CS230

Inferring Shading Parameters from Graphically Generated Images

Madeleine J. Yip
Computer Science
Stanford University
mjyip@stanford.edu

Ruta Joshi
Computer Science
Stanford University
ruta@stanford.edu

Anthony Carrington
Computer Science
Stanford University
acarring@stanford.edu

Abstract

We use deep learning to reverse engineer a 3D image generated with MaterialX, a look development platform for computer graphics used for film. Given a computer-generated image of a mesh, shaded using 10 floating point properties, we regress the float values that define each property. We use transfer learning on existing VGG16 and ResNet50 architectures to evaluate the performance of different architectures on this domain. The network architecture we propose can be used for film and photography applications in which the properties of an image need to be known in order to regenerate similar images using a different platform.

1 Introduction

In film, animation, video game graphics, and design, computer generated images often need to be replicated for different mediums, and the properties used to render an image are sometimes lost in translation between platforms. It can be highly expensive for firms to replicate shaded images without prior knowledge of their properties. Being able to derive the properties of an image from the image itself will allow video game and film creators to generate photo-realistic imagery given images of the real world.

The input to our algorithm is a 224x224x3 RGB image, generated using MaterialX and portraying a mesh shaded based on 10 floating point properties. We then use a neural network built off of existing image processing networks to output a vector of 10 floats that correspond to the input image properties.

The image properties, which correspond to the autodesk standard surface parameters, and their respective ranges are below:

- Base (float from 0 to 1): the multiplier for base color
- Base color (RGB vector of 3 values, 0 to 255 each, normalized to be 0 to 1): main color of material
- Specular (float from 0 to 1): how concentrated the highlight is on a surface. (ie. shiny plastic vs dull wood. Shiny plastic with specular value closer to 1 has a more concentrated highlight). The default value is 0.5 for metals, since specular value is irrelevant for metals.
- Specular roughness (float from 0 to 1): the roughness of the surface.
- Specular color (RGB vector of 3 values, 0 to 255 each, normalized to be 0 to 1): the color of the highlight.
- Metalness (float 0 to 1): whether the material is a metallic or more plastic like material.

Given that each color property is a vector of 3 values, we are regressing to a total of 10 floats, each of which take on values from 0 to 1. Properties for training data are sampled uniformly at random over the range of each value.

2 Related Work

The concept of extracting graphics properties from a rendered image has been explored to some degree by other groups in recent years. Several prior researchers have tackled the area of "Deep Inverse Rendering," i.e. using visual cues from images to output the properties that went into producing those images.

Prior studies have focused on using custom encoder-decoder layers for local feature extraction and fully-connected layers for global feature extraction. For instance, in 2018, Deschaintre et. al. used 200,000 images to demonstrate that a rendering-aware deep neural network can capture a spatially varying bidirectional reflectance distribution function (SVBRDF), which is used to identify the reflectance behavior of a surface being graphically rendered.(1) Gao et. al. expanded upon this research in 2019 by developing an autoencoder for deep inverse rendering.The key contribution of this paper to the field is in capturing SVBRDF information from an arbitrary number of input images by using autoencoders to reduce the large quantity of image pixel information and therefore augment and improve the model proposed by Deschaintre et. al. In our work, we pivoted from custom encoder/decoder layers to test if well-known, pre-trained image classification neural networks like VGG16 and ResNet50 could perform better. (2)

Indeed, others researchers have focused on the problem of adapting existing image processing architectures to the problem of regression. In 2015, Belagiannis et. al. proposed a regression model using convolutional neural networks that worked better than traditional neural networks. This was the first step in successive attempts by researchers to improve the performance of convolutional neural networks on regression tasks. (3). In fact, in 2019 Lathuiliere et. al. demonstrated that deep regression can be done successfully with general purpose image processing frameworks, such as VGG-16 and ResNet-50. This was the first comprehensive analysis of the performance of these pretrained networks for the purpose of regression instead of classification. Our choice to compare the performance of VGG16 and ResNet50 with layers added for transfer learning stemmed from the insights we gleaned from this paper. We are building off of their work by testing different hyperparameters, using sigmoid output activations instead of linear, and adding custom layers instead of training earlier layers. (4)

Similarly, in 2017, Iglovikov et. al used a simple VGG-type neural network to regress bone-age from 12.6k radiological images of the hand. They evaluated their model in two distinct ways: normal regression mean-squared-error (MAE) and classification MAE, whereby MAE was done on softmax outputs from bone ages being separated into classes over their age-range (0 to 239 months). Ultimately, they found that their classification 'bins' method actually performed slightly better, so we decided to include this method in evaluating our own VGG16 model. (5)

