# Использование Keras для обучения нейронной сети с собственным набором изображений в Google Colab

2020
Обнинск

# Содержание

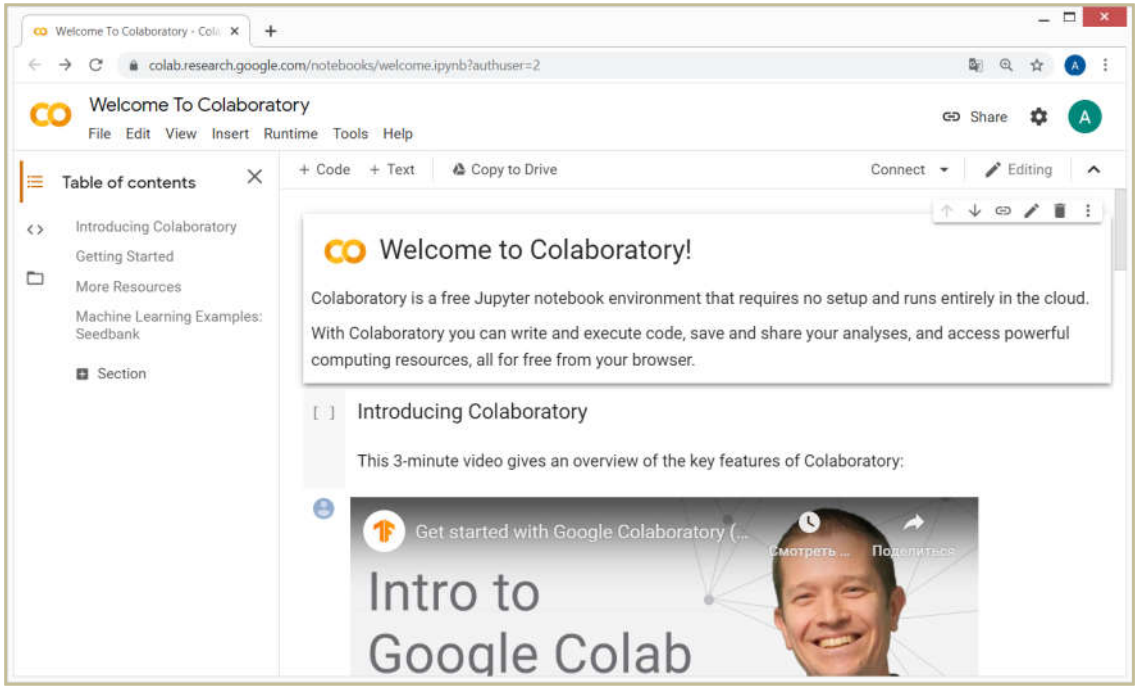
- Знакомство с Google Colab
- Описание задачи
- Создание набора данных
- Создание, обучение и оценка нейронной сети
- Создание, обучение и оценка свёрточной нейронной сети

TensorFlow

# Google Colab

- Google Colab – облачный сервис для исследований в области глубокого обучения

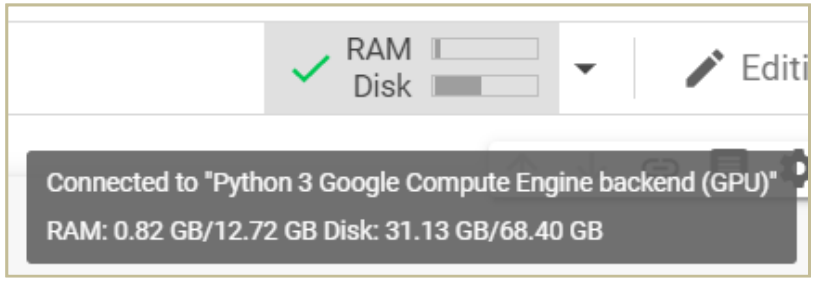
https://colab.research.google.com/notebooks/welcome.ipynb



# Google Colab

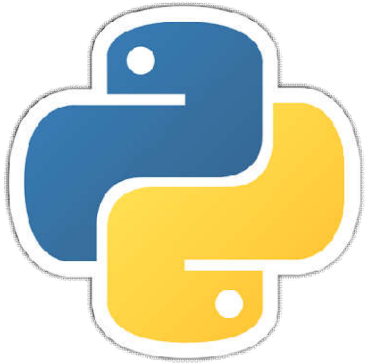
Предоставляемые возможности:

- машина с ОС Ubuntu 18.04 и графическим процессором Tesla K80
- сессия длительностью 12 часов
- 12.72 Гб оперативной памяти
- объем дискового пространства зависит от загруженности сервиса



