The normal inference task estimate the surface normal of a 3D object. There are two main approach to solve the task. Geometry based approach estimate the surface normal of an object based on the geometry principle. Point cloud is a common surface geometry information which samples the surface point position in 3D space. Another approach is photometric stereo based approach, which utilizes a set of images under different illumination conditions. In this chapter, geometry and photometric stereo based approach are introduced separately, then a deep learning based approach using geometry information is proposed. In the end, as the main work of the thesis, another deep learning method is proposed based on both geometry and photometric stereo information.

1 Geometry based normal estimation

1.1 Approach

The geometry based normal estimation uses point cloud of the object surface as input. Given a structured point cloud $V^{W\times H\times 3}$ to estimate the normal map $N^{W\times H\times 3}$, where each normal $\mathbf{n}^{3\times 1}$ at point $\mathbf{v}^{3\times 1} \in N$ is a unit vector with its direction point outward of the surface and perpendicular to the tangent plane of the surface at point $\mathbf{p}^{3\times 1}$.

The idea behind the neighbor based method is to fit a plane Π using the k neighbors $\mathbf{p}_1, ..., \mathbf{p}_k \in \mathbb{R}^3$ of the point $\mathbf{p}$, calculate the normal $\tilde{\mathbf{n}}$ of the plane.

It is under the assumption that the point and its neighbors are located in the same plane. This is usually not hold for the most of the surface, but if k in a suitable scale and point cloud is dense enough, it is enough to get an accurate and sharp result. As shown in Figure **??**.

Specifically, it is not necessary to find the exact plane equation to solve the normal. Instead, the normal can be derived based on an equation system. The normal $\tilde{\mathbf{n}}$ of plane Π is perpendicular to all the vector on the plane Π, we can construct k vectors on plane Π use k neighbors $\mathbf{p}_1, ..., \mathbf{p}_k \in \mathbb{R}^3$ of point $\mathbf{p}$. For the simplicity, we can choose $\mathbf{p}_1$ as the base point, then $k-1$ vectors can be construct as follows

$$\mathbf{v}_i = \mathbf{p}_1 - \mathbf{p}_i \quad \text{for } i = 2, ..., k \tag{1}$$

and each of them satisfied $\mathbf{v}_i \cdot \tilde{\mathbf{n}} = 0$. Then, the equation system can be constructed as

$$A \cdot \mathbf{n} = 0 \tag{2}$$

where $A \in \mathbb{R}^{(k-1)\times 3}$ is the vector matrix vertically stacked by $\mathbf{v}_1, ..., \mathbf{v}_{k-1}$. In order to avoid trivial solution, one more constraint should be added

$$\|\mathbf{n}_{3\times 1}\|_2^2 = 1$$

which also let the normal to be a unit vector.

To calculate a valid normal, at least 3 points are required to construct, i.e. $k >= 3$. For the sake of robust, more points can be used to reduce the measuring

error. For the case $k > 3$, since the surface vectors are actually not in the same plane, the equation system is likely over-determined . Then the equation system mentioned above can be converted to follow optimization problem

$$\begin{aligned} \min \quad & \|A\mathbf{n}\|^2 \\ \text{s.t.} \quad & \|\mathbf{n}\|^2 = 1 \end{aligned} \tag{3}$$

which can be solved by singular value decomposition(SVD). Let the decomposition of

$$A = U\Sigma V^T$$

The solution i.e. normal is the last column of V.

At last, all the normals should point ot view point $\mathbf{s}$, thus the direction of a normal should be inverted if

$$\mathbf{n} \cdot (\mathbf{p} - \mathbf{s}) > 0 \tag{4}$$

Repeat the procedure for all the points in the point cloud to get the entire normal map.

1.2 Evaluation

The neighbor based method can predict the normal map in a good way when the given point cloud is dense, as shown in Figure **??**. It can successfully predict the smooth surface of the dragon object, especially the flakes and the tails of the dragon.

