```
try:
  # Colab only
  %tensorflow_version 2.x
except Exception:
    pass

from __future__ import absolute_import, division, print_function, unicode_literals

# TensorFlow и tf.keras
import tensorflow as tf
from tensorflow import keras

# Вспомогательные библиотеки
import numpy as np
import matplotlib.pyplot as plt

print(tf.__version__)
# These are all the modules we'll be using later. Make sure you can import them
# before proceeding further.
from __future__ import print_function
import matplotlib.pyplot as plt
import numpy as np
import os
import sys
import tarfile
from IPython.display import display, Image
from scipy import ndimage
from sklearn.linear_model import LogisticRegression
from six.moves.urllib.request import urlretrieve
from six.moves import cPickle as pickle

# Config the matlotlib backend as plotting inline in IPython
%matplotlib inline
```

```
TensorFlow 2.x selected.
2.1.0
```

```
url = 'http://commondatastorage.googleapis.com/books1000/'
last_percent_reported = None

def download_progress_hook(count, blockSize, totalSize):
  """A hook to report the progress of a download. This is mostly intended for users with
  slow internet connections. Reports every 1% change in download progress.
  """
  global last_percent_reported
  percent = int(count * blockSize * 100 / totalSize)

  if last_percent_reported != percent:
    if percent % 5 == 0:
      sys.stdout.write("%s%%" % percent)
```

```
      sys.stdout.flush()
    else:
      sys.stdout.write(".")
      sys.stdout.flush()

    last_percent_reported = percent

def maybe_download(filename, expected_bytes, force=False):
  """Download a file if not present, and make sure it's the right size."""
  if force or not os.path.exists(filename):
    print('Attempting to download:', filename)
    filename, _ = urlretrieve(url + filename, filename, reporthook=download_progress_hoo
    print('\nDownload Complete!')
  statinfo = os.stat(filename)
  if statinfo.st_size == expected_bytes:
    print('Found and verified', filename)
  else:
    raise Exception(
      'Failed to verify ' + filename + '. Can you get to it with a browser?')
  return filename

train_filename = maybe_download('notMNIST_large.tar.gz', 247336696)
test_filename = maybe_download('notMNIST_small.tar.gz', 8458043)
```

```
Attempting to download: notMNIST_large.tar.gz
0%....5%....10%....15%....20%....25%....30%....35%....40%....45%....50%....55%....60%...
Download Complete!
Found and verified notMNIST_large.tar.gz
Attempting to download: notMNIST_small.tar.gz
0%....5%....10%....15%....20%....25%....30%....35%....40%....45%....50%....55%....60%...
Download Complete!
Found and verified notMNIST_small.tar.gz
```

```
num_classes = 10
np.random.seed(133)

def maybe_extract(filename, force=False):
  root = os.path.splitext(os.path.splitext(filename)[0])[0]  # remove .tar.gz
  if os.path.isdir(root) and not force:
    # You may override by setting force=True.
    print('%s already present - Skipping extraction of %s.' % (root, filename))
  else:
    print('Extracting data for %s. This may take a while. Please wait.' % root)
    tar = tarfile.open(filename)
    sys.stdout.flush()
    tar.extractall()
    tar.close()
  data_folders = [
    os.path.join(root, d) for d in sorted(os.listdir(root))
    if os.path.isdir(os.path.join(root, d))]
  if len(data_folders) != num_classes:
    raise Exception(
```

```
      'Expected %d folders, one per class. Found %d instead.' % (
        num_classes, len(data_folders)))
  print(data_folders)
  return data_folders

train_folders = maybe_extract(train_filename)
test_folders = maybe_extract(test_filename)
```

```
Extracting data for notMNIST_large. This may take a while. Please wait.
['notMNIST_large/A', 'notMNIST_large/B', 'notMNIST_large/C', 'notMNIST_large/D', 'notMNI
Extracting data for notMNIST_small. This may take a while. Please wait.
['notMNIST_small/A', 'notMNIST_small/B', 'notMNIST_small/C', 'notMNIST_small/D', 'notMNI
```

```
# task 3
# Разделите данные на три подвыборки:
# обучающую (200 тыс. изображений),
# валидационную (10 тыс. изображений)
# и контрольную (тестовую) (19 тыс. изображений);
def split_dataset(dataset):
    learn_dataset = dataset[0:450000]
    print(len(learn_dataset))
    test_dataset = dataset[450001:461955]
    print(len(test_dataset))
    return learn_dataset,test_dataset

def randomize_list(dataset):
    from random import shuffle
    shuffle(dataset)
    return dataset

from tqdm import tqdm
import hashlib
def md5(fname):
    hash_md5 = hashlib.md5()
    with open(fname, "rb") as f:
        for chunk in iter(lambda: f.read(4096), b""):
            hash_md5.update(chunk)
    return hash_md5.hexdigest()

def get_all_files_recursively(path):
    return [os.path.join(dp, f) for dp, dn, filenames in os.walk(path) for f in filename

def remove_dublicates(dataset):
    dublicate_remover = dict()
    for file in dataset:
        dublicate_remover[md5(file)] = file
    return list(dublicate_remover.values())

dataset = get_all_files_recursively("notMNIST_large")
# task 4
```

```
# Проверьте, что данные из обучающей выборки не пересекаются
# с данными из валидационной и контрольной выборок.
# Другими словами, избавьтесь от дубликатов в обучающей выборке.
dataset = remove_dublicates(dataset)
print(len(dataset))
dataset = randomize_list(dataset)
learn_dataset,test_dataset = split_dataset(dataset)
```

```
461955
450000
11954
```

```
from sklearn.linear_model import LogisticRegression
# from tqdm.notebook import tqdm
import numpy as np
from PIL import Image

alphabet = {'A':0, 'B':1, 'C':2, 'D':3, 'E':4, 'F':5, 'G':6, 'H':7, 'I':8, 'J':9}
class_names = "ABCDEFGHIJ"
# def Learn(X_train, y_train, X_test, y_test):
#     clf = LogisticRegression(random_state=0).fit(X_train, y_train)
#     predicted = clf.predict(X_test)
#     score = clf.score(X_test, y_test)
#     print(score)
#     return score

def GetClassData(path):
    return alphabet[path.split("/")[1]]


def GetLearnData(learn_dataset, test_dataset, train_len=len(learn_dataset)):
    X_train, y_train, X_test, y_test = [],[],[],[]
    for index in tqdm(range(train_len)):
        path = learn_dataset[index]
        try:
            img = Image.open(path)
        except:
            continue
        arr = np.array(img)
        X_train.append(arr)
        y_train.append(GetClassData(path))
    for path in tqdm(test_dataset):
        try:
            img = Image.open(path)
        except:
            continue
        arr = np.array(img)
        X_test.append(arr)
        y_test.append(GetClassData(path))
    return (X_train, y_train), (X_test, y_test)

