NOVEL SEMI-AUTOMATIC METHOD TO OPTIMIZE MULTI-LAMP HIGH FLUX SOLAR SIMULATORS

An Undergraduate Research Scholars Thesis

by

ARSHAD MOHAMED ALI, MOHAMMED HASSAN, and SAFEER HAFEEZ

Submitted to the Undergraduate Research Scholars program at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by Research Advisors: Dr. Konstantinos Kakosimos
Dr. Mohammed Al-Hashimi

May 2019

Major: Chemical Engineering

TABLE OF CONTENTS

ABSTRACT

Novel Semi-Automatic Method to Optimize Multi-Lamp High Flux Solar Simulators

Arshad Mohamed Ali, Mohammed Hassan, and Safeer Hafeez
Department of Chemical Engineering
Texas A&M University at Qatar

Research Advisor: Dr. Konstantinos Kakosimos
Department of Chemical Engineering
Texas A&M University at Qatar

Research Advisor: Dr. Mohammed Al-Hashimi
Department of Science
Texas A&M University at Qatar

For multi-lamp high flux solar simulators (HFSS), it is often difficult to obtain a required flux distribution by manipulating the lamp position of multiple lamps at once. Each lamp has three degree of freedom. Thus manual optimization can be tedious for human operators. Thus, this project aims to create a semi-automatic method to determine the optimal location of the lamps to give the required flux distribution. A convolutional neural network is used to develop a mathematical model that performs the above function.

At the same time, an automated method to collect data from the HFSS was devised. Furthermore, an in-house algorithm to characterize the irradiance was developed. Since large amount of data was required, an optical simulator called TracePro was used to generate the data for training as well as validation. This project serves as proof of concept of using machine learning to optimize HFSS. In the long term, the proposed methodology is expected to facilitate initial deployment of the HFSS. It will also assist on the dynamic control of reactor conditions i.e. emulating variable overcast or daily sunlight variability.

DEDICATION

We all would like to dedicate this thesis to our families without whose support this work would not be possible.

ACKNOWLEDGEMENTS

We thank our advisors Dr. Konstantinos Kakosimos for who provided us with all the aid and guidance in this research.

We thank our friend, Bassam Khalil for his assistance in the lab with preparing the experimental setup and data collection.

We would also like to thank all of our professors and the staff of the chemical engineering department for their support throughout our educational journey in Texas A&M University at Qatar.

This research was supported by the Undergraduate Research Experience Program award [UREP 22-024-1-002] from the Qatar National Research Fund (a member of the Qatar Foundation).

NOMENCLATURE

HFSS	High Flux Solar Simulator
TAMUQ	Texas A&M University at Qatar
GS	Grey Scale
CNN	Convolutional Neural Network
CCD	Charge Coupled Device
RMSE	Root Mean Squared Error

CHAPTER I

INTRODUCTION

The rapid industrial growth of the world coupled with high population growth has led to a rise in greenhouse gas emissions. Consequently, one of the challenges facing engineers is using cleaner sources of energies for industries and the like. One such source of energy is solar energy. Solar energy is abundantly available and is cleaner than conventional energy sources such as fossil fuels. However, solar energy has its own associated challenges. Firstly, the energy is of low density (1 kW/m^2) and so must be collected so that it can be used for high energy applications. Furthermore, weather conditions interfere with solar energy availability. As a result, testing concentrators, photo catalysts, etc. can be a time-consuming endeavor in places with erratic weather. Thus, solar simulators are an important tool to overcome these limitations for conducting solar energy research. [1,4,5]

Solar simulators typically consist of multiple light sources coupled with ellipsoidal or parabolic reflectors. These reflectors concentrate light to a predefined focal plane. The advantage of solar simulators is you can concentrate the light to regions of different areas on the focal plane. In this way, one can control the incident flux on equipment such as reactors placed on the focal plane. The flux output of solar simulators can range from 30 to 100 kW/m^2. [5] Furthermore, solar simulators do not rely on weather conditions giving researchers more flexibility.

A major use of solar simulators is to test photo catalysts such as TiO_2. Such catalysts can be used for applications such as waste-water treatment.[6] Another application of solar simulators is for advanced aging of materials. In accelerated aging, information from experiments at high levels of accelerating variables such as temperature or radiation is used to obtain long term

performance of material at lower levels of accelerating variables.[7] Finally, due to the energy output of the simulators, high temperatures can also be attained. Thus, such simulators are also apt to for high temperature research applications such as catalyst development.

To reach high flux intensities, solar simulators often have multiple source-reflector pairs. The HFSS at TAMUQ consists of seven Xe-arc light sources coupled with ellipsoidal reflectors. The light sources have 49 kW electrical power and are capable of delivering a minimum of 20 kW thermal power to a focal area in the range of 2-100 cm diameter. To obtain different peak fluxes and different focal areas, the sources are to be moved in the X, Y and Z direction which means three degrees of freedom per source. As a result, this can be a tedious process especially when all seven lamps are used.

Thus, this project aims to create a semi-automatic method to determine the optimal location of the lamps to give the required flux and illuminated region. The use of convolutional neural networks is proposed to develop a mathematical model that outputs the lamp positions based on a required flux map. However, to develop a neural network, large amount of data is required. As a proof of concept, data is generated from the optical simulator TracePro and used to train the network. In parallel, an automated method to collect data from the HFSS is devised. The data is treated using an in-house algorithm. The algorithm is based on flux mapping method from literature. This data collection method is developed for future development of this methodology.

The proposed methodology is expected to facilitate initial deployment of high flux solar simulator. It will also assist on the dynamic control of reactor conditions i.e. emulating variable overcast or daily sunlight variability.

Literature Review

A brief literature review on solar simulators and machine learning was conducted. The results of this review is summarized below.

Solar Simulators

Several multi-lamp HFSSs have been built around the world. One of the first large simulators was designed and built in Switzerland by researchers working at the Paul Scherrer Institute (PSI). The design objective of this 10 lamp HFSS was to maximize the amount of radiant energy transferred to the target plane. This was determined by calculating the fraction of electrical power supplied to the lamp that is incident as radiant power on the target surface. The analysis for this design was conducted using a Monte-Carlo ray tracing software to determine the optimal reflector shape for a specific focal length. The target was placed at the second focal point of the ellipsoidal reflector. The researchers determined a maximum transfer efficiency of 34% for the optimal reflector. A mean flux incident on a 6 cm diameter target at the focal point was predicted to be 5.9 MW/m^2 for the 10 lamp array. However, the prototype performance exceeded these predictions, with mean flux values exceeding 6.8 MW/m^2.[8]

A 45 kW, 18 lamp HFSS was recently built collaboratively by researchers from Australian National University (ANU) and Ecole Polytechnique Fédérale de Lausanne (EPFL). Leveque et. al. have documented an experimental and numerical characterization method for this simulator. Raw images of the target obtained using a CCD camera were corrected for dark current, normalized by the exposure time and calibrated with heat flux measurements to create radiative flux maps. The measured peak flux was around 21.7 MW/m^2 for the 18 lamp array. Monte-Carlo ray tracing software was used to obtain results numerically, which were then

calibrated with experimental results. A 4.2% difference between the results has been reported, with lamp efficiency of 39.4%.[9]

Javad et. al. developed and characterized an adjustable flux solar simulator in Texas A&M at Qatar, which was the first of its kind in the Middle East. A 7 kW xenon short arc lamp coupled with a truncated ellipsoid reflector was used as the light source. The flux mapping method was used to evaluate the performance of the HFSS with the help of a CCD camera and a heat flux gage. The performance criteria chosen included flux distribution, temporal instability, peak flux and conversion efficiency, among others. The input current to the lamp was adjusted in the range of 113-153 A to obtain the minimum and maximum peak flux output, yielding different flux distributions. A peak flux of 3.583 MW/m^2 was reported at an input current of 153 A with conversion efficiency of 47%. The simulator was reported to be capable of adjusting its peak flux in the range of 2.074-3.583 MW/m^2.[3] A list of several other multi-lamp HFSSs is shown in Table 1.[1,10,11]

Table 1: Summary of HFSS around the World

Developer	Radiative Power (kW)	Efficiency	No. of lamps	Peak Flux (kW/m^2)	Target (mm)
Niigata University (TIT)	133	25.0%	19	>3000	200
DLR	21	33.3%	10	N.A.	100 cm^2
Minnesota University	9.2	20.2%	7	7300	60
GIT	6	14.3%	7	>6500	40
IET	6.4	22.9%	4	N.A.	300
KTH	19.7 *	23.4%	12	6730*	200
Swinburne University	12	28.6%	7	927	175
JFCC	8	10.7%	20	37.7	4000x70d
Synlight	320	N.A.	149	11000	200x200
***Ray Tracing Estimations**					

Overview of Neural Networks

Convolutional Neural networks are a sub-classification of a broader family of machine learning methods called deep learning. Machine learning is using certain statistical models and methods to have the machine itself develop a mathematical model that relates the inputs and outputs. This is different from traditional programming in the sense, that traditionally users develop a mathematical model to obtain the output from the input. Once an model is obtained from the 'machine learning code', it can be used to guess the output for any input similar to the

input used to develop (train) the algorithm.[12] This concept is better explained using Figure 1 below.

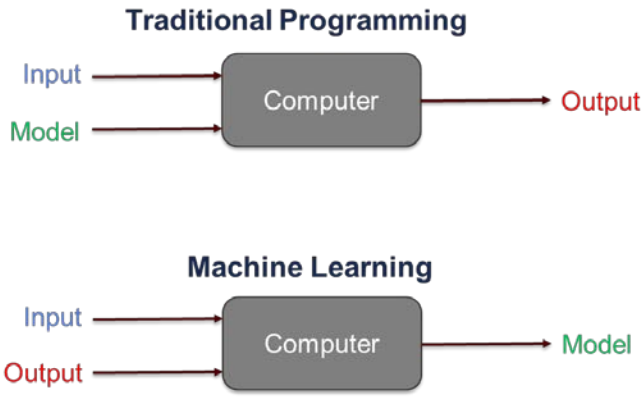


Figure 1: Difference between Traditional Programming & Machine Learning

Thus, machine learning is extremely useful in cases where large number of inputs and outputs exists with often complex relationships. Deriving models for such cases can be tedious and extremely difficult even. However, machine learning simplifies the process and derives an empirical model by using different methods. One such group of methods is called deep learning. Deep learning is used for data sets such as images and flux maps which have features such as circles. The code learns to correlate the presence and characteristics of these features to the input. It can be considered as a pattern recognition software of sorts. [13,14]

One such model within deep learning is the convolutional neural network (CNN) which is used primarily with visual imagery due to ease of use. These models derive a mathematical model by using several layers between the input and output. Each layer takes information from the layer before it and multiplies it by a certain factor called weight and then adds constant called a weight. The process is repeated until the output layer is reached. The goal of machine learning is to derive weights and biases that convert the input to the output. The weights, biases make up the bulk of the empirical mathematical model. [14,15] Figure 2 visually describes what a CNN in a simplistic manner.

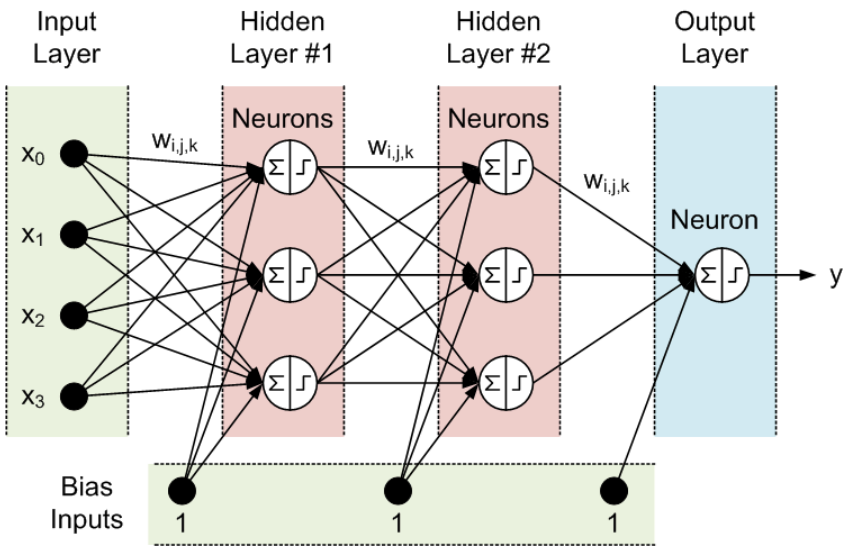


Figure 2: Visual depiction of a CNN [16]

Any neural network can be used for either classification or regression problems. Classification models are used to sort data into pre-defined sets. However, regression models are used to predict data points outside the current set. [17] Figure 3 below illustrates the difference between the two.

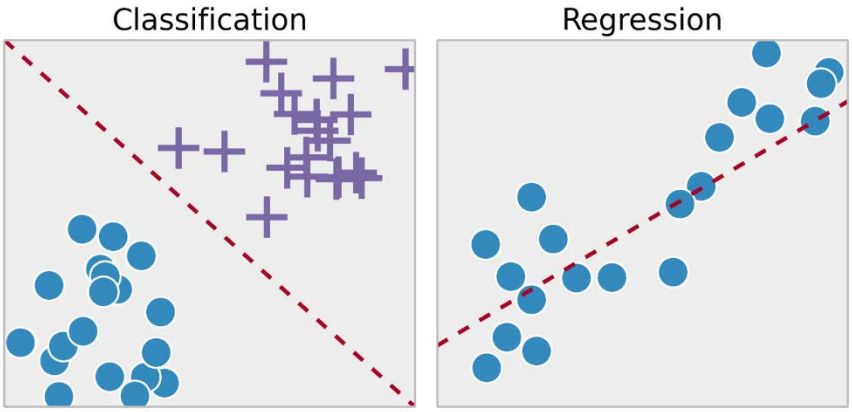


Figure 3: Difference between classification and regression problems [18]

A CNN finds the weights and biases by an iterative approach. It guesses weights and biases and then finds the error between the guessed output and the actual output provided by the user. The goal of any CNN network is to final the global minimum of this error function. This

whole algorithm is called gradient descent. Several approaches exist to find this global minimum. These are described below:[19]

- Stochastic gradient descent: This approach calculates the error for each output and updates the model and then moves to the next example. This model is easy to implement and may be faster for some cases. However, since it updates the weights and biases for every output, it requires huge computing capacity. The process is also noisy as it updates for every data point.[19]
- Batch gradient descent: This approach calculates the error for each output but updates the model when error for all data points have been calculated. One cycle through the training set is called an epoch. This method is less computationally intensive and possesses less noise than the last method. Since this method leads to a more stable convergence, the training may terminate prematurely when it finds a local error minimum rather than a global error minimum.[19,20]
- Mini-batch gradient descent: This approach splits the training data set into multiple subsets. The error for each set is computed and the model is updated after each mini-batch is run. It is the most common form of approach used in deep learning as it is a compromise between the robustness of stochastic descent and the speed/efficiency of batch descent.[19,20]

Before training any CNN, several hyper parameters need to set/tuned. Hyper parameters are parameters that are set before the CNN starts learning. These define the learning process and guide it. These are different from parameters. Parameters are properties learnt during the training such as weights and biases.[21,22] The different hyper parameters that need to be tuned are described below:

- Learning Rate: Learning rate is the amount by which each weight is updated during the training process. It is perhaps the most important hyper-parameter that requires tuning in any machine learning code. It controls the speed at which the model learns. A high learning rate improves learning speed but may not reach the optimal solution, i.e. Global minimum. A small learning rate reduces learning rate but will eventually reach the optimal solution (if rate is low enough). Thus, this parameter must be carefully set to obtain the optimal solution without taxing the computational resources. [23]
- Mini-Batch size: As explained earlier, mini-batch size is the size of the subsets the main data set is divided into. This parameter controls the speed of the learning process primarily with a considerable effect on the accuracy as well. [19]
- Epochs: One sweep through the whole data set is called an epoch.[20]
- Hidden layers: These are layers that are used to treat the data and obtain the weights and biases. Basically, these are layers other than the input and output layer. The main layer used here is a convolutional layer which calculates the weights and biases for the model. Other layers include the pooling layer which reduces the spatial size of the data from a previous layer to reduce computation speed and control overfitting. [20,24]
- Filter size: Each layer scans the input image in the form of a moving matrix called a filter. All the values inside the filter are multiplied by a weight matrix followed by addition of the bias. The size of this filter is another crucial and important hyper parameter. If the filter is too big it may miss may minute features in the image and see more general features. Thus, filter sizes are kept small to better see the features. [24]
- Filter number: The number of filters within a layer are neurons that connect to the same area on the previous layer. Having more filters can allow for a better fit for the model.

