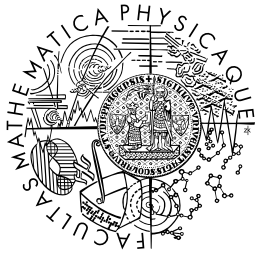


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Rustam Fadeev

HDR image semantic segmentation

Department of Software and Computer Science Education

Supervisor of the bachelor thesis: doc. RNDr. Elena Šikudová, Ph.D.
Study programme: Computer Science
Study branch: General Computer Science

Prague 2022

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I want to express sincere gratitude to my supervisor, doc. RNDr. Elena Šikudová, Ph.D., for the help and support in writing this thesis. I thank my family and friends for believing in me.

Title: HDR image semantic segmentation

Author: Rustam Fadeev

Department: Department of Software and Computer Science Education

Supervisor: doc. RNDr. Elena Šikudová, Ph.D., Department of Software and Computer Science Education

Abstract: This work attempts to create a pipeline that accepts the high dynamic range (HDR) input in the .exr format, processes it, and feeds it to the deep neural network, which can perform a semantic segmentation task, detecting the sky. Currently, to our knowledge, available semantic segmentation models cannot accept HDR input in the .exr format. Several models that are trained on HDR input are presented and analyzed here.

Keywords: HDR Semantic Segmentation Deep Learning Sky imagery

Contents

Introduction

The high dynamic range (HDR) imagery provides much more quality than the low dynamic range (LDR). Moreover, in some fields, HDR images can be essential for usage. Self-driving car cameras must detect road signs, other cars, and people. However, if the sun is shining right into the camera, LDR recording may lose much essential information. The same is valid for nature photography, where dynamic range can vary greatly. SkyGAN Mirbauer et al. [2022] project from Charles University uses HDR input of sky to generate realistic clouds. There, LDR input would be insufficient. It is probably because the HDR format usually weighs more and requires more processing to be displayed on non-HDR devices than LDR, it is not so widespread. To our knowledge, no available semantic segmentation models can accept and work with HDR input.
This thesis has several goals:

- Experiment with training the networks on the HDR input and with different ways to process such input. Understand if it can provide good results.
- Present the model that can accept the HDR input in .exr format and segment it, detecting pixels with the sky.

Presented here model may help the SkyGAN project Mirbauer et al. [2022]. Produced segmentation can be used as a mask, with which it is possible to automatically cut out only the sky from images and use that as input. The central assumption of this work is that if the HDR image contains more information than LDR, then the model trained on such input will show better results than the one trained on LDR.

1. Background

1.1 Semantic Segmentation

"Semantic segmentation, or image segmentation, is the task of clustering parts of an image together which belong to the same object class. It is a form of pixel-level prediction because each pixel in an image is classified according to a category." Paperswithcode [2022] In other words, it is a task where each image pixel is annotated with some class, such as sky or non-sky. This task is considered more complex than image classification, where the program must identify what is on the image and sometimes where it is located.

1.2 Machine Learning

The classical definition of machine learning comes from Mitchell [1997]: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

The two most common ways of training (E) are supervised and unsupervised. Supervised training required annotated (labeled) dataset. In other words, such a dataset contains both input and ground-truth output. Unsupervised training requires a dataset with only raw input. Semantic segmentation in this work (and predominantly too) is considered supervised; however, unsupervised semantic segmentation exists Hamilton et al. [2022]. Regression and classification are the two most common types of tasks (T). Regression is when the program's output is a continuous value $x \in \mathbb{R}$. For example, it can be the price of the house. In the classification task, the program outputs one of k pre-defined categories. For example, detecting if the image contains a dog or a cat. Semantic segmentation is a classification task: each pixel belongs to some category (sky or non-sky). A labeled unit in semantic segmentation is called a "mask," an array equal to input dimensions where each cell (pixel) has the correct category. If the task is a binary classification, then a mask usually contains values of 0 (non-sky) or 1 (sky). There are many different types of measures (P). Usually, the choice of measure depends on the task. Examples of measures are F-score, precision, error rate, and others. In semantic segmentation, the most common measure is the IoU score.

1.2.1 The IoU score

Interception over Union is also called the Jaccard index. It is defined by a simple formula $J(A, B) = \frac{|A \bigcap B|}{|A \bigcup B|}$. It is a measure between 0 and 1, representing a percentage between 0% and 100% of overlapping between predicted and ground-truth values.

1.2.2 Hypothesis

In supervised training, simplified workflow is this: Input X (Training Set) → Learning algorithm → Output Y (Hypothesis). Function h maps from X's to Y's. The goal is, given a training set, to learn a function h: X → Y. For example, $h_\theta(x) = \theta_0 + \theta_1 x$. Θ is called a parameter.

1.2.3 Loss function

In machine learning, the loss function (also called the cost function) measures the error between the output of the model and the ground truth. If the error is high, the loss function would also be significant. Given hypothesis $h_\theta(x) = \theta_0 + \theta_1 x$, we need to choose θ_0 and θ_1, so that $h_\theta(x)$ is as close as possible to the ground truth. In other words, the program's purpose is to change its parameters to minimize the loss. The loss is a function of the parameter $\theta J(\theta)$. For example, when there are two parameters, $J(\theta_1, \theta_2)$, the plot of the loss function is 3D, and the program's goal is to find its global minima.

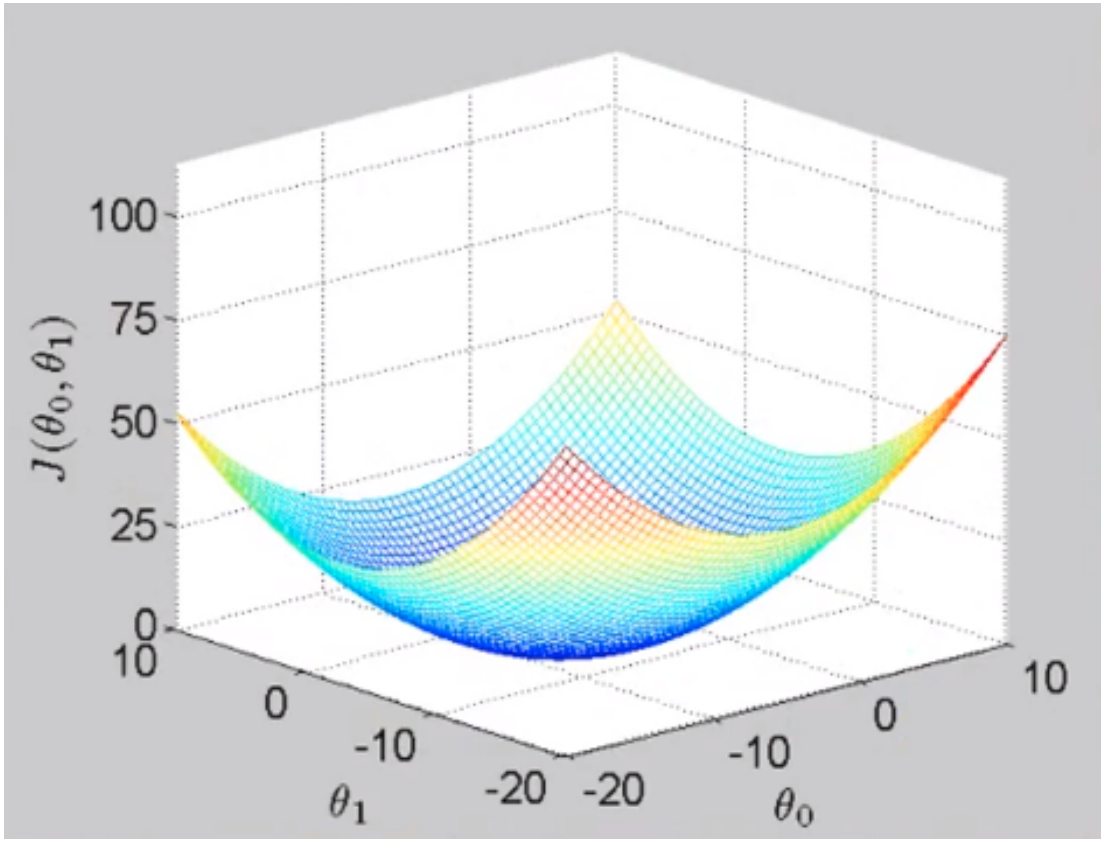


Figure 1.1: Plot of a convex Loss function $J(\theta_1, \theta_2)$. Credits: Stackoverflow [2022].