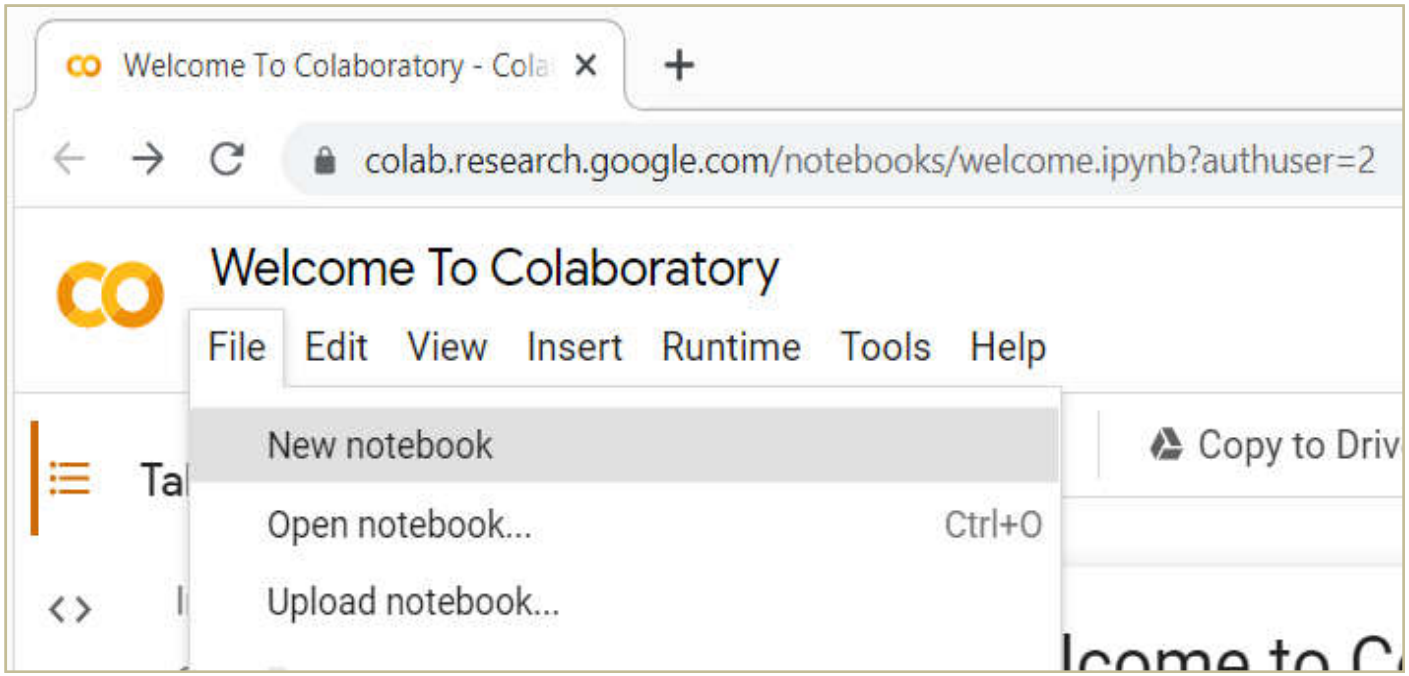
# Google Colab

- для работы с Colab оптимальным браузером является Google Chrome, однако он работает с большинством основных браузеров
- поддерживаются версии Python 2.7 и 3.6
- нужные пакеты устанавливаются через pip или apt-get



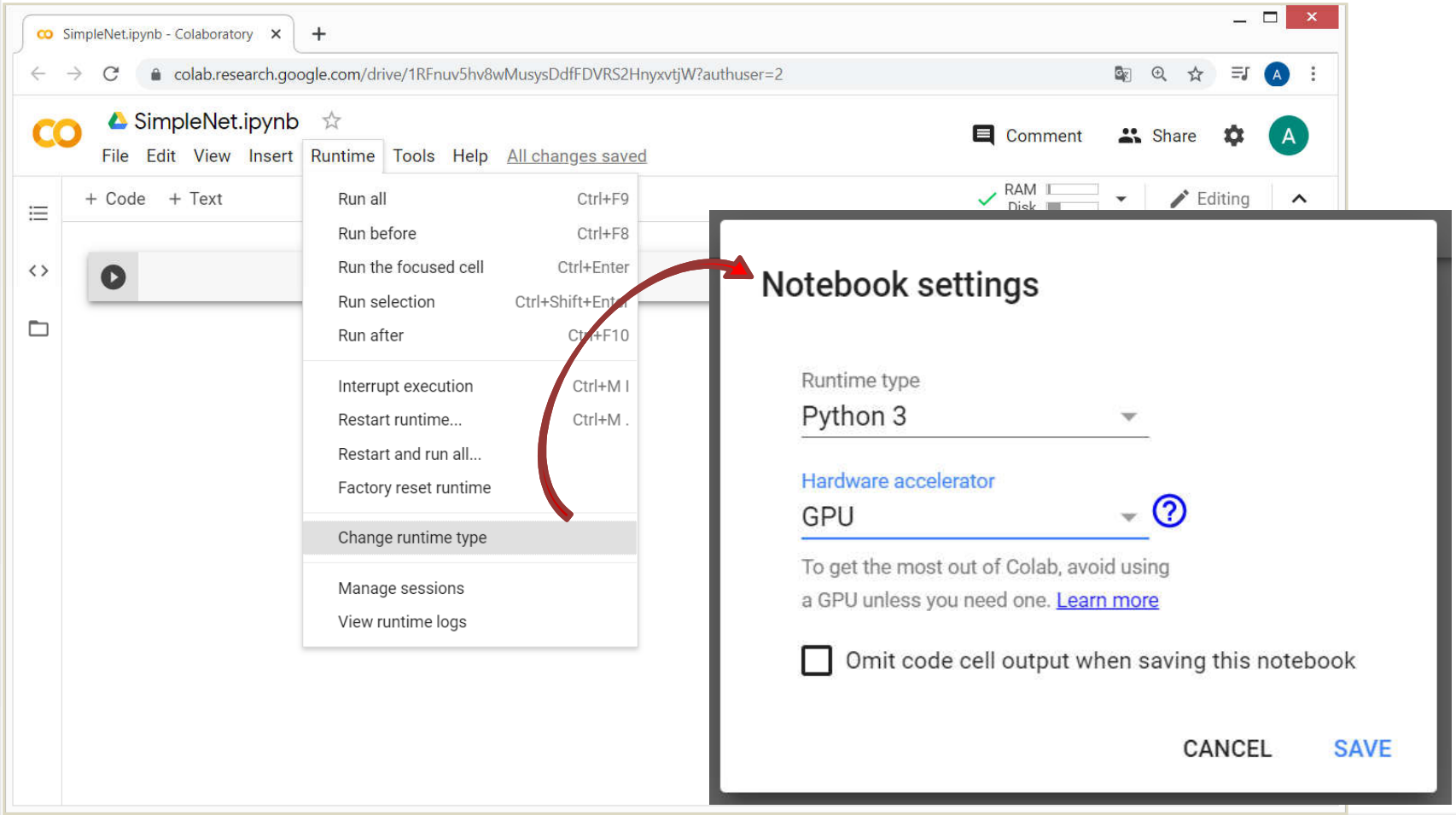
# Google Colab

Можно сказать, что Colab является аналогом гугл-документов для Jupyter Notebook. Для начала работы нужно создать новый блокнот.



