A Project Report
On

VIDEO QUALITY ASSESSMENT USING DEEP LEARNING

BY

BHAVANI BHAMIDIPATY 2013A7PS186H

RUSHABH GANDHI 2014A3PS352H

Under the supervision of

Prof. ANAND M. NARSIMHAMURTHY

SUBMITTED IN FULFILLMENT OF THE REQUIREMENTS OF

CS F266: Study Oriented Project

CS F366: Lab Oriented Project

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI

HYDERABAD CAMPUS

(MAY 2017)

Acknowledgement

We would like to thank Prof. Anand Narasimhamurthy, Associate Professor, Birla Institute of Technology and Science Pilani, Hyderabad Campus, for having given us the opportunity to work on this project under his guidance. Neural networks require high-end hardware to run efficiently which we could have no way of obtaining if not for Prof. K.C.S. Murthy and Prof. Abhishek Thakur. We would thank them for providing access to hardware resources.

We are also grateful to the administration of BITS-Pilani Hyderabad for providing us the chance for the students to develop their academic skills and logical thinking through open ended research activities.

Birla Institute of Technology and Science-Pilani,

Hyderabad Campus

CERTIFICATE

This is to certify that the project report entitled “VIDEO QUALITY ASSESSMENT USING DEEP LEARNING” submitted by Bhavani Bhamidipaty (ID No. 2013A7PS186H) and RUSHABH GANDHI (ID No. 2014A3PS352H) in fulfillment of the requirements of the course CS F266, Study Oriented Project Course and, embodies the work done by them under my supervision and guidance.

Date: (ANAND M. NARSIMHAMURTHY)

BITS- Pilani, Hyderabad Campus

Abstract

Deep learning is an advanced branch of machine learning which has evolved significantly and publicly fairly recently. It has a wide range of applications such as computer vision, financial prediction and natural language processing to name a few.

With the advent of intelligent machine learning techniques, it has now become possible for computers to mimic human vision with image processing and deep learning albeit with errors. It is the aim of this project to evaluate video distortion level. This would be done with Deep Neural Networks and a dataset trained on a collection of stock videos.

The primary dataset used for the project is the LIVE video dataset from UT Austin. Convolutional neural networks under consideration are 3D Convo Convolutional neural network and VGGNet which is a kind of deep neural network (DNN) for image classification.

Table Of Contents

Installation And Setup

Conda and pip

Anaconda is an easy-to-install free package manager, environment manager, Python distribution, and collection of over 720 open source packages offering free community support. The project code is in python and uses a number of libraries like openCV, TensorFlow, Keras, SciPy, NumPy and scikit-learn. To allow robust and compatible installation of these libraries and integrating them with the code, I installed anaconda.

It contains most of the machine learning (ML) and media processing libraries that may be required. Anaconda can be installed from a windows installer as well as by command-line. After that installing a library can be done by the command

```
conda install
```

It also covers any dependencies needed for the installation of other packages. Any package not found under the umbrella of conda has conventionally been found in the metadata lists of pip which is a package management system used to install and manage software packages written in Python.

TensorFlow, Theano and Keras

A number of Deep Learning libraries and packages were researched such as Caffe, Torch, TensorFlow, etc. All these are specially engineered for Deep Learning computations. It was found that Keras with a TensorFlow backend was the best choice for this project.

TensorFlow is an open source software library for ML and meets any needs for systems capable of building and training neural networks to detect and decipher patterns and correlations, analogous to the learning and reasoning which humans use. It is built for Python and has a large support community. It is also a reputed package as a number of top companies constantly work on optimizing it better (Google Engineers Group among them).

Keras is built on top of Theano and TensorFlow and provides an easy interface for quick high level prototyping. Keras uses numpy and scipy, pyyaml, HDF5 and h5py (optional, required if you use model saving/loading functions),cuDNN (optional but recommended if you use CNN) as dependencies.

Video processing is a space-expensive activity and so would require the GPU ram to be fully utilized. As the machine used for the project has an nVidia graphic card, the CUDA library drivers were best

suited as they abstract away the GPU dependence to give smooth computation. The cuDNN library provides fast and efficient functions for Deep Learning algorithms.