1.2.4 Deep Learning

Deep Learning is a class of Machine Learning algorithms that use layers to learn Deng and Yu [2014]. Mostly, it is neural networks.

1.3 Neural Network

In neural network, inputs x_1, x_2, x_3 ... are called Input Wires, h(x) hypothesis is Output Wire. Θ parameters in Neural Networks are called weights and usually represented as w.

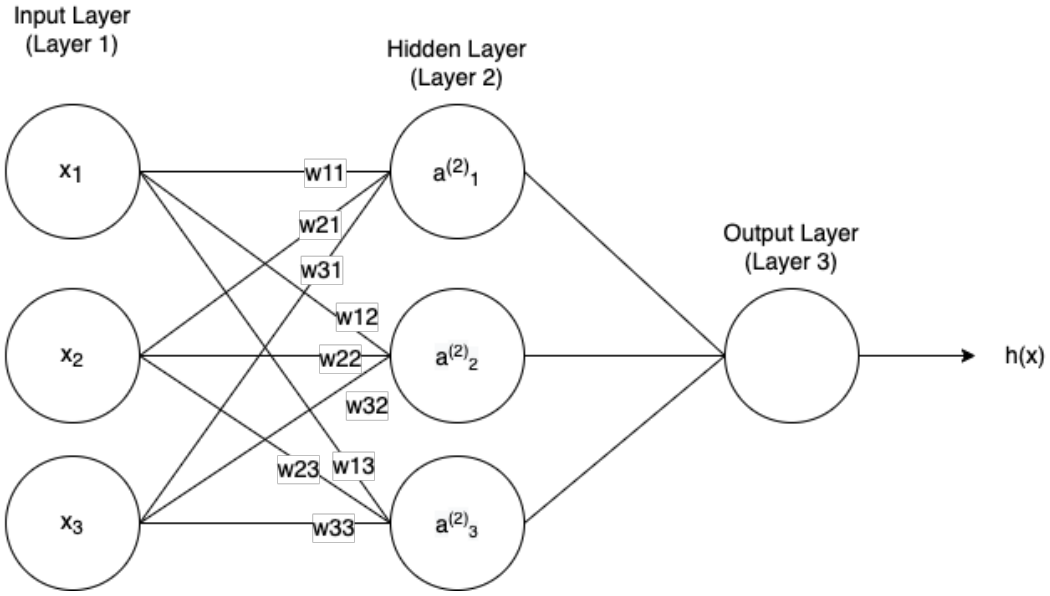


Figure 1.2: Simple 3 layer neural network.

A neural network is a group of artificial neurons (nodes) divided into layers: the more layers - the deeper the network. The First Layer of inputs is called the input layer, the last layer that outputs the hypothesis is called the output layer, and the layers in between are called hidden layers. Input and Hidden layers also contain "bias," values equal to 1, x_0, and a_0, but are not represented in the Figure 1.2. Amount of nodes and layers can vary. The node i in the hidden layer l is represented by activation unit a_i^l. $\Theta^{(j)}$ is a matrix of weights controlling the function mapping from layer j to layer j+1. g is the activation function. It must be differentiable. For example, the activation function can be sigmoid, ReLU, or tanh. The h(x) hypothesis is also an activation function.
Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$
ReLU: $Relu(z) = max(0, z)$
Tanh: $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{1-e^{-2x}}{1+e^{-2x}}$
The program computes the gradient of the loss function to minimize it, and weights are changed accordingly to get the lowest point in the loss function surface. Optimizers, such as Adam and Stochastic Gradient Descent, perform this task in neural networks.

Different combinations of layers are called architectures. The saved trained neural network is called a model.

1.3.1 Loss function

Binary Cross-Entropy

"Cross-entropy measures the difference between two probability distributions for a given random variable or set of events" Jadon [2020]. $CE(p_t) = -log(p_t)$ where p_t is a probability.

In other words, it attempts to measure the differences in information content between the actual and predicted image masks. It is generally based on the Bernoulli distribution and works best with equal data distribution and for binary classification. Therefore binary cross-entropy may not adequately evaluate image masks with a very heavy class imbalance. In the task at hand, the sky is a dominant class.

Jaccard loss

This loss is also called Generalized IoU loss. "The Jaccard loss measures the similarity between finite sample sets A and B as the Intersection over Union (IoU): $J(A,B) = \frac{|A \bigcap B|}{|A \bigcup B|}$" David Duque-Arias [2021]. The loss function must be differentiable and set operations are not. Therefore Jaccard loss is only an approximation of the IoU score. It is computed with the following formula: $sum(|A * B|)/(sum(|A|) + sum(|B|) - sum(|A * B|))$.

Focal loss

"Focal loss can be seen as a variation of Binary Cross-Entropy. It down-weights the contribution of easy examples and enables the model to focus more on learning hard examples. It works well for highly imbalanced class scenarios." Jadon [2020] Considering the percentage of images with only sky or ground, the focal loss seems to be a good function for the task.
$FL(p_t) = -(1-p_t)^{\sigma} log(p_t)$ where p_t is a probability.

1.4 Convolutional Neural Network

"Convolutional networks Bengio and Lecun [1997], also known as convolutional neural networks, or CNNs, are a specialized kind of neural network for processing data that has a known grid-like topology." Goodfellow et al. [2016] Images can be represented as a grid, where each pixel is a cell. The network uses a linear mathematical operation called convolution. "Convolutional networks are neural networks that use convolution in place of general matrix multiplication in at least one of their layers" Goodfellow et al. [2016]. The convolutional operation awaits two arguments, input, and kernel, then produces an output, the feature map. 1.3 is an example of convolution operation with 2D data.

1.4.1 Kernel

Applying different kernels provides benefits—for example, a vertical edge detector. Let us say we have input 6x6, white on the left, gray on the right, and a

kernel 3x3. If we apply to it our filter (kernel), which we can represent as three vertical lines from white (1) to gray (0) to black (-1), we will get the vertical edge in the middle of the resulting 4x4 image. Many other such kernels act as detectors. However, they are not hand-engineered but are learned during the neural network's training and are treated as parameters.

1.4.2 Pooling

Applying the fxf kernel to the nxn image will produce $(n-f+1)x(n-f+1)$ output. Resulting image shrinks. We can use the filter only a couple of times before the image "disappears." Central pixels are used much more often than pixels on the edges, and essential information is lost. To solve this, we "pad" the image by adding extra pixels to the image's border. For example, a 6x6 image becomes 8x8, and applying a 3x3 kernel to it produces 6x6 output again. Applying the same kernel to 6x6 input would produce 4x4 output. Moreover, now edges of 6x6 input are employed by kernel more often, as it "moved" more to the middle now.

1.4.3 Advantages

The advantages of convolutional neural networks are parameter sharing and sparsity of connections. Parameter sharing - feature detector (like vertical edge detector) that's useful in one part of the image is probably helpful in another. Sparse connections - in each layer, each output value depends only on a small number of inputs, thus, it is less prone to overfitting.

