**Title:** Line to Live (Conversion of 2D image to texture for 3D models using Tensorflow and opencv Python)

**Scope  of Work:** Conversion of 2D image to texture for 3D model using conventional 3D graphics design software is tedious and involves high degree of human intervention, however if a Deep Learning model is so prepared that it can produce texture of 3D models of a group of similar looking objects from a single 2D image, it can save considerable time which can be further utilized in development of other aspects of the model including aesthetics.

**Abstract:** A single image is only a projection of 3D object into a 2D plane, so some data from the higher dimension space usually get lost in the lower dimension representation. Therefore from a single-view 2D image, there will never be enough data to construct its 3D component. A method to create the 3D perception from a single 2D image requires prior knowledge of the 3D shape in itself.

We reason that, if  2D projections are of any good, then we can render different 2D projections from *new* viewpoints, it should resemble the projections from the ground truth 3D model too.

         Input: 2D Image

         Output: mapped texture of a 3D model

In our application we presented this in a fun filled manner that can map visitor engagement and  inculcate learning attitude among the masses, specifically among young students. Here visitors are supposed to paint line drawing of a fish among a few available variants and the fish comes alive in the screen in the form of a 3D model generating the texture from the image painted by the visitor. Another aspect worth mentioning in this regard, is use of tensorflow 2.3 GPU edition implemented with python working together as the backend of the application and Unity 3D working as the front end of the software. Since both the applications are graphics intensive, GPU version of Tensorflow 2.3 has been opted for development.

**Proposed approach:** The process of developing a 3d model is yet not a job a machine can perform independently without human intervention. The process complexity gets many folded when it comes to texturing, starting from bump map generation to wrap up, it needs minutes to be taken care of. Here in our noble approach we tried automating the second phase of the problem i.e. texture generation for a 3D model with a given pretext. Here, the system is fed with a 2D image and it generates the 3D texture with the help of image processing and AI. In order to impose relevance, we have tried it with 3D models of two different types of fishes, a koi fish and a gold fish. To engage our valued patrons, the kids in particular, we provide them with line drawing of any of the two fishes mentioned above and ask them to colour it the way they want to see the fish coming live in a virtual aquarium in real-time.

In a nutshell two different yet concurrent applications run parallely to make it happen. The process of converting an image into a texture is done with the help of a python script at the backend. The steps are as follows:

1. line drawing of a fish is served to a client through a web interface. Then it is painted and sent to the server.

2. A python scrip loads the image from disk.

3. An SSD Mobile net V2 FPN Lite 640 x 640 model trained with tensorflow 2 tries to classify the image.

4. If classification returns valid ID, unwanted parts of the image is filtered.

5. Regions other than that of interest are made transparent so that it doesn't produce glitches.

6. A replica of the filtered image is produced to generate a bilateral texture for the sketched fish.

7. Texture map of eyes and scales are overlapped on the base image.

8. Images so produced are stiched and mapped into a single image for further processing by an Unity 3D front end application.

9. The image is saved on disk.

10. This entire process runs on Nvidia Quadro M4000 GPU with compute capability 5.2 to leverage the cores of the intel xeon cpu to run resource intensive Unity front end.

The front end of the application is developed with Unity 3D editor and it involves the following steps

1. continuous scanning of the image saved by the previously mentioned python back end.

2. producing normal map from the image scanned.

3. overlaying the normal map on the 3D model.

4. superimposing the image on top of the bump map.

5. applying shader according to the input image.

6. finalization of texture and displaying the same in real time.

**Process Flow and Logic Flow:**

**Process flow for feeding through web interface:**

Connect to local WiFi network

Have QR Code scanner app installed ?

Open QR code scanner in your mobile device

If you don’t have QR code scanner app type the ip address directly in your browser i.e. 192.168.1.33:5000

No Yes

Scan the QR code and open the link scanned in a browser

Select image of a fish from the icons displayed on screen

Pick a color of your choice from the palette and adjust brush diameter to paint

Fill color and make patterns if felt like, color filling may even not be precise

Send/Share the image pressing the send / share button located on top of the screen

Wait for a while for the fish painted and shared to appear on the display in front

Want to change the texture or try another fish?