For the installation of these packages, conda command is fully equipped. It takes cares to install all dependencies for TensorFlow and Keras.
The conda command takes care of installing all the dependencies required for installation of Theano.

MATLAB

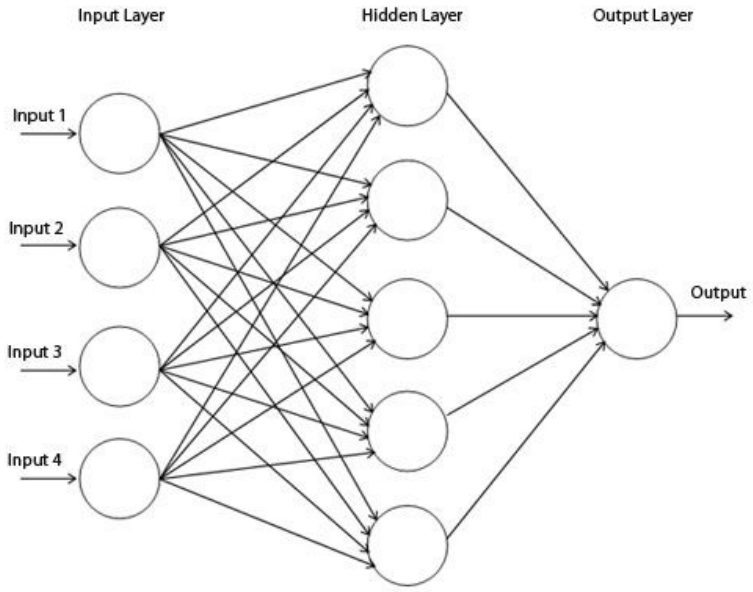
MATLAB (matrix laboratory) is a multi-paradigm numerical computing environment and fourth-generation programming language. A proprietary programming language developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including Python.

The LIVE video data is given in YUV2(4:2:0) format which allows for maximum data compression. Although this is optimal for file transfer over the internet, it is harder to extract the video from the bytestream. For this purpose MATLAB would be used to convert the YUV data format to AVI data format to allow feature analysis.

MATLAB is fairly straight-forward to install from their website after purchasing a Student Edition License and downloading the packages.

Artificial Neural Network

Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated.



Neural networks help us cluster and classify. You can think of them as a clustering and classification layer on top of data you store and manage. They help to group unlabeled data according to similarities among the example inputs, and they classify data when they have a labeled dataset to train on. ANNs are considered nonlinear statistical data modeling tools where the complex relationships between inputs and outputs are modeled or patterns are found.

ANNs have three layers that are interconnected. The first layer consists of input neurons. Those neurons send data on to the second layer, which in turn sends the output neurons to the third layer. Training an artificial neural network involves choosing from allowed models for which there are several associated algorithms.

Deep Learning

Deep Learning is a part of machine learning which is exceptionally effective at learning patterns. It utilizes a learning algorithms that derive the meaning out of data that are not conventionally visible

to theoretical models, by using a hierarchy of multiple layers that mimic the neural networks of our brain. On inputting massive data these models tend to understand the data and respond in many useful ways.

What distinguishes Deep Learner from traditional learner is that learned features of the huge data input are passed through different layers of abstraction. It employs a series of nonlinear transformations in each of the hierarchical predictors. Every transformation is a layer whose input is output from its previous layer and its output is input to next layer. Depth of the architecture is defined as the number of layers in the model.

Deep Neural Network

Deep-learning networks are distinguished from the more commonplace single-hidden-layer neural networks by their depth; that is, the number of node layers through which data passes in a multistep process of pattern recognition.

Traditional machine learning relies on shallow nets, composed of one input and one output layer, and at most one hidden layer in between. More than three layers (including input and output) qualifies as “deep” learning. So deep is a strictly defined, technical term that means more than one hidden layer.

In deep-learning networks, each layer of nodes trains on a distinct set of features based on the previous layer’s output. The further you advance into the neural net, the more complex the features your nodes can recognize, since they aggregate and recombine features from the previous layer.

