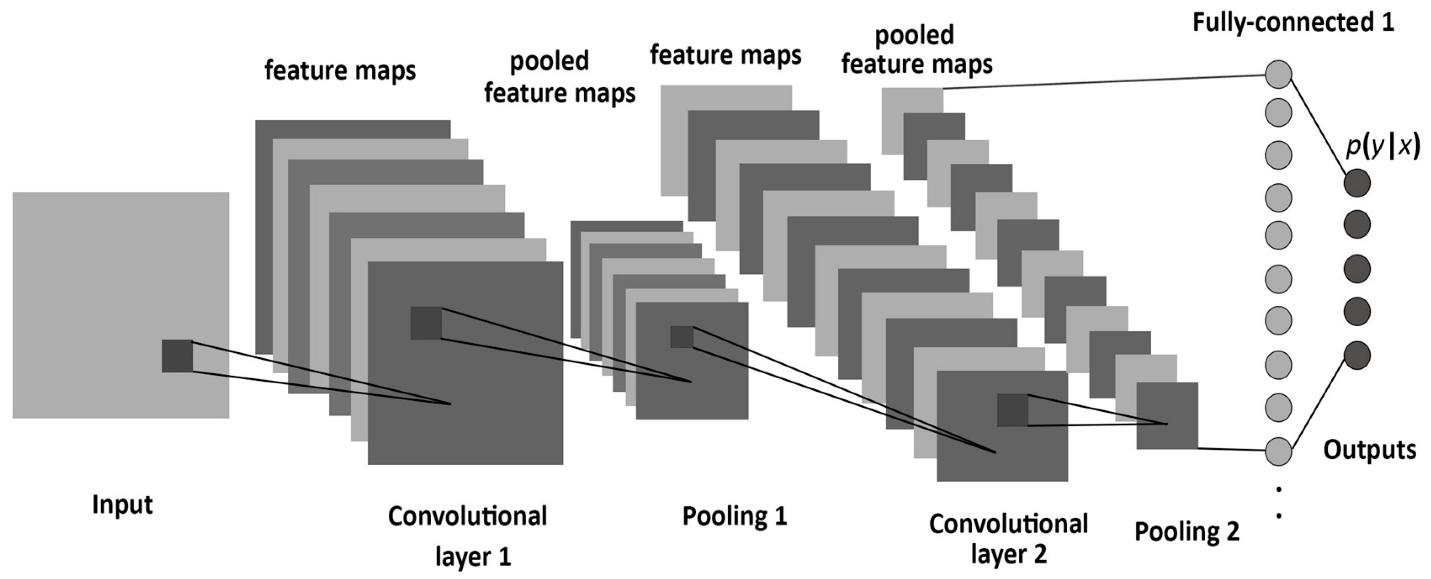
**https://medium.com/@ipaar3/how-i-built-a-convolutional-image-classifier-using-tensorflow-from-scratch-f852c34e1c95**

**Top highlight**

[](https://medium.com/@ipaar3?source=post_page-----f852c34e1c95----------------------)

[Paarvendhan](https://medium.com/@ipaar3?source=post_page-----f852c34e1c95----------------------)

[Jun 1, 2018](https://medium.com/@ipaar3/how-i-built-a-convolutional-image-classifier-using-tensorflow-from-scratch-f852c34e1c95?source=post_page-----f852c34e1c95----------------------) · 13 min read



the\_attractive\_reader\_magnet\_image.jpg

How I built a Convolutional Image classifier using Tensorflow from Scratch. **(Without using Dogs Vs Cats, From getting images from google to saving our trained model for reuse.)**

I’m just very tired of the same implementation everywhere in the internet.

Though it is from scratch, here I don’t explain the theory because you can get many better explanations online with visualizations too. However, Execution and CNNs are briefly explained.

***What are we gonna do:***

* We will build a ***3 layered community standard CNN Image classifier*** to classify whether the given image is a image of Batman or Superman.
* Learn how to build a model from scratch in Tensoflow.
* How to train and test it.
* How to save and use it further.

**Tl;Dr :** Procedure and code in[***Github.***](https://github.com/perseus784/BvS)

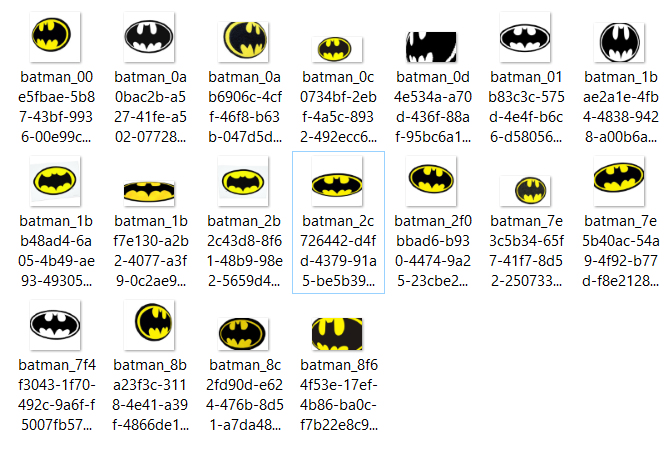
**Collecting Data:**

[***Google Images Downloader***](https://github.com/hardikvasa/google-images-download). This is what I’ve used and it’s fast, easy, simple and efficient.



I’ve collected 300 images each for Supes and Batsy respectively, But more data is highly preferable. Try to collect as much clean data as possible.

**Augmentation:**

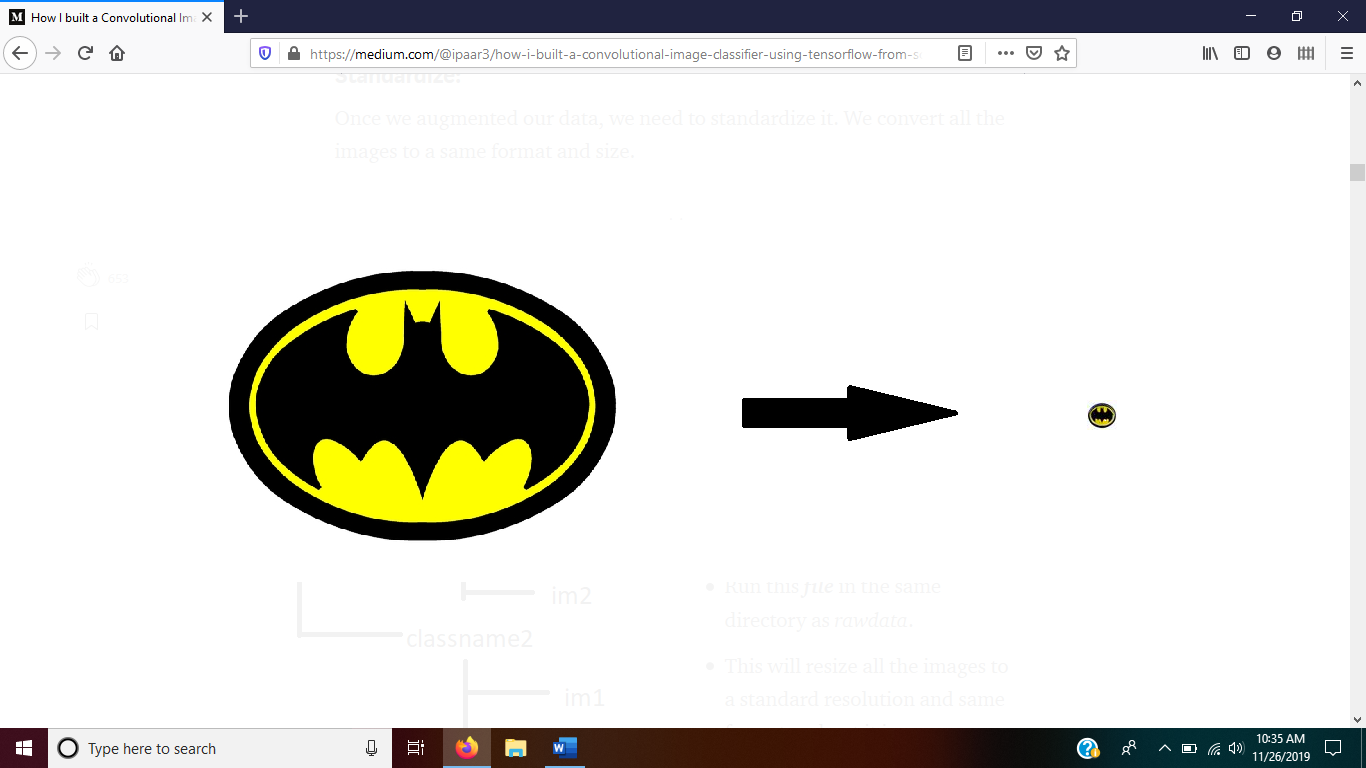
Like I said, 300 is not a number at all in Deep learning. So, we must Augment the images to get more images from whatever we collected. You can use the following to do it easily, [***Augmentor***](https://github.com/mdbloice/Augmentor)*. T*he code that I’ve used is in *Github* that is mentioned at the end.

Same Image, Augmented using various transformations. I have had 3500 images each after augmentation.

***Careful:*** *While Augmenting, be careful about what kind of transformation you use. You can mirror flip a Bat Logo* ***but cannot make it upside down.***

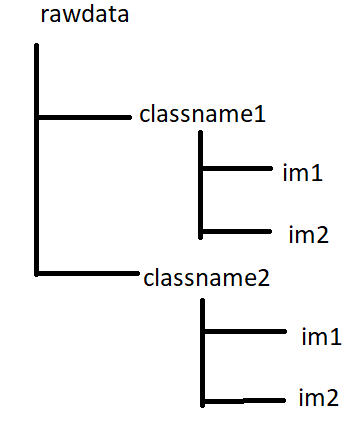
**Standardize:**

Once we augmented our data, we need to standardize it. We convert all the images to a same format and size.



resizing

***Procedure:***



* Make a folder named rawdata in the current working directory.
* Create folders with their respective class names and put all the images in their respective folders.
* Run this ***file*** in the same directory as *rawdata*.
* This will resize all the images to a standard resolution and same format and put it in a new folder named *data*

**Note:** As I embedded it in *trainer.py*, it is unnecessary to run it explicitly.

**Update:** I’ve added the *data* folder itself online found [here](https://drive.google.com/open?id=1GUPBBdLlqStnxjhISkxT1qOf1XPnmRcF). Just download and extract in the same folder as the project.

