputs, targets)
        train_err += train_err_batch
        train_acc += train_acc_batch
        train_batches += 1
    # And a full pass over the validation data:
    val_acc = 0
    val_batches = 0
    for batch in iterate_minibatches(X_test, y_test, batch_size):
        inputs, targets = batch
        val_acc += accuracy_fun(inputs, targets)
        val_batches += 1

    # Then we print the results for this epoch:
    f.write("Epoch {} of {} took {:.3f}s".format(epoch + 1, num_epochs, time.time()
    "  training loss (in-iteration):\t\t{:.6f}".format(train_err / train_batches) +
    "  train accuracy:\t\t{:.2f} %".format(train_acc / train_batches * 100) + '\n'
    "  validation accuracy:\t\t{:.2f} %".format(val_acc / val_batches * 100) + '\n'
    )
    print("Epoch {} of {} took {:.3f}s".format(epoch, num_epochs, time.time() - sta
    print("  training loss (in-iteration):\t\t{:.6f}".format(train_err / train_batc
    print("  train accuracy:\t\t{:.2f} %".format(train_acc / train_batches * 100))
    print("  validation accuracy:\t\t{:.2f} %".format(val_acc / val_batches * 100))
    # adjust learning rate as in paper
    # 32k and 48k iterations should be roughly equivalent to 41 and 61 epochs

    if (epoch+1) == 35 or (epoch+1) == 42:
        print("Declining learning rate in 10 times...")
        learning_rate *= 0.1
        updates = lasagne.updates.momentum(loss, all_weights, learning_rate=learnin
        train_fun = theano.function([input_X, target_y], [loss, accuracy], updates=
```

```
/home/ec2-user/anaconda3/lib/python3.6/site-packages/ipykernel/__main
__.py:17: DeprecationWarning: This function is deprecated. Please cal
l randint(0, 8 + 1) instead
```

```
Epoch 35 of 80 took 378.958s
  training loss (in-iteration):         0.235847
  train accuracy:               99.69 %
  validation accuracy:          92.33 %
Epoch 36 of 80 took 378.969s
  training loss (in-iteration):         0.235088
  train accuracy:               99.73 %
  validation accuracy:          92.44 %
Epoch 37 of 80 took 378.956s
  training loss (in-iteration):         0.234784
  train accuracy:               99.69 %
  validation accuracy:          92.46 %
Epoch 38 of 80 took 378.969s
  training loss (in-iteration):         0.233121
  train accuracy:               99.76 %
  validation accuracy:          92.33 %
Epoch 39 of 80 took 378.958s
  training loss (in-iteration):         0.232883
  train accuracy:               99.74 %
  validation accuracy:          92.41 %
Epoch 40 of 80 took 378.986s
  training loss (in-iteration):         0.232297
  train accuracy:               99.72 %
  validation accuracy:          92.38 %
Epoch 41 of 80 took 378.974s
  training loss (in-iteration):         0.231457
  train accuracy:               99.72 %
  validation accuracy:          92.52 %
Declining learning rate in 10 times...
Epoch 42 of 80 took 377.420s
  training loss (in-iteration):         0.230771
  train accuracy:               99.76 %
  validation accuracy:          92.52 %
Epoch 43 of 80 took 377.422s
  training loss (in-iteration):         0.230191
  train accuracy:               99.75 %
  validation accuracy:          92.50 %

---------------------------------------------------------------------
------
KeyboardInterrupt                         Traceback (most recent call
 last)
<ipython-input-16-175ce292d8fa> in <module>()
     14     for batch in iterate_minibatches(X_train, y_train, batch_
size, True, True):
     15         inputs, targets = batch
---> 16         train_err_batch, train_acc_batch= train_fun(inputs, t
argets)
     17         train_err += train_err_batch
     18         train_acc += train_acc_batch

/home/ec2-user/anaconda3/lib/python3.6/site-packages/theano/compile/f
unction_module.py in __call__(self, *args, **kwargs)
    882         try:
    883             outputs =\
--> 884                 self.fn() if output_subset is None else\
    885                 self.fn(output_subset=output_subset)
    886         except Exception:

KeyboardInterrupt:
```

Тут уже достигается необходимый уровень точности, поэтому прервем обучение

In [17]:

```
test_acc = 0
test_batches = 0
for batch in iterate_minibatches(X_test, y_test, 500):
    inputs, targets = batch
    acc = accuracy_fun(inputs, targets)
    test_acc += acc
    test_batches += 1
print("Final results:")
print("  test accuracy:\t\t{:.2f} %".format(
    test_acc / test_batches * 100))

if test_acc / test_batches * 100 > 92.5:
    print ("Achievement unlocked: колдун 80 уровня")
else:
    print ("Нужно больше магии!")
```

```
Final results:
  test accuracy:                92.62 %
Achievement unlocked: колдун 80 уровня
```

In [ ]:

```

```

Заполните форму

https://goo.gl/forms/EeadABISlVmdJqgr2 (https://goo.gl/forms/EeadABISlVmdJqgr2)

In [ ]:

```

```