**Advanced Augmentation- weather aspect**

“Image Augmentation” is a common technique for generating new images different than the original image in some aspect. This is done to generalize the learning mechanism and make it robust to sudden changes in the scenarios.

Just like the human brain which has driven a car on a sunny day finds it difficult initially to drive in rain, but slowly gets accustomed to it. An artificial neural network too finds it confusing to drive in a new environment unless it has seen it earlier. Their are various augmentation techniques like flipping, translating, adding noise or changing color channel. I tried to look at the weather part of it. I found it very easy after a while to introduce various weather scenarios in an image.

Lets’ have a look.

I’ll first show you an original test image and will then augment it.



**Sunny and Shady**

After adding random sunny and shady effect, image’s brightness changes. This is an easy and quick transformation to perform.

def add\_brightness(image):  
 image\_HLS = cv2.cvtColor(image,cv2.COLOR\_RGB2HLS) ## Conversion to HLS  
 image\_HLS = np.array(image\_HLS, dtype = np.float64)   
 random\_brightness\_coefficient = np.random.uniform()+0.5 ## generates value between 0.5 and 1.5  
 image\_HLS[:,:,1] = image\_HLS[:,:,1]\*random\_brightness\_coefficient ## scale pixel values up or down for channel 1(Lightness)  
 image\_HLS[:,:,1][image\_HLS[:,:,1]>255] = 255 ##Sets all values above 255 to 255  
 image\_HLS = np.array(image\_HLS, dtype = np.uint8)  
 image\_RGB = cv2.cvtColor(image\_HLS,cv2.COLOR\_HLS2RGB) ## Conversion to RGB  
 return image\_RGB

Brightness of an image can easily be changed by changing the pixel values of “Lightness”- channel 1 of image in HLS color space. Converting the image back to RGB gives the same image with enhanced or suppressed lighting.

**Sunny**



**Shady**



**Shadows**

To a car, a shadow is nothing but dark portions in the image, which can also be bright at times. So a self driven car should always learn to drive with or without shadows. Randomly changing brightness on the hills or in the woods often boggle a car’s perception if not trained properly. This is even more prevalent on sunny days and differently tall buildings in a city, allowing beams of light to peep through.

Brightness is good for perception but uneven, sudden or too much brightness create perception issues. Let’s generate some fake shadows.

def generate\_shadow\_coordinates(imshape, no\_of\_shadows=1):  
 vertices\_list=[]  
 for index in range(no\_of\_shadows):  
 vertex=[]  
 for dimensions in range(np.random.randint(3,15)): ## Dimensionality of the shadow polygon  
 vertex.append(( imshape[1]\*np.random.uniform(),imshape[0]//3+imshape[0]\*np.random.uniform()))  
 vertices = np.array([vertex], dtype=np.int32) ## single shadow vertices   
 vertices\_list.append(vertices)  
 return vertices\_list ## List of shadow vertices

def add\_shadow(image,no\_of\_shadows=1):  
 image\_HLS = cv2.cvtColor(image,cv2.COLOR\_RGB2HLS) ## Conversion to HLS  
 mask = np.zeros\_like(image)   
 imshape = image.shape  
 vertices\_list= generate\_shadow\_coordinates(imshape, no\_of\_shadows) #3 getting list of shadow vertices  
 for vertices in vertices\_list:   
 cv2.fillPoly(mask, vertices, 255) ## adding all shadow polygons on empty mask, single 255 denotes only red channel  
   
 image\_HLS[:,:,1][mask[:,:,0]==255] = image\_HLS[:,:,1][mask[:,:,0]==255]\*0.5 ## if red channel is hot, image's "Lightness" channel's brightness is lowered   
 image\_RGB = cv2.cvtColor(image\_HLS,cv2.COLOR\_HLS2RGB) ## Conversion to RGB  
 return image\_RGB

OpenCV’s fillPoly() function is really handy in this case. Create some random vertices and impose the polygon on an empty mask using fillPoly(). This being done, the only thing left to do is to check the mask for hot pixels and reduce the “Lightness” in the HLS image wherever these hot pixels are found.

