Московский государственный технический университет

им. Н.Э. Баумана

УТВЕРЖДАЮ:

Гапанюк Ю.Е.

"__"_____2019 г.

Курсовая работа по курсу «Технологии машинного обучения»

Пояснительная записка
(вид документа)

писчая бумага
(вид носителя)

20
(количество листов)

ИСПОЛНИТЕЛИ:

студент группы ИУ5-61 _____
Водка И.Э.
"__"_____ 2019 г.

Москва – 2019

Содержание

1. Задание:

1.1. Общее задание

Схема типового исследования, проводимого студентом в рамках курсовой работы, содержит выполнение следующих шагов:

- Поиск и выбор набора данных для построения моделей машинного обучения. На основе выбранного набора данных студент должен построить модели машинного обучения для решения или задачи классификации, или задачи регрессии.
- Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.
- Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.
- Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения. В зависимости от набора данных, порядок выполнения пунктов 2, 3, 4 может быть изменен.
- Выбор метрик для последующей оценки качества моделей. Необходимо выбрать не менее трех метрик и обосновать выбор.
- Выбор наиболее подходящих моделей для решения задачи классификации или регрессии. Необходимо использовать не менее пяти моделей, две из которых должны быть ансамблевыми.
- Формирование обучающей и тестовой выборок на основе исходного набора данных.
- Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров. Производится обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки.
- Подбор гиперпараметров для выбранных моделей. Рекомендуется использовать методы кросс-валидации. В зависимости от используемой библиотеки можно применять функцию GridSearchCV, использовать перебор параметров в цикле, или использовать другие методы.
- Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.
- Формирование выводов о качестве построенных моделей на основе выбранных метрик. Результаты сравнения качества рекомендуется отобразить в виде графиков и сделать выводы в форме текстового описания. Рекомендуется постройение графиков обучения и валидации, влияния значений гиперпарметров на качество моделей и т.д.

Приведенная схема исследования является рекомендуемой. Возможно выполнение курсовой работы на нестандартную тему, которая должна быть предварительно согласована с ответственным за прием курсовой работы.

1.2. Задание данной курсовой работы

Для своей курсовой работы я выбрал нестандартную тему. Причины такого поступка:

- Требования, изложенные выше, характерны как для данной курсовой работы, так и для лабораторных работ, выполненных ранее.
- Тем не менее, многие интересные аспекты работы с технологиями машинного обучения не были рассмотрены в практических работах в данном семестре.
- Значит, нет проблемы в том, чтобы использовать собственный вариант выполнения курсовой работы, т.к. третий раз закреплять одно и то же – хорошо, но рассмотреть что-то новое – тоже неплохо.

Соответственно, задание курсовой работы было придумано самостоятельно и выглядит следующим образом:

Необходимо разработать систему, предварительно обученную на обучающей выборке и определяющую по фотографии, сделана ли фотография в городе России или Великобритании. Т.е., классификатор.

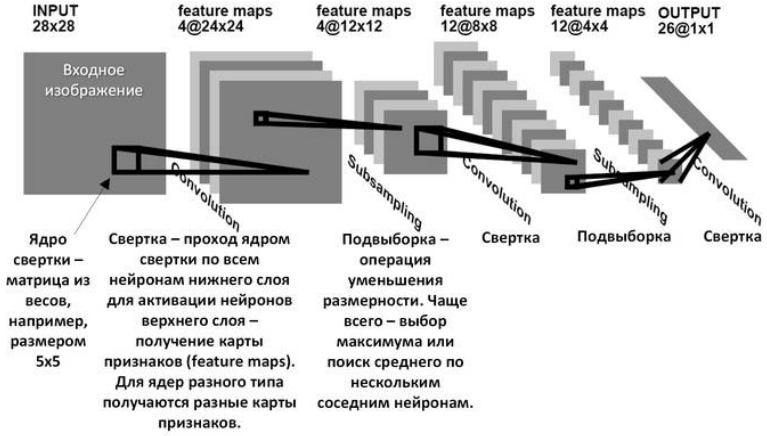
Используемые технологии:

- Node.JS для получения выборки фотографий из Google Street View;
- Python (Anaconda) + Jupyter Notebook;
- Keras (сверточная нейронная сеть)
- прочие инструменты (Matplotlib, Pandas, Numpy, Sklearn...)