However, it failed in the areas such as hindleg, horn and the mouth, which consists mainly by sharp edges. This is because the neighbors points in these area do not hold the assumption of coplanarity, the normals of these neighbors can be very different. The neighbor based method is depended on a well-chosen

Figure 1: Normal map of a dragon object predicted by neighbor based method. k=2, angle error=5 **Left**: ground-truth normal map **Middle**: predicted normal map, **Right**: Error map

parameter k. Figure **??** shows the evaluation on different k values. When $k = 1$, the average error of the whole image is the lowest one, most of the normals are close to the ground-truth but the outline edges, which are the areas that surface

normal changed extremely sever. For the case $k = 2$, the sharp edges are more smooth and cause more error, like the eyes area of the dragon. Compare to the first case, the outline edge error goes better. Most of the edge errors are reduced when $k = 2$, since more neighbor points join the evaluation and it reduces the effect of outliers. However, for the area of horn outline, hindleg outline, the error goes worse. In this case, most of the neighbors of these points are outliers and thus failed this approach. $k = 3$ and $k = 4$ further increase the angle errors in $k = 2$.

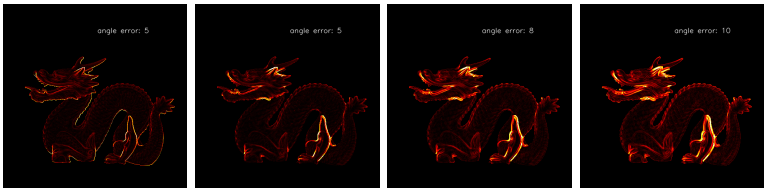

Figure 2: Error map of neighbor based method with different k values. From left to right, $k = 1, 2, 3, 4$ separately.

The performance of neighbor based method is good enough for a well chosen k, and it can be further improved by simply use different k for different area prediction. However, for the case of noised point cloud as input, the approach will broken, since the noise will failed the neighbor selection and also reduce the number of possible neighbors of each point for a fixed k.

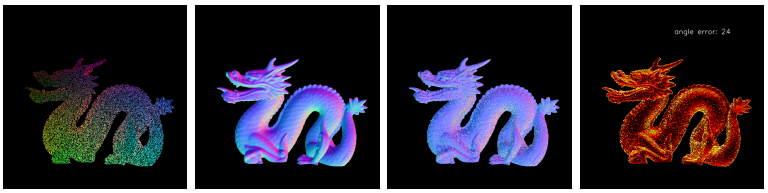

Figure 3: Evaluation of neighbor based method on a noised dragon model

2 Gated Convolution neural network for surface normal estimation

Recently, deep learning based method achieved a great succeed for image processing. (**yolov3**, **efficientDet**) These network architectures use a batch of RGB / Grayscale images as input and are employed for classification problems. Usually, the images are convoluted with a convolution layer and downsampling with pooling layers. The outputs of the network consist of a single value to represent the index of the corresponding class (**efficientDet**) or with a set of values to represent the position of bounding boxes.(**yolov3**). However, in many other vision tasks, like normal map inference, the output is demanded as the

same shape as the input. Instead of predicting one or several classes for the whole input matrix, the class for each pixel requires for prediction. In this case, the traditional network architecture is not suitable anymore.

It is worth to noticed that, the output of normal inference CNN model is not one or several labels but an entire image or normal map with same size. Recently,**unet** proposed an architecture called UNet for biomedical image segmentations. The architecture is shown in Figure **??**.The first half network is a usual classification convolutional network, the second half replace the pooling layers and traditional fc layers in the traditional CNNs to upsampling layers, thus in the end of the second half, the output is able to back to the input size. The proposed network can successfully assigned each pixel a class for segmentation tasks. Under this symmetric network, an input image is downsampled 4 times and upsampled 4 times. Output image has exactly the same size as input image. The downsampling and upsampling both have large number of feature channels, which guarantee the network propagates the information to higher resolution layers.