Since the more layers in a neural network, the more accurate the predictions, deep NN is the perfect choice for this project. The network would be trained on the data derived from the LIVE dataset using cost calculation which would give an accuracy rate of prediction.

Image and Video Classification

Image classification is a field where CNNs have been extensively used to learn a wide range of features such as cars, hats, people, actions, etc. into a model. Input images are then fed into the model and features are extracted and regressions are performed to predict the object or feature at certain accuracy levels.

For video classification, RNNS, LSTMs, GRUs, CNNs can all be used for various levels of expertise, problem fields and accuracy/efficiency levels. One basic technique which has shown promise on using different architectures is deep CNNs. One approach to consider is a 3D Convolution Network. Deep 3D Convolutional Neural Networks (3D-CNN) are traditionally used for object recognition, video data analytics and human gesture recognition.

Depending on the degree of information that is available from the original video as a reference in the quality assessment, the objective methods are further divided into full reference (FR), reduced reference (RR), and no-reference (NR) as follows:

- FR methods: With this approach, the entire original image/video is available as a reference. Accordingly, FR methods are based on comparing distorted image/video with the original image/video.

- RR methods: In this case, it is not required to give access to the original image/video but only to provide representative features about texture or other suitable characteristics of the original image/video. The comparison of the reduced information from the original image/video with the corresponding information from the distorted image/video provides the input for RR methods.

- NR methods: This class of objective quality methods does not require access to the original image/video but searches for artifacts with respect to the pixel domain of an image/video, utilizes information embedded in the bitstream of the related image/video format, or performs quality assessment as a hybrid of pixel-based and bitstream-based approaches.

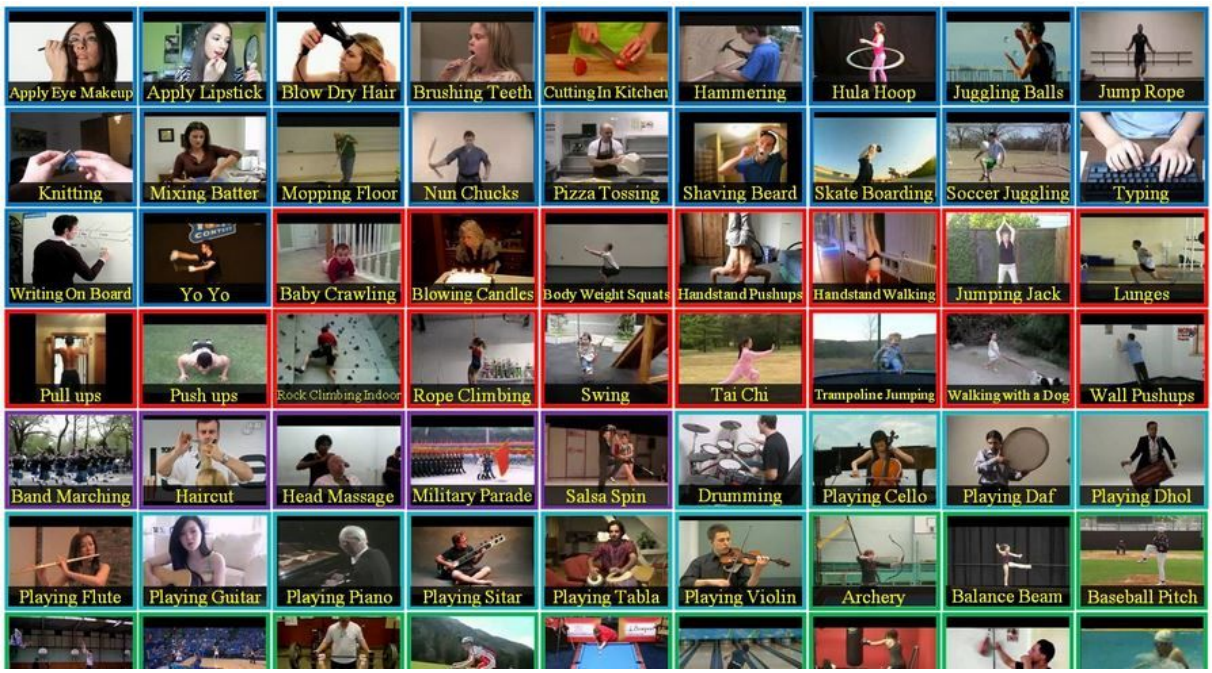
UCF 101, LIVE dataset and Transfer Learning

