Configuration Manual: Fall risk monitoring scheme based on human posture estimation using Transfer learning

MSc Research Project
Data Analytics

Chitra Raghavi Balasubramanian
Student ID: 18183409

School of Computing
National College of Ireland

Supervisor: Christian Horn

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Chitra Raghavi Balasubramanian
Student ID:	x18183409
Programme:	Data Analytics
Year:	2019-2020
Module:	M.Sc. Research Project
Supervisor:	Christian Horn
Submission Due Date:	17/08/2020
Project Title:	Configuration manual: Fall risk monitoring scheme based on human posture estimation using Transfer learning
Word Count:	1096
Page Count:	12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Chitra Raghavi Balasubramanian
Date:	17th August 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	Q
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	Q
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	Q

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual: Fall risk monitoring scheme based on human posture estimation using Transfer learning

Chitra Raghavi Balasubramanian
18183409

1 Introduction

This configuration manual provides information which are related to the research "Fall risk monitoring scheme based on human posture estimation using Transfer learning" from beginning. This document delivers detailed review of the environmental setup, tools and libraries used for building, executing, and testing this research.

2 Hardware Specification

Operating System	Windows 10, Ubuntu 14.04.6
Processor	Intel(R) Core (TM) I7-6500U
Installed Memory (RAM)	8.00GB
System type	64-bit Operating System

3 Software Specification

- VMware Workstation (15.5) Pro for running multiple OS
- Google Colaboratory
- Python 2.7.14 for Image Processing
- Email – For accessing to a Gmail Account
- Python 3 for running deep learning algorithm

3.1 Setting up VMware Workstation

The project used Ubuntu 14.04.6 through VMware Workstation for performing the pre-processing step using Python 2.7.14.

3.2 Environmental Setup of Google Colaboratory

This section shows the Google Colaboratory setup for performing this experiment. A Gmail Id was used to access the Colab notebook: figure 1: shows the sign in step for Google Colab.

