Практическая работа №3: Классификация цветов с помощью свёрточных нейронных сетей.

Работа выполнена: Осининой Татьяной, R3237

Задание:

в работе необходимо познакомится с различными архитектурами сверхточных нейронных сетей и их обучением на GPU (англ. graphics processing, графический процессор) на языке программирования Python 3 и фреймворка Torch (PyTorch). Для этого предлагается использовать ресурсы Google Colab - Colaboratory, для выполнения вычислений на GPU. После с ознакомления, выполнить практическое задане в конце данной тетради (notebook).

Рассмотрим Датасет содержащий 4242 изображения цветов размеченных по 5 видам (тюльпан, ромашка, подсолнух, роза, одуванчик). Данный набор данных можно скачать по ссылке с сайте kaggle.

Загрузите папку с картинками на гугл диск, чтобы не загружать ее каждый раз заново при перезапуске колаба. Структура файлов (можно посмотреть в меню слева) может быть такой: "/content/drive/My Drive/data/flowers". Обязательно подключите аппаратный ускоритель (GPU) к среде выполнения, чтобы вычисления были. В меню сверху: Среда выполнения -> Сменить среду выполнения

Первым делом разберите более детально код выполнив код ниже.

Подготовка

```
#connect to disk
from google.colab import drive
drive.mount('/content/drive/')
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive
```

Загружаем библиотеки. Фиксируем random.seed для воспроизводимости

```
import numpy as np # linear algebra
import os
import torch
import torchvision
from torchvision.datasets.utils import download_url
from torch.utils.data import random_split
```

```
from torchvision.datasets import ImageFolder
from torchvision import transforms
from torchvision.transforms import ToTensor
from torch.utils.data.dataloader import DataLoader
import torch.nn as nn
import torch.nn.functional as F
import random
from tqdm import tqdm
import matplotlib.pyplot as plt

random.seed(0)
torch.manual_seed(0)
```

```
<torch._C.Generator at 0x7f9256ee84d0>
```

Выбираем на чем будем делать вычисления - CPU или GPU (cuda)

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print(device)
```

```
cuda
```

```
#connect the folder with images
from google.colab import drive

drive.mount('/content/drive', force_remount=True)

FOLDERNAME = '/MyDrive/flowers'

assert FOLDERNAME is not None, "[!] Enter the foldername."
#/content/drive/MyDrive/flowers
%cd drive/MyDrive
#%cp -r $FOLDERNAME ../../
#%cd ../../
%cd flowers
#!bash get_datasets.sh
#%cd ../../
```

```
Mounted at /content/drive
/content/drive/MyDrive
/content/drive/MyDrive/flowers
```

```
prepare_imgs = torchvision.transforms.Compose(
    [
        torchvision.transforms.Resize((224, 224)), #приводим картинки к одному размеру
        torchvision.transforms.ToTensor(), # упаковывем их в тензор
        torchvision.transforms.Normalize(
            mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225] # нормализуем картинки п
        ),
    ]
)
```

```
# задаем датасет. Лейблы - имена папок:
dataset = ImageFolder('/content/drive/MyDrive/flowers', transform=prepare_imgs)
```

```
dataset.imgs[2]
```

```
('/content/drive/MyDrive/flowers/daisy/10172379554_b296050f82_n.jpg', 0)
```

```
class ValueMeter(object):
  """
  Вспомогательный класс, чтобы отслеживать loss и метрику
  """
  def __init__(self):
      self.sum = 0
      self.total = 0

  def add(self, value, n):
      self.sum += value*n
      self.total += n

  def value(self):
      return self.sum/self.total

def log(mode, epoch, loss_meter, accuracy_meter, best_perf=None):
  """
  Вспомогательная функция, чтобы
  """
  print(
      f"[{mode}] Epoch: {epoch:0.2f}. "
      f"Loss: {loss_meter.value():.2f}. "
      f"Accuracy: {100*accuracy_meter.value():.2f}% ", end="\n")

  if best_perf:
      print(f"[best: {best_perf:0.2f}]%", end="")
```

Сверточная нейросеть с нуля

Вручную прописываем слои