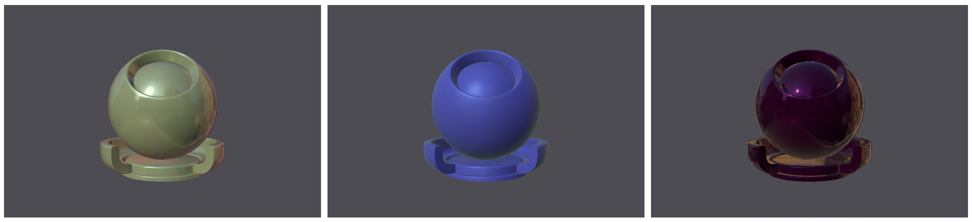
3 Dataset and Features

We generated the dataset by running a C++ script that loops through MaterialX files (mtlx) which describe each image to be generated by specifying the float values for each parameter. The script saves each generated 3D image as a JPG file - we use low resolution since it does not affect human performance and because smaller images are faster to use for training. To do this, we first generated 10,000 vectors, each containing 10 floats sampled uniformly at random from 0 to 1. We generated a mtlx format file for each vector and used the MaterialX graphics rendering engine to render 10,000 images of a mesh shaded with each of the vectors.

We split our dataset into 65% for training, 15% for validation, and 20% for test, since we only have 10,000 images to work with. We preprocess our images by cropping the inputs to 224x224x3, which is the input size for VGG16 and ResNet50 architectures provided by Keras. We preprocess the label vectors by normalizing the RGB values used to generate the image to floating point values from 0 to 1.

We used the raw images as input, without featurization. However, we train our network by freezing the pre-trained weights of VGG16 (and ResNet50, since we tried both) to get a feature map that is passed into novel architecture.

Some examples of the generated data are below:

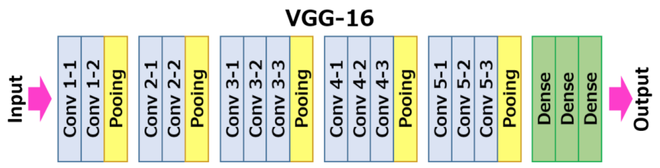


(a) 8000.jpg (b) 8001.jpg (c) 8002.jpg

Figure 1: Example datapoints

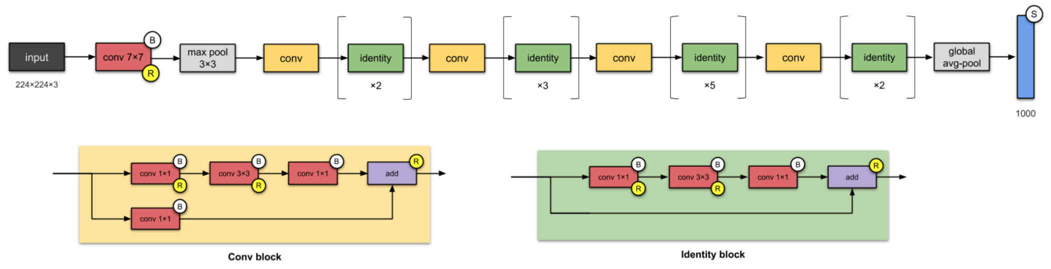
4 Methods

Our team created multiple learning models that all utilized VGG16 architecture or ResNet50 architecture pre-trained on Imagenet from Keras with a backend in tensorflow. VGG-16 is a model that achieves a top-5 accuracy of 92.7% on Imagenet classification and is notable for its performance and frequent usage in the deep-learning field. The model contains 5 blocks that each have 2 or 3 convolution layers and a max pooling layer. While VGG16 normally ends in 3 fully-connected (dense) layers and a softmax classification layer, we modified the output layer into a sigmoid layer of 10 neurons for regressing values between 0 and 1. We did the same for the ResNet50 models.



(a) Standard VGG16 architecture

For the ResNet models, we used a standard ResNet50 architecture, which consists of 5 stages, each with a convolution and identity block. The primary advantage of ResNet is the skip connections from one convolution layer to the next, which speed up training and solve the vanishing gradient problem, thus allowing deeper networks. We replaced output global average pooling layers with a 10 unit sigmoid layer, as in our VGG16 models.