```

```
(train_images, train_labels), (test_images, test_labels)  = GetLearnData(learn_dataset,

train_images = np.asarray(train_images)
test_images = np.asarray(test_images)
train_labels = np.asarray(train_labels)
test_labels = np.asarray(test_labels)
```

```
100%|██████████| 450000/450000 [00:50<00:00, 8955.67it/s]
100%|██████████| 11954/11954 [00:01<00:00, 8945.71it/s]
```

```
train_images = train_images / 255.0
test_images = test_images / 255.0
```

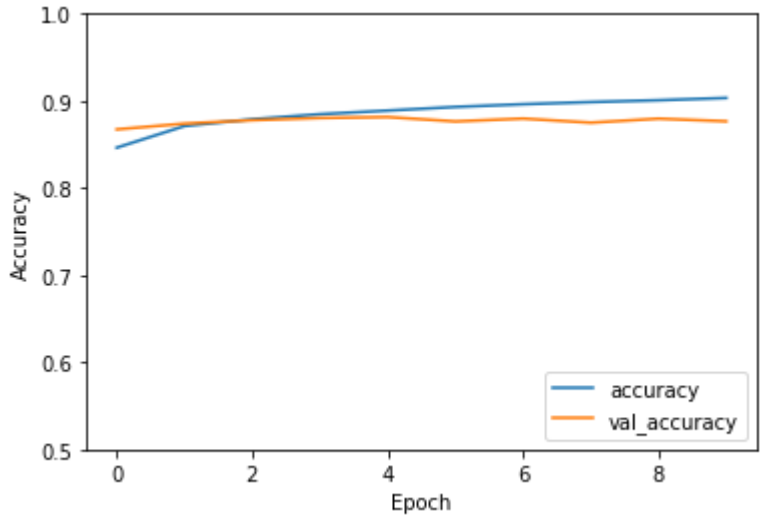
```
train_images = np.expand_dims(train_images, axis=3)
test_images = np.expand_dims(test_images, axis=3)
```