```
model = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2), # output: 64 x 16 x 16

            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2), # output: 128 x 8 x 8
```

```
            nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2), # output: 256 x 4 x 4

            nn.Flatten(),
            nn.Linear(256*28*28, 1024),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Linear(512, 5))
model.to(device) # отправляем модель на девайс (GPU)
```

```
     Sequential(
       (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
       (1): ReLU()
       (2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
       (3): ReLU()
       (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
       (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
       (6): ReLU()
       (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
       (8): ReLU()
       (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
       (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
       (11): ReLU()
       (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
       (13): ReLU()
       (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
       (15): Flatten(start_dim=1, end_dim=-1)
       (16): Linear(in_features=200704, out_features=1024, bias=True)
       (17): ReLU()
       (18): Linear(in_features=1024, out_features=512, bias=True)
       (19): ReLU()
       (20): Linear(in_features=512, out_features=5, bias=True)
     )
```

Задаем гиперпараметры для обучения:

Задаем параметры и функцию для обучения. Разбиваем датасет на train/validation

```
batch_size = 32 # размер батча
optimizer = torch.optim.Adam(params = model.parameters()) # алгоритм оптимизации
lr = 0.001 # learning rate
```

Разбиваем датасет на train и validation

Задаем dataloader'ы - объекты для итеративной загрузки данных и лейблов для обучения

```
train_set, val_set = torch.utils.data.random_split(dataset, [len(dataset)-1000, 1000])
print('Размер обучающего и валидационного датасета: ', len(train_set), len(val_set))
loaders = {'training': DataLoader(train_set, batch_size, pin_memory=True,num_workers=2, sh
           'validation':DataLoader(val_set, batch_size, pin_memory=True,num_workers=2, shu
```

```
Размер обучающего и валидационного датасета:  3317 1000
```

Функция для подсчета Accuracy

```
def accuracy(outputs, labels):
    _, preds = torch.max(outputs, dim=1)
    return torch.tensor(torch.sum(preds == labels).item() / len(preds))
```

Функция для обучения и валидации модели

```
def trainval(model, loaders, optimizer, epochs=10):
    """
    model: модель, которую собираемся обучать
    loaders: dict с dataloader'ами для обучения и валидации
    optimizer: оптимизатор
    epochs: число обучающих эпох (сколько раз пройдемся по всему датасету)
    """
    loss_meter = {'training': ValueMeter(), 'validation': ValueMeter()}
    accuracy_meter = {'training': ValueMeter(), 'validation': ValueMeter()}

    loss_track = {'training': [], 'validation': []}
    accuracy_track = {'training': [], 'validation': []}

    for epoch in range(epochs): # итерации по эпохам
        for mode in ['training', 'validation']: # обучение - валидация
            # считаем градиаент только при обучении:
            with torch.set_grad_enabled(mode == 'training'):
                # в зависимоти от фазы переводим модель в нужный ружим:
                model.train() if mode == 'training' else model.eval()
                for imgs, labels in tqdm(loaders[mode]):
                    imgs = imgs.to(device) # отправляем тензор на GPU
                    labels = labels.to(device)
                    bs = labels.shape[0]  # размер батча (отличается для последнего батча

                    preds = model(imgs) # forward pass - прогоняем тензор с картинками чер
                    loss = F.cross_entropy(preds, labels) # считаем функцию потерь
                    acc = accuracy(preds, labels) # считаем метрику

                    # храним loss и accuracy для батча
                    loss_meter[mode].add(loss.item(), bs)
                    accuracy_meter[mode].add(acc, bs)