2. Введение

Задача обработки изображений – весьма сложная, но с ней неплохо справляются **свёрточные нейронные сети**.

Цитата из Википедии:



Свёрточная нейронная сеть *(англ. convolutional neural network, CNN)* — специальная архитектура искусственных нейронных сетей, предложенная Яном Лекуном в 1988 году и нацеленная на эффективное распознавание образов, входит в состав технологий глубокого обучения (англ. deep learning). Использует некоторые особенности зрительной коры, в которой были открыты так называемые простые клетки, реагирующие на прямые линии под разными углами, и сложные клетки, реакция которых связана с активацией определённого набора простых клеток. Таким образом, идея свёрточных нейронных сетей заключается в чередовании свёрточных слоёв (англ. *convolution layers*) и субдискретизирующих слоёв (англ. *subsampling layers* или англ. *pooling layers*, слоёв подвыборки). Структура сети — однонаправленная (без обратных связей), принципиально многослойная. Для обучения используются стандартные методы, чаще всего метод обратного распространения ошибки. Функция активации нейронов (передаточная функция) — любая, по выбору исследователя.

Название архитектура сети получила из-за наличия операции свёртки, суть которой в том, что каждый фрагмент изображения умножается на матрицу (ядро) свёртки поэлементно, а результат суммируется и записывается в аналогичную позицию выходного изображения.

Работу в этой курсовой работе я выполнял на “инженерном” уровне, не сильно вдаваясь в математические подробности, однако основы архитектуры я рассмотрел.

Подробнее о выполнении работы – в основной части данного документа.

Ещё один важный этап выполнения – **подготовить хороший датасет**. Вместо использования готового датасета я написал небольшой скрипт на Node.JS, загружающий изображения с Google Street View. Далее эти данные следовало обработать (описывается в первой части основного раздела).

Наконец, после завершения работы нейронной сети были сделаны **выводы с графиками и небольшими расчётами**.

