WEATHER CLASSIFICATION FROM VIDEOS USING CONVOLUTIONAL NEURAL NETWORKS

Bhuwan Sapkota and Samrid KC

APPLICATIONS/USES

- Detection and observation of weather conditions for outdoor video surveillance systems.
- Vehicle assistant driving systems can utilize the model to make better driving decisions based on weather conditions. Currently they use expensive sensors for this purpose, but this model can replace or work in conjunction with these sensors.
- It can be also used to classify videos/images based on weather conditions in databases and in different photo apps(e.g. Google Photos).

DATA SOURCE

- **Joint Attention for Autonomous Driving (JAAD) Dataset**

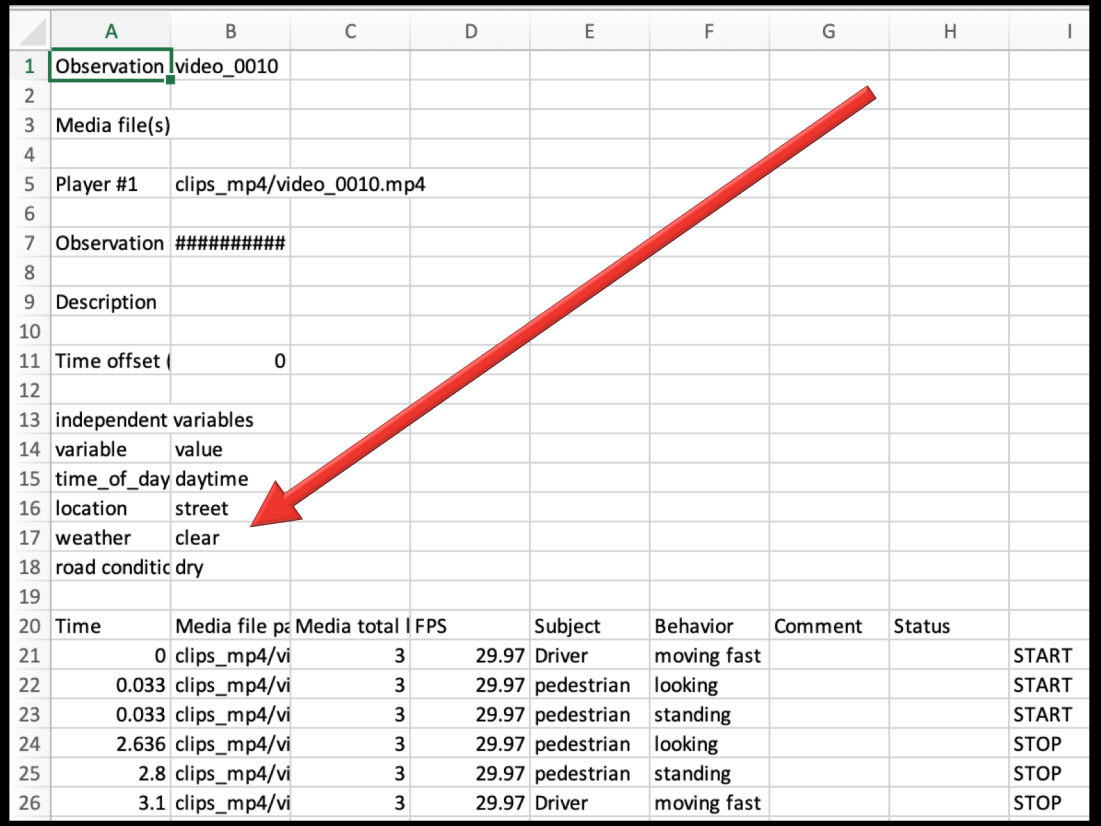
(http://data.nvision2.eecs.yorku.ca/JAAD_dataset/)

- The dataset contains 346 videos.
- The dataset contains HD dashcam videos, which are mostly 7-8 seconds.
- These videos are 30fps.
- The videos were classified according to the weather types(clear, cloudy, rain, snow) i.e. four different classes of weather.
- 166 clear, 143 cloudy, 28 rain, and only 6 snow.
- Example video

(http://jaad-explore.nvision2.eecs.yorku.ca/clips/video_0179.mp4)

DATA LABELING

- A Github repo containing behavioral data for each video in tsv files was also provided alongside the dataset.
- We wrote a Python script to extract this data, and label the videos and their frames accordingly.