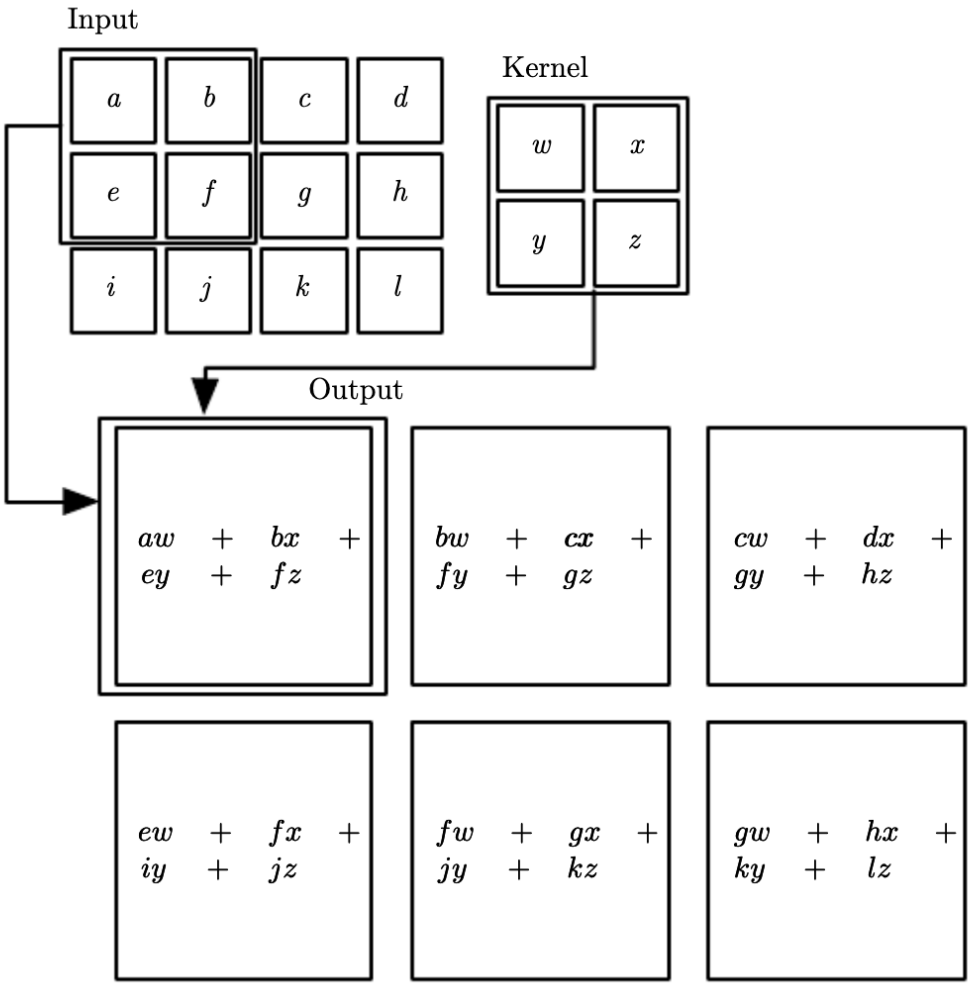


Figure 1.3: 2D convolution operation. Credits: Goodfellow et al. [2016].

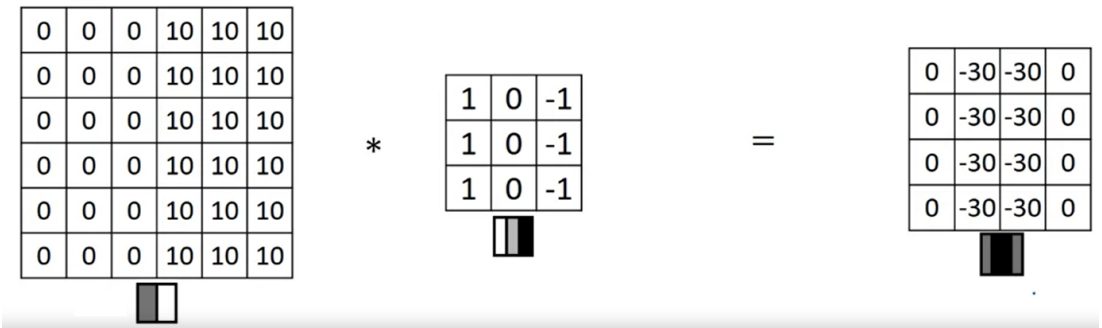


Figure 1.4: Kernel edge detector. Credits: DeepLearning.ai and lectures by Andrew Ng

1.5 HDRI

1.5.1 Dynamic Range

High Dynamic Range Imaging. The dynamic range describes the ratio between the maximum and minimum measurable light intensities (luminosity) in Colour [2022b] Reinhard et al. [2006]. In photography, dynamic range is measured in exposure value (EV) differences, which describes the total light range by powers of 2. In contrast with Low Dynamic Range (LDR) image, an HDR image contains details and information in the very bright and dark areas of the image instead of being just pure white and black colors. Usually, an HDR image is created by taking several LDR images with different exposures and combining them into one HDR image 1.5.

Figure 1.5: Example of image merging for creating HDR photo. Credits: in Colour [2022b].

1.5.2 Bit depth

"Bit depth quantifies how many unique colors are available in an image's color palette in terms of the number of 0's and 1's, or "bits," which are used to specify each color." in Colour [2022a] LDR images use 8-bit depth, meaning Red, Green, and Blue (RGB) colors can be represented with values from 0 to 255. HDR images use 32-bit depth (although 16-bit is also possible) for each color. Every HDR image file stores much more information than LDR and thus weighs more. OpenEXR OpenEXR [2022] Reinhard et al. [2006] format is a popular standard for storing HDR data. It supports 16-bit floating-point, 32-bit floating-point, and 32-bit integer pixels OpenEXR [2022]. The file extension of this format is .exr. HDR images are not widespread, and most available applications and libraries support only LDR.

2. Related Works

2.1 ResNet

Residual Network was first introduced here He et al. [2015]. While, in theory, a deeper neural network should be better than its shallow counterpart with fewer parameters, it is more complicated in practice. The main obstacle is the problem of vanishing/exploding gradients He et al. [2015] which depends on the number of layers. During gradient descent, as we backpropagate from the final layer back to the first layer, we are multiplying by the weight matrix on each step, and thus the gradient can decrease exponentially quickly to zero (or, in rare cases, grow exponentially quickly and "explode," from gaining tremendous values). Shortcut connections are introduced in Residual Networks to solve this problem 2.1. "The shortcut connections simply perform identity mapping, and their outputs are added to the outputs of the stacked layers" He et al. [2015]. After n layers from the decided shortcut (called X), we add up the output of the n-th layer and X. This is also called a skip connection.

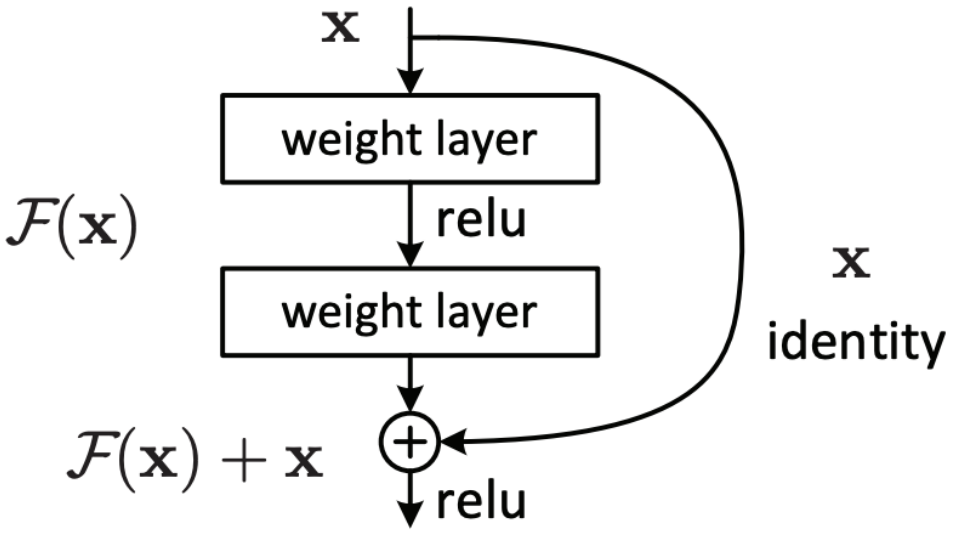


Figure 2.1: Building block of ResNet with a shortcut. Credits: He et al. [2015].

Authors introduce versions with 34, 50, 101, and 152 layers.