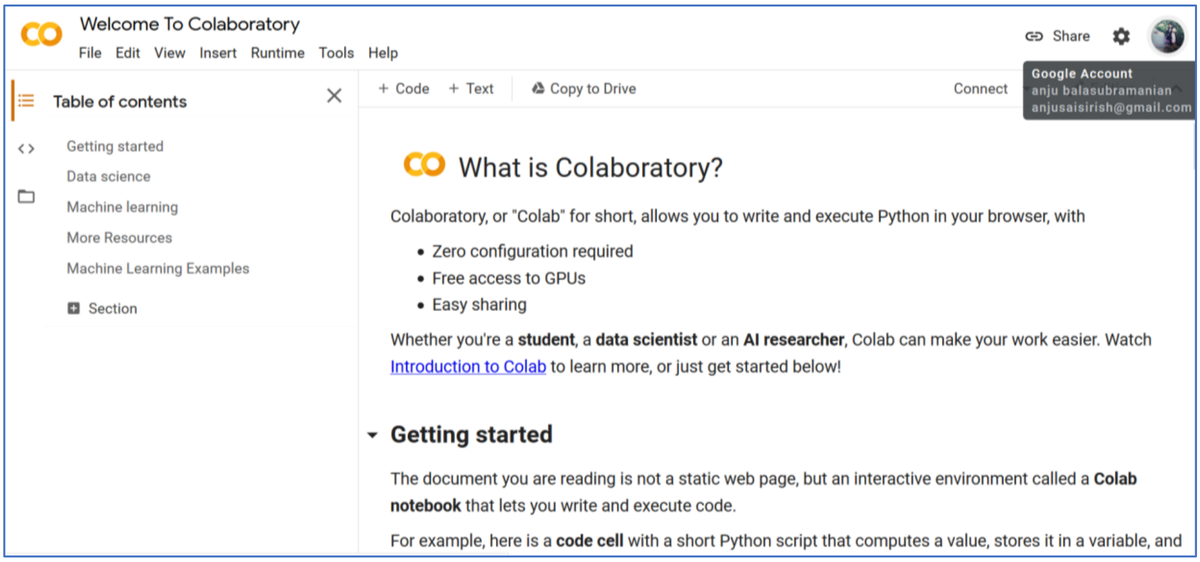


Figure 1: Gmail Id to access Google Colab

4 Dataset Used for Experiment

Figure 2 shows the dataset used for fall detection experiment. The following link will provide access to the dataset http://fenix.univ.rzeszow.pl/~mkepski/ds/uf.html. This dataset was provided by the University of Rzeszow and was created by Michal Kepski. The dataset contains 30 falls and 40 activities of daily life events. They have provided access to both video files and image frames as png files for conducting academic tasks. The image frames extracted from the video files are acquired for this experiment. Also, the dataset contains image frames of Depth data and RGB data from camera 0 and camera 1. The RGB data from Camera 0 was downloaded for performing the classification task. Figure 3: shows the dataset for Fall sequences and Activities of daily life.

