Dissertation Type: Research


DEPARTMENT OF COMPUTER SCIENCE

# Deep Learning for Illumination Estimation from Stereo Images

Gavin Parker

---

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Engineering in the Faculty of Engineering.

---

Wednesday 9th May, 2018

‘

# Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of MEng in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Gavin Parker, Wednesday 9$^{th}$ May, 2018

# Executive Summary

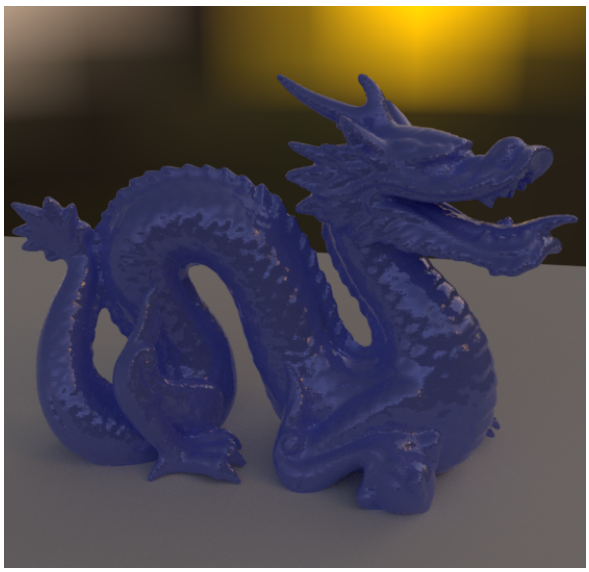


Figure 1: Ground Truth lighting

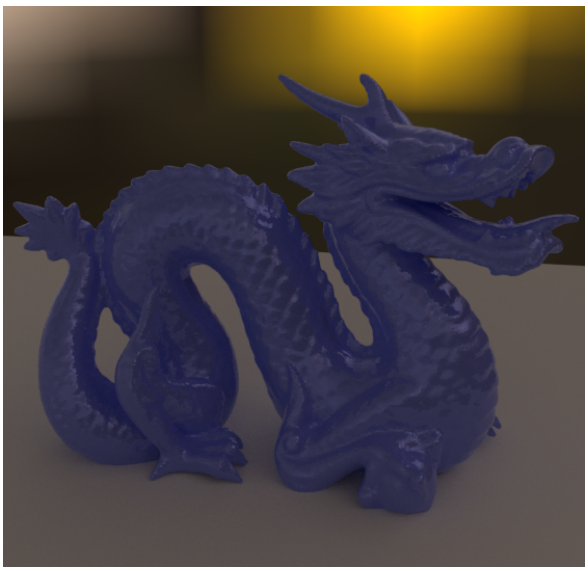


Figure 2: Our Predicted Lighting

In this paper we present a system for estimating the lighting in photographs and video streams of real scenes, that improves on previous work by eliminating the need for known geometry. We present a *Convolutional Neural Network* that exploits stereo views of an object to learn the environment lighting via reflectance. This environment lighting can then be used to realistically render and composite new objects.

Augmented Reality(AR) is a growing topic in computer graphics and computer vision, that involves superimposing virtual objects on images. The aim is to give the illusion that the object is part of the real scene. However, current AR solutions do little to estimate the lighting in the scene and so the added objects lack realism. The motion picture industry solves this problem by taking images of a mirror ball, which has known geometry and material properties. This is a slow process that must be performed beforehand, and is clearly inappropriate for live AR applications. The ability to estimate the lighting from arbitrary objects would make it possible to realistically augment existing or even live footage.

The appearance of an object is a combination of material, geometry and lighting. This makes estimation of any one of these factors from a single image a difficult task. Previous attempts at this task have relied on significant constraints; the material of the object must be very reflective, and the exact shape must be known beforehand. Furthermore current methods use a *Reflectance Mapping* step which comes with a significant performance penalty, and is vulnerable to inaccurate geometry estimates. We have produced a CNN that exploits learned stereo matching to infer surface properties and produce a usable lighting approximation. To do so we performed the following work:

- Replicated the work of Stamatios Georgoulis et al.[35] in Tensorflow by building CNNs that can interpolate sparse reflectance maps and predict environment map lighting from single objects with provided geometry. Achieved equivalent results on known surface geometry and confirmed the limitations of estimated geometry.

- Implemented a dataset generator that could render objects with realistic lighting and material properties for training our model. Produced a high-detail stereo image dataset of 55,000 image-lighting pairs.
- Augmented our large dataset of HDRI environment maps with Google Street View images, automatically tone-mapped by HDR-ExpandNet.
- Created a new Siamese architecture to encode geometry from stereo images and estimate lighting conditions. This achieved similar performance to previous work without the need for known surface geometry.
- Improved upon the Siamese architecture with a Cosine Similarity step, to achieve good performance with over 2X faster inference time.

Our network is able to achieve equivalent results to previous work that relies on known geometry, and outperforms those that use estimated surface normals in both accuracy and inference speed.

# Contents

# Supporting Technologies

- *Blender*[1] was used to evaluate the quality of produced HDRIs, and to create scenes for training data.
- The *BlueCrystal4* was used to train models across multiple GPUs.
- *Cycles*[2] renderer was used to produce photorealistic synthetic data.
- *Google Street View*[3] was used as a source of ground truth LDR environments.
- *HDR-ExpandNet*[31] was used to generate extra training panoramas.
- *HDRIHaven*[4] was used as a source of ground truth environment maps.
- *IKEA Dataset*[16] was used to train the final suite of models.
- *OpenCV*[5] libraries were used for image parsing and manipulation in experiments and final models.
- *Scikit-Image*[6] was used for SSIM measurements.
- The *Stanford Dragon*[7] test model was used for rendering.
- *Tensorflow*[8] was used to construct and train deep learning models.

[1] **blender.org**.
[2] **cycles-renderer.org**.
[3] **cloud.google.com/maps-platform/**.
[4] **hdrihaven.com**.
[5] **opencv.org**.
[6] **scikit-image.org/**.
[7] **graphics.stanford.edu/data/3Dscanrep/**.
[8] **tensorflow.org**.

# Notation and Acronyms

| | | |
|---|---|---|
| AI | : | : Artificial Intelligence |
| AR | : | Augmented Reality |
| BR(S)DF | : | Bidirectional Reflectance(Scattering) Distribution Function |
| CAD | : | Computer-Aided Design |
| CGI | : | Computer Generated Imagery |
| CNN | : | Convolutional Neural Network |
| CV | : | Computer Vision |
| GPU | : | Graphics Processing Unit |
| s HDR(I) | : | High Dynamic Range (Imagery) |
| LAB | : | Lightness*a*b |
| ML | : | Machine Learning |
| MSE | : | Mean Squared Error |
| RELU | : | Rectified Linear Unit |
| RGBD | : | Red Green Blue Depth |
| RM | : | Reflectance Map |
| SLAM | : | Simultaneous Localization and Matching |
| sRGB | : | standard Red Green Blue |
| SSIM | : | Structural Similarity |

# Chapter 1

# Context

Compositing digital characters and objects onto scenes realistically is a challenging task that has seen decades of interest from the motion picture industry. The primary aim is to fool the viewer into believing the digitally generated content was part of the original photograph or video. While it is sometimes possible to simply composite on a digital image, and tweak the appearance afterwards using image processing techniques, this is slow and often impractical. In an ideal situation, full details are known of the scene in which the original images were taken. In this case the problem is reduced to one of computer graphics, and the new content can be rendered within a digitized version of the target environment. Unfortunately this is almost never the case. Even expensive industry solutions that involve days of preparation and data gathering can only estimate a few scene parameters, leaving much of the work for digital artists. This results in great difficulty in scenarios where access to the scene is limited, such as editing existing footage. With Augmented Reality, the situation is even more challenging, as it must be possible to composite interactive digital assets on moving footage in real-time.
To render objects with realism it is vital to have a good understanding of the lighting within the scene, so that the reflectance of the object's materials can be calculated, and shadows can be cast. This data is arduous to capture manually, and in many cases it is infeasible. In our work we attempt to solve this problem, by using Deep Learning to estimate the lighting from two images of an object already within the target scene.

## 1.1 Computer Generated Imagery

In this section we will cover the physical properties of light and material, and how they are represented digitally. From this we can gain an understanding of how objects get their appearance, and how lighting could be derived from it. We then explain how computer graphics can be added to real imagery, and the problems this task presents.

### 1.1.1 Rendering

Traditional computer graphics involves creating virtual scenes, with geometry represented as a series of surfaces in 3D space. Each surface is given properties such as texture and material, and virtual lights are placed in the scene with given intensities and colors. Given the position and size of a virtual camera, images can be produced by calculating the resultant color at every point on the screen, produced by a combination of lighting and surface properties. This can be achieved through *Raytracing* which involves simulating the paths of all light rays that meet the camera within the scene, as shown in Figure 2.1 1.1.1. While physically accurate this is a slow process and impractical for many use cases. Often in computer graphics we aim to produce a responsive series of frames in *real-time*. In real-time rendering tasks a process called *Rasterisation* is used, where the geometry is culled so that only surfaces in view of the camera need to be considered. This, in combination with lighting and material approximations, makes it possible for convincing characters and objects to be rendered at 60 frames per second.

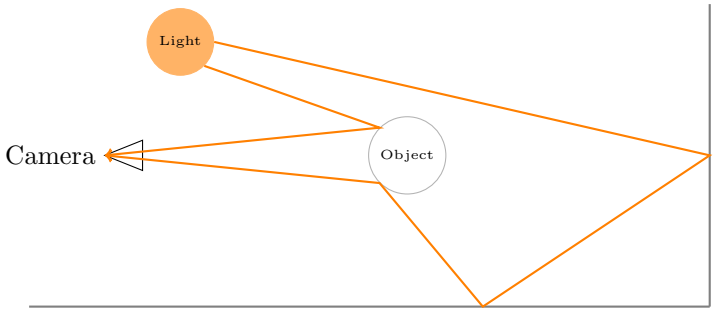



Figure 1.1: The direct illumination is a combination of the light source properties and the object material. The indirect ray is a combination of the light source properties and the material of all the surfaces it collides with

### 1.1.2 Light

The material of a surface defines how it reflects or absorbs incoming light, and as a result characterizes its colour under different lighting conditions. In the physical world no surface is perfectly even on a microscopic scale, making the angle and intensity of photon reflections at different wavelengths difficult to predict. Furthermore many photons are actually absorbed by the material depending on it's color properties and the lights wavelength. Visible light is made up of a vast supply of photons, which can even be considered as a wave with properties like wavelength or amplitude. If we consider light on a macroscopic scale it is far easier to predict and represent. We often use the concept of 'color' to refer to the wavelengths of light which an object reflects, where lighter objects reflect more wavelengths and darker objects reflect fewer, and absorb more photons. It is essential to note that the color something takes on is actually representative of the wavelengths of light that it reflects towards the light sensing device, be it an eye or a camera. Very smooth objects such as mirrors reflect most wavelengths, at very predictable angles, and so take on different colors depending on the viewpoint, whereas some materials tend to reflect similar wavelengths in all directions. These behaviors are captured in the *Rendering Equation*, formalized by Immel et al. [1],

$$L_0(x, w_0, lambda) = L_e(x, w_0, lambda) + \int_{ohm} f_r(x, w_i, w_o, lambda) L_i(x, w_i, lambda) dw_i$$

Which is roughly:

$$Reflectance = EmmitedRadiance + \int Material * IncomingRadiance + IncomingLight \cdot SurfaceNormal$$

which Computer Graphics attempts to approximate to produce the correct pixel colors. We can see from this equation, that if we have understanding of the material of a surface, and the surface normal, we can derive some knowledge of the incoming light from the resulting color.

### 1.1.3 Materials

In Computer Graphics, materials are often referred to as having *Diffuse* and *Specular* properties, which capture the aforementioned behavior. This is in fact an approximation that allows for the efficient rendering of surfaces, though does not capture the light behavior perfectly. Diffuse refers to the wavelength, or color, of light that is reflected off a surface equally in every direction. This can be thought of as the 'base colour', that is somewhat invariant to the viewpoint of the observer. A material that is entirely diffuse is often reffed to as *Lambertian*. Specularity refers to how reflective an object is, and is often used to approximate the light being reflected directly from a light source towards the observer. The *Phong Shading* model combines these two elements to roughly represent a variety of materials, under the assumption that reflective materials have a dominant component that can be model led almost as a mirror. For example an wooden panel will be very unreflective and may contain no specular component at all, as its color remains fairly consistent when observed from every direction. On the other hand a plastic ball will have a dominant color but also reflect a lot of the incoming light, which is modeled as a

combination of the incoming light direction and the surface angle. Modern Raytracing makes use of a far more accurate model called a *Bidirectional Reflectance Distribution Function*, which defines a probability distribution function. This function gives the probability of an incoming photon with angle of incidence i, reflecting with an angle of j. Over many photons this is able to accurately capture the reflective behavior of a surface, and is often used in combination with properties like *Albedo* and *Subsurface Scattering* to result in an almost photo-realistic surface. It is this complex reflection behavior that makes estimating the light on a surface difficult, and why most solutions reduce the problem of lighting estimation to lambertian surfaces.

### 1.1.4 Indirect Illumination

A product of the reflective nature of surfaces results in *indirect lighting*, an effect that is extremely difficult to capture in real-time rendering. The incoming light on a surface is made of photons arriving directly from light sources as well as photons being reflected from other surfaces, as demonstrated in 1.1.1. Often when rendering is expensive it is appropriate to approximate the lighting in the scene to contain only *direct* and *environment* lighting. In this approximation, all objects are lit by the environment light, which represents an approximate light in all directions at all points in the scene. This is very cheap to render, as provided there is no direct lighting, all points on a surface receive a constant amount of light. Direct lighting refers to lighting objects based on their proximity to a given light source. Unfortunately this approximation results in very unconvincing images as in reality the light at a point is made up of all the photons being reflected from that point. The *Phong* shading model ignores the effect of photons being reflected many times within a scene, and so misses much of the subtle detail. A common way of calculating indirect lighting is *Global Illumination*, which involves tracing the paths of many light rays from light sources as they are reflected within the scene. To save computation it is often performed in reverse, where rays are drawn from the camera and reflected a constant number of times. The illumination of the final reflection is then computed from the direct lighting, which is then passed between the reflected surfaces to achieve the total lighting at the original scene point. Global illumination is a very expensive procedure and inappropriate for real-time rendering, where other approximations can be used, such as *Environment Maps* or *Baked GI*. Environment Maps are spherical images that aim to capture all the incoming light at a point. They can be used as an efficient approximation for lighting at a single point, as shown in 1.2.