Задание 1. Реализуйте нейронную сеть с двумя сверточными слоями, и одним полносвязным функцией активации. Какова точность построенное модели?

```
model = models.Sequential()
model.add(layers.Conv2D(28, (1, 1), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.Conv2D(56, (1, 1), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(56, activation='relu'))
model.add(layers.Dense(10))
```

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)

print("Точность модели:",test_acc)
```

```
Train on 449998 samples, validate on 11954 samples
Epoch 1/10
449998/449998 [==============================] - 529s 1ms/sample - loss: 0.5125 - accura
Epoch 2/10
449998/449998 [==============================] - 501s 1ms/sample - loss: 0.4272 - accura
Epoch 3/10
449998/449998 [==============================] - 537s 1ms/sample - loss: 0.3985 - accura
Epoch 4/10
449998/449998 [==============================] - 536s 1ms/sample - loss: 0.3786 - accura
Epoch 5/10
449998/449998 [==============================] - 501s 1ms/sample - loss: 0.3619 - accura
Epoch 6/10
449998/449998 [==============================] - 500s 1ms/sample - loss: 0.3474 - accura
Epoch 7/10
449998/449998 [==============================] - 548s 1ms/sample - loss: 0.3356 - accura
Epoch 8/10
449998/449998 [==============================] - 572s 1ms/sample - loss: 0.3257 - accura
Epoch 9/10
449998/449998 [==============================] - 567s 1ms/sample - loss: 0.3171 - accura
Epoch 10/10
449998/449998 [==============================] - 531s 1ms/sample - loss: 0.3084 - accura
11954/11954 - 2s - loss: 0.4178 - accuracy: 0.8766
Точность модели: 0.87661034
```

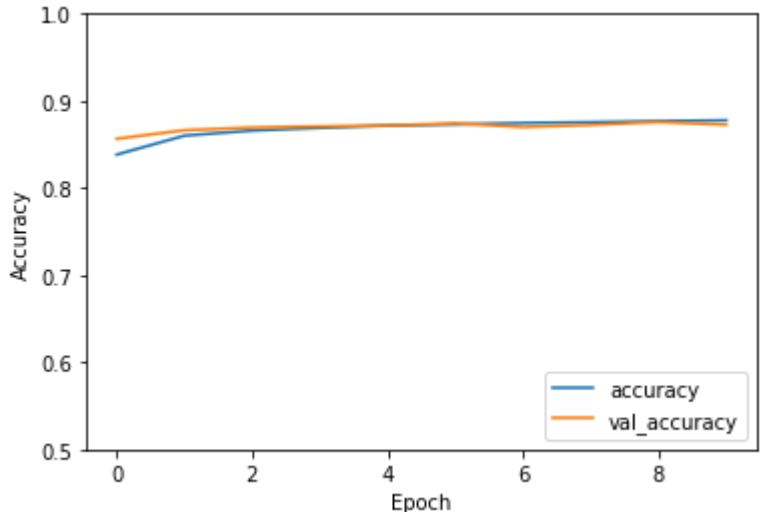


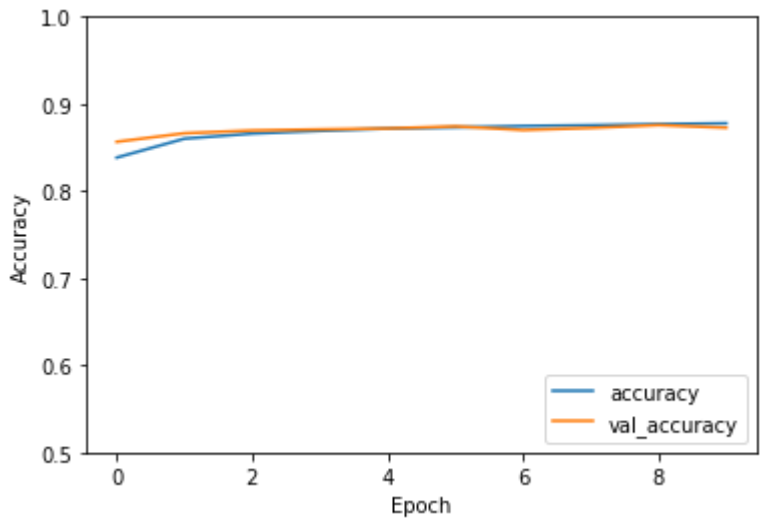
Задание 2. Замените один из сверточных слоев на слой, реализующий операцию пулинга (Poo среднего. Как это повлияло на точность классификатора?