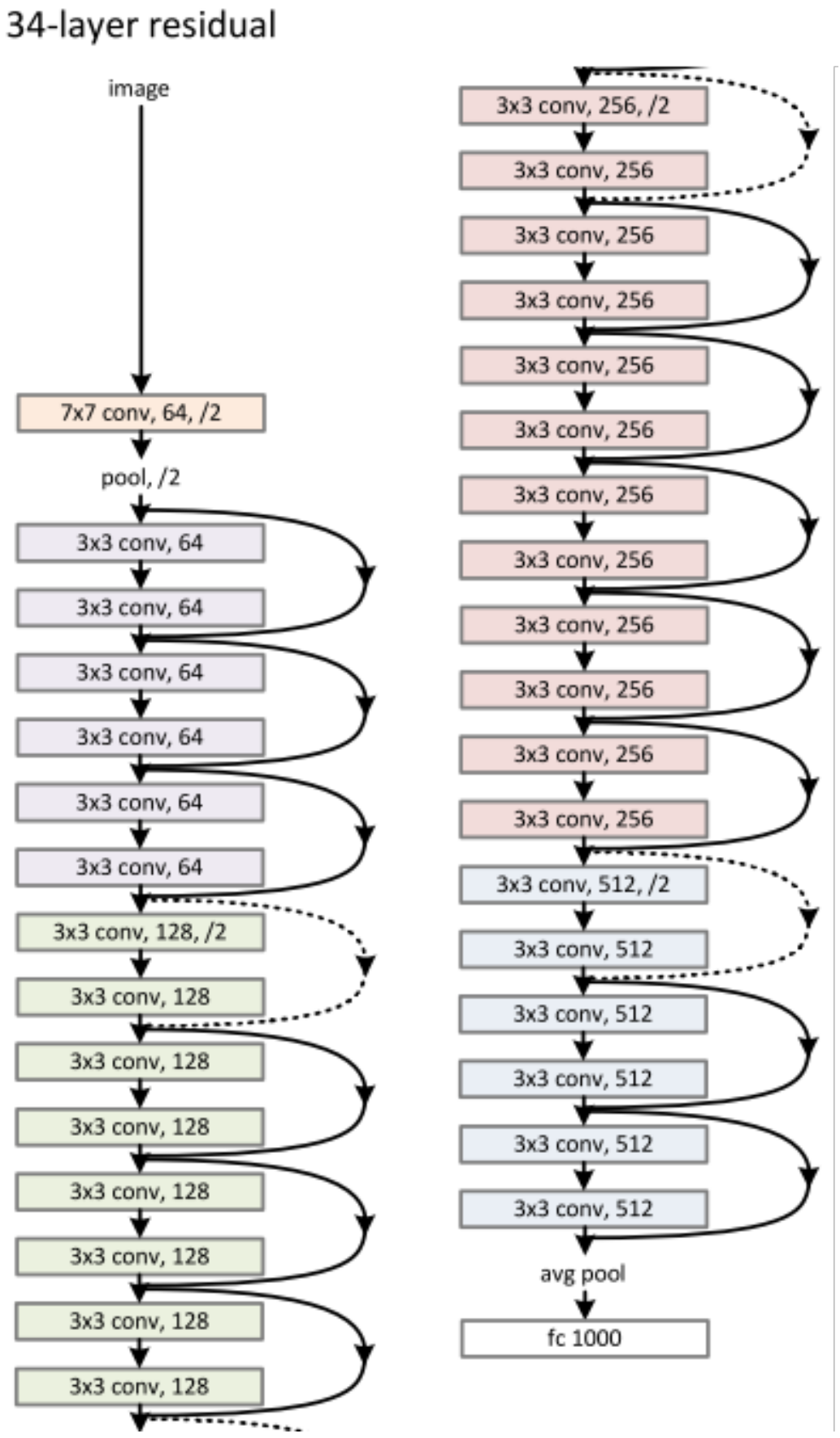


Figure 2.2: ResNet-34 architecture. Credits: He et al. [2015].

2.2 HDR and Deep Learning

High Dynamic Range Imaging has been researched in Deep Learning. Many papers are dedicated to creating HDR images from LDR input, for example, Yu et al. [2021] Nazarczuk et al. [2022]. Others work with HDR compression methods Feng et al. [2022].

2.3 HDR image semantic segmentation

A minimal amount of information about this topic was found. First, a paper about sky segmentation on HDR images is done via a graph-cut algorithm Dev et al. [2018]. Second, there is a paper "Domain adaptation of HDR training data for semantic road scene segmentation by deep learning" Weiher [2019].

3. Experiments

3.1 Creation of the dataset

From the beginning, the idea was to use Charles University's dataset of HDR images. The final model would have to detect the sky on these images of .exr format, so later, the sky could be automatically cut out and used in SkyGAN, another project of Charles University. Since we treat semantic segmentation as a supervised algorithm, these images would have to be manually annotated for training and validation. Therefore, it was clear that a custom dataset would have to be created. Nevertheless, a search of existing HDR datasets was performed as well.

3.1.1 Existing HDR datasets

Today's datasets for computer vision training are composed overwhelmingly of LDR images. The found datasets with HDR images are the Singapore HDR Whole Sky Imaging SEGmentation dataset (SHWIMSEG) S. Dev [2018] and the Cityscapes Cordts et al. [2016]. The former dataset was not created for Deep Learning purposes but for a graph-cut algorithm, which does not require much data Dev et al. [2018]. It contains 52 HDR images with a resolution of 500x500 centered on the sun. It does not contain any non-sky objects. Images have labeled clouds as 'one' and the sky as 'zero.' The latter dataset was created for Deep Learning and contained different images of the city taken during a car ride. Compared to other datasets, the sky class rarely appears Cordts et al. [2016]. However, the amount of annotated sky pixels is pretty high Cordts et al. [2016]. Moreover, according to the Cityscapes website, it is "Open sky, without leaves of tree. Includes thin electrical wires in front of the sky." Cityscapes [2022] Access to this dataset requires registration and providing a reason for usage. Based on the number of images or their content (lack of trees, horizon, only urban imagery, or focus on non-sky content), it was decided that these datasets would not contribute a lot to the project at hand.

3.1.2 Custom dataset

Here will be discussed the process of creating a custom dataset for deep learning.

Source

The created custom dataset consists of 8K (8192x4096) panoramic images from the HDR dataset by Charles University and a public asset library "Poly Haven." Haven [2022]
The former is photographs of the sky in different locations and times, primarily in nature. They are initially fish-eye stereographic-up images of 1024x1024 resolution that are later projected to equirectangular panoramas in 8K resolution.
PTGUI program was used to convert them to such panoramas. Here are the settings used in PTGUI. For input lens settings: Lens type is set to "Circular Fisheye," actual horizon field of view to 180 degrees, focal length to 17mm,

and the focal length multiplier was set to 0.636. For output, panorama settings: projection set to "Equirectangular (for spherical panoramas)," field of view to 360°x180°; image parameters: pitch 90°.
Because of these transitions, the lower part of equirectangular panoramas consists of pure blackness (zero values); there is no information. Also, because images are initially of low resolution, 1024x1024, and stereographic-up images have compressed horizon more than the zenith, unlike the equirectangular images, the resulting panoramas have the objects on the horizon out of focus 3.1.

Figure 3.1: Image with a blurry horizon.