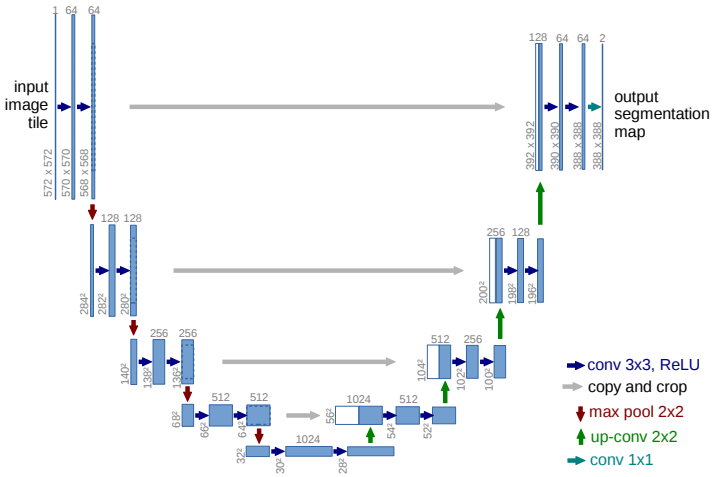

Figure 4: The structure of UNet. **unet**

The UNet is based on standard convolution layers to construct the network. This is reasonable for image processing task with full-dense input, since no missing pixels exist. However, for the input of noised point cloud, the valid and invalid pixels will be treated equally if we still perform standard convolution layers. Since the aim of the network is not learning the pattern of noise, but the noise with eternally changing patterns will confuse the network, and it fails the normal inference, a mask is required to distinguish two kinds of pixels.

pncnn0 use binary mask to indicate valid pixels, and further use normalized convolution to predict the output. The normalized convolution is shown as follows

$$O(x,y) = \begin{cases} \dfrac{\Sigma_i^k \Sigma_j^k W(i,j) \cdot I(x-i,y-j) \cdot M(x-i,y-j)}{\Sigma_i^k \Sigma_j^k W(i,j) \cdot M(x-i,y-j)}, & \text{if } \sum_i^k \sum_j^k M(i,j) > 0 \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

where k is the kernel size, (x, y) is the position in input, (i, j) is the displacement in kenel, M is the corresponding mask. A binary mask uses 1 to indicate valid pixels and 0 otherwise. $\odot$ denotes element-wise multiplication.

Normalized convolution layer added the weight to the mask. However, a initialization for the mask is still required, and the propagation of the mask remain a tricky task.

2.1 Gated Convolution

gated˙activation proposed a gated activation unit to model more complex interactions comparing to standard CNN layers, which mainly inspired by the multiplicative units exist in Long Short-Term Memory proposed by **lstm** and Rated Recurrent Unit (GRU) proposed by **gru**. **gconv** employed the same gated unit solving for the free-form image inpainting task. The proposed network use 3 channel RGB images as input and estimate the missing pixels.

The structure is shown in Figure 5. Instead of using a mask as input to indicate valid pixels, it employs a standard convolution layers to learn this mask directly from data. The valid pixels are then activated by a Sigmoid function. Then it imply element-wise multiplication with the feature map. Formally, the gated convolution is described as follows, the layer with input size (N, C_{in}, H, W) and output size $(N, C_{out}, H_{out}, W_{out})$:

$$o(N_i, C_{o_j}) = \sigma(\sum_{k=0}^{C_{in}-1} w_g(C_{o_j}, k) \star i(N_i, k) + b_g(C_{o_j})) * \phi(\sum_{k=0}^{C_{in}-1} w_f(C_{o_j}, k) \star i(N_i, k) + b_f(C_{o_j})) \tag{6}$$

where ϕ is LeakyReLU function, σ is sigmoid function, thus the output values are in range $[0, 1]$, $\star$ is the valid 2D cross-correlation operator, N is batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels, $w(C_{o_j}, k)$ denotes the weight of j-th output channel corresponding k-th input channel, $i(N_i, k)$denotes the input of i-th batch corresponding k-th input channel, $b(C_{o_j})$ denotes the bias of j-th output channel.