Yes No

**Logic Flow for feeding image through web interface:**

Navigate to the folder containing the Unity 3D Application and run the exe

Open virtual environment in Anaconda for Tensorflow GPU, navigate to the folder containing the python script for Image processing and recognition, run the script

Open a virtual environment in Anaconda, navigate to the folder containing the python script for python server and run the script

Load the image from Disk and check for time stamp of last updation

Load the image from Disk and apply necessary formatting to convert it into an numpy array to run image recognition on it using tensorflow gpu

Scan the designated ip and port to check for valid subscribers/clients

Updated within last 2 seconds ?

No

Is any of the clients sharing an image ?

No

No Yes

Is it valid image of a fish ?

Create a normal map from the image loaded, prepare appropriate shader, generate necessary material and apply the texture along with other maps to its source

Yes

Receive the image and save into a designated folder in the disk

Yes

Crop the image and create a difference image substracting the current image from the image loaded and validated in the previous iteration

Spawn an istance of the 3D model of a fish and apply the material so prepared along with texture for final wrap up

Animate the model and let it traverse a path within the viewport of the camera

Is it the same image scanned in the previous iteration ?

Yes

No

Is traversal complete?

No

Yes

Increase brightness and mask the image as per masking map already present in the disk

Bring the fish out of the viewport of the camera and destroy it

Enhance color of the image and overlay scale and eye map on the image. Load a blank canvas to prepare the final texture

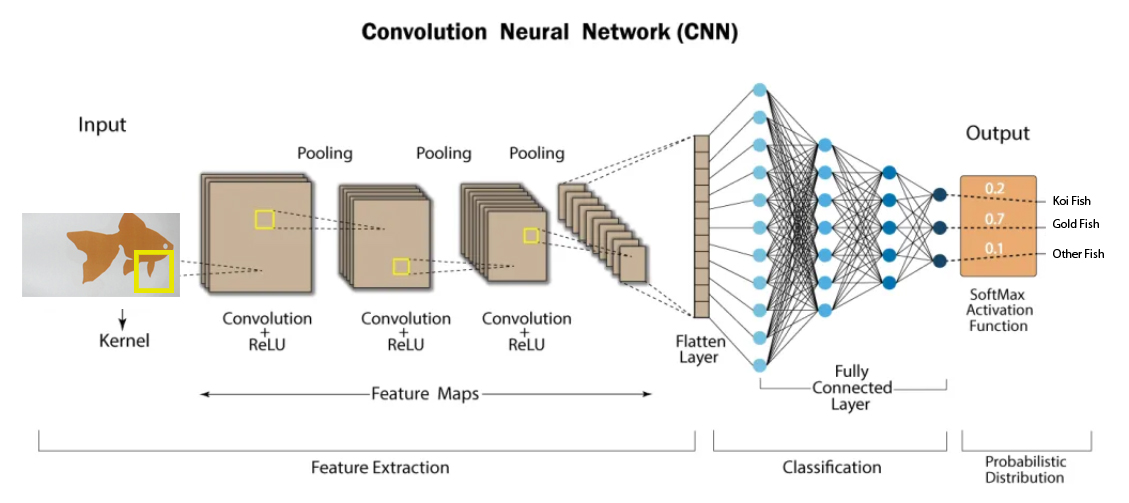
Paste one face of the processed image on canvas, flip / rotate it as per requirement and paste the other face of the texture

Save the texture on disk at a particular location for further utilization in Unity

**Image Classification using Tensorflow GPU:**

Deep learning is a subfield of machine learning which uses artificial neural networks that is inspired by the structure and function of the human brain. Despite being a very new approach, it has become very popular recently. Deep learning has achieved much higher success in many applications where machine learning has been successful at certain rates. In particular It is preferred in the classification of big data sets because it can provide fast and efficient results. In this study, we used Tensorflow, one of the most popular deep learning libraries to classify XML dataset, which is frequently used in data analysis studies. Using Tensorflow, which is an open source artificial intelligence library developed by Google, we have studied and found Rectified Linear Unit performs the best. In this Study, Convolutional Neural Network (CNN) and SoftMax classifier are used as deep learning artificial neural network.