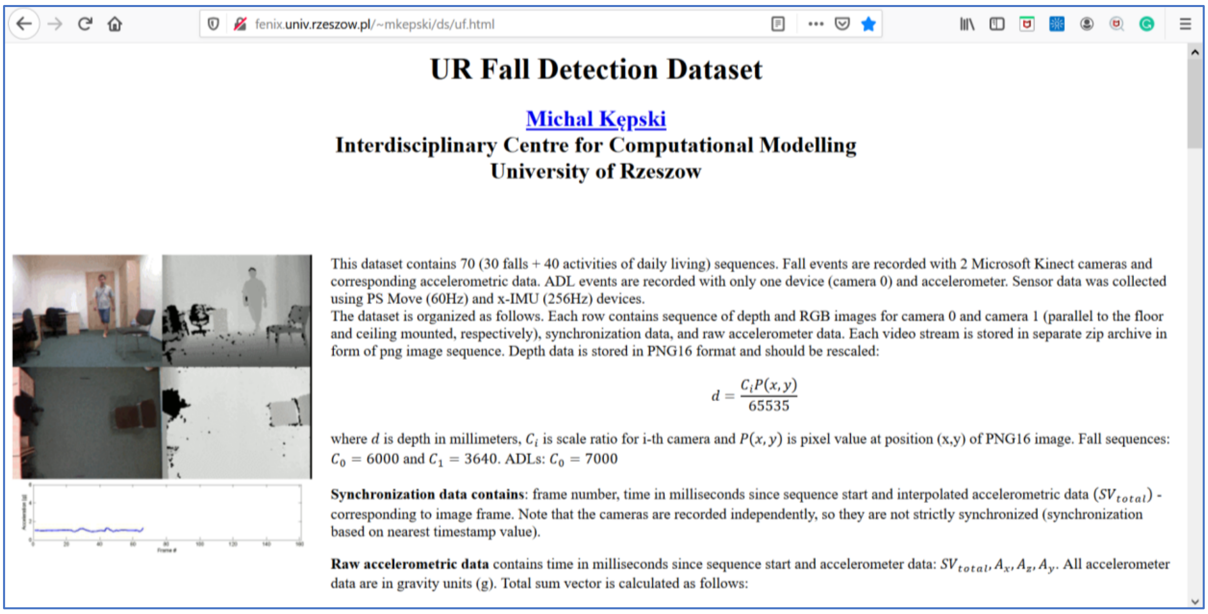


Figure 2: UR fall detection dataset

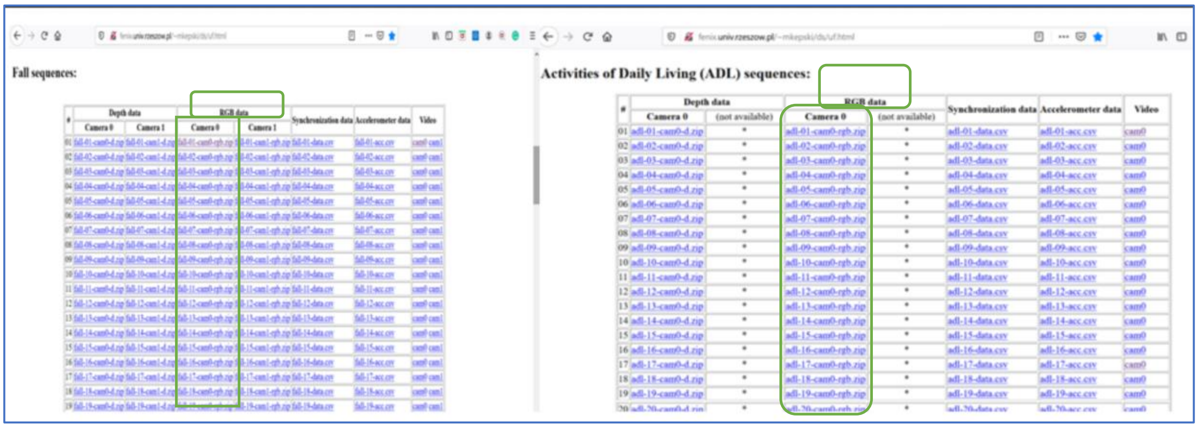


Figure 3: Fall sequence and Activities of daily life

5 Required libraries

Ubuntu 14.04 was used for creating the optical flow images and Windows 10 was used to perform further steps.

5.1 Libraries used for image processing using optical flow algorithm

Some libraries mentioned below are downloaded before using it

- import os
- import cv2
- import glob
- import sys

5.2 Libraries used in Google Colab for building the model

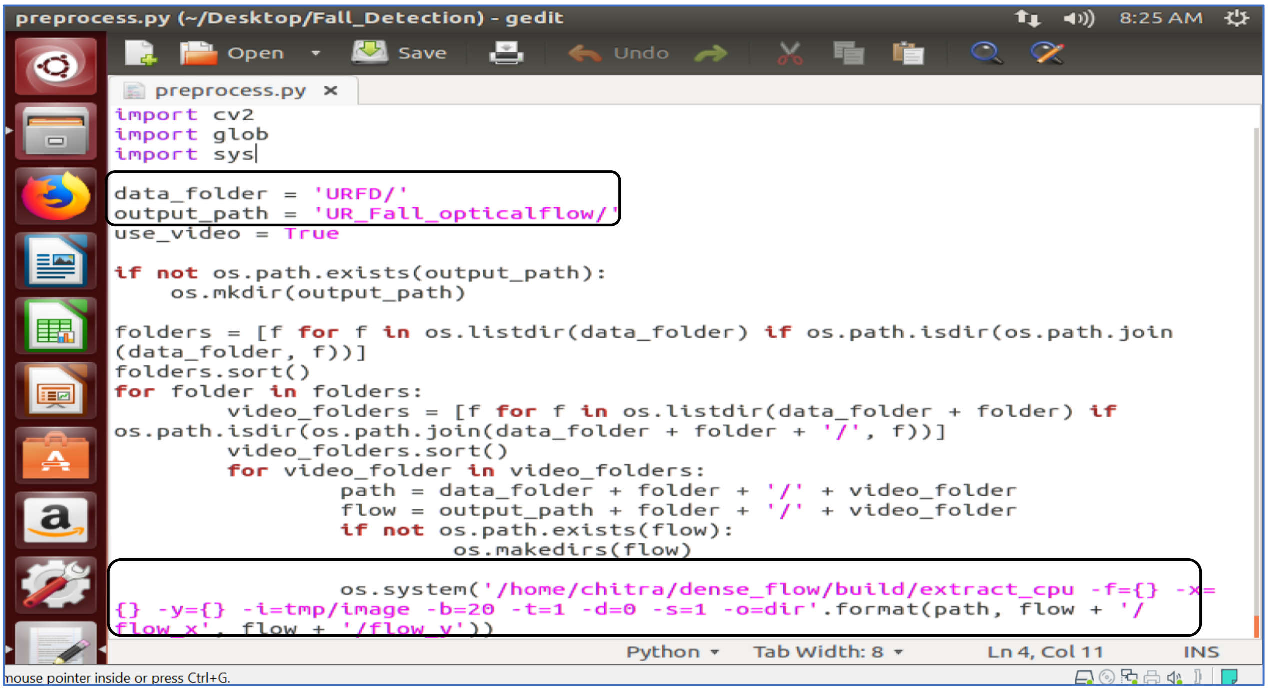
All the libraries used to perform the experiment was shown in figure. These libraries were used for pre-process, build and run the models.


```
import json
import math
import os
import cv2
from PIL import Image
import numpy as np
from keras import layers
from keras.applications import ResNet50,MobileNet, DenseNet201, InceptionV3, NASNetLarge, InceptionResNetV2, NASNetMobile
from keras.callbacks import Callback, ModelCheckpoint, ReduceLROnPlateau, TensorBoard
from keras.preprocessing.image import ImageDataGenerator
from keras.utils.np_utils import to_categorical
from keras.models import Sequential
from keras.optimizers import Adam
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import cohen_kappa_score, accuracy_score
import scipy
from tqdm import tqdm
import tensorflow as tf
from keras import backend as K
import gc
from functools import partial
from sklearn import metrics
from collections import Counter
import json
import itertools
from keras.applications import vgg16, resnet50
```