2.2 Architecture

Based on the implementation mentioned above, the architecture roughly follows on UNet proposed by **unet**, as shown in Figure 6.

Instead of using pooling layers for down/up samplings, gated convolution layer with stride $(1, 1)$ is used. The gated convolution using Sigmoid function for gating layer and LeakyReLU function for feature layer. All the layers are gated convolution layer with the exception of last two layers, which instead uses

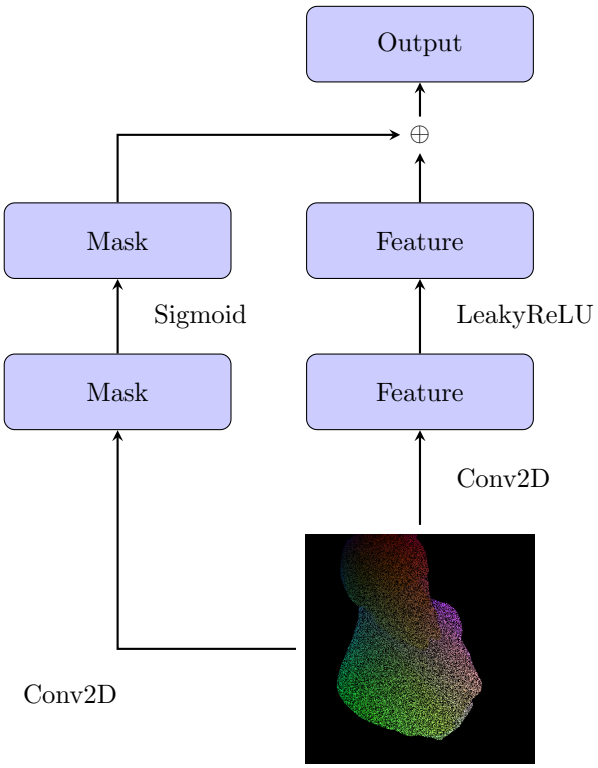

Figure 5: Gated Convolution Layer, where $\oplus$ denotes element-wise multiplication.

standard convolution layer to scale the output in range $[-1, 1]$. Other than the last two layers which use 1×1 kernels, all the gated convolution layer use 3×3 kernels with 1×1 padding.

The input is 3D vertex with size $512 \times 512 \times 3$, and output is $512 \times 512 \times 3$ normal map, which has the same resolution. There are 3 times downsamplings, each scale with 3 gated convolution layers, the third layer has stride 2. The upsampling part interpolate the feature maps 3 times with 1 gated convolution layer in each scale.

It keeps the skip connection in UNet to remain the fine detail features.

2.3 Loss Function

L1 Loss L1 loss, also known as absolute error loss, which calculates the absolute difference between the prediction and the ground truth. It leads to the median of the observations.

