

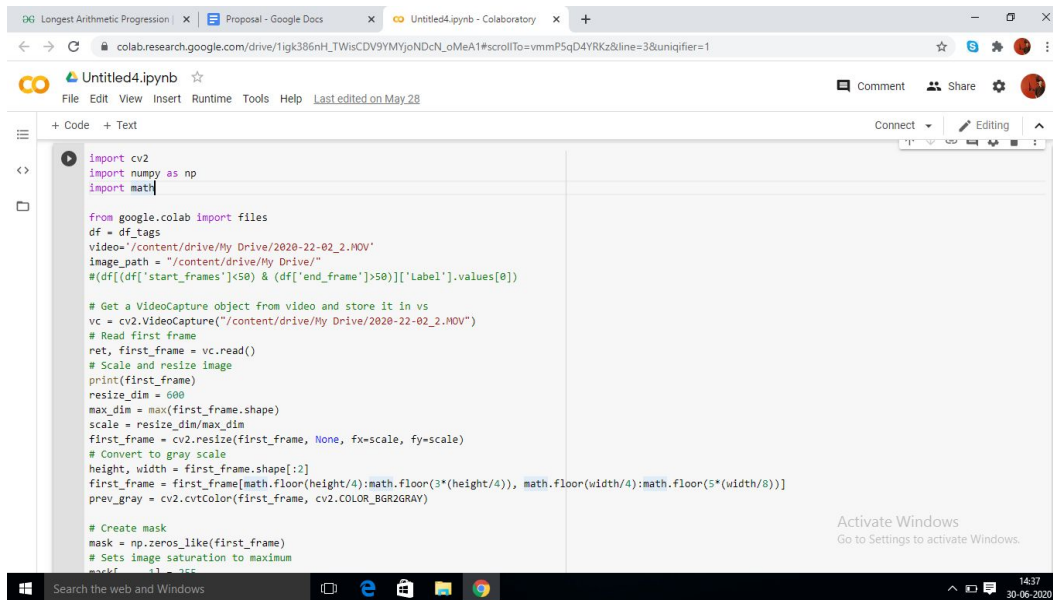
Problem Statement

Objective:

Crop out the clips from the provided videos where batsmen played a shot through supervised computer vision models.

Steps:

1. Calculate Optical Flow between consecutive frames(at a difference of $\frac{1}{2}$ seconds).



```
import cv2
import numpy as np
import math

from google.colab import files
df = df_tags
video = "/content/drive/My Drive/2020-22-02_2.HOV"
image_path = "/content/drive/My Drive/"
#(df[(df['start_frames']<50) & (df['end_frame']>50)]['Label'].values[0])

# Get a VideoCapture object from video and store it in vs
vc = cv2.VideoCapture("/content/drive/My Drive/2020-22-02_2.HOV")
# Read first frame
ret, first_frame = vc.read()
# Scale and resize image
print(first_frame)
resize_dim = 600
max_dim = max(first_frame.shape)
scale = resize_dim/max_dim
first_frame = cv2.resize(first_frame, None, fx=scale, fy=scale)
# Convert to gray scale
height, width = first_frame.shape[:2]
first_frame = first_frame[math.floor(height/4):math.floor(3*(height/4)), math.floor(width/4):math.floor(5*(width/8))]
prev_gray = cv2.cvtColor(first_frame, cv2.COLOR_BGR2GRAY)

# Create mask
mask = np.zeros_like(first_frame)
# Sets image saturation to maximum
mask[0:1] = 255
```

Longest Arithmetic Progression | Proposal - Google Docs | Untitled4.ipynb - Colaboratory

colab.research.google.com/drive/1gk386nH_TWiScDV9YMYjoNDcN_oMeA1#scrollTo=vmmP5qD4YRKz&line=38uniquifier=1

Untitled4.ipynb

File Edit View Insert Runtime Tools Help Last edited on May 28

+ Code + Text

```
# Sets image saturation to maximum
mask[... , 1] = 255
# cv2.imshow('a', first_frame)
# cv2.waitKey(5000)
# cv2.destroyAllWindows()
video = video.split('/')[-1]
#print (video)
#out = cv2.VideoWriter('video.mp4',-1,1,(600, 600))
images = []
labels = []
success = True
fps = int(vc.get(cv2.CAP_PROP_FPS))
count = 1
prev_frame = count
curr_frame = count
folders = ["shots", "non_shots"]
clip_prefix = "clip_"
subfolder_fix = "/"
clip_ext = ".jpg"
loc = 0
#path = os.getcwd()
#for folder in folders:
#    if not os.path.exists(os.path.join(path, folder)):
#        os.makedirs(os.path.join(path, folder), exist_ok=True)
while(success):
    # Read a frame from video
    success, frame = vc.read()
    if success == False:
        break
```

Activate Windows
Go to Settings to activate Windows.