Since many hyper parameters exist to control overfitting, large filter number are chosen to obtain the correct model. Even if the number of filters are larger than the optimal value, it does not negatively affect training. However, if it is lower than the optimal value, it can impair the training process.[20,24]

- Stride: The stride is the rate at which the filter moves across the image. Typically small strides are used to capture more of the features.[24]
- Momentum: When weights are updated after an epoch, the weights are often based on the exponential weighted average of the prior updates. The weightage of the prior updates is called momentum. It is designed to speed up learning. It helps direct the learning in one direction.[23]
- Weight decay regularization coefficient: A regularization coefficient is added to control overfitting. This factor controls the learning capacity of the software. A common method is the L2 regularization.[20]

There exist some other hyper-parameters such as sparsity of activation, neuron non-linearity and others within specific hidden layers. However, these are not typically changed. There does not exist any one certain way to find the values. These parameters are typically found by using trial and error. However, there exist heuristics from experimentation or experience that limit the interval of search. The methodology adopted in this project is described in the methodology section.

CHAPTER II

SETUP & METHODOLOGY

High Flux Solar Simulator Setup

The HFSS consists of seven 6000W Xe short arc lamps manufactured by Osram. The setup is designed to provide energy in excess of 4000 kW/m^2 on 20x20 cm area. The radiation is concentrated on a 60mm diameter target at a focal distance of 3m, achieving temperatures in excess of 2000°C. The Xe lamps were chosen as their emission spectrum closely resembled solar spectrum. The spectrum emitted by these lamps is also more stable and continuous despite voltage variations making them easier to work with. [1] These lamps are placed within highly polished ellipsoidal reflectors coated with Al and SiO_2 for reflectance and protection. Each lamp-reflector combination is equipped with its own fan for cooling air and 3D maneuvering mechanism for lamp positioning. The whole 'Sun in the Box' setup is equipped with an air handling unit and an air duct system to cool the unit when more than four lamps were active. The setup also had several sensors installed into it as safety precautions to protect users from hazards such as exposure to high flux, temperature, etc. For example, the system cannot be operated unless the doors to the HFSS are closed and locked. The system also had an emergency shutdown when system temperature crossed a pre-specified threshold. The exterior of the HFSS solar simulator as created in SolidWorks is shown in Figure 4 below. A picture of the seven lamps can be seen in Figure 5 below.

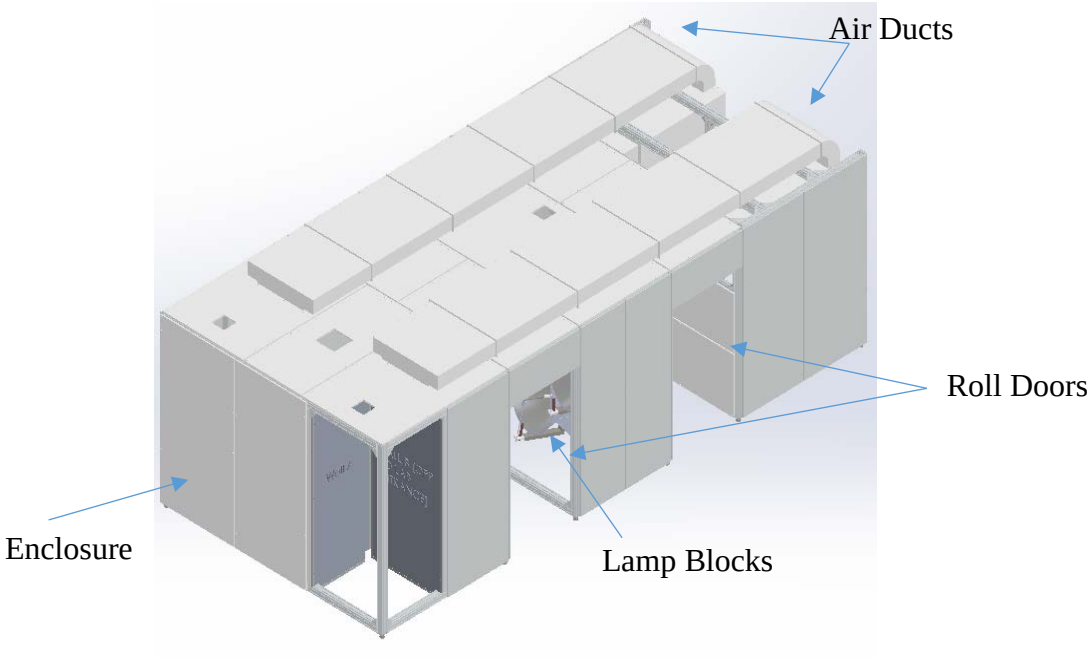


Figure 4: Isometric View of Solar Simulator

Figure 5: Picture of the HFSS interior

A Lambertian target (25x25 cm) is mounted on a XYZ slider in front of the seven lamps. A Lambertian target was used as it ensures light is reflected equally in all directions. Thus, the target appears equally bright irrespective of viewing angle. As a result, the camera used to monitor the target can be placed at any angle. The XYZ slider movement has an accuracy of up to 0.1mm in the all three directions. This allowed for precise control of the location of the target

to generate the required data for flux characterization. A heat flux gage (radiometer) was located at the center of the lambertian target. The gage measured incident flux by measuring the output voltage of the gage. This output voltage was used to determine the incident flux using a calibration curve. The calibration was performed by the gage manufacturers at the start of the experiment. The flux was calibrated to reliably measure up to 4850 kW/m^2. The Lambertian target and flux gage were both water cooled. Thermocouples were placed on the back of the Lambertian target as well as water inlet and outlet. The thermocouples were placed to ensure that the target does not overheat and that the cooling water is at the desired temperature. A sketch of the target by Jawad et. al is shown in Figure 6 below.

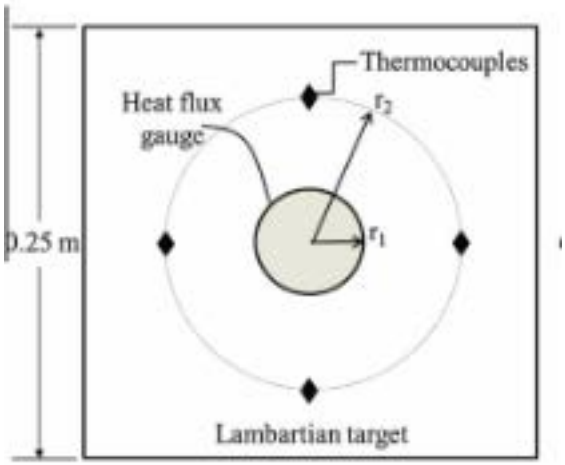


Figure 6: Schematic illustration of lambertian target [3]

An 8 bit charged-coupled device (CCD) camera was used capture images of the Lambertian target. The CCD camera had a neutral density filter to protect the camera from the high flux reflected at it. The CCD camera was placed at an angle and thus the images required perspective correction as described in methodology below.

Data collection hardware was installed in the space between the grey walls shown in Figure 4. The hardware consisted of DAQ modules to collect temperature and flux readings. A

Galil controller was installed to move the Lambertian target. All of these modules interfaced with a LabVIEW program described briefly in the methodology.

Methodology

The methodology is broadly divided into three categories. These are automated acquisition of large amount of data, flux characterization, neural network training/ application. These are described in more detail below.

Data Acquisition

Flux characterization and neural network training requires the collection of a large amount of specific data. For flux characterization, flux values across the target on different axes along with 30 or more images are needed. This collection can be tedious as the incident flux drops in an exponential manner as one moves away from the target. Thus, close to the target, the flux measurements are collected at small increments of distances (such as 0.25mm) to capture as much data as possible as described later on. This can lead to collection of 80 data points or more in one axis alone. On the other hand, neural network training requires the above mentioned flux data and corresponding images at several different lamp positions. This process is highly tedious and prone to error for human operators and can be forbiddingly time consuming when collecting data for multiple lamps. Thus, an automated LabVIEW program (VI) was developed to collect the flux data, images and move the target without the need for human oversight. The overall program structure was based on state machine architecture with tab control for different states. It was combined with the use of shared variables and subVIs to modularize the program. The user can program the distance intervals the slider has to move from the default position in x, y and z direction along with the number of data sets to be taken in each interval. Sequential structure

programming was used to achieve this. Thus, the system can automatically collect all the data associated with one lamp position.

The flux was collected in three intervals in the x and y axis. For the first 1 cm from the center in the positive x direction, the flux measurements were collected every 0.25 mm. For the next 2 cm, the flux was collected every 0.5 mm. For the next 3 cm, data was collected at every 2 mm. For the remaining length, flux measurements were collected every 2 mm. The same method was applied to the negative x direction and both positive and negative y direction. X and Y axis were both measured to check for spatial non-uniformity and allow for redundant verification for the characterization. This resembled the work done by Jawad et. al. [3] This particular method of data collection was chosen as flux dropped rapidly as one moved away at the center. The drop was steepest closest to the center. Thirty flux measurements at a rate of one sample per second were collected at every target position and averaged. Thirty data points were chosen to minimize standard deviation. Similarly, 40 images were taken at the following exposures and averaged.

- 1/120 s
- 1/1000 s
- 1/1250 s
- 1/2000 s

Flux Characterization

Characterization for a solar simulator setup includes determining parameters such as temporal instability, spatial non-uniformity, peak flux and flux density distribution. As part of this project, peak flux and flux density distribution were determined as they were considered most pertinent to the project. However, the data collected can be used to find other parameters if the need arises. Characterization was done using the flux mapping method. This method allows

for determination of the flux distribution without direct measurement with high accuracy. The method correlates the radiative incident flux values to the grayscale values of the pictures taken of the target. The grayscale value is a representation of the brightness of a pixel in a grayscale image. After characterization, each pixel value corresponds to a specific incident radiation flux.

Normally, one or multiple lamps can be characterized at once. For our testing of the method, only the central lamp of the setup was used.

Image Processing

An 8 bit CCD camera is used to collect pictures of the target. Multiple pictures at different exposure times were collected. All the subsequent image treatment is done on MATLAB. The images at each exposure are first turned into matrices to be averaged and remove some of the noise caused by the camera. Additional image processing involves implementing a projective transformation algorithm to reorient the distorted images taken by the camera. This is done to eliminate the effect of taking the pictures at an angle and give the correct shape of the target. The final averaged image of each exposure is shown in Figure 7.

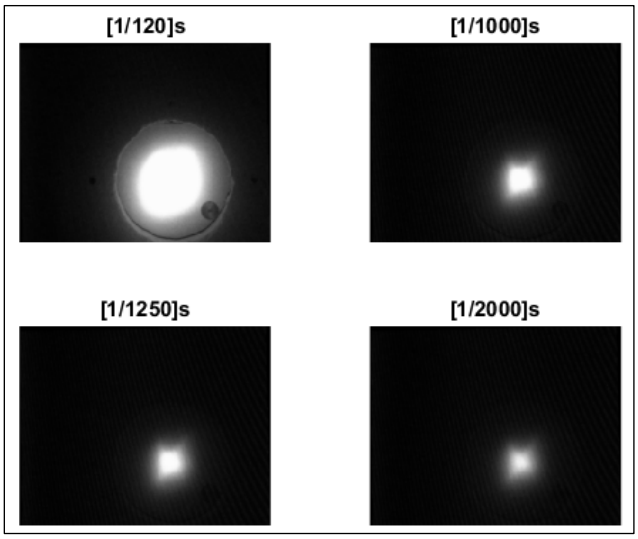


Figure 7: Images at different exposures

The information in the images is increased through taking the weighted average of the different exposure images based on exposure time using equation 1 shown below.

$$\text{Normalized image}(i,j) = \sum_{n=1}^{N} \frac{P_n(i,j)}{t_{expo,n}} \quad (1)$$

Where $P_n(i, j)$ = GS value of the pixel (i,j) in the image with exposure n

t_{expo} = exposure time.

The result is a single image with information from all exposures as shown in Figure 8.

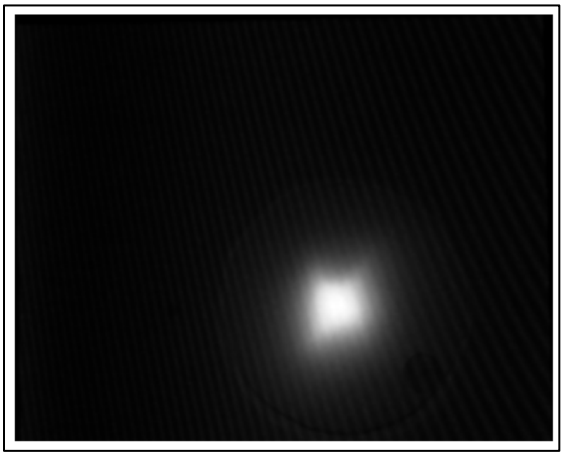


Figure 8: Weighted average taken from four exposures

The flux gauge had a diameter of 2.525 cm. Thus, greyscale values within a square of similar area were averaged to correlate to the flux values. This square was moved at the same rate the flux gage was moved. Figure 9 represents an example of the area that would be averaged to determine the pixel value at the origin.

Figure 9: Method for averaging greyscale values corresponding to different flux measurements

To obtain a characterization curve, the greyscale values were plotted against the incident flux and a straight line of best fit was used to obtain the curve. This correlation is then used in MATLAB code to create a flux map of the incident flux on the target. Thus, the flux map presents the distribution of the incident flux on the complete target. The advantage of this method is that a few measurements in one or two axes can define the flux across the whole target. Furthermore, images of the target can be used as an indication of the incident flux without direct measurement of flux on the target.

TracePro Simulation

A Monte-Carlo ray tracing software called TracePro was used in order to aid in the development of the machine learning algorithm. Its main purpose was to provide a proof of concept that could later be applied on the actual setup using experimentally obtained data. The use of this software allowed for rapid data acquisition, when compared to experimentally obtained data. This meant that a large amount of data could be generated rapidly which could then be used in the training and testing phases in the development of the neural network model.

The first step in using TracePro is developing and defining the geometry of the setup. This includes defining the geometry of reflectors, the lamps array and the target. This was done

on SolidWorks, using the exact dimensions of the actual setup. An image of the setup designed on SolidWorks and imported into TracePro is shown in Figures 10 and 11.