$$L_1(\tilde{y} - y) = |\tilde{y} - y|$$

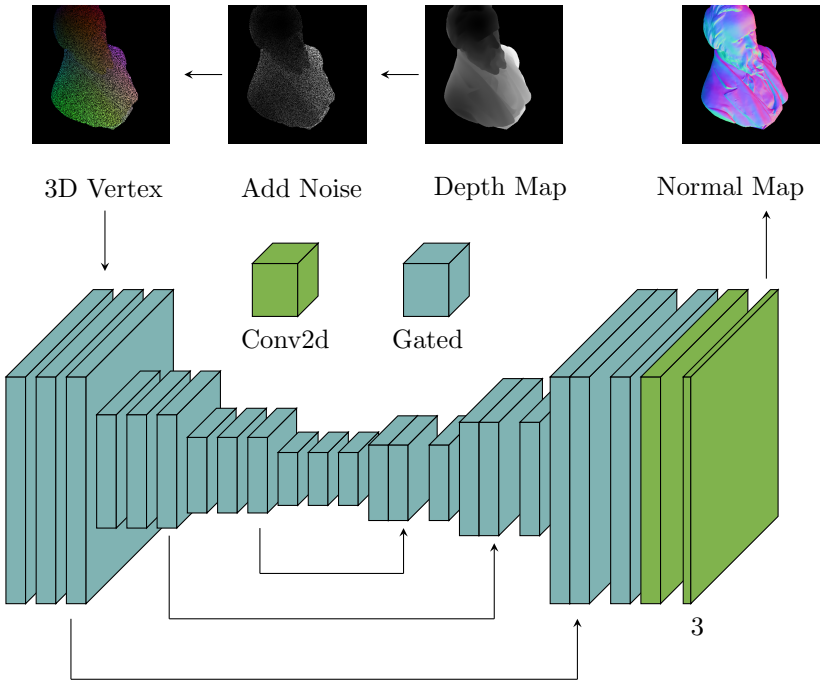

Figure 6: Basic Normal Neural Network model based on Gated Convolution layer and UNet architecture.

L2 Loss The standard loss function for optimization in regression problems is the L2 loss, also known as squared error loss, which minimize the squared difference between a prediction and the actual value. It leads to the mean of the observations.

$$L_2(\tilde{y} - y) = \|\tilde{y} - y\|_2^2$$

Masked L2 Loss with penalty for outliers(mask-l2) The background pixels of the input data are not considered in the normal inference task, they are saved as black pixels in the input data. These pixels should not considered in the loss function, i.e. invalid pixels. Therefore, a valid mask is required to distinguish the background and the main object. Specifically, using a matrix with the same width and height as the output, for each pixel, 0 is invalid, 1 is valid. Furthermore, depends on the specific task, the output should be constraint in a range. For normal output, the range is $[-1, 1]$. Thus for the outliers out of this range, a outlier mask can be applied to give them a penalty.

$$\begin{aligned} l(x, y) &= L = \{l_1, ..., l_N\}^T \\ l_{n \in N} &= \|mask_{obj} \odot mask_{ol} \odot (\tilde{y}_n - y_n)\|_2^2 + \|mask_{obj} \odot mask_{nol} \odot (\tilde{y}_n - y_n)\|_2^2 \end{aligned} \tag{7}$$

where x is input, y is target, N is the batch size.$mask_{obj}$ is the mask of the object, i.e. 1 means it is an pixel on the object, 0 is an pixel on the background. $mask_{ol}$ is the mask for the outliers, i.e. 1 means outliers, 0 means non outlier, $mask_{nol}$ is exactly the inverse of $mask_{ol}$. p is the penalty of the outliers, it is set as 1.4.

Reversed Huber Loss **berhu-loss** proposed Reversed Huber loss to combine both L1 and L2 loss. L1 loss is for small values whereas L2 for large values

$$\mathcal{B}(y) = \begin{cases} |y| & |y| \leq c \\ \dfrac{y^2 + c^2}{2d} & |x| > c \end{cases} \tag{8}$$

where $c = 0.2 \max(|\tilde{y} - y|)$.

3 Guided normal inference using GCNN

3.1 Intrinsic image decomposition

Intrinsic image model was introduced by **intrinsic-image**. It interprets that an image I can be decomposed as the element-wise product between the reflectance R of the object and the shading S produced by the interaction between light and objects.

$$I = R \odot S$$

It interprets the observed image into reflectance image and the shading image. As shown in Figure 7

Figure 7: Intrinsic image analysis of the bus object. From left to right, original image, reflectance image, shading image, light image, normal image

The equation can be further decomposed based on different surface models. If assume the object surfaces are Lambertian surfaces, i.e. the surface which reflect light in all directions, the shading image can be decomposed as the product of the radiance of incoming light L_0, the cosine of the angle of incidence, which is the dot product of the surface normal N and the light source direction L.

$$I = \rho \odot (L_0 \mathbf{L} \cdot \mathbf{N})$$

note that the surface normal N and light direction L are unit vectors thus they have only two degrees of freedom.