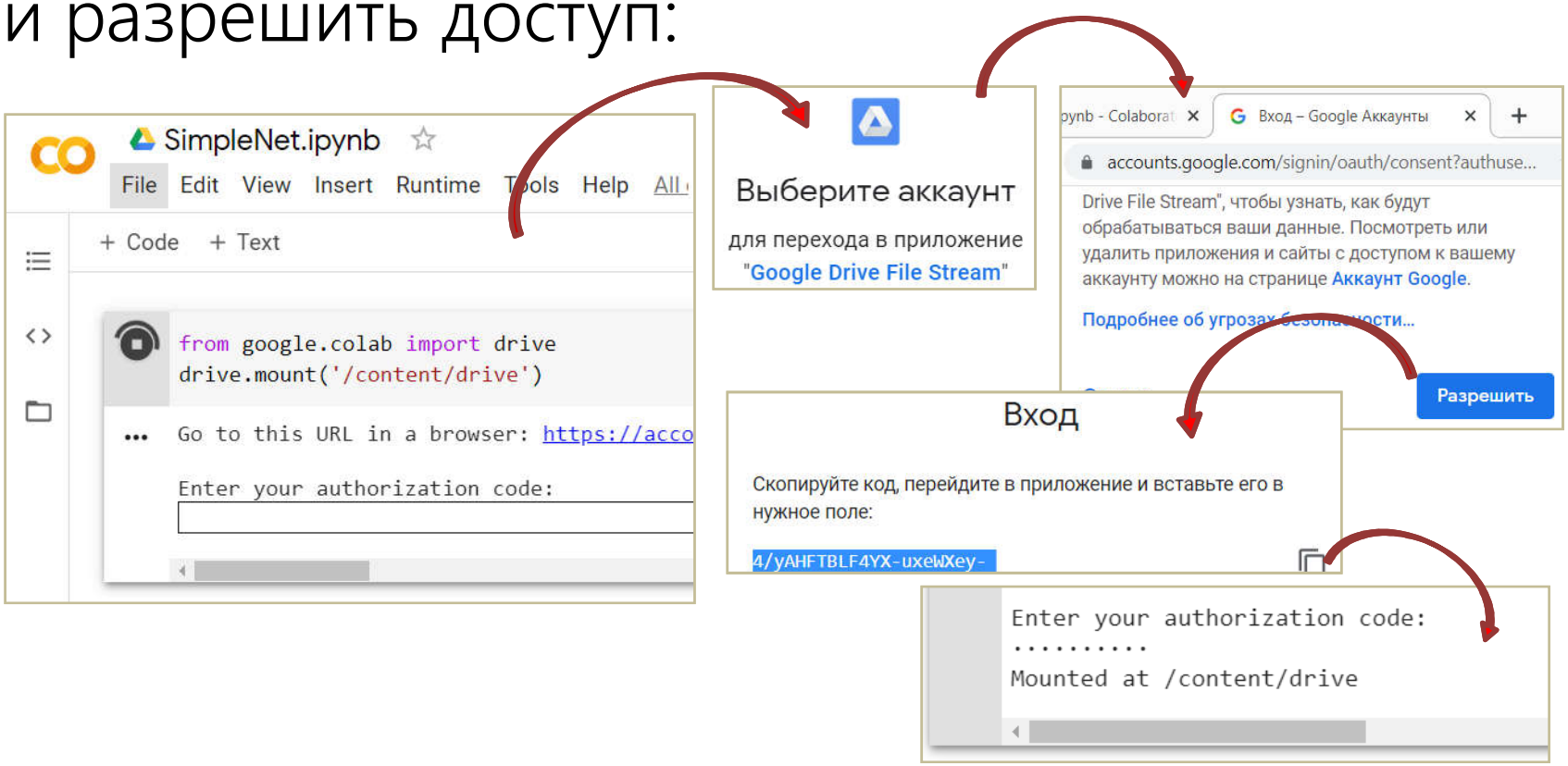
# Google Colab

## Затем необходимо включить GPU



# Google Colab

Предоставляется возможность монтирования Google Drive. Нужно написать две строки кода и разрешить доступ:



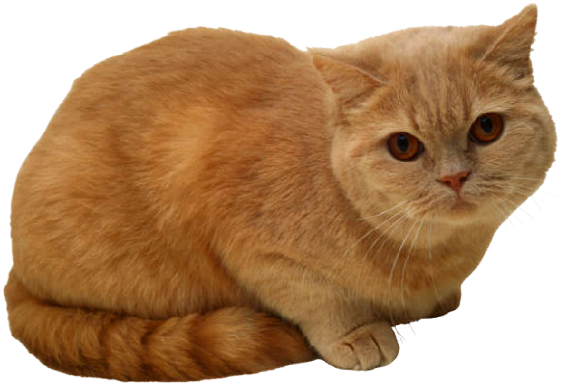
# Google Colab

- Итак, теперь можно использовать предоставленные вычислительные ресурсы для решения задач.

- Важно помнить, что данные текущей сессии удаляются по истечении 12 часов, поэтому необходимо сохранять их на локальную машину или на Google Drive.

# Описание задачи

Решаемая задача формулируется следующим образом: «необходимо классифицировать входное изображение как содержащее кота, собаку или панду».



# Прежде чем начать

- animals.zip (архив с фото для обучения)

https://drive.google.com/file/d/1Gvf02l7FpGGVUP_3Mb0UDvXJ9SR5x11P/view?usp=sharing

- images.zip (архив с тестовыми фото)

https://drive.google.com/file/d/1fAt0lWJu83ba-fgc0bWx6joNuKcCuQc-/view?usp=sharing

- SimpleNet.ipynb (блокнот, содержащий код)

https://colab.research.google.com/drive/1RFnuv5hv8wMusysDdfFDVRS2HnyxvtjW

*animals.zip и images.zip нужно положить в корень Google Drive!*

# Работа с набором данных

При решении задач компьютерного зрения одной из основных проблем является поиск или создание набора обучающих данных.

Для решения задачи будет использован готовый набор, содержащий 3000 изображений (по 1000 каждого класса).