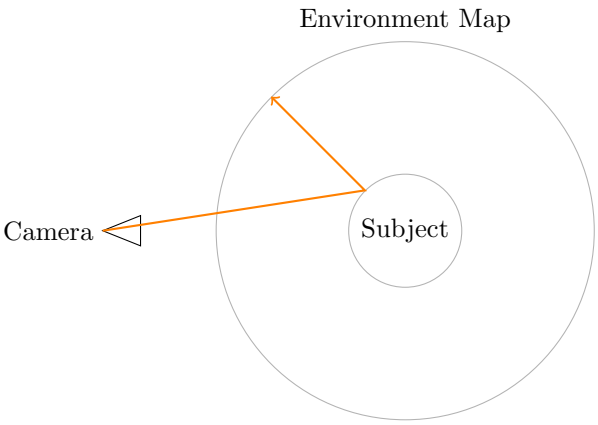



Figure 1.2: An environment map being used to render a sphere to the camera. This process is reversed with a perfectly reflective sphere to obtain the environment map

## 1.2 High Dynamic Range Imagery

*High Dynamic Range* (HDR) refers to image formats that capture a wide range of light intensities. Images are normally stored in RGB format, which captures Red, Green and Blue components of pixels in an 8 bit range of 0-255. This is a very limiting range when considering the brightness of pixels. When taking photographs, cameras only record a subset of the light entering the lens, based on the *Exposure Value*. As a result, photographers can take brighter or darker images by altering the camera exposure. To produce

HDR images, the photographer must take multiple photographs at different exposures and *composite* them together. The benefit of HDR in rendering is that it can be used as a light source, rather than simply a texture or reflection map. If combined with environment maps, entire scenes can be rendered without the need for approximate environment or directional lighting.

### 1.2.1 Digital Compositing

*Digital Compositing* is an extension of computer graphics and computer vision, where virtual objects are rendered on a real video stream to appear as if they are part of the original scene. This creates additional challenges as many of the parameters that are used in computer graphics are not available. While the material and geometry of the added objects is known, the geometry and lighting of the real scene must be estimated or manually recorded beforehand. The latter option tends to be used in films, where the layout of the scene and camera movements are known to the digital artist. In this case the geometry of the scene can be measured and recreated in 3D modeling or CAD software for a virtual approximation.
Similarly, the lighting at points in the scene can be captured using a light probe, often a reflective sphere, and taking composite photographs at different exposures. This process results in an Environment Map which captures the color and intensity of all the incoming light at a single point. With this data, additional characters can be rendered entirely in the virtual space, interacting with the approximated geometry and being lit by blended combinations of light probes across the scene. Provided that the virtual camera moves exactly as the real camera, the added object can be masked and overlaid onto the original video of the scene.
While accurate lighting and geometry are enough to render a single accurate composite image, more is required for convincing video. In computer graphics the appearance of a rendered object depends on the position and orientation of the camera. When using a real camera we need to know the position of the camera in the real environment. In video footage, the camera tends to move, while the digital addition should remain in place in the 3D environment. To achieve this, the relative movement of the camera must be known and compensated for. In the film industry, where the scene can be accessed beforehand, it is common to place high-contrast stationary markers within the scene. Once the footage has been collected, it is then possible to track the movement of these markers, and therefore the camera. If explicit markers are unavailable, it is possible to make use of features of the real environment as markers, such as high-contrast patterns on carpets or walls. This method is less robust, as often scenes may not contain enough stationary features to accurately track the camera. This is a significant difficulty in AR, with recent commercial solutions making use of the inertia sensors available in mobile devices.

## 1.3 Depth Estimation

To estimate the lighting in a scene, it is important to have some understanding of the geometry within it, to exploit reflectance, shadows and environment context. Our work makes use of the parallax effect, taking inspiration from computer vision techniques for depth estimation from stereo imagery. Here we briefly cover how depth and geometry can be calculated.

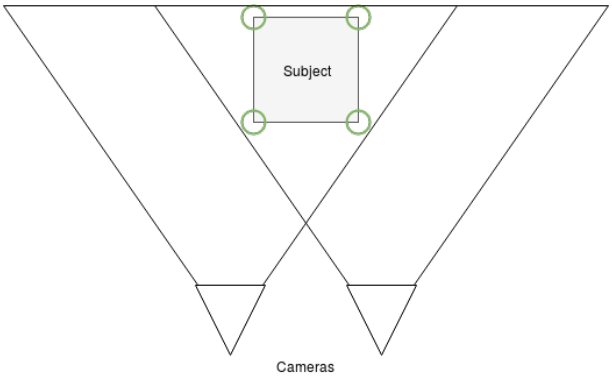



Figure 1.3: Example of Stereo Imagery. Points that appear further apart in the 2 camera places are further away from the camera due to parallax.

### 1.3.1 Depth Cameras

One of the most important components for scene understanding tasks is that of geometry. In the field of robotics especially, it is vital to be able to understand the placement of obstacles within the scene, while being tolerant of material and lighting changes. This problem is often paired with the task of motion or camera tracking, such that an actor can understand how they fit in 3D space. Furthermore, many of the techniques for lighting estimation previously mentioned require some understanding of the geometry of the scene. Geometry estimates tend to come in the form of depth and surface normals, which represent the distance from the camera and the surface normal direction (usually in camera space) for every pixel on the screen.

One method for collecting depth information is to use a depth camera, with specialized hardware to calculate depth for every pixel. An example of this would be the Microsoft Kinect camera, popular in computer vision tasks for its low cost. This calculates depth by projecting a pattern of infrared points onto the subject, and measuring how the distance between the points changes as objects move within the scene. For many cases this is impractical, as the depth resolution is limited, and it's reliance on IR makes it unusable outside. It is however an excellent tool for collecting ground truth depth data for other systems [21].

Another approach for capturing depth information is the use of stereo cameras, which record two separate images of the scene as demonstrated in 1.3. If the relationship between the two cameras is known, and the views of the cameras intersect, the epipolar constraint can be used to calculate the distance of every point from the camera. If the translation between the two cameras is known, this can be made easier by using *rectified images*, such that the two views share a common plane. In this case, the center point of each camera view projects onto a unique point in the other's view. If corresponding image points can be found, the epipolar constraint can be used to find the exact projected 3D point. Even if the camera properties are unknown, if enough matching image points can be found, at least the relative depth of the images can be calculated.

### 1.3.2 Stereo Matching

Finding matching points between two images is a difficult problem, as some points may be duplicated or even occluded. This is known as stereo matching, and often involves the comparison of image patches. One technique is to find the disparity of image patches, using a technique like *Normalized Cross Correlation*. This is very slow, and the accuracy of the results depends on the size of the image patch. Similar block-matching methods that compare pixel values like *Sum of Squared Differences* or *Sum of Absolute Differences* can be used but suffer the same accuracy penalty. There is a trade-off in most stereo matching algorithms between reliability and accuracy, as detecting the movement of single pixels is difficult, as on

flat surfaces the value may not change for large regions. Furthermore, small image patches are easily affected by changes in lighting and material. On the other hand large sizes can only calculate a lower-resolution depth image. More modern techniques use CNNs, such as [15], to extract features at different granularities, before attempting to find the disparity between them. This has shown to be incredibly effective, combining the effects of global methods and block matching. It is this technique that we aim to exploit in our work to get a better understanding of scene depth, and therefore normals and reflectance.

### 1.3.3 SLAM

Stereo matching, or Disparity mapping is used to find the depth of points within a scene, from a single view pair. However it is often the case that we need an understanding of the geometry of an entire scene, which requires combining the calculated depth from multiple camera transforms. If the stereo cameras can be tracked, then it is possible to combine the calculated depth information to build a point cloud, or 3D point representation of the scene. This can be achieved with *Simultaneous Localization and Mapping*, which combines depth estimation with camera tracking. *SLAM* involves finding salient features between successive camera views to track the camera movement. When the movement between two frames is found, the markers can then be used as 3D points in a point cloud, along with the depth information calculated from the stereo cameras. It is also possible, if difficult, to perform monocular SLAM, using a single camera. As with camera tracking, this usually involves a combination of marker-less tracking and inertia sensors. SLAM on mobile devices with standard cameras is still a tricky problem, as estimated markers tend to be sparse and point clouds computationally expensive.

## 1.4 Machine Learning

The lighting in a scene is determined by complex relationships between geometry and materials. Hand crafting a robust system that is able to take into account different image features, without relying on limiting assumptions is difficult. We approach this problem using Deep Learning, where we train a model to learn the optimal features to capture the input and reproduce a lighting output.

### 1.4.1 Deep Learning

If we wish to predict a more complicated lighting model we must be able to take into account many more features than what we can feasibly define by hand. In this case we can look to the field of AI, and especially *Convolutional Neural Networks*. With AI it is possible to train a model without explicit features, by using an approximation of neural dynamics in real brains. In broad terms, we programmatically create a series of connected neurons in a layered structure, with each neuron having a number of inputs and some number of outputs. The neurons have a very simple behavior - they multiply each input by some *weight* and provide the accumulated result, y, as the output:

$$y_k = \sum_{j=0}^{m} w_{kj} x_j$$

By training on known results we can adjust these weights so that the final neuron layer (which matches our desired output data shape) contains the desired output.
Convolutional Neural Networks are a recent development in AI that allows for reasonably fast AI models that take images as input. Rather than having a neuron for each color of each pixel, we can instead learn the weightings for a *kernel*. A kernel, in this context, is a matrix which can be passed across an image as a *convolution* to extract features,

$$\sum_{n_1=-s}^{s} \sum_{n_2=-s}^{s} f[n_1, n_2] \cdot g[x - n_1, y - n_2]$$

and often resembles a WxH matrix. Convolutions are used frequently in computer vision, as they are an efficient way of computing many popular image features such as *Harris Corners* or *Sobel Edges*. In a CNN we intend to learn the weights of many kernels to quickly extract features from the input image. Furthermore we are able to pass further convolutions across the results from previous convolutional layers to learn more complex features. For example the first layer could extract edges while a second

layer extracts the presence of 'X' shapes. CNNs make it possible to train deep and complex models on image data, resulting in robust models that often vastly outperform their traditional ML counterparts. The performance of CNNs is heavily impacted by the choice of metrics used to decide the weights and behavior for each neuron. When given inputs and weights a neuron may or not fire depending on an *activation function*. In biological neurons, this is a complex non-linearity that is difficult to efficiently capture in artificial neural networks. Instead we use approximations like the *Rectified Linear Unit*, or 'Relu',

$$f(x) = max(0, x)$$

to decide whether to fire after calculating the output value. Furthermore during training it is important to consider how to adjust the weights to converge on the optimum. For this we use a *Loss Function*, which computes an error value given the desired, or ground truth, value and our models prediction. The rate of change of this error is then used to iteratively adjust the weights of each neuron based on its influence.

### 1.4.2 Image Regression in Deep Learning

The prime application of CNNs has been in image classification, where the model must distinguish between a set number of classes. However it is possible to use CNNs for regression tasks that output other images using the concept of *deconvolutional* layers. These learn an interpolation matrix that increases the dimensionality of the input. Popular image regression architectures often involve a series of convolutional layers followed by a series of deconvolutional layers. The aim of the convolutional layers is to learn a deep representation of the content of the image, often removing the original spatial dimension,and producing a compressed representation of the data. Rather than 'shallow' features like edges or colors, these features ideally represent some understanding of the scene such as presence of some object or shape of the environment. The deconvolutional layers can then interpolate this data into a new image that will often share many common features with the original. For example in style transfer, the convolutional layers attempt to extract the content of the image while the deconvolutional layers interpolate the image features with weights representing some artistic style. In our work we use a CNN to extract learned image features from input images of a target object, before making use of deconvolutions to extract a lighting representation. The purpose of the intermediate layers is to learn to approximate the lighting without explicit constraints.

### 1.4.3 Training Data

The difficulty in the application of CNNs often lies in the *training* stage; while a deep enough network should be able to learn a mapping for any function, the difficulty in providing data to capture every case increases with the problem complexity. If a network is trained to recognize images of dogs, it will only be able to learn features from dog images that it has seen as input. As a result, if a new dog is provided that lacks some of the features of the training data, it may be miss-classified. Regression tasks in particular attempt to learn a complex function and so need a vast amount of training data to be able to generalize. This becomes problematic where the training data is hard to collect and classify. In the case of image classification, many photographs can be taken and marked with their intended class. With image regression this can become an extremely laborious task as the desired output needs to be created by hand. Often there are large datasets already available that can help speed up the process, especially if the datasets are labelled, or if features can be extracted by hand. However it is sometimes necessary to manually create a dataset to train a network to face a new task. Fortunately there are still some steps that can be taken to accelerate this process, such as data augmentation. This usually involves manipulating some training inputs in such a way as to introduce variances while maintaining the classification, and can greatly increase the amount of data available. Another approach is to create synthetic data, by rendering new images, such as the one in 1.4. It is important to note however that the synthetic data must be accurate enough such that a model trained on it can generalize to real world data. It is often sensible to mix real world and synthetic data when training a model for the best results. Lighting information is very arduous to capture manually, and so we use synthetic data for our learning task.

Figure 1.4: An example of syntethic data used for our work.

# Chapter 2

# Technical Background

## 2.1 Commercial Lighting Solutions

Figure 2.1: A mirror sphere used to capture Environment Maps

### 2.1.1 Light Capture

Lighting solutions in the motion picture industry tend to involve precalculating or capturing the lighting in a scene before adding virtual characters. The captured lighting can then be used for reflection mapping, where reflections are added to rendered objects. In the film 'Flight of the Navigator', Randal Kleiser first demonstrated the technique of reflection mapping in a feature film [5]. Here the reflectance was computed using a simple rectangular photograph, although more accurate implementations use spherical photographs of a gazing ball such as 2.1. This technique was enhanced with *HDR*. HDR refers to *High Dynamic Range*, which is an image format that captures the entire range of light intensity. Because of this, HDR images can actually be used as light sources for rendering. Later films used a semi-automated system for capturing *HDR* environment maps using a rotating fisheye-lens camera. Once placed, the camera was able to take shots at multiple angles and at a range of exposures, before compositing the shots together to produce a usable HDR lighting map. Often these HDR maps are not used in isolation, but along with footage of a proxy for the virtual addition, so filmmakers can qualitatively measure how the materials react with the set lighting. These techniques allow filmmakers to capture an approximation of lighting with limited camera equipment. It also makes it easier to create complex panning shots of scenes with real and virtual elements. Over time, camera equipment has become more advanced and it is faster to take high quality shots at multiple exposures. As a result, later films such as 'The Curious Case of Benjamin Button' are able to utilize multiple lighting probes to illuminate characters. By taking lighting samples across the scene, virtual characters can then be lit partly according to their position in the world by blending environment maps together. This creates a more convincing final render as the approximate indirect lighting from nearby objects becomes more apparent. The downside of these Hollywood techniques is that the lighting data must be captured beforehand for every environment used. This is a long, laborious and expensive process as the cameras required to take accurate shots at different exposures need to be extremely accurate to produce correct composites. It is also important to note that compositing images is difficult and also requires human input to match black levels, tones and camera properties that are hard to automate.

### 2.1.2 Light Estimation

Recent commercial AR platforms do contain some level of lighting estimation from input video data. Apple's ARKit is able to estimate the color and intensity of the ambient lighting in some scenes [6], by analyzing the pixels of the incoming video frame. This technique adds some realism to AR applications, as objects can be dimmed or brightened with the scene, so as not to stand out. However this is a very rough estimate of lighting, and contains no directional information or approximation of indirect illumination Google's ARCore has similar functionality, taking an average of the luminance values across the video to estimate the intensity [3]. It is possible to estimate the light direction from a face [37] however, by using the tracked face as a light probe. By using a face, the application is able to restrict the scenario to a range of likely geometries and materials that make up human faces. The platform is able to produce not only the primary light direction and intensity, but also an approximation of the environment lighting as spherical harmonics. While incredibly useful for some application such as the retouching of portrait photographs, the technique is not robust to all scenes as it relies on the approximation of some known geometry.