3. Основная часть

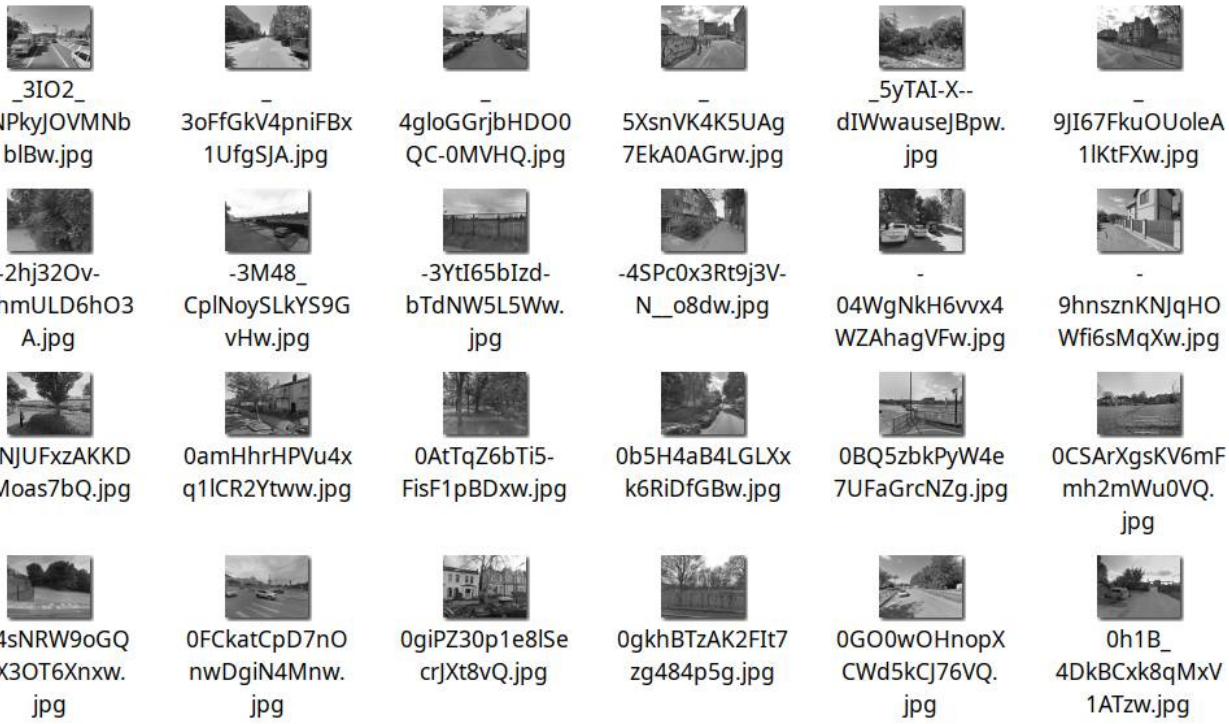
3.1. Подготовка датасета

```
const request = require('request-promise');
const { execSync } = require('child_process');
const fs = require('fs');

const mkdirp = require('mkdirp-promise');
const sharp = require('sharp');

const IMAGE_SIZE = 400;
const ATTEMPTS = 10;
const ROUND_RATIO = 100000;

const GENERATE_TEST = true;

// Генерирует случайные координаты в прямоугольнике
const randomCoordsInSquare = ({lat1, lng1, lat2, lng2}) => {
  const minLat = Math.min(lat1, lat2);
  const maxLat = Math.max(lat1, lat2);
  const minLng = Math.min(lng1, lng2);
  const maxLng = Math.max(lng1, lng2);

  const lat = minLat + Math.random() * (maxLat - minLat);
  const lng = minLng + Math.random() * (maxLng - minLng);

  return { lat: Math.round(lat * ROUND_RATIO) / ROUND_RATIO, lng: Math.round(lng
* ROUND_RATIO) / ROUND_RATIO };
};

// Находит картинку в “прямоугольнике” координат
const findPictureForSquare = async (square, country) => {
  const makeUri = (lat, lng) => {
    return [

'https://maps.googleapis.com/maps/api/js/GeoPhotoService.SingleImageSearch',
        `?pb=!1m5!1sapiv3!5sUS!11m2!1m1!1b0!2m4!1m2!3d${lat}!4d${lng}!2d100!
3m18!2m2!1sru!2sRU!`,
        '9m1!1e2!11m12!1m3!1e2!2b1!3e2!1m3!1e3!2b1!3e2!1m3!1e10!2b1!3e2!4m6!1e1!
1e2!1e3!1e4!1e8!1e6',
        '&callback=respond'
      ].join('');
  };


  let downloaded = false, attemptsRemaining = ATTEMPTS, city, panoId;

  while(!downloaded && attemptsRemaining > 0) {
    const coords = randomCoordsInSquare(square);
    const uri = makeUri(coords.lat, coords.lng);
    const result = await request.get(uri);

    const respond = async (results) => {
      if (results.length === 1 || !Array.isArray(results[1][3][2])) {
        console.error('Attempt #' + (ATTEMPTS - attemptsRemaining + 1)
          + ' failed! ' + JSON.stringify(coords));
        downloaded = false;
        attemptsRemaining--;
        await execSync('sleep 0.05');
      } else {
        console.log('Success!');
```

```
        city = results[1][3][2][0][0];
        panoId = results[1][1][1];
        downloaded = true;
      }
    };

    eval(result);
  }

  if (!downloaded) {
    console.error('Failed!');
    return;
  }

  const rawDir = `${__dirname}/raw/${panoId}`;
  await mkdirp(rawDir);
  execSync(`google_streetview --pano=${panoId} -size=${IMAGE_SIZE}x${IMAGE_SIZE}
--save_downloads=${rawDir}`);

  const newPath = `${__dirname}/` + GENERATE_TEST ? 'images_test' :
'images_train';
  await mkdirp(newPath);

  sharp(`${rawDir}/gsv_0.jpg`)
    .resize({ height: Math.round(IMAGE_SIZE * 0.75), width: IMAGE_SIZE })
    .grayscale()
    .toFile(`${newPath}/${panoId}.jpg`);

  const line = [panoId, country].join(',');
  fs.appendFileSync(GENERATE_TEST ? 'dataset_test.csv' : 'dataset_train.csv',
line + "\n");
  console.log(line);
};

// В этом словаре – список мест, из которых берутся фотографии
const places = {
  moscow: {
    count: 10,
    coords: {lat1: 55.894966, lng1: 37.382917, lat2: 55.610124, lng2:
37.819175},
    country: 'ru',
  },
  spb: {
    count: 5,
    coords: {lat1: 60.039628, lng1: 30.145742, lat2: 59.881093, lng2:
30.537374},
    country: 'ru',
  },
  ekb: {
    count: 3,
    coords: {lat1: 56.862860, lng1: 60.538415, lat2: 56.822548, lng2:
60.688518},
    country: 'ru',
  },
  nino: {
    count: 3,
    coords: {lat1: 56.333229, lng1: 43.898472, lat2: 56.277297, lng2:
44.087820},
    country: 'ru',
  },
  kazan: {
    count: 3,
    coords: {lat1: 55.880916, lng1: 49.043136, lat2: 55.744434, lng2:
49.238918},
```

```
    country: 'ru',
  },
  krasnodar: {
    count: 3,
    coords: {lat1: 45.067626, lng1: 38.936263, lat2: 45.014970, lng2:
39.047241},
    country: 'ru',
  },

  london: {
    count: 10,
    coords: {lat1: 51.548547, lng1: -0.207508, lat2: 51.439893, lng2: 0.029800},
    country: 'uk'
  },
  birmingham: {
    count: 5,
    coords: {lat1: 52.538597, lng1: -1.991987, lat2: 52.424620, lng2: -
1.782344},
    country: 'uk',
  },
  liverpool: {
    count: 3,
    coords: {lat1: 53.449859, lng1: -2.999910, lat2: 53.359085, lng2: -
2.822562},
    country: 'uk',
  },
  manchester: {
    count: 3,
    coords: {lat1: 53.499370, lng1: -2.287626, lat2: 53.455022, lng2: -
2.191537},
    country: 'uk',
  },
  southampton: {
    count: 3,
    coords: {lat1: 50.953016, lng1: -1.479036, lat2: 50.898672, lng2: -
1.317257},
    country: 'uk',
  },
  bristol: {
    count: 3,
    coords: {lat1: 51.469197, lng1: -2.620845, lat2: 51.448589, lng2: -
2.554304},
    country: 'uk',
  },
};

(async () => {
  Object.entries(places)
    .forEach(async ([placeName, placeData]) => {
      console.log('Handling: ' + placeName);
      for (let i = 0; i < placeData.count; i++) {
        await findPictureForSquare(placeData.coords, placeData.country);
      }
  });
})();
```