6. Optical flow images generator in Ubuntu:

Initially, under Ubuntu 14.04 Operating System, Python 2.7.14 was used for pre-processing the image. The software tool provided in the following link were used to compute the optical flow images https://github.com/yjxiong/dense_flow/tree/opencv-3.1.This tool was offered to extract dense optical flow from videos with the help of OpenCV. The steps and command provided in the above-mentioned link were used to compute the optical flow images.

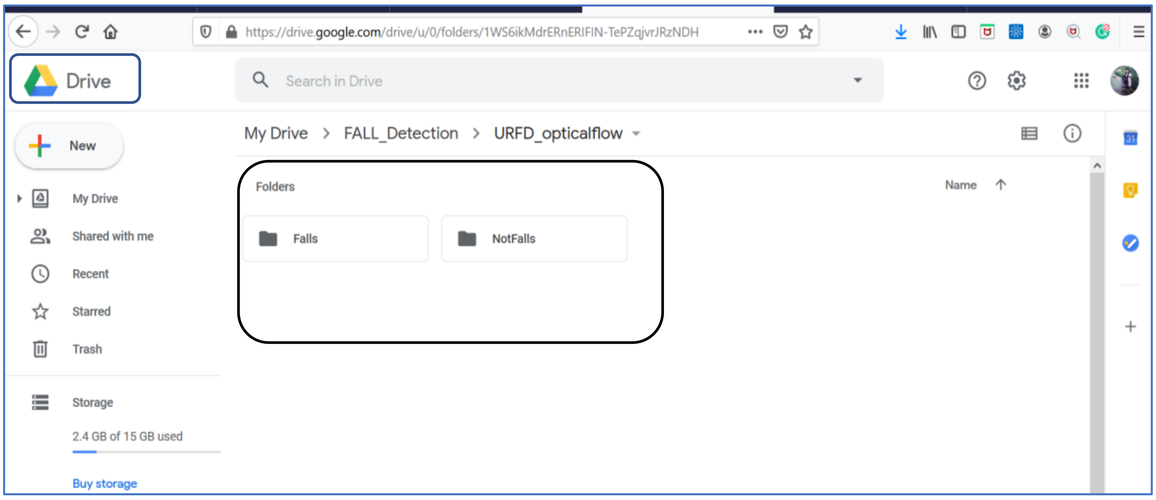
Two folders were created one folder named “URFD” (i.e., original UR-fall detection data downloaded from the website that consists of RGB images). Another folder named “UR_Fall_opticalflow” (i.e., empty folder that saves the optical flow images after generating the code). Also, the “URDF” has two separate folder called “Fall” and “Not-fall”. Where “Fall” folder contains image frames of fall and “Not-fall” folder contains Activities of daily life.