The first layer is Convolution layer. Imagine reading the input matrix from the top left corner for pic. The software then selects a small matrix called a filter (or neuron or core). The filter then generates a convolution i.e. it moves along the input image. The task of the filter is to multiply that value by the original pixel value. All these multiplications are additive. Finally, get the number. The filter reads only the image in the upper left corner, so move the unit further to the right and perform the same operation. After passing through all filters, the matrix is acquired, but less than the input matrix.



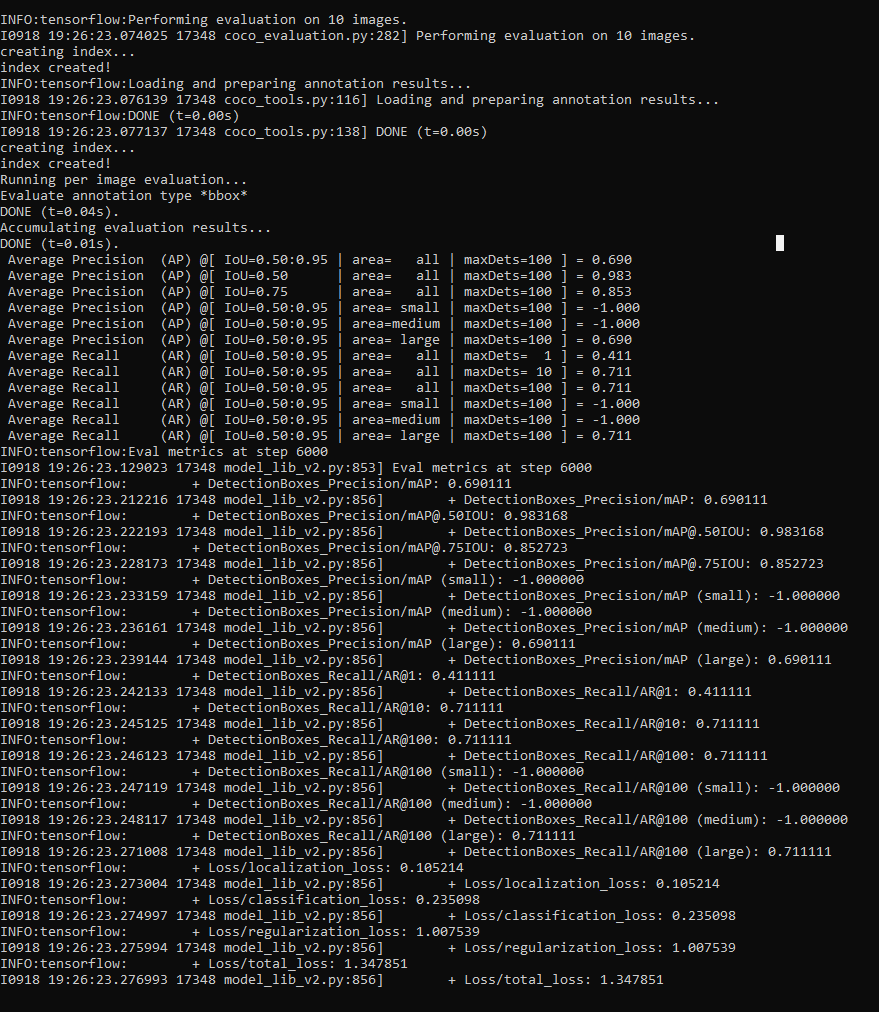
From a human point of view, this process is like recognizing the borders and simple colors of a picture. However, the entire network is required to identify high-level attributes such as fins and tails of a fish. This network consists of several convolutional networks that combine nonlinear and pooling layers. When the image pass over the convolution layer, the result of the first layer enters the input to the second layer. This also further for each convolutional layer.

A non-linear layer is added after each convolution operation. There is an activation function that brings nonlinear characteristics. Without this attribute, the network would not be powerful and the response would not be variable.