                    # если мы в фазе обучения
                    if mode == 'training':
```

```
                optimizer.zero_grad() # обнуляем прошлый градиент
                loss.backward() # делаем backward pass (считаем градиент)
                optimizer.step() # обновляем веса
        # в конце фазы выводим значения loss и accuracy
        log(mode, epoch, loss_meter[mode], accuracy_meter[mode])

        # сохраняем результаты по всем эпохам
        loss_track[mode].append(loss_meter[mode].value())
        accuracy_track[mode].append(accuracy_meter[mode].value())
    return loss_track, accuracy_track
```

Обучаем базовую модель

Проверим загрузку видеокарты, прежде чем запустить обучение:

```
!nvidia-smi
```

```
Fri Dec 17 09:19:13 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 495.44       Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:00:04.0 Off |                    0 |
| N/A   75C    P0    78W / 149W |   1321MiB / 11441MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Запускаем обучение на 10 эпох

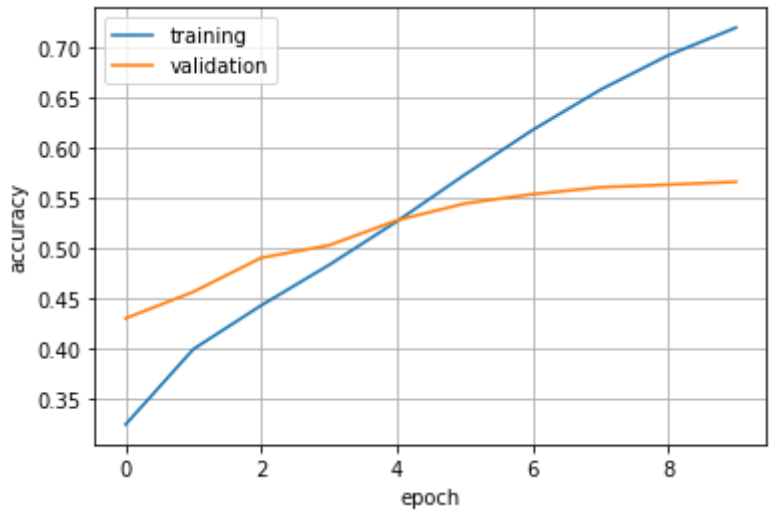
```
loss_track, accuracy_track = trainval(model, loaders, optimizer, epochs=10)
```

```
100%|██████████| 104/104 [01:17<00:00,  1.35it/s]
[training] Epoch: 0.00. Loss: 1.51. Accuracy: 32.44%
100%|██████████| 32/32 [00:10<00:00,  3.10it/s]
[validation] Epoch: 0.00. Loss: 1.27. Accuracy: 43.00%
100%|██████████| 104/104 [01:16<00:00,  1.36it/s]
[training] Epoch: 1.00. Loss: 1.36. Accuracy: 39.95%
100%|██████████| 32/32 [00:11<00:00,  2.91it/s]
[validation] Epoch: 1.00. Loss: 1.19. Accuracy: 45.65%
100%|██████████| 104/104 [01:16<00:00,  1.37it/s]
[training] Epoch: 2.00. Loss: 1.28. Accuracy: 44.30%
```

```
100%|██████████| 32/32 [00:10<00:00,  3.06it/s]
[validation] Epoch: 2.00. Loss: 1.15. Accuracy: 49.03%
100%|██████████| 104/104 [01:15<00:00,  1.37it/s]
[training] Epoch: 3.00. Loss: 1.21. Accuracy: 48.32%
100%|██████████| 32/32 [00:10<00:00,  2.98it/s]
[validation] Epoch: 3.00. Loss: 1.14. Accuracy: 50.27%
100%|██████████| 104/104 [01:16<00:00,  1.37it/s]
[training] Epoch: 4.00. Loss: 1.13. Accuracy: 52.66%
100%|██████████| 32/32 [00:10<00:00,  3.00it/s]
[validation] Epoch: 4.00. Loss: 1.12. Accuracy: 52.76%
100%|██████████| 104/104 [01:15<00:00,  1.37it/s]
[training] Epoch: 5.00. Loss: 1.03. Accuracy: 57.30%
100%|██████████| 32/32 [00:10<00:00,  3.03it/s]
[validation] Epoch: 5.00. Loss: 1.17. Accuracy: 54.43%
100%|██████████| 104/104 [01:16<00:00,  1.37it/s]
[training] Epoch: 6.00. Loss: 0.93. Accuracy: 61.75%
100%|██████████| 32/32 [00:10<00:00,  3.05it/s]
[validation] Epoch: 6.00. Loss: 1.24. Accuracy: 55.36%
100%|██████████| 104/104 [01:16<00:00,  1.36it/s]
[training] Epoch: 7.00. Loss: 0.84. Accuracy: 65.76%
100%|██████████| 32/32 [00:09<00:00,  3.23it/s]
[validation] Epoch: 7.00. Loss: 1.36. Accuracy: 56.05%
100%|██████████| 104/104 [01:16<00:00,  1.36it/s]
[training] Epoch: 8.00. Loss: 0.76. Accuracy: 69.19%
100%|██████████| 32/32 [00:10<00:00,  2.99it/s]
[validation] Epoch: 8.00. Loss: 1.55. Accuracy: 56.31%
100%|██████████| 104/104 [01:16<00:00,  1.37it/s]
[training] Epoch: 9.00. Loss: 0.69. Accuracy: 71.95%
100%|██████████| 32/32 [00:10<00:00,  3.09it/s][validation] Epoch: 9.00. Loss: 1.75.
```