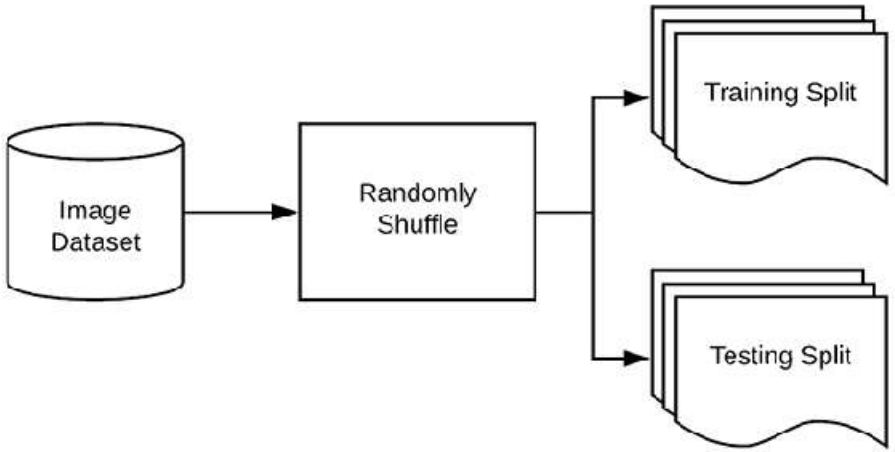
Для составления собственных наборов можно использовать метод, приведенный здесь:

https://www.pyimagesearch.com/2017/12/04/how-to-create-a-deep-learning-dataset-using-google-images/

# Работа с набором данных

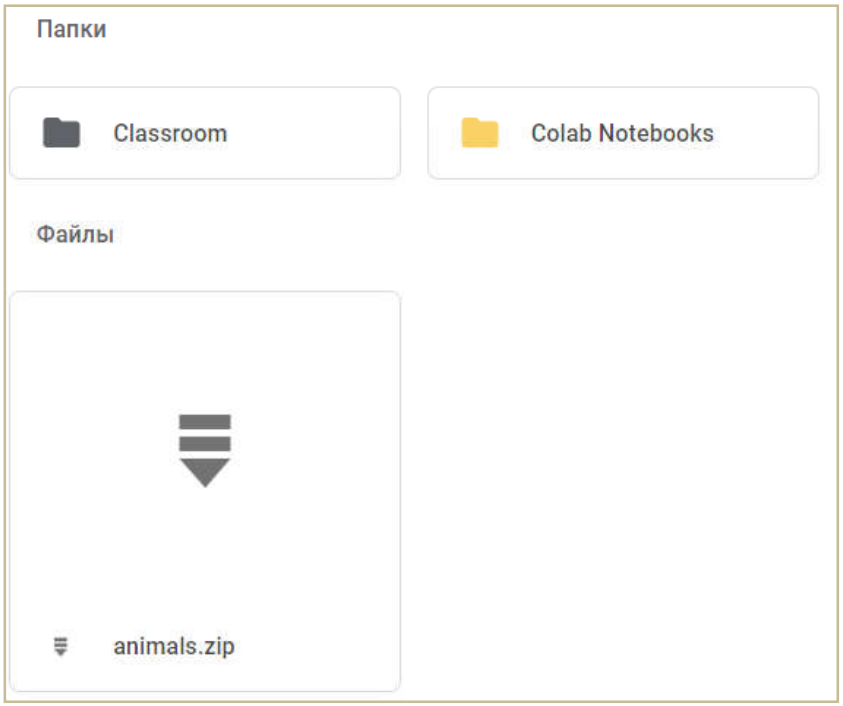
С датасетом будет необходимо проделать следующие действия:

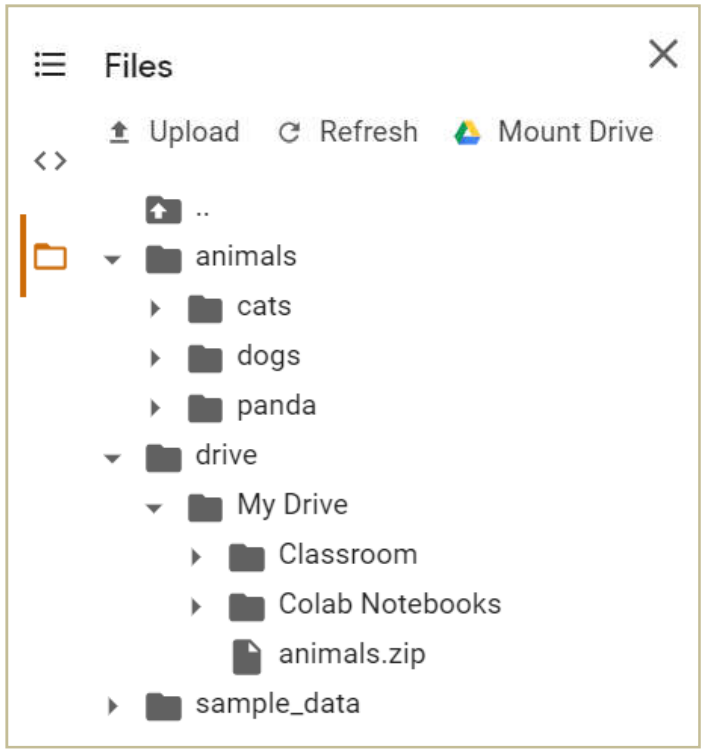
- упорядочить свой набор изображений на диске (для датасета animals это уже сделано)
- загрузить изображения и метки класса с диска
- разделить данные обучающую и тестовую выборки



# Работа с набором данных

В разделе «Работа с данными» блокнота SimpleNet.ipynb датасет загружается на удаленную машину с Google Drive



# Работа с набором данных

Далее будут прокомментированы основные фрагменты кода SimpleNet.ipynb (раздел «Создание и обучение простой нейронной сети»).

Сначала инициализируются списки для данных (data) и меток (labels). Позже это будут массивы NumPy.

```
# инициализируем данные и метки
data = []
labels = []
```

# Работа с набором данных

- dataset: путь к набору изображений на диске
- model: путь к выходному файлу модели
- label_bin: путь к выходному бинаризованному файлу метки
- plot: путь к выходному файлу графика обучения

```
dataset = "/content/animals"
model_ = "/content/output/simple_nn.model"
label_bin = "/content/output/simple_nn_lb.pickle"
plot = "/content/output/simple_nn_plot.png"
```

# Работа с набором данных

Случайным образом перемешиваются изображения, пути к которым содержатся в imagePaths.