После запуска и некоторого ожидания получаем папку с фотографиями:



Также получаем файл “dataset_train.csv” следующего вида:

```
file,country
_kdYaGTQTUV1eqQZmfTDvw,ru
e94B91epyBFaUHMCKsXR7g,uk
BoDz-ngJQjhS_PszrCEvJA,ru
VhdZx6hrKM-VJWSjDQk2mw,uk
HT_VTtPVt7RONFBnRe-rvg,uk
P1dkmyQnUJYxBbf8mfCQ6Q,uk
Pc_R4Bzn6Q5aXG5Yf-tl9A,ru
sKKRClFHoDzPAE82YGGm3g,ru
_JZ0Dh1crToE8JPyRySiuA,uk
CUhnn0Bw6h3FYgQzuaSBnQ,uk
Wg_vkIBuZo8rxqsPpN8K_g,ru
Q_0BMBXJMmQ65GeNJvWqqw,ru
vKlOvyLmqhqBffE3g94DzA,uk
ZEdc0xugROGBQkNM8CJM0A,uk
3JfkUl8SVcAw_qAYa3KS-A,uk
tn4_Cw5Un16LnJDpU5JM9w,uk
Wt7Tgh-aIq69lkYlt2lFQg,ru
…
```

Аналогично поступаем с тестовым датасетом. Наконец, все файлы готовы, и мы можем приступать непосредственно к разработке классификатора.

3.2. Jupyter Notebook

In [2]:

```
from PIL import Image
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

Убедимся, что используются GPU. Так скорость обучения увеличивается в разы.

In [4]:

```
from keras import backend as K
K.tensorflow_backend._get_available_gpus()
```

Out[4]:

```
['/job:localhost/replica:0/task:0/device:GPU:0']
```

Сбор данных из датасета

In [5]:

```
def load_image(dirname, file):
    arr = np.array(Image.open('./' + dirname + '/' + file +
'.jpg').convert('L'))
    return arr.reshape(400, 300, 1)
```

In [6]:

```
dataset = pd.read_csv('./dataset.csv')
dataset['image'] = dataset.apply(lambda x: load_image('images', x.file), axis=1)
dataset['label'] = dataset.apply(lambda x: np.array([x.country == 'ru',
x.country == 'uk']), axis=1)
dataset.iloc[0].image.shape
```

Out[6]:

```
(400, 300, 1)
```

Описание модели глубокого обучения на Keras

In [7]:

```
from keras.models import Sequential
from keras.layers import Dense, Activation, Conv2D, MaxPooling2D,
BatchNormalization, Dropout, Flatten

N_IMAGES = len(dataset)
IMG_W = 400
IMG_H = 300

model = Sequential()

model.add(Conv2D(32, kernel_size=(3, 3), use_bias=False, input_shape=(IMG_W,
IMG_H, 1)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
```

```
model.add(Activation('relu'))

model.add(Conv2D(64, kernel_size=(3,3), use_bias=False))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Conv2D(64, kernel_size=(3,3), use_bias=False))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Conv2D(128, kernel_size=(3,3), use_bias=False))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Conv2D(64, kernel_size=(3,3), use_bias=False))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Conv2D(32, kernel_size=(3,3), use_bias=False))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Dropout(0.2))
model.add(Flatten())

model.add(Dense(128, use_bias=False))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Dense(2, use_bias=False))
model.add(BatchNormalization())
model.add(Activation('softmax'))
```

In [8]:

```
dataset['image'].iloc[0].shape
```

Out[8]:

```
(400, 300, 1)
```

Обучение модели

Компилируем модель, указывая лосс и используемый оптимизатор (стандартные значения, в общем):

In [9]:

```
from keras import optimizers
from keras import losses

model.compile(loss=losses.mean_squared_error, optimizer='sgd')
```

In [10]:

```
x = np.array(dataset['image'].tolist())
```

```
y = np.array(dataset['label'].tolist())
print 'x shape is', x.shape
print 'y shape is', y.shape

x shape is (4672, 400, 300, 1)
y shape is (4672, 2)
```

Запускаем непосредственно обучение, применяя k-fold cross validation:

In [11]:

```
from sklearn.model_selection import train_test_split
from keras.callbacks import EarlyStopping, ModelCheckpoint

# set early stopping criteria
patience = 2 # this is the number of epochs with no improvement after which the
training will stop
early_stopping = EarlyStopping(monitor='val_loss', patience=patience, verbose=1)

#define the model checkpoint callback -> this will keep on saving the model as a
physical file
model_checkpoint = ModelCheckpoint('./checkpoint.h5', verbose=1,
save_best_only=True)

n_folds = 7
epochs = 5
batch_size = 10
model_history = []

def fit_and_evaluate(t_x, val_x, t_y, val_y, used_epochs, used_batch_size):
    results = model.fit(t_x, t_y, epochs=used_epochs,
batch_size=used_batch_size, callbacks=[early_stopping, model_checkpoint],
              verbose=1, validation_split=0.1)
    print "Validation Score: ", model.evaluate(val_x, val_y)
    return results

for i in range(n_folds):
    print "Training on fold: ", i+1
    t_x, val_x, t_y, val_y = train_test_split(x, y, test_size=0.1, random_state
= np.random.randint(1,1000, 1)[0])
    model_history.append(fit_and_evaluate(t_x, val_x, t_y, val_y, epochs,
batch_size))
    print "=======" * 12 + "\n\n\n"
```

```
('Training on fold: ', 1)
Train on 3783 samples, validate on 421 samples
Epoch 1/5
3783/3783 [==============================] - 74s 20ms/step - loss: 0.2476 - val_loss: 0.2081

Epoch 00001: val_loss improved from inf to 0.20808, saving model to ./checkpoint.h5
Epoch 2/5
3783/3783 [==============================] - 57s 15ms/step - loss: 0.2168 - val_loss: 0.2008

Epoch 00002: val_loss improved from 0.20808 to 0.20075, saving model to ./checkpoint.h5
Epoch 3/5
3783/3783 [==============================] - 57s 15ms/step - loss: 0.2017 - val_loss: 0.1902

Epoch 00003: val_loss improved from 0.20075 to 0.19018, saving model to ./checkpoint.h5
Epoch 4/5
3783/3783 [==============================] - 57s 15ms/step - loss: 0.2006 - val_loss: 0.1936

Epoch 00004: val_loss did not improve from 0.19018
Epoch 5/5
3783/3783 [==============================] - 57s 15ms/step - loss: 0.1938 - val_loss: 0.1948