Upcoming AR applications are experimenting with new methods of integrating the scene lighting with virtual augmentations. Magic Leap is making use of tailored hardware called a 'Light Field' camera [38], which involves the use of an array of microlenses to capture additional information from the scene. Each microlens is able to capture information from multiple camera points, so that extra depth and lighting information can be extracted from the frame. This is a significant improvement on IR depth cameras such as the Microsoft Kinect, which can't be used outside. An example consumer Light-Field Camera is shown in 2.2 By extracting depth, and therefore surface geometry information, it is far easier to reason about the reflectance of objects within the scene and estimate the lighting. However, light-field cameras are extremely expensive and impractical for mobile AR. Most mobile devices contain normal high quality cameras as well as accurate inertia sensors, which can be used in combination with stereo matching for a similar depth estimation model.

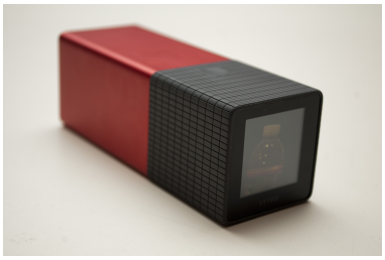


Figure 2.2: A commercial light-field camera from *Lytro*

## 2.2 Lighting Estimation Research

### 2.2.1 Direct Lighting

Early techniques for lighting estimation were able to find the direction of a single primary light source, under significant constraints. In [17], Peter Nillius and Jan-Olof Eklundh were able to estimate a single light direction, provided the input image contained an appropriate lambertian surface. By making use of an 'occluding contour' it is possible to infer some detail of the geometry, as the surface of the visible edge of a contour that occludes itself will always be perpendicular to the view direction of the observer, shown in 2.2.1. Using the lambertian lighting model, along with the edge intensity it is possible to make a prediction using a Bayesian network for the incoming light direction. This is a simple method that exploits inherent properties of real geometry, but has too many prerequisites to be practical. While it captures direction, it also ignores color, intensity and any indirect lighting. Similarly, Varma demonstrates a technique to estimate the lighting direction from textured images [25]. If the surface is constrained to be a Gaussian surface with uniform *Albedo* (diffuse color), the incident light direction can be estimated by the shadows cast by the height changes in the material. While also very constrained, it is

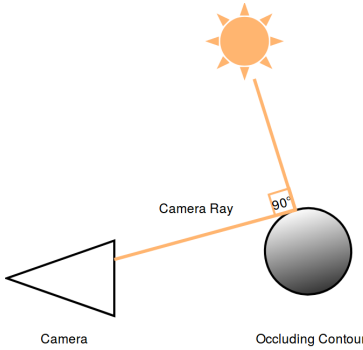



Figure 2.3: A self occluding contour. The surface normal is orthogonal to the camera ray, and so the brightness of resulting pixel compared to the rest of the surface gives a good indication as to the illumination arriving from that direction.

shown that even strict models can generalize to produce useful results in this field.
There has been some work in inverse lighting specifically for the rendering of AR objects, notably that of [12]. In this work the user must simply provide a ping-pong ball, representative of a Lambertian sphere. Then the least-squares algorithm is performed on the lighting equation, using features of the sphere as constraints. This results in an approximation of illumination as a series of point lights and spherical harmonics which can be used for rendering. The downside however is that the process of presenting the system with a known ball can take upwards of 20 seconds, and the resultant lighting is suitable for only primarily diffuse objects. This does however demonstrate how powerful even basic knowledge of the geometry and material of the scene can be for deriving the lighting.

### 2.2.2 Outdoor Lighting

In [2], Lalonde et al. are able to estimate a more complete set of parameters for environmental lighting from single outdoor photographs. By segmenting the image into the sky and surfaces, and extracting features like shadows, the authors are able to reliably predict the sun's position. While robust for outdoor applications, this approach does not consider light intensity or color. Furthermore, the reliance on outdoor scenes restricts its usage in AR. Scott Wehrwein et al. demonstrate a technique for sun direction and shadow detection from multiple input images [8]. They use illumination rations to extract shadows, after constructing a sparse 3D representation of the scene from stereo matching of multiple images. By exploiting shadow boundaries on the derived surface normals, it is then possible to determine the light direction. This work shows how multiple views can be used to infer knowledge of the geometry of the scene, to make the problem of lighting estimation more constrained. However, like previous work it assumes a perfectly Lambertian surface and clear outside conditions, restricting the possible lighting conditions significantly.

### 2.2.3 Shadows

It is clear that estimating a full representation of the illumination within a scene is very difficult given the number of unknowns. However predicting only the direct light sources is still very valuable. This makes up the majority of the light within a scene, and can be combined with image color values to give an approximation of the indirect lighting. Shadows are a good indicator of the direction and intensity of direct lighting. They not only give some indication of direction based on shape, but also the intensity through the contrast of color values between areas that are shaded and unshaded. Takahiro et al. demonstrate techniques for recovering illumination from shadows in [18]. This work compares two methods, *Spherical Harmonics* and *Haar Wavelets* for estimating the illumination from cast shadows on known geometry. Spherical Harmonics are functions defined over a whole sphere, and are useful for defining the light reflectance at a point, as in Global Illumination. The work demonstrates how this method fails to capture higher frequency illumination values, and presents *Haar Wavelets* as an alternative. This method is able to approximate functions as a series of linear combinations of Haar functions, and is able to more efficiently capture a higher range of light frequencies at a point.

### 2.2.4 Experimental Methods

An interesting approach from Microsoft Research [9] attempts to reconstruct the shape and reflectance of an object from a video with known motion and camera pose. This is achieved by decoupling the estimation of geometry from the estimation of illumination, and refining initial estimates over a series of video frames. While the lighting is not known in the scene, the camera pose and motion must be known and so this work is only applicable under carefully controlled environments. However, it effectively demonstrates how multiple views of an object can be used to refine predictions of scene understanding. It also uses reasonable assumptions of static lighting and material.
An emerging trend in lighting estimation research is the use of 'light fields' as a way of representing the light on an entire scene. A light field represents the radiance of light flowing in each direction in each point in space, resulting in a 5D function. Capturing and storing light field data is difficult as for a given area, light must be sampled in all directions at all points, which is impractical using traditional cameras. The use of microlens light-field cameras, which sample incoming light rays at a range of angles, has made capturing light-fields much less costly, resulting in datasets becoming available such as that of [10]. Light fields require immense quantities of storage capacity and contain far more light information than is required in the field of AR, making their use unpractical. Instead most work has been centered

around the estimation of light at a single point, either at the camera view or in the scene, that can be seen as a sparse sampling from the light-field. A compressed example is shown in 2.4

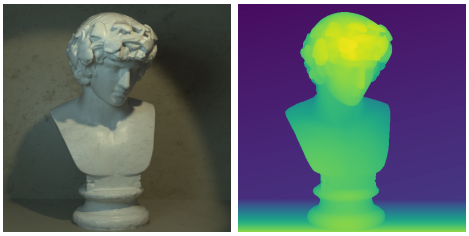


Figure 2.4: An example of the extra data captured by light-field imaging.

## 2.3 Deep Learning for scene understanding

### 2.3.1 Geometry Estimation

To calculate the lighting of a scene, some assumptions or approximations must be made about the material or geometry to constrain the problem. It is therefore important to consider previous techniques to extract these features from images, especially geometry estimation which has significant applications in the field of robotics. In 2015, David Eigen et al. demonstrated a CNN approach to estimate depth, surface normals and semantic labels from single input images [19]. Here they use a 'multi-scale' architecture, where outputs from early features passed as input to later deconvolutional layers, as well as subsequent convolutional layers. This technique can often lead to faster inference as important features are 'short-circuited' closer to the output, reducing the need for large deep layers at inference time. The results of this work are extremely promising, and suggest that a single RGB image may be enough to infer a complex understanding of the scene. If it is possible to understand variations in the illumination and material in a scene to find geometry then it follows that it is possible to understand geometry and material to find illumination. However, the network shown is very deep and trained primarily on indoor scenes. Its use in the field of AR is limited, as the hardware is much more limited in terms of GPU resources. Furthermore, while the network is able to predict depth and surface normals on a large scale, successfully finding walls and floors, it struggles with more complex geometry. Some progress has recently been made to improve performance in geometry estimation by incorporating more traditional SLAM techniques. In their work, Luo et al. show increased performance in stereo matching by including a dot product step [15]. In stereo matching tasks, two images are used to compute a disparity map, which corresponds to a relative measure of depth due to the *parallax effect*. This work uses a Siamese network, where the same features are extracted from each of the two images. As a result, using a dot product computes a similarity between the two sets of feature outputs. This is equivalent to the *Cosine Similarity* of the image vectors, provided they are normalized. This greatly reduces the number of layers that would otherwise be required to map the input image to an output disparity map, while still producing accurate results.

### 2.3.2 Synthetic Datasets

With it's usefulness in robotics, geometry estimation has been a primary focus of scene understanding tasks. Geometry datasets are easier to source, with depth cameras becoming cheap and easily available. Other important information like material, texture, semantic class, Albedo an many others are almost impossible to capture efficiently from real scenes using hardware. Instead these parameters must be hand labeled, which is expensive, slow and leads to noisy results. The recent availability of high quality synthetic datasets such as SUN-RGBD [26] and the very recent Scenenet-RGBD [32] however, have provided more opportunities for learning scene parameters that are more difficult to retrieve from real scenes. Some of this detailed data is shown in 2.5 For example, Rajpura et al. demonstrate how semantic segmentation networks can be improved by augmenting real data with the readily available synthetic data [28]. Some networks must be trained entirely on synthetic data, such as that of Narihira et al. which utilizes the synthetic MPI Sintel dataset to predict shading and Albedo, which can't be captured by hand. Unfortunately, many synthetic datasets do not generalize well to real data, and it remains to be seen whether virtual scenes with the level of detail needed for lighting estimation can be rendered fast enough to be practical for research, although recent datasets appear promising.

Figure 2.5: Example synthetic Scenenet RGB-D data, containing RGB, depth, semantic instance, semantic class and optical flow.

### 2.3.3 Deep Learning for Illumination Estimation

Most of the previous work has had to rely on significant assumptions for the scene. This constrains the lighting to fit known phenomena which can be exploited to estimate the illumination. It is clear that due to the unconstrained nature of the problem, that some assumptions about geometry, material and lighting must be made. Rather than making the required assumptions to fit a model, another approach is building a model to find and exploit patterns from real data using machine learning. Given enough scene features and data it is possible to learn approximations and assumptions to produce a more robust model. In [13], Yannick Hold-Geoffroy et al. use a CNN-based technique to estimate sky parameters from outdoor photographs to a high enough degree of accuracy that virtual objects can be re-rendered. In this work, the output illumination model is constrained to the Hosek-Wilkie model with parameters representing the sun position, ground Albedo and light wavelength. A dataset is then constructed by selecting small Low Dynamic Range (LDR) sections of real HDR scene panoramas, resulting in a large number of sample photographs, and HDR lighting ground-truths. This work demonstrates extraordinary performance, but is limited in output format by the lighting model which is constrained to outdoor images with clear skies. This ignores the indirect lighting from scene objects and indoor scenes, but demonstrates how CNNs with the right data can be used to exploit more subtle image features to predict lighting.
There have been advancements in the field towards estimating the lighting in indoor scenes, such as that by [14]. In this work, Gardner et al. use a very large dataset of indoor HDRI environment maps to train a CNN to predict the environment map at the cameras position. This environment map can then be warped to provide an approximate environment map for a given position. The resulting network is very accurate for predicting light direction and intensity. However the network is very deep, making it difficult to deploy on mobile devices. While the warping operator user is innovative, it fails to accurately capture the lighting at points in 3d space, and instead provides a rough approximation. Finally, the produced lighting requires a manual tuning step if it is to match the scene in hue, though this is the still very promising work in the field of lighting prediction.

## 2.4 Predicting Light Probes

The common approach for capturing the lighting is to take photographs of a mirror ball. By using a mirror ball, we have known geometry and predictable material properties. One approach to lighting estimation is to use an arbitrary object in place of a mirror ball, given some understanding of its shape. An example of this is 'Deep Reflectance Maps' [20] by Konstantinos Rematas et al. In this work it is demonstrated that better accuracy can be achieved by constraining the CNN input to a specific feature, in this case a sparse reflectance map, provided the geometry is known. A reflectance map is a mapping from surface normal direction to light reflectance in that direction. By taking the brightest pixel for every surface normal on an object, a sparse representation of it's reflectance can be built and fed to a CNN for interpolation. The approximate workflow is shown in 2.6 This work makes use of a large synthetic dataset but still results in good performance on real world data, allowing the researchers to then modify the shape of specular virtual objects while still maintaining a realistic surface appearance. An important feature of this work is that the researchers present results from reflectance maps produced from perfect surface normals, and from those produced by noisy estimated normals. It is demonstrated that the estimated normals are unsuitable for accurate reflectance mapping, and actually result in poorer predictions than a direct appearance-to-lighting network.

| Model | Geometry Prerequisite | Material Prerequisite | Direct Lighting | Indirect Lighting |
|---|---|---|---|---|
| Mirror Ball | sphere | mirrored | ✓ | ✓ |
| Nillius et al. | occluding contour | Lambertian | ✓ | X |
| Varma et al. | Isotropic Gaussian Surface | Constant Albedo | ✓ | X |
| Lalonde et al. | Outside with floor& sky | X | ✓ | X |
| Wehrwein et al. | Outside & clear | Lambertian | X | X |
| Georgoulis et al. | Static Known | Constant Albedo | ✓ | ✓ |
| Our Proposed Solution | Static unknown | Constant Albedo | ✓ | ✓ |

Figure 2.7: Summary of current methods

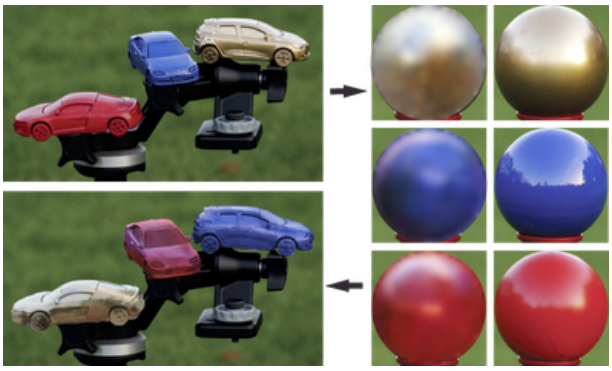


Figure 2.6: The *Deep Reflectance Maps* pipeline. Estimating the full reflectance map from the geometry and depth data allows for modification of the material.