Search the web and Windows

Longest Arithmetic Progression | Proposal - Google Docs | Untitled4.ipynb - Colaboratory

colab.research.google.com/drive/1gk386nH_TWiScDV9YMYjoNDcN_oMeA1#scrollTo=vmmP5qD4YRKz&line=38uniquifier=1

Untitled4.ipynb

File Edit View Insert Runtime Tools Help Last edited on May 28

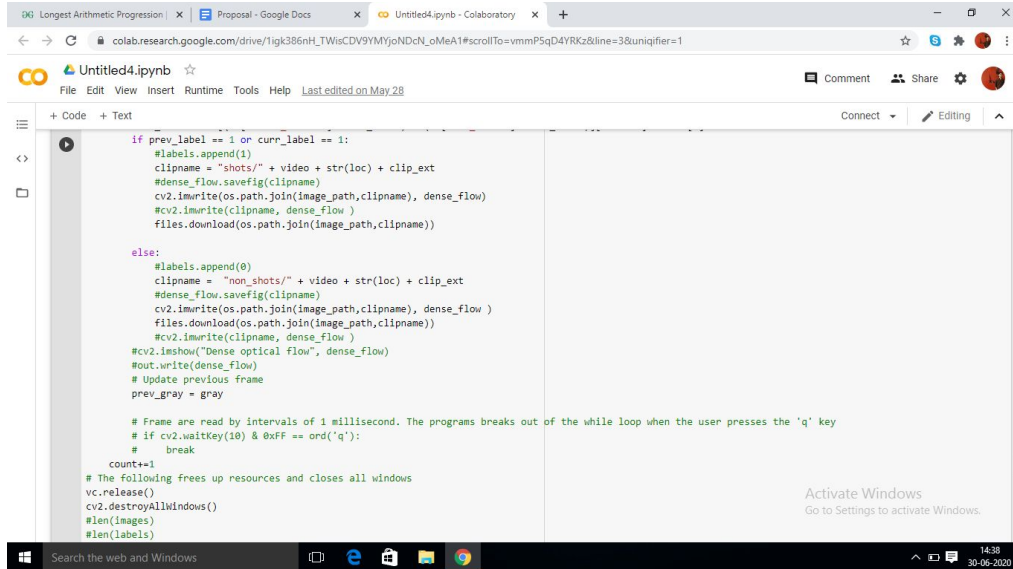
+ Code + Text

```
elif count%(fps/2) == 0:
    # Convert new frame format's to gray scale and resize gray frame obtained
    loc = loc+1
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    gray = cv2.resize(gray, None, fx=scale, fy=scale)
    gray = gray[math.floor(height/4):math.floor(3*(height/4)), math.floor(width/4):math.floor(5*(width/8))]
    #prev_gray = prev_gray[0:height, math.floor(width/4):math.floor(3*(width/4))]
    # Calculate dense optical flow by Farneback method
    # https://docs.opencv.org/3.0-beta/modules/video/doc/motion_analysis_and_object_tracking.html#calcopticalflowfarneback
    flow = cv2.calcOpticalFlowFarneback(prev_gray, gray, None, pyr_scale = 0.5, levels = 5, winsize = 11, iterations = 5, poly_n = 5, poly_sigma = 1.1, flags
    # Compute the magnitude and angle of the 2D vectors
    magnitude, angle = cv2.cartToPolar(flow[... , 0], flow[... , 1])
    # Set image hue according to the optical flow direction
    mask[... , 0] = angle * 180 / np.pi
    # Set image value according to the optical flow magnitude (normalized)
    mask[... , 2] = cv2.normalize(magnitude, None, 0, 255, cv2.NORM_MINMAX)
    # Convert HSV to RGB (BGR) color representation
    rgb = cv2.cvtColor(mask, cv2.COLOR_HSV2BGR)

    # Resize frame size to match dimensions
    frame = cv2.resize(frame, None, fx=scale, fy=scale)
    frame = frame[math.floor(height/4):math.floor(3*(height/4)), math.floor(width/4):math.floor(5*(width/8))]
    # Open a new window and displays the output frame
    dense_flow = cv2.addWeighted(frame, 1, rgb, 2, 0)
    #images.append(dense_flow)
    prev_frame = curr_frame
    curr_frame = count
    prev_label = df[(df['start_frames']<prev_frame) & (df['end_frame']>prev_frame)]['Label'].values[0]
    curr_label = df[(df['start_frames']<curr_frame) & (df['end_frame']>curr_frame)]['Label'].values[0]
```