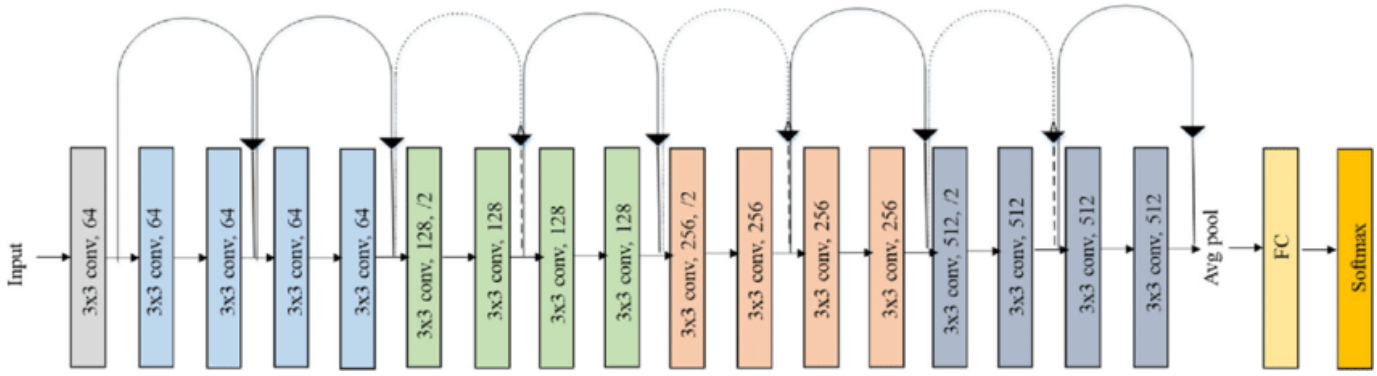
```
from matplotlib import pyplot as plt
%matplotlib inline
plt.plot(accuracy_track['training'], label='training')
plt.plot(accuracy_track['validation'], label='validation')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.grid()
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f9251313e10>
```



Fine-tuning предобученной модели

Теперь попробуем поработать с предобученной сетью ResNet-18



Практическое задание

В пракическом задание необходимо обучить еще одну сверточную архитектуру для задач классификации цветов.

В выбранной Вами архитектуре также необходимо разобраться с основными её параметрами и принципами работы.