This work was then extended for illumination estimation in 'What is Around the Camera', by using a precalculated sparse reflectance map in combination with a segmented background image. In this work, a full HDR environment map is predicted to a degree of accuracy that allows the superimposition of rendered geometry with fairly accurate lighting. Both of these works use a Convolutional-Deconvolutional approach to produce a new image as output. The limitations of this work however are due to required prior calculations rather than restrictive assumptions about the scene. While the network can handle objects of multiple materials, they must be segmented and separate sparse Reflectance Maps(RM) calculated. Furthermore the calculation of the sparse RM relies upon knowledge of the surface normals of the object being used. The researchers point to previous work on the usage of CNNs for surface normal and depth estimation but fail to demonstrate the accuracy of their work with these more noisy estimated surface normals. It is also apparent that there is significant data loss in the calculation of the sparse reflectance map, which makes significant assumptions about the object in question. The model ignoring the effects of shadows and how lighting changes based on both position and surface orientation.
These works demonstrate that if surface geometry is known, a network can be trained to accurately reproduce lighting conditions, using an object as a probe. However they point to 2 significant shortcomings that we attempt to address in this work. Firstly, the use of sparse reflectance maps in isolation ignores a lot of useful information on the object that could be used to estimate the lighting. As a result, a direct approach of predicting illumination from a single image view can often outperform cases where the reflectance map is too sparse or the material is very diffuse. Secondly, modern surface normal estimation techniques from single images are not accurate enough to constrain the problem of lighting estimation. We believe that the use of a network that exploits multiple views of an object, provided the scene parameters remain constant, will be able to more accurately predict the geometry through learned stereo matching. Furthermore, a network that simultaneously learns geometry and lighting will be able to exploit more scene features such as shadows, that would otherwise be unavailable in an explicit reflectance mapping. We summarize previous attempts and our proposal in 2.7

# Chapter 3

# Implementing Prior Art

## 3.1 Deep Reflectance Maps

It was important to reproduce the preceding work, to assess its applicability in the field of AR and create a fair benchmark for evaluating our new models and approach. We began with a reimplementation of *Deep Reflectance Maps*, a recent work that demonstrated promising results from a CNN based approach. This work was a necessary prerequisite to *What is around the Camera?*, which demonstrates how the reflectance map of an object can be extracted and used to predict it's lighting. The project was provided with a dataset for replication, but without any source code. Fortunately the architecture was well documented and so easily reproduced in a modern deep learning framework. For this work we used Tensorflow, a python deep learning framework that gave us the flexibility to add unique network features such as a reflectance mapping step, as well as export the model to different platforms.

The network in question is a Convolutional-Deconvolutional network trained to produce dense reflectance maps from single sparse reflectance maps as input. The training data was provided as the segmented 256x256 RGB image of an object, in this case the *Shapenet* car class [34], along with surface normals. The ground truth dense reflectance map was provided as a 64x64 spherical RGB image.
The network architecture is fairly simple, consisting of *Encode* blocks made up of convolutional layers, followed by *Batch Normalization*, *Max-Pooling* and the *RELU* activation function. In each block the convolutions are repeated a given number of iterations, with only the Max-Pooling step changing the dimensionality of the tensor, as shown in 3.1. These Encode blocks were also used in all subsequent models for feature extraction. These features were easy to implement in Tensorflow, leaving most of the time for training and optimization.

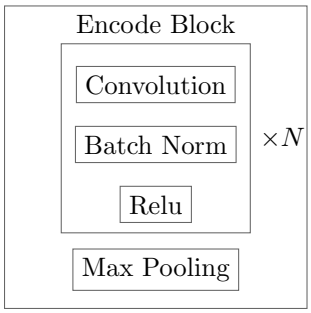



Figure 3.1: An *Encode* Block, used to construct our models. The first step can be repeated an arbitrary number of times, without modifying the size of the output.

Due to the quantity of training data provided (50,000 samples) it was necessary to pipeline the input data stream to make best use of the hardware resources available. We make use of a single GPU node on the University of Bristol supercomputer, *BlueCrystal4*. These nodes contain Nvidia P100 GPUs, making them ideal for training large neural networks. However I/O speed is still somewhat limited, resulting in a bottleneck in reading the data. By using a pipeline, it is possible to use the CPU for preprocessing operations such as data conversion and image resizing, leaving the GPU for the more intensive training process. This resulted in much faster training and allowed for a larger batch size than was originally used in the previous work.

### 3.1.1 Reflectance Mapping

It was necessary to first construct a system for extracting a sparse reflectance map from the inputs according to the approximation stated in the work. A reflectance map is a map from every surface normal to a luminance value. In essence it is a representation of the brightest reflection seen on an object for each surface angle. Of course, there are multiple points with the same surface normal but different reflectance levels due to shadowing and self-shading. To handle this we model the object as a sphere, on which each point has a unique surface normal, by using only the brightest point for each surface normal. By treating the object as a sphere we ignore shadows on the object and make the assumption that the light source is far away enough that the objects relative size is negligible. We extract the reflectances by building a sparse reflectance map,

$$R_i = max(n \cdot n_i \cdot L)$$

where the reflectance $R$, at angle $i$ on the sphere is the maximum illumination, $L$, of all pixels sharing $i$ in the original image. It is important to note that we only consider a half-sphere as surfaces facing away from the camera are naturally occluded. Also some information is lost by taking a maximum illumination, as we make the rough assumption that any darker pixels with the same surface orientation must be due to self-shading. This is the case as we are modeling the target in question as a point object, with all lights infinitely far away, which is the approximation being made with the use of Environment Maps.
The process for calculating the reflectance is mathematically intensive, but can be represented in a graph form as using matrix multiplication and boolean masks. This was initially implemented sequentially in

$$\begin{aligned}
&diffs \leftarrow sphere \times input^T \\
&reflectance \leftarrow mask(intensity, diffs > \cos(0.0872665)) \\
&indices \leftarrow argmax(reflectance) \\
&reflectance_map \leftarrow colour[indices]
\end{aligned}$$

the numpy mathematics package as a proof-of-concept but was too slow to operate on the number of training samples required. By migrating the implementation to use native tensorflow operations, the per-batch runtime was reduced from 6 seconds to 20ms, making it feasible to use during training, rather than as a preprocessing step.

### 3.1.2 Training

The model was trained for 50 epochs to produce results of similar quality to the original. Furthermore it was possible to compare the results to an end-to-end implementation which only used the original images as input. Surprisingly the results were very similar, with the end-to-end approach sometimes even outperforming the refectance mapped version. The reason for this was the assumption of good quality surface normals. If ground truth surface normals are used, the reflectance mapped version vastly outperforms the others. Unfortunately, even CNN based methods for estimating surface normals from single images are fragile and result in very noisy estimates.

## 3.2 Dematerial Network

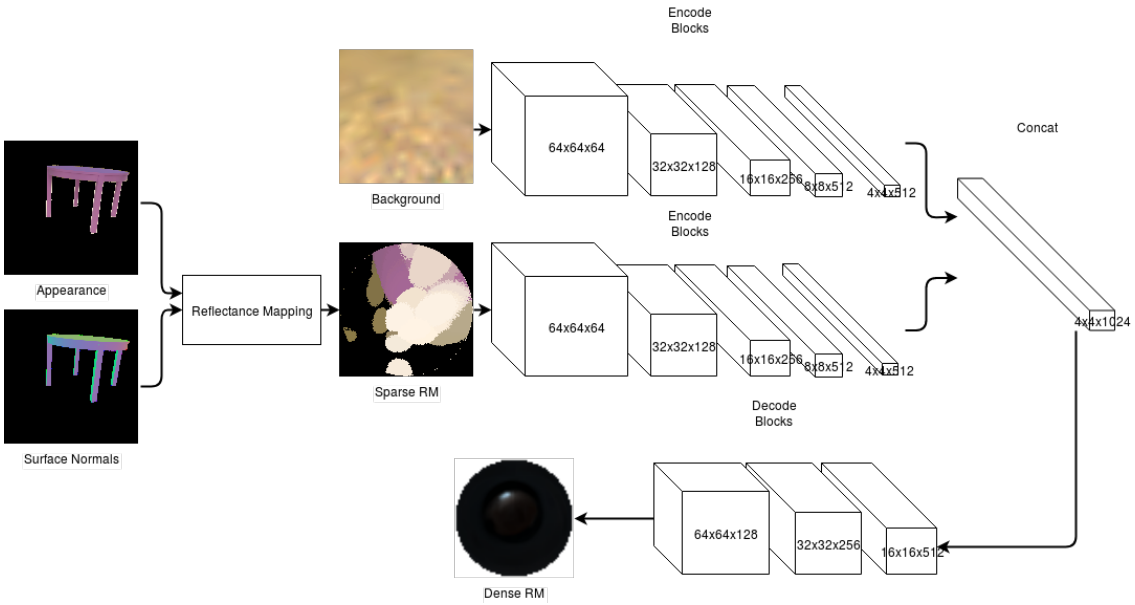



Figure 3.2:
The suggested architecture for 'What is around the camera', including the implied reflectance mapping step from 'Deep Reflectance Maps'.

During the implementation of the previous work, we constructed wrapper functions to use as building blocks to quickly construct *Encode* and *Decode* blocks, with variable layer counts for future networks. This made implementing the network from 'What is around the camera', called the Dematerial network, far easier. This work was provided with both the original dataset of synthetic and real images, as well as source code. However the source code was provided for the *MatConvNet* matlab library, which was slow, lacked documentation and unavailable on many platforms. As such we thought it beneficial to also reproduce this work in Tensorflow, using many of the utilities created for the last model. This network is very similar, taking a sparse reflectance map, along with a segmented background, as input and producing a High Dynamic Range environment map as output.

### 3.2.1 HDR Formatting

Unfortunately this task presented difficulties in data formatting that resulted in poor training until they were resolved. Firstly to encourage the network to extract luminance, the input data was first converted to CiE LAB color space using a Tensorflow utility from the *Pix2Pix* network[1]. The CiE LAB color space separates the luminance from the other color channels, making it more appropriate for lighting tasks. While the conversion proved accurate for the traditional RGB color inputs, it was unable to convert HDR formatted images, in this case the ground truth environment map. This is due to how HDR images are stored, as they capture a larger range of light intensities than RGB. For this task we used the Radiance RGBE file format. The data is stored as RGB values with an exponent, but interpreted in python as

[1] **https://github.com/affinelayer/pix2pix-tensorflow/blob/master/pix2pix.py**.

floating point RGB values with a very large range. For correct color conversion, this data needed to be formatted in such as was as to reduce the range to 0-1 without data loss. After experimenting with different ways of formatting the data we decided to use a logarithmic representation, after shifting the values to be above 1. In this format we were able to fully reconstruct the original HDR image, and the LAB space outputs of the network could also be converted into HDR without any loss of quality.

### 3.2.2 Training

The original work consisted of two models, one that used single material objects and one that used segmented multi-material objects. The multi-material network was a very simple improvement on the original, where the object was first segmented into 3 materials, and fed as input to 3 identical convolutional networks. The output of these convolutions was then concatenated and fed into the deconvolutional step to again produce a single HDR environment map. As we were able to achieve good results with the single material network, we thought it was unnecessary to also train the multi-material model, which effectively provided a denser reflectance map to the network. Again, we trained over 50 epochs, each consisting of 55,000 samples. The output to this architecture was a 64x64 HDR environment map that could then be imported into graphics software for qualitative analysis. Being such a low resolution, it bared little structural resembles to the original full resolution environment map. It did however appear to be a good approximation of the low-resolution version, and would result in very reasonable lighting when used to render objects, when compared to the original, as shown in 3.3.
It was found that the quality of the approximation depended heavily on the density of the provided reflectance map, and the complexity of the lighting in the scene. In cases where the scene was outside, even with very sparse reflectance maps, the network was able to infer features such as sky and ground color as well as approximate sun position. This was due to the simplicity of the scene lighting, leading to cases where the lighting conditions could be adequately approximated almost entirely from the background image. Some indoor scene, especially those containing a mix of synthetic lights and windows resulted in very poor predictions. In these scenes, very dense reflectance maps had to be provided to produce good results.
It is important to note that while these results were promising, they were based on reflectance maps obtained using perfect surface normals, limiting their use in practical applications. The aim of our further work was to eliminate the need to separate geometry estimation.

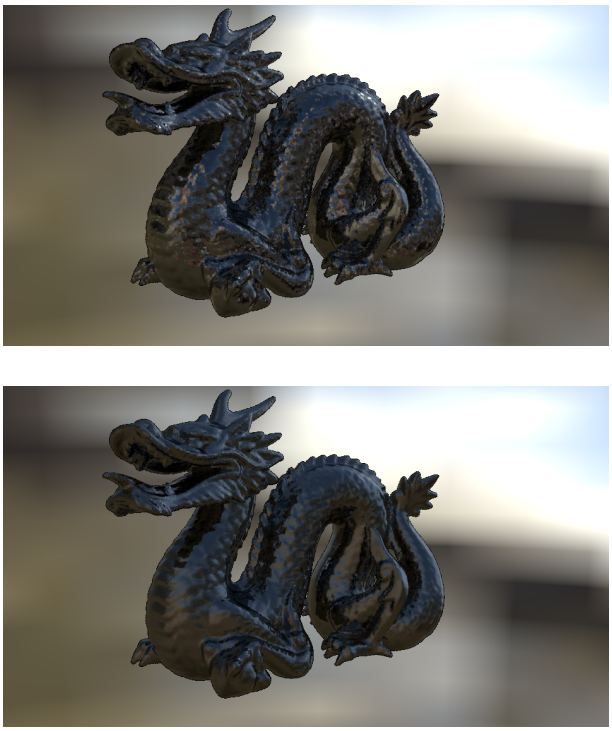


Figure 3.3: An example render using the ground truth lighting (above), and the lighting predicted from the Dematerial network.(below). The lighting predicted is feasible, but lacks much of the detail produced by bright individual lights. Note the original background has been composited in to be identical.

# Chapter 4

# New Stereo Model

## 4.1 Motivation

It was apparent from the results of the previous work that the sparse reflectance map and background combination was not enough data to construct a robust model for both indoor and outdoor scenes. By explicitly calculating the reflectance, ignoring shadow information and assuming point-like objects, much of the appearance data is removed. Furthermore the surface normal calculation required is impractical from a single image, due to the unconstrained nature of an object's appearance. When estimated surface normals are used, the resulting performance is poorer than simply providing the image to the network.

### 4.1.1 Suggested Method

Instead of relying on explicit reflectance maps, we took a new approach to design models that could exploit the additional spatial information of stereo images to infer surface normals, allowing the network to learn it's own model of reflectance. While this approach requires more information (multiple images), this information is available in use cases such as AR and film editing. One consideration was to simply use a stereo matching technique to calculate a relative depth map from the stereo images and find the surface normals. These estimated normals could then be used in the previous network to estimate the environment lighting. However stereo matching is a difficult task that often involves human intervention, relying on extracting identical convolutional features from the two images and finding their spatial differences. Most stereo matching techniques are slow, and while they can produce a good estimate of the depth of objects in a scene, they cannot produce surface normals to the degree of accuracy we require. Instead we construct a model based on learned stereo matching, providing the model with two views of an object, allowing it to learn an internal representation of geometry and lighting. By using two views with provided to two sets of convolutional layers, the network can extract similar features and exploit consistencies between views.