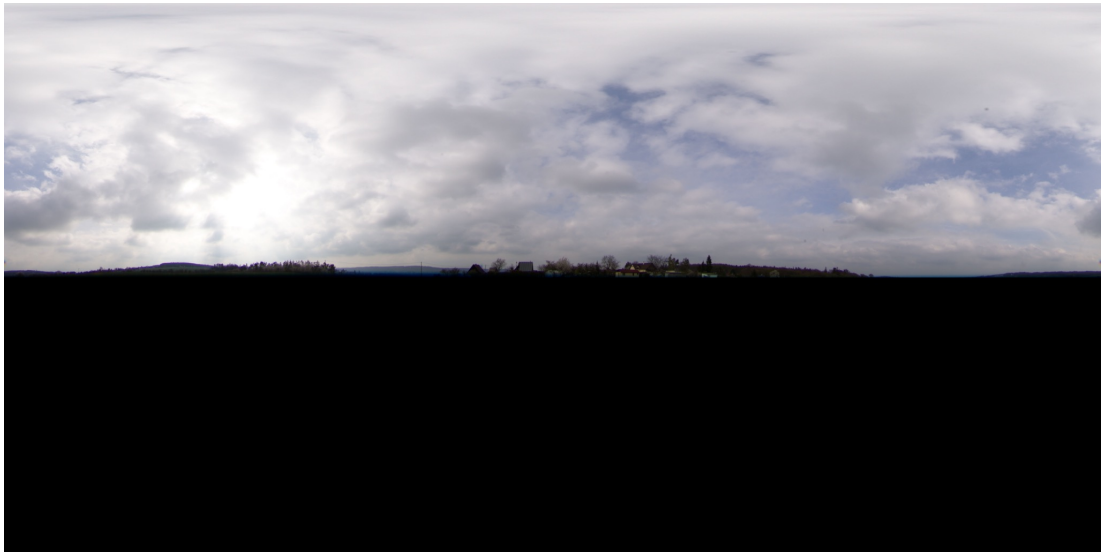
The latter dataset consists of high-quality HDRI panoramic images. They are taken in different locations and times, mostly in nature. It makes them suitable for the task. They can be downloaded in various resolutions, including 8K, and in both LDR .jpg and HDR .exr formats.

Annotation

The label-studio tool Heartex [2022a] was used to annotate (label) images. The tool accepts only LDR input, so the images in .jpg format were used for this purpose. All images were manually annotated using polygons Heartex [2022b]. The tool's output is COCO format in a .json file Consortium [2022] and a folder 'images' with all used images. Images were annotated with high precision, but keeping in mind that the final model will still show only an approximation of labeled data. The goal is to annotate the sky and avoid labeling non-sky pixels as the sky and making too risky annotations 3.3. For example, challenging and tiny closed sky regions between tree branches were ignored and treated as non-sky.

In this tool 2 classes were created: 1 = sky, 2 = non-sky. Everything that is not annotated, such as vast chunks of ground and horizon, results having a value of 0, meaning non-sky. The sky was labeled using polygons with the value of 1. Polygons with the value of 2 were used for objects in the sky, such as a fly or dirt

(a) Poly Haven image. Credits: Mischok [2021].



(b) Charles University's image.

Figure 3.2: Typical image examples

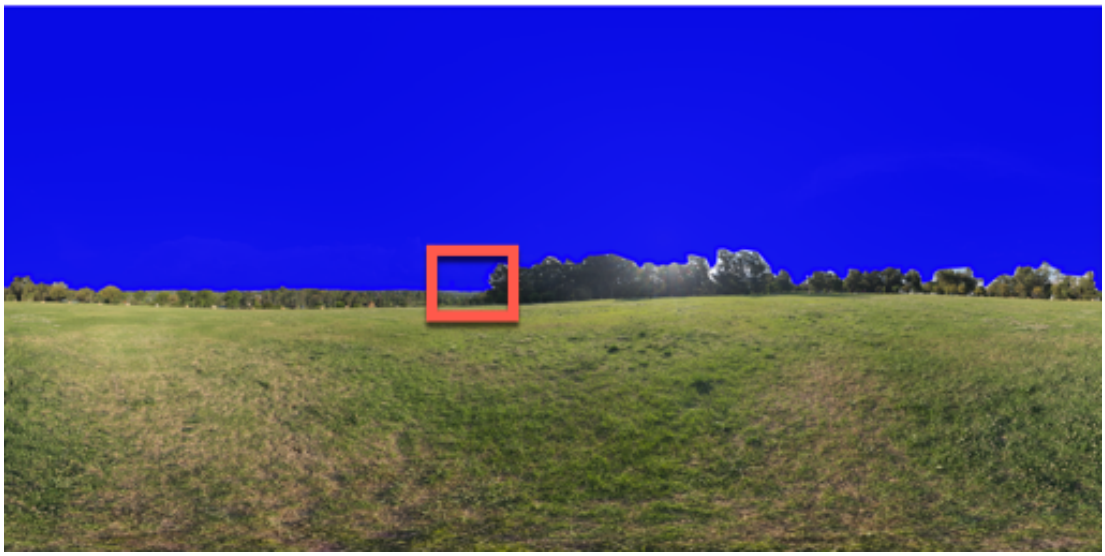
on the camera lens. Later in the code, the value of 2 is changed to 0. On average, annotating one 8K image took 30-40 minutes. The tool proved to be an adequate solution for manual annotation, even though sometimes moving polygons around would freeze it, but it can be due to the high resolution of images.

Updates

The custom dataset was expanded throughout work with more images from both datasets. Therefore, several different versions of the dataset will be referenced. The first version (v1 for short) contains 46 images from the university's dataset and 29 images from Poly Haven, resulting in 75 images. Both second (v2) and third (v3) versions contain 138 images, with 29 Poly Haven images and 109 from the university's dataset. Also LDR dataset was created. It is identical to v1,

but with several extra images taken on the phone. These extra images are HDR which are tone-mapped to LDR.

Figure 3.3: Example of non-annotated closed region of the sky.



(a) Annotated image, full size. Red square represents where it will be zoomed.

(b) Zoomed annotated image with visible polygon nodes.

Figure 3.4: Example of annotated image.

3.2 Architecture

3.2.1 ResNet

ResNet-34 was chosen as the backbone for training all the models. Experiments were done with deeper ResNets, but they showed worse results. That may be due to the small dataset size. Such case was also noticed in this work et al [2020], where ResNet-34 performed better than ResNet-50. The deeper the network, the more vast dataset it requires.
Another reason for choosing ResNet-34 was the speed of training. Many different experiments needed to be done with a limited time and hardware. ResNet-50 naturally has more trainable parameters, weighs more, and trains longer.

3.3 Methods

Here will be quickly discussed methods used in processing images before they are fed into a network. Exact implementations will be discussed in Models section 4.

3.3.1 Patching

Feeding a full 8K image as input to the network requires much memory, especially when a batch of such images is used. Cutting out smaller patches from images is a better method to work with high-resolution pictures. It is not only memory efficient because it is possible to load fewer data into memory, but also randomizing the order of such patches containing different information may make the model more robust, as it cannot predict the following input. Also, the original input resolution becomes irrelevant for the model, as the picture will be cut into predefined-sized patches. The dataset's patches were always divided into training and validation sets with a ratio of 75%/25% respectively in all three versions. Patches were randomly distributed. Later, patches and their masks are read from sets in random order to make the model more robust and not expect the same input.

3.3.2 Batch size

Batch size defines how many inputs will be fed to network simultaneously.

3.3.3 Preprocessing

Preprocessing functions in Machine Learning are used to change images' values to satisfy pre-trained model expectations and ensure knowledge transfer. The input is scaled accordingly if the original model was trained with float values in the range 0...1. The original ResNet-34 backbone trained on the "imagenet" dataset did not use any scaling He et al. [2015]. Therefore, preprocessing function from the "segmentation models" library (used in this work) does not do anything, as was confirmed by the library author in this forum discussion LeeJayHui [2019]. However, unprocessed LDR input with 8-bit colors always guarantees to have integer values in the strict range 0...255. Thus, not processing HDR input with

float values outside the 0...255 range can be wrong. Despite ResNet models initially not having preprocessing functions, frameworks such as Tensorflow still ensure some input normalization for ResNet models TensorFlow [2022b] to be consistent with other models they provide. Nevertheless, available preprocessing functions are primarily created for LDR input with 8-bit colors. For example, they divide by 255 to normalize the image into the 0...1 range, which is a maximal value for LDR images. Using such functions with HDR input is wrong. Since originally no scaling was done and different resources use different functions, it is unclear how vital scaling is for the model in this case. It was decided to preprocess HDR input our way.