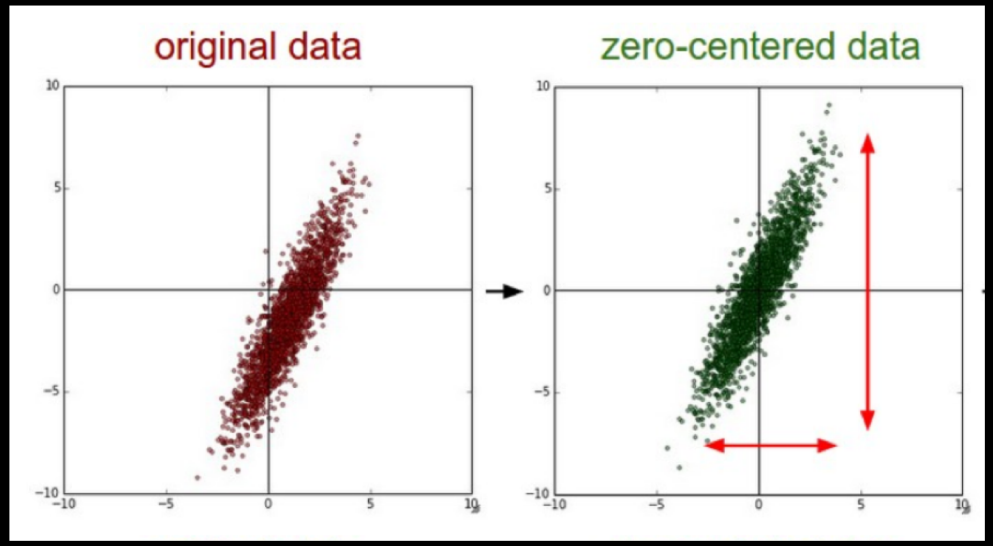


	A	B	C	D	E	F	G	H	I
1	Observation	video_0010							
2									
3	Media file(s)								
4									
5	Player #1	clips_mp4/video_0010.mp4							
6									
7	Observation	##########							
8									
9	Description								
10									
11	Time offset (	0							
12									
13	independent	variables							
14	variable	value							
15	time_of_day	daytime							
16	location	street							
17	weather	clear							
18	road conditic	dry							
19									
20	Time	Media file pa	Media total l	FPS	Subject	Behavior	Comment	Status	
21	0	clips_mp4/vi	3	29.97	Driver	moving fast			START
22	0.033	clips_mp4/vi	3	29.97	pedestrian	looking			START
23	0.033	clips_mp4/vi	3	29.97	pedestrian	standing			START
24	2.636	clips_mp4/vi	3	29.97	pedestrian	looking			STOP
25	2.8	clips_mp4/vi	3	29.97	pedestrian	standing			STOP
26	3.1	clips_mp4/vi	3	29.97	Driver	moving fast			STOP

DATA PREPROCESSING

- Almost all videos were 30 fps, some were 60 fps. But since there was very little difference between these frames, we decided to only extract one frame per second.
- We ended up with around 2705 frames out of the 346 videos.
- The images were all resized to 228 x 128 maintaining the original aspect ratio of 16:9. Videos were originally 1920 x 1080.
- Lastly the data was zero centered

  before being fed into the network.



METHODS EXPLORED

- We started by implementing a vanilla CNN using Keras. The results were not so great at first.
- We tried many different variations of convolution and pooling layers and the accuracy was always around 0.45.
- We even toyed with the idea of using a pre-trained model like VGG16.
- The images were not zero centered at first. After we zero centered the data, the vanilla CNN that we had implemented first produced better results with accuracy of around 0.90.
- We wanted to experiment using deeper CNNs and even more frames but we did not have the resources in our local machines to run such a large network.