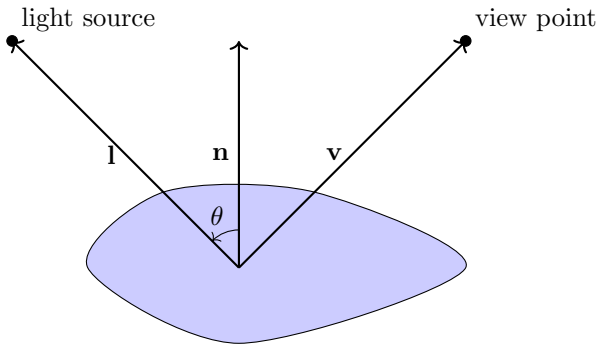

Figure 8: The surface normal, source light direction and the view point direction, where θ denotes the angle between light direction and the normal.

The equation can be further rearranged as follows

$$I = \mathbf{g} \cdot \mathbf{L} = (L_0 \rho \odot \mathbf{N}) \cdot (\mathbf{L})$$

The shape from shading method employed the equation mentioned above to predict the both surface albedo ρ and the normal $\mathbf{N}$ with knowning light source direction $\mathbf{L}$. More specifically, a set of k image for the same scene have been captured based on different light projections. Then, for each pixel (x, y) in the image, an equation system can be set up

$$\begin{pmatrix} L_1^T \\ L_2^T \\ \cdots \\ L_k^T \end{pmatrix} g(x,y) = \begin{pmatrix} I_1(x,y) \\ I_2(x,y) \\ \cdots \\ I_3(x,y) \end{pmatrix}$$

for the simplicity, L_i^T for $1 \leq i \leq k$ denotes the light direction at position (x, y) in the image k The equation can be solved based on least square methods. Since normal $N(x, y)$ is unit vector, thus we have

$$\|g(x,y)\|_2 = \|L_0 \rho(x,y) N(x,y)\|_2 = L_0 \rho(x,y)$$

Then the normal can be obtained as follow

$$N(x,y) = \frac{g(x,y)}{L_0 \rho(x,y)}$$

In another word, the surface normal including the albedo can be obtained directly based on images and light directions.

3.2 Light Map

The light map L can be derived from vertex map and the light source position. As shown in Figure 8, the incoming light direction is a vector point from light source to the surface point, therefor it can be calculated as follows

$$L(x,y) \quad = \frac{V(x,y) - (s_x, s_y)}{\|V(x,y) - (s_x, s_y)\|_2} \tag{9}$$

where (s_x, s_y) is the light source position and V is the vertices, both (s_x, s_y) and V are with respect to the camera space. The light direction map L is normalized since only the direction of the light is considered. The light map is a matrix corresponding the light direction of whole image where each pixel corresponding with each other, thus it has the same size as the vertex map.

It is important to note that due to the exist noise in the vertex map, the getting light map is only semi-dense, as shown in Figure 9.

In order to ease this issue, a light inpainting model has been trained based on the GCNN network with the identity architecture but the input and output, which takes the semi-dense light map as input and predict the fully dense light map as output.

