**Ваша первая нейронная сеть на графическом процессоре (GPU).**

**https://habr.com/ru/post/488564/**

**Устанавливаем окружение**

Основным языком для разработки ML моделей является Python. А наиболее популярной платформой для его использование по Linux является [Anaconda](https://www.anaconda.com/distribution/).  
Установим ее на наш сервер.  
Начинаем с обновления локального менеджера пакетов:

sudo apt-get update

Устанавливаем curl (служебная программа командной строки):

sudo apt-get install curl

Скачиваем последнюю версию Anaconda Distribution:

cd /tmp

curl –O **https://repo.anaconda.com/archive/Anaconda3-2021.05-Linux-x86\_64.sh**

Запускаем установку:

bash Anaconda3-2019.10-Linux-x86\_64.sh

В процессе установки необходимо будет подтвердить лицензионное соглашение. При успешной установке вы должны будете увидеть это:

Thank you for installing Anaconda3!

Для разработки ML моделей сейчас создано множество фреймворков, мы работаем с наиболее популярными: [PyTorch](https://pytorch.org/) и [Tensorflow](https://www.tensorflow.org).  
Использование фреймворка позволяет увеличить скорость разработки и использовать уже готовые инструменты для стандартных задач.  
В этом примере будем работать с PyTorch. Установим его:

conda install pytorch torchvision cudatoolkit=10.1 -c pytorch

Теперь нам необходимо запустить Jupyter Notebook — популярный у ML специалистов инструмент разработки. Он позволяет писать код и сразу видеть результаты его выполнения. Jupyter Notebook входит в состав Anaconda и уже установлен на нашем сервере. Необходимо подключится к нему из нашей настольной системе.  
Для этого мы сначала запустим Jupyter на сервере указав порт 8080:

jupyter notebook --no-browser --port=8080 --allow-root

Далее открыв в нашей консоли Cmder еще одну вкладку (верхнее меню — New console dialog) подключимся по порту 8080 к серверу через SSH:

ssh -L 8080:localhost:8080 root@server-ip-or-hostname

При вводе первой команды нам будет предложены ссылки для открытия Jupyter в нашем браузере:

To access the notebook, open this file in a browser:

file:///root/.local/share/jupyter/runtime/nbserver-18788-open.html

Or copy and paste one of these URLs:

http://localhost:8080/?token=cca0bd0b30857821194b9018a5394a4ed2322236f116d311

or http://127.0.0.1:8080/?token=cca0bd0b30857821194b9018a5394a4ed2322236f116d311

Воспользуемся ссылкой для localhost:8080. Скопируйте полный путь и вставьте в адресную строку локального браузера вашего ПК. Откроется Jupyter Notebook.  
Создадим новый ноутбук: New — Notebook — Python 3.  
Проверим корректную работу всех компонентов которые мы установили. Введем в Jupyter пример кода PyTorch и запустим выполнение (кнопка Run):

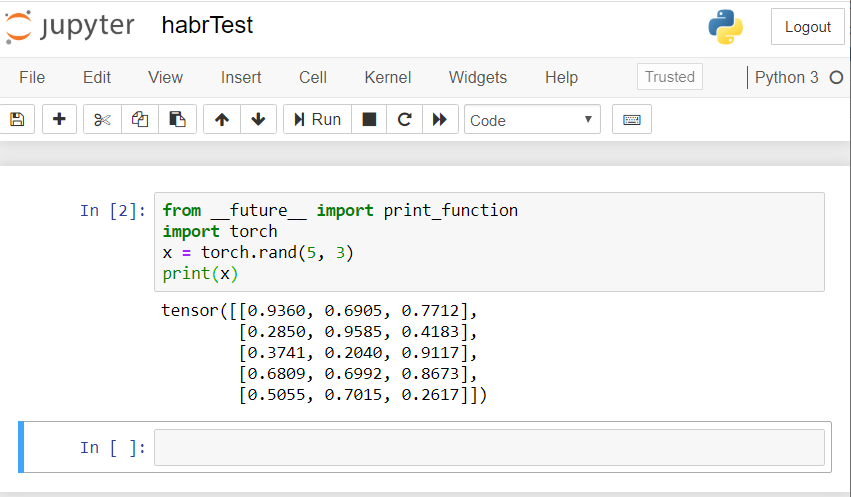
from \_\_future\_\_ import print\_function

import torch

x = torch.rand(5, 3)

print(x)