Посмотрите как использовать модели в PyTorch, выберите одну и используя transfer learning до-обучите модель на классификацию цветов. Чтобы это сделать замените ____ в ячейках ниже на работающий код.

```
# Выберите модель из списка доступных в PyTorch моделей
# Не забудьте указать, что она модель должна быть предобучена!
import torchvision.models as models
model = models.googlenet(pretrained=True)
```

```
def set_parameter_requires_grad(model):
  """
  Функция для заморозки весов модели
  """
  for param in model.parameters():
    param.requires_grad = False
```

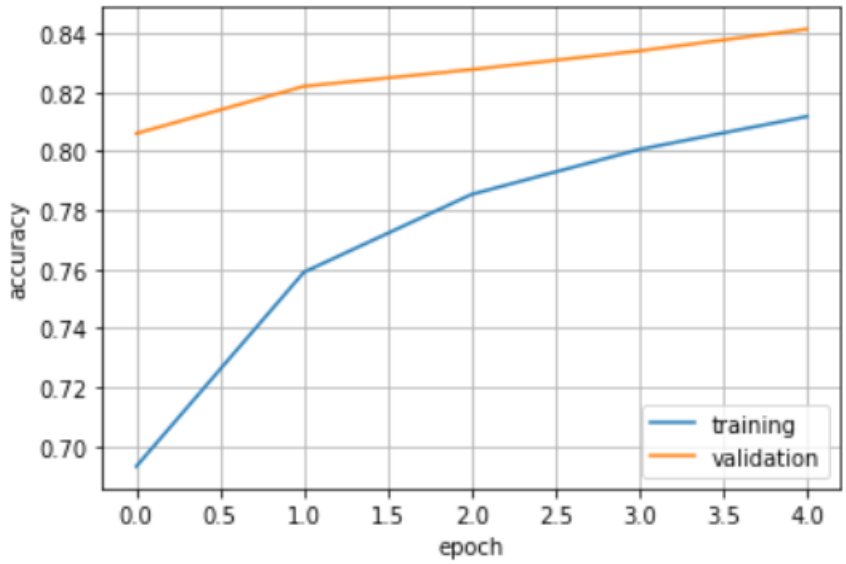
```
set_parameter_requires_grad(model) # передайте модель в функцию для "заморозки" градиента
```

```
model.fc = nn.Linear(1024, 5)# Меняем последний слой модели


# Проверим все ли сработало правильно, выведем веса, которые будут обучаться
for name, param in model.named_parameters():
    if param.requires_grad:
        print(name)

 fc.weight
 fc.bias


model.to(device) # Отправляем модель на GPU
optimizer = torch.optim.Adam(params = model.parameters()) # алгоритм оптимизации
loss_track, accuracy_track = trainval(model, loaders, optimizer, epochs=5) # запускаем обу
```

```
100%|██████████| 104/104 [00:22<00:00,  4.64it/s]
[training] Epoch: 0.00. Loss: 0.95. Accuracy: 69.31%
100%|██████████| 32/32 [00:08<00:00,  3.92it/s]
[validation] Epoch: 0.00. Loss: 0.63. Accuracy: 80.60%
100%|██████████| 104/104 [00:22<00:00,  4.62it/s]
[training] Epoch: 1.00. Loss: 0.75. Accuracy: 75.90%
100%|██████████| 32/32 [00:07<00:00,  4.40it/s]
[validation] Epoch: 1.00. Loss: 0.55. Accuracy: 82.20%
100%|██████████| 104/104 [00:22<00:00,  4.55it/s]
[training] Epoch: 2.00. Loss: 0.66. Accuracy: 78.53%
100%|██████████| 32/32 [00:07<00:00,  4.50it/s]
[validation] Epoch: 2.00. Loss: 0.52. Accuracy: 82.77%
100%|██████████| 104/104 [00:23<00:00,  4.51it/s]
[training] Epoch: 3.00. Loss: 0.61. Accuracy: 80.06%
100%|██████████| 32/32 [00:06<00:00,  4.59it/s]
[validation] Epoch: 3.00. Loss: 0.49. Accuracy: 83.40%
100%|██████████| 104/104 [00:22<00:00,  4.60it/s]
[training] Epoch: 4.00. Loss: 0.57. Accuracy: 81.18%
100%|██████████| 32/32 [00:07<00:00,  4.53it/s][validation] Epoch: 4.00. Loss: 0.47.
```

```
plt.plot(accuracy_track['training'], label='training')
plt.plot(accuracy_track['validation'], label='validation')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.grid()
plt.legend()
```