Функция paths.list_images найдёт пути ко всем входным изображениям в каталоге датасета перед тем, как они будут отсортированы и перемешаны.

```
# берём пути к изображениям и рандомно перемешиваем
imagePaths = sorted(list(paths.list_images(dataset)))
random.seed(42)
random.shuffle(imagePaths)
```

# Работа с набором данных

Далее для каждого из imagePath:

а) изображение image загружается в память;

б) его размер изменяется, затем происходит сглаживание; сглаживание данных позволяет легко передавать необработанные интенсивности пикселей в нейроны входного слоя (для VGGNet в сеть будут передаваться сразу все данные, поскольку она является свёрточной);

```
# цикл по изображениям
for imagePath in imagePaths:
  # загружаем изображение, меняем размер на 32x32 пикселей (без учета
  # соотношения сторон), сглаживаем его в 32x32x3=3072 пикселей и
  # добавляем в список
  image = cv2.imread(imagePath)
  image = cv2.resize(image, (32, 32)).flatten()
```

# Работа с набором данных

в) изменённое изображение добавляется к массиву данных;

г) метка класса изображения извлекается из его пути и добавляется к остальным меткам (список меток содержит классы, соответствующие каждому изображению в массиве данных)

```
    data.append(image)

    # извлекаем метку класса из пути к изображению и обновляем
    # список меток
    label = imagePath.split(os.path.sep)[-2]
    labels.append(label)
```

# Создание обучающей и тестовой выборок

Далее данные делятся на обучающую и тестовую выборки.

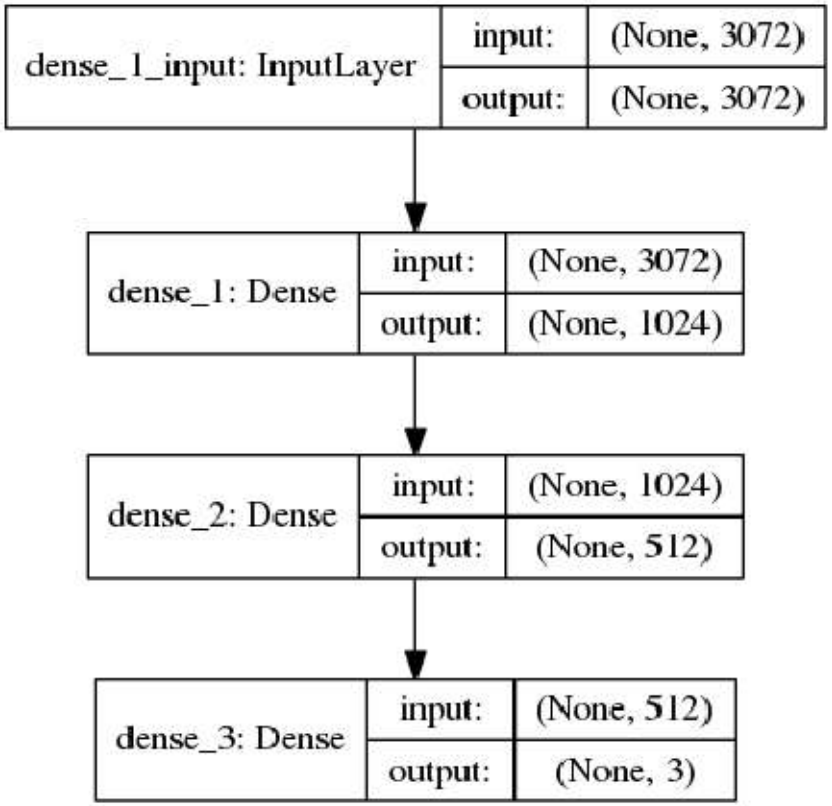
Scikit-learn предоставляет удобную функцию train_test_split, которая разделяет данные.

trainX и testX – это изображения, а trainY и testY – соответствующие метки.

```
# разбиваем данные на обучающую и тестовую выборки, используя 75%
# данных для обучения и оставшиеся 25% для тестирования
(trainX, testX, trainY, testY) = train_test_split(data,
  labels, test_size=0.25, random_state=42)
```

# Определение архитектуры модели

Будет создана сеть с одним входным слоем, одним выходным и двумя скрытыми:



# Определение архитектуры модели

input_shape равен 3072, т.к. во входном изображении 32x32x3=3072 пикселей.

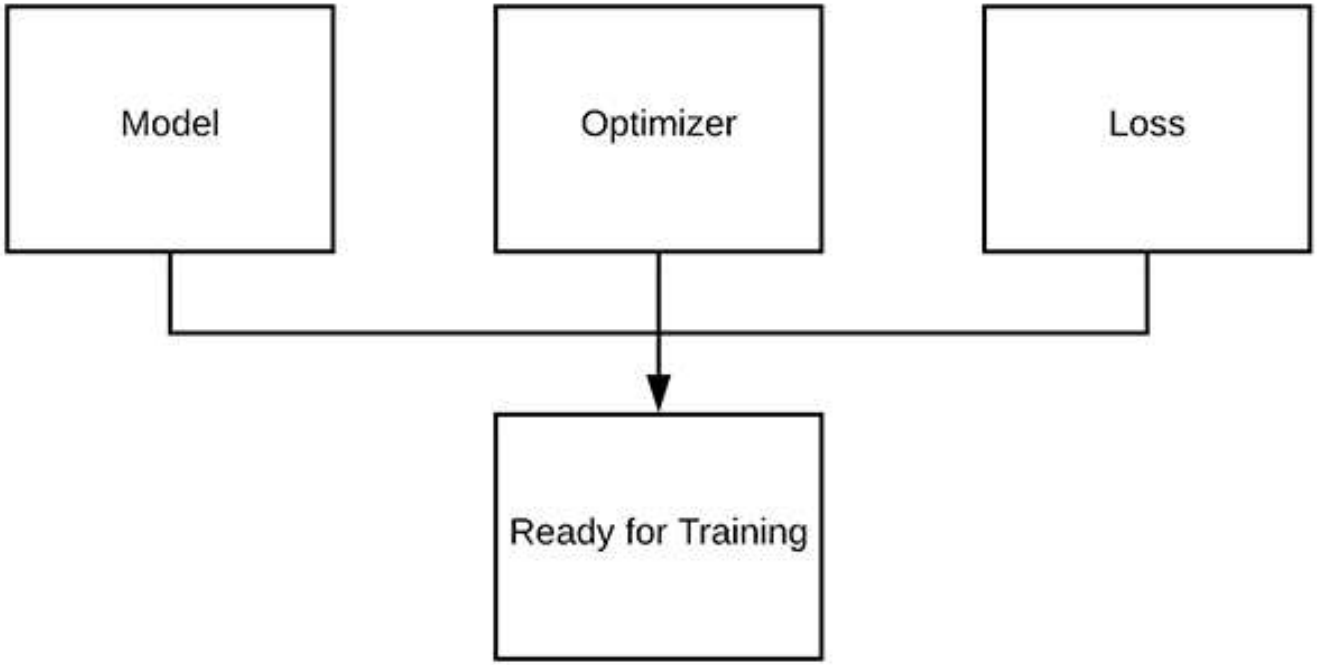
Первый скрытый слой имеет 1024 узла, второй – 512 узлов.