Figure 4.1: Generated data examples - Left, Right, Surface Normals, Environment Map (Tonemapped to RGB)

## 4.2 Dataset Creation

The most difficult part of many deep learning tasks, including this one, is the creation of an unbiased and generalized dataset. For the purposes of this task we needed a large dataset of stereo image pairs accompanied by an HDR environment map. Capturing this data in the wild is a very laborious process, and requires expensive camera equipment. Instead we took the more practical approach of generating and utilizing synthetic data, which presents its own set of unique problems. For this task we make use of the open source 3D modelling program 'Blender' and the accompanying raytracer *Cycles*. This made it possible to create photo-accurate renders with realistic lighting and materials. The data must be accurate in terms of lighting, material and geometry if the model is to be able to generalize to real world scenarios. To ensure accurate geometry we used the IKEA dataset [16] of 3D models, which includes a large range of furniture accurately modeled. The objects represent a good analogue for real geometry one might find in scenes, with some objects self-occluding and self shadowing. The models contained a mix of flat surfaces, edges and curves although being furniture the dataset contained a bias towards flat surfaces. We believe this is reasonable however, as most scenes in AR applications and films will be dominated by flat surfaces such as floors and walls.

To introduce further variance into the dataset, we make sure to apply different materials to the furniture to represent the range of materials found in real scenes. It is important that these materials react realistically to the lighting in the scene, rather than relying on explicitly lambertian surfaces. For this task we make use of Blender's 'Principled BSDF' shader [23], referred to as an 'Uber' shader for it's ability to represent any range of materials. Bidirectional Scattering Distribution Functions are able to accurately simulate the reflection of light on a surface, and by varying parameters such as albedo color, reflectiveness and subsurface scattering we are able to approximate a range of real materials. While most surfaces can be represented using this shader, our dataset generator does contain some restrictions. Notably, while we vary material properties and color, we do not apply texture to the object, so the subtle variations that can be seen in materials like cloth or carbon fiber are not simulated. However we believe that in most cases these subtleties are not visible in low quality video streams from AR platforms, and do not have a significant impact on the properties of most surfaces.

To generate our ground truth environments, and to light the objects we use HDRI images taken from[1]. These environment maps are used as both backgrounds and light sources by blender to illuminate the

[1] **https://hdrihaven.com/**.

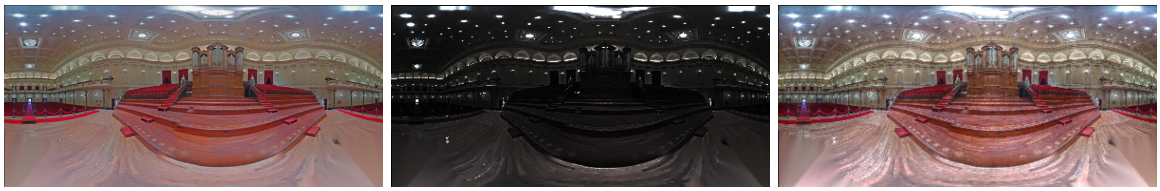


Figure 4.2: An example Low Dynamic Range (LDR) panorama, with dark and light exposures extracted the predicted HDR environment.

Source: Google

provided objects. These HDRI images are captured from real environments and are claimed to be accurately color corrected. To further augment the dataset and prevent over-fitting we introduce random rotation. An important difference in this network is that we do not provide the network an explicit background image, in fact the resolution of the background is lowered to give a depth-of-field effect. In real stereo images, the background would also move with the camera, however capturing this data is impractical. Fortunately due to the parallax effect, the background appears to move far less than the foreground anyway and so this approximation does not cause significant issues. For each stereo pair we load a random model from our dataset, apply a random material and select a random environment. We then move our virtual camera to a random orientation, within a reasonable range, facing the target object. We also re-render the environment map to have it's forward direction aligned with that of the observer camera. This prevents the network from simply learning to select from our modest set of HDRIs, and instead encourages it to learn to extract the environment entirely from the input. This generator was used to produce 55,000 image sets, making use of 28-core Intel processors on the Blue Crystal 4 supercomputer. After experimentation, it was found that due to the small image sizes and work distribution of path tracing, that high core-count CPUs outperformed the GPU renderer implementation. Examples of our generated data are shown in 4.1.

### 4.2.1 Bootstrapping Extra Light Probes

A significant issue with generating the data is the lack of large datasets of HDRI environment maps. We initially used a dataset of 175 environment maps from HDRIHaven.com, an improvement over the 105 in [20]. We found that our validation scores on unseen lighting continued to improve indicating that our model was not over-fitting to the lighting scenarios presented. Qualitatively however it appeared that the model would primarily learn features from the training lightmaps and was at times effectively selecting the closest previously seen environment. This could be confirmed by training a model without the background present, which would perform significantly worse. While this may be effective at producing realistic lighting, the aim of this work is to encourage the model to learn how the reflectance of an object maps to the lighting conditions. To mitigate this behavior and reduce over-fitting to the environment we aimed to increase the size of our lighting dataset considerably.
Collecting additional HDRI panoramas would be infeasible due to the lack of equipment and time so we took the approach of producing HDR estimates from existing LDR content. The biggest available dataset of LDR panoramas is Google Street View [30], with both indoor and outdoor scenes. While this dataset is huge and provides a more realistic sample of lighting conditions, it is only available in low dynamic range. Fortunately, there is plenty of previous work in the area of converting LDR content to HDR, particularly using CNNS to find light sources and tone-map content. We make use of HDR-Expandnet [31], a very recent work that uses both a local and global branches to produce accurate tone-mapping predictions. It is important to note that this network is intended as a general purpose image regressor, and is not trained on panoramas in particular. It would certainly be possible to improve results by retraining, or fine-tuning, the network on our existing set of HDR panoramas to produce better light probes. We make use of the image stitching and inpainting tools within OpenCV to create panoramas from different views on Google Street View, and use the aforementioned HDR-Expandnet to bolster our dataset with an additional 237 lightmaps. An example is shown in 4.2 Our models are trained both with and without these additions to find any issues with over-fitting or malformed data produced by ExpandNet.

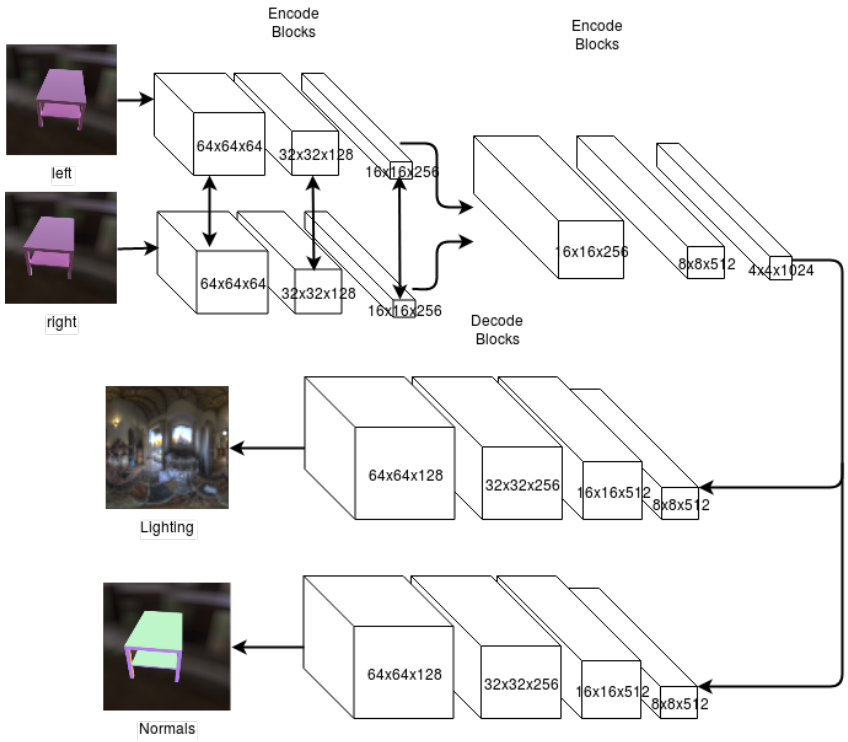



Figure 4.3:
The first stereo image architecture. Weights are shared in the first 3 encode blocks, and the outputs are simply concatenated.

## 4.3 Model Architecture

### 4.3.1 Siamese Layers

To construct a model that produces good results while minimizing network size, it is important to carefully consider the problem being solved and the features being learned. By using stereo views we aim to find correspondence between similar features and an understanding of geometry due to the parallax effect. To this end we present results from a *Siamese* network, where the weights in the convolutional layers are shared, such that the same features are extracted from each view. The two inputs are fed into two identical 'Encode' blocks, consisting of a number of convolutional layers with batch normalization and max pooling. However, during the training process, only the weights for the first path are trained. These weights are duplicated in the second path to ensure that the same convolutions are being performed on each image of the pair. The output of these two paths is then concatenated before being passed into the deconvolutional steps, as shown in 4.3.

### 4.3.2 Cosine Similarity Pyramid

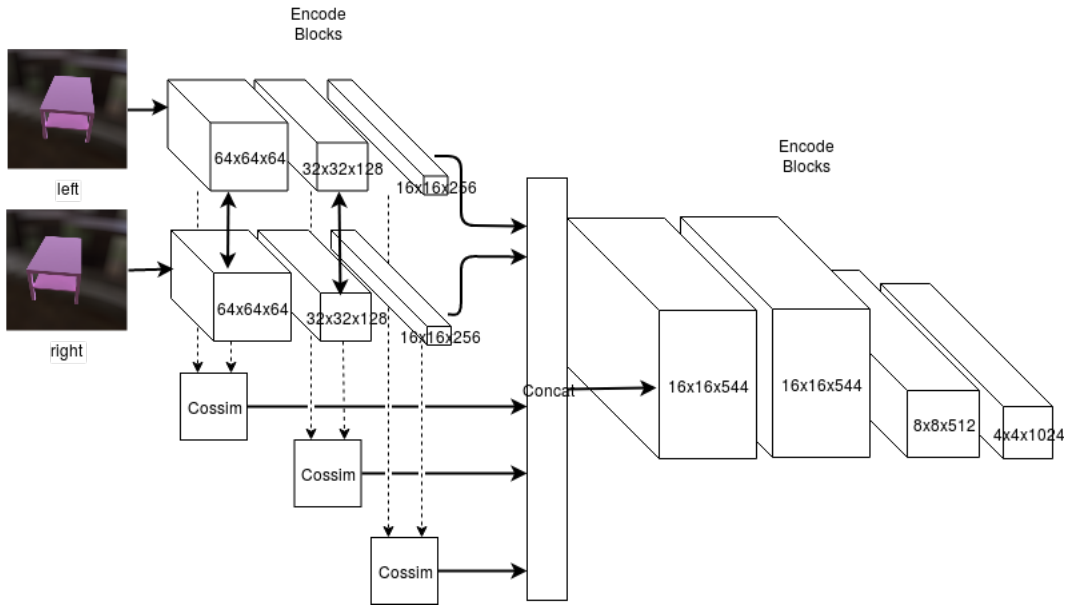



Figure 4.4: The Cosine Similarity Pyramid as implemented. Decode path removed for brevity.

We can further encourage the network to perform stereo matching by using the dot product technique as suggested in [15]. After using shared weights to extract the same features from each image, we then perform a dot product operation on the L2-Normalised output to find the *Cosine Similarity*

$$\boldsymbol{a} \cdot \boldsymbol{b} = |a||b| \cos(\theta)$$

which represents the cosine of the angle between the two inputs. On a raw image, if two pixels are identical in value, they will have a cosine similarity of 1, and so we would find the nearest matching points from the values closest to 1. Instead of comparing individual pixels, we instead compare the intermediate representation, to find the similarity between features rather than pixels. The resultant correlations then represent the feature disparity maps. The correlation can then be concatenated with the original features before being passed as input to further convolutional layers, starting with a *Fusion* layer.
A drawback of traditional block-disparity methods for stereo matching is that the block size significantly affects the accuracy of the calculated disparity. Larger block sizes have the important benefit of reliability, as it is easier to find unique matching blocks within the two images than it is to find matching pixels, for example. In our *Cosine Similarity* step, we are effectively performing a block disparity on a spatially reduced representation of the image. Industry standard stereo matching methods deal with this by using a pyramid of block sizes, and fusing the results. To avoid suffering the same problems, we present a different model using a *Cosine Similarity Pyramid*, where we perform our similarity step of each intermediate stereo output, as shown in 4.3.2. This way, we can estimate similarity of features at different scales, exploiting both the robustness of large blocks and the accuracy of small ones. Each similarity is concatenated to the stereo output as before. This method does present a significant performance compromise, requiring more computation and memory to produce our full intermediate similarity representation.

### 4.3.3 Training Details

We experimented with different model parameters to achieve the best validation accuracy. For each architecture, we varied the number of convolutional layers per encode-block, making it easier to test different model depths without major changes to the codebase. We discovered early on that deeper models consistently perform better than their shallow counterparts, but somewhat mitigate the benefits of our Cosine Similarity steps. However it is important to consider the computational performance of the result and so we test all architectures at a range of depths.
The original Dematerial network was trained with a *Gradient Descent* optimizer, using a momentum on the learning rate such that it decreases over time. The idea is that the learning rate decreases as the

model reaches the optimum, to fine-tune the weightings. We found that we could achieve faster training using the *Adaptive Momentum Optimizer*(ADAM), which maintains a per-parameter learning rate using the second moments of the gradients after loss calculation to produce new weights. We were also able to use a higher learning rate of 1.7e-5. Often the performance of details such as these are very dependent on implementation details like the framework used or the hardware being trained on. As a result our choice to make these changes is not alone an indication that the previous work could be improved in the same way.
Our deep model uses 9 convolutional layers per input branch, 9 convolutional layers for branch fusion and 9 more to produce the environment map. In comparison the original Dematerial network used 13 convolutional layers per branch, and 15 to produce the environment map. To evaluate our new model features we experiment with different model depths.
For our prediction task we make the following assumptions:

- Provided objects consist of a single material with constant albedo. Materials can contain Diffuse, Specular and Subsurface properties.
- Geometry and lighting remain static between both stereo views.
- Stereo view distance is kept constant, though object size can vary. Views are also parallel, and images are not manually rectified.
- Subjects are modeled as floating objects.
- Camera intrinsics are kept constant.

## 4.4 Experiments

To determine the best features for the task, we evaluated the performance of a suite of configurations:

- Dematerial model with known geometry.
- Basic Concatenation model without shared weights.
- Basic Concatenation model with shared weights.
- Single Dot Product model.
- Single Dot Product model without background data.
- Single Dot Product model with augmented data.
- Cosine Similarity Pyramid model.
- Cosine Similarity Pyramid model with Multi-Scale features.

All models were trained for 100 epochs, with a learning rate of 1.7e-8, and evaluated on 1000 validation samples. A batch size of only 24 was used, due to memory requirements of the larger networks.