METHOD EXPLORED

- Our model architecture

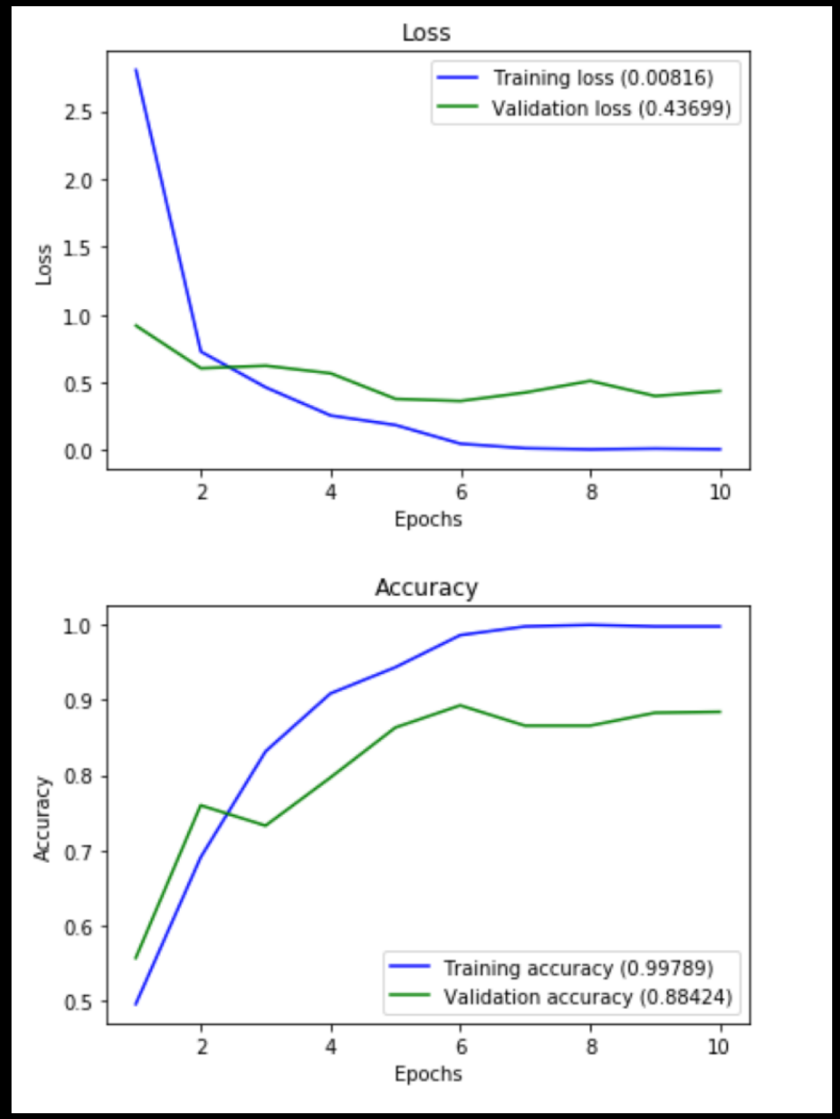
Layer (type)	Output Shape	Param #
ConvolutionLayer1 (Conv2D)	(None, 126, 226, 48)	1344
MaxPoolingLayer1 (MaxPooling	(None, 63, 113, 48)	0
ConvolutionLayer2 (Conv2D)	(None, 61, 111, 96)	41568
MaxPoolingLayer2 (MaxPooling	(None, 30, 55, 96)	0
FlatteningLayer (Flatten)	(None, 158400)	0
FullyConnectedLayer (Dense)	(None, 100)	15840100
OutputLayer (Dense)	(None, 4)	404