CONVOLUTION 3D and TRANSFER LEARNING

One of techniques used for training convolutional neural networks on video data is Convolution3D (C3D) which uses 3d Convolutions on stacks of frames of video data. This works by stacking the three dimensional video frames on top of one another in a fourth dimension and use convolutional filters of 3D spatial dimensions to create activation maps/ feature maps that effectively capure generic spatiotemporal features of the videos related to objects, actions, scenes, and other frequently related categories . Using C3D features and simple linear SVM on top of it, this method achieved state of the art performance on scene classification (96.7% on YUPENN and 77.7% on Maryland), object classification (15.3% on egocentric object) and action similarity labeling (72.9% on ASLAN) problems in the video domain. It was also the most accurate method that did not use optical flow on the action recognition data set UCF 101.

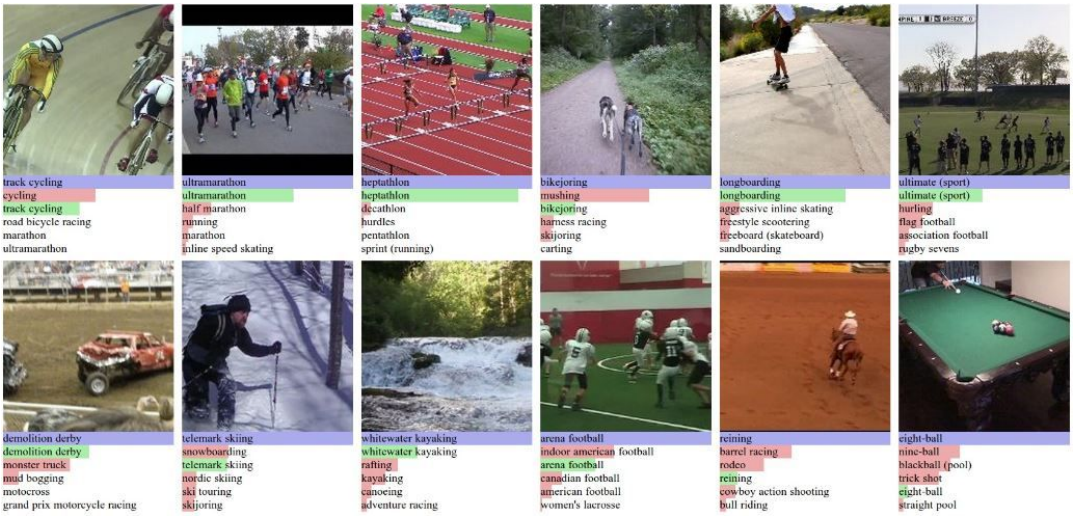
For this project we largely used three datasets - UT Austin's LIVE video quality assessment dataset, UCF 101 action recognition dataset and SPORTS 1M action recognition dataset.

UT Austin's LIVE video quality assessment data set was the primary data under consideration and consists of video sample data with the corresponding subject score (mean opinion score - MOS) along with its standard deviation. The motive behind taking subjective scores was to make quality predictions that are *in agreement* with subjective opinion of human observers as one of the main application scopes of the project would be to in QA research.



UCF 101 action recognition dataset and SPORTS 1M action recognition dataset are popular for their extensive classification and sheer quantity of data. One of the advantages of using CNNs is how

seamlessly transfer learning allows us to retrain an existing model. By using these datasets, we take advantage of local spatio-temporal information and suggest a multiresolution, foveated architecture on top of the existing model as a promising way of speeding up the training.