Количество узлов выходного слоя равно числу возможных меток классов

```
# определим архитектуру 3072-1024-512-3 с помощью Keras
model = Sequential()
model.add(Dense(1024, input_shape=(3072,), activation="sigmoid"))
model.add(Dense(512, activation="sigmoid"))
model.add(Dense(len(lb.classes_), activation="softmax"))
```

# Компиляция модели

После определения архитектуры нейронной сети необходимо скомпилировать её.



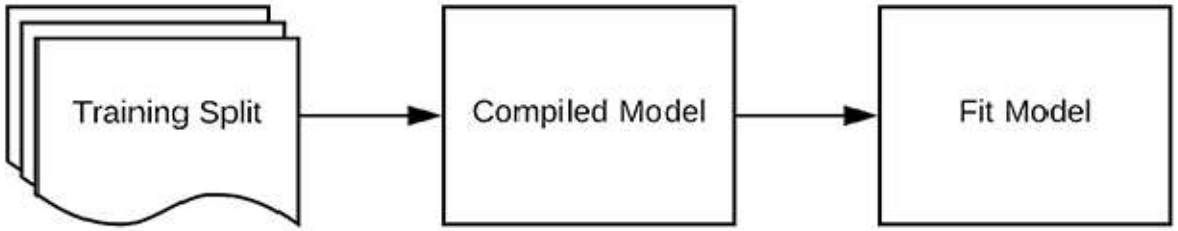
# Компиляция модели

Модель компилируется с использованием метода стохастического градиентного спуска (SGD) и категориальной кросс-энтропии. Категориальная кросс-энтропия используется почти для всех нейросетей, обученных выполнять классификацию. Исключение – когда имеется только два класса (в этом случае используется бинарная кросс-энтропия).

```
# компилируем модель, используя SGD как оптимизатор и категориальную
# кросс-энтропию в качестве функции потерь (для бинарной классификации
# следует использовать binary_crossentropy)
opt = SGD(lr=INIT_LR)
model.compile(loss="categorical_crossentropy", optimizer=opt,
  metrics=["accuracy"])
```

# Обучение модели

Далее необходимо обучить модель.




Параметр batch_size контролирует размер каждой группы данных для передачи. Мощные GPU могут обрабатывать большие пакеты, но рекомендуется отталкиваться от 32 и 64.

```
# обучаем нейросеть
H = model.fit(trainX, trainY, validation_data=(testX, testY),
  epochs=EPOCHS, batch_size=32)
```

# Оценка модели

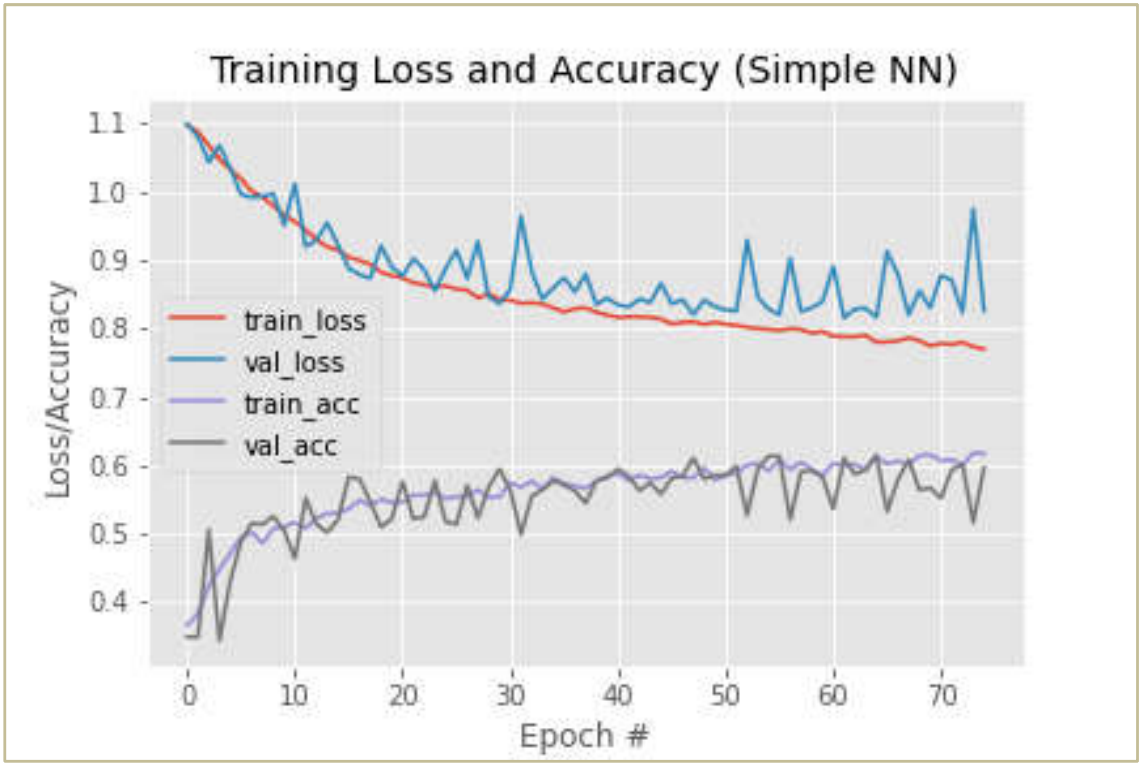
Далее модель оценивается с помощью тестовой выборки. Для оценки модели Keras используется комбинация методов .predict и classification_report из scikit-learn.

```
# оцениваем нейросеть
predictions = model.predict(testX, batch_size=32)
print(classification_report(testY.argmax(axis=1),
  predictions.argmax(axis=1), target_names=lb.classes_))
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| cats | 0.56 | 0.50 | 0.53 | 236 |
| dogs | 0.46 | 0.52 | 0.49 | 236 |
| panda | 0.75 | 0.75 | 0.75 | 278 |
| accuracy | | | 0.60 | 750 |
| macro avg | 0.59 | 0.59 | 0.59 | 750 |
| weighted avg | 0.60 | 0.60 | 0.60 | 750 |

# Оценка модели

Построены графики потерь при обучении, потерь при оценке, точности обучения, точности оценивания.



# Сохранение модели

Обученная модель сохраняется на диск, потом её можно будет использовать, не занимаясь обучением снова.