Epoch 00005: val_loss did not improve from 0.19018
Epoch 00005: early stopping
468/468 [==============================] - 4s 8ms/step
```

```
('Validation Score: ', 0.1868088143503564)
================================================================================


('Training on fold: ', 2)
Train on 3783 samples, validate on 421 samples
Epoch 1/5
3783/3783 [==============================] - 59s 16ms/step - loss: 0.1850 - val_loss: 0.1722

Epoch 00001: val_loss improved from 0.19018 to 0.17217, saving model to ./checkpoint.h5
Epoch 2/5
3783/3783 [==============================] - 57s 15ms/step - loss: 0.1756 - val_loss: 0.1914

Epoch 00002: val_loss did not improve from 0.17217
Epoch 3/5
3783/3783 [==============================] - 57s 15ms/step - loss: 0.1727 - val_loss: 0.1947

Epoch 00003: val_loss did not improve from 0.17217
Epoch 00003: early stopping
468/468 [==============================] - 2s 4ms/step
('Validation Score: ', 0.19958432540934309)
================================================================================


('Training on fold: ', 3)
Train on 3783 samples, validate on 421 samples
Epoch 1/5
3783/3783 [==============================] - 57s 15ms/step - loss: 0.1732 - val_loss: 0.2177

Epoch 00001: val_loss did not improve from 0.17217
Epoch 2/5
3783/3783 [==============================] - 57s 15ms/step - loss: 0.1645 - val_loss: 0.1732

Epoch 00002: val_loss did not improve from 0.17217
Epoch 3/5
3783/3783 [==============================] - 57s 15ms/step - loss: 0.1618 - val_loss: 0.1757

Epoch 00003: val_loss did not improve from 0.17217
Epoch 4/5
3783/3783 [==============================] - 57s 15ms/step - loss: 0.1517 - val_loss: 0.2352

Epoch 00004: val_loss did not improve from 0.17217
Epoch 00004: early stopping
468/468 [==============================] - 2s 4ms/step
('Validation Score: ', 0.21811382275106561)
================================================================================


('Training on fold: ', 4)
Train on 3783 samples, validate on 421 samples
Epoch 1/5
3783/3783 [==============================] - 57s 15ms/step - loss: 0.1554 - val_loss: 0.1584

Epoch 00001: val_loss improved from 0.17217 to 0.15843, saving model to ./checkpoint.h5
Epoch 2/5
3783/3783 [==============================] - 57s 15ms/step - loss: 0.1477 - val_loss: 0.1578

Epoch 00002: val_loss improved from 0.15843 to 0.15779, saving model to ./checkpoint.h5
Epoch 3/5
3783/3783 [==============================] - 57s 15ms/step - loss: 0.1456 - val_loss: 0.1638

Epoch 00003: val_loss did not improve from 0.15779
Epoch 4/5
3783/3783 [==============================] - 57s 15ms/step - loss: 0.1402 - val_loss: 0.1447

Epoch 00004: val_loss improved from 0.15779 to 0.14469, saving model to ./checkpoint.h5
Epoch 5/5
3783/3783 [==============================] - 57s 15ms/step - loss: 0.1342 - val_loss: 0.1829

Epoch 00005: val_loss did not improve from 0.14469
468/468 [==============================] - 2s 4ms/step
('Validation Score: ', 0.16414279917366484)
================================================================================


('Training on fold: ', 5)
```

```
Train on 3783 samples, validate on 421 samples
Epoch 1/5
3783/3783 [==============================] - 57s 15ms/step - loss: 0.1423 - val_loss: 0.1678

Epoch 00001: val_loss did not improve from 0.14469
Epoch 2/5
3783/3783 [==============================] - 57s 15ms/step - loss: 0.1349 - val_loss: 0.1755

Epoch 00002: val_loss did not improve from 0.14469
Epoch 3/5
3783/3783 [==============================] - 61s 16ms/step - loss: 0.1337 - val_loss: 0.1778

Epoch 00003: val_loss did not improve from 0.14469
Epoch 00003: early stopping
468/468 [==============================] - 2s 5ms/step
('Validation Score: ', 0.16832360905459803)
================================================================================



('Training on fold: ', 6)
Train on 3783 samples, validate on 421 samples
Epoch 1/5
3783/3783 [==============================] - 58s 15ms/step - loss: 0.1308 - val_loss: 0.1045