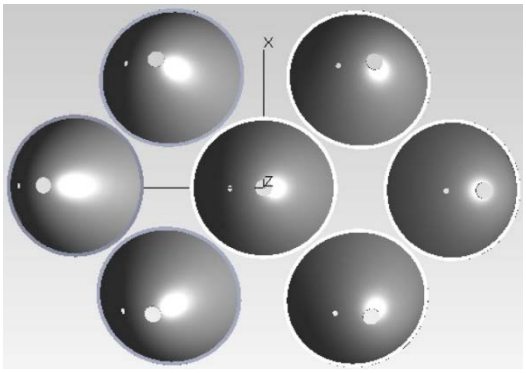


Figure 10: Front view of lamp array

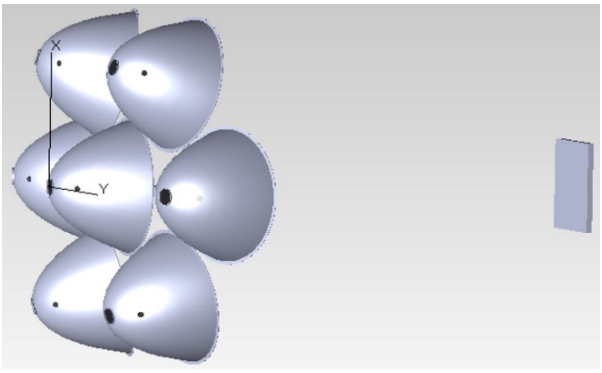


Figure 11: Side view of lamp array

The setup modelled in TracePro is exactly the same as the actual setup in all ways except one-the light source. Initial effort was made to model the light source as close to the real one as possible. An arc was also modeled between the anode and cathode of the lamp to simulate the light generated when the light is switched on based on brilliance data provided by OSRAM. [25,26] The lamp depiction provided by the manufacturer is shown in Figure 12 while the lamp created in Solidworks is shown in Figure 13.

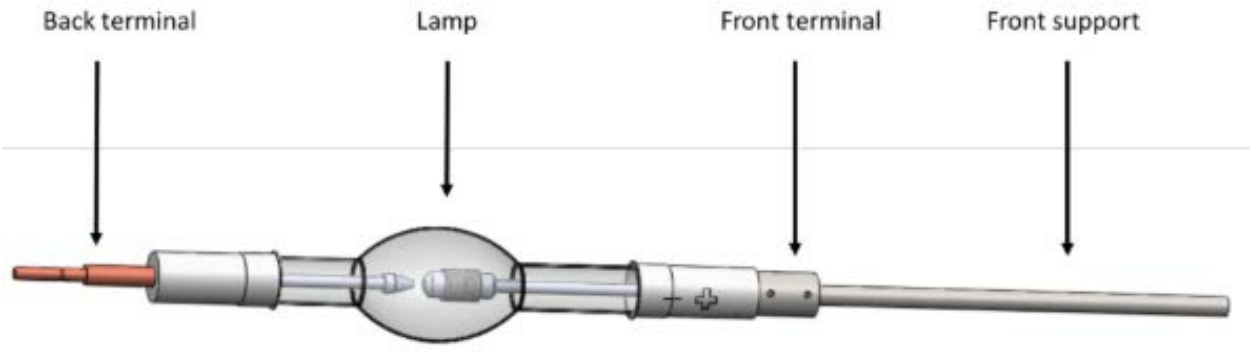


Figure 12: Depiction of lamp geometry provided by OSRAM[25]

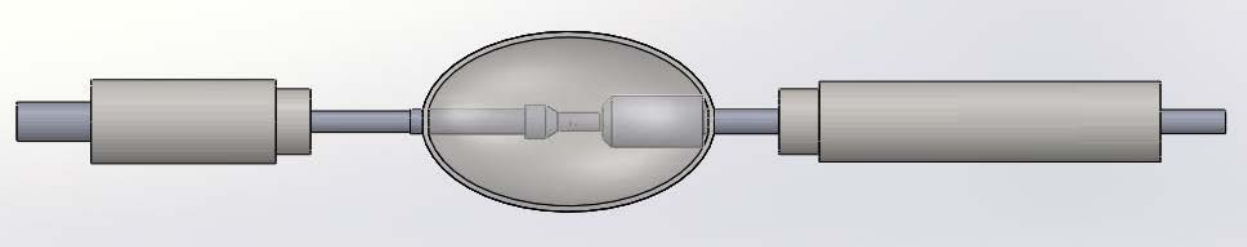


Figure 13: Lamp geometry created on SolidWorks

As shown in Figure 12 & 13, the geometry was complicated which slowed down TracePro simulations. Since modelling the lamp is outside the scope of the research, a simpler approach was taken. In order to design the source with reasonable accuracy, literature review was conducted on arc lamps. A detailed product description published by the lamp manufacturer (OSRAM) was used to make the assumptions about the light source. As seen in Figure 14, the first 4 layers with highest brilliance (200, 150, 100 & 75 kcd/cm^2) along with the plasma ball emit approximately 94% of the light that is produced in the arc. [26] The plasma ball alone accounted for 50% of the light that is produced in the arc. [3] These layers can be approximated to form a sphere with a diameter of approximately 5 mm. Therefore, keeping this approximation in mind along with the actual dimensions of the setup, the light source is modelled as a sphere with a diameter of 5 mm. The lamp was placed at the focal point which is 9 cm from the base of the reflectors. The focal point was taken as the origin for data generation purposes. As per product specifications, the arc lamp has a flux of 6 kW and produces light with a wavelength of 500 nm. The radiation from the source was also assumed to be lambertian in nature.

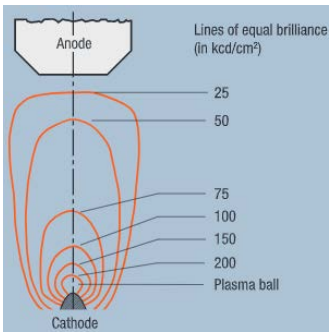


Figure 14: Brilliance distribution in the arc of a xenon lamp [26]

Finally, the reflector and target surface properties had to be defined to be able to start simulations. The reflectors were defined as perfect reflectors with specular reflectance of 0.92. The target was assumed to be lambertian and also a perfect absorber with dimensions of 25x25 cm. It should be noted that the dimensions of the target were skewed very slightly when the geometry was imported from SolidWorks into TracePro. Nevertheless, this did not affect the results.

Simulations can now be run to obtain peak flux and flux distribution plots at the target at varying lamp source positions. It should be noted that simulations were run with only one lamp activated in order to reduce the simulation time as a large number of simulations had to be run to obtain the data required for neural network training. Thus, macros were programmed within TracePro using the language ‘Scheme’ to obtain data automatically. The lamp source was moved relative to the origin in 0.25 mm intervals for 10 mm. The directions the lamp was moved in is shown in Figure 15 below. The lamp was moved in positive and negative X,Y and Z directions as well as all binary combinations of the three axes. The diagonal movements were done at an angle of 45 degrees. A large data set (721 flux distributions) was taken to allow for a good variety of data available for the neural network training.

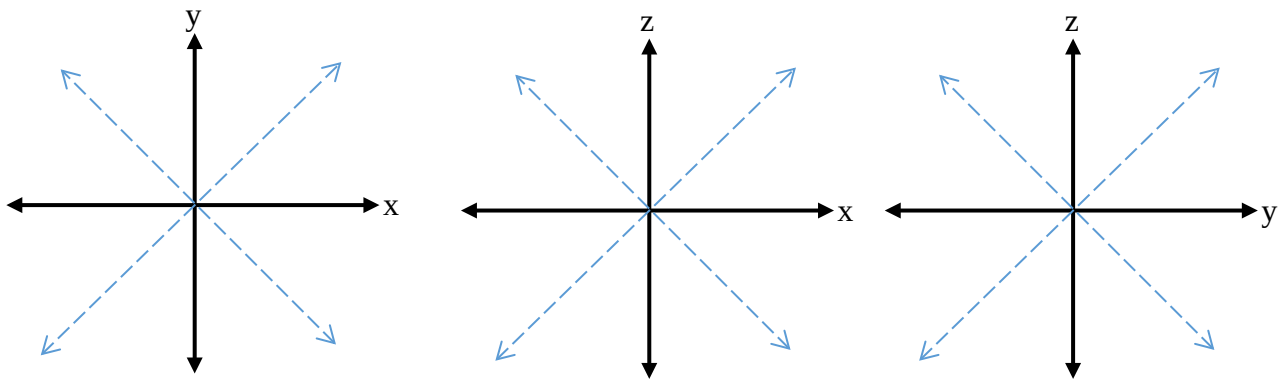


Figure 15: Lamp movement directions

The plots were obtained by plotting irradiance against position and smoothing the resulting image and setting the number of pixels to 20 on the plot display options.

Convolutional Neural Networks

As part of developing the machine learning code, several key decisions had to be made, from type of machine learning method used to the gradient descent method to the value of various hyper parameters. The methodology behind these decisions is explained in more detail below.

First, deep learning neural networks were used as they were most apt working with data with features (corners, circles, etc.) such as the flux maps used in this project. Within deep learning the CNN was chosen due to their wide use with visual imagery. Typically, CNN models are used for classification purposes. However, the goal of this project was regression, since coordinates for any flux map not in the collected data set had to be predicted. Thus, it was decided to modify a typical CNN with a regression layer at the end.

The mini-batch gradient descent method was used due to it wide use within the deep learning community as well as its distinct advantages over other methods outlined in the literature review. The hyper parameters were either determined by trial and error or set from heuristics obtained from literature. The trial and error was done by looking at the training mini-

batch loss and root mean squared error (RMSE). The mini-batch loss is the error across each mini-batch. RMSE is the overall representation of the fit of a regression line fit to the data and thus is a measure of the overall error function. [27] Thus, to obtain the correct model, these variables were monitored to minimize them. At the same time, the computational time was also monitored and attempts to minimize it were also made. The effect of the hyper parameters was considered to be independent. [20] The rationale behind choosing intervals for trial and error or the value of each hyper-parameter is described briefly below.

The *learning rate* can either be the same throughout the process or it can drop slowly throughout the process ('adaptive'). This allows to use a larger running rate at the start of training and decrease it as training progresses to fine tune the final model. This decreases computational time as at the start of the training process, errors are large, thus larger learning rates are required. If the adaptive method is used, the initial learning rate, learning rate drop factor and drop period need to be configured. The initial learning rate is the most important factor because if it is too large, the training will never converge and if it is too small, it will converge after a long time. Thus, this parameter was determined through trial and error. The typical range of this factor is 10^{-6} to 1. The drop factor is by what multiple is the learning rate dropped after a specified epoch drop period. These factors are not as crucial since if the learning drop rate drops too fast it will slow down the process but it will not give wrong values. Thus, this values were left to the user's discretion to modify to speed up the learning process. The user modified these parameters depending on the system being used as well as number of epochs and data set size. A powerful system would allow for a larger drop rate at smaller drop intervals and vice versa. [20,23]

The *mini-batch size* was set to be a fraction of the total number of data used for training. This would allow for integer number of batches with equal sizes. Of the 721 points, 620 points were randomly used for each run. 620 points were chosen to reduce the computing load. The remaining 101 points were used after the final training for additional accuracy metric described below. All the 721 points were randomized to remove any biases before feeding it to the machine learning code. Since only 620 points were used, the testable mini-batch sizes were few. The batch sizes tested were 20, 31, 62, 124, and 155. Batch sizes smaller than 20 were not tested as that would slow down the computation excessively. Batch sizes above 155 would transform the system into a batch gradient descent system.[19] The number of iterations within each epoch is defined as the total batch dived by the number of mini batches. Thus, minibatch size also impacted number of iterations within each epoch.

The main *hidden layer* used in the code was convolutional layer and the ReLU layer. The convolutional layer contains the neurons and calculates the weights and biases. It is the most important layer. The *number of layers* was manipulated as well as the filter size and number of filters within the layers. Four or more layers were used based on heuristics. The *filter sizes* of 2, 3, 4 and 5 were tried since small filters generally give better results. The *number of filters* were varied from 5 to 20. A stride size of 1 was used as it is the recommended value. [24]

The ReLU layer is a linear activation that outputs the input if it is positive, else it outputs a zero. As a result, it is frequently used between convolutional layers especially in CNNs among many other networks as it also overcomes the vanishing gradient problem in many models. [28] Thus, ReLU layer was used after each convolutional layer to improve performance. Zero-padding was not used since the flux maps are large and there are no features near the edge of the

maps. Thus, even if values near the edge are not read, it would not affect the training process. Other layers such as pooling are being phased out and thus were not used. [29]

A *momentum* value of 0.9 was used as it is the most common value used for training. [14] Furthermore, the effect of this factor is less compared to the other factors and hence was not determined through trial and error. *L2 regularization* was used to control the overfitting with the MATLAB default value of 0.0001. [27] This value was not to be changed unless overfitting was found at the end of the training process.

Once all optimal hyper parameters were obtained, the model was allowed to train for a larger epoch period of 100 epochs on the 620 data points. The final network was used to predict the coordinates of the remaining 101 images. The predicted and test coordinates were compared with two accuracy metrics. These were the A_{sum} and $A_{threshold}$. The equations for the first is shown in equation 2 below.

$$A_{Sum} = [\sum abs(x_{predict} - x_{test}), \sum abs(y_{predict} - y_{test}), \sum abs(z_{predict} - z_{test})] \quad (2)$$

Where $x_{predict}$, $y_{predict}$ and $z_{predict}$ = coordinates predicted by the train network

x_{test}, y_{test} and z_{test} = coordinates obtained from TracePro for testing.

The second accuracy was based on an acceptable threshold for error in each coordinate. The $A_{threshold}$ was a three column vector (x, y, z accuracies) of the percentage of predictions whose error was below an acceptable threshold. Since this is the first attempt at machine learning, the acceptable threshold was set at a moderate value of 1.

Finally, for further validation, three predicted coordinates from the CNN were taken and corresponding flux maps were obtained. Then, the original flux map used to predict the coordinates were compared to the new flux maps. This was done since it was possible that

different X, Y and Z combinations can give very similar flux maps. This was considered as additional validation for the trained network. The error between the two flux maps was computed by subtracting the two flux maps and normalizing it to the largest error present in the map.

CHAPTER III

RESULTS & DISCUSSION

The results of flux characterization process, TracePro simulations and convolutional neural network training are explained below.

Flux Characterization

Data was collected using an automated LabVIEW system. While the system was able to collect the flux data at user-specified intervals and increments, the system could not change the lamp position. The code used to move the lamp motors was locked and thus, needed more time for reprogramming which put it out of the period of this project. Nonetheless, the system collects the data for one lamp position in a time efficient manner. It can reduce the time period of time collection from 3 hours to 1 hour per lamp position by removing the human operator. The data obtained from this program for one lamp position close to the lamp focus was treated using the in-house algorithm. The calibration curve obtained is shown in Figure 16 below.

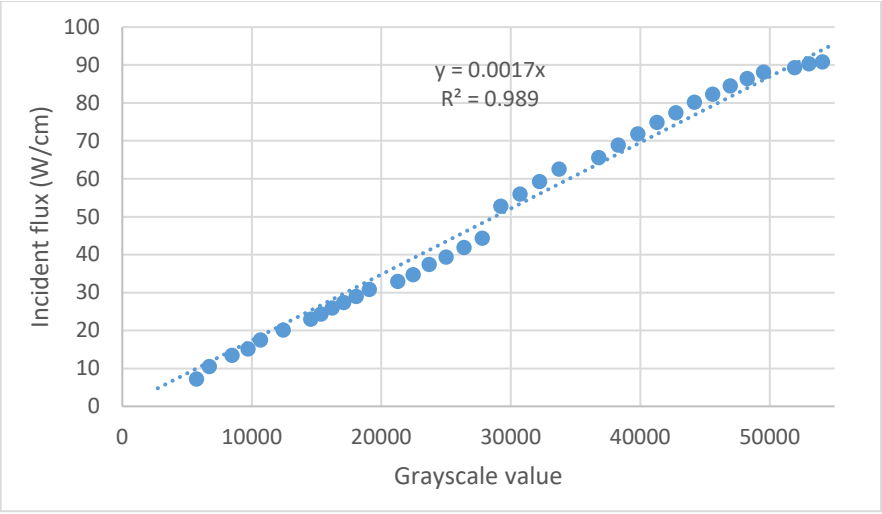


Figure 16: Characterization Curve for one lamp

As seen from Figure 16, the characterization curve obtained is linear with a high R-squared value of 0.99. This matches experiments done earlier by Jawad et. al.[3] Thus, this confirms the accuracy of the in-house algorithm developed to average the greyscale values and create a calibration curve. Figure 17 shows the flux map obtained by using the characterization curve to transform the weighted average greyscale image in Figure 8.