**Getting Serious:**

Okay, till now it’s just scripting work. I haven’t gone into details since the steps are rudimentary. From now on I will go step by step with explanation of what I’m doing in the code.

**Building a model:**

Don’t let it fool you with it’s complex behavior, we have at least a **billion times** complicated thing sitting on top of our head.

From Scratch:

* I just created a file named model.py
* Create a class name model\_tools with following functions:

class model\_tools:  
 *# Defined functions for all the basic tensorflow components that we needed for building a model.* def add\_weights(self):return 0  
 def add\_biases(self):return 0  
 def conv\_layer(self):  
 return 0  
 def pooling\_layer(self):  
 return 0  
 def flattening\_layer(self):  
 return 0  
 def fully\_connected\_layer(self):return 0  
 def activation\_layer(self):return 0

Now we are gonna define every function with it’s parameters,

***add\_weights:***

Parameters: *shape.*

It will return a connection of the given shape with some random initialised values whenever it is called. *tf.truncated\_normal* is used to generate more randomized initial values for that shape.

def add\_weights(self,shape): # a common method to create all sorts of weight connections  
 # takes in shapes of previous and new layer as a list e.g.   
 [2,10]  
 # starts with random values of that shape.  
 return tf.Variable(tf.truncated\_normal(shape=shape,stddev=0.05))

**add\_biases:**

Parameters: *shape*

Biases are initialised with some constant for that shape. Returns bias variable.

def add\_biases(self,shape):  
 # a common method to add create biases with default=0.05  
 # takes in shape of the current layer e.g. x=10  
 return tf.Variable(tf.constant(0.05, shape=shape))

**conv\_layer:**

Parameters: *layer, kernel, input\_shape, output\_shape, stride\_size.*

*Strides:* Think of these as jump values for the sliding window in the convolutional map.

Convolution occurs here.

layer - takes in last layer.  
kernel - kernel size for convoluting on the image.  
input\_shape - size of the input image.  
ouput\_shape - size of the convoluted image.  
stride\_size - determines kernel jump sizedef conv\_layer(self,layer, kernel, input\_shape, output\_shape, stride\_size):  
 weights = self.add\_weights([kernel, kernel, input\_shape,  
 output\_shape])  
 biases = self.add\_biases([output\_shape]) #stride=[image\_jump,row\_jump,column\_jump,color\_jump]=[1,1,1,1]  
 stride = [1, stride\_size, stride\_size, 1] #does a convolution scan on the given image  
 layer = tf.nn.conv2d(layer, weights, strides=stride,   
 padding='SAME') + biases   
 return layer

*More explanation is given in Architecture section.*

**pooling\_layer:**

Parameters: previous\_layer, kernel, stride

def pooling\_layer(self,layer, kernel\_size, stride\_size):  
 # basically it reduces the complexity involved by only taking the important features alone  
 # many types of pooling is there.. average pooling, max pooling..  
 # max pooling takes the maximum of the given kernel  
 #kernel=[image\_jump,rows,columns,depth]  
 kernel = [1, kernel\_size, kernel\_size, 1]  
 #stride=[image\_jump,row\_jump,column\_jump,color\_jump]=[1,2,2,1] mostly  
 stride = [1, stride\_size, stride\_size, 1]  
 return tf.nn.max\_pool(layer, ksize=kernel, strides=stride, padding='SAME')

**flattening\_layer:**

Parameters: layer

As the name says, it converts all multidimensional matrices into a single dimension.

def flattening\_layer(self,layer):  
 #make it single dimensional  
 input\_size = layer.get\_shape().as\_list()  
 new\_size = input\_size[-1] \* input\_size[-2] \* input\_size[-3]  
 return tf.reshape(layer, [-1, new\_size]),new\_size

**fully\_connected\_layer:**

Parameters: *previous layer, shape of previous layer, shape of output layer.*

This is a vanilla layer. It connects the previous layer with the output layer. Here is where the **mx+b** operation occurs.

def fully\_connected\_layer(self,layer, input\_shape, output\_shape):  
 #create weights and biases for the given layer shape  
 weights = self.add\_weights([input\_shape, output\_shape])  
 biases = self.add\_biases([output\_shape])  
 #most important operation  
 layer = tf.matmul(layer,weights) + biases # mX+b  
 return layer

**activation\_layer:**

Parameters: *layer*

we use Rectified linear unit Relu. it's the standard activation layer used.  
There are also other layer like sigmoid,tanh..etc. but relu is more efficent.  
**function**: 0 if x<0 else x.

def activation\_layer(self,layer):  
 return tf.nn.relu(layer)

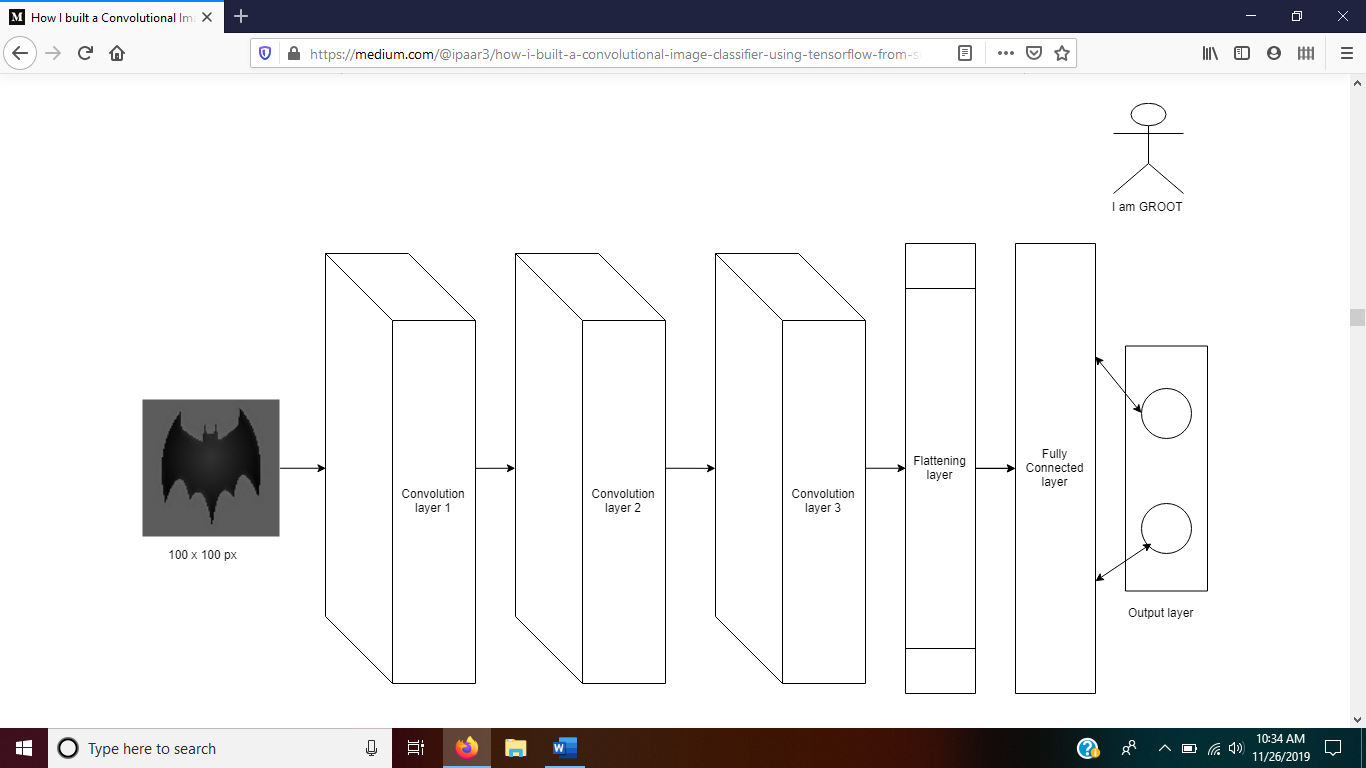
**Architecture:**

Now we have to put all the elements that we have seen above in a way to make it work for us.

A neural network is a black box, we won’t have any control over what happens inside those connections. but at each layer, we can get insights through which it is possible for us to calculate what combination of sequence of these functions will give us good results.

As I said, we are going to build a really standard system. So, we can use a standard architecture which is found in most successful models.