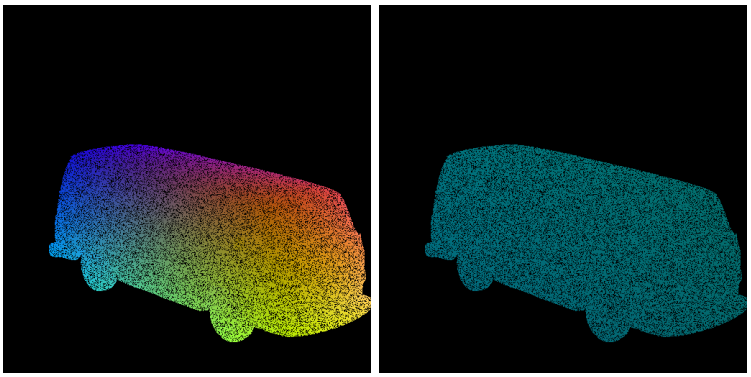

Figure 9: The light map calculated from vertex map and the light source

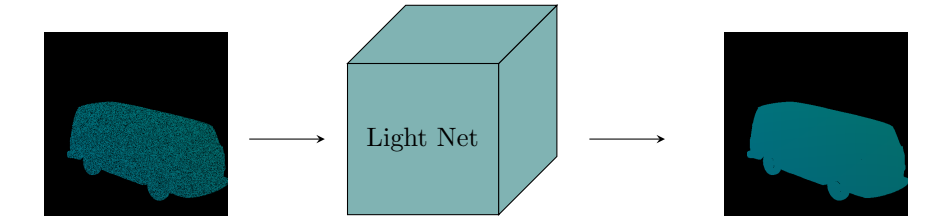

Figure 10: Light Net for light inpainting based on GCNN architecture.

3.3 VIL Net

Based on above implementations, we propose a light and image guided network called Vertex-Image-Light Network (VIL-Net). The structure is based on GCNN model which use downsampling two times as shown in Figure 11. As mentioned in the name, the network utilizes vertex map, light map and image map to accomplish the normal inference task.

The network can be consider in two parts. The first part extracts the feature maps. It deals with two kinds of input, the vertex map $X_1 \in \mathbb{R}^{w \times h \times 3}$, and the concatenation of light and image map $X_2 \in \mathbb{R}^{w \times h \times 4}$. The network extracts the geometry features X_v from vertex map X_1 (represented as a regression function v)

$$v : X_1 \to X_v$$

and the photometric features X_l from image and the light map X_2 (represented as a regression function l)

$$l : X_2 \to X_l$$

where the two encoders have the same network architecture based on the down-sampling part of GCNN model. After the feature extraction, 2 extra layers are added. First, a concatenate layer is added to fuse the vertex feature, and the

image and light map feature getting from encoder. Second, a fused feature map is predicted from all the feature maps base on a single gated convolution layer. (represented as a regression function m)

$$m : [X_v X_l] \rightarrow X_f$$

Then the network interpolates the feature maps X_f 3 times using interpolation and gated convolution layers to inference the normal map N. Meanwhile, the skip connections from UNet have been kept. Thus the feature maps $X_{df_1,df_2,...}$ of each down-sampling time in the regression model v are considered in the upsamling part.

$$n : X_f, X_{df_1,df_2,...} \rightarrow N$$

With the help of an extra image-light encoder, the network gained more information of the object surface, which is supposed to predict the surface normal more accurate. In this scenario, the output is still the surface normal, thus the training loss can be the same as GCNN model.

3.4 An3 Net

The first branch (shown above) takes a light map introduced in 3.2 as the input, the structure is the same as GCNN architecture except that the last two standard convolution layers, the skip connections are kept to connect the 3 down/up samplings. The second branch (shown below) takes image as the input, the structure is the same as the first branch other than the input image is 1 channel but not 3 channels. The third branch takes the 3D vertex map as the input. The structure is based on GCNN architecture. However, in order to merge the other two branches, the vertex branch equips 4 times fusions in the up sampling part. Specifically, the first fusion locates immediately after the last gconv layer of the last down sampling, the second fusion after the second gconv layer of first up sampling, the third fusion after the second gconv layer of second up sampling, the fourth fusion after the second gconv layer of the third up sampling. Each fusion follows by an interpolation layer, a gconv layer to reduce the channel back to 32, a skip connection concatenate layer and another gconv layer to reduce the channel back to 32. After the fourth fusion, a gconv layer used for channel reduction, 2 standard conv layer for output prediction.

3.5 Loss Function

For the case of normal output, the loss function is the same as Mask-L2 loss as introduced in 2.3. For the case of the product of albedo and normal, the loss function utilized a scaled Mask-L2 loss, which gives the range of inliers between $[0, 255]$.