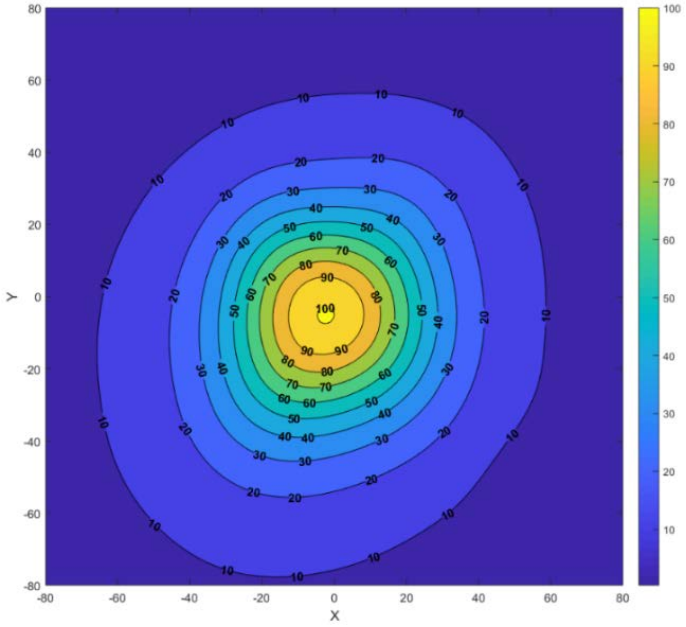


Figure 17: Flux map generated by applying characterization curve to Greyscale Image

The flux map shown in Figure 17 has concentric circle profiles. This result is also expected when the lamp is placed near the focal point of the reflector. Thus, the proposed system to generate flux map/ coordinate data sets is functional. However, the system is limited by the time it requires to collect one data set (~1 hours). Thus, to generate a large data set of 620 or images as used in this training would require 620 hours which is extremely difficult given that the experiment cannot be run unattended and requires a human operator to move the lamp position after each data set is collected. Thus, another time-efficient alternative is to use TracePro data for training and experimental data for validation and re-training.

TracePro Simulation

Some selected flux maps obtained from the TracePro simulation are shown in Figure 18, 19 as well as Appendix A. Maps with a highly concentric nature as well as distorted nature were picked. This was done to better understand what coordinates or combinations thereof introduces distortion into the flux maps. Random distortions in the flux map would make CNN training difficult. Thus, if such randomly distorted flux maps are not required in real-life experimentation, such coordinates could be avoided from the sample data.

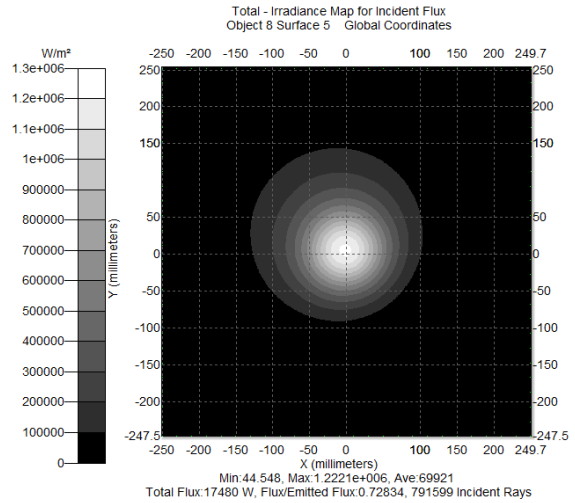


Figure 18: Flux distribution at target with source at (0, 0.25, 0)

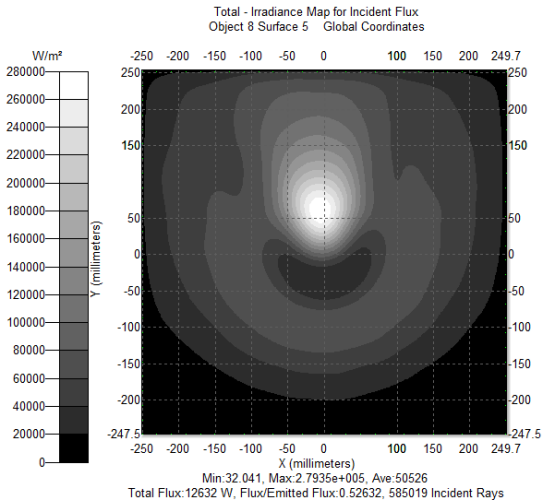


Figure 19: Flux distribution at target with source at (0, 8.25, -8.25)

As seen in the above figures and figures in Appendix A, moving away from the focal point introduces random distortions into the system. However, in Figure 19, majority of the flux still retains its concentric nature albeit at a different position. The z-direction has the greatest effect on the flux map distortion. This is expected since movement in the z-direction focuses and defocuses the lamp. Thus, it changes the amount of energy that reaches the lamp much more than any parameter.

Convolutional Neural Network Training

Several of the aforementioned hyper parameters were determined via experimentation once the type of neural network model was selected. The results of these will be discussed in this section.

Learning rate

The ideal initial leaning rate is determined through obtaining the mini-batch RMSE for different learning rates and plotting it against the number of iterations. For the different leaning rates, the leaning rate drop factor is kept at 0.2 for every epoch for a total five epochs and a mini-batch size of 31. The results for four different learning rates is shown in Figure 20.

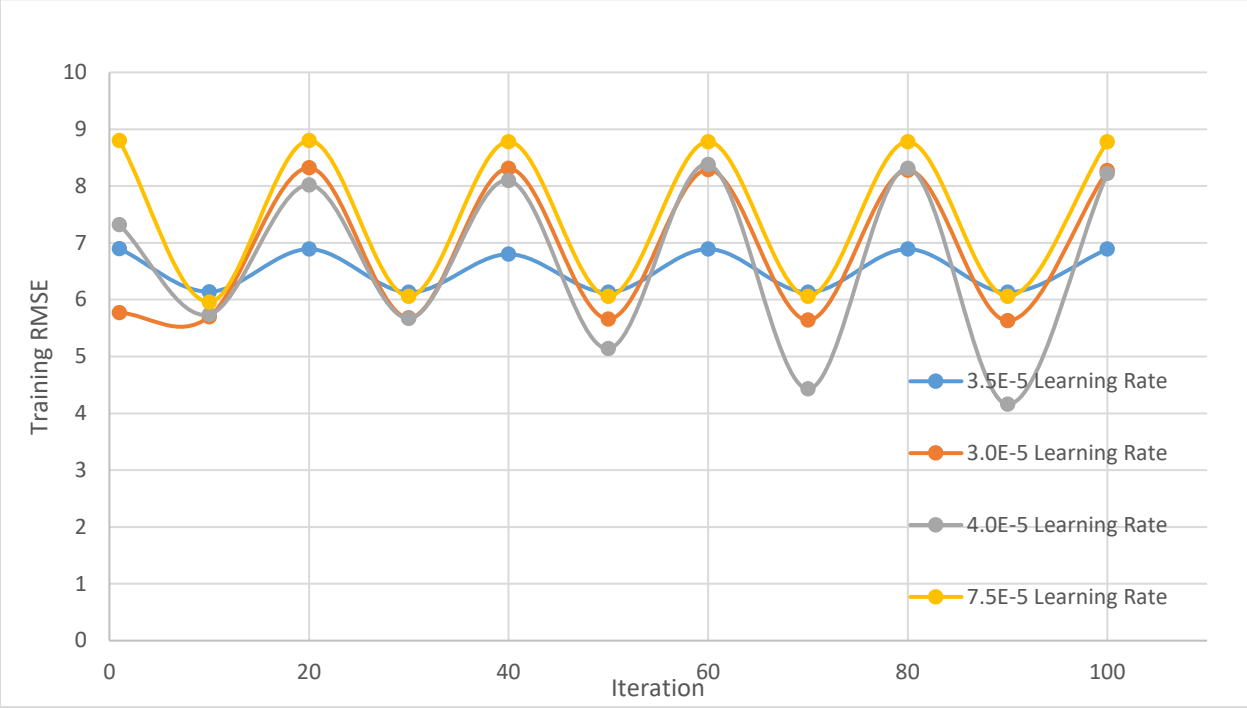


Figure 20: Comparison of training loss for different learning rates over 5 epochs

In Figure 20, the training RMSE for the learning rate of $3.5x10^{-5}$ is seen to be the most stable with less oscillation in comparison to the remaining rates. Although the RMSE for the $4x10^{-5}$ learning rate is not the lowest at some iterations, it is reasonably close to the minimum value at each dip. Additionally, at the peaks, the $3.5x10^{-5}$ has the lowest RMSE by a great

margin. Overall, the learning rate of $3.5x10^{-5}$ has consistently a low RMSE with respect to the other three learning rates and was therefore chosen for further testing.

For further learning rate comparison, the training loss of the same rates is plotted against the iteration as shown in Figure 21.

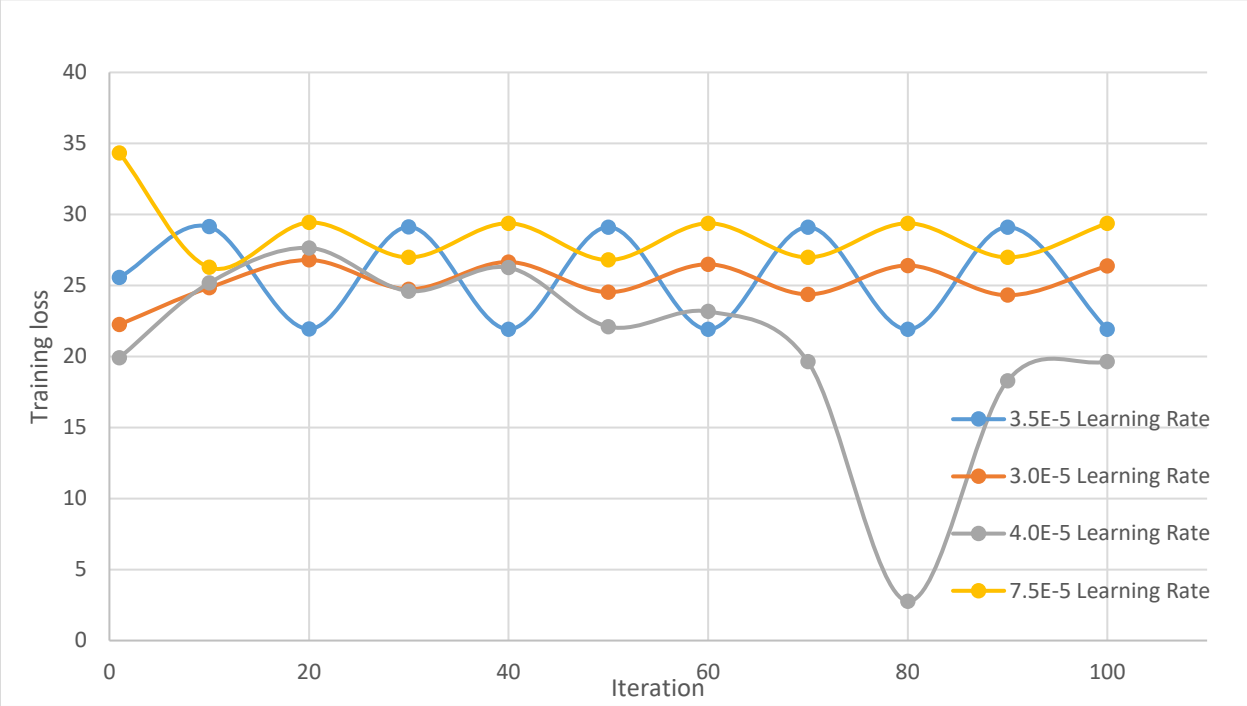


Figure 21: Comparison of training RMSE for different learning rates over 5 epochs

The training loss shown in Figure 21 shows the rate of $3.5x10^{-5}$ to have the largest oscillatory behavior compared to the other graphs. Also the peaks of $3.5x10^{-5}$ coincide with the troughs of the other learning rates. Nevertheless, it can be seen that the learning rate $3.5x10^{-5}$ drops to the lowest training loss in comparison to the remaining rates while its highest training loss is close to the rest. The only exception seen is that of $4.0x10^{-5}$ learning rate where it drops unexpectedly at 80 iteration and ends at 100 iteration with a slightly lower training loss than the $3.5x10^{-5}$ rate. The steep drop can been seen as an outlier that doesn't follow the general trend and the slight difference at the end still leaves the $3.5x10^{-5}$ rate as the best option.

In the subsequent testing of the remaining hyper-parameters, the learning rate of $3.5x10^{-5}$ is used based on the results discussed above.

Mini-batch size

The training loss and training RMSE were recorded and plotted against number of epochs as well as against number of iteration to compare performance of the model for five possible mini-batch sizes. These graphs are plotted against epochs and iterations because varying the mini-batch size varies the number of iterations per epoch. For example, smaller mini-batch size results in more iterations for the same number of epochs. Therefore, in order to have comparable results and make an informed decision, these training criterion are plotted against number of epochs as shown in Figures 22 & 23 and against iterations as shown in Figures 24 & 25.

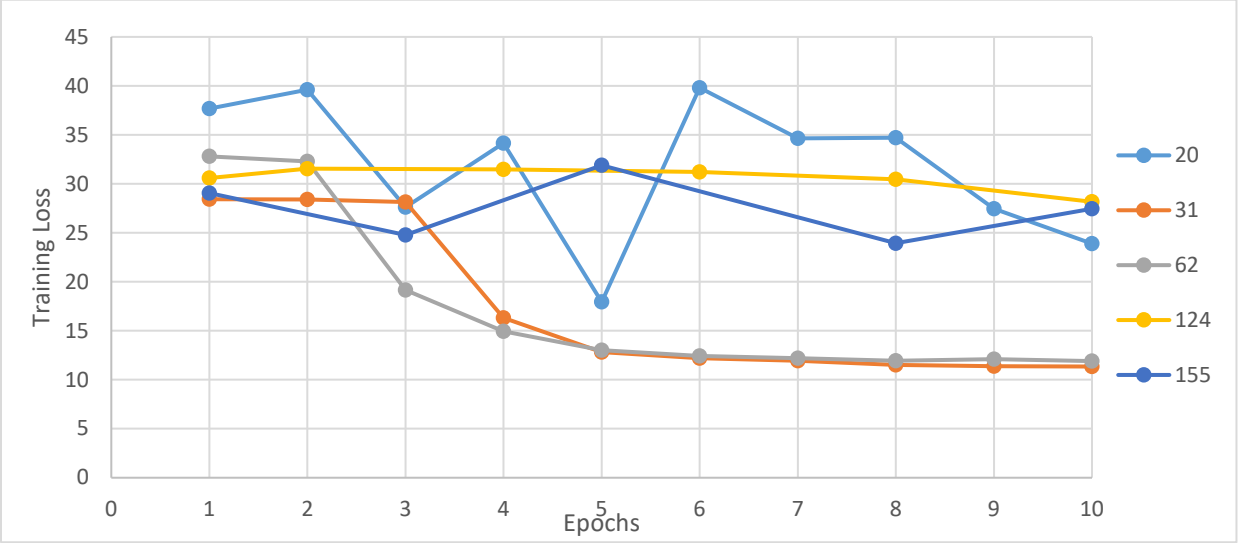


Figure 22 Comparison of training loss for different mini-batch sizes over 10 epochs

As seen in Figure 22, the mini-batch sizes of 31 and 62 eventually provide the smallest training loss. A mini-batch size of 62, however, drops the training loss at a faster rate initially and converges to approximately the same final value of around 11.