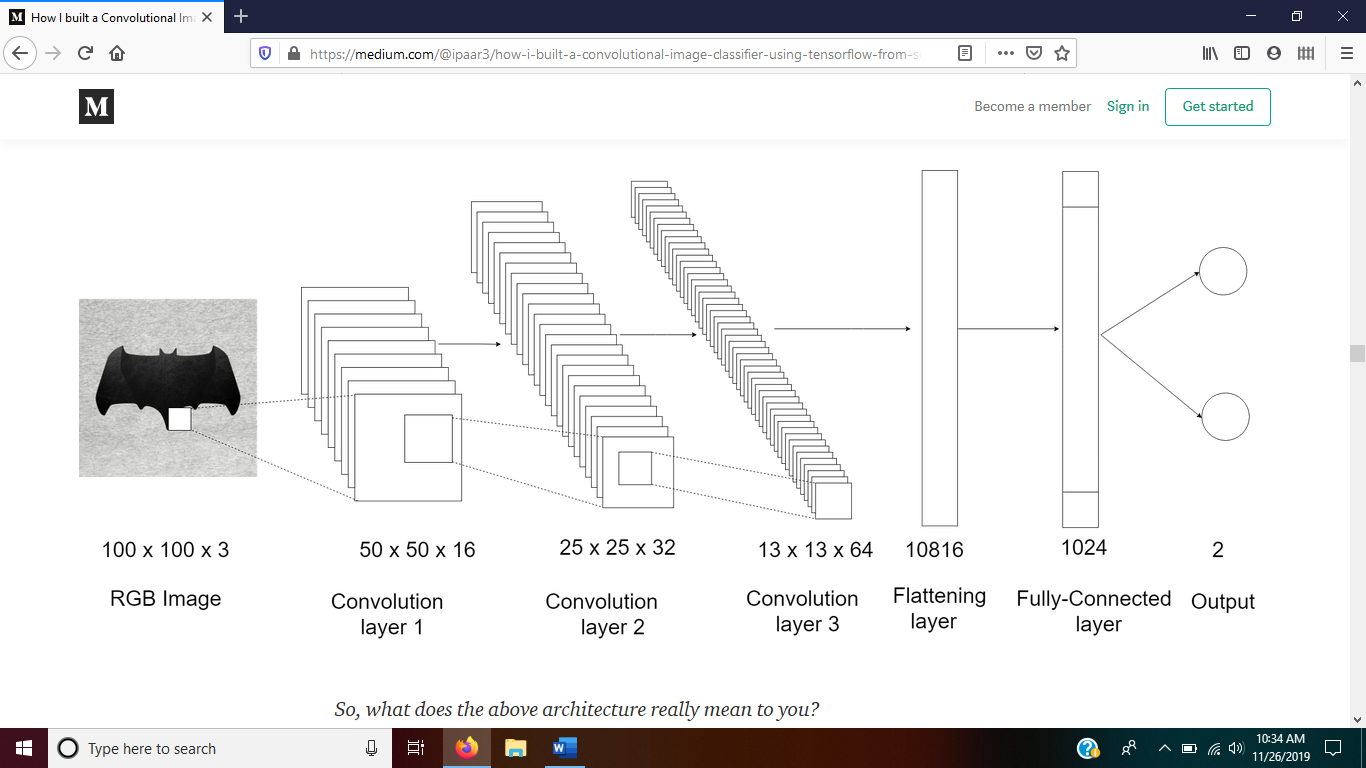
**A Simple Architecture:**



*#MODEL ARCHITECTURE:  
 #level 1 convolution* network=model.conv\_layer(images\_ph,5,3,16,1)  
 network=model.pooling\_layer(network,5,2)  
 network=model.activation\_layer(network)  
 print(network)  
  
 *#level 2 convolution* network=model.conv\_layer(network,4,16,32,1)  
 network=model.pooling\_layer(network,4,2)  
 network=model.activation\_layer(network)  
 print(network)  
  
 *#level 3 convolution* network=model.conv\_layer(network,3,32,64,1)  
 network=model.pooling\_layer(network,3,2)  
 network=model.activation\_layer(network)  
 print(network)  
  
 *#flattening layer* network,features=model.flattening\_layer(network)  
 print(network)  
  
 *#fully connected layer* network=model.fully\_connected\_layer(network,features,1024)  
 network=model.activation\_layer(network)  
 print(network) *#output layer* network=model.fully\_connected\_layer(network,1024,no\_of\_classes)  
 print(network)

**A Brief Architecture:**

In our architecture, we have 3 convolutional layers. I chose 3 because it seemed like a optimum choice for a small classifier. There are no rules for the size or dimensions of each convolutional layers.



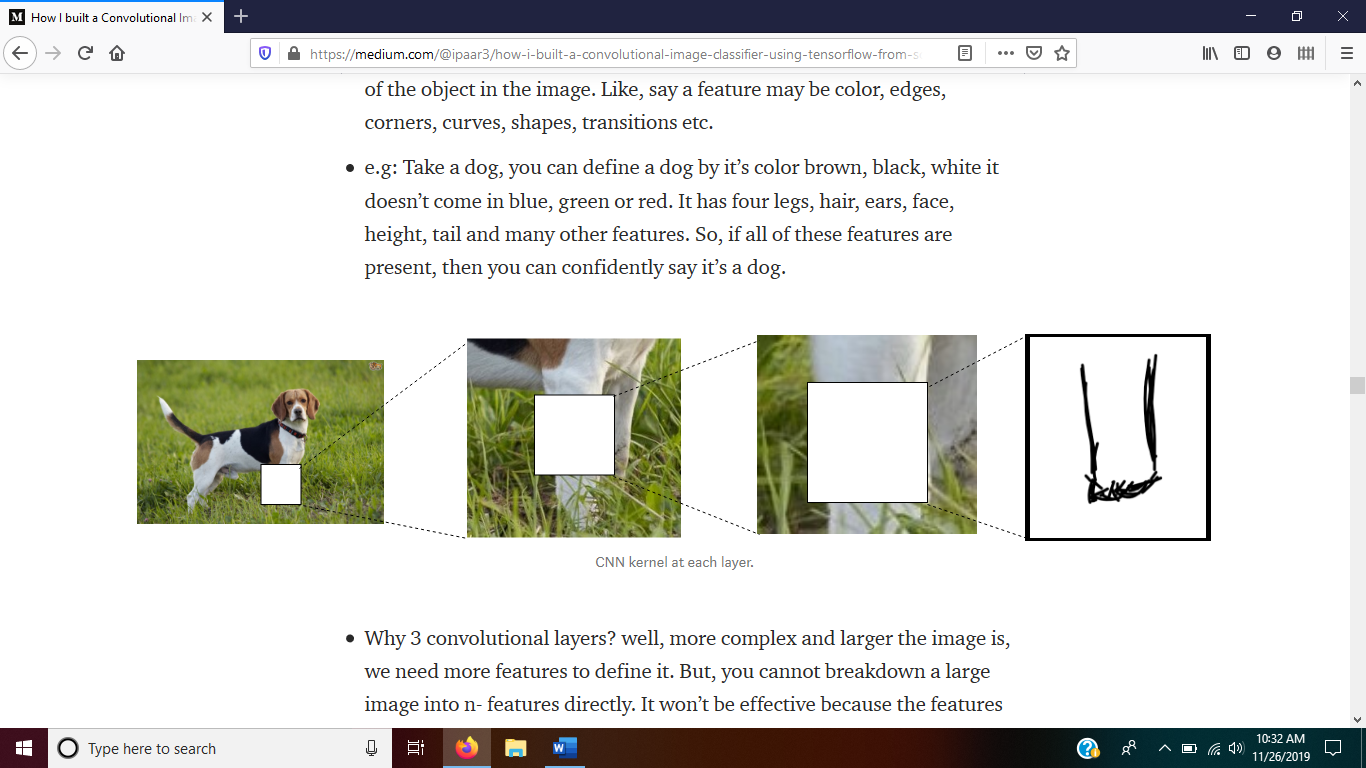
*So, what does the above architecture really mean to you?*

**Explaining Convolutional Layers:**

The last three layers are no rocket science, it is self- explanatory. So, lets talk about those convolutional layers.You can see the dimensional change in each convolutional layer.

Take an image. Now we are going to define this single image as 16 features for the first convoltion of 50 x 50 height and width .

* Okay, why 16? well, it doesn’t have any particular reason. It just works well like in most architectures. So, what this intuitively means is when you put back all the 16 features, you’ll get your image back.
* Okay, what are those 16 features and how to select them? hmm, remember people say Neural networks are black boxes?, we are gonna see it now. Those 16 features are not defined by us and we don’t select any particular feature. It is inside the black box and we don’t have control over it.
* Okay, inferences at least? yeah, we can have inferences but it’s just not humanly readable. It just learns whatever it sees through those pictures and we can’t reason with it. But to explain it, say feature define a feature of the object in the image. Like, say a feature may be color, edges, corners, curves, shapes, transitions etc.
* e.g: Take a dog, you can define a dog by it’s color brown, black, white it doesn’t come in blue, green or red. It has four legs, hair, ears, face, height, tail and many other features. So, if all of these features are present, then you can confidently say it’s a dog.



CNN kernel at each layer.

* Why 3 convolutional layers? well, more complex and larger the image is, we need more features to define it. But, you cannot breakdown a large image into n- features directly. It won’t be effective because the features won’t connect with each other due to the vastness of the image. So, it is good to level down and get feature maps as we go.
* So, what is it really learning?.. It is learning which set of features define an object. In layer 2, which set of features defines these features in layer 1. Same goes for all the layers in the network.
* As we go deeper, we reduce the size of the feature map and increase the number of features. So when you think of it, a group of points, edges, corner features forms a particular shape. A group of shapes, transitions, colors, pattern features forms a leg. A Group of leg features in that image along with head, body, color, tail features form a dog. So, remember a dog is convoluted into points and edges.

**Done with it:**

Okay, I’ve run out of patience. I’m sure you have too. So, lets jump straight without so much explanation.