(a) Standard Resnet50 architecture

To summarize, we tested the following baseline models, adding layers and tuning hyperparameters starting from these architectures:

- Logistic Regression (baseline model to see if the problem is learnable)
- VGG16 with Dense 10 neuron sigmoid layer output
- VGG16 with bins (1001 bins for each of 10 regression outputs) for classification (See loss function below)
- ResNet50 with Dense 10 layer sigmoid layer output

Our aim is to approximate shading models to a reasonable amount of significant digits since the difference of rendered images diminishes past a certain precision. To compare how structuring the data as continuous and discrete would affect accuracy, we used two different loss functions for each datatype. For continuous data, we used the L2 loss function to compare model outputs $\hat{y}$ to the label vectors y that correspond to the input images.

$$L_i = \frac{1}{2} \|y_i - \hat{y}_i\|_2^2$$

For the discretized data, we encoded the data as one-hot vectors by binning each property in 1001 possible sections. We ultimately decided that three significant digits seemed reasonable for each "step" in the discretized data. Because a traditional multi-categorical cross-entropy loss function does not take into the account the distance between categories, we modified the multi-categorical cross-entropy loss function to work with ordinal data:

$$L_i = -\left(1 + \frac{|c_{y_i} - c_{\hat{y}_i}|}{M-1}\right) \sum_{c=1}^{M} y_{o,c} \log(p_{o,c}),$$

where c_{y_i} is the true bin number, $c_{\hat{y}_i}$ is the predicted bin number and M is the number of bins (1001).

For all of our VGG16 and ResNet50 models, we froze the layers of the pretrained model for maximum transfer learning and only updated the weights of our newly added layers during transfer learning. To make our images fit the input constraints for VGG16, we cropped the images as closely as we could to the shading model and resized the images to be 224 X 224 X 3.

5 Experiments/Results/Discussion

The results of training our VGG16 and ResNet50 modified models are in the table shown. VGG16 V1 is plain vanilla VGG16 (base model) with a dense output layer of 10 sigmoid neurons. VGG16 V2 is the base model with an added convolutional layer, tanh activation, spatial dropout, and a dense output layer. ResNet V2 has the same added layers as VGG16 V2, with ResNet50 as the base model. ResNet50 V3 has the base ResNet model with added average pooling, 2 dense layers, spatial dropout, and batch normalization. Bins V1 is vanilla VGG16 with an added output layer of 10 one-hot vectors. Bins V2 has the same added layers as VGG16 (V2) and ResNet50 (V2).

Model	Train Loss	Train MAE	Val Loss	Val MAE	Test Loss	Test MAE	Epochs
VGG16 V1	0.0772	0.2017	0.1178	0.1127	0.1200	0.351	51
VGG16 V2 [Added layers]	0.0862	0.2517	0.0862	0.2533	0.0857	0.2522	63
ResNet50 V2 [Added layers]	0.0857	0.2521	0.0864	0.2530	0.0864	0.2528	66
ResNet50 V3 [Added layers]	0.0443	0.163	0.0948	0.2615	0.09	0.26	13

Bins Model	Train MSE	Train MAE	Val MSE	Val MAE	Test MSE	Test MAE	Epochs
Bins V1	0.1626	0.4032	0.1742	0.4173	0.1742	0.4174	68
Bins V2	0.2272	0.4767	0.2331	0.4828	0.2250	0.4744	21

We used the Adam optimizer for all models, with accuracy tracked for the VGG models and mean absolute error tracked for the ResNet50 models. We use a default learning rate of 0.001 for the Adam optimizer for all models except for ResNet50 V2 for which we used learning rate 0.01 to speed up training. For most models, we used a mini-batch size of 25 images per batch, which made for 260 batches in the training set, 60 batches in the validation set, and 80 batches in the test set. We picked this batch size primarily so that we could speed up training, and by testing a few different batch sizes for performance. The exception was ResNet50 V3, for which we used batch size 50 to speed up training, since we added more layers to this model than to the others.

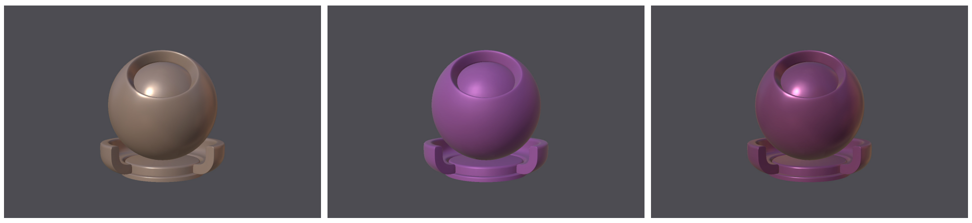
We also added layers to some of our models which were trained during transfer learning by freezing the weights of the base architecture. In all V2 models, we added one convolutional layer of 30 5x5 filters, a tanh activation, spatial dropout with parameter 0.4, and a dense output layer. This allowed us to reduce the dimensionality of the feature map outputted by VGG16 or ResNet50 respectively and to pass it into our newly added layers to learn a mapping to the 10 shading parameter float values.