Deep Learning has not been used to tackle the problem of video quality assessment as yet. Also the data available for Video Quality Assessment (UT Austin's LIVE data set, for example) is not large enough to train a deep Convolutional Neural Network from scratch. Further, the amount of computational resources required to train a modest sized CNN model on video data is quite large. In order to work around all of these points we used transfer learning to leverage the power of C3D features to work for the Video Quality Assessment task.

We found an open sourced version of C3D model that had a pretrained tensorflow model trained on Sports 1M dataset and fine-tuned on UCF 101. Typically for transfer learning there are two possible approaches that one might take. The first one is that we consider the weights of the pretrained model as trainable weights. This method considers the pretrained model as initialization of the new model to be trained and these weights are fine-tuned using backpropogation. Though this method works slightly better, it very computationally expensive. So we resorted to the second method of transfer learning which used the weights of the pretrained model as a fixed feature extractor. A new fully connected shallow network is trained on top of these features and only this freshly added shallow neural network's weights are trained using backprop.

Approach

LIVE video dataset provides specifications of each video as a YUV file which basically stores a bitstream of each frame of the video. Using this, each video was converted into .avi format and from there a number of preprocessing activities were performed. 16 equally spaced out frames were extracted and labelled with the corresponding MOS and its standard deviation. As the model takes in a specific frame resolution, the frames were resized to that giving a slightly distorted image to the human eye. Each frame was fed as 3 channels of data to the model for the 3 color channels (RGB).

So the input to the model consists of n data units i.e. video input. Each video input consists of 16 frames X (112 X 112) dimensions X 3 color channels. Additionally, each video's label is fed along with it which contains the unique id of the video hierarchically, its MOS and standard deviation.

Each video was fed to the the C3D model and generic features were extracted using forward propagation. We took isolated the 4096 dimensional feature representation of each video from the penultimate layer and trained a two layered fully connected ANN on top of it with a 512 nodal hidden layer all converging to an output layer of a single node that was trained to output the Mean Opinion Score (MOS) of the video. In earlier experiments the final layer had two nodes to output the standard deviation of the MOS as well. However it was realized later that standard deviation is not a natural property of the video itself but the uncertainty in the subjective opinions of each person involved in the data annotation process.

The fully connected net was trained using RMSprop optimizer using a Mean Squared Error as the loss function. In order to try to reduce overfitting, dropouts were used with a probability of 0.5. Of the 150 videos present in the dataset, random 10 videos and their corresponding labels were kept aside for testing purposes and the shallow net was trained on 140 of these videos. Training and testing accuracies were compared to get a feel of the bias variance trade off in the model.

Methodology

1. Research and review of literature
    - Literature reviewed so far have been listed in BIBLIOGRAPHY
2. Extract video frames from YUV files
    - This has been done using MATLAB (code given in APPENDIX)
3. Compress and extract video skips into .AVI files
    - (Also in APPENDIX)
4. Create model to suit to the LIVE data set model (Focused on regression)
5. FULL REFERENCE
    - Subtraction Matrix of Original and Distorted Video must be fed into pre-trained model to give score with accuracy level.
6. NO REFERENCE
    - Only distorted video fed to the pre-trained model.

Results

1) The pretrained model performed well on UCF 101 as expected with an accuracy of 76 percent.
2) 2D CNNs performed well on image quality assessment data set.
3) Using transfer learning the training set mean squared error went down to as low as 0.8.
4) However the mean squared error on the test data did not come down and remained as high at 83.

Conclusion and Further Scope

The huge difference in the training and test errors clearly reflect that the model is overfitting on the small LIVE data set, even after the use of drop out. Also it might mean that the model is not able to capture enough information about the videos needed to perform the subtle task of subjective video quality assessment.