**Training:**

* We have built our network. Now it is time to pass in some data and get those neurons fired.
* We have 1000s of images. Even though they are small in size, it is complex enough as it goes deep. So, we divide our images into small batches and send it to network.
* One complete cycle of all the images passed through the network remarks an epoch. Our network cannot learn all the features of an image at once, it needs to see it multiple times and also compare it all the other images that it have seen and decide which set of features of the image made it as a class A image or a class B image.
* ***“Show and Teach”*** We show the network this is an image of a dog and ask it to learn features of these over iterations and comparing with the original image. Comparison is nothing but how different the predicted value is to the expected output. we calculate that using Squared Error.
* We are going to use *softmax cross entropy* which is basically like finding squared error but this is more efficient and better.
* We found errors for individual images now we can average it and get the total error rate. It is also known as ***cost.***
* Now, we need to reduce this cost using some learning technique. Reducing the cost means what particular set of neurons should be fired in order that error is minimum. So, we have many variables(neurons) which should be optimized. There are many optimizers but it all began with the virtous ***Gradient Descent.***
* We are going to use an advanced technique as Gradient descent is old and slower. *Adam Optimizer,* It is almost the best choice in all kinds of networks and optimization.
* So, the image placeholder will have the images for that batch size and we are going to run our network using the Adam Optimizer with our image data. The code is given below with explanation of comments:

def trainer(network,number\_of\_images):  
 *#find error like squared error but better* cross\_entropy=tf.nn.softmax\_cross\_entropy\_with\_logits\_v2(logits=  
 network,labels=labels\_ph)  
  
 *#now minize the above error  
 #calculate the total mean of all the errors from all the nodes* cost=tf.reduce\_mean(cross\_entropy)  
 *#Now backpropagate to minimise the cost in the network.* optimizer=tf.train.AdamOptimizer().minimize(cost)session.run(tf.global\_variables\_initializer())  
 counter=0 for epoch in range(epochs):  
 tools = processing\_tools() for batch in range(int(number\_of\_images / batch\_size)):  
 counter+=1 images, labels = tools.batch\_dispatch() if images == None:break loss = session.run([cost], feed\_dict=   
 {images\_ph: images, labels\_ph: labels}) print('loss', loss) session.run(optimizer, feed\_dict={images\_ph: images,   
 labels\_ph: labels})  
 print('Epoch number ', epoch, 'batch', batch,   
 'complete')

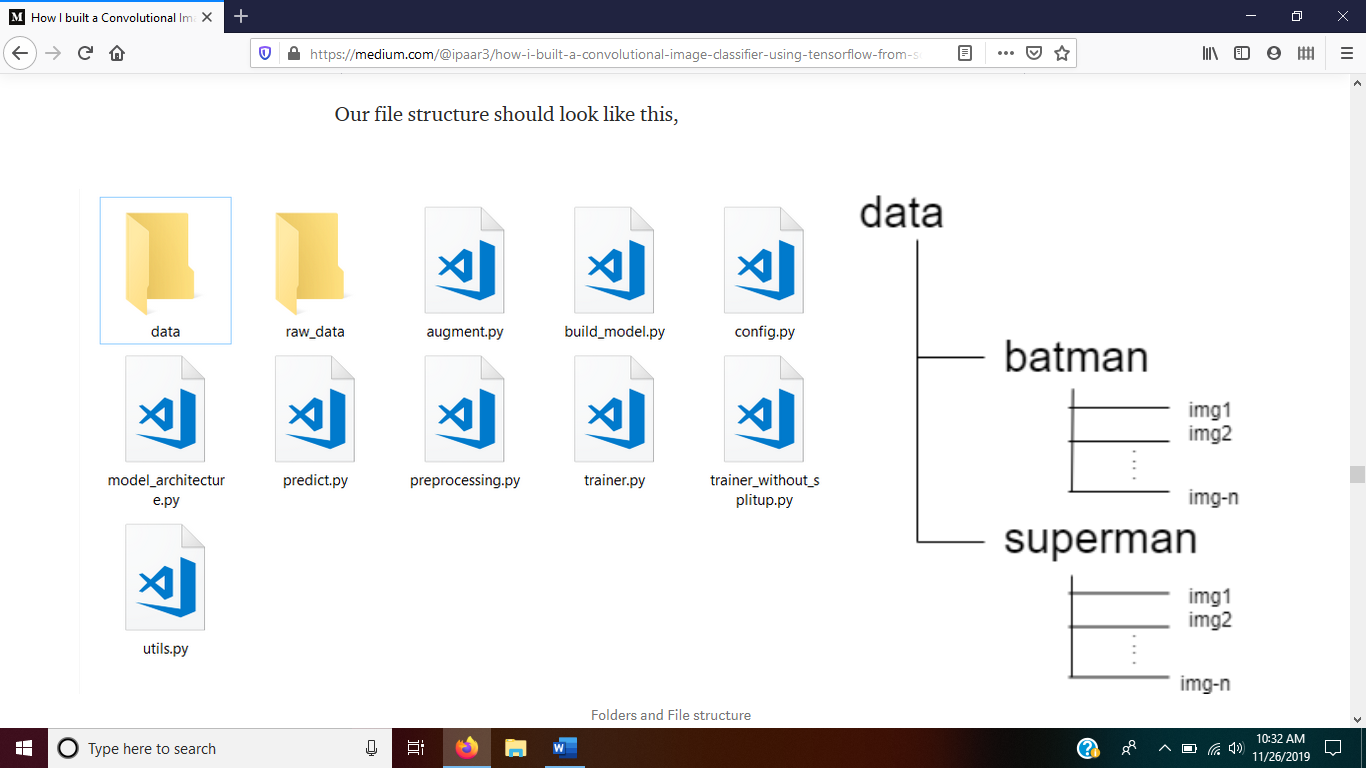
**Actual Training itself:**

* Clone this [repo](https://github.com/perseus784/BvS).
* Augment the images using Augmentor that is mentioned above.
* Put the images in their respective folders in *rawdata.*

rawdata/batman: 3810 images  
rawdata/superman: 3810 images

**Update:** If you want to train it with same data, I’ve uploaded the *data* folder [here](https://drive.google.com/open?id=1GUPBBdLlqStnxjhISkxT1qOf1XPnmRcF). Just download and extract in the same folder.

Our file structure should look like this,



Folders and File structure

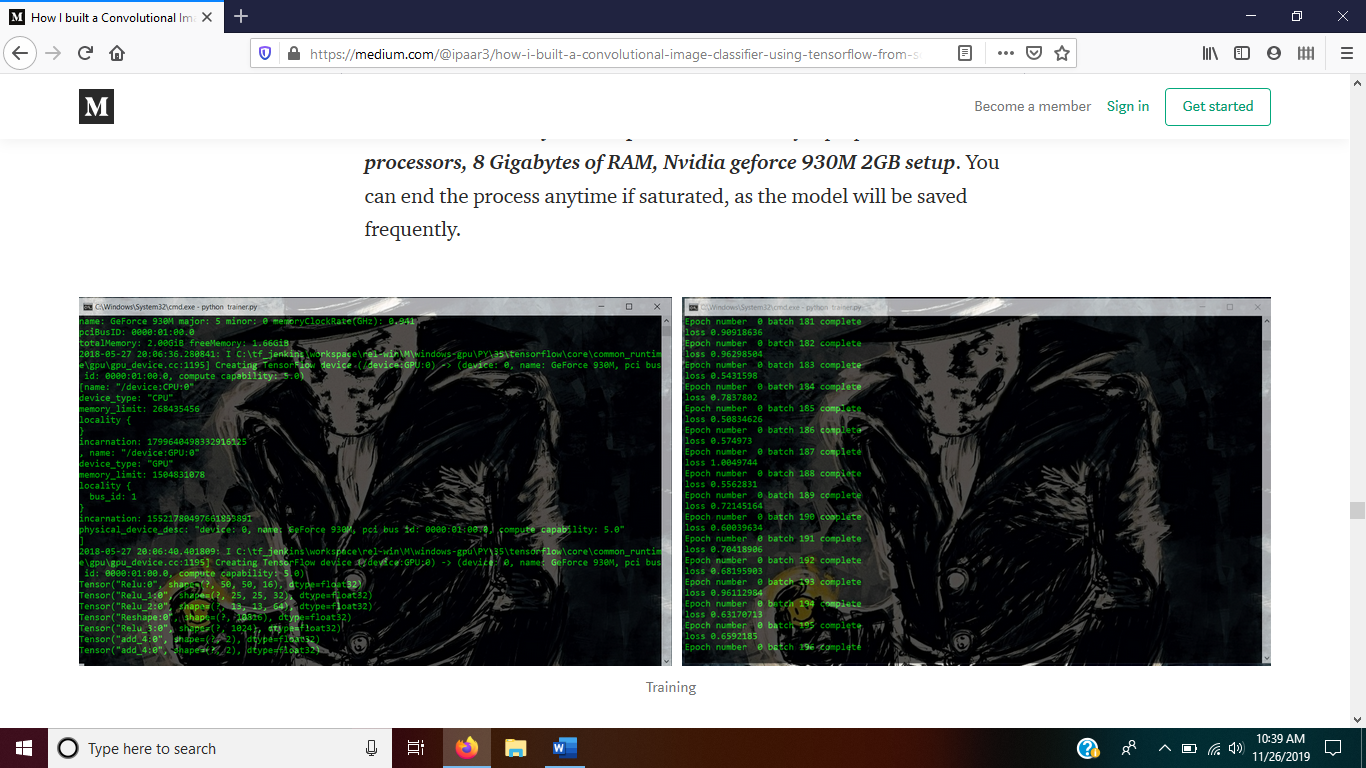
***data*** folder will be generated automatically by [trainer.py](https://github.com/perseus784/BvS/blob/master/trainer.py) from *raw\_data* if *data* folder does not exist.

***Configuration:***

If you want to edit something, you can do it using [config.py](https://github.com/perseus784/BvS/blob/master/config.py) file.

raw\_data='rawdata'data\_path='data'height=100width=100all\_classes = os.listdir(data\_path)number\_of\_classes = len(all\_classes)color\_channels=3epochs=300batch\_size=10model\_save\_name='checkpoints\\'

* Run [trainer.py](https://github.com/perseus784/BvS/blob/master/trainer.py).
* Wait for few hours.
* For me it took ***8 hrs for 300 epochs***. I did it in my laptop which has ***i5 processors, 8 Gigabytes of RAM, Nvidia geforce 930M 2GB setup***. You can end the process anytime if saturated, as the model will be saved frequently.