**Random shadow polygon on the road**



**Snow**

Well this is something new. We often wonder how would our vehicle behave on snowy roads. One way to test that is to get pics of snow clad roads or do something on the images to get a similar effect. This effect is not a complete alternative to snowy roads but is a good try and test approach.

def add\_snow(image):  
 image\_HLS = cv2.cvtColor(image,cv2.COLOR\_RGB2HLS) ## Conversion to HLS  
 image\_HLS = np.array(image\_HLS, dtype = np.float64)   
 brightness\_coefficient = 2.5   
 snow\_point=140 ## increase this for more snow  
 image\_HLS[:,:,1][image\_HLS[:,:,1]<snow\_point] = image\_HLS[:,:,1][image\_HLS[:,:,1]<snow\_point]\*brightness\_coefficient ## scale pixel values up for channel 1(Lightness)  
 image\_HLS[:,:,1][image\_HLS[:,:,1]>255] = 255 ##Sets all values above 255 to 255  
 image\_HLS = np.array(image\_HLS, dtype = np.uint8)  
 image\_RGB = cv2.cvtColor(image\_HLS,cv2.COLOR\_HLS2RGB) ## Conversion to RGB  
 return image\_RGB

Yup!! that’s it. This code generally whitens the darkest parts of the image, which are generally roads, trees, mountains, etc. using the same HLS “Lightness” increase method used in other approaches above. This technique doesn’t works good for dark images but can be modified to do so. Here’s what you get.

**winter is here**



You can tweak some parameters in the code for more or lesser snow than this. I have tested this on other images too and this technique gives me chills.

**Rain**

Yes, you heard that right. Why not rain ? When humans experience difficulty driving in rain why should vehicles be spared from that. In fact this is one of the situation in which I want my self driving car to be trained the most. Slippery roads and blurred visions are risky and cars should know how to tackle that.

def generate\_random\_lines(imshape,slant,drop\_length):  
 drops=[]  
 for i in range(1500): ## If You want heavy rain, try increasing this  
 if slant<0:  
 x= np.random.randint(slant,imshape[1])  
 else:  
 x= np.random.randint(0,imshape[1]-slant)  
 y= np.random.randint(0,imshape[0]-drop\_length)  
 drops.append((x,y))  
 return drops  
   
   
def add\_rain(image):  
   
 imshape = image.shape  
 slant\_extreme=10  
 slant= np.random.randint(-slant\_extreme,slant\_extreme)   
 drop\_length=20  
 drop\_width=2  
 drop\_color=(200,200,200) ## a shade of gray  
 rain\_drops= generate\_random\_lines(imshape,slant,drop\_length)  
   
 for rain\_drop in rain\_drops:  
 cv2.line(image,(rain\_drop[0],rain\_drop[1]),(rain\_drop[0]+slant,rain\_drop[1]+drop\_length),drop\_color,drop\_width)  
 image= cv2.blur(image,(7,7)) ## rainy view are blurry  
   
 brightness\_coefficient = 0.7 ## rainy days are usually shady   
 image\_HLS = cv2.cvtColor(image,cv2.COLOR\_RGB2HLS) ## Conversion to HLS  
 image\_HLS[:,:,1] = image\_HLS[:,:,1]\*brightness\_coefficient ## scale pixel values down for channel 1(Lightness)  
 image\_RGB = cv2.cvtColor(image\_HLS,cv2.COLOR\_HLS2RGB) ## Conversion to RGB  
 return image\_RGB

What I did here is that again I generated random points all over the image and then used the OpenCV’s line() function to generate small lines all over the image. I have also used a random slant in the rain drops to have a feel of actual rain. I have also reduced image’s brightness because rainy days are usually shady and also blurry because of the rain. Blur filter dimension and no. of rain drops can be changed for desired effect.