# Chapter 5

# Critical Evaluation

## 5.1 Experimental Results

### 5.1.1 Effect of Shared Weights

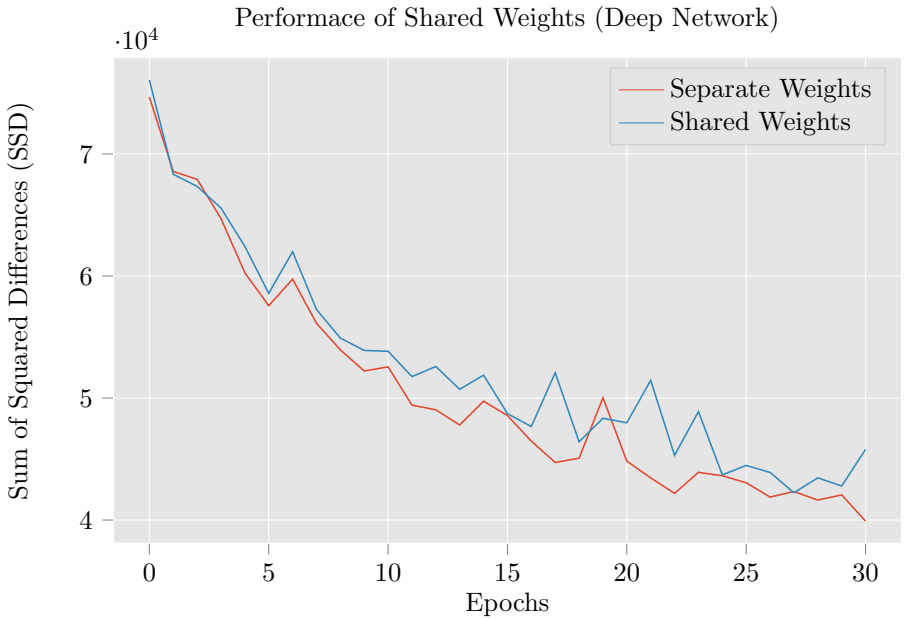



Figure 5.1: SSD between the predicted lighting and ground truth on the validation set, at each epoch during the training process. In both models, 3 Convolutions are used for each Encode/Decode block, and the outputs of the stereo branches are concatenated.

We found that using shared weights on our basic concatenation model had very little effect on the accuracy of results, shown in 5.1. In our training process we use the L1 norm loss to determine how to reweigh our graph. For the siamese network, new weights are only calculated for the left branch, and then duplicated to the right branch. When the weights are trained separately, they will eventually converge on, or even outperform the original weightings.

### 5.1.2 Concatenation VS Dot Product

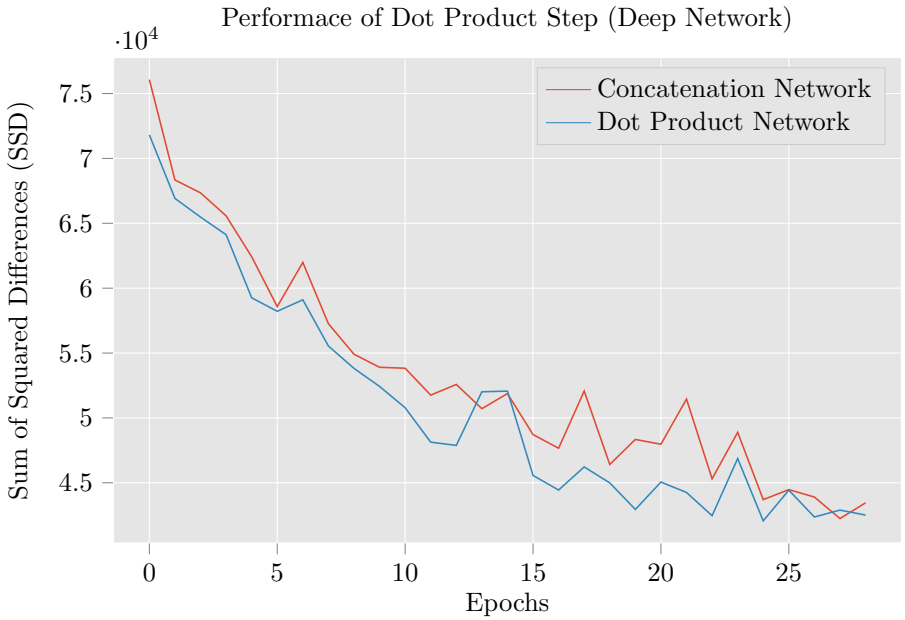



Figure 5.2: SSD between the predicted lighting and ground truth on the validation set, at each epoch during the training process. In the first model, the outputs from the stereo branches are concatenated, in the other we use normalization and dot product to determine Cosine Similarity. Both models use 3 layers per block.

The dot product step results in a small improvement in accuracy (shown in 5.2) and a significant improvement in training speed. The dot product step is effectively a manually calculated similarity measure between the two views, and so the model can converge on the depth features earlier in the training process. We believe that the performance difference is minimal as the model is already very deep, and training would likely converge on the depth features eventually. As a result we also show the effect with a much shallower model.

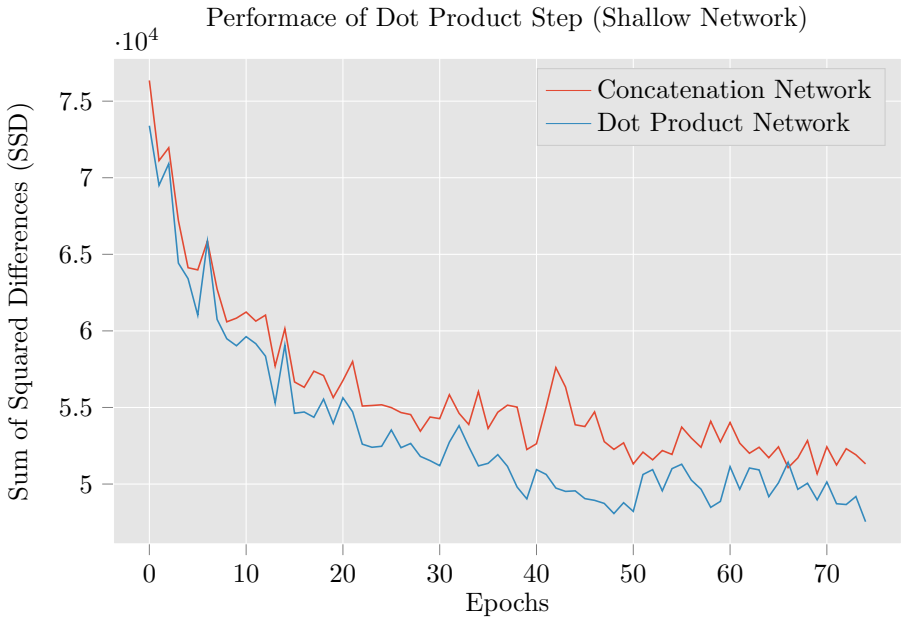



Figure 5.3: In the first model, the outputs from the stereo branches are concatenated, in the other we use normalisation and dot product to determine Cosine Similarity. Both models use 1 convolutional layer per block.

The shallow model uses 1 Convolutional layer per Encode block, resulting in 3 convolutional layers per stereo branch, 3 more for branch fusion, and 3 to produce the lighting. The memory requirements for this graph much lower than the deep model, but the performance is very similar. In 5.3 we can see a consistent, although modest, improvement in performance from using the Cosine Similarity. We believe a limiting factor in this approach for lighting estimation is that many differences in stereo views are not due to the relative movement of geometry relied upon for depth calculation. For example, specularity depends heavily on view direction. By performing the dot product we are treating these features the same as geometric features.
Another issue with this is that we are also performing the dot product on features extracted from the background. In our case the background is modeled to be infinitely far away, and while it contains useful contextual and hue information, should not be used in the similarity measure.

### 5.1.3 Effect of Background

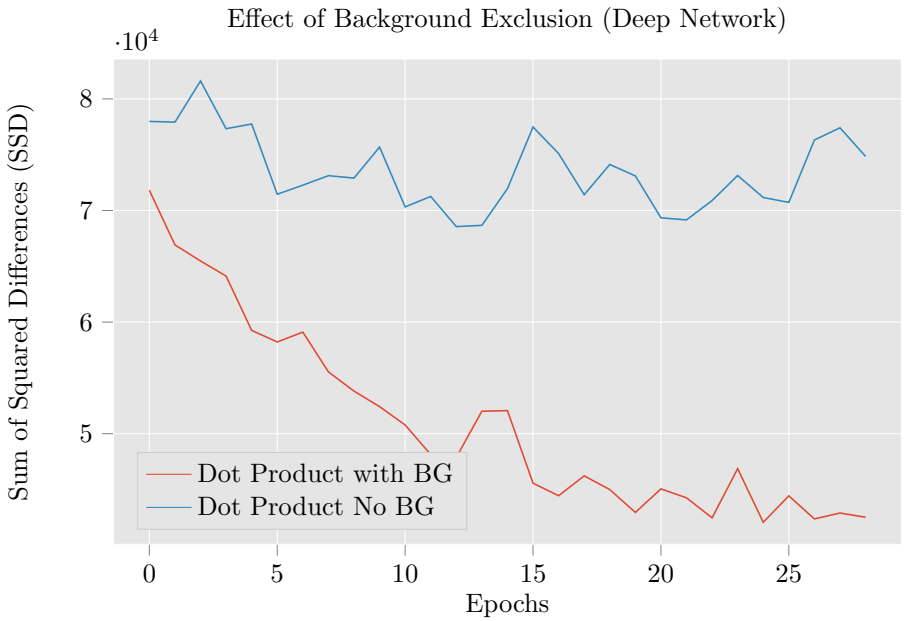



Figure 5.4: In the first model, the object is composited onto the background, while in the second model, the background is not present and replaced with 0 values.

The model is able to infer some of the lighting without the background, shown in 5.4, indicating that it is learning the lighting conditions from the subject. However, adding the background improves the performance significantly. The background not only gives an indication of the physical environment the subject inhabits, but also the colors and intensities to expect. This data, in theory, should make it easier to determine the difference between reflectance, albedo and shadows on the subject as well as indicate the hue of the resulting environment map. Our results indicate that the background information is useful for estimating the lighting accurately. For a robust model we cannot exclude the background, however it cannot be included in our Cosine Similarity steps. To solve this we present a new network where the background is given it's own branch.

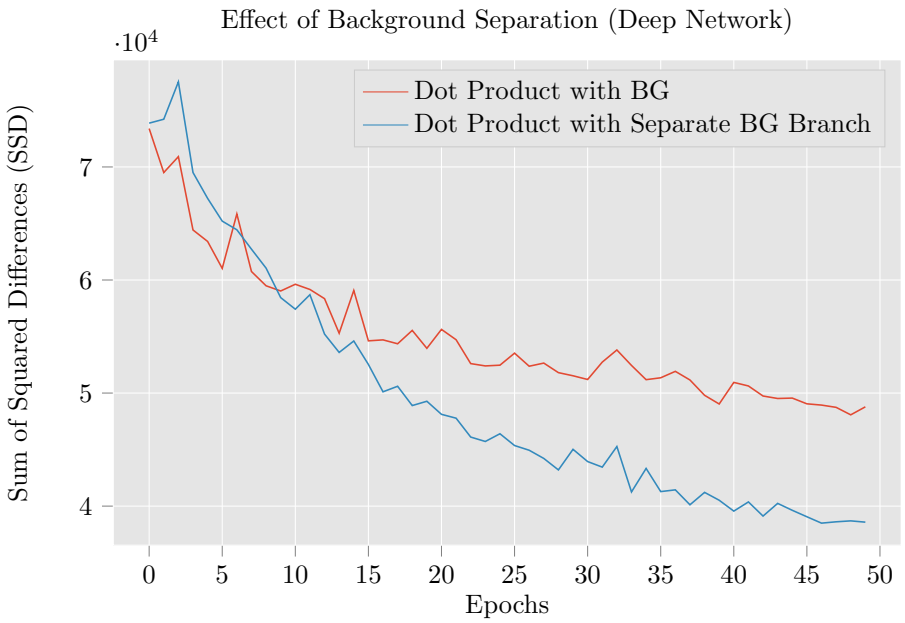



Figure 5.5: In the first model, the object is composited onto the background, while in the second model, the object is provided on a 0 value background, while the real background is provided to a different branch. The output of this branch is concatenated to the dot product output.

A separate background, as used by the Dematerial network, results in significant improvements over our original network. This is demonstrated in 5.5. In fact, the shallow network now outperforms the deeper networks that didn't include our augmentations.

### 5.1.4 Cosine Similarity Pyramid

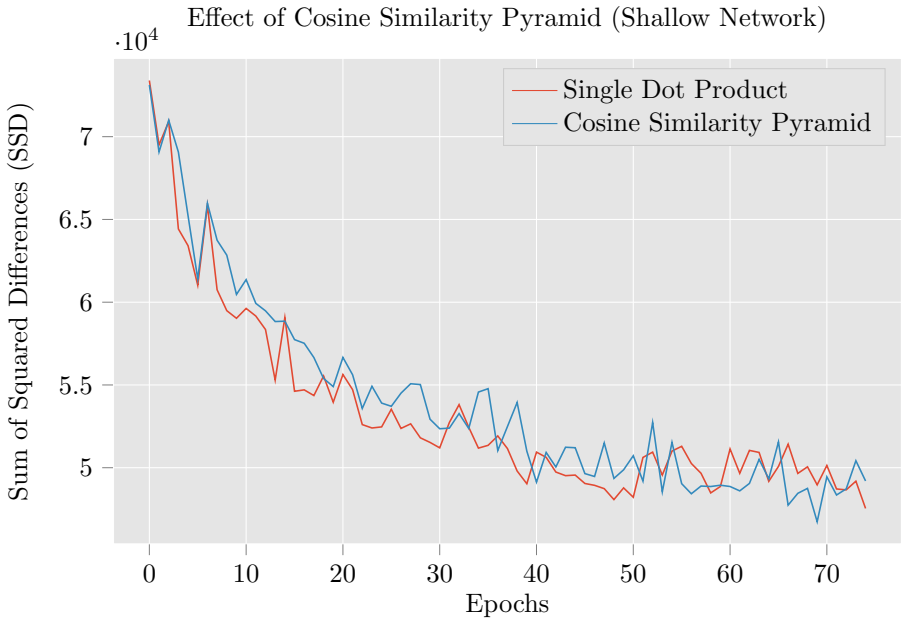



Figure 5.6: Comparison of model with single Similarity layer, and model with a pyramid of similarity measurements.

With the gain in performance from performing a Cosine Similarity on the stereo branch output, we expected that we could gain additional performance by finding similarities at a range of layer depths. In theory, we would find the feature similarity across receptive field sizes, giving more depth information to

the fusion network. However we found this had little effect on accuracy, shown in 5.6. We believe that the features are unnecessary in the scenario provided - that of a single floating nearby object. In more complex scenarios, such as depth inference across a full scene, finding similarity at a range of receptive field sizes may use useful. Furthermore if the features are useful, the fusion network would need to be deeper to extract the useful information.