```
# сохраняем модель и бинаризатор меток на диск
model.save(model_)
f = open(label_bin, "wb")
f.write(pickle.dumps(lb))
f.close()
```

# Распознавание с использованием обученной модели

- image – путь к входному изображению
- model – путь к нашей обученной и сериализованной модели
- label_bin – путь к бинаризатору меток
- width – ширина изображения для модели; тут необходимо указать ширину, для которой предназначена модель
- height – высота входного изображения; также должна соответствовать конкретной модели
- flatten – надо ли сглаживать изображение

# Распознавание с использованием обученной модели

Модель и бинаризатор меток загружаются в память, после чего производится распознавание.

```
# загружаем модель и бинаризатор меток
model = load_model(model_)
lb = pickle.loads(open(label_bin, "rb").read())
```

```
# делаем предсказание на изображении
preds = model.predict(image)
print(preds)
```

```
[[0.00831385 0.01194306 0.97974306]]
```

# Распознавание с использованием обученной модели

В массиве предсказаний находится наибольшее значение, на изображении рисуется метка соответствующего класса.

```
# находим индекс метки класса с наибольшей вероятностью
# соответствия
i = preds.argmax(axis=1)[0]
label = lb.classes_[i]

# рисуем метку класса + вероятность на выходном изображении
text = "{}: {:.2f}%".format(label, preds[0][i] * 100)
cv2.putText(output, text, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
  (0, 0, 255), 2)
```

# Распознавание с использованием обученной модели

Для изображения panda.jpg:

```
# показываем выходное изображение
from google.colab.patches import cv2_imshow
cv2_imshow(output)
```

# Создание и обучение свёрточной нейронной сети

Использование стандартной нейронной сети прямого распространения для классификации изображений – не лучшее решение. Вместо этого разумнее использовать свёрточные нейронные сети (CNN), предназначенные для работы с интенсивностями пикселей и изучения различающих фильтров (их использование позволяет классифицировать изображения с высокой точностью).

В разделе «Создание и обучение свёрточной нейронной сети» используется уменьшенный вариант VGGNet. VGGNet-подобные модели имеют общие особенности:

- используются только свёрточные фильтры 3x3;
- свёрточные слои (“convolution layers”) чередуются со слоями подвыборки (“pooling layers”).

# Создание и обучение свёрточной нейронной сети

Далее определяется класс SmallVGGNet. Для сборки (build) требуется 4 параметра: ширина входных изображений (width), высота (height), глубина (depth) и число классов (classes). Т.к. используются RGB-изображения, то при вызове build передается depth = 3.

Сначала инициализируется последовательная (Sequential) модель. Затем определяется порядок каналов.

```
class SmallVGGNet:
  @staticmethod
  def build(width, height, depth, classes):
    # инициализируем модель и размер входного изображения
    # для порядка каналов “channel_last” и размер канала
    model = Sequential()
    inputShape = (height, width, depth)
    chanDim = -1
```

# Создание и обучение свёрточной нейронной сети

В этом блоке добавляются слои CONV=>RELU=>POOL.

Первый слой CONV имеет 32 фильтра размером 3x3.

В этой архитектуре сети используется функция активации ReLU (Rectified Linear Unit). Также используются: пакетная нормализация (Batch Normalization), функция максимума (MaxPooling) и метод исключения (Dropout).

```
# слои CONV => RELU => POOL
model.add(Conv2D(32, (3, 3), padding="same",
  input_shape=inputShape))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

# Создание и обучение свёрточной нейронной сети

Добавляются следующие слои. Размеры фильтра остаются прежними (3x3), а общее число фильтров увеличивается с 32 до 64.

```
# слои (CONV => RELU) * 2 => POOL
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

# Создание и обучение свёрточной нейронной сети

Добавляются следующие слои. Число фильтров удвоилось с 64 до 128, а размер остался прежним. Увеличение общего количества фильтров при уменьшении размера входных данных в CNN — обычная практика.

```
# (CONV => RELU) * 3 => POOL layer set
model.add(Conv2D(128, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(128, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(128, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

# Создание и обучение свёрточной нейронной сети

Полностью связанные слои в Keras обозначаются как Dense. Последний слой соединён с тремя выходами (так как в наборе данных три класса). Слой softmax возвращает вероятность принадлежности к определённому классу для каждой метки.

```
    # первый (и единственный) набор слоев FC => RELU
    model.add(Flatten())
    model.add(Dense(512))
    model.add(Activation("relu"))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))

    # классификатор softmax
    model.add(Dense(classes))
    model.add(Activation("softmax"))

    # возвращаем собранную архитектуру нейронной сети
    return model
```

# Создание и обучение свёрточной нейронной сети

Далее нейросеть SmallVGGNet обучается на наборе данных Animals. Процесс почти не отличается от предыдущего примера.

```
# обучаем нейросеть
H = model.fit_generator(aug.flow(trainX, trainY, batch_size=BS),
  validation_data=(testX, testY), steps_per_epoch=len(trainX) // BS,
  epochs=EPOCHS)
```

# Оценка свёрточной нейронной сети

Далее модель оценивается с помощью тестовой выборки. Для оценки модели Keras используется комбинация методов .predict и classification_report из scikit-learn.

```
# оцениваем нейросеть
predictions = model.predict(testX, batch_size=32)
print(classification_report(testY.argmax(axis=1),
  predictions.argmax(axis=1), target_names=lb.classes_))
```