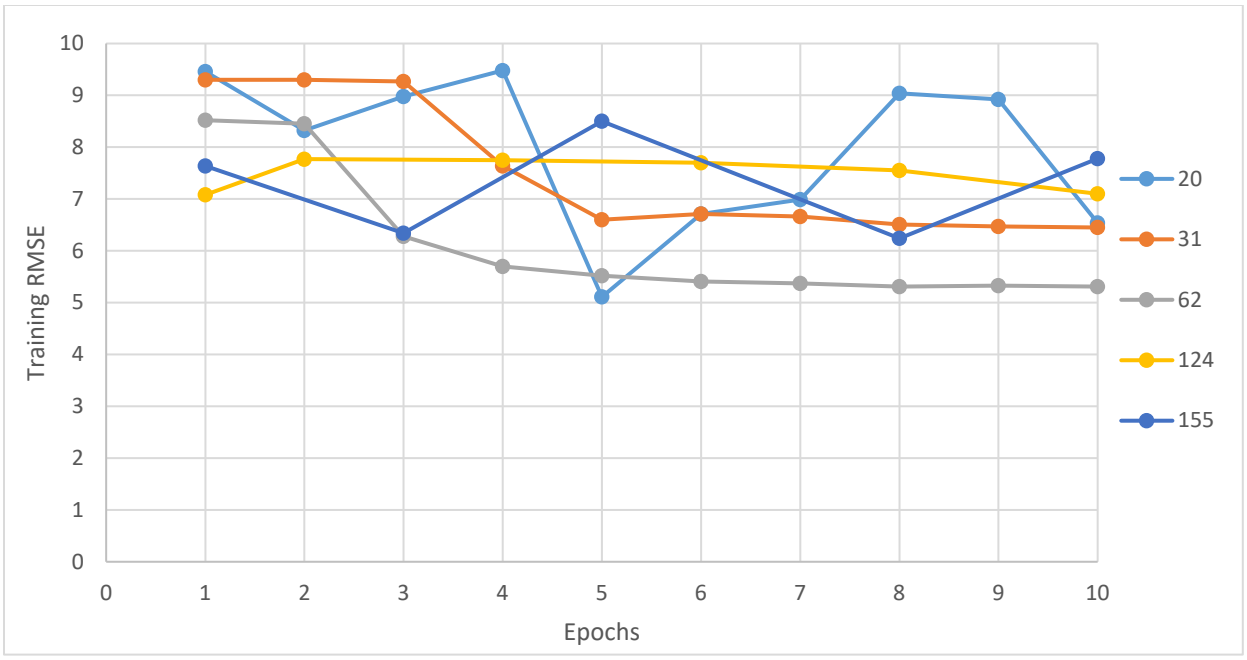


Figure 23 Comparison of training RMSE for different mini-batch sizes over 10 epochs

It is observed in Figure 23 that a mini-batch size of 62 drops faster and converges to a lower value when compared to the trend for mini-batch size of 31. Therefore, the performance of mini-batch sizes of 62 and 31 are quite similar, with mini-batch size of 62 performing slightly better.

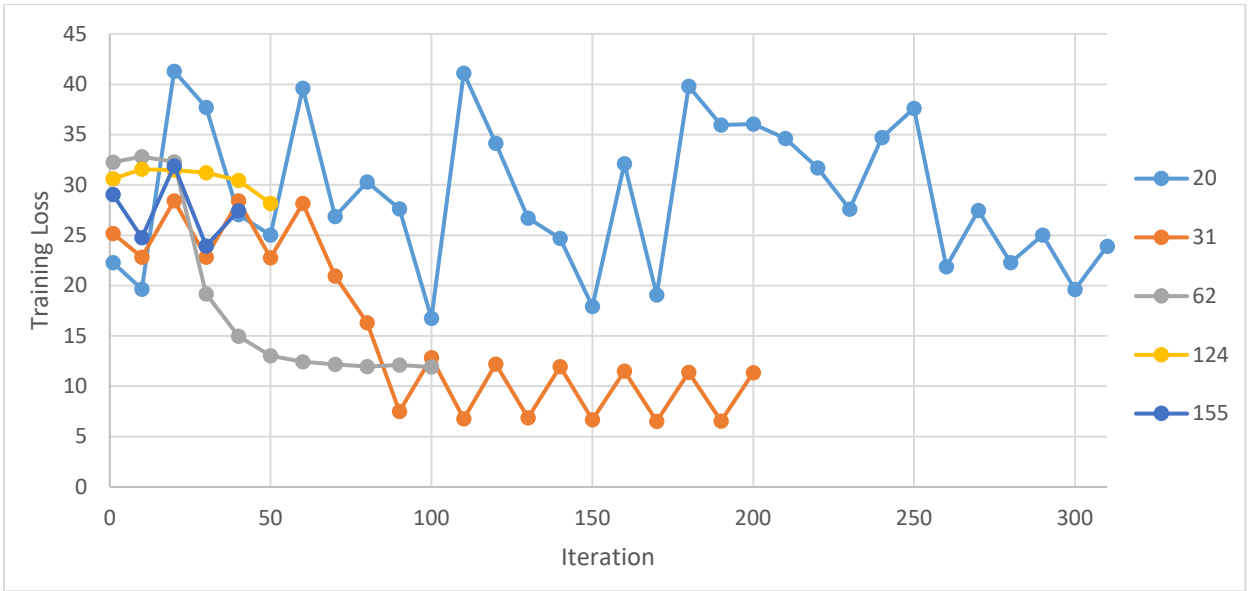


Figure 24: Comparison of training loss for different mini-batch sizes over 10 epochs worth of iterations

Figure 24 shows that a mini-batch size of 31 oscillates around two values but it clearly approaches a value lower than that achieved by a mini-batch size of 61. Therefore, given enough epochs, the model would have approached a much lower value of training loss.

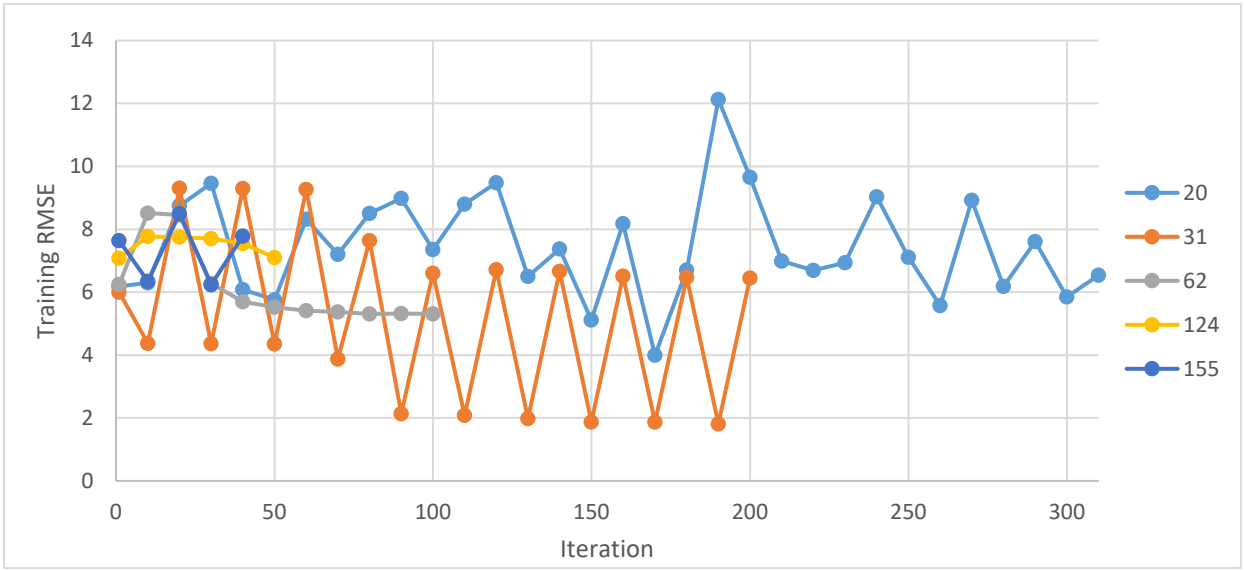


Figure 25: Comparison of training RMSE for different mini-batch sizes over 10 epochs worth of iteration

A similar trend is observed in Figure 25 as that in Figure 24. Furthermore, literature review suggests that for many applications, a batch size of 31 is optimal. Therefore, based on these results as well as literature review, a mini-batch size of 31 is best and was used in the final model. [30]

Number of convolutional layers

For the testing of the number of convolutional layers, the leaning drop rate of 0.75 at each epoch for a total of 10 epoch and a mini batch size of 31 is used. In total, five different number of layers were tested to determine the best. The minimum number of layers possible is found to be four as lower number of layers result in infinite training loss and training RMSE. The remaining number of layers are 5-9 layers. However, it was found from the results obtained that the training loss and training RMSE for six or more layers is the same. This is expected due to the limited effect that increasing the layers has on the train network. Therefore, for the

comparison purposes, six layers is treated as a representative of higher number of layers. Figure 26 shows a comparison between four, five, six and seven layers in terms of training loss at each iteration.

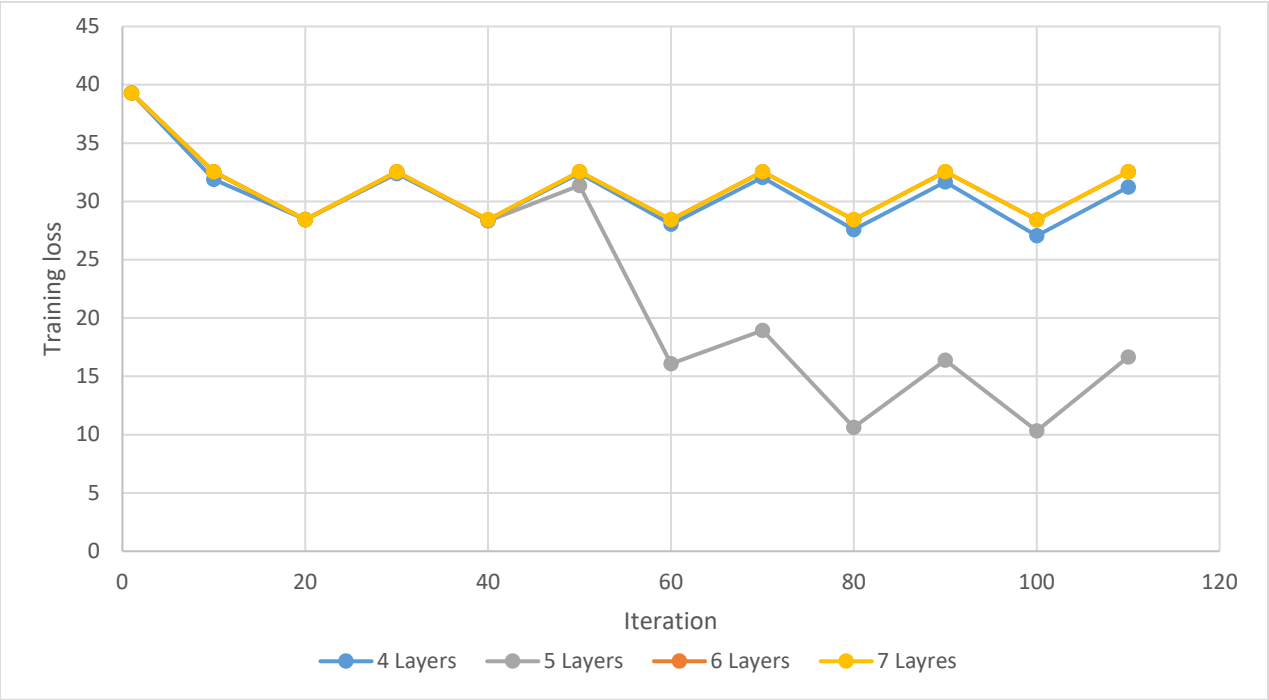


Figure 26 Comparison of training loss for different number of layers over 10 epochs

The plot of six layers cannot be seen in Figure 26. This is because the plot of seven layers is perfectly eclipsing the plot of 6 layers. Thus indicating that both have the same values. The training loss of the four and five layers are both consistent and overlapping for the most part. After 60 iteration, four layers is seen to be slightly better than six layers with a lower training loss. However, the biggest drop in training loss is exhibited by the five layers.

Figure 27 shows a comparison of training RMSE as a function of iteration for four, five, six and seven layers.

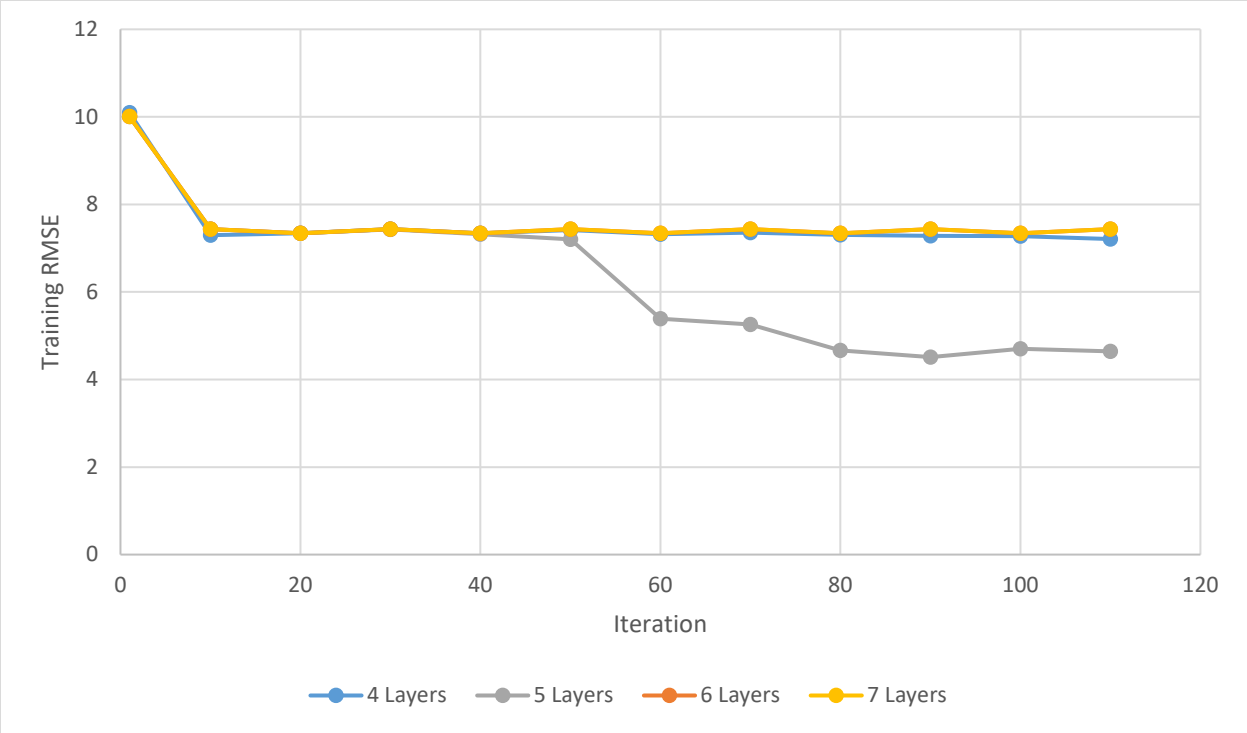


Figure 27: Comparison of training RMSE for different number of layers over 10 epochs

In Figure 27 six and seven layers can be seen to completely eclipse each other due to the exact equal values while four layers is almost completely overlapping with the two. This trend is similar to Figure 26. Five layers are shown to have the largest drop in training RMSE and the lowest possible value.