Epoch 00001: val_loss improved from 0.14469 to 0.10448, saving model to ./checkpoint.h5
Epoch 2/5
3783/3783 [==============================] - 58s 15ms/step - loss: 0.1244 - val_loss: 0.1847

Epoch 00002: val_loss did not improve from 0.10448
Epoch 3/5
3783/3783 [==============================] - 58s 15ms/step - loss: 0.1179 - val_loss: 0.1075

Epoch 00003: val_loss did not improve from 0.10448
Epoch 00003: early stopping
468/468 [==============================] - 2s 4ms/step
('Validation Score: ', 0.11283582665471949)
================================================================================



('Training on fold: ', 7)
Train on 3783 samples, validate on 421 samples
Epoch 1/5
3783/3783 [==============================] - 58s 15ms/step - loss: 0.1185 - val_loss: 0.1024

Epoch 00001: val_loss improved from 0.10448 to 0.10243, saving model to ./checkpoint.h5
Epoch 2/5
3783/3783 [==============================] - 57s 15ms/step - loss: 0.1078 - val_loss: 0.0930

Epoch 00002: val_loss improved from 0.10243 to 0.09300, saving model to ./checkpoint.h5
Epoch 3/5
3783/3783 [==============================] - 57s 15ms/step - loss: 0.1064 - val_loss: 0.1170

Epoch 00003: val_loss did not improve from 0.09300
Epoch 4/5
3783/3783 [==============================] - 58s 15ms/step - loss: 0.1013 - val_loss: 0.1035

Epoch 00004: val_loss did not improve from 0.09300
Epoch 00004: early stopping
468/468 [==============================] - 2s 4ms/step
('Validation Score: ', 0.11498103360844474)
================================================================================
```

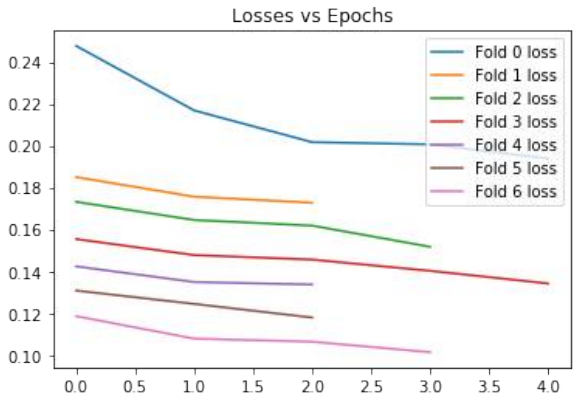
Графики

По горизонтальной оси - номер эпохи, а по вертикальной - показатель loss (средний квадрат ошибки). Здесь видим, что с каждым фолдом loss уменьшается.

In [13]:

```
plt.title('Losses vs Epochs')
```

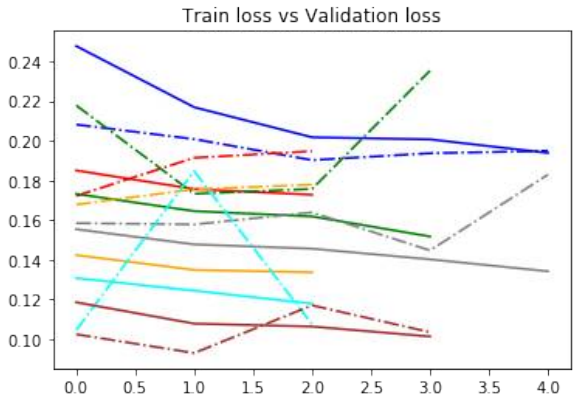
```
for i in range(0, n_folds):
    plt.plot(model_history[i].history['loss'], label='Fold ' + str(i) + ' loss')
plt.legend()
plt.show()
```



Ещё один график - здесь обычными линиями обозначены изменения функции loss, а пунктирными - изменение loss при валидации. Важно, чтобы ошибка при валидации не росла: таким образом мы уменьшаем переобучение.

In [15]:

```
# Validation loss must not rise!
plt.title('Train loss vs Validation loss')

colors = ['blue', 'red', 'green', 'gray', 'orange', 'cyan', 'brown', 'magenta',
'black', 'purple']

for i in range(0, n_folds):
    plt.plot(model_history[i].history['loss'], label='Fold ' + str(i) + ' loss',
color=colors[i])
    plt.plot(model_history[i].history['val_loss'], label='Fold ' + str(i) + '
val loss', color=colors[i], linestyle = "dashdot")