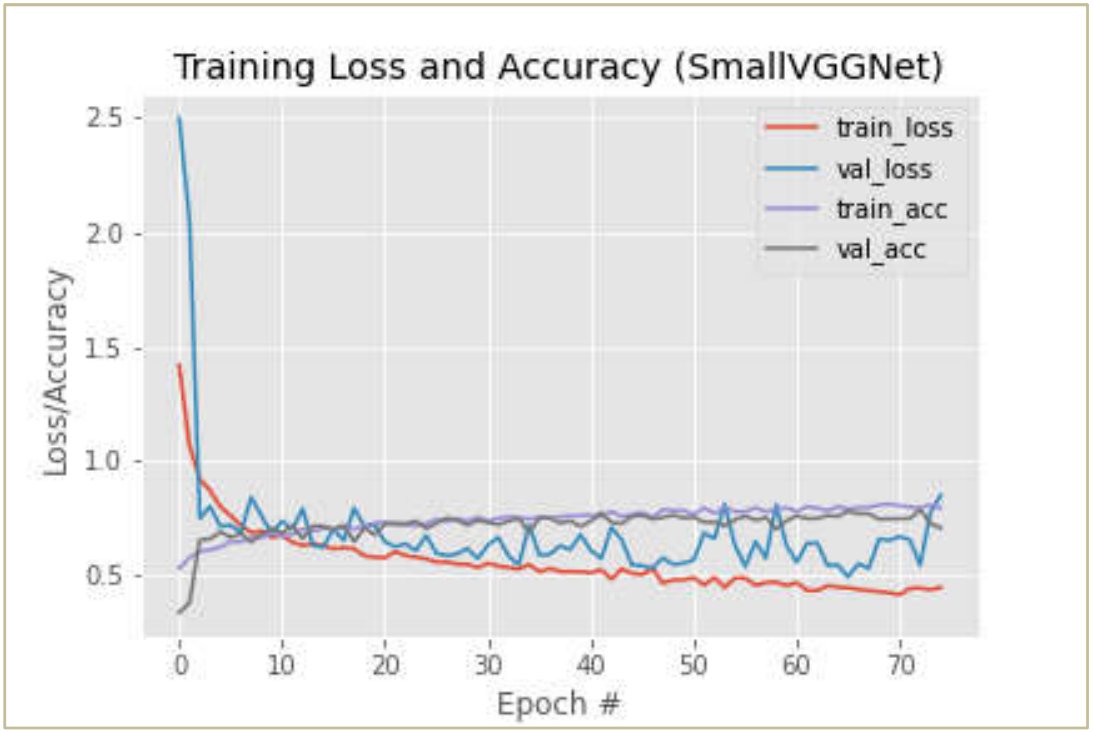
```
              precision    recall  f1-score   support

        cats       0.71      0.63      0.67       254
        dogs       0.70      0.58      0.64       249
       panda       0.77      1.00      0.87       247

    accuracy                           0.73       750
   macro avg       0.73      0.74      0.72       750
weighted avg       0.73      0.73      0.72       750
```

# Оценка свёрточной нейронной сети

Построены графики потерь при обучении, потерь при оценке, точности обучения, точности оценивания.

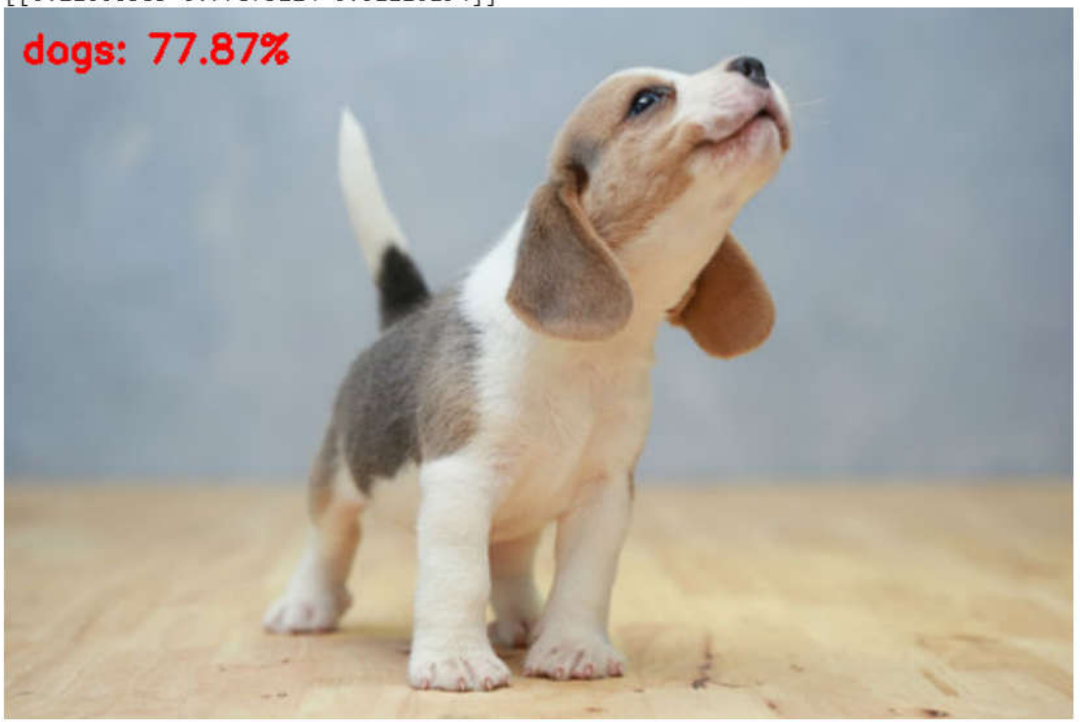


# Распознавание с использованием свёрточной нейросети

Для изображения dog.jpg:

```
# показываем выходное изображение
from google.colab.patches import cv2_imshow
cv2_imshow(output)
```

```
[[0.21006583 0.77873224 0.01120194]]
```



# Итоги

Как можно увидеть из полученных оценок, с использованием свёрточной нейронной сети на наборе изображений Animals достигнута точность в 73%. Это выше, чем предыдущее значение 60%.