Log transformation function

For this work, log transformation was used for processing the input. It was first introduced here Eilertsen et al. [2017] and then modified in SkyGAN project Mirbauer et al. [2022]. In this work was used the modified implementation. It roughly sets the input to -1...+1 range but can go overbound.

```
transform_cfg = dnnlib.EasyDict(
    # Multiplier value applied before the log transform
    input_mul = 1, # e.g. 2**8 would shift it 8 EV steps up
    # The epsilon constant for log-mapping input HDR images.
    log_epsilon = 1e-3,
    # separate the value space around 1.0 into two separate mapping
       functions
    # - x < 1: log(x) => expansion of the value space
    # - x >= 1: pow(x, 1./log_pow) => adjustable compression of the
       value space
    log_split_around1 = False,
    log_pow = 7.5,
    # Shift the value up/down (after the log transform) - can be used
       to ensure zero-mean
    output_bias = 2.5,
    output_scale = 1/2.2/2,
)

# Applies the log-transformation that converts linear HDR images to a
   form suitable for a
# neural network.
def log_transform(x):
    if transform_cfg.input_mul != 1.0:
        x = x * transform_cfg.input_mul
    x = x + transform_cfg.log_epsilon

    log_x = np.log(x)
    pow_x = np.power(x, 1./transform_cfg.log_pow) - 1.0

    if transform_cfg.log_split_around1:
        x = np.where(x < 1.0, log_x, pow_x)
    else:
        x = log_x
```

```
    return (x + transform_cfg.output_bias) * transform_cfg.output_scale

# Inverse of log_transform(x)
def invert_log_transform(y):
    y = y / transform_cfg.output_scale - transform_cfg.output_bias

    exp_y = np.exp(y)
    pow_y = np.power(y + 1.0, transform_cfg.log_pow)

    if transform_cfg.log_split_around1:
        y = np.where(y < 0.0/transform_cfg.input_mul, exp_y, pow_y)
    else:
        y = exp_y
    y = y - transform_cfg.log_epsilon

    if transform_cfg.input_mul != 1.0:
        y = y / transform_cfg.input_mul
    return y
```

Normalize function

This function was used to normalize input to the range 0...1.

```
def normalize(x):
    if np.max(x)-np.min(x) == 0.0:
        return x
    return (x-np.min(x))/(np.max(x)-np.min(x))
```

3.3.4 Augmentations

Augmentation is the process of changing the image somehow, usually automatically. For example, it is possible to change brightness, crop, rotate a picture, and do many other effects randomly. When the dataset is not extensive and diverse, feeding slightly different images to the network every epoch can improve the final model in generalizing predictions. Essentially, it increases the size of the dataset with newly changed images.
However, it is important to be mindful of them. Changing brightness, perspective, or other image property too much can harm the model. The model would learn only or primarily from extreme examples, which may not happen naturally. Therefore, limiting the extent of augmentation and how often it can happen is practical. In this work, the library "Albumentations" is used Buslaev et al. [2020]. Here is the list of used augmentations:

- **Horizontal Flip**. Mirrors the image. This augmentation happens 50% of the time, i.e., applied to 50% of all patches.
- **Shift, Scale, and Rotate**. For both scale and shift, the factor is limited to a range of -0.2...+0.2. The picture is rotated up to 25 degrees clockwise

or counter-clockwise(-25...+25 degrees). This augmentation happens 40% of the time. If scaling or shifting goes out of the border,
the padding "cv2.BORDER_REFLECT_101" is used. It repeats the border pixel in the pad instead of zero value, like in "same" padding.

- **Perspective**. It performs a random four-point perspective transform of the image Albumentations [2022]. The standard deviation is in the range between -0.1 and +0.1. It happens 20% of the time.
  Padding is "cv2.BORDER_REFLECT_101". This augmentation is an example of the need to be careful with extensive augmentations. When the factor is large, it creates random mistakes in the image.

Experiments with brightness and contrast augmentations showed that log transformation and normalization later negate these changes. Therefore, they were not used.

3.3.5 Callbacks

In this work two different callbacks were used during draining the model. One was saving model checkpoints. Throughout the duration of training, it was monitoring the validation IoU score. If at the end of some epoch the score increased, comparing to the previous best result, the model's weights are saved. Another callback used was TensorBoard. It was recording progress of training, loss value and IoU score. These logs are available in the attachment. TensorBoard was used only in model 2 and 3.

3.3.6 Generator

Working with HDR input requires more memory than working with LDR, as HDR images naturally weigh more. Using generators is a common way to work with heavy data. Instead of loading full training or validation set into memory, generators read only the necessary and pre-defined amount of images (a batch) from the directory, process it, and feed them to the network. Moreover, such generators have built-in augmentation tools to ease work with data. The problem is that available data generators work only with standard formats, such as .jpg, .png, and .tiff (or an interchangeable .tif).The .tiff file format works well for storing HDR data. However, the libraries that these generators use are limited by themselves. For example, the popular python "Pillow" ("PIL") library used in Keras Generator does not work with multidimensional float32 TIF files. Writing custom generators is a common way to solve such problems. That is why a custom Data Generator was implemented. The custom Data Generator is a class that extends the Sequence layer of Keras: 'class CustomDataGen(tf.keras.utils.Sequence)'. It implements the functions "__init__," (called once the generator is initiated), "__getitem__" (called when the model requests the next batch of images), and "on_epoch_end" (called at the end of an epoch, when both training and validation sets are fully processed). If a flag "shuffle" is set, then the order of images read is shuffled at the end of each epoch. Then the library "simpleimageio" ("sio") is used to open the float32 .tiff file and save it as a NumPy array. The "PIL" library opens the mask saved as a .png file. After that, both

the single patch and its mask are augmented, as described in 3.3.4. After the patch is augmented, it can be processed and normalized. It is crucial to augment the image before the log transformation and normalization. If augmentations are applied after normalization, values of images would be outside the range 0...1. Together with the generator, it is not even needed to save the augmented images. Different augmentation is applied every time the picture is loaded into memory.

3.3.7 Prediction with voting

When the image is cut to patches, these patches may overlap. It is generally a good thing. For example, In training it creates more images. However, it can also be used in prediction. Overlapping regions will be predicted several times. This can be used to improve final segmentation result. It improves it also because now final prediction on the pixel is based on much broader area around it.
However, main obstacle for this is merging patches together to create the image with original dimensions. For that a function was implemented called "reconstructMask". It expects predicted patches were created from top-left corner to bottom-right corner and are numerated consequently. This way it can know where the patch belongs in the original image. The function creates zero array with original dimensions. It goes through the patches with predictions, knowing to what part of image it belongs, and records how many times a pixel was voted to be a sky. If the pixel is voted a sky, one vote added, if non-sky, then one vote subtracted. In the end, it sets all pixels with result bigger than zero to be sky.

Figure 3.5: Example of improvement that voting prediction provides. Left: prediction without voting and overlapping. Right: prediction with voting.

Figure 3.6: Left: no voting, no overlap. The patch created sharp slope. This happens often. Right: voting makes the edges where patches were smoother.

Prediction with voting was used in models 2 and 3. However, their recorded validation results are not affected by this, as it is not applied during training. Moreover, training is done by randomly shuffling patches between sets and within them, making such method impossible to use.

Also, patching for prediction is made with ”valid” padding. It means, that most strides won't cover the full image. For prediction, stride of 100 pixels was used. It creates non-predicted line on the right edge and on the bottom of the image. In most cases only very small stride can cover the image fully: 8 or 16 pixels. It requires a lot of memory to create so many patches. On the hardware used for this thesis it was impossible. However, this can be solved by writing a custom function that patches images. It should cut one patch, save it on disk and delete from the GPU or RAM memory. Due to time limitations, it was not implemented in this work.

3.3.8 Illustration of predictions

After prediction is made, it is saved in code as NumPy array and also is saved on the disk as .png file for the illustration via the ”matplotlib” library. ”Matplotlib” library saves 0 values (non-sky) as purple color and 1 values (sky) as yellow. Later, to illustrate the predictions, Photoshop was used to put this .png file on top of .exr file with 20-50% opacity. Different opacity was used for different images, that is why pictures in figures may have different colors.