Here is the result:

**Fake rain but not much blur**



**Fog**

This is yet another scenario that hampers the vision of a self driving car a lot. Blurry white fluff in the image makes it very difficult to see beyond a certain stretch and reduces the sharpness in the image. Fog intensity is an important parameter to train a car for how much throttle has to be given. For coding such a function random patches can be taken all over the image and image’s lightness can be increased inside those patches. With a simple blur, this gives a nice hazy effect.

def add\_blur(image, x,y,hw):  
 image[y:y+hw, x:x+hw,1] = image[y:y+hw, x:x+hw,1]+1  
 image[:,:,1][image[:,:,1]>255] = 255 ##Sets all values above 255 to 255  
 image[y:y+hw, x:x+hw,1] = cv2.blur(image[y:y+hw, x:x+hw,1] ,(10,10))  
 return image

def generate\_random\_blur\_coordinates(imshape,hw):  
 blur\_points=[]  
 midx= imshape[1]//2-hw-100  
 midy= imshape[0]//2-hw-100  
 index=1  
 while(midx>-100 or midy>-100): ## radially generating coordinates  
 for i in range(250\*index):  
 x= np.random.randint(midx,imshape[1]-midx-hw)  
 y= np.random.randint(midy,imshape[0]-midy-hw)  
 blur\_points.append((x,y))  
 midx-=250\*imshape[1]//sum(imshape)  
 midy-=250\*imshape[0]//sum(imshape)  
 index+=1  
 return blur\_points  
   
def add\_fog(image):  
 image\_HLS = cv2.cvtColor(image,cv2.COLOR\_RGB2HLS) ## Conversion to HLS  
 mask = np.zeros\_like(image)   
 imshape = image.shape  
 hw=100  
 image\_HLS[:,:,1]=image\_HLS[:,:,1]\*0.8  
 haze\_list= generate\_random\_blur\_coordinates(imshape,hw)  
 for haze\_points in haze\_list:   
 image\_HLS[:,:,1][image\_HLS[:,:,1]>255] = 255 ##Sets all values above 255 to 255  
 image\_HLS= add\_blur(image\_HLS, haze\_points[0],haze\_points[1], hw) ## adding all shadow polygons on empty mask, single 255 denotes only red channel  
 image\_RGB = cv2.cvtColor(image\_HLS,cv2.COLOR\_HLS2RGB) ## Conversion to RGB  
 return image\_RGB

Coding this was the hardest of all the functions above. I have tried a radial approach to generate patches here. Since on a foggy day usually most of the fog is at the far end of the road and as we approach near, vision keeps clearing itself.

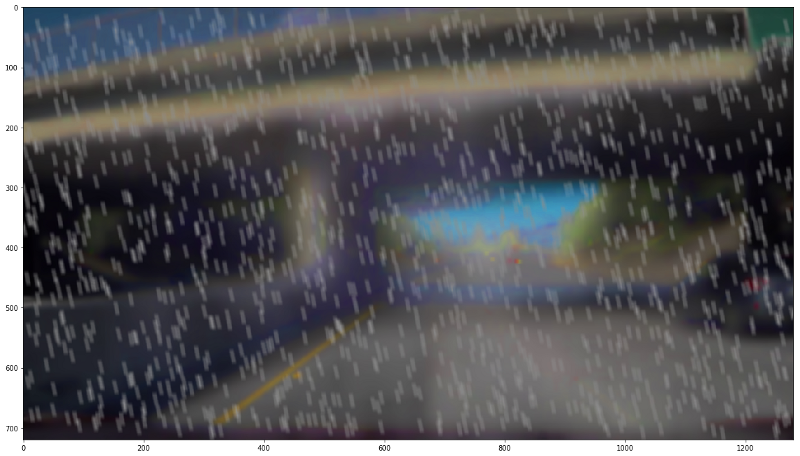
**Foggy Highway**



It’s a real difficult task for a machine to detect nearby cars and lanes in such a foggy condition, and is a good way to train and test the robustness of the driving model.

**Torrential rain**

I thought of making the rain part a little better by combining fog and rain. As there is always some haze during rains and it’s good to train the car for that also. No new function is required for this. The effect can be achieved by sequentially calling both.



The car on the right is barely visible in this image and this is a real world scenario. We can hardly make out anything on the road in heavy rain.

I hope this article will help you train the model in various weather conditions. For all other articles go to [medium](https://medium.com/@er.ujjwalsaxena) or [wordpress](https://erujjwalsaxena.wordpress.com/)

Enjoy !!