Activate Windows
Go to Settings to activate Windows.

Search the web and Windows



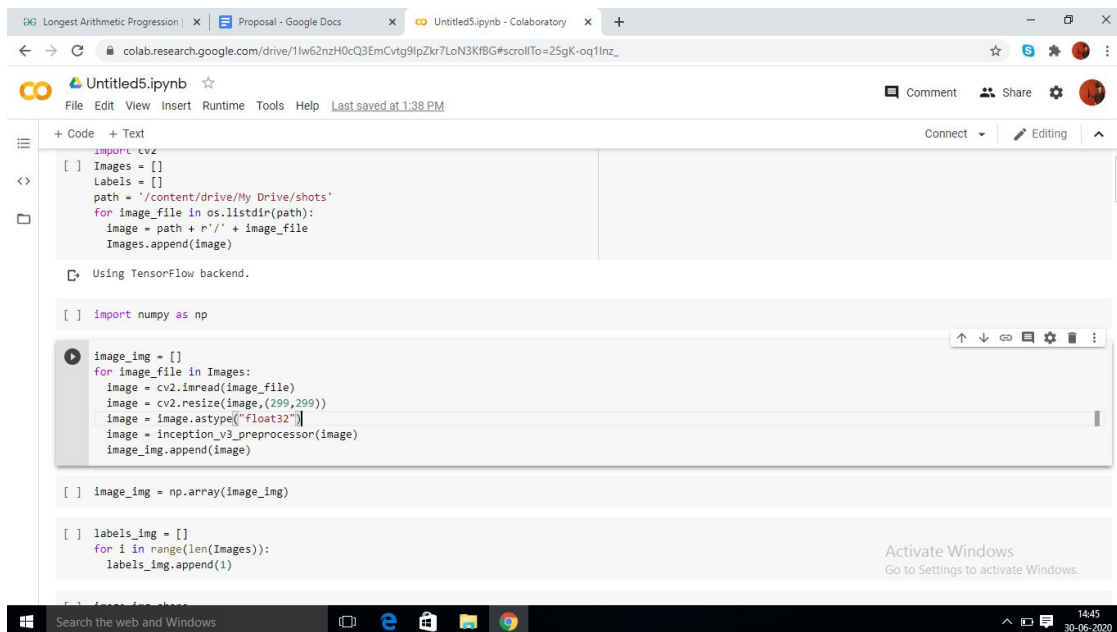
The screenshot shows a Jupyter Notebook titled 'Untitled4.ipynb' in a Colaboratory environment. The code is written in Python and uses OpenCV and NumPy for image processing. It includes comments explaining the logic, such as reading frames at intervals of 1 millisecond and breaking the loop when the 'q' key is pressed. The code also includes a warning to activate Windows.

```
if prev_label == 1 or curr_label == 1:
    #labels.append(1)
    clipname = "shots/" + video + str(loc) + clip_ext
    #dense_flow.savefig(clipname)
    cv2.imwrite(os.path.join(image_path, clipname), dense_flow)
    #cv2.imwrite(clipname, dense_flow)
    files.download(os.path.join(image_path, clipname))
else:
    #labels.append(0)
    clipname = "non_shots/" + video + str(loc) + clip_ext
    #dense_flow.savefig(clipname)
    cv2.imwrite(os.path.join(image_path, clipname), dense_flow)
    #cv2.imwrite(clipname, dense_flow)
    files.download(os.path.join(image_path, clipname))
    #cv2.imwrite(clipname, dense_flow)
    #cv2.imshow("Dense optical flow", dense_flow)
    #out.write(dense_flow)
    # Update previous frame
    prev_gray = gray

    # Frame are read by intervals of 1 millisecond. The programs breaks out of the while loop when the user presses the 'q' key
    # if cv2.waitKey(10) & 0xFF == ord('q'):
    #     break

    count+=1
# The following frees up resources and closes all windows
vc.release()
cv2.destroyAllWindows()
#len(images)
#len(labels)
```