We use Mean Absolute Error as the metric for comparison across models. The mean absolute error is the average of the absolute value of the difference between the output float and input shading parameter for each of the 10 outputs.

$$MAE = \frac{1}{10}\sum_{i=1}^{10} |\hat{y}_i - y_i|$$

Based on our training, we found that the most useful model without bins was ResNet50 V3. This was the network for which we replaced the output layer with an average pooling layer, a 512 neuron dense layer, dropout with a parameter of 0.5, and batch normalization. We used a learning rate of 0.001 for this model and tuned the batch size to be 50 instead of 25 for faster training. This resulted in 130 training data batches, 30 validation data batches, and 40 test data batches. This model had the lowest training loss (0.0433). The test loss and MAE for all of the non-bins models was similar, indicating that the task was difficult to improve by changing the model.

Qualitatively, the ResNet50 V3 model was able to capture metalness and highlight (specularity) information appropriately, as you can see from figure 4. However, the inferred colors were noticeably different from the colors of the input data. Refer to Figure 1 for the images as generated with their ground truth labels for comparison.



(a) 8000_pred.jpg (b) 8001_pred.jpg (c) 8002_pred.jpg

Figure 4: Predicted Images using ResNet50 V3

To improve further, we turned to the classification by discretizing the outputs into bins. Unfortunately, the bins models did not perform better than the regression models. The MAE of those models was higher than expected, likely because we used too many bins for each float. We discretized by making 1001 bins for each float value from 0 to 1. In the future, given more time, we would train using only 101 bins (the extra 1 is for the 0th bin) in order to reduce the variability of the output space. We would also design a more meaningful loss function to compare the output image prediction with the input image prediction instead of comparing the values of the 10 shading parameters. We believe that both bins models overfit to the training set. The cross entropy loss function values are not included in the tables shown because the magnitude of the loss varies greatly from the loss used for the regression models. VGG16 with bins V1 had train loss 32.8706 and validation loss 209.7871. Version 2 had train loss 44.8922 and validation loss 187.3383. We did not evaluate these models on the test set, given that the overfitting lowered performance below our regression models.

6 Conclusion/Future Work

In summary, we discovered that ResNet50 models work slightly better than VGG16 models for training, and that classification is a simpler task for VGG16, but not entirely appropriate for our problem space. We believe our best model was ResNet50 V3, which performed best on the training set and performed at par with other models on the validation and test sets. Based on prior research, we believe that ResNet50 V3 performed slightly better than other models due to the simpler network architecture that was being learned, as compared to our VGG models.

Throughout analyzing VGG16 and ResNet50, we experimented with different learning rates, batch sizes, optimizers, output metrics (regression MSE vs classification MSE), and adding additional training layers, such as batch normalization, dropout, convolution layers, and dense layers. Given more computational power, we would certainly increase the dataset size from 10,000, as training with even that number of images is time and compute expensive. This would help improve the variance in results like our ResNet50 V3 model, which had around 0.16 MAE on training and 0.26 MAE on the test set. We would also experiment with unfreezing some of the earlier layers of VGG16 and ResNet50 instead of just training additional layers on top. This would help reduce bias by capturing features that are determined earlier in the models.

7 Contributions

Madeleine Yip - data generation, image preprocessing, VGG16 transfer learning, VGG16 binning architecture, model evaluation and binning loss function
Ruta Joshi - baseline logistic model, data generation, label preprocessing, ResNet50 transfer learning models, model evaluation, paper, poster
Anthony Carrington - image preprocessing, ResNet50 transfer learning models, model evaluation, paper, poster

8 Code

All code for this project can be found at https://github.com/rutajoshi/cs230project.

References

[1] V. Deschaintre, M. Aittala, F. Durand, G. Drettakis, and A. Bousseau, "Single-image svbrdf capture with a rendering-aware deep network," *ACM Transactions on Graphics (SIGGRAPH Conference Proceedings)*, vol. 37, p. 15, aug 2018.

[2] D. Gao, X. Li, Y. Dong, P. Peers, K. Xu, and X. Tong, “Deep inverse rendering for high-resolution svbrdf estimation from an arbitrary number of images,” *ACM Transactions on Graphics (TOG)*, vol. 38, no. 4, p. 134, 2019.

[3] V. Belagiannis, C. Rupprecht, G. Carneiro, and N. Navab, “Robust optimization for deep regression,” in *Proceedings of the IEEE international conference on computer vision*, pp. 2830–2838, 2015.

[4] S. Lathuilière, P. Mesejo, X. Alameda-Pineda, and R. Horaud, “A comprehensive analysis of deep regression,” *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[5] V. Iglovikov, A. Rakhlin, A. Kalinin, and A. Shvets, “Pediatric bone age assessment using deep convolutional neural networks,” 12 2017.

- https://autodesk.github.io/standard-surface/