The nonlinear layer follows by the pool layer. Applies to the width and height of the image and performs a down sampling operation on them. As a result, the volume of the image is reduced. In other words, if some functions (as for example boundaries) are identified in the previous convolution operation, the detailed image of the subsequent processing becomes unnecessary and the image is compressed to less detail image.

After completing a series of convolutional layers, nonlinear layers, and pooling layers, one needs to connect the fully connected layers. This layer gets the output information from the convolutional network. Attaching a fully connected layer to the end of the network produces an N-dimensional vector. Where N is the number of classes from which the model selects the desired class.

On suucessful completion of training we evaluated the model and found the following details:



**Code snippets and Explanation of Code:**

Python script responsible for formatting and classification of the loaded image:

image\_rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

    image\_expanded = np.expand\_dims(image\_rgb, axis=0)

    input\_tensor = tf.convert\_to\_tensor(image)

    input\_tensor = input\_tensor[tf.newaxis, ...]

    detections = detect\_fn(input\_tensor)

    num\_detections = int(detections.pop('num\_detections'))

    detections = {key: value[0, :num\_detections].numpy()

                for key, value in detections.items()}

    detections['num\_detections'] = num\_detections

    detections['detection\_classes'] = detections['detection\_classes'].astype(np.int64)

Python script to generate difference image to verify if same image has been received by the server to avoid redundancy in processing :

difference = cv2.subtract(croppedImage, goldImage\_old)

imgD = cv2.cvtColor(difference, cv2.COLOR\_BGR2RGB)

imageD = Image.fromarray(imgD)

imageD = imageD.convert("RGBA")

datas1 = imageD.getdata()

brightPixel\_count = 0

for k in range(len(datas1)):

pixelValue1 = datas1[k]

        RD = pixelValue1[0]

        GD = pixelValue1[1]

        BD = pixelValue1[2]

        if(RD > 20 or GD > 20 or BD > 20):

         brightPixel\_count += 1

                #print('Bright Pixel Count: ', brightPixel\_count)

         if(brightPixel\_count < 1500 and brightPixel\_countOld > 1500):

          trigger = 1

         if(trigger == 1 and brightPixel\_count < 1500 and brightPixel\_countOld < 1500):

               print('Time to save Gold Fish Image, Image chage Event occured')

               detectedID = 1

               trigger = 0

brightPixel\_countOld = brightPixel\_count

goldImage\_old = croppedImage

Application: 2D image to 3D conversion without using costly depth camera and scanning software

Function responsible for adjusting brightness of the image used to finally produce texture for the model :

def increase\_brightness(img, value=30):

    hsv = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)

    h, s, v = cv2.split(hsv)

    lim = 255 - value

    v[v > lim] = 255

    v[v <= lim] += value

    final\_hsv = cv2.merge((h, s, v))

    img = cv2.cvtColor(final\_hsv, cv2.COLOR\_HSV2BGR)

    return img

Code to generate masked partial texture based on mask image:

img = cv2.cvtColor(croppedImage, cv2.COLOR\_BGR2RGB)

        image1 = Image.fromarray(img)

        image2 = Image.open('resourceImages\_gold/transMap\_goldNew.png')

        image1 = image1.convert("RGBA")

        image2 = image2.convert("RGBA")

        datas1 = image1.getdata()

        datas2 = image2.getdata()

        newData = []

        i = 0

        for i in range(len(datas2)):

            pixelValue1 = datas1[i]

            pixelValue2 = datas2[i]

            if pixelValue2[0] == 0 and pixelValue2[1] == 0 and pixelValue2[2] == 0:

                newData.append((255,255,255,0))

            else:

                R = pixelValue1[0]

                G = pixelValue1[1]

                B = pixelValue1[2]

                A = pixelValue2[3]

                newData.append((R,G,B,A))

        image1.putdata(newData)

        image1.save("producedImages\_gold/maskedG.png", "PNG")