Investigation of all of the following options might make the model generalize to a better extent:

1) Ensemble learning methods: Ensembles can be formed in two ways. One way would be to sample different locations of the video frames and average out the results from these separate models. The other method would be to get different 16 sample frames from different time instances for each video. This will help in data augmentation and hopefully reduce overfitting.
2) Reducing the number of parameters in the fully connected Neural network trained on top of the 4096 dimensional features. Reducing the number of parameters always reduces overfitting, The neural network can be made thinner and less deeper. Instead of choosing a neural network on top, support vector regression can also be explored since C3D+SVM is known to give good results.
3) Fine tuning the pretrained model: The pretrained model is trained on a million videos while the live data set contains just 150 videos. So freezing all the weights of the pretrained model is not a good idea. Also fine tuning the layers might also support addition of batch norm and dropouts in the pretrained model that will add to the regularization. However computational expense of this method is quite high.
4) A different pretrained model: Though C3D is a good model for videos, better ones have come out that use optical flow which currently hold the state of the art accuracy. Using those for the pretrained model can also be considered.
5) Training from scratch instead of transfer learning. Possible only with very high end computational resources.

Appendix

Part A

loadFileYuv.m

```
% function mov = loadFileYuv(fileName, width, height, idxFrame)
function [mov,imgRgb] = loadFileYuv(fileName, width, height, idxFrame)
% load RGB movie [0, 255] from YUV 4:2:0 file
% idxFrame is index of Frame to extract
fileId = fopen(fileName, 'r');

subSampleMat = [1, 1; 1, 1];
nrFrame = length(idxFrame);
disp(nrFrame)
for f = 1 : 1 : nrFrame
    % search fileId position
    sizeFrame = 1.5 * width * height;
    fseek(fileId, (idxFrame(f) - 1) * sizeFrame, 'bof');

    % read Y component
    buf = fread(fileId, width * height, 'uchar');
    imgYuv(:, :, 1) = reshape(buf, width, height).'; % reshape

    % read U component
    buf = fread(fileId, width / 2 * height / 2, 'uchar');
    imgYuv(:, :, 2) = kron(reshape(buf, width / 2, height / 2).', subSampleMat); % reshape and upsample

    % read V component
    buf = fread(fileId, width / 2 * height / 2, 'uchar');
    imgYuv(:, :, 3) = kron(reshape(buf, width / 2, height / 2).', subSampleMat); % reshape and upsample

    % normalize YUV values
    % imgYuv = imgYuv / 255;

    % convert YUV to RGB

    imgRgb = reshape(convertYuvToRgb(reshape(imgYuv, height * width, 3)), height, width, 3);
    % imgRgb = ycbcr2rgb(imgYuv);
    name = ['ActualBackground' fileName(3) '_' num2str(idxFrame) '.jpg'];
    imwrite(imgRgb,name,'jpg');
    mov(f) = im2frame(imgRgb);
    %   mov(f).cdata = uint8(imgRgb);
    %   mov(f).colormap =  [];
    %     imwrite(imgRgb,'ActualBackground.bmp','bmp');

    figure, imshow(imgRgb);
    %Image = imread(name, 'bmp');
    %figure, imshow(Image);
end
fclose(fileId);
```

img2Video.m

```
outputVideo = VideoWriter('shuttle_out.avi');
outputVideo.FrameRate = 1;
open(outputVideo);


for ii = 1:5
   disp(ii)
   nameFile = ['ActualBackground1_' num2str(ii) '.jpg'];
   img = imread(nameFile,'jpg');
   writeVideo(outputVideo,img)
end
close(outputVideo);


implay('shuttle_out.avi')
```

convertYuvToRgb.m

```
function rgb = convertYuvToRgb(yuv)
% convert row vector YUV [0, 255] in row vector RGB [0, 255]

load conversion.mat; % load conversion matrices

yuv = double(yuv);
%disp(yuv);
yuv(:, 2 : 3) = yuv(:, 2 : 3) - 127;
rgb = (yuvToRgb *yuv.').';

rgb = uint8(clipValue(rgb, 0, 255));
```

clipValue.m

```
function val = clipValue(val, valMin, valMax)
% check if value is valid

for i = 1 : 1 : size(val(:))
    if val(i) < valMin
        val(i) = valMin;
    elseif val(i) > valMax
        val(i) = valMax;
    end
end
```