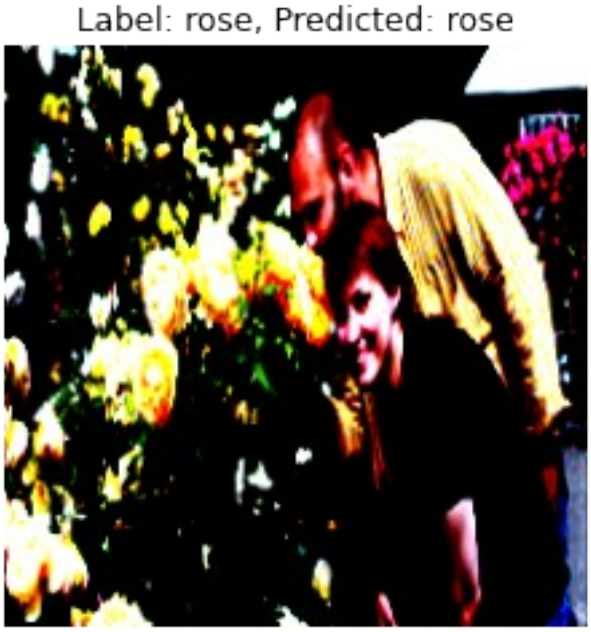
```
<matplotlib.legend.Legend at 0x7f923ca60e10>
```




```
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

def predict_image(img, model):
    # Преобразование to a batch of 1
    xb = img.unsqueeze(0).to(device)
    # Получение прогнозов от модели
    yb = model(xb)
    # Выбираем индекс с наибольшей вероятностью
    model, preds  = torch.max(yb, dim=1)
    # Получение метки класса
    return dataset.classes[preds[0].item()]

for i in range(1,15):
  img, label = val_set[i]
  plt.imshow(img.clip(0,1).permute(1, 2, 0))
  plt.axis('off')
  plt.title('Label: {}, Predicted: {}'.format(dataset.classes[label],predi
ct_image(img, model)))
  plt.show
  print('Label:', dataset.classes[label], ',Predicted:', predict_image(img
, model))
```

```
Label: rose ,Predicted: dandelion
Label: rose ,Predicted: tulip
Label: dandelion ,Predicted: dandelion
Label: sunflower ,Predicted: sunflower
Label: tulip ,Predicted: tulip
Label: rose ,Predicted: rose
Label: rose ,Predicted: tulip
Label: sunflower ,Predicted: sunflower
```

```
Label: rose ,Predicted: rose
Label: sunflower ,Predicted: sunflower
Label: tulip ,Predicted: tulip
Label: daisy ,Predicted: daisy
Label: tulip ,Predicted: tulip
Label: rose ,Predicted: rose
```




```
Сохраняем веса модели:
weights_fname = '/content/drive/MyDrive/COLAB/weights_fname.pth'
torch.save(model.state_dict(), weights_fname)
```

Вопросы.

Как работает выбранная вами модель сверточной нейронной сети? Какие параметры?

В чем основные отличия между сверточной нейронной сетью и "обычной" полносвязной нейронной сетью?

Что такое transfer learning?

Что такое функция для заморозки весов модели?