```
model = models.Sequential()
model.add(layers.Conv2D(28, (1, 1), activation='relu', input_shape=(28, 28, 1)))
# слой, реализующий операцию пулинга
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(56, activation='relu'))
model.add(layers.Dense(10))

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
```

```
                metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)

print("Точность модели:",test_acc)
```

```
449998/449998 [==============================] - 134s 298us/sample - loss: 0.5463 - accu
449998/449998 [==============================] - 134s 298us/sample - loss: 0.5463 - accu
Epoch 2/10
    32/449998 [..............................] - ETA: 3:21 - loss: 0.3861 - accuracy: 0.
449998/449998 [==============================] - 133s 295us/sample - loss: 0.4718 - accu
449998/449998 [==============================] - 133s 295us/sample - loss: 0.4718 - accu
Epoch 3/10
    32/449998 [..............................] - ETA: 4:21 - loss: 0.1927 - accuracy: 0.
449998/449998 [==============================] - 134s 298us/sample - loss: 0.4514 - accu
449998/449998 [==============================] - 134s 298us/sample - loss: 0.4514 - accu
Epoch 4/10
    32/449998 [..............................] - ETA: 3:34 - loss: 0.3756 - accuracy: 0.
449998/449998 [==============================] - 133s 296us/sample - loss: 0.4395 - accu
449998/449998 [==============================] - 133s 296us/sample - loss: 0.4395 - accu
Epoch 5/10
    32/449998 [..............................] - ETA: 4:07 - loss: 0.6413 - accuracy: 0.
449998/449998 [==============================] - 132s 293us/sample - loss: 0.4310 - accu
449998/449998 [==============================] - 132s 293us/sample - loss: 0.4310 - accu
Epoch 6/10
    32/449998 [..............................] - ETA: 2:56 - loss: 0.3562 - accuracy: 0.
449998/449998 [==============================] - 128s 285us/sample - loss: 0.4252 - accu
449998/449998 [==============================] - 128s 285us/sample - loss: 0.4252 - accu
Epoch 7/10
    32/449998 [..............................] - ETA: 2:53 - loss: 0.4763 - accuracy: 0.
449998/449998 [==============================] - 128s 283us/sample - loss: 0.4209 - accu
449998/449998 [==============================] - 128s 283us/sample - loss: 0.4209 - accu
Epoch 8/10
    32/449998 [..............................] - ETA: 2:45 - loss: 0.5424 - accuracy: 0.
449998/449998 [==============================] - 131s 292us/sample - loss: 0.4174 - accu
449998/449998 [==============================] - 131s 292us/sample - loss: 0.4174 - accu
Epoch 9/10
    32/449998 [..............................] - ETA: 3:16 - loss: 0.4173 - accuracy: 0.
449998/449998 [==============================] - 137s 304us/sample - loss: 0.4141 - accu
449998/449998 [==============================] - 137s 304us/sample - loss: 0.4141 - accu
Epoch 10/10
    32/449998 [..............................] - ETA: 3:41 - loss: 0.5957 - accuracy: 0.
449998/449998 [==============================] - 138s 308us/sample - loss: 0.4109 - accu
449998/449998 [==============================] - 138s 308us/sample - loss: 0.4109 - accu
11954/11954 - 1s - loss: 0.4266 - accuracy: 0.8727
11954/11954 - 1s - loss: 0.4266 - accuracy: 0.8727
Точность модели: 0.8726786
Точность модели: 0.8726786
```

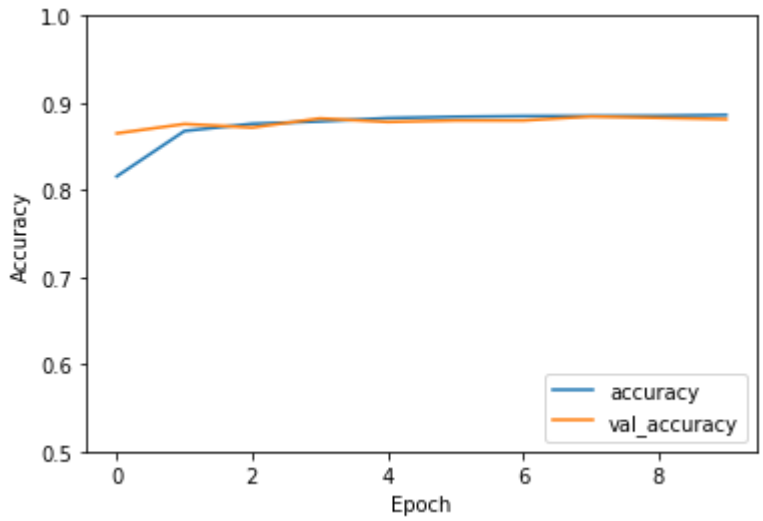


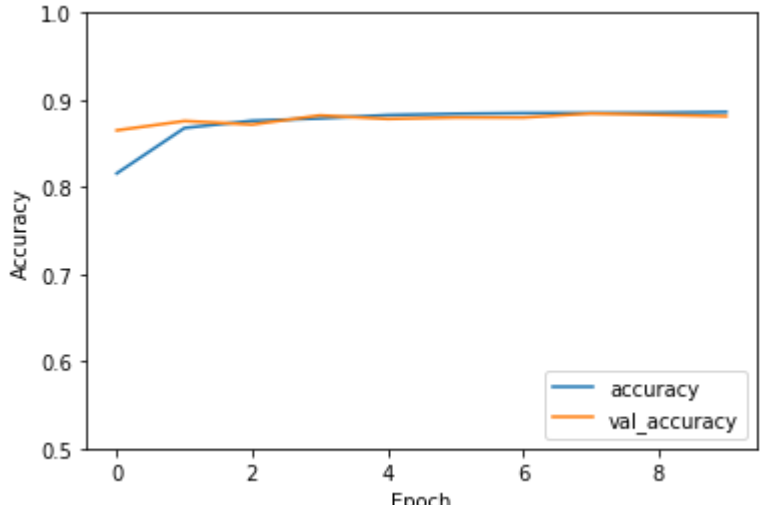


Задание 3. Реализуйте классическую архитектуру сверточных сетей LeNet-5 (http://yann.lecun.c