Training

Feel free to play with the variables.

**Saving our model:**

Once training is over, we can see a folder named checkpoints is created which contains our model for which we trained. These two simple lines does that for us in tensorflow:

saver = tf.train.Saver(max\_to\_keep=4)saver.save(session, model\_save\_name)

***You can get my pretrained model*** [***here***](https://drive.google.com/open?id=18ZzIYCkdTfYQQ1-tzpcfMuxzDwOJ0CU6)***.***

**Yeah, you’ve done it.**

Yes, you have built your own accurate image classifier using CNNs from scratch. Now, let’s get the results of what we built.

To do that, we need a script that can run our model and classify the image.

We have three files in our checkpoints folder,

* .meta file — it has your graph structure saved.
* .index — it identifies the respective checkpoint file.
* .data — it stores the values of all the variables.

**How to use it?**

Tensorflow is so well built that, it does all the heavy lifting for us. We just have to write four simple lines to load and infer our model.

#Create a saver object to load the model  
saver = tf.train.import\_meta\_graph  
 (os.path.join(model\_folder,'.meta'))#restore the model from our checkpoints folder  
saver.restore(session,os.path.join('checkpoints','.\\'))#Create graph object for getting the same network architecture  
graph = tf.get\_default\_graph()#Get the last layer of the network by it's name which includes all the previous layers too  
network = graph.get\_tensor\_by\_name("add\_4:0")

Yeah, simple. Now that we got our network as well as the tuned values, we have to pass an image to it using the same placeholders(Image, labels).

im\_ph= graph.get\_tensor\_by\_name("Placeholder:0")  
label\_ph = graph.get\_tensor\_by\_name("Placeholder\_1:0")

If you run it now, you can see the output as [1234,-4322] like that. While this is right as the maximum value index represents the class, this is not as convenient as representing it in 1 and 0. Like this [1,0]. For that we should include a line of code before running it,

network=tf.nn.sigmoid(network)

While we could have done this in our training architecture itself and nothing would have changed, I want to show you that, you can add layers to our model even now, even in prediction stage. Flexibility.

**Inference time:**

Your training is nothing, if you don’t have the will to act.

— Ra’s Al Ghul.

To run a simple prediction,

* Edit the image name in [predict.py](https://github.com/perseus784/BvS/blob/master/predict.py).
* Download the model files and extract in the same folder.
* Run [predict.py](https://github.com/perseus784/BvS/blob/master/predict.py).

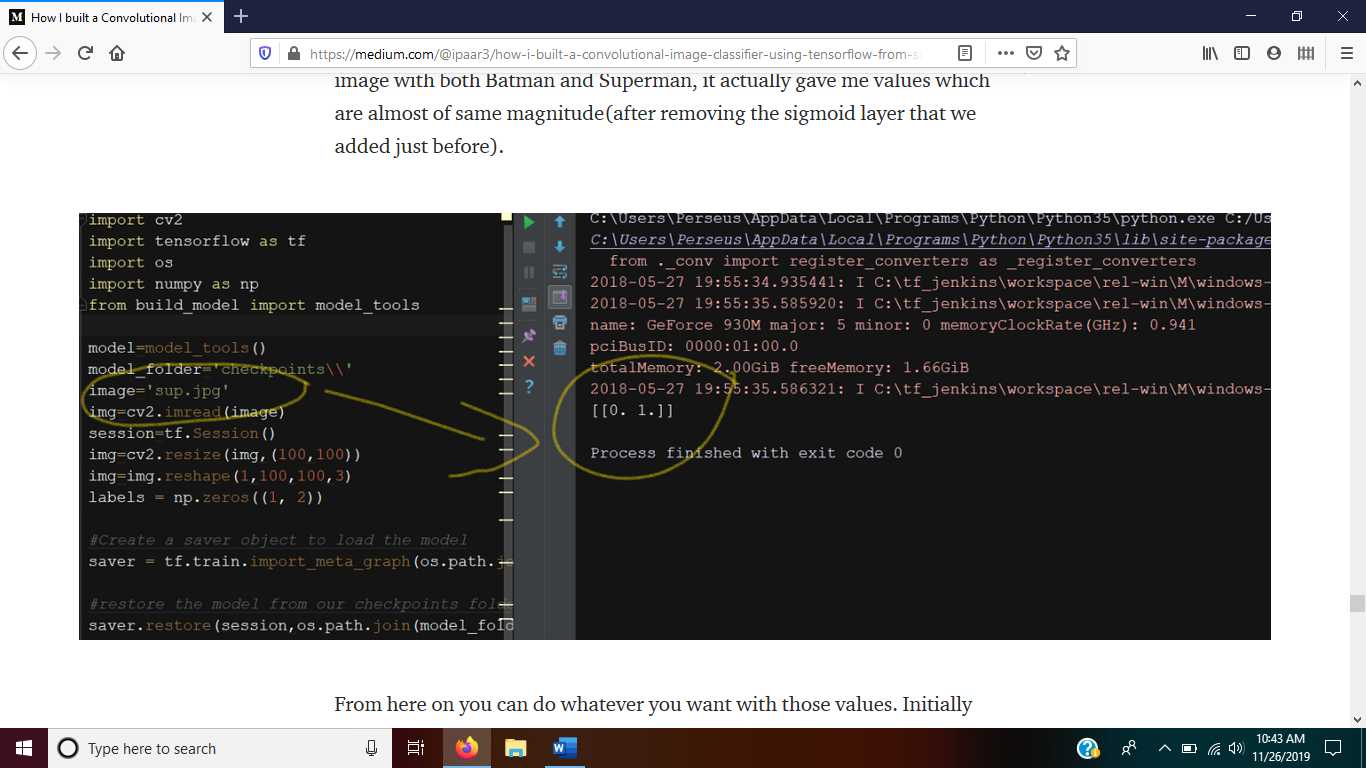
image='sup.jpg'  
img=cv2.imread(image)  
session=tf.Session()  
img=cv2.resize(img,(100,100))  
img=img.reshape(1,100,100,3)  
labels = np.zeros((1, 2))*# Creating the feed\_dict that is required to be feed the io:*feed\_dict\_testing = {im\_ph: img, label\_ph: labels}result=session.run(network, feed\_dict=feed\_dict\_testing)  
print(result)

You can see the results as [1,0](Batman), [0,1](Superman) corresponding to the index.

*please note that this is not a output in one-hot encoding.*

**Accuracy:**

It is actually pretty good. It is almost right all the time. I even gave it an image with both Batman and Superman, it actually gave me values which are almost of same magnitude(after removing the sigmoid layer that we added just before).

From here on you can do whatever you want with those values. Initially loading the model will take some time(70 seconds) but once the model is loaded, you can put a for loop or something to throw in images and get output in a second or two!

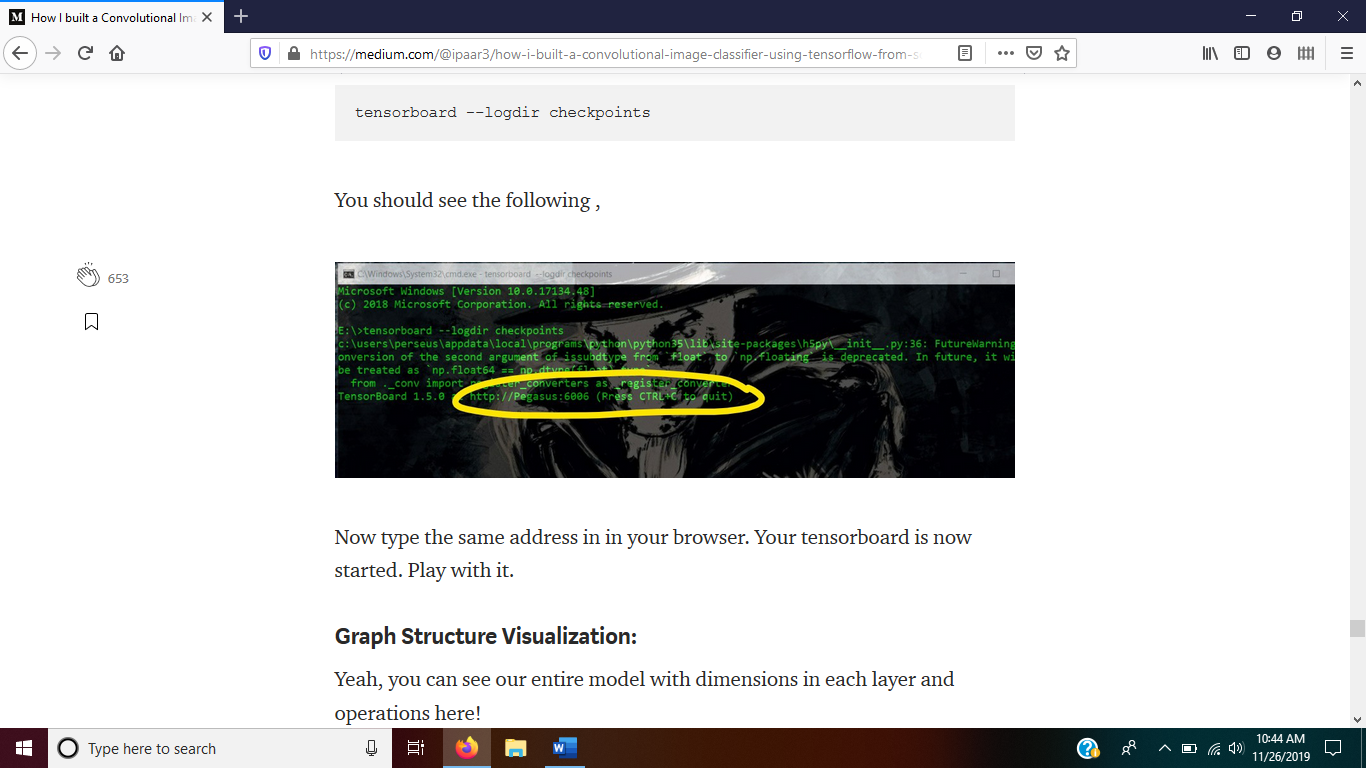
**Tensorboard:**

I have added some additional lines in the training code for tensorboard options. Using tensorboard we can track progress of our training even while training and after. You can also see your network structure and all the other components inside it.It is very useful for visualizing the things happening.