The trends in Figure 26 and Figure 27 are similar and indicate that five layers is the ideal number of layers to use.

Filter size

The optimal filter size was determined by analyzing the effect of four different filter sizes on the training loss and mini-batch RMSE, and the plots for these are displayed in Figures 28 & 29 below. The data for filter size of 5 is not displayed because it diverged to infinity in both cases. Thus only filter sizes of 2, 3 and 4 are compared.

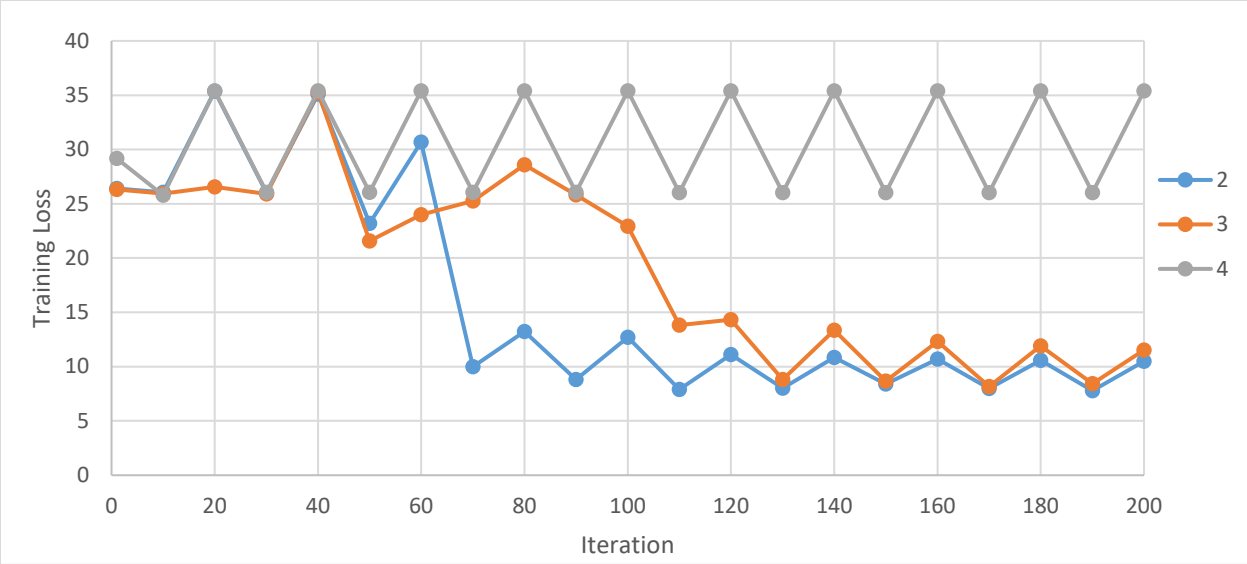


Figure 28: Comparison of training loss for different filter sizes over 10 epochs

In Figure 28, it is observed that filter sizes of 2 and 3 provide very close performance after 150 iterations. However, the training loss drops at a faster rate for filter size of 2 and even converges to a slightly lower final value than that of 3.

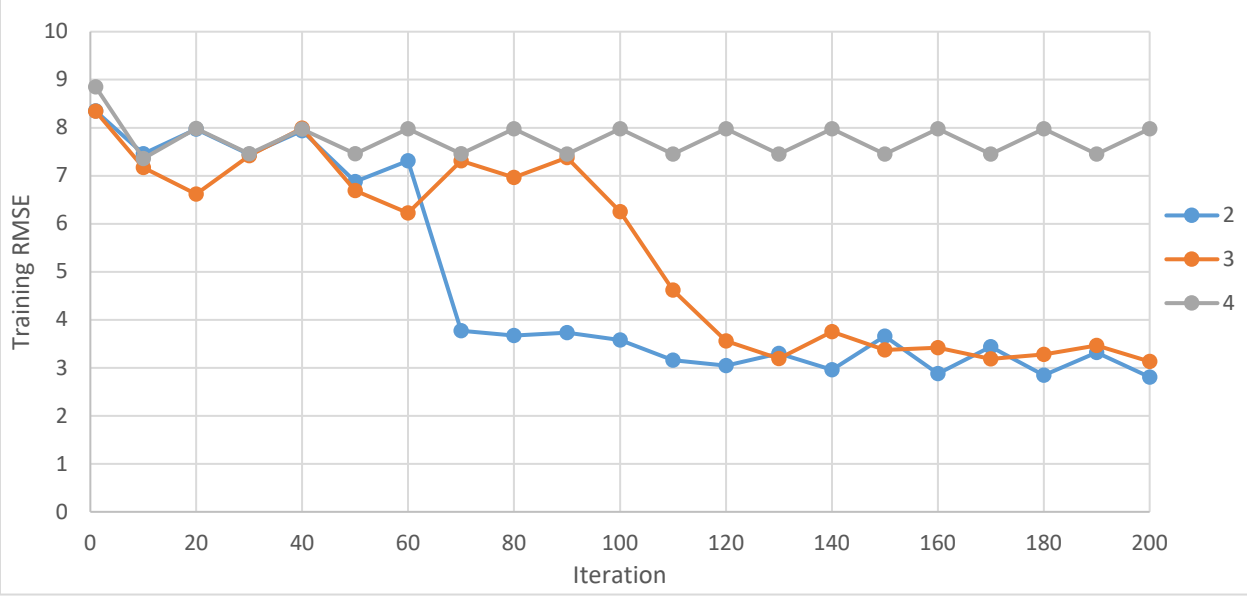


Figure 29 Comparison of training RMSE for different filter sizes over 10 epochs

A similar trend as Figure 28 is observed in Figure 29, where a filter size of 2 eventually converges to the lowest training RMSE value. As expected, a smaller filter size results in the best performance. Therefore, a filter size of 2 was used in the final model.

Number of filters

The number of filters were manipulated and the corresponding training loss as well as training RMSE across iterations is shown in Figure 30 and 31 respectively below. Any number of filters below 5 gave NaN error (numerical overflow error which signifies lack of any possible convergence).

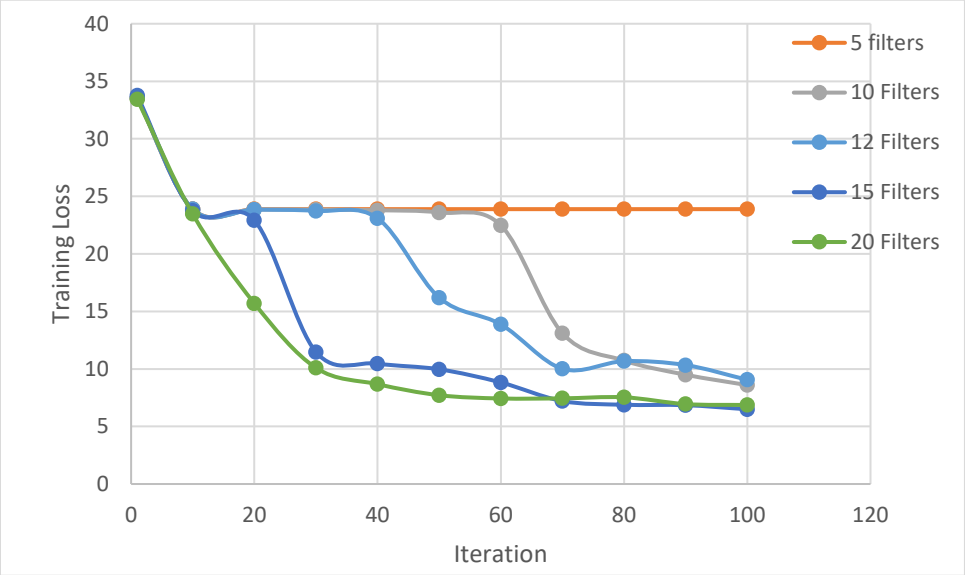


Figure 30: Comparison of training loss across 100 iterations for different number of filters

From Figure 30, it can be seen that 5 filters give sub-par performance. The performance increases as the number of filters is increased. The improvement in training loss from 15 to 20 filters is minimal. 20 filters reaches a lower training loss faster than 15 filters. However, 15 filters is too computationally intensive as 15 filters takes 693 seconds compared to 433 seconds for the same number of epochs. Thus, a value in between 10 and 15 was picked and tested.

Twelve filters gives a final performance similar to 15 and 20 filters after 100 iterations. To confirm that 12 filters is optimal, Figure 31 can be analyzed.

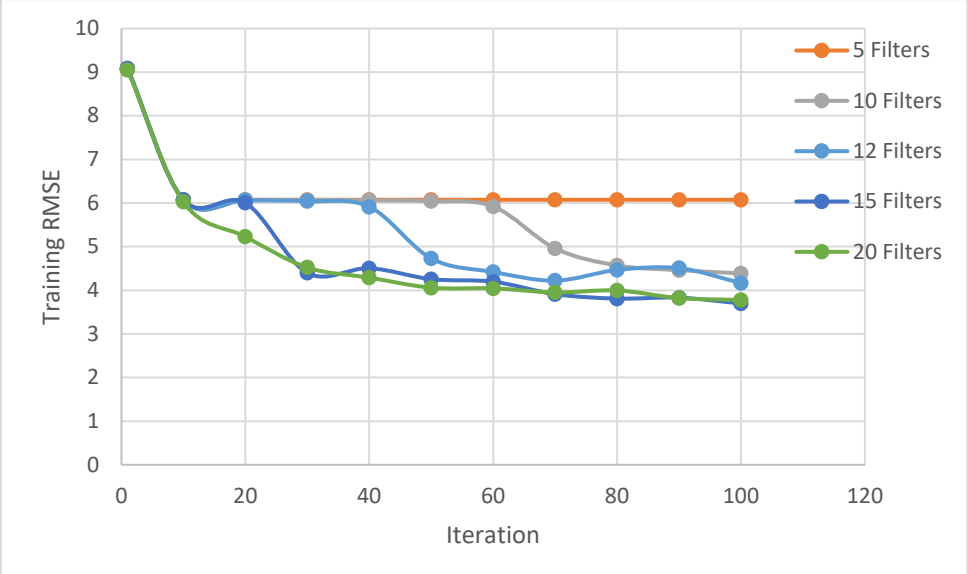


Figure 31: Comparison of training RMSE across 100 iterations for different number of filters

Looking at RMSE, it is clear that 12 filters give performance very close to 14 and 20 filters but at a lower computational requirement. Thus, 12 filters was chosen as the optimal number of filters for this particular case.

Validation of Final Model

The CNN was trained using the aforementioned hyper parameters for 100 epochs to allow for reasonable convergence to be achieved. The model resulting from this final training was then used for validation. The accuracy metrics (A_{sum} and $A_{threshold}$) were obtained for the final set of data and are shown in Table 2 below.

Table 2: Accuracy metrics for final validation test

	x	y	z
A_{sum}	25.04	31.07	289.02
$A_{threshold}$	89%	87%	30%

As seen from Table 2, the accuracy in the x and y directions is great but the accuracy in the z direction is much poorer. This result is most likely because the effect of the movement in the z-direction on the flux map is much more significant than the effect of x and z direction as mentioned earlier. Thus, the method sampling data needs to be updated to better reflect the sensitivity of the z coordinate on the flux map. One method is to generate more data for coordinates where the z direction changes while the x and y direction remains the same.

For further validation, 3 of the trained model's 101 predictions of lamp coordinates were used to obtain flux distribution maps from TracePro and were then compared against the flux maps for the actual coordinates those predictions were made for. The following 3 coordinates were randomly selected for the validation.

Table 3: Validation coordinates

Test #		x	y	z
1	Test Coordinate	3.25	-3.25	0.00
	Predicted Coordinate	3.27	-3.74	-1.89
2	Test Coordinate	0.00	0.00	-8.75
	Predicted Coordinate	-0.01	-0.04	-1.09
3	Test Coordinate	0.00	-5.50	5.50
	Predicted Coordinate	0.34	-5.78	1.34

Test #1

As seen in Table 3, the prediction made by the model is quite accurate for the x and y coordinate. The z coordinate prediction is off by around 1.9 mm. Figures 32 & 33 below are used to determine the error in the prediction by analyzing the flux distribution maps of the actual and predicted coordinates.

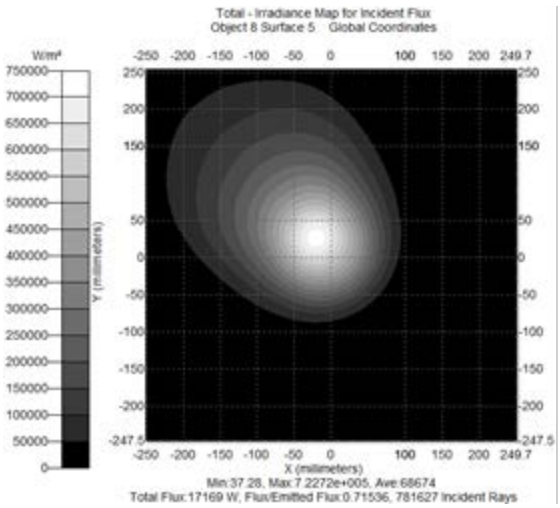


Figure 32: Flux distribution of actual coordinates

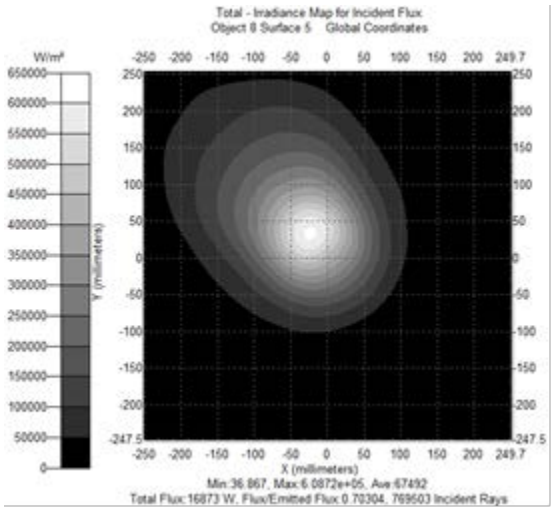


Figure 33: Flux distribution of predicted coordinates

Figures 32 & 33 show that the prediction made by the model accurately matches the actual flux map in terms of the flux distribution. The error in the peak flux was calculated to be approximately 148 kW/m^2. To better visualize the difference between actual and predicted flux maps, a plot of the error was created by subtracting the flux values of actual map from the predicted map. The relative error plot for this lamp arrangement is shown in Figure 34 below.

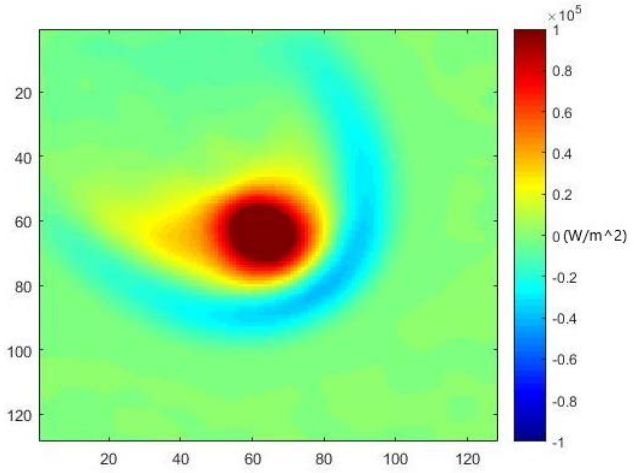


Figure 34: Relative error between actual and predicted flux map at (3.25, -3.25, 0)

Figure 34 shows that the error between actual flux map and the predicted one is small for the outer region and quite large for the central region. In this case, all predicted coordinates are

close to test coordinates since test coordinates are closer to 0. Smaller test coordinates imply lesser distortion which implies higher guess accuracy.