Code for color enhancement and overlaying scale map:

enh\_col = ImageEnhance.Color(image1) #txImage

color = 1.85

txImage = enh\_col.enhance(color)

bgImage = Image.open('resourceImages\_gold/bgTr.png')

backGround = bgImage.copy()

backGround = backGround.convert("RGBA")

txFlip = ImageOps.flip(txImage)

backGround.paste(txImage, (30, 490), txImage)  #30, 490

backGround.paste(txFlip, (25, 23), txFlip) #25, 23

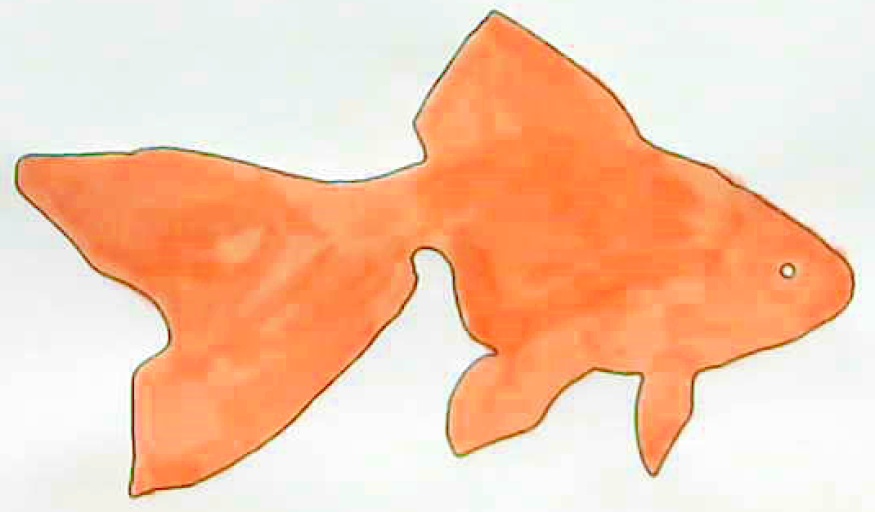
overLay = Image.open("resourceImages\_gold/scaleOverlay.png")

backGround.paste(overLay, (0, 0), overLay)

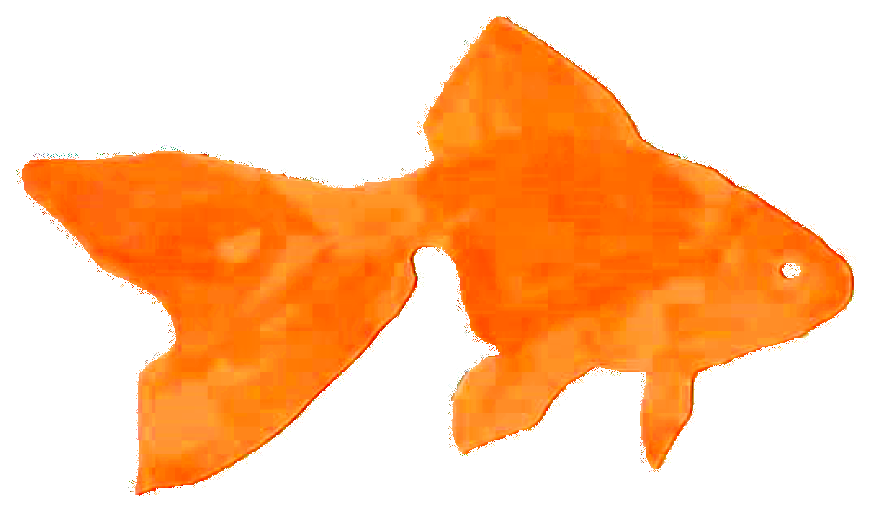
backGround.save('producedImages\_gold/finalTx.png', "PNG")

Input and Output Images:

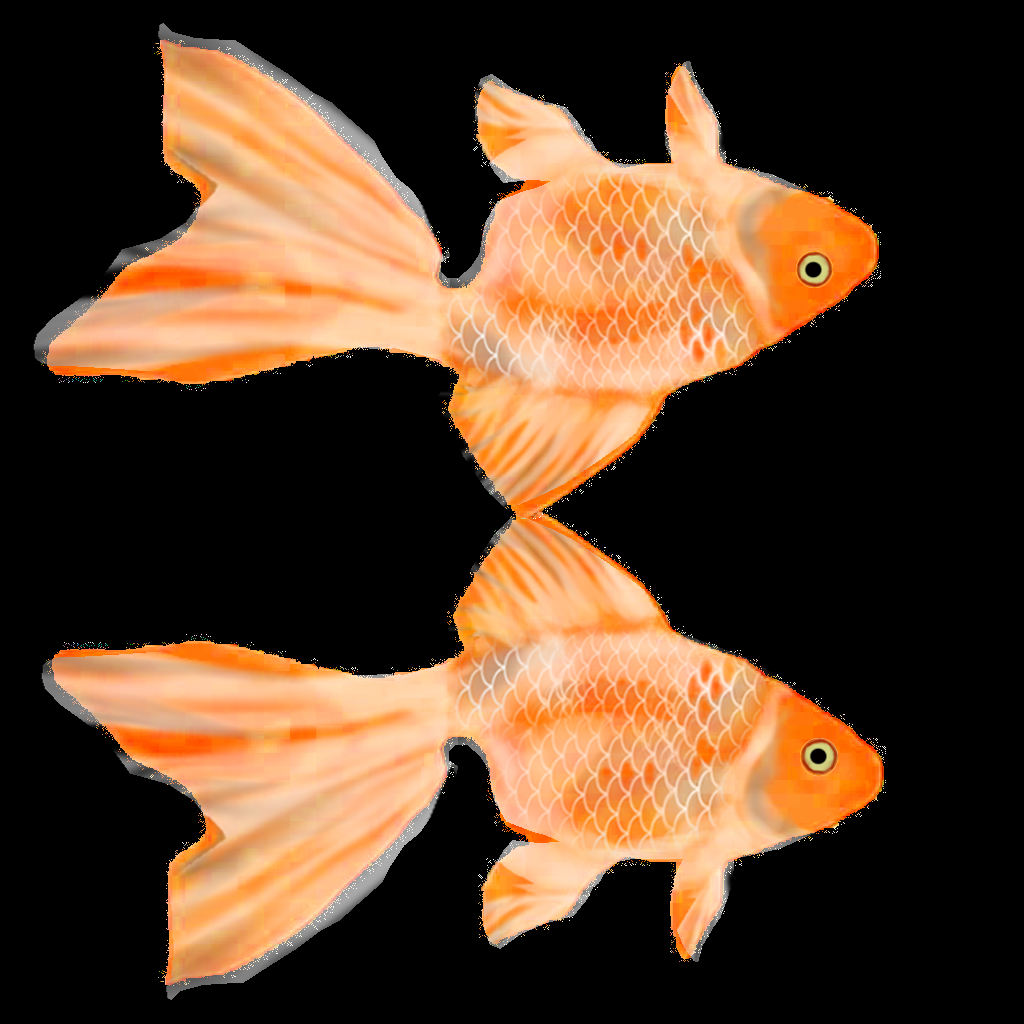
Scanned Image



After masking, white-balancing and color correction



Final Texture



Finally generated 3D model with the above texture:



**Discussion:**

While our model is capable of producing natural textures of comparable quality to non-parametric texture synthesis methods, our synthesis procedure is computationally more expensive. Nevertheless, both in industry and academia, there is currently much effort taken in order to make the evaluation of deep neural networks more efficient. Since our texture synthesis procedure builds exactly on the same operations, any progress made in the general field of deep convolutional networks is likely to be transferable to our texture synthesis method. Thus we expect considerable improvements in the practical applicability of our texture model in the near future.

**References:**

1. <https://paperswithcode.com/paper/texture-synthesis-using-convolutional-neural>
2. <https://arxiv.org/abs/1505.07376>
3. <https://dl.acm.org/doi/10.5555/2969239.2969269>
4. <https://ieeexplore.ieee.org/abstract/document/7813173>
5. <https://github.com/armaanpriyadarshan/Training-a-Custom-TensorFlow-2.x-Object-Detector>