To start it, just go to the directory and open command line,

tensorboard --logdir checkpoints

You should see the following ,



Now type the same address in in your browser. Your tensorboard is now started. Play with it.

**Graph Structure Visualization:**

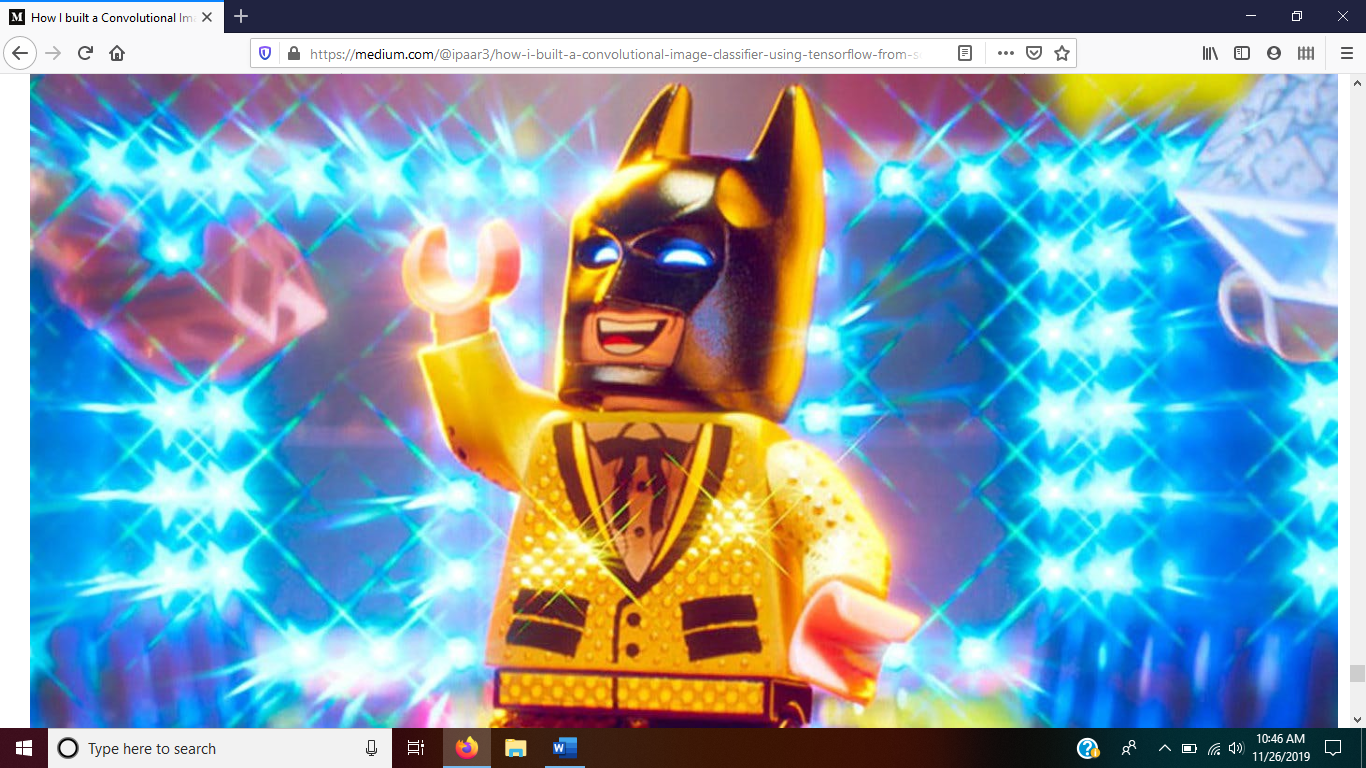
Yeah, you can see our entire model with dimensions in each layer and operations here!



**Future Implementations:**

While this works for Binary classification, it will also work for Multiclass classification but not as well. We might need to alter architecture and build a larger model depending on the number of classes we want.

And Batman wins!!!



**If you liked this article, please help others find it by holding that clap icon for a while. Thank you!**

*I’ve written a introduction post for* [*Reinforcement Learning*](https://medium.com/@ipaar3/saturnmind-94586f0d0158)*. Please read that too.*

Any suggestions, doubts, clarifications please do contact me. Mail: ipaar3@gmail.com or comment or raise an issue in Github.

***Thank you again!.***

* [Machine Learning](https://medium.com/tag/machine-learning)
* [Image Classification](https://medium.com/tag/image-classification)
* [TensorFlow](https://medium.com/tag/tensorflow)
* [Deep Learning](https://medium.com/tag/deep-learning)
* [Neural Networks](https://medium.com/tag/neural-networks)

[](https://medium.com/@ipaar3?source=follow_footer--------------------------follow_footer-)

Written by

[**Paarvendhan**](https://medium.com/@ipaar3?source=follow_footer--------------------------follow_footer-)

[See responses (9)](https://medium.com/p/f852c34e1c95/responses/show?source=follow_footer--------------------------follow_footer-)

**More From Medium**

**Related reads**

**[A Simple Image Classification Walkthrough With Tensorflow 2.0](https://towardsdatascience.com/neural-networks-a-simple-image-classification-walkthrough-a0bca500f836?source=post_recirc---------0------------------)**

[Jagerynn Ting Verano](https://medium.com/@jagerynnv?source=post_recirc---------0------------------)

[Jagerynn Ting Verano](https://medium.com/@jagerynnv?source=post_recirc---------0------------------) in [Towards Data Science](https://towardsdatascience.com/?source=post_recirc---------0------------------)

[Jun 5](https://towardsdatascience.com/neural-networks-a-simple-image-classification-walkthrough-a0bca500f836?source=post_recirc---------0------------------) · 8 min read

**845**

**Related reads**

**[Transfer Learning for Image Classification using Keras](https://towardsdatascience.com/transfer-learning-for-image-classification-using-keras-c47ccf09c8c8?source=post_recirc---------1------------------)**

[George Seif](https://medium.com/@george.seif94?source=post_recirc---------1------------------)

[George Seif](https://medium.com/@george.seif94?source=post_recirc---------1------------------) in [Towards Data Science](https://towardsdatascience.com/?source=post_recirc---------1------------------)

[Dec 19, 2018](https://towardsdatascience.com/transfer-learning-for-image-classification-using-keras-c47ccf09c8c8?source=post_recirc---------1------------------) · 4 min read

**902**

**Also tagged Deep Learning**

**[Microsoft Introduces Icebreaker to Address the Famous Ice-Start Challenges in Machine Learning](https://towardsdatascience.com/microsoft-introduces-icebreaker-to-address-the-famous-ice-start-challenges-in-machine-learning-10ee0363ede3?source=post_recirc---------2------------------)**

[Jesus Rodriguez](https://medium.com/@jrodthoughts?source=post_recirc---------2------------------)

[Jesus Rodriguez](https://medium.com/@jrodthoughts?source=post_recirc---------2------------------) in [Towards Data Science](https://towardsdatascience.com/?source=post_recirc---------2------------------)

[Nov 26](https://towardsdatascience.com/microsoft-introduces-icebreaker-to-address-the-famous-ice-start-challenges-in-machine-learning-10ee0363ede3?source=post_recirc---------2------------------) · 5 min read

**9**

**[Discover Medium](https://medium.com/about?autoplay=1&source=post_page-----f852c34e1c95----------------------)**

Welcome to a place where words matter. On Medium, smart voices and original ideas take center stage - with no ads in sight. [Watch](https://medium.com/about?autoplay=1&source=post_page-----f852c34e1c95----------------------)