4. Models

Three different HDR models will be discussed here, plus one LDR model. The first HDR model was trained on dataset v1, the second on dataset v2, and the third on dataset v3, while the LDR model was trained on LDR dataset. The IoU score is the primary metric in all models. ResNet-34 is used as a network architecture for all three models. ADAM optimizer with a default learning rate of 0.001 was used everywhere.

4.1 Model 1

4.1.1 Ground

Dataset v1 was created for this model. As discussed before, the university's images have nil information that starts in the middle of them, at 2048 pixels. Dataset v1 contained many nil information patches with pitch blackness, as nothing was cut out from the original images.

4.1.2 Patches and statistics

Images were cut into square patches of resolution 256x256, overlapping with 100 pixels stride. They also are cut using "valid" padding, meaning the patch is not created if it does not fit fully in the image. Here is the total number of patches created, and their content is divided into training and validation sets. The numbers given are amount of pairs (patch, mask).
Training set:

- Total: 175500
- Only Sky: 74998 (42.7%)
- Only Ground: 85988 (49%)
- Both Sky and Ground (Horizon): 14514 (8.3%)

Validation set:

- Total: 58500
- Only Sky: 24789 (42.4%)
- Only Ground: 28767 (49.2%)
- Both Sky and Ground (Horizon): 4944 (8.4%)

Note that we have approximately the same percentage of patches showing both ground and sky, only showing the ground and solely containing the sky between training and validation sets. This is true for all created datasets. The batch size of 128 was used for training Model 1.

4.1.3 Processing

As used in the SkyGAN project Mirbauer et al. [2022], the only input processing applied was the log transformation function.

4.1.4 Results

The loss function used for this model is a sum of Binary Cross-Entropy and Jaccard loss (IoU loss). Despite the imbalanced dataset, the binary cross-entropy performed adequately. The model was trained without a break for 100 epochs on dataset v1 without augmentations. This model achieved a high IoU score of 99.92% and a loss of 0.021 in the validation set. Lack of augmentations may have contributed to such a high IoU score and pretty high binary cross-entropy and Jaccard loss. Visually the model shows promising results. However, as other models will show, they can be better.

Model predicts sky and ground well enough. However, it is not good at noticing small non-sky objects in the sky and wires. A batch size of 128 may have negatively affected the generalization power of the model. Training takes a lot of time, around 10 days. With other deeper architectures it may take even longer. The model was trained on a single GPU RTX A6000 with 48GB of memory, which allowed it to fit a big batch size. After the 70th epoch (almost a week of training), the model started improving itself much more often. If before improvement in valid IOU score happened every 20 epochs, now it started happening every two. Note: predictions showed for this model are not improved through voting method with overlapping patches described in the previous chapter.
Since the training set does not include augmented images and contains only "valid" patches, it does not contain an image with something non-sky on top. Moreover, as described before, during manual labeling, closed regions of the sky were not annotated as the sky. That is why the model learned to avoid labeling the actual sky when the patch has a non-sky object on top. However, at later epochs, it improved and learned to label such cases better. 4.1

(a) Model 1 at epoch 51 fails to segment the patch of sky with an object above

(b) Model 1 at epoch 100 got better segmenting the patch of sky with an object above.

Figure 4.1: Model 1 is struggling predicting sky, when on top there is a non-sky object.

Patches containing dirt on them, tiny dots, or an effect because of the clouded lens would be a big problem for the model initially. It would label the whole or most of the patch as non-sky. It may be because patches of sky in the dataset were mostly always clear. In dataset v1, only a couple of images have dirt. Later epochs the model 1 learned to predict such patches nicely. 4.2

(a) Model 1 at epoch 31 struggles to predict a patch with dirt dot.

(b) Model 1 at epoch 100 successfully segments the patch of sky with dirt. This, however, does not happen that often.

Figure 4.2: Model 1 results on predicting dirt.

Wires are a big problem for this model. As we will see later, other models struggle with segmenting sky areas with wires too, but their results are better. 4.3

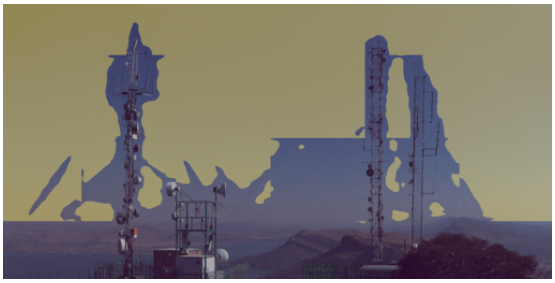
Fog - a problem that makes the model think that foggy area is the sky. The model may be mixing the fog with clouds. This problem remains in future models. 4.5

The dataset contains some very dark images. It may be why sometimes model 1 predicts very dark areas as a sky. 4.6

(a)

(b)

Figure 4.3: Model 1 at epoch 100. Bad performance on wires



(a) Model 1 detecting wires at epoch 23.

(b) Model 1 detecting wires at epoch 100.

Figure 4.4: Model 1 at epoch 23 and 100 detecting wires.

(a)

(b)

Figure 4.5: Model 1 at epoch 100 predicts ground covered with fog as a sky.

(a)

(b)

Figure 4.6: Model 1 at epoch 100 predicts some dark parts as a sky.

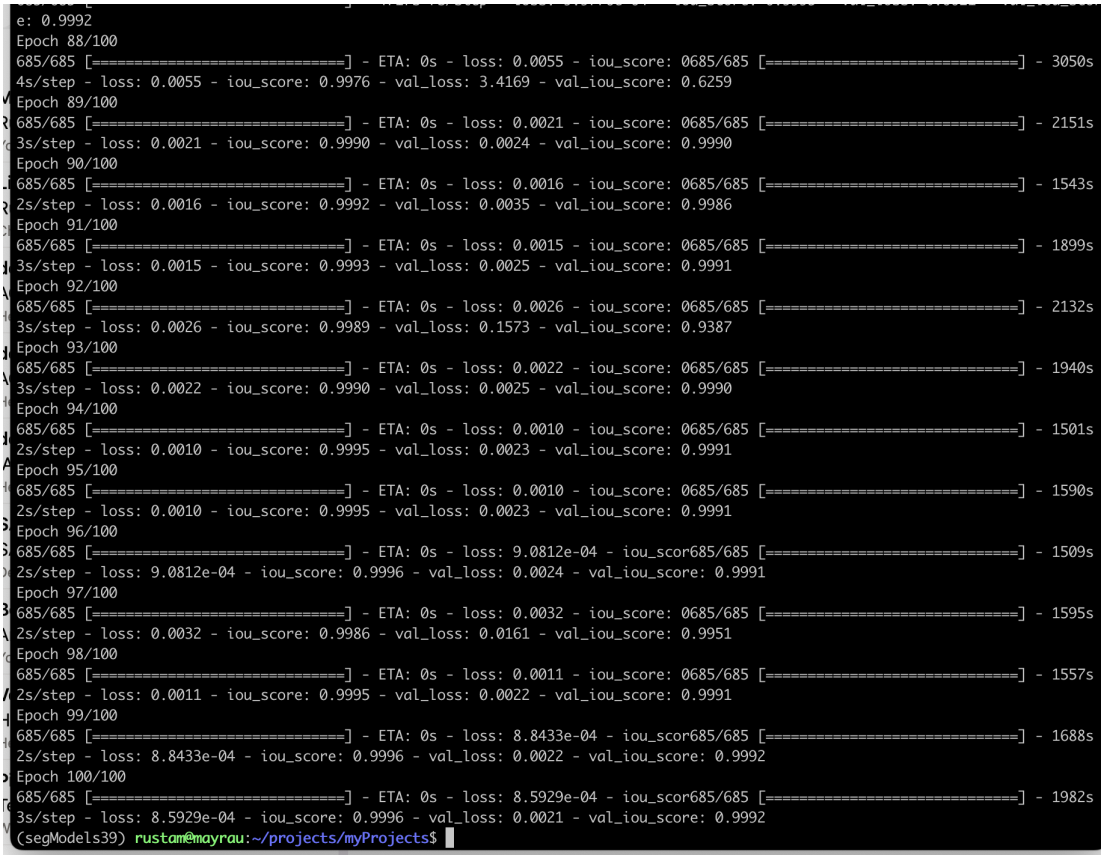


Figure 4.7: Model 1 recorded performance at last epochs.