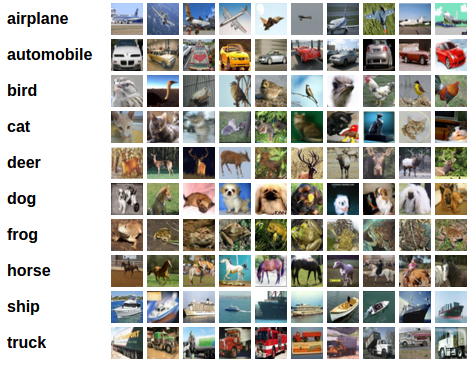
Результат должен быть примерно таким:



Если у вас аналогичный результат — значит мы все настроили правильно и можем приступать к разработке нейронной сети!

**Создаем нейронную сеть**

Будем создавать нейронную сеть для распознавания изображений. За основу возьмем данное [руководство](https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html). (https://pytorch.org/tutorials/beginner/deep\_learning\_60min\_blitz.html)  
  
Для тренировки сети мы будем использовать общедоступный набор данных CIFAR10. У него есть классы: «самолет», «автомобиль», «птица», «кошка», «олень», «собака», «лягушка», «лошадь», «корабль», «грузовик». Изображения в CIFAR10 имеют размер 3x32x32, то есть 3-канальные цветные изображения размером 32x32 пикселей.



Для работы мы будем использовать созданный PyTorch пакет для работы с изображениями — torchvision.  
  
Мы сделаем следующие шаги по порядку:

* Загрузка и нормализация наборов обучающих и тестовых данных
* Определение нейронной сети
* Тренировка сети на тренировочных данных
* Тестирование сети на тестовых данных
* Повторим тренировку и тестирование с использованием GPU

Весь приведенный ниже код мы будем выполнять в Jupyter Notebook.

**Загрузка и нормализация CIFAR10**

Скопируйте и выполните в Jupyter следующий код:

import torch

import torchvision

import torchvision.transforms as transforms

transform = transforms.Compose(

[transforms.ToTensor(),

transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,

download=True, transform=transform)

trainloader = torch.utils.data.DataLoader(trainset, batch\_size=4,

shuffle=True, num\_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,

download=True, transform=transform)

testloader = torch.utils.data.DataLoader(testset, batch\_size=4,

shuffle=False, num\_workers=2)

classes = ('plane', 'car', 'bird', 'cat',

'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

Ответ должен быть такой:

Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz

Extracting ./data/cifar-10-python.tar.gz to ./data

Files already downloaded and verified

Выведем несколько тренировочных образов для проверки:

import matplotlib.pyplot as plt

import numpy as np

# functions to show an image

def imshow(img):

img = img / 2 + 0.5 # unnormalize

npimg = img.numpy()

plt.imshow(np.transpose(npimg, (1, 2, 0)))

plt.show()

# get some random training images

dataiter = iter(trainloader)

images, labels = dataiter.next()

# show images

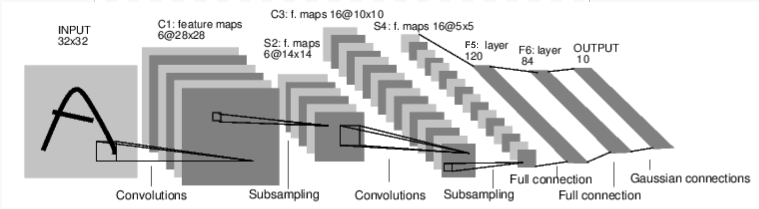
imshow(torchvision.utils.make\_grid(images))

# print labels

print(' '.join('%5s' % classes[labels[j]] for j in range(4)))



**Определение нейронной сети**

Рассмотрим сначала как работает нейронная сеть по распознаванию изображений. Это простая сеть прямой связи. Он принимает входные данные, пропускает их через несколько слоев один за другим, а затем, наконец, выдает выходные данные.  
  