Total params: 15,883,416
Trainable params: 15,883,416
Non-trainable params: 0

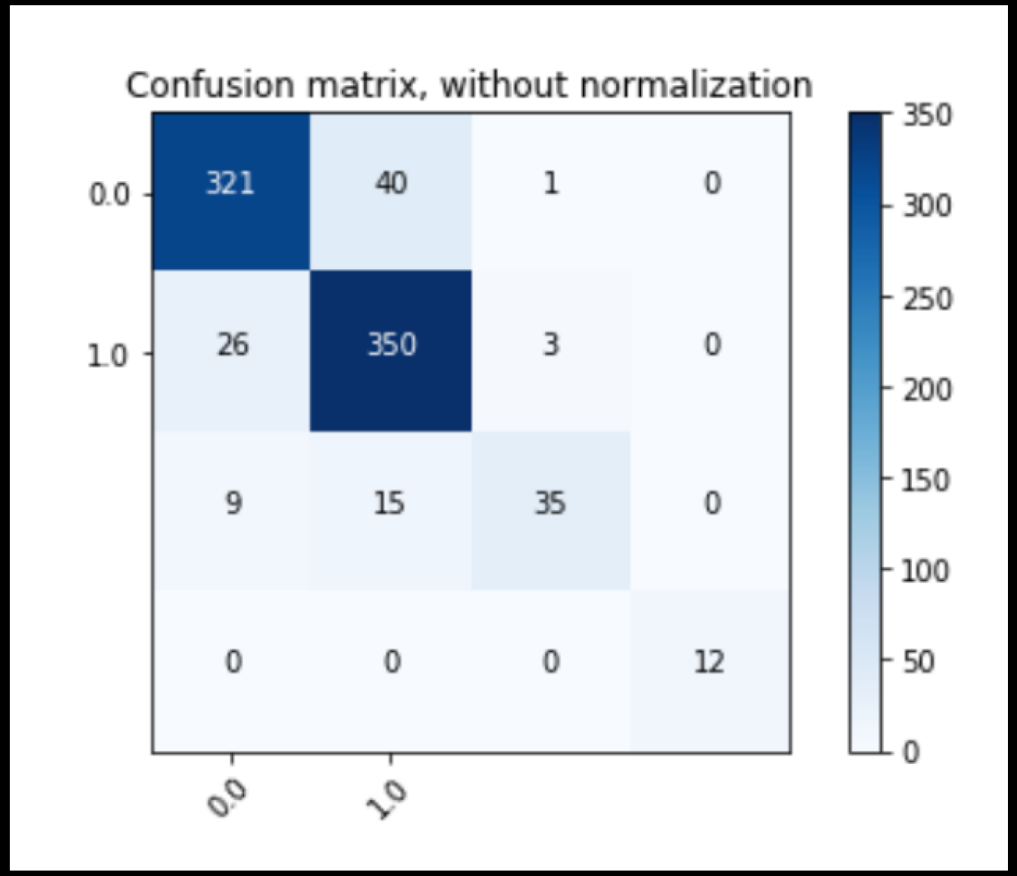
None

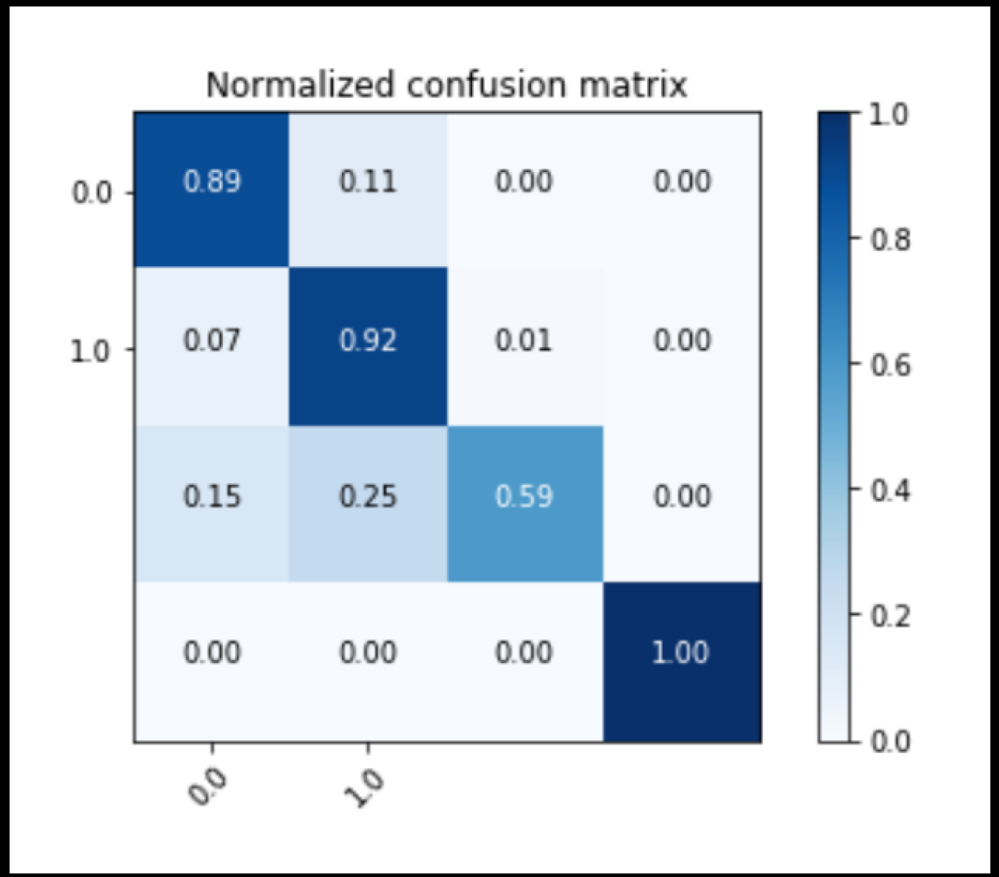
RESULTS AND EVALUATION

```
Train on 1894 samples, validate on 812 samples
Epoch 1/10
 - 107s - loss: 2.8014 - acc: 0.4952 - val_loss: 0.9191 - val_acc: 0.5567
Epoch 2/10
 - 107s - loss: 0.7276 - acc: 0.6906 - val_loss: 0.6049 - val_acc: 0.7599
Epoch 3/10
 - 93s - loss: 0.4642 - acc: 0.8316 - val_loss: 0.6237 - val_acc: 0.7328
Epoch 4/10
 - 98s - loss: 0.2569 - acc: 0.9087 - val_loss: 0.5675 - val_acc: 0.7968
Epoch 5/10
 - 96s - loss: 0.1864 - acc: 0.9435 - val_loss: 0.3794 - val_acc: 0.8633
Epoch 6/10
 - 101s - loss: 0.0492 - acc: 0.9863 - val_loss: 0.3638 - val_acc: 0.8929
Epoch 7/10
 - 100s - loss: 0.0157 - acc: 0.9979 - val_loss: 0.4261 - val_acc: 0.8658
Epoch 8/10
 - 95s - loss: 0.0072 - acc: 1.0000 - val_loss: 0.5117 - val_acc: 0.8658
Epoch 9/10
 - 97s - loss: 0.0131 - acc: 0.9979 - val_loss: 0.4006 - val_acc: 0.8830
Epoch 10/10
 - 89s - loss: 0.0082 - acc: 0.9979 - val_loss: 0.4370 - val_acc: 0.8842
```