# plt.legend()
plt.show()
```



Результат

In [65]:

```
from matplotlib.font_manager import FontProperties
test_images = pd.read_csv('./dataset_test.csv')
```

```
predictions = []
guesses = 0

font = FontProperties()
font.set_size('large')

for index, row in test_images.iterrows():
    image = row['file']
    real_country = row['country']

    test_image = load_image('images_test', image)
    prediction = model.predict(test_image.reshape(1, 400, 300, 1))
    if prediction[0][0] > 0.50:
        guessed_country = 'ru'
        confidence = prediction[0][0]
    elif prediction[0][1] > 0.50:
        guessed_country = 'uk'
        confidence = prediction[0][1]

    if guessed_country == real_country:
        guesses += 1
        verdict = 'SUCCESS'
        color = 'b'
    else:
        verdict = 'FAILURE'
        color = 'r'

    plt.imshow(Image.open('./images_test/' + image + '.jpg'), cmap="inferno")
    plt.text(
        0,
        -10,
        verdict + "\n" + image + ".jpg\nreal country = " + real_country
             + ",\nguessed country = " + guessed_country
             + ",\nconfidence = " + str(confidence * 100) + "%",
        fontproperties=font,
        color=color
    )
    plt.axis('off')
    plt.show()
```

SUCCESS
TkYmbfuxFWHXT_sxdmvnYA.jpg
real country = uk,
guessed country = uk,
confidence = 0.575528

SUCCESS
3jqtIzQcznOzExfQSh16Hw.jpg
real country = uk,
guessed country = uk,
confidence = 0.961929

SUCCESS
QyMUVdLL2rvc9paP9NqFQg.jpg
real country = ru,
guessed country = ru,
confidence = 0.879323

SUCCESS
W2HBrww7uZGvzU7_4Vm7ug.jpg
real country = uk,
guessed country = uk,
confidence = 0.988308

SUCCESS
-_NClRAiN9dhOy2f-ki2jg.jpg
real country = ru,
guessed country = ru,
confidence = 0.562479

FAILURE
H0fw7ZfPePi7O5mb2AaTnA.jpg
real country = uk,
guessed country = ru,
confidence = 0.915967

SUCCESS
pSnR35sNniFhNQxqVmYHyw.jpg
real country = ru,
guessed country = ru,
confidence = 0.567515

FAILURE
pM0amffa8fkmn6bld9U59Q.jpg
real country = uk,
guessed country = ru,
confidence = 0.98214

FAILURE
4PO-FDhrXycYfhKrMh4FMg.jpg
real country = ru,
guessed country = uk,
confidence = 0.981155

SUCCESS
9lCn-1B8gNEy6j0vNlESJw.jpg
real country = ru,
guessed country = ru,
confidence = 0.696675

FAILURE
0_YxItbAnKqKxZw2v7388A.jpg
real country = ru,
guessed country = uk,
confidence = 0.681674

SUCCESS
NClVN2Tv5Wr8S2D_6C5j4w.jpg
real country = ru,
guessed country = ru,
confidence = 0.600631

[...и так далее...]

In [27]:

```
print 'Угадано:', guesses
print 'Ошибок:', total - guesses
print 'Всего:', total
print 'Точность:', guesses / float(total) * 100, '%'
```

```
Угадано: 83
Ошибок: 25
Всего: 108
Точность: 76.8518518519 %
```

Заключение

Машинное обучение – очень перспективная технология, которая привлекает множество специалистов, инвестиций и пользователей в последнее время. Не обошло оно стороной и нашу кафедру, и всех нас.

Поэтому я не смог обойти его стороной и просто сделать шаблонную курсовую работу. Мне было интересно ощутить, насколько сильны алгоритмы машинного обучения на практике.

Мне действительно понравилось выполнять курсовую работу, и на мой взгляд она неплохо справилась со своей задачей, но её также можно улучшить:

- увеличить обучающую выборку;
- доработать модель глубокого обучения;
- подобрать гиперпараметры, увеличить число эпох и folds;
- закрашивать небо одним цветом, чтобы уменьшить его влияние на оценку;
- и т.д. и т.п.

Список использованных источников

1. Лекции по курсу ТМО (Гапанюк Ю.Е., 2019)
2. Google Street View - https://www.google.com/streetview/
3. Документация Keras - https://keras.io/