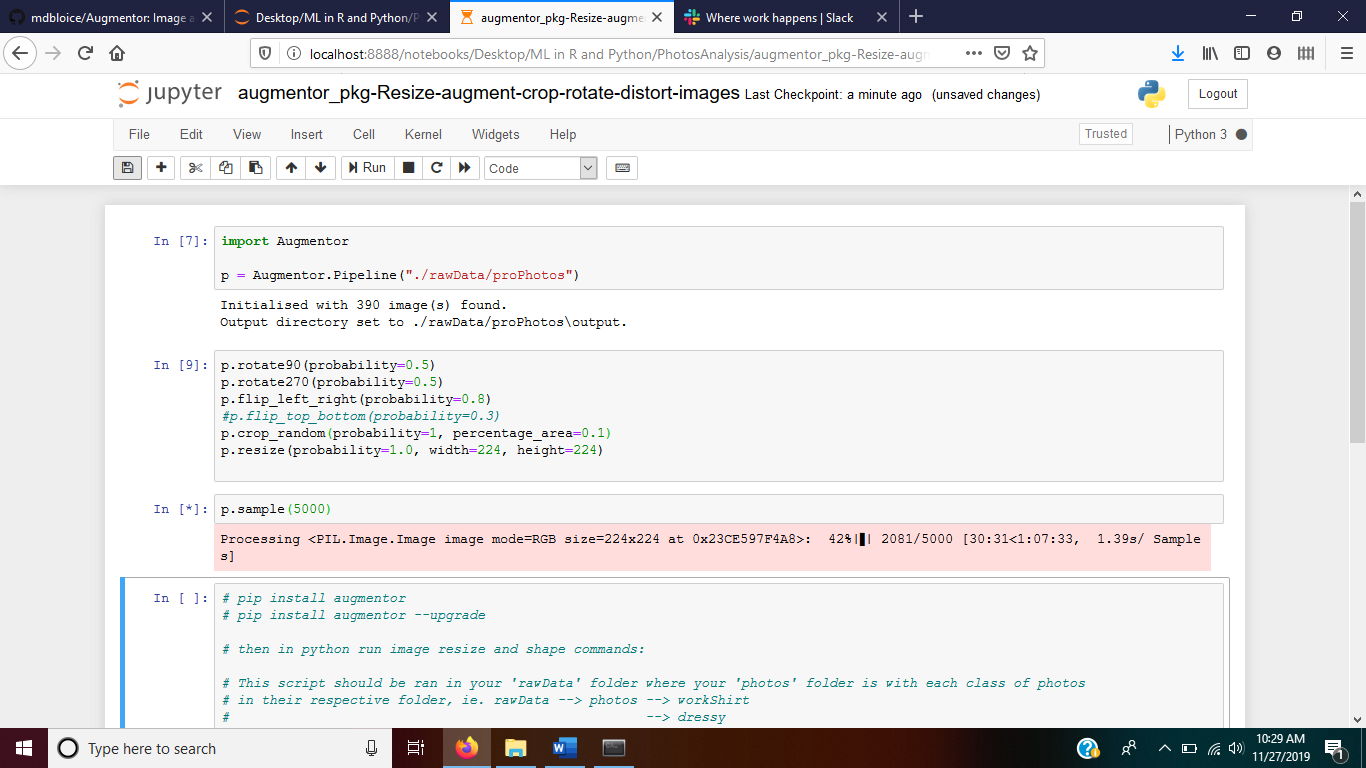
**[Make Medium yours](https://medium.com/topics?source=post_page-----f852c34e1c95----------------------)**