  
  
Создадим подобную сеть в нашей среде:

import torch.nn as nn

import torch.nn.functional as F

class Net(nn.Module):

def \_\_init\_\_(self):

super(Net, self).\_\_init\_\_()

self.conv1 = nn.Conv2d(3, 6, 5)

self.pool = nn.MaxPool2d(2, 2)

self.conv2 = nn.Conv2d(6, 16, 5)

self.fc1 = nn.Linear(16 \* 5 \* 5, 120)

self.fc2 = nn.Linear(120, 84)

self.fc3 = nn.Linear(84, 10)

def forward(self, x):

x = self.pool(F.relu(self.conv1(x)))

x = self.pool(F.relu(self.conv2(x)))

x = x.view(-1, 16 \* 5 \* 5)

x = F.relu(self.fc1(x))

x = F.relu(self.fc2(x))

x = self.fc3(x)

return x

net = Net()

Определим так же функцию потерь и оптимизатор

import torch.optim as optim

criterion = nn.CrossEntropyLoss()

optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

**Тренировка сети на тренировочных данных**

Начинаем тренировку нашей нейронной сети. Обращаю внимание что после этого как вы запустите на выполнение этот код, нужно будет подождать некоторое время до завершения работы. У меня это заняло 5 мин. Для обучения сети нужно время.

for epoch in range(2): # loop over the dataset multiple times

running\_loss = 0.0

for i, data in enumerate(trainloader, 0):

# get the inputs; data is a list of [inputs, labels]

inputs, labels = data

# zero the parameter gradients

optimizer.zero\_grad()

# forward + backward + optimize

outputs = net(inputs)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

# print statistics

running\_loss += loss.item()

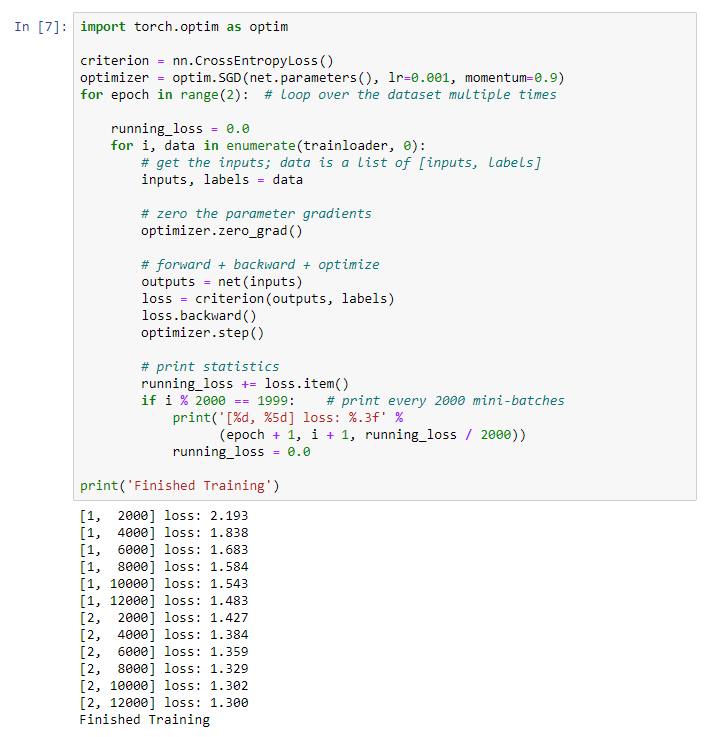
if i % 2000 == 1999: # print every 2000 mini-batches

print('[%d, %5d] loss: %.3f' %

(epoch + 1, i + 1, running\_loss / 2000))

running\_loss = 0.0

print('Finished Training')

Получим такой результат:  


Сохраняем нашу обученную модель:

PATH = './cifar\_net.pth'

torch.save(net.state\_dict(), PATH)

**Тестирование сети на тестовых данных**

Мы обучили сеть использую набор обучающих данных. Но нам нужно проверить, научилась ли сеть вообще чему-либо.  
  