2. Apply Image preprocessing techniques on captured Optical Flows.



The screenshot shows a Jupyter Notebook titled 'Untitled5.ipynb' in a Colaboratory environment. The code is written in Python and uses OpenCV and NumPy for image preprocessing. It includes comments explaining the logic, such as reading images from a directory and applying preprocessing techniques like resizing and normalization. The code also includes a warning to activate Windows.

```
import cv2
Images = []
Labels = []
path = '/content/drive/My Drive/shots'
for image_file in os.listdir(path):
    image = cv2.imread(os.path.join(path, image_file))
    Images.append(image)

Using TensorFlow backend.

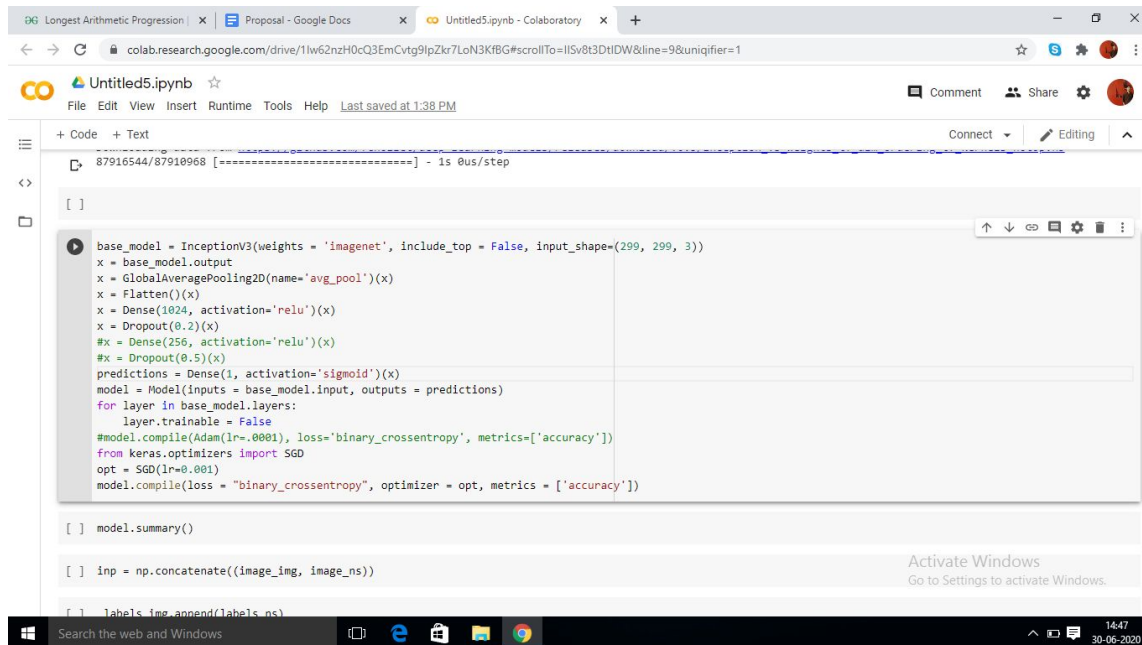
import numpy as np

image_img = []
for image_file in Images:
    image = cv2.imread(image_file)
    image = cv2.resize(image, (299, 299))
    image = image.astype("float32")
    image = inception_v3_preprocessor(image)
    image_img.append(image)

image_img = np.array(image_img)

labels_img = []
for i in range(len(Images)):
    labels_img.append(1)
```