Test #2:

For the second test, the x and y coordinate predictions were very accurate, but again, the prediction for z coordinate was quite poor. Comparison of Figures 35 & 36 further shows the accuracy of the prediction made by the model.

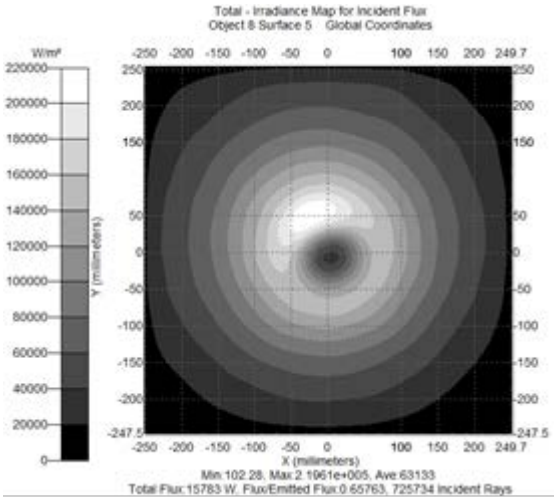


Figure 35: Flux distribution of actual coordinates

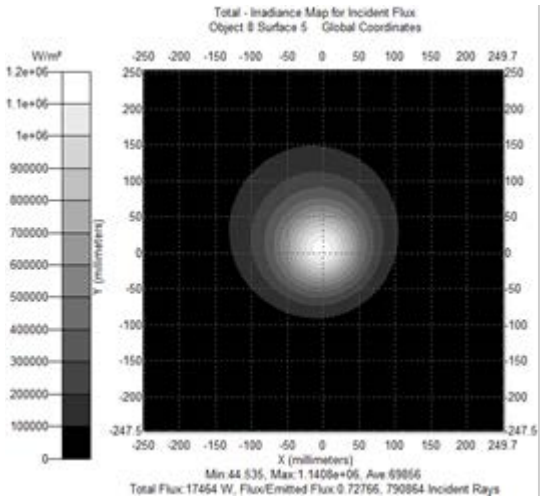


Figure 36: Flux distribution of predicted coordinates

Figures 35 & 36 show that the prediction made by the model poorly matches the actual flux map in terms of the flux distribution as well as the peak flux. The error in the peak flux was calculated to be approximately 1193 kW/m^2. The relative error plot for this test is shown in Figure 37 below.

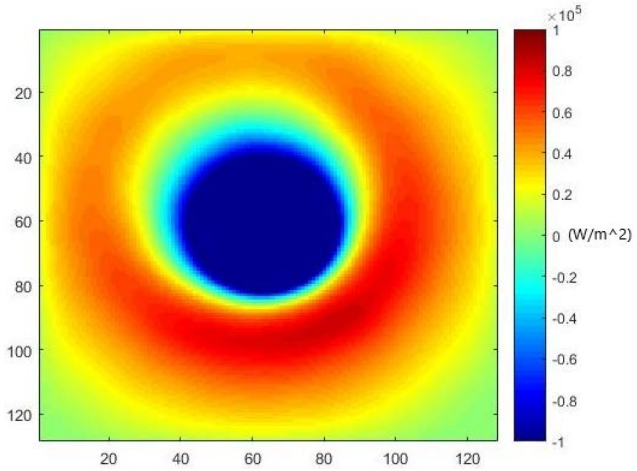


Figure 37: Relative error between actual and predicted flux map at (0, 0, -8.75)

Figure 37 shows that the error between actual flux map and the predicted one is large in the center due to the dark spot in the actual flux map, leading to the large negative error. The error is also quite large in the surrounding region due to the large radius of bright spot in actual flux map, leading to a large positive error. Overall, the relative error between the actual flux map and the predicted one is large for this particular lamp arrangement. It is also interesting to note, that the z-coordinate is -8.75 for the test case which would greatly increase the distortion in the test flux map. The increased distortion due to the z-coordinate most likely leads to the poor predictions of the z-coordinate by the trained network. There is otherwise no significant error in the other two coordinates.

Test #3:

For the third test, the x and y coordinate predictions were reasonably accurate, but again, the prediction for z coordinate was poor. Comparing flux maps in Figures 36 & 37 further shows the accuracy of the prediction made by the model.

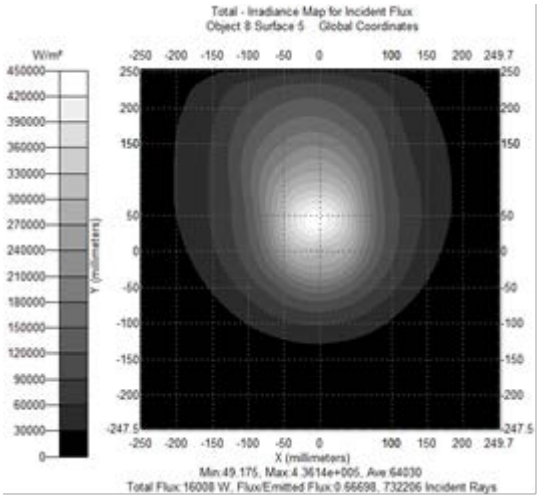


Figure 38: Flux distribution of actual coordinates

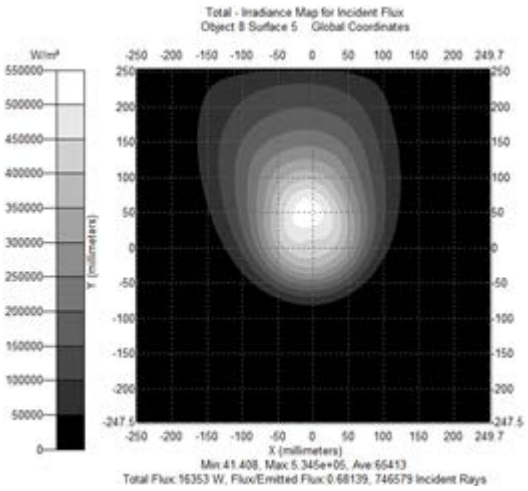


Figure 39: Flux distribution of predicted coordinates

As can be seen in Figure 38 & 39, the model predicts the general flux distribution fairly accurately. The error in the peak flux was calculated to be approximately 125 kW/m^2. The relative error plot for this test is shown in Figure 40 below.

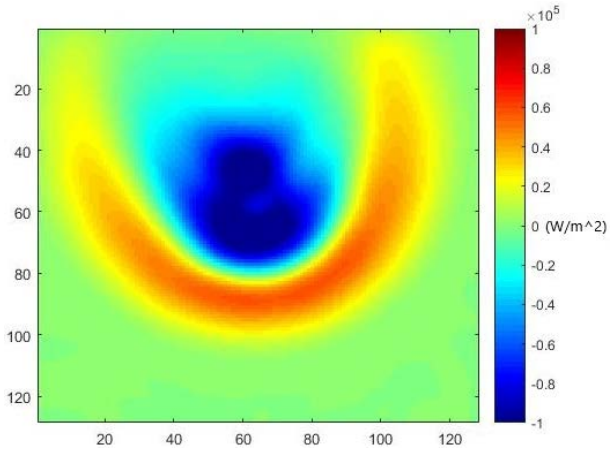


Figure 40: Relative error between actual and predicted flux map at (0, -5.5, 5.5)

Figure 40 shows that the error between actual flux map and the predicted one is small for the outer region and quite large for the central and its surrounding region. The negative error is large due to the bright spot in predicted flux map and the positive error is due to the more spread

out flux distribution in the actual flux map. In this case, the y and z coordinates are not close to 10 mm, which gives better testing performance.

Thus from all the above tests it is clear that the CNN predicts some cases well and some cases poorly. The main difference between the great and poor predictions is the z coordinate. Thus, the z-coordinate prediction of the trained network still needs to be improved as suggested by the accuracy metric in Table 2. This lends further credibility to the assertion that machine learning can be used to predict lamp coordinates for a given flux map.

CHAPTER IV

CONCLUSION

The automated LabVIEW system developed to collect data reduced collection time from 3 hours to 1 hour for each flux map-coordinate pair. However, this data collection time is still large and can be greatly reduced if the calibration curve can be used to convert raw CCD images to flux map. The in-house algorithm based on taking greyscale weighted average of multiple exposure images and using it within the flux mapping method gave excellent results. The calibration curve obtained from this method displayed highly level of linearity.

From, the TracePro simulation it can be concluded that flux maps shape is in the form of concentric circles when lamp source is close to the focal point. As the source is moved away, the flux maps become more distorted with great distortion noticed near the end of the range tested (10 mm) in all directions. The effect of the movement in the z-direction is much greater than the impact of movement in x or y direction.

The convolutional neural network was determined to be a good tool to address the problem proposed in this thesis. Similarly, the choice of mini batch gradient descent was also confirmed to be a good choice. The accuracy metrics used for final validation of the trained network displayed an excellent accuracy for x and y coordinate guesses. However, the guesses were found to be poor for the z coordinate. The error in z coordinate is most likely due to the higher effect of the z coordinate on the flux map shape which necessitates a different sampling method to improve accuracy of coordinate prediction.

The optimal hyper parameters for training the convolutional neural network for this case were determined. These hyper parameters are listed below:

- Learning rate: $3.5x10^{-5}$
- Learning rate drop rate: 0.75
- Mini-batch size: 31
- Number of convolutional layers: 5
- Filter size: 2
- Number of filters: 12
- Stride: 2
- Momentum: 0.9
- Regularization: L2 regularization

Despite significant error in z coordinate guesses, the overall success of the trained network in the x and y coordinate validate the initial hypothesis. Thus, neural networks can be used to predict lamp coordinates in multiple lamp high flux solar simulators.

CHAPTER V

RECOMMENDATIONS FOR FUTURE WORK

Based on the work done so far, some recommendations for future work in this field are summarized below:

- A more time-efficient method of collecting experimental data-sets to train the convolutional neural network need to be developed. These could be based on using the calibration curve to transform images from multiple lamp positions into respective flux maps. Data collection would be reduced to taking images which greatly reduce data collection time. Thus, the calibration curve needs to be tested for several lamp positions to confirm if one calibration curve can be used for several lamp positions.
- Use experimental data for validation of the trained network and for retraining the network.
- Increase the number of training epochs using more powerful computing systems such as supercomputers.
- Further investigate the reason for higher error in z coordinate predictions. Possible parameters to play with is increasing or decreasing the number of samples which move in the Z direction.
- Collect more data for coordinates closer to or greater than 10mm in x, y and z direction to better train for image distortion.
- Check the utility of other popular programming languages such as python for developing a network training code. Languages such as python have a larger dedicated community which may improve the code development experience.

REFERENCES

(1) Gallo, A.; Marzo, A.; Fuentealba, E.; Alonso, E. High Flux Solar Simulators for Concentrated Solar Thermal Research: A Review. *Renew. Sustain. Energy Rev.* **2017**, *77* (January), 1385–1402.

(2) Lorentzou, S.; Dimitrakis, D.; Zygogianni, A.; Karagiannakis, G.; Konstandopoulos, A. G. Thermochemical H2O and CO2 Splitting Redox Cycles in a NiFe2O4 Structured Redox Reactor: Design, Development and Experiments in a High Flux Solar Simulator. *Sol. Energy* **2017**, *155*, 1462–1481.

(3) Sarwar, J.; Georgakis, G.; LaChance, R.; Ozalp, N. Description and Characterization of an Adjustable Flux Solar Simulator for Solar Thermal, Thermochemical and Photovoltaic Applications. *Sol. Energy* **2014**, *100*, 179–194.

(4) Abbas, H. F.; Wan Daud, W. M. A. Hydrogen Production by Methane Decomposition: A Review. *Int. J. Hydrogen Energy* **2010**, *35* (3), 1160–1190.

(5) Codd, D. S.; Carlson, A.; Rees, J.; Slocum, A. H. A Low Cost High Flux Solar Simulator. *Sol. Energy* **2010**, *84* (12), 2202–2212.

(6) Pekakis, P. A.; Xekoukoulotakis, N. P.; Mantzavinos, D. Treatment of Textile Dyehouse Wastewater by TiO2 Photocatalysis. *Water Res.* **2006**, *40* (6), 1276–1286.

(7) Escobar, L. A.; Meeker, W. Q. A Review of Accelerated Test Models. *Stat. Sci.* **2006**, *21* (4), 552–577.

(8) Petrasch, J.; Coray, P.; Meier, A.; Brack, M.; Häberling, P.; Wuillemin, D.; Steinfeld, A. A Novel 50 KW 11,000 Suns High-Flux Solar Simulator Based on an Array of Xenon Arc Lamps. *J. Sol. Energy Eng.* **2007**, *129* (4), 405.

(9) Levêque, G.; Bader, R.; Lipiński, W.; Haussener, S. Experimental and Numerical Characterization of a New 45 KW_el Multisource High-Flux Solar Simulator. *Opt. Express* **2016**, *24* (22), A1360.

(10) Let there be light: German scientists test “artificial sun” https://phys.org/news/2017-03-scientists-artificial-sun-german-lab.html (accessed Jan 26, 2019).

(11) World’s largest artificial Sun rises in Germany https://newatlas.com/dlr-artificial-sun/48579/ (accessed Jan 26, 2019).

(12) Bishop, C. M. *Pattern Recognition and Machine Learning*; Springer, 2006.

(13) Schmidhuber, J. Deep Learning in Neural Networks: An Overview. *Neural Networks* **2015**, *61*, 85–117.

(14) Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*.

(15) LeCun, Y.; Bengio, Y.; Hinton, G. Deep Learning. *Nat. Int. J. Sci.* **2015**, *521*.

(16) machine learning - Neural Networks: Does the input layer consist of neurons? - Stack Overflow https://stackoverflow.com/questions/28288489/neural-networks-does-the-input-layer-consist-of-neurons (accessed Apr 6, 2019).

(17) Korbut, D. Machine Learning Algorithms: Which One to Choose for Your Problem https://blog.statsbot.co/machine-learning-algorithms-183cc73197c (accessed Apr 6, 2019).

(18) Edell, L. Is Machine Learning the New EPM Black? | Musings on ML, Deep Learning & AI https://scorecardstreet.wordpress.com/2015/12/09/is-machine-learning-the-new-epm-black/ (accessed Apr 6, 2019).

(19) Brownlee, J. A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/ (accessed Apr 6, 2019).

(20) Bengio, Y. *Practical Recommendations for Gradient-Based Training of Deep Architectures*; 2012.

(21) Boyle, T. Hyperparameter Tuning – Towards Data Science https://towardsdatascience.com/hyperparameter-tuning-c5619e7e6624 (accessed Apr 6, 2019).

(22) machine learning - What is the difference between model hyperparameters and model parameters? - Data Science Stack Exchange https://datascience.stackexchange.com/questions/14187/what-is-the-difference-between-model-hyperparameters-and-model-parameters (accessed Apr 6, 2019).

(23) Brownlee, J. How to Configure the Learning Rate Hyperparameter When Training Deep Learning Neural Networks https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/ (accessed Apr 6, 2019).

(24) CS231n Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/convolutional-networks/#pool (accessed Apr 6, 2019).

(25) XBO HSLA XBO 6000W/HSLA | OSRAM PIA https://www.osram.com/pia/ecat/XBO HSLA-XBO-Specialty lamps/com/en/GPS01_1028554/PP_EUROPE_Europe_eCat/ZMP_55893/ (accessed Mar 4, 2019).

(26) *XBO Theatre Lamps Technology and Application*.

(27) Training Options https://www.mathworks.com/help/deeplearning/ref/trainingoptions.html (accessed Apr 6, 2019).