## 5.2 Final Model Accuracy

### 5.2.1 Metrics

From our experiments we determined to choose our shallow stereo model, with a separated background and a Cosine Similarity block as our final model for evaluation. To evaluate the accuracy of the model we use a suite of image comparison measures on the produced HDRI environment maps. It is essential that the metrics used suitably evaluate the image similarity, and so we consider which image properties are most influential of the resulting lighting. The most basic estimation should be able to capture the direction and intensity of direct lights within the scene. Often there are few direct lights, but their effect on lighting dominates the overall scene illumination. Better predictions will be able to capture some information on indirect lighting, such as sky or wall color and brightness.
We start with two basic difference metrics, the L1 norm and L2 norm, which are both per-pixel Minkowski distances. These measures are fast to calculate and simple to implement and so are often used as the loss function in image regression tasks. We use the L1 norm for training loss, and the L2 norm for evaluation. L2 norm, also called *Mean Squared Error*, involves the squaring of pixel differences, and exaggerates the effect of large pixel differences on the overall similarity score. In the HDR domain this can mean that direct lights, such as the sun, which are predicted to be in a slightly different direction, will have a severe impact on the similarity, despite having a minor effect on the resulting illumination. When we inspect light sources in sRGB photographs, it appears as if the light source is large and emitting light from a radius, as the higher brightness values fall outside of the cameras exposure bracket. In HDR, light sources tend to contained within a few pixels, with surrounding pixels having exponentially lower intensities.
Another commonly used image comparison metric is the SSIM [22],

$$SSIM(x,y) = \frac{(2u_x u_y + c_1)(2\sigma_{xy} + c_2)}{(u_x^2 + u_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

which aims to predict the accuracy of an image with human perception. It is a measure of structure, which takes luminance and contrast into account, and so is better suited for HDR. To measure dissimilarity we use the DSSIM

$$DSSIM(x,y) = \frac{1 - SSIM(x,y)}{2}$$

Finally, we assess the models ability to identify the direction of the prevailing light source, by taking the distance between the brightest pixel in the ground truth and predicted image. For assessment we use a validation set of 1000 images, under new environment maps and models with random materials and rotations.

### 5.2.2 Results

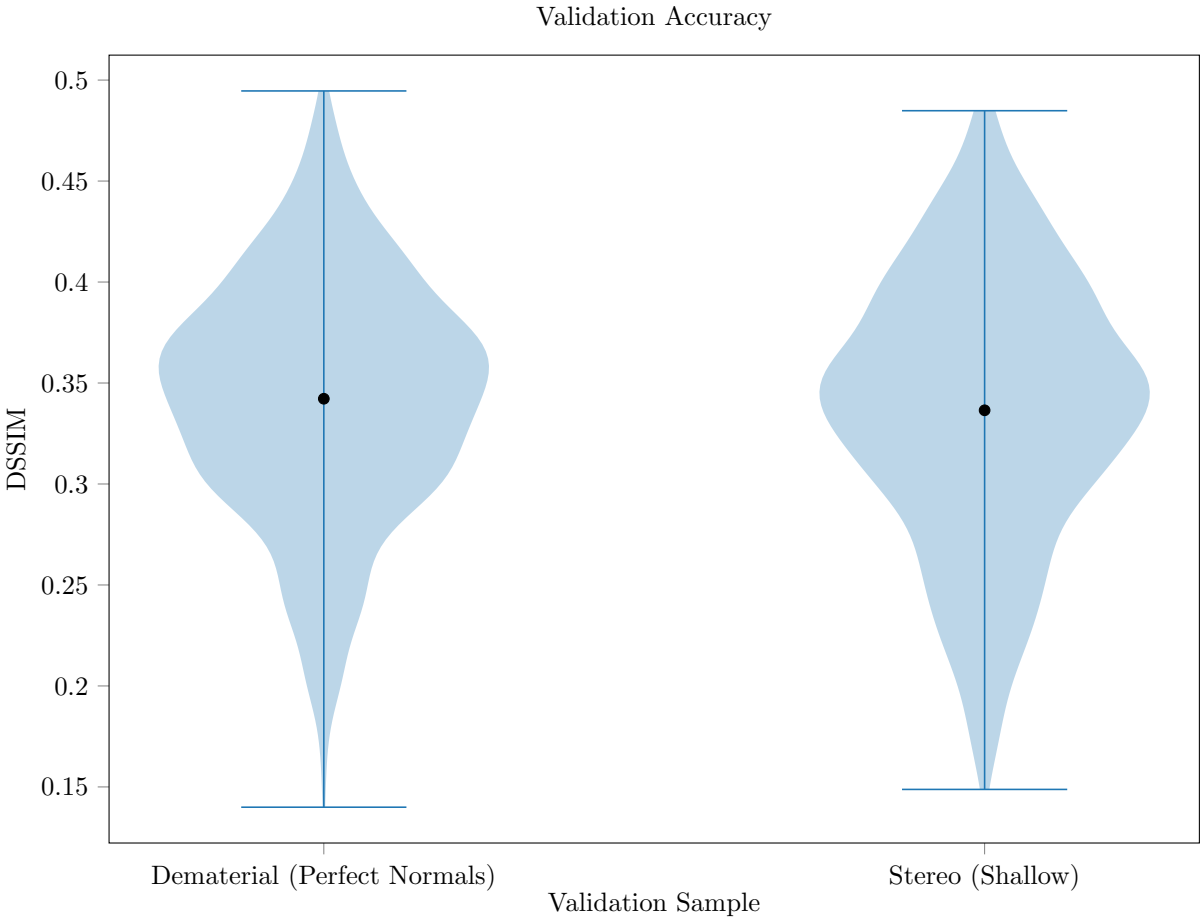



Figure 5.7: Comparison with the original work and our models (lower DSSIM is better). All models are retrained on our training set, and tested on our validation set.

On our synthetic benchmark, our stereo model achieved a Mean Squared Error, between the predicted and ground truth light maps of 1134.83, with the 50th and 75th percentile being 1.92 and 7.42 respectively. This implies that while the model performs well in most tested scenarios, there are some cases where the predictions are vastly incorrect, as shown in 5.8. It is also clear from the qualitative results that the MSE is not the ideal metric in our case. While it favors structure and sky location, it seems to fail to capture the primary light direction. As a result, renders produced with the best MSE results tend to have good environment lighting, but completely incorrect shadowing due to a failure to predict direct lighting.

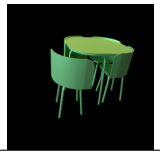
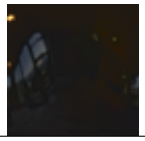
| MSE Evaluation | | | | |
|---|---|---|---|---|
| | Input Data | Ground Truth Environment | Predicted Environment | MSE Score |
| 25th Percentile(Synth) |  |  |  | 0.56 |
| 50th Percentile(Synth) |  |  |  | 1.92 |
| 75th Percentile(Synth) |  |  |  | 7.42 |

Figure 5.8: Demonstration on test set.

For the DSSIM test, the stereo model achieves a respectable average of 0.34, with 50th and 75th percentiles of 0.34 and 0.38. Here the spread of results is much tighter, and we can see from 5.9 that the metric captures the structure of the environment very well. These results are better than many of the models of Georgoulis et al. [35], which all rely on known geometry. We compare these in 5.7, though it is important to note that the Dematerial network uses perfect, not estimated normals.

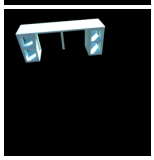
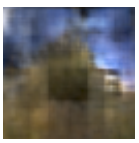
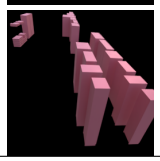
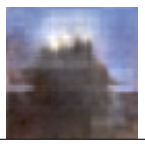
| DSSIM Evaluation | | | | |
|---|---|---|---|---|
| | Input Data | Ground Truth Environment | Predicted Environment | DSSIM Score |
| 25th Percentile(Synth) |  |  |  | 0.30 |
| 50th Percentile(Synth) |  |  |  | 0.34 |
| 75th Percentile(Synth) |  |  |  | 0.38 |

Figure 5.9: Demonstration on test set.

We assess the models ability to predict the main light source using pixel distance between the highest values in the predicted and ground truth environment. Our model produces an average pixel distance of 2, with the maximum being 45 for a 64x64 map. The 50th and 75th percentiles are 20 and 29 respectively. This demonstrates a limitation in our models ability to predict primary lighting, which is a significant issue especially if the environment map is used to produce shadows. We must note however that many of the environments used for validation do not contain a single primary light direction, with many being outdoor overcast scenarios or indoor scenes with soft lighting, which can be seen in 5.10.

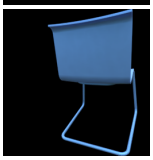
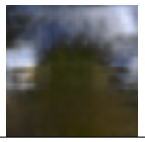
| Direct Light Direction Evaluation | | | | |
|---|---|---|---|---|
| | Input Data | Ground Truth Environment | Predicted Environment | Average Distance between brightest pixels |
| 25th Percentile(Synth) |  |  |  | 0.3 |
| 50th Percentile(Synth) |  |  |  | 20 |
| 75th Percentile(Synth) |  |  |  | 29 |

Figure 5.10: Demonstration on test set.

Our method outperforms the original 'What is around the camera?' work for single material objects with estimated surface normals. Furthermore our network uses fewer layers, and removes the need for known surface geometry in favor of stereo views of the object. We must note that the the differences in training data between the original work and ours have a significant impact on the resulting accuracy. The original model was trained on classes from the Shapenet 'Car' class, mostly consisting of curved objects. This means that the image contains more surface normals to estimate light from, and build a denser reflectance map. The original work also only contained materials from the MERL BRDF dataset, capturing 100 different common materials. Our material range is more complete, using materials generated from BSDF shader parameters.

## 5.3 Final Model Performance

It is not only important to assess the accuracy of the predicted lighting, but also the resource requirements. A model that produces rough approximations with a lower memory footprint and computation time is still valuable, especially in the field of AR. For this we assess the memory usage and average inference time for our validation set. Testing the model performance across different hardware configurations is beyond the scope of this project. Our model achieved an average per-image inference time of 20ms using an Nvidia GTX 1070 consumer GPU. For reference, this is just over 50 frames per second, above the minimum frame-rate expected for smooth non-interactive video, and approaching the 60fps target for interactivity. The performance of the model is acceptable for use in video editing on home desktop computers and laptops. For use in mobile AR however, the inference time is a little too slow to be used live on very frame. However as lighting conditions tend to remain constant for long periods of time, running inference every frame would be wasteful. A more practical approach would be to recalculate the lighting every time a major change in luminance values in the frame is detected, or when the user places an object - there is little to gain from calculating the lighting in physical areas where virtual objects will not be placed.

# Chapter 6

# Conclusion

## 6.1 Achievements

We have presented a CNN approach for estimating the lighting at an object that exploits multiple views, eliminating the need for known surface geometry. To do so, we implemented prior research in the Tensorflow framework, confirmed the findings of Georgoulis et al. and identified the main limitation of their approach. To address the need for known surface information, we researched depth estimation techniques and proposed a new model to exploit a learned stereo matching. To train our models on enough data we collected a set of 175 HDR environment maps, the largest dataset used for this task outside of [36]. Furthermore, we experimented with methods to augment this dataset with Google Street View data, using cutting-edge work in LDR-HDR prediction. We presented a method to assist the model's ability to infer depth using a Cosine Similarity step, performing the dot product on the outputs from convolutions with shared weights. Finally, we train a model that performs as well as the previous work, with a smaller memory and computation footprint, without known geometry.

## 6.2 Conclusions

### 6.2.1 Reflectance Mapping

Firstly, reflectance mapping is restricted in it's usefulness to objects with sufficient reflectance and curvature. The technique does provide a clear benefit, in removing the understanding of geometry from the problem. Furthermore by mapping the reflectance onto the inside of a sphere, we are given a representation that is easier to interpret to an environment map. However, if surface angles are missing from the subject, the incident light at that angle will be difficult to recover. Similarly, diffuse objects make it difficult to extract the full range of light intensity as the specular component is less prominent. The most obvious limitation of reflectance mapping however, is the need for very accurate surface geometry representations. If the normals provided are noisy or incorrect as in most surface estimation techniques, the approach breaks down. The reflectance mapping technique presented by [20] is only an approximation of reflectance, and is only valid for objects with uniform materials and no shadowing. In reality this limits it's use to purely convex objects. We believe taking a more general approach such as ours is the way forward if more complex scenes containing shadows, textures and multiple materials are to be considered.

### 6.2.2 Dot Product Layer

We have also demonstrated that taking explicit steps from existing stereo matching techniques can accelerate the training of deep neural networks, as well as reducing inference time. In our case, the problem of disparity matching and depth estimation is well formalized and there are many existing solutions. We can take advantage of this by incorporating steps from these, and treating our CNN as a model for learning parameters and refining results. A downside of this however is that we can potentially miss better learned solutions. We have demonstrated that this is not the case, with our dot product model outperforming generic deep models. While the dot product is a very optimized operation, that receives much attention from driver and framework developers, it still uses a significant amount of GPU memory. On our system this is not a problem, but it may cause performance issues on mobile devices. When we compute the dot product we are interested in the most similar features between two images, not necessarily the similarity

between each and every features. A more memory conservative approximation that captures this behavior could potentially improve performance.

### 6.2.3 Training Data

We recognize that synthetic training data is limited in it's ability to generalize. However we believe that our work demonstrates that the task of lighting estimation from stereo views is possible. By using physically accurate shaders, ray tracing and environment lighting captured from real scenes, our model is exposed to the majority of variables encountered in real data. Given enough training data captured from real scenes, augmented with better synthetic data our model should be able to learn real world conditions. Collecting real lighting data is arduous and expensive. Attempts to automatically augment LDR panoramas, as we have attempted, will struggle to capture the intricacies of reflectance. For work in lighting estimation to reach commercial application, real high quality datasets of environment lightings need to be available. This process would require an automated process of HDR panorama capturing, exposure compositing and finally stitching. The process could be improved even further with dedicated hardware to capture panoramas.

### 6.2.4 Loss Functions and Metrics

We believe that the SSIM is a good metric for evaluating the similarity of lighting predictions to the real environment. However it is still clearly limited, as it is designed for use with regular non-spherical images. While the objects we render with our lighting predictions often have the correct hue, and reflect sky and floor color well, the main light directions are often missing. This proves an even bigger issue if the environment map is used for shadows. For assessment we present an alternative measure, but it is clearly a narrow evaluation that cannot be applied to indoor images with multiple light sources.
The aim of our work is to predict a lighting representation that is feasible enough to use for compositing new renders. We do not expect to fully recover all the details of the environment that are not in view of the camera. If we are to create an evaluation metric, or even a loss function that captures the quality of lighting, we need to consider how the lighting affects rendered objects. We believe that a good approach to this would be to use a differentiable raytracer to render composite objects with the generated lighting. Traditional image comparison metrics could then be used to assess the performance. Given that most deep learning frameworks use a computation graph, much like many commercial renderers, this is not beyond the realm of possibility. Furthermore it would then be possible to experiment with different lighting representations to environment maps, such as a primary and secondary light vector.