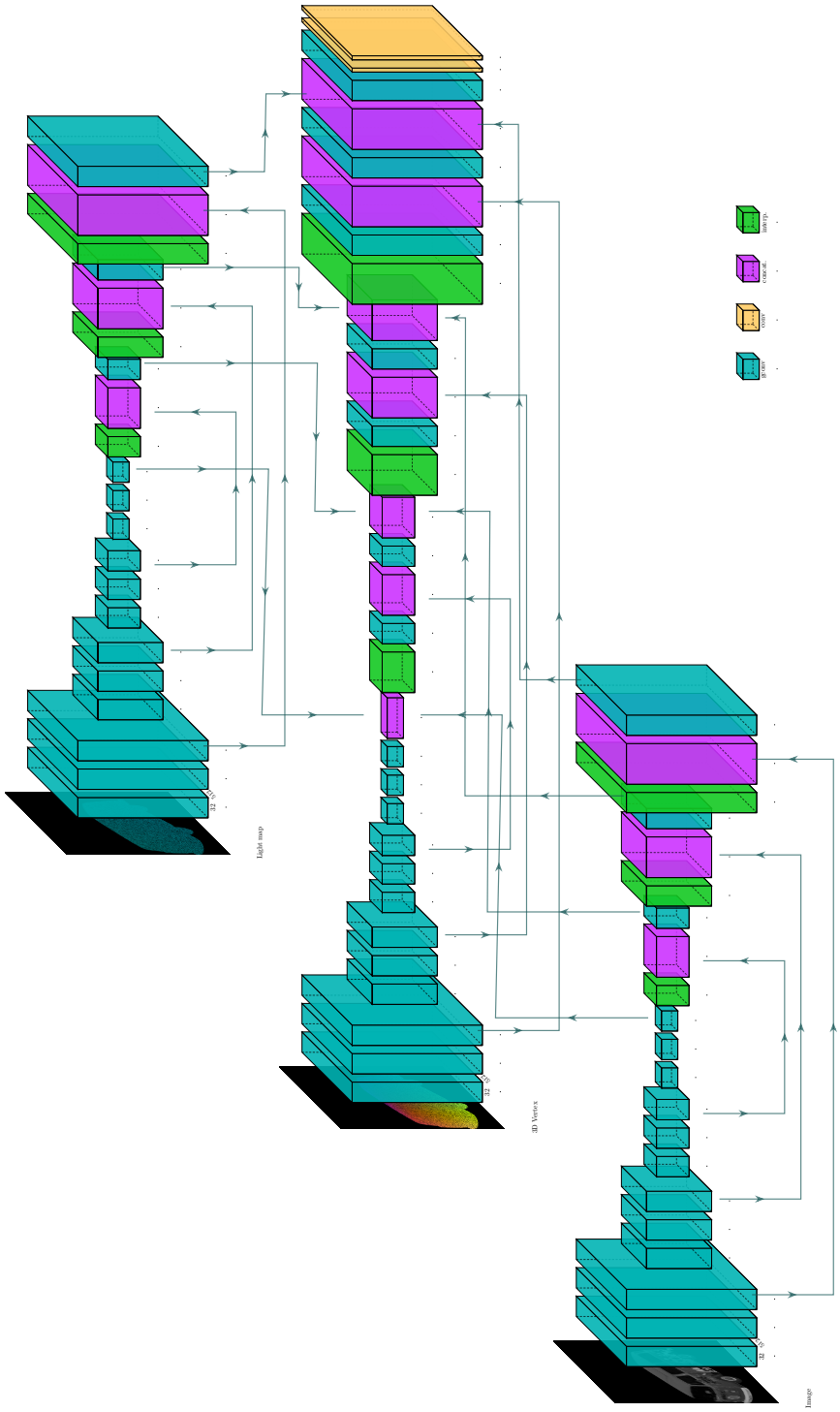

Figure 11: The architecture of TriGNet