```
model = keras.Sequential()
model.add(layers.Conv2D(filters=6, kernel_size=(3, 3), activation='relu', input_shape=(2
model.add(layers.AveragePooling2D())
model.add(layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu'))
model.add(layers.AveragePooling2D())
model.add(layers.Flatten())
model.add(layers.Dense(units=120, activation='relu'))
model.add(layers.Dense(units=84, activation='relu'))
model.add(layers.Dense(units=10, activation = 'softmax'))

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)

print("Точность модели:",test_acc)
```

```
    32/449998 [..............................] - ETA: 3:20 - loss: 1.6034 - accuracy: 0.
449998/449998 [==============================] - 158s 351us/sample - loss: 1.5781 - accu
449998/449998 [==============================] - 158s 351us/sample - loss: 1.5781 - accu
Epoch 6/10
    32/449998 [..............................] - ETA: 7:49 - loss: 1.5861 - accuracy: 0.
449998/449998 [==============================] - 175s 389us/sample - loss: 1.5767 - accu
449998/449998 [==============================] - 175s 389us/sample - loss: 1.5767 - accu
Epoch 7/10
    32/449998 [..............................] - ETA: 3:52 - loss: 1.5831 - accuracy: 0.
449998/449998 [==============================] - 168s 373us/sample - loss: 1.5759 - accu
449998/449998 [==============================] - 168s 373us/sample - loss: 1.5759 - accu
Epoch 8/10
    32/449998 [..............................] - ETA: 5:27 - loss: 1.5550 - accuracy: 0.
449998/449998 [==============================] - 170s 378us/sample - loss: 1.5758 - accu
449998/449998 [==============================] - 170s 378us/sample - loss: 1.5758 - accu
Epoch 9/10
    32/449998 [..............................] - ETA: 5:16 - loss: 1.5549 - accuracy: 0.
449998/449998 [==============================] - 169s 376us/sample - loss: 1.5756 - accu
449998/449998 [==============================] - 169s 376us/sample - loss: 1.5756 - accu
Epoch 10/10
    32/449998 [..............................] - ETA: 5:16 - loss: 1.5849 - accuracy: 0.
449998/449998 [==============================] - 181s 403us/sample - loss: 1.5747 - accu
449998/449998 [==============================] - 181s 403us/sample - loss: 1.5747 - accu
11954/11954 - 2s - loss: 1.5795 - accuracy: 0.8813
11954/11954 - 2s - loss: 1.5795 - accuracy: 0.8813
Точность модели: 0.88129497
Точность модели: 0.88129497
```





Задание 4. Сравните максимальные точности моделей, построенных в лабораторных работах различия?

Номер лабораторной работы	Максимальная точность
1	0.8068
2	**0.8889**
3	0.8812

Объяснение: в лабораторных работах исопльховались различные методы классификации изо оказалось использование метода из второй лабораторной работы т.к метод использует стоха классификациии.