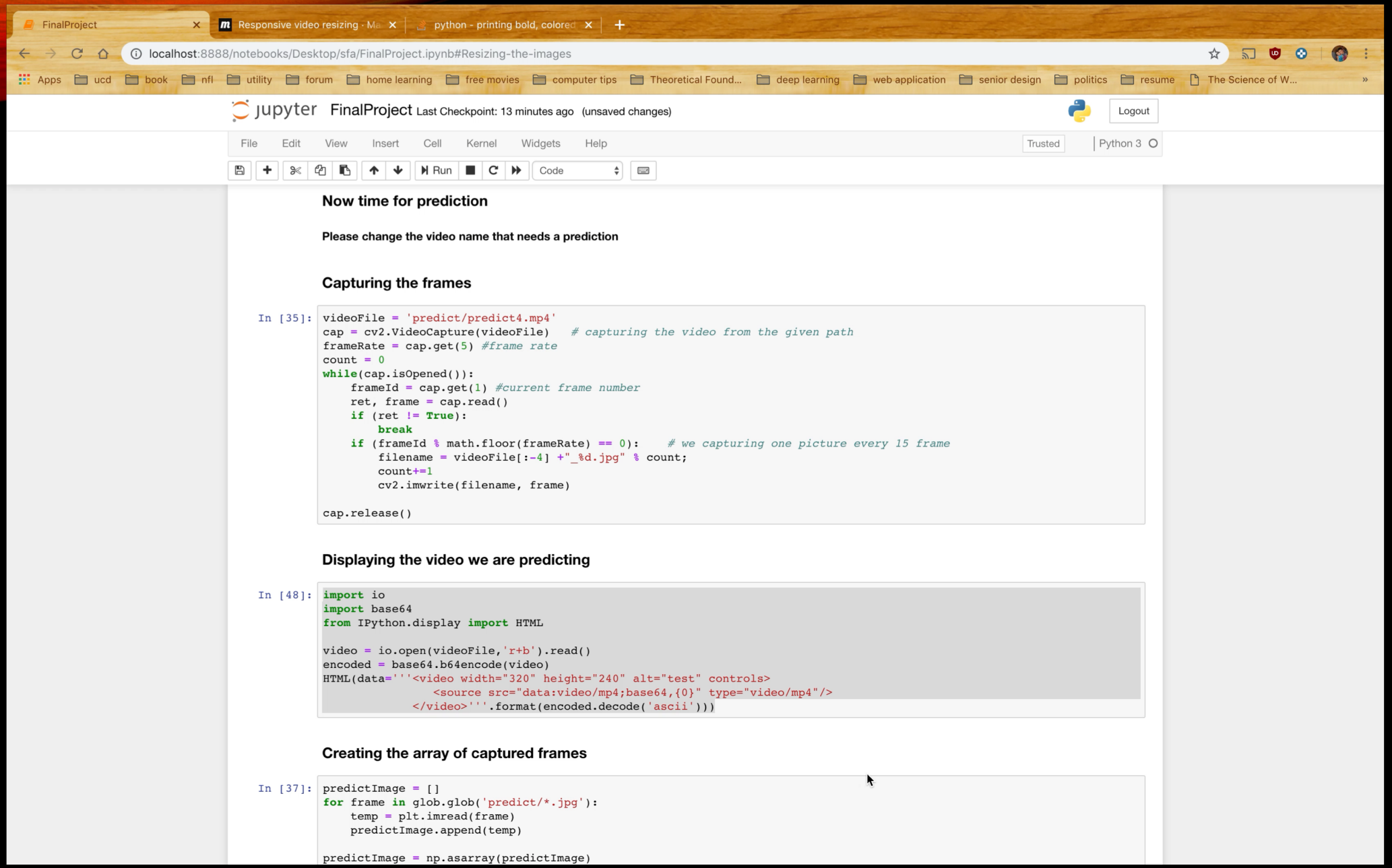


RESULTS AND EVALUATION





DEMO


FinalProject
Responsive video resizing - Ma
python - printing bold, colored
localhost:8888/notebooks/Desktop/sfa/FinalProject.ipynb#Resizing-the-images
Apps ucd book nfl utility forum home learning free movies computer tips Theoretical Found... deep learning web application senior design politics resume The Science of W...
jupyter FinalProject Last Checkpoint: 13 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Now time for prediction
Please change the video name that needs a prediction
Capturing the frames
In [35]:
videoFile = 'predict/predict4.mp4'
cap = cv2.VideoCapture(videoFile) # capturing the video from the given path
frameRate = cap.get(5) #frame rate
count = 0
while(cap.isOpened()):
frameId = cap.get(1) #current frame number
ret, frame = cap.read()
if (ret != True):
break
if (frameId % math.floor(frameRate) == 0): # we capturing one picture every 15 frame
filename = videoFile[:-4] +"_%d.jpg" % count;
count+=1
cv2.imwrite(filename, frame)
cap.release()
Displaying the video we are predicting
In [48]:
import io
import base64
from IPython.display import HTML
video = io.open(videoFile,'r+b').read()
encoded = base64.b64encode(video)
HTML(data='''<video width="320" height="240" alt="test" controls>
<source src="data:video/mp4;base64,{0}" type="video/mp4"/>
</video>'''.format(encoded.decode('ascii')))
Creating the array of captured frames
In [37]:
predictImage = []
for frame in glob.glob('predict/*.jpg'):
temp = plt.imread(frame)
predictImage.append(temp)
predictImage = np.asarray(predictImage)

EXTENDED IDEAS

- Use a larger dataset, especially with more videos of snowy and rainy weather.
- We can also develop this model to classify videos of more categories of weather such as different level of snow/rain/haze.
- If we had the opportunity to train our model on a machine with better computational power, we would have experimented with different and deeper CNNs to achieve even better results.

QUESTIONS?