Мы проверим это, предсказав метку класса, которую выводит нейронная сеть, и проверив ее на предмет истинности. Если прогноз верен, мы добавляем образец в список правильных прогнозов.  
Давайте покажем изображение из тестового набора:

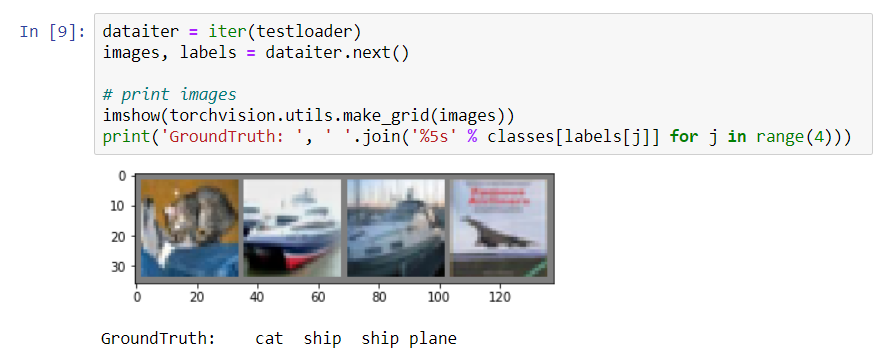
dataiter = iter(testloader)

images, labels = dataiter.next()

# print images

imshow(torchvision.utils.make\_grid(images))

print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range(4)))

  
  
Теперь попросим нейронную сеть сообщить нам что на этих картинках:

net = Net()

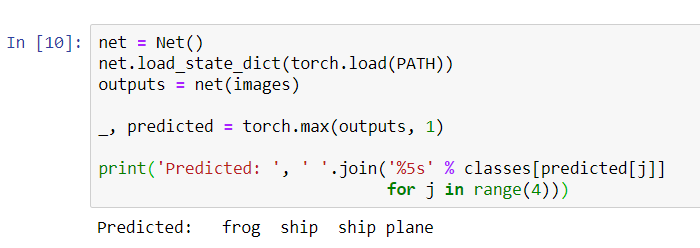
net.load\_state\_dict(torch.load(PATH))

outputs = net(images)

\_, predicted = torch.max(outputs, 1)

print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]

for j in range(4)))

  
  
Результаты кажутся довольно хорошими: сеть определила правильно три картинки из четырех.  
  
Давайте посмотрим, как сеть работает во всем наборе данных.

correct = 0

total = 0

with torch.no\_grad():

for data in testloader:

images, labels = data

outputs = net(images)

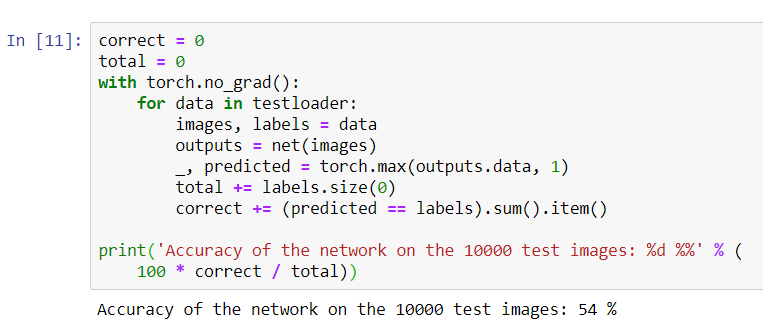
\_, predicted = torch.max(outputs.data, 1)

total += labels.size(0)

correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %d %%' % (

100 \* correct / total))

  
  
Похоже сеть что-то знает и работает. Если бы он определяла классы наугад, то точность бы была 10%.  
  
Теперь посмотрим какие классы сеть определяет лучше:

class\_correct = list(0. for i in range(10))

class\_total = list(0. for i in range(10))

with torch.no\_grad():

for data in testloader:

images, labels = data

outputs = net(images)

\_, predicted = torch.max(outputs, 1)

c = (predicted == labels).squeeze()

for i in range(4):

label = labels[i]