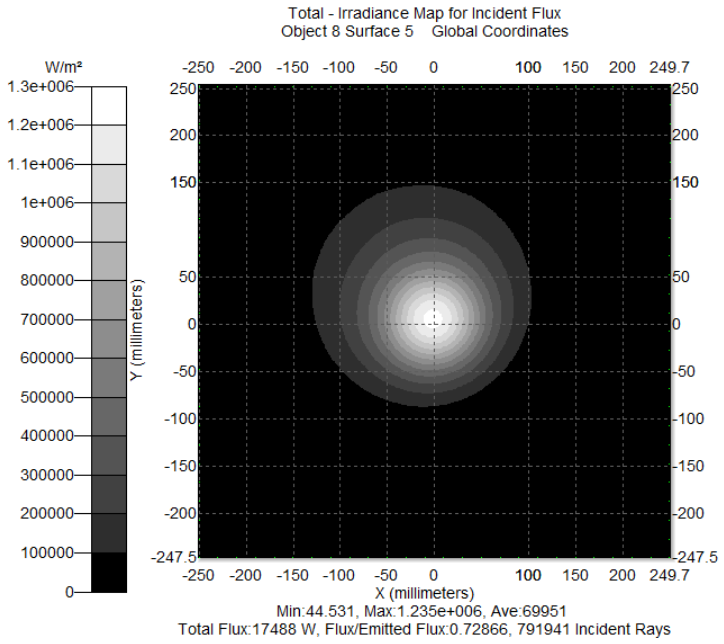
(28) Brownlee, J. A Gentle Introduction to the Rectified Linear Activation Function for Deep Learning Neural Networks https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/ (accessed Apr 7, 2019).

(29) Springenberg, J. T.; Dosovitskiy, A.; Brox, T.; Riedmiller, M. *STRIVING FOR SIMPLICITY: THE ALL CONVOLUTIONAL NET*.

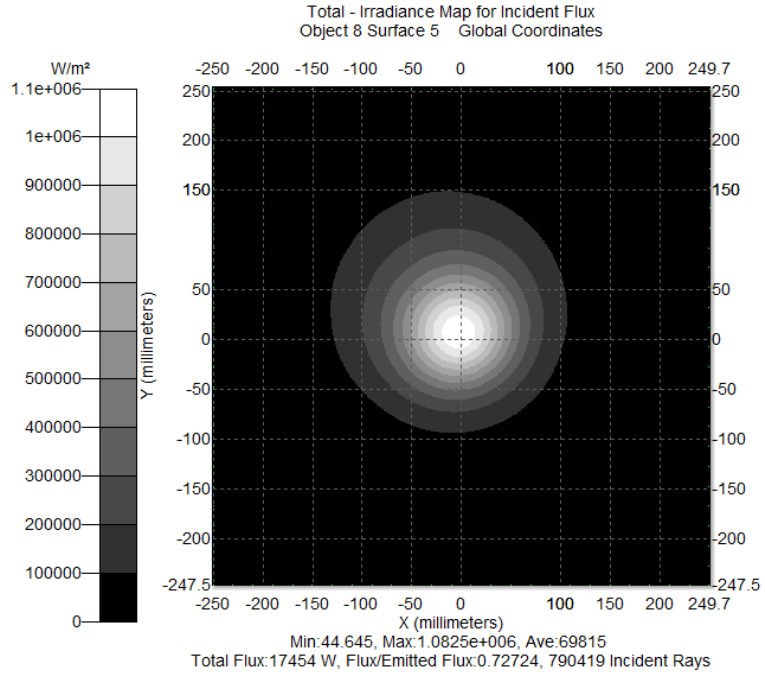
APPENDIX

Selected TracePro Flux Maps

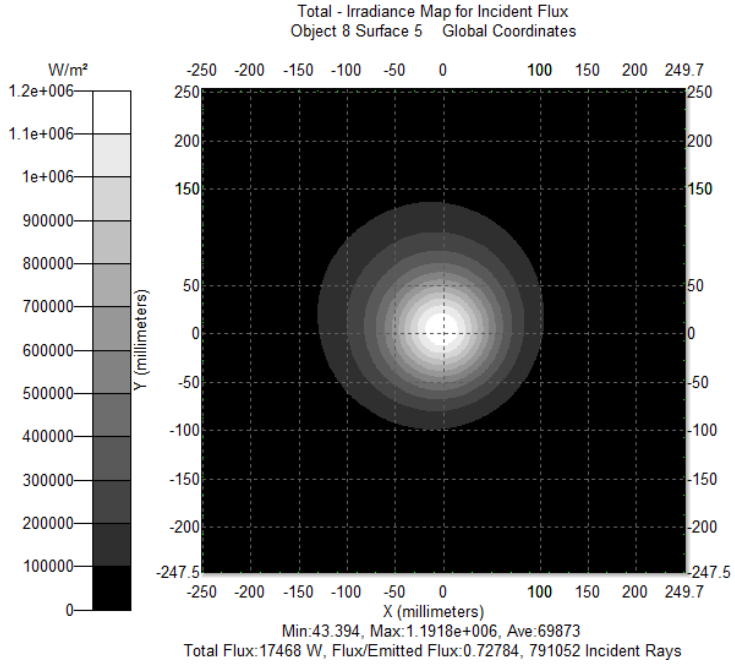
1. Position: (0, 0, 0.5)



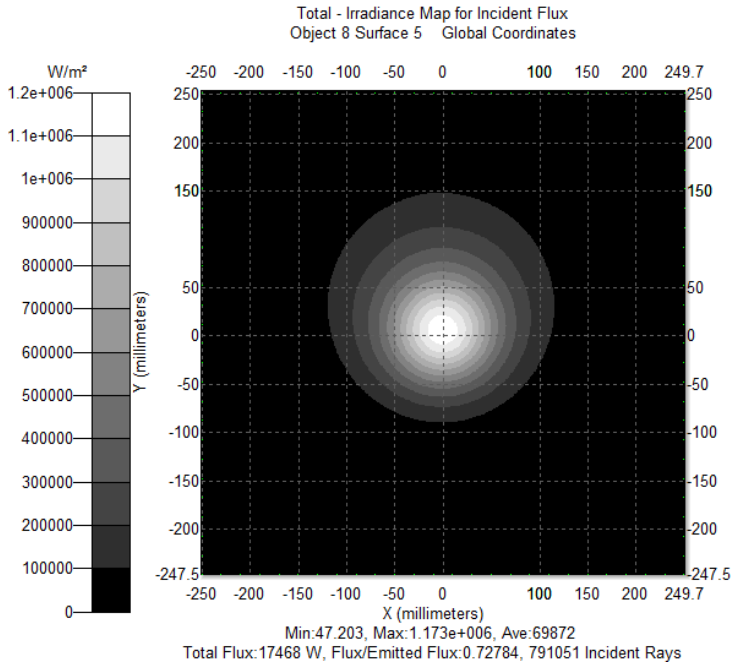
2. Position: (0, 0, -1.75)



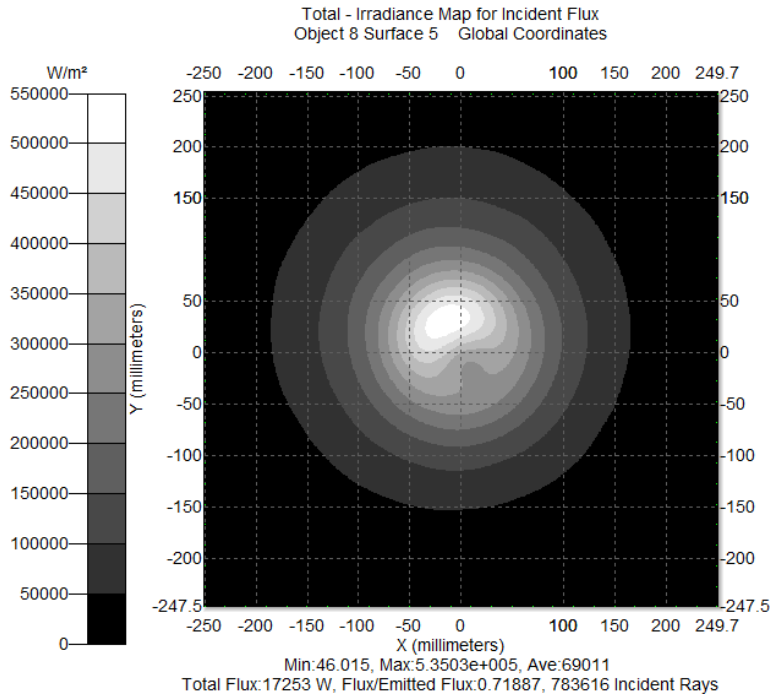
3. Position: (0, 0.75, -0.75)



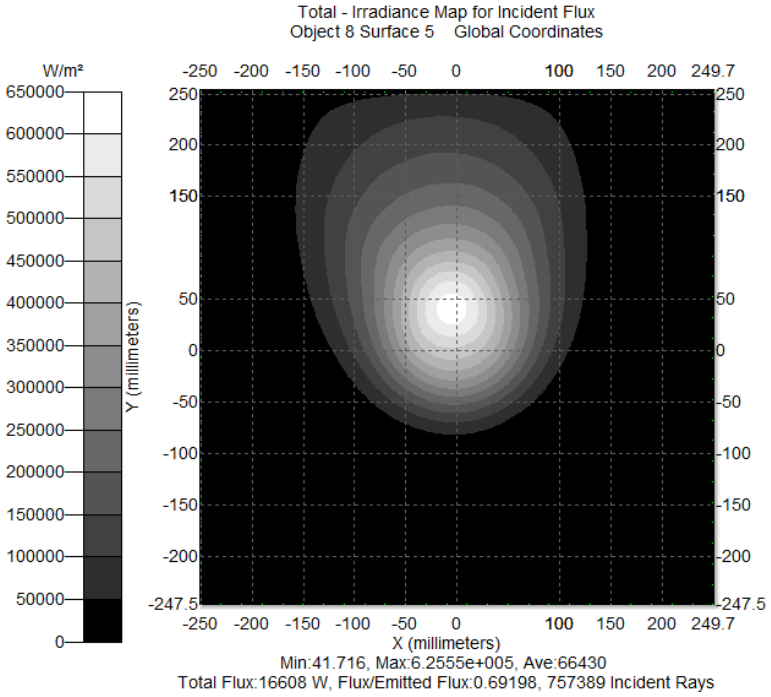
4. Position: (-0.75, 0, 0.75)



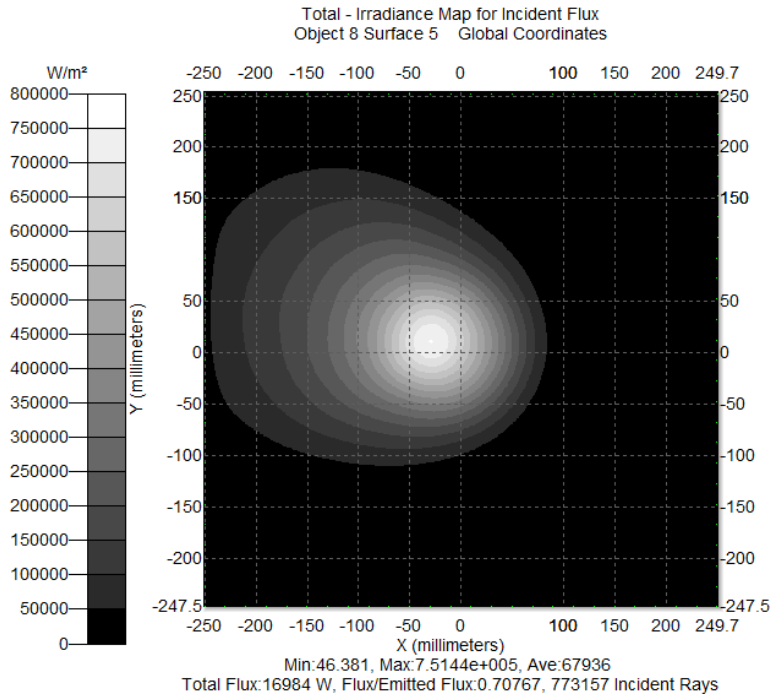
5. Position: (0, 0, -5.25)



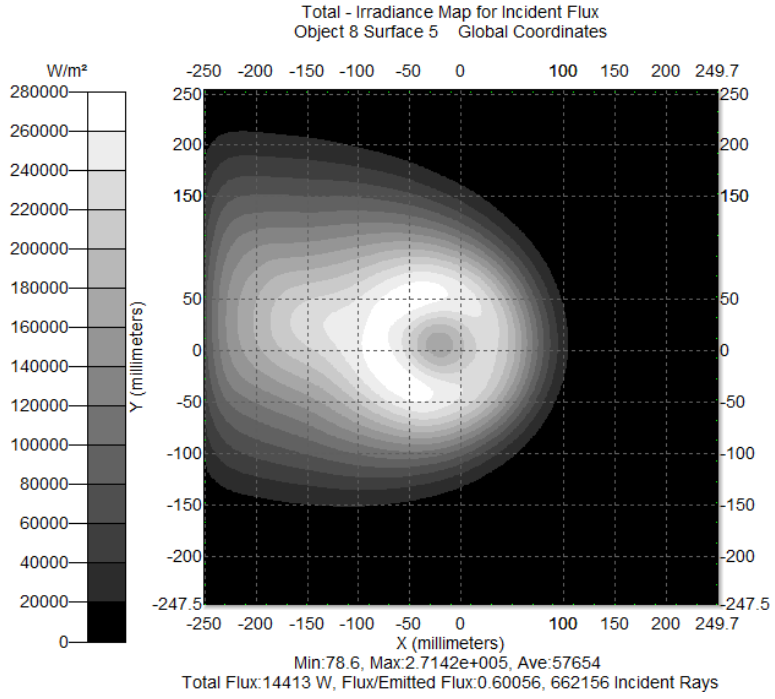
6. Position: (0, -5.25, 0)



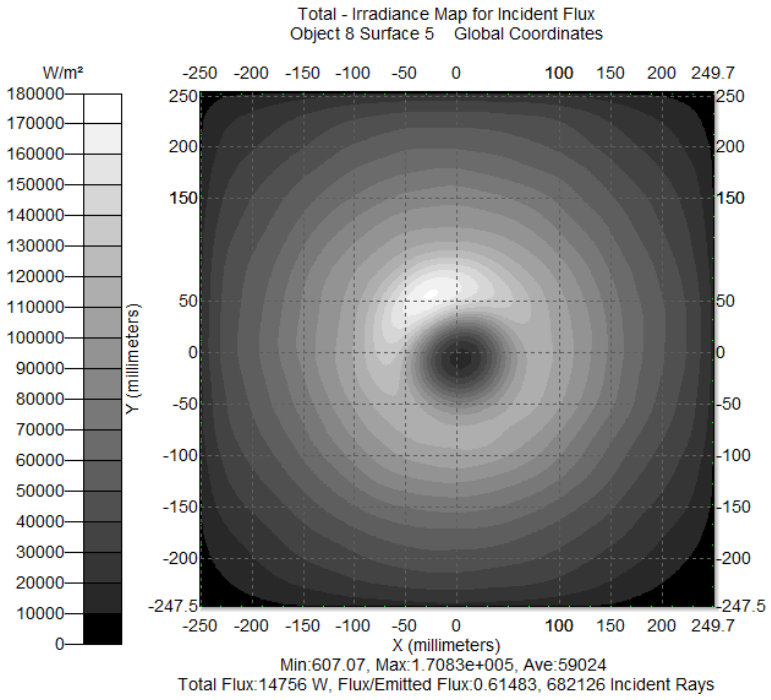
7. Position: (5.25, 0, 0)



8. Position: (10, 0, 0)



9. Position: (0, 0, -10)



10. Position: (0, 9.5, -9.5)