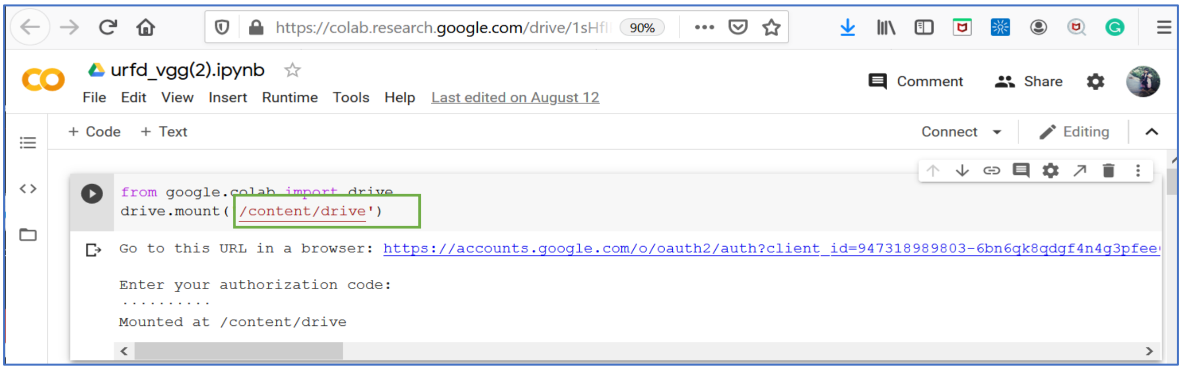
7. Google Colaboratory for building the classification model:

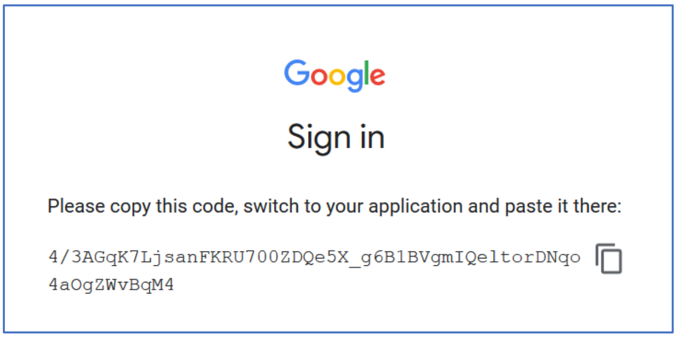
1. Upload the data folder on Google Drive:

   Once, the optical flow images of fall detection dataset were generated. It was uploaded on Google Drive.



2. The dataset uploaded to Google Drive. Then, the folders in drive were accessed through Mounting Google Drive locally. Then the drive can be accessed through the authorization code.





3. The images in the URFD_optical flow folder can be accessed through the following code.

```python
#Transfer 'jpg' images to an array IMG
def Dataset_loader(DIR, RESIZE, sigmaX=10):
  data_folder = DIR
  folders = [f for f in os.listdir(data_folder)]
  folders.sort()
  IMG = []

  for folder in folders:

    read = lambda imname: np.asarray(Image.open(imname).convert("RGB"))
    for IMAGE_NAME in (os.listdir(DIR+folder)):
        PATH = os.path.join(DIR+folder,IMAGE_NAME)
        _, ftype = os.path.splitext(PATH)
        if ftype == ".jpg":
            img = read(PATH)

            img = cv2.resize(img, (RESIZE,RESIZE))

            IMG.append(np.array(img))
    return IMG

fall_train = np.array(Dataset_loader('/content/drive/My Drive/FALL_Detection/URFD_opticalflow/Falls/',224))
not_fall_train = np.array(Dataset_loader('/content/drive/My Drive/FALL_Detection/URFD_opticalflow/NotFalls/',224))
```

4. Image augmentation performed using ImageDataGenerator

urfd_vgg(2).ipynb
File Edit View Insert Runtime Tools Help Last edited on August 12
Comment Share
+ Code + Text Connect Editing

```python
from keras.applications import vgg16, resnet50

BATCH_SIZE = 16

# Using original generator
train_generator = ImageDataGenerator(
        zoom_range=2,
        rotation_range = 90,
        horizontal_flip=True,
        vertical_flip=True,
    )
```