Part B

ProcessVideoToFrames.ipynb

```
def extractAndSaveFrames(name,skip):
    if(name[3] == '_'):
        rank = int(name[2])
    else:
        rank = int(name[2:4])
    print(rank)

    rankflag = 0

    typea = name[0:2]
    if(rank/10 < 1):
        rankflag = 2
    else:
        rankflag = 3
    fps = int(str(name[2 + rankflag: 4 + rankflag]))
    length = getVidDur(str(name))

    i=0
    num = 1
    toskip = int(length/skip)
    for num in range(1,int(length),toskip):
        vidcap = cv2.VideoCapture(name)
        vidcap.set(cv2.CAP_PROP_POS_MSEC,(num/fps)*1000)
        success,image = vidcap.read()
        if success:
            cv2.imwrite(typea + str(rank) + '_' + str(i) + ".jpg", image)
        i = i + 1
        if(i == skip):
            break
```

```
def extractAll(genre,fps,numFrames):
        rest = 'fps.avi'
        for i in range(1,numFrames+1):
            filename = genre + str(i) + '_' + str(fps) + rest
            extractAndSaveFrames(filename,numFrames)
```

Part C

c3d_model.py

```python
def conv3d(name, l_input, w, b):
    return tf.nn.bias_add(
                tf.nn.conv3d(l_input, w, strides=[1, 1, 1, 1, 1], padding='SAME'),
                b
                )

def max_pool(name, l_input, k):
    return tf.nn.max_pool3d(l_input, ksize=[1, k, 2, 2, 1], strides=[1, k, 2, 2, 1], padding

def inference_c3d(_X, _dropout, batch_size, _weights, _biases):

    # Convolution Layer
    conv1 = conv3d('conv1', _X, _weights['wc1'], _biases['bc1'])
    conv1 = tf.nn.relu(conv1, 'relu1')
    pool1 = max_pool('pool1', conv1, k=1)

    # Convolution Layer
    conv2 = conv3d('conv2', pool1, _weights['wc2'], _biases['bc2'])
    conv2 = tf.nn.relu(conv2, 'relu2')
    pool2 = max_pool('pool2', conv2, k=2)

    # Convolution Layer
    conv3 = conv3d('conv3a', pool2, _weights['wc3a'], _biases['bc3a'])
    conv3 = tf.nn.relu(conv3, 'relu3a')
    conv3 = conv3d('conv3b', conv3, _weights['wc3b'], _biases['bc3b'])
    conv3 = tf.nn.relu(conv3, 'relu3b')
    pool3 = max_pool('pool3', conv3, k=2)

    # Convolution Layer
    conv4 = conv3d('conv4a', pool3, _weights['wc4a'], _biases['bc4a'])
    conv4 = tf.nn.relu(conv4, 'relu4a')
    conv4 = conv3d('conv4b', conv4, _weights['wc4b'], _biases['bc4b'])
    conv4 = tf.nn.relu(conv4, 'relu4b')
    pool4 = max_pool('pool4', conv4, k=2)

    # Convolution Layer
    conv5 = conv3d('conv5a', pool4, _weights['wc5a'], _biases['bc5a'])
    conv5 = tf.nn.relu(conv5, 'relu5a')
    conv5 = conv3d('conv5b', conv5, _weights['wc5b'], _biases['bc5b'])
    conv5 = tf.nn.relu(conv5, 'relu5b')
    pool5 = max_pool('pool5', conv5, k=2)

    # Fully connected layer
    pool5 = tf.transpose(pool5, perm=[0,1,4,2,3])
    dense1 = tf.reshape(pool5, [batch_size, _weights['wd1'].get_shape().as_list()[0]]) # Res
    dense1 = tf.matmul(dense1, _weights['wd1']) + _biases['bd1']

    dense1 = tf.nn.relu(dense1, name='fc1') # Relu activation
    dense1 = tf.nn.dropout(dense1, _dropout)

    dense2 = tf.nn.relu(tf.matmul(dense1, _weights['wd2']) + _biases['bd2'], name='fc2') # F
    dense2 = tf.nn.dropout(dense2, _dropout)

    # Output: class prediction
    out = tf.matmul(dense2, _weights['out']) + _biases['out']

    #out_regression=tf.matmul(dense2,_weights['out2'])+_biases['out2']



    return dense2
```