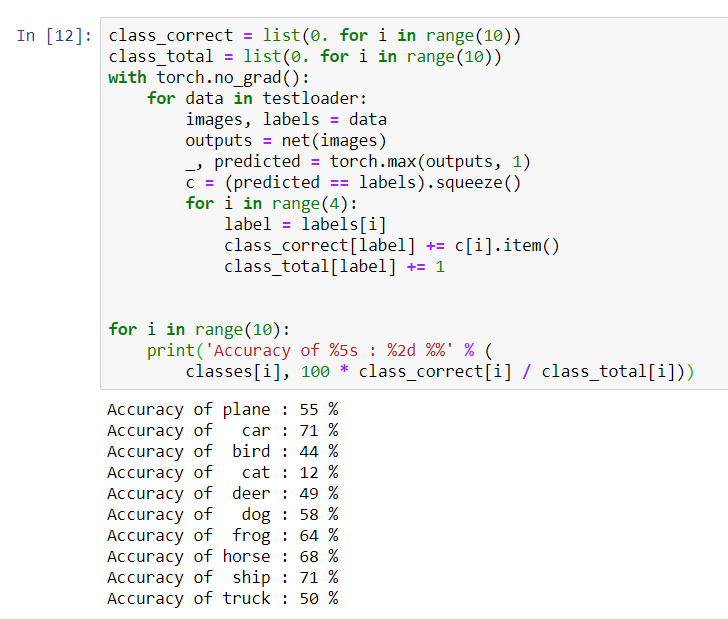
class\_correct[label] += c[i].item()

class\_total[label] += 1

for i in range(10):

print('Accuracy of %5s : %2d %%' % (

classes[i], 100 \* class\_correct[i] / class\_total[i]))

  
  
Похоже что лучше всего сеть определяет автомобили и корабли: 71% точности.  
  
Итак сеть работает. Теперь попробуем перенести ее работу на графический процессор (GPU) и посмотрим, что поменяется.

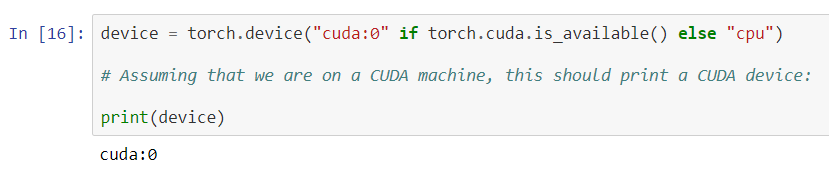
**Тренировка нейронной сети на GPU**

Сначала объясню коротко что такое CUDA. CUDA (Compute Unified Device Architecture) — платформа параллельных вычислений, разработанная NVIDIA, для общих вычислений на графических процессорах (GPU). С помощью CUDA разработчики могут значительно ускорить вычислительные приложения, используя возможности графических процессоров. На нашем сервере, который мы приобрели, данная платформа уже установлена.  
  
Давайте сначала определим наше GPU как первое видимое устройство cuda.

device = torch . device ( "cuda:0" if torch . cuda . is\_available () else "cpu" )

# Assuming that we are on a CUDA machine, this should print a CUDA device:

print ( device )

  
  
Отправляем сеть на GPU:

net.to(device)

Так же нам придется отправлять входы и цели на каждом шаге и в GPU:

inputs, labels = data[0].to(device), data[1].to(device)

Запустим повторное обучение сети уже на GPU:

import torch.optim as optim

criterion = nn.CrossEntropyLoss()

optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

for epoch in range(2): # loop over the dataset multiple times

running\_loss = 0.0

for i, data in enumerate(trainloader, 0):

# get the inputs; data is a list of [inputs, labels]

inputs, labels = data[0].to(device), data[1].to(device)

# zero the parameter gradients

optimizer.zero\_grad()

# forward + backward + optimize

outputs = net(inputs)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

# print statistics

running\_loss += loss.item()

if i % 2000 == 1999: # print every 2000 mini-batches

print('[%d, %5d] loss: %.3f' %

(epoch + 1, i + 1, running\_loss / 2000))

running\_loss = 0.0

print('Finished Training')

В этот раз обучение сети продолжалось по времени около 3 минут. Напомним, что тот же этап на обычном процессоре длился 5 минут. Разница не существенная, это происходит потому что наша сеть не такая большая. При использовании больших массивов для обучения разница между скоростью работы GPU и традиционного процессора буде возрастать.  
  
На этом кажется все. Что нам удалось сделать:

* Мы рассмотрели, что такое GPU и выбрали сервер на котором он установлен;
* Мы настроили программное окружение для создания нейронной сети;
* Мы создали нейронную сеть для распознавания изображений и обучили ее;
* Мы повторили обучение сети с использованием GPU и получили прирост в скорости.