5. Assigning labels to folder contains optical flow images. Optical folder that was loaded into the drive contains two folders namely “Falls” and “Not-falls”. A balanced distribution of labels was created (NumPy array of 0’s and 1’s). An array of zeroes is used for labelling the fall optical flow images. Whereas, an array of ones is used for labelling the not-fall optical flow images. X contains attributes of falls and not-falls images and Y contains corresponding labels of falls and not-falls. Then, the data were shuffled, each time during training process, the data changes randomly.

urfd_vgg(2).ipynb
File Edit View Insert Runtime Tools Help Last edited on August 12
Comment Share
+ Code + Text Connect Editing

```python
# Create labels
fall_train_label = np.zeros(len(fall_train))
not_falltrain_label = np.ones(len(not_fall_train))

# Merge data
X = np.concatenate((fall_train, not_fall_train), axis = 0)
Y = np.concatenate((fall_train_label, not_falltrain_label), axis = 0)


from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.4,random_state=0)

# Shuffle train data
s = np.arange(X_train.shape[0])
np.random.shuffle(s)
X_train = X_train[s]
Y_train = Y_train[s]

# Shuffle test data
s = np.arange(X_test.shape[0])
np.random.shuffle(s)
X_test = X_test[s]
Y_test = Y_test[s]


# To categorical
Y_train = to_categorical(Y_train, num_classes= 2)
Y_test = to_categorical(Y_test, num_classes= 2)
```

6. The dataset was split into training set and testing set with a 60:40 ratio. Where 60% of data is randomly selected for training and 40% is for testing.

```
x_train, x_val, y_train, y_val = train_test_split(
    X_train, Y_train,
    test_size=0.4,random_state=0
)
```

8. Implementation of Deep Learning and Transfer Learning

8.1 Convolutional neural network used for implementing the architecture

```
def build_model(backbone, lr=1e-4):
    model = Sequential()
    model.add(backbone)
    model.add(layers.GlobalAveragePooling2D())
    model.add(layers.Dropout(0.5))
    model.add(layers.BatchNormalization())
    model.add(layers.Dense(2, activation='softmax'))


    model.compile(
        loss='binary_crossentropy',
        optimizer=Adam(lr=lr),
        metrics=['accuracy']
    )

    return model
```

8.2 Creating VGG-16

```
K.clear_session()
gc.collect()


vgg_model = vgg16.VGG16(weights="imagenet",
                        include_top=False,
                        input_shape=(224,224,3))

base_model = vgg16.VGG16
trainable_layers = 4

base_model = base_model(weights="imagenet", include_top=False, input_shape=(224,224,3))

# Freeze all but the last 4 layers
for layer in base_model.layers[:-trainable_layers]:
        layer.trainable = False


model = build_model(base_model ,lr = 1e-4)
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 7, 7, 512)         14714688
_________________________________________________________________
global_average_pooling2d (Gl (None, 512)               0
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
batch_normalization (BatchNo (None, 512)               2048
_________________________________________________________________
dense (Dense)                (None, 2)                 1026
=================================================================
Total params: 14,717,762
Trainable params: 7,081,474
Non-trainable params: 7,636,288
_________________________________________________________________
```

8.3 Creating ResNet-50

```
from keras.applications import vgg16, resnet50
K.clear_session()
gc.collect()

resnet = ResNet50(
    weights='imagenet',
    include_top=False,
    input_shape=(224,224,3)
)


base_model = resnet50.ResNet50
trainable_layers = 10

base_model = base_model(weights="imagenet", include_top=False, input_shape=(224,224,3))

# Freeze all but the last 4 layers
for layer in base_model.layers[:-trainable_layers]:
        layer.trainable = False


model = build_model(base_model ,lr = 1e-4)
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet50 (Functional)        (None, 7, 7, 2048)        23587712
_________________________________________________________________
global_average_pooling2d (Gl (None, 2048)              0
_________________________________________________________________
dropout (Dropout)            (None, 2048)              0
_________________________________________________________________
batch_normalization (BatchNo (None, 2048)              8192
_________________________________________________________________
dense (Dense)                (None, 2)                 4098
=================================================================
Total params: 23,600,002
Trainable params: 4,473,858
Non-trainable params: 19,126,144
_________________________________________________________________
```