## 6.3 Limitations

### 6.3.1 Robustness of Reflectance Mapping

While an improvement in practicality over previous work, our system is still bound by several limitations to varying degrees of severity. By eliminating the explicit reflectance mapping step we reduce the amount of preprocessing that must be performed on the input data, and allow our model to learn more subtleties and lighting features in the images. However, our system is far less robust than using an explicit reflectance mapping, if surface normals can be provided. By using reflectance maps the model does not need to learn any geometric features, significantly constraining the problem to that of differentiating the material from reflected light on a spherical surface. In cases where reasonable preprocessing and human input can be applied it would be sensible to use the previous work for more consistent results. For example it is easy to envisage an application where users simply highlight an object in a video frame, and select from rough geometric shapes to use in place of surface normals.

### 6.3.2 Non-Parallel Views

The term 'stereo views' simply refers to two images of the same scene, and how these views differ must be considered to make robust stereo systems. In our work we use parallel cameras with the same distance between them. This reduces the number of variables for our model to learn, and helps us formalize and understand the problem of surface estimation as one of calculating disparity. This not only limits the use cases of our solution to those using stereo hardware or panning shots, but also limits how much geometry and reflectance information we can capture from the scene. One option would be to use angled stereo,

with the two shots facing towards the same focal point. This would capture more angles of the subject and therefore more possible reflectance directions. Alternatively, a more general stereo camera system could be learned using horizontal image translation. With HIT, one image is translated according to the stereo transform to account for the camera distance. This way the images provided to the network could be produced by a range of stereo camera configurations.

### 6.3.3 Over-fitting to Environments

Another concern is that of over-fitting to trained environments. We demonstrate our model on a validation set containing new sets of lighting. However our training set is small and comes from a single source, that produces lighting maps for use in 3d rendering. As a result many of the environments would have been selected for how pleasing the lighting is, rather than how representative they are of likely conditions. For example overcast outdoor shots and small indoor scenes are very underrepresented. As a result, our best predictions consistently come from sunny outdoor scenes. Unfortunately finding HDRI environment data is very difficult and often comes with these problems. Fortunately a new dataset has recently become available from Gardner et al. [36].

### 6.3.4 Uniform Material

As with previous works our system uses some assumptions about the input data to predict lighting, mainly that the given object is of uniform albedo. While this is a fair assumption, as many real world objects can be approximated with uniform color, it does in fact limit the models ability to predict surfaces. Disparity mapping and stereo matching rely of finding salient points, features and blocks in multiple views. Under uniform albedo it can be hard to find these features. Furthermore, the light reflected into the camera from an object can change depending on the view direction, breaking the assumption that two image points of similar appearance map to the same geometric point.

## 6.4 Future Work

Our work demonstrates progress in lighting estimation, by inferring object geometry and exploiting changes in shading and reflectance between multiple views. However the model presented is limited in scope, and could be extended into more useful models of scene lighting.

### 6.4.1 Multiple Light Probes

Our model effectively uses an arbitrary object in an image as an approximate light probe for the environment. Often individual probes are missing surface normals, and are not always reflective enough. To get a better estimate it would be sensible to combine the inputs of multiple probes in the network. This could simply be achieved with multiple input stereo branches, joining the input through concatenation. However, the lighting on different objects in the same scene can differ drastically, as their relative position to the light source is different. This would present new challenges in weighing the light maps based on their position to produce an average scene lighting.

### 6.4.2 Texture Resilient Inference

Our model is only demonstrated on objects of uniform albedo, rather than those with varied colors and materials. This constrains the problem as lighter parts of the surface can be assumed to be more brightly lit than darker ones, rather than having different material properties. Konstantinos Rematas et al. tackle this problem with preprocessing step, where an object is masked into 3 materials using a k-means clustering on the pixel values. Each masked object section is then fed into the network and combined in intermediate layers. The downside of this is that the system can only cope with a set number of materials, with each material maintaining a constant albedo. An ideal system would be able to differentiate between changes in pixel values that are due to light and those due to material. This can somewhat be achieved by utilizing the CIE LAB color space, but there are still occurrences where it is difficult to tell the difference between a dark material and a shadow. This is a difficult problem but can be somewhat approximated by finding surfaces with constant albedo or repeating patterns. Fortunately in indoor scenes, most objects are made of single explicit materials, making this easier to solve. Outdoor

scenarios typically have less complicated lighting patterns, with the sun providing a single light source. In this case it can be assumed that shadows are cast in the same direction.

### 6.4.3 Video Inference

We have presented a system that is able to make use of parallel stereo views to infer the lighting at a point. However both suggested use cases of AR and video editing involve video footage with 6 degrees-of-freedom camera movement. This introduces a vast amount of complexity into the stereo matching problem; commercial solutions still struggle to solve camera movement that involves simultaneous rotation and translation. There has been recent progress in monocular slam to solve this, that integrates the inertia sensors with the camera from a mobile phone to determine roughly when the camera is being translated. In our case, a rotating camera may actually prove beneficial, as we are interested in the makeup of an object within the scene. By capturing information from different angles of our subject we can discover more about it's geometry and how it's reflections change with viewing angle. It could be useful to make use of an arbitrary number of video frames to continually refine the estimation.
This could be achieved with a recurrent neural network, that maintains an internal representation between frames. Using the inertia sensors in a mobile device along with camera tracking, the predicted lighting could be rotated to match the orientation of previous predictions. By averaging over time it would be possible to produce a more accurate prediction. Another option would be to rotate the input. If camera tracking can be used to separate camera rotation from translation, input frames could be composited into a rough panorama. This could be provided as a much more complete input to a network, while also maintaining spatial similarity to the produced environment map.
Another more novel approach would be to take inspiration from Microsoft Research [9], where the geometry, lighting and material estimations are treated as separate problems and refined. We have seen how deep learning on stereo frames can be used to accurately predict lighting. It is likely that a similar method could accurately predict the surface normals or material. It may be possible to improve on this work by building multiple networks that are trained together to refine their respective predictions over a series of frames.

# Bibliography


[1] D. S. Immel, M. F. Cohen, and D. P. Greenberg, "A radiosity method for non-diffuse environments," *SIGGRAPH Comput. Graph.*, vol. 20, no. 4, pp. 133–142, Aug. 1986, ISSN: 0097-8930. DOI: 10.1145/15886.15901. [Online]. Available: http://doi.acm.org/10.1145/15886.15901.

[2] J.-F. Lalonde, A. A. Efros, and S. G. Narasimhan, "Estimating natural illumination from a single outdoor image," in *IEEE International Conference on Computer Vision*, Oct. 2009.

[3] P. Debevec, "Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography," in *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '98, New York, NY, USA: ACM, 1998, pp. 189–198, ISBN: 0-89791-999-8. DOI: 10.1145/280814.280864. [Online]. Available: http://doi.acm.org/10.1145/280814.280864.

[4] J. F. Blinn and M. E. Newell, "Texture and reflection in computer generated images," *Commun. ACM*, vol. 19, no. 10, pp. 542–547, Oct. 1976, ISSN: 0001-0782. DOI: 10.1145/360349.360353. [Online]. Available: http://doi.acm.org/10.1145/360349.360353.

[5] M. Anderson. (Jul. 15, 2009). Jeff kleiser discusses the early cgi of flight of the navigator, [Online]. Available: http://www.denofgeek.com/movies/14631/jeff-kleiser-discusses-the-early-cgi-of-flight-of-the-navigator.

[6] Apple. (). Arlightestimate, [Online]. Available: https://developer.apple.com/documentation/arkit/arlightestimate.

[7] (Feb. 23, 2018). Environmental light, [Online]. Available: https://developers.google.com/ar/reference/unity/prefab/Environmental_Light.

[8] S. Wehrwein, K. Bala, and N. Snavely, "Shadow detection and sun direction in photo collections," in *2015 International Conference on 3D Vision*, Oct. 2015, pp. 460–468. DOI: 10.1109/3DV.2015.58.

[9] R. Xia, Y. Dong, P. Peers, and X. Tong, "Recovering shape and spatially-varying surface reflectance under unknown illumination," *ACM Transactions on Graphics*, vol. 35, no. 6, Dec. 2016. DOI: https://doi.org/10.1145/2980179.2980248.

[10] C. Hazirbas, L. Leal-Taix, and D. Cremers, "Deep depth from focus," in *Arxiv preprint arXiv:1704.01085*, Apr. 2017.

[11] D. Samaras and D. Metaxas, "Coupled lighting direction and shape estimation from single images," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, 1999, 868–874 vol.2. DOI: 10.1109/ICCV.1999.790313.

[12] M. Aittala, "Inverse lighting and photorealistic rendering for augmented reality," *The Visual Computer*, vol. 26, no. 6, pp. 669–678, Jun. 2010, ISSN: 1432-2315. DOI: 10.1007/s00371-010-0501-7. [Online]. Available: https://doi.org/10.1007/s00371-010-0501-7.

[13] Y. Hold-Geoffroy, K. Sunkavalli, S. Hadap, E. Gambaretto, and J.-F. Lalonde, "Deep outdoor illumination estimation," in *IEEE International Conference on Computer Vision and Pattern Recognition*, 2017.

[14] M.-A. Gardner, K. Sunkavalli, E. Yumer, X. Shen, E. Gambaretto, C. Gagne, and J.-F. Lalonde, "Learning to predict indoor illumination from a single image," *ACM Trans. Graph.*, vol. 36, no. 6, 176:1–176:14, Nov. 2017, ISSN: 0730-0301. DOI: 10.1145/3130800.3130891. [Online]. Available: http://doi.acm.org/10.1145/3130800.3130891.

[15] W. Luo, A. G. Schwing, and R. Urtasun, "Efficient deep learning for stereo matching," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, pp. 5695–5703. DOI: 10.1109/CVPR.2016.614.

[16] J. J. Lim, H. Pirsiavash, and A. Torralba, "Parsing ikea objects: fine pose estimation," *ICCV*, 2013.

[17] P. Nillius and J. O. Eklundh, "Automatic estimation of the projected light source direction," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, 2001. DOI: 10.1109/CVPR.2001.990650.

[18] T. Okabe, I. Sato, and Y. Sato, "Spherical harmonics vs. haar wavelets: Basis for recovering illumination from cast shadows," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, vol. 1, Jun. 2004. DOI: 10.1109/CVPR.2004.1315013.

[19] D. Eigen and R. Fergus, "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture," in *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015, pp. 2650–2658. DOI: 10.1109/ICCV.2015.304.

[20] K. Rematas, T. Ritschel, M. Fritz, E. Gavves, and T. Tuytelaars, "Deep reflectance maps," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016. [Online]. Available: https://ivi.fnwi.uva.nl/isis/publications/2016/RematasCVPR2016.

[21] K. Khoshelham and E. O. Elberink, "Accuracy and resolution of kinect depth data for indoor mapping applications," in *Sensors 2012, 12, 14371454. 2013*, p. 8238.

[22] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, Apr. 2004, ISSN: 1057-7149. DOI: 10.1109/TIP.2003.819861.

[23] (). Principled bsdf shader, [Online]. Available: https://docs.blender.org/manual/en/dev/render/cycles/nodes/types/shaders/principled.html.

[24] J. Zhang and J.-F. Lalonde, *ArXiv preprint arXiv:1703.10200*, 2017.

[25] M. Varma and A. Zisserman, "Estimating illumination direction from textured images," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, vol. 1, Jun. 2004, DOI: 10.1109/CVPR.2004.1315030.

[26] S. Song, S. P. Lichtenberg, and J. Xiao, "Sun rgb-d: A rgb-d scene understanding benchmark suite.," in *CVPR*, vol. 5, 2015, p. 6.

[27] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[28] P. S. Rajpura, M. Goyal, H. Bojinov, and R. S. Hegde, "Dataset augmentation with synthetic images improves semantic segmentation," *CoRR*, vol. abs/1709.00849, 2017. arXiv: 1709.00849. [Online]. Available: http://arxiv.org/abs/1709.00849.

[29] T. Narihira, M. Maire, and S. X. Yu, "Direct intrinsics: Learning albedo-shading decomposition by convolutional regression," *CoRR*, vol. abs/1512.02311, 2015. arXiv: 1512.02311. [Online]. Available: http://arxiv.org/abs/1512.02311.

[30] *Google street view api*, https://developers.google.com/maps/documentation/streetview/intro, Accessed: 30-05-2018.

[31] D. Marnerides, T. Bashford-Rogers, J. Hatchett, and K. Debattista, "Expandnet: a deep convolutional neural network for high dynamic range expansion from low dynamic range content," *ArXiv e-prints*, Mar. 2018. arXiv: 1803.02266 [cs.CV].

[32] J. McCormac, A. Handa, S. Leutenegger, and A. J.Davison, "Scenenet rgb-d: 5m photorealistic images of synthetic indoor trajectories with ground truth," 2016.

[33] P. Isola, J. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," *CoRR*, vol. abs/1611.07004, 2016. arXiv: 1611.07004. [Online]. Available: http://arxiv.org/abs/1611.07004.

[34] A. X. Chang, T. A. Funkhouser, L. J. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, "Shapenet: An information-rich 3d model repository," *CoRR*, vol. abs/1512.03012, 2015. arXiv: 1512.03012. [Online]. Available: http://arxiv.org/abs/1512.03012.

[35] S. Georgoulis, K. Rematas, T. Ritschel, M. Fritz, T. Tuytelaars, and L. Van Gool, "What is around the camera?" In *The IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017.

[36] M.-A. Gardner, K. Sunkavalli, E. Yumer, X. Shen, E. Gambaretto, C. Gagné, and J.-F. Lalonde, "Learning to predict indoor illumination from a single image," *ACM Transactions on Graphics (SIGGRAPH Asia)*, vol. 9, no. 4, 2017.

[37] R. Yi, C. Zhu, P. Tan, and S. Lin, "Faces as lighting probes via unsupervised deep highlight extraction," *CoRR*, vol. abs/1803.06340, 2018. arXiv: 1803.06340. [Online]. Available: http://arxiv.org/abs/1803.06340.

[38] R. Ng, M. Levoy, M. Brédif, G. Duval, M. Horowitz, and P. Hanrahan, "Light field photography with a hand-held plenoptic camera," *Computer Science Technical Report CSTR*, vol. 2, no. 11, pp. 1–11, 2005.

# Appendix A

# Demo Application

To demonstrate the practical capability of the model, we have produced a piece of software with an accompanying GUI that represents a suggested use case. In the application, a user is able to load and view a video file of their choice and attempt to find an environment map of the scene. This is achieved by selecting an object in the video to use as a light probe. Provided the object is sufficiently clear, and meets all of the requirement for good results mentioned in the work, the frames will be fed into the trained neural network and an output written to a file. The output is in a standard format to be loaded into 3D rendering software so that the user can then use it in combination with camera tracking to superimpose new objects.
The application is built in python and relies on the OpenCV image processing framework as well as Tensorflow. Image selection is performed with a single click, which triggers a flood fill algorithm at the selected pixel in LAB color space. The chosen region is then converted to a bounding box, and a snapshot at that frame is stored. The object in the bounding box is tracked with a standard 'Kernalised Correlation Filter', provided as part of OpenCV. When it is detected that the object has moved far enough in the x dimension, another snapshot is taken. Both snapshots are shown to the user for confirmation that they are suitable, before being fed into the trained model. We believe this example gives a good indication of how we imagine a system like ours could be deployed.