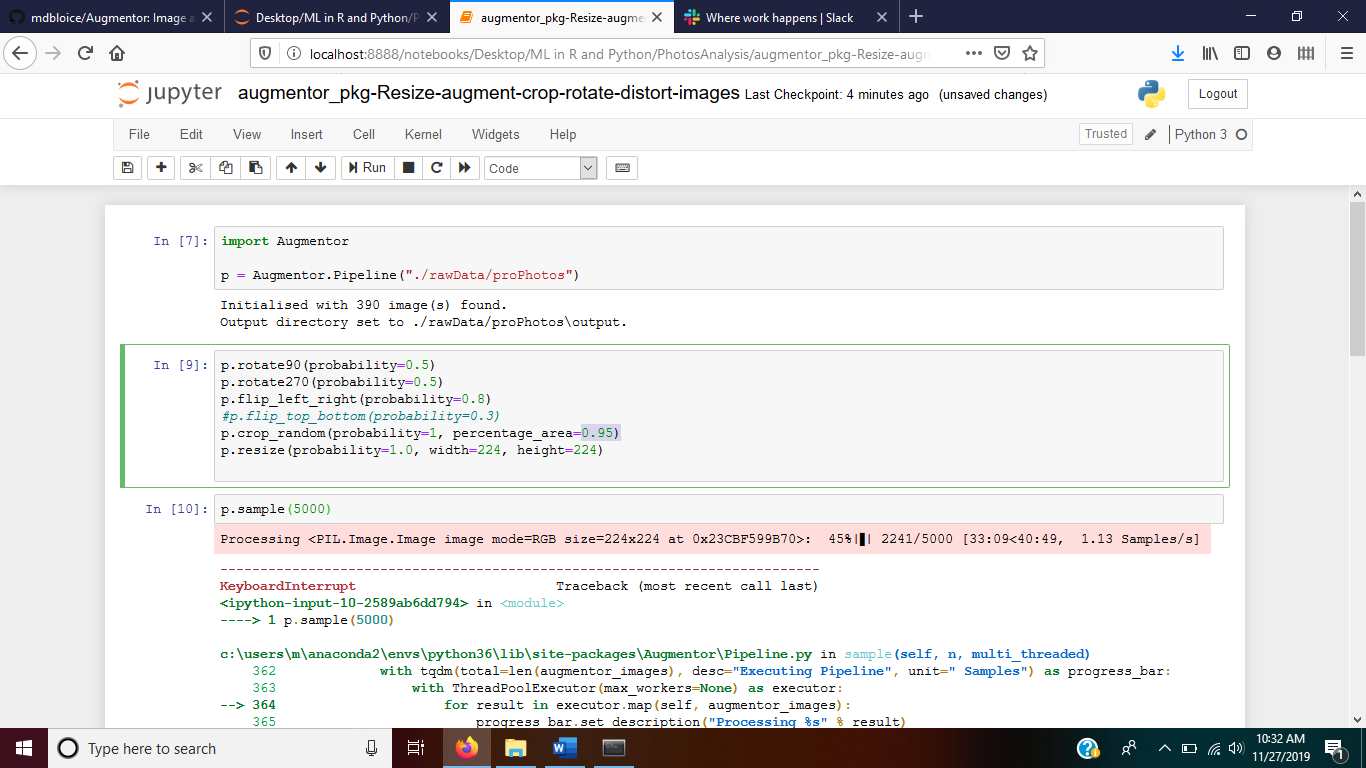
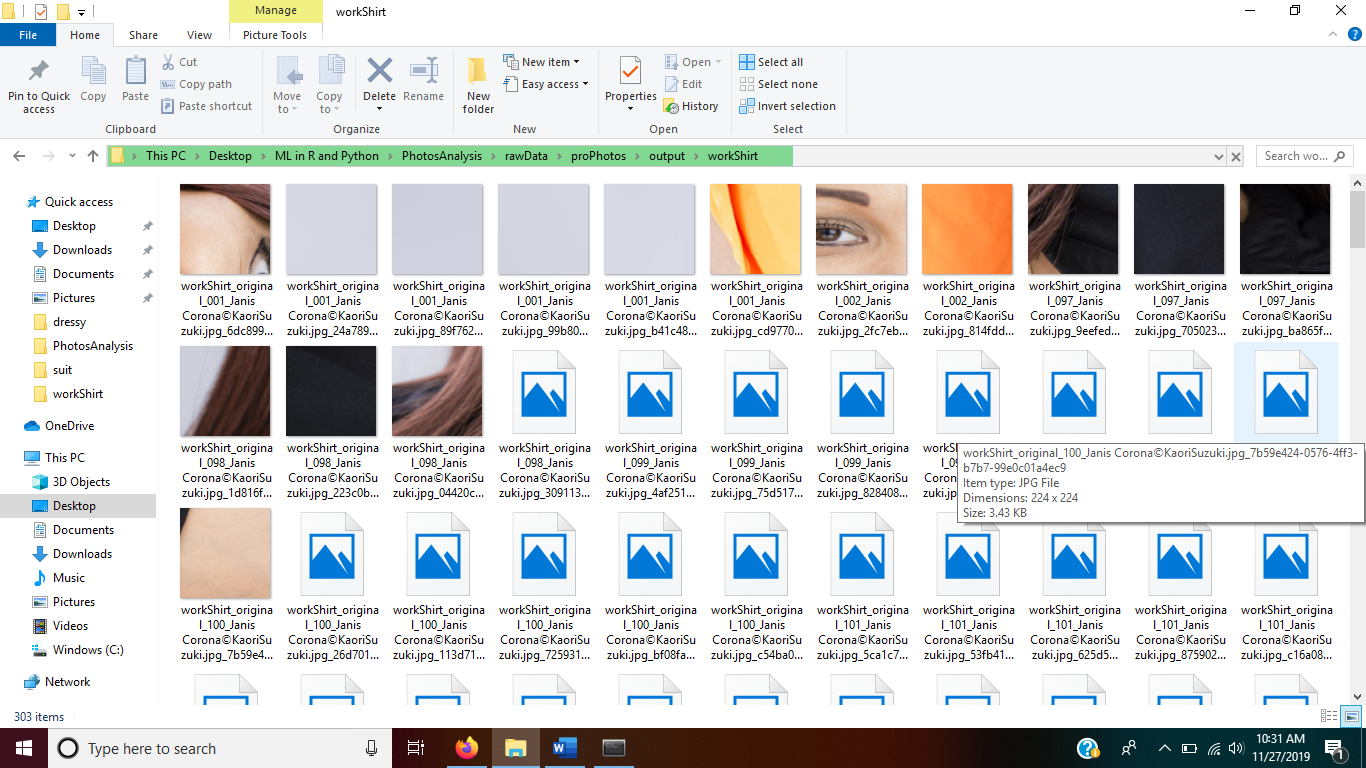
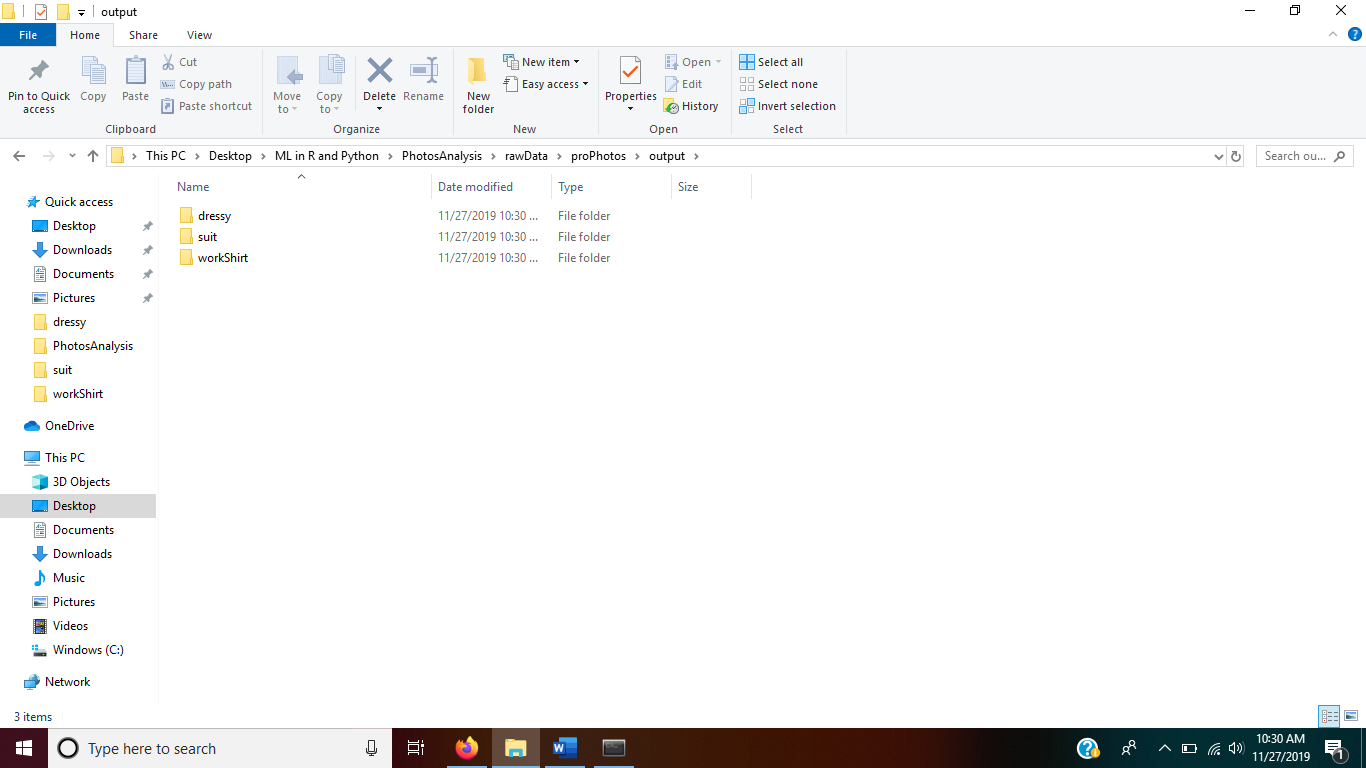
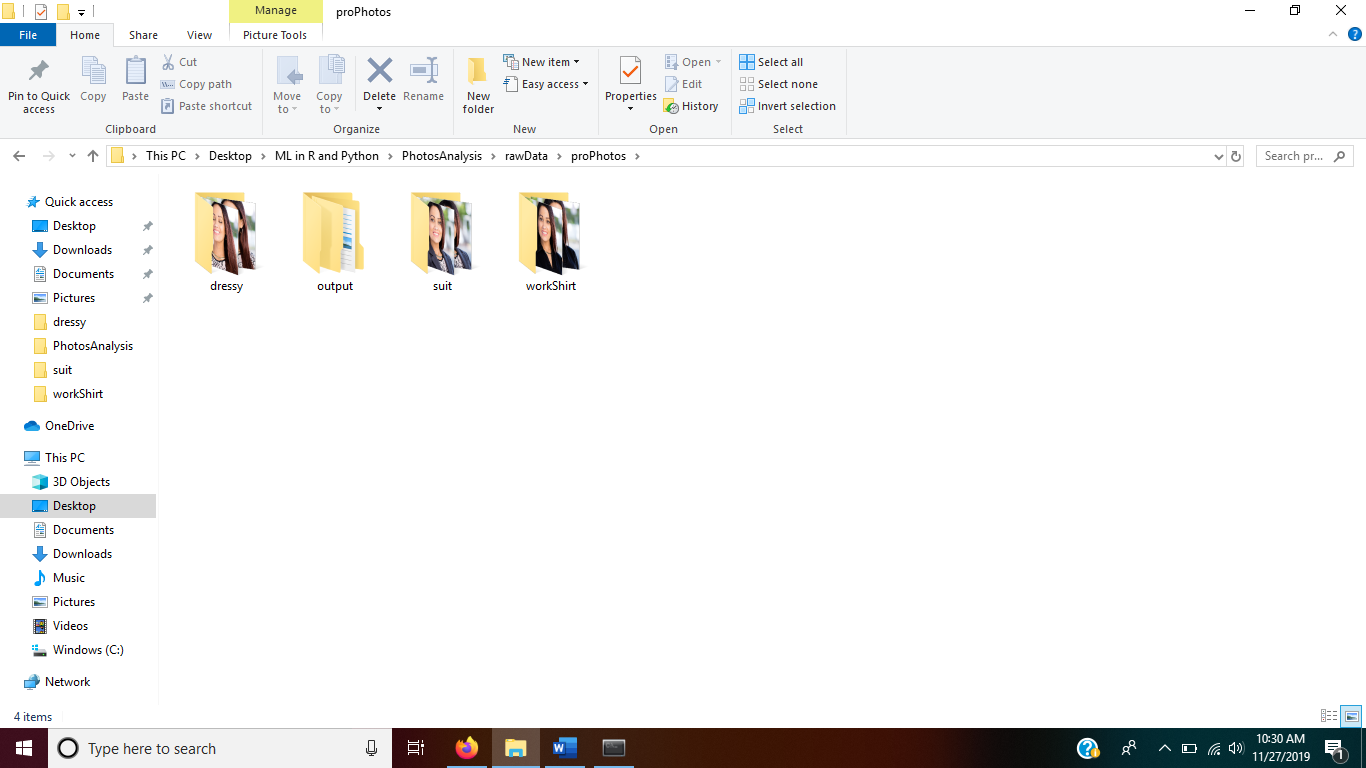
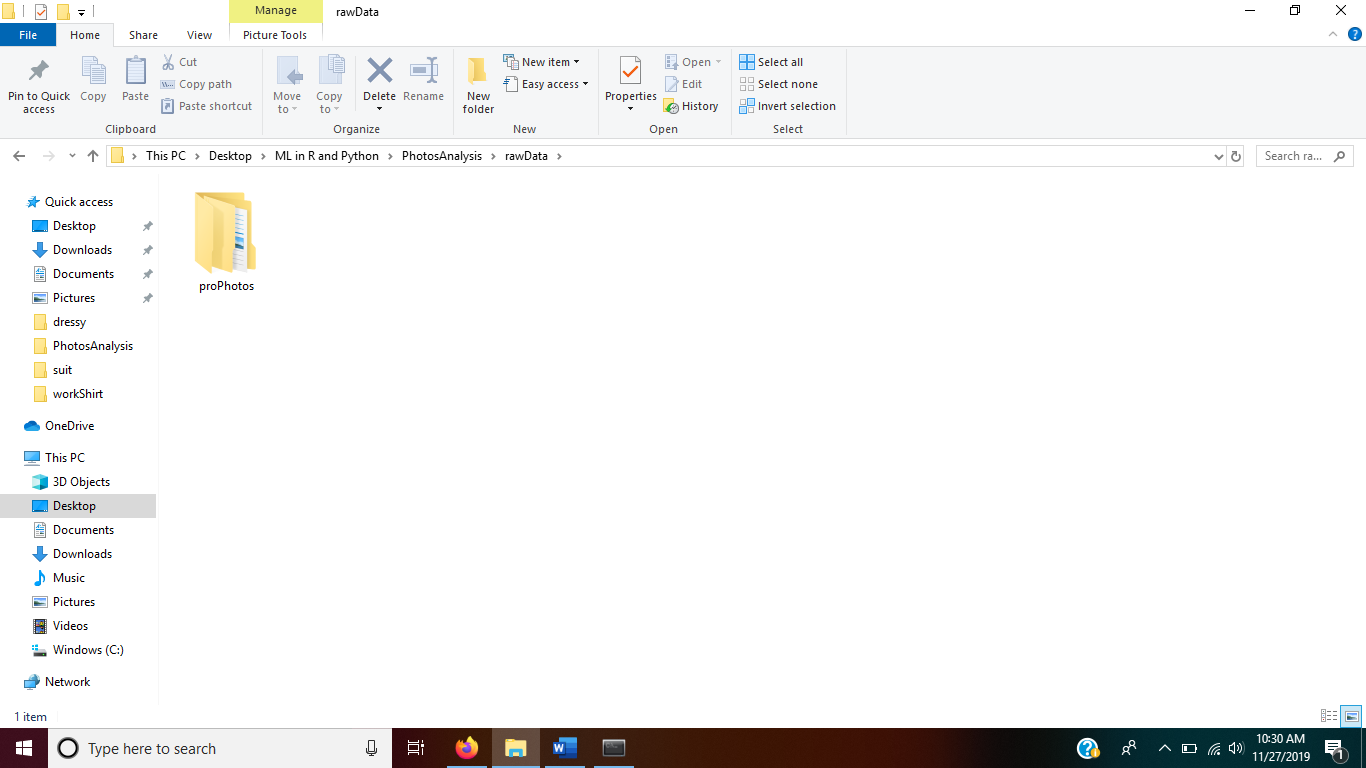
Follow all the topics you care about, and we’ll deliver the best stories for you to your homepage and inbox. [Explore](https://medium.com/topics?source=post_page-----f852c34e1c95----------------------)

**[Become a member](https://medium.com/membership?source=post_page-----f852c34e1c95----------------------)**

Get unlimited access to the best stories on Medium — and support writers while you’re at it. Just $5/month. [Upgrade](https://medium.com/membership?source=post_page-----f852c34e1c95----------------------)

[About](https://medium.com/about?autoplay=1&source=post_page-----f852c34e1c95----------------------)[Help](https://help.medium.com/?source=post_page-----f852c34e1c95----------------------)[Legal](https://medium.com/policy/9db0094a1e0f?source=post_page-----f852c34e1c95----------------------)





Changed to a larger crop size, was 0.10 but changed to 0.95, this keeps most of image not 0.10 which throws out 0.90 of image.