8.4 Creating DenseNet-201

```
[ ] from keras.applications import vgg16, resnet50, densenet
    K.clear_session()
    gc.collect()

    densenet201 = DenseNet201(
        weights='imagenet',
        include_top=False,
        input_shape=(224,224,3)
    )



    base_model = densenet.DenseNet201
    trainable_layers = 10

    base_model = base_model(weights="imagenet", include_top=False, input_shape=(224,224,3))

    # Freeze all but the last 4 layers
    for layer in base_model.layers[:-trainable_layers]:
            layer.trainable = False


    model = build_model(base_model ,lr = 1e-4)
    model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
densenet201 (Functional)     (None, 7, 7, 1920)        18321984
_________________________________________________________________
global_average_pooling2d (Gl (None, 1920)              0
_________________________________________________________________
dropout (Dropout)            (None, 1920)              0
_________________________________________________________________
batch_normalization (BatchNo (None, 1920)              7680
_________________________________________________________________
dense (Dense)                (None, 2)                 3842
=================================================================
Total params: 18,333,506
Trainable params: 294,082
Non-trainable params: 18,039,424
```

9. Training the model for 10 epochs

```
[ ] history = model.fit_generator(
        train_generator.flow(x_train, y_train, batch_size=BATCH_SIZE),
        steps_per_epoch=x_train.shape[0] / BATCH_SIZE,
        epochs=10,
        validation_data=(x_val, y_val),
        callbacks=[learn_control, checkpoint]
    )
```

9.1 10-cross fold validation

```
[ ] def evaluate_model(X_train, X_val, y_train, y_val,model):

        model.fit(X_train, y_train, validation_data = (X_val,y_val), epochs=1, batch_size=16, verbose=2)#, callbacks = callbacks)
        _, val_acc = model.evaluate(X_val, y_val, verbose = 1)
        return  model,val_acc


    n_folds = 10
    cv_scores, model_history = list(), list()
    for _ in range(n_folds):
        # evaluate model
        model, test_acc = evaluate_model(x_train, x_val, y_train, y_val,model)
        print('>%.3f' % test_acc)
        cv_scores.append(test_acc)
        model_history.append(model)

    print('Estimated Accuracy %.3f (%.3f)' % (np.mean(cv_scores), np.std(cv_scores)))
```

9.2 Testing Data

```
[ ] Y_pred = model.predict(X_test)

[ ] accuracy_score(np.argmax(Y_test, axis=1), np.argmax(Y_pred, axis=1))
```

```
Y_pred = model.predict(X_test)

tta_steps = 10
predictions = []

for i in tqdm(range(tta_steps)):
    preds = model.predict_generator(train_generator.flow(X_test, batch_size=BATCH_SIZE, shuffle=False),
                                    steps = len(X_test)/BATCH_SIZE)

    predictions.append(preds)
    gc.collect()

Y_pred_tta = np.mean(predictions, axis=0)
```

References

- Dataset Source: http://fenix.univ.rzeszow.pl/~mkepski/ds/uf.html
- Optical flow extractor tool: https://github.com/yjxiong/dense_flow/tree/opencv-3.1
- Code reference for CNN: https://keras.io/guides/sequential_model/,
- https://www.programcreek.com/python/example/89688/keras.layers.GlobalAveragePooling2D
- https://www.tensorflow.org/api_docs/python/tf/keras/applications/VGG16
- https://keras.io/api/applications/densenet/
- https://towardsdatascience.com/exploring-confusion-matrix-evolution-on-tensorboard-e66b39f4ac12