TransferLearning.ipynb

```
def placeholder_inputs(batch_size):

    images_placeholder = tf.placeholder(tf.float32, shape=(batch_size,
                                                           c3d_model.NUM_FRAMES_PER_CLIP,
                                                           c3d_model.CROP_SIZE,
                                                           c3d_model.CROP_SIZE,
                                                           c3d_model.CHANNELS))
    labels_placeholder = tf.placeholder(tf.int64, shape=(batch_size))
    return images_placeholder, labels_placeholder

def _variable_on_cpu(name, shape, initializer):
  #with tf.device('/cpu:%d' % cpu_id):
    with tf.device('/cpu:0'):
        var = tf.get_variable(name, shape, initializer=initializer)
    return var
def _variable_with_weight_decay(name, shape, stddev, wd):
    var = _variable_on_cpu(name, shape, tf.truncated_normal_initializer(stddev=stddev))
    if wd is not None:
        weight_decay = tf.multiply(tf.nn.l2_loss(var), wd, name='weight_loss')
        tf.add_to_collection('losses', weight_decay)
        return var

#def _get_new_variable(name,shape,initializer)

```

In [56]:

```
model = Sequential()
model.add(Dense(512, input_dim=4096))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.compile(optimizer='rmsprop',
              loss='mse')
model.fit(X_train,labels_train, epochs=1000,batch_size=16)
```

```
Epoch 18/1000
140/140 [==============================] - 0s - loss: 68.4689
Epoch 19/1000
140/140 [==============================] - 0s - loss: 55.4590
Epoch 20/1000
140/140 [==============================] - 0s - loss: 77.7548
Epoch 21/1000
140/140 [==============================] - 0s - loss: 25.3932
Epoch 22/1000
140/140 [==============================] - 0s - loss: 51.2619
Epoch 23/1000
140/140 [==============================] - 0s - loss: 58.2461
Epoch 24/1000
140/140 [==============================] - 0s - loss: 50.4743
Epoch 25/1000
140/140 [==============================] - 0s - loss: 43.3520
Epoch 26/1000
140/140 [==============================] - 0s - loss: 44.1089
Epoch 27/1000
```

Bibliography

- Muhammad Shahid, Andreas Rossholm, Benny Lövström and Hans-Jürgen Zepernick *No-reference image and video quality assessment: a classification and review of recent approaches* http://jivp.eurasipjournals.springeropen.com/articles/10.1186/1687-5281-2014-40

- Yue-Hei Ng, J., Hausknecht, M., Vijayanarasimhan, S., Vinyals, O., Monga, R., & Toderici, G. (2015). *Beyond short snippets: Deep networks for video classification*. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 4694-4702).

- Chollet, F. (2016, July 05). *Building powerful image classification models using very little data. Retrieved from Keras Blog* https://blog.keras.io/building-powerful-image-classificationmodels-using-very-little-data.html

- A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. *Large-scale video classification with convolutional neural networks*. In Proc. CVPR http://vision.stanford.edu/pdf/karpathy14.pdf

- Basura Fernando, Stephen Gould *Learning End-to-end Video Classification with Rank-Pooling* http://jmlr.org/proceedings/papers/v48/fernando16.pdf

- Matthew D. Zeiler and Rob Fergus *Visualizing and Understanding Convolutional Networks* https://www.cs.nyu.edu/~fergus/papers/zeilerECCV2014.pdf

- H.R. Sheikh, Z.Wang, L. Cormack and A.C. Bovik, *LIVE Image Quality Assessment Database Release 2* https://live.ece.utexas.edu/research/quality

- H.R. Sheikh, M.F. Sabir and A.C. Bovik, *A statistical evaluation of recent full reference image quality assessment algorithms, IEEE Transactions on Image Processing,* vol. 15, no. 11, pp. 3440-3451, Nov. 2006.

- Z. Wang, A.C. Bovik, H.R. Sheikh and E.P. Simoncelli, *Image quality assessment: from error visibility to structural similarity*, *IEEE Transactions on Image Processing*, vol.13, no.4, pp. 600- 612, April 2004