3. Apply transfer-learning from pretrained models (VGG16, Inception-v3, etc) to extract features.



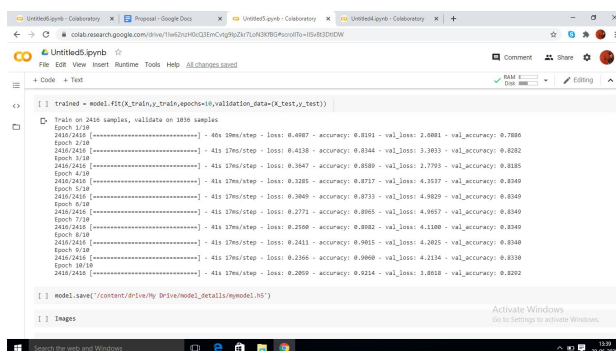
```
base_model = InceptionV3(weights = 'imagenet', include_top = False, input_shape=(299, 299, 3))
x = base_model.output
x = GlobalAveragePooling2D(name='avg_pool')(x)
x = Flatten()(x)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.2)(x)
#x = Dense(256, activation='relu')(x)
#x = Dropout(0.5)(x)
predictions = Dense(1, activation='sigmoid')(x)
model = Model(inputs = base_model.input, outputs = predictions)
for layer in base_model.layers:
    layer.trainable = False
#model.compile(Adam(lr=0.001), loss='binary_crossentropy', metrics=['accuracy'])
from keras.optimizers import SGD
opt = SGD(lr=0.001)
model.compile(loss = "binary_crossentropy", optimizer = opt, metrics = ['accuracy'])

[ ] model.summary()

[ ] inp = np.concatenate((image_img, image_ns))

[ ] labels_img.append(labels_ns)
```

4. Train a binary classifier DL model to predict inference from each Optical Flow Image - Whether the shot is being played or not (WIP Updates: classification 82% accuracy recorded).



```
trained = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test))

Train on 2416 samples, validate on 1816 samples
Epoch 1/10
2416/2416 [=====] - 40s 18ms/step - loss: 0.4987 - accuracy: 0.8191 - val_loss: 2.6861 - val_accuracy: 0.7886
Epoch 2/10
2416/2416 [=====] - 41s 17ms/step - loss: 0.4136 - accuracy: 0.8344 - val_loss: 3.3893 - val_accuracy: 0.8282
Epoch 3/10
2416/2416 [=====] - 41s 17ms/step - loss: 0.3647 - accuracy: 0.8589 - val_loss: 2.7793 - val_accuracy: 0.8185
Epoch 4/10
2416/2416 [=====] - 41s 17ms/step - loss: 0.3285 - accuracy: 0.8717 - val_loss: 4.3537 - val_accuracy: 0.8349
Epoch 5/10
2416/2416 [=====] - 41s 17ms/step - loss: 0.3649 - accuracy: 0.8733 - val_loss: 4.9829 - val_accuracy: 0.8349
Epoch 6/10
2416/2416 [=====] - 41s 17ms/step - loss: 0.2773 - accuracy: 0.8965 - val_loss: 4.9637 - val_accuracy: 0.8349
Epoch 7/10
2416/2416 [=====] - 41s 17ms/step - loss: 0.2588 - accuracy: 0.8982 - val_loss: 4.1188 - val_accuracy: 0.8349
Epoch 8/10
2416/2416 [=====] - 41s 17ms/step - loss: 0.2411 - accuracy: 0.9055 - val_loss: 4.1025 - val_accuracy: 0.8349
Epoch 9/10
2416/2416 [=====] - 41s 17ms/step - loss: 0.2368 - accuracy: 0.9068 - val_loss: 4.2124 - val_accuracy: 0.8338
Epoch 10/10
2416/2416 [=====] - 41s 17ms/step - loss: 0.2059 - accuracy: 0.9214 - val_loss: 3.8818 - val_accuracy: 0.8292

[ ] model.save('content/drive/My Drive/model_details/model.h5')

[ ] Images
```

Dataset(s)

Overview:

3-4 hours of videos during training sessions (equals approx. 640 cricket shots)

End to End Model

Input:

The video for which highlight is to be calculated.

Output:

The output of the model will be a list of labels(0- corresponding optical flow image is not a part of shot & 1- corresponding optical flow image is a part of shot). There will be a total of (video length(in seconds)*2) entries in the corresponding list of labels.