4.2 Model 2

4.2.1 Ground

Dataset v2 was used in training this model. It contains a minimal amount of ground images. It was decided that training the model on many negative examples might be unnecessary. However, it was discovered that this assumption was false, and the model performed much worse predicting ground. Since the nil information starts at 2048's pixel, all images were cut in half from that place, leaving only the upper part.

4.2.2 Patches and statistics

It was decided to use patches of size 224x224, as it was done in the original ResNet-34 model pre-trained on the "imagenet" dataset He et al. [2015]. Patches overlap with 100 pixels stride. This time "same" padding was used, meaning if the patch does not fit fully in the image, it is still created with zero values where it goes over the border.
Training:

- Total: 167669
- Only Sky: 127863 (76.3%)
- Only Ground: 6276 (3.7%)
- Both Sky and Ground (Horizon): 33530 (20%)

Validation:

- Total: 69976
- Only Sky: 53796 (76.9%)
- Only Ground: 2203 (3.1%)
- Both Sky and Ground (Horizon): 13968 (20%)

A batch size of 64 was used in training.

4.2.3 Processing

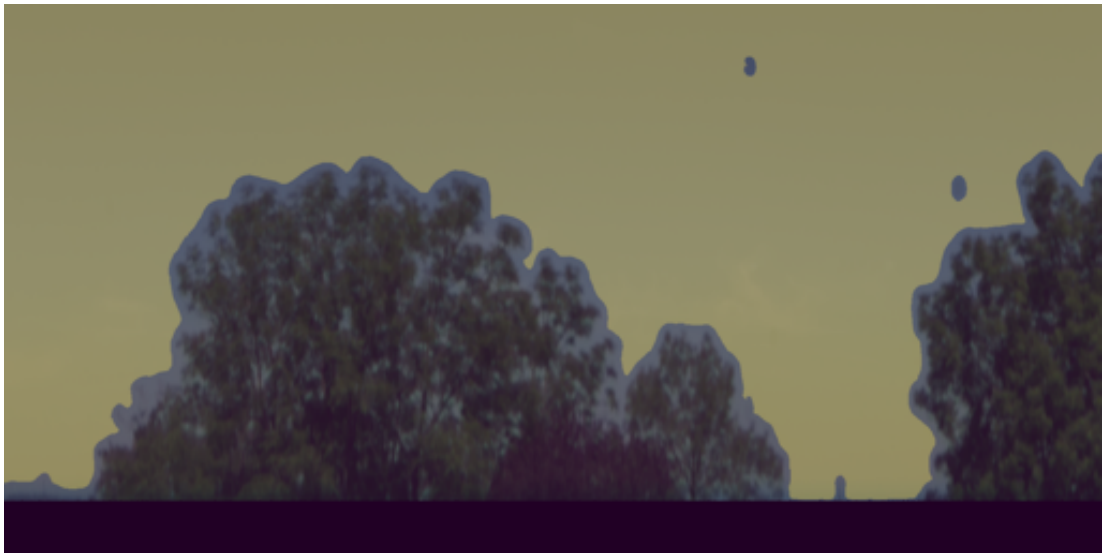
In dataset v2, it was decided to normalize the input to the range of 0...1. It is done by applying the log transformation function, then the tanh function, and then the "normalize" function. The tanh function normalizes input to the -1...1 range, "fixing" the range of log transformation if it goes overbound. Then, applying "normalize" gives the desired range.

4.2.4 Results

Model 2 was trained using the focal loss with the standard gamma of 2.2. This model achieved a validation focal loss of 0.000792 and IoU 97% and showed excellent results in segmenting the sky and noticing non-sky objects.

Model 2 often detects very small dirt dots that all other trained models miss. 4.8

On Figure 4.9 is the example of how model 2 notices blurry stains created from the dirty camera lens.



(a) Model 2 predicts horizon and notices stains.

(b) Model 2 detected a blurry stain.

(c) How the stain looks like on the original image.

Figure 4.9: Model 2 can detect blurry stains.

Model 2 is also more sensitive to wires, than other models. However, it is still not perfect 4.10c. Detecting wires stays one of the hardest tasks for models. 4.10

(a) Model 2 successfully detects wires and avoids them.

(b)

(c) Model 2 struggles to correctly predict some wires.

Figure 4.10: Model 2 results on wires.

To understand why we must look at the used dataset and loss function.

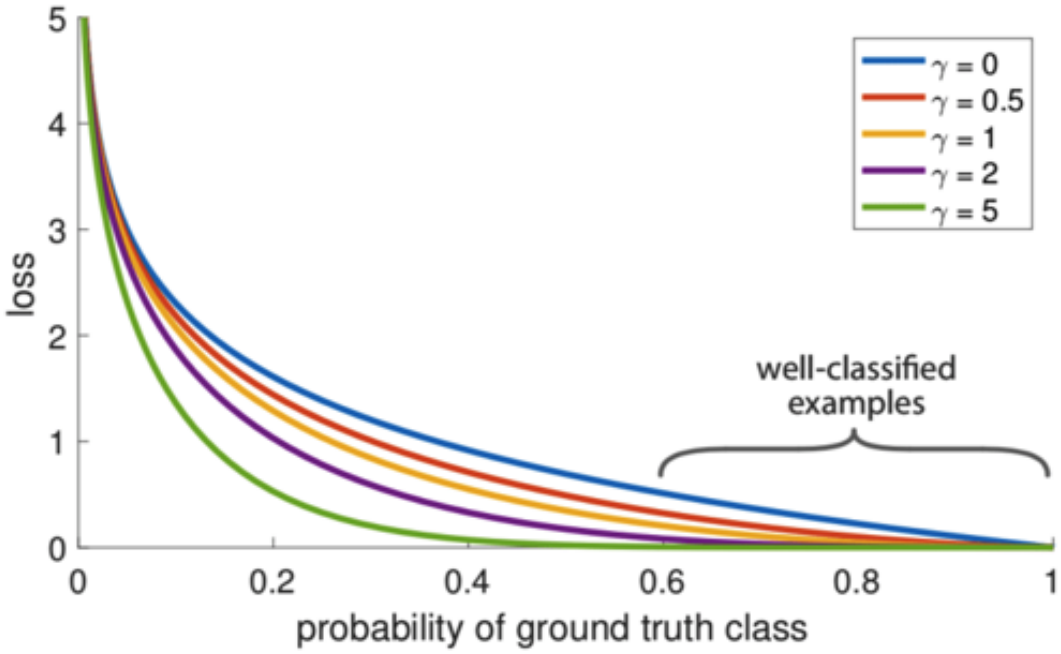
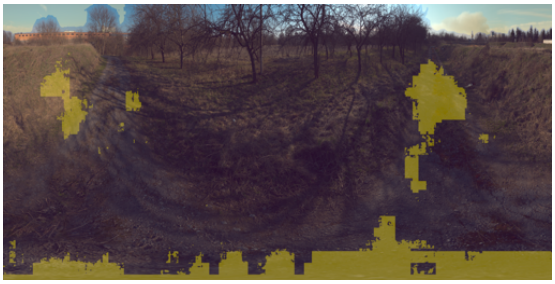


Figure 4.11: Focal loss function results depending on used gamma.

On the graph 4.11, focal loss with gamma=0 equals binary cross-entropy. For illustration, predicting the pixel as the sky has a probability of 60% in the model, resulting in 0.5 cross-entropy loss. The higher the loss, the more the model focuses on predicting this pixel correctly to decrease this loss. However, using focal loss with gamma=2, we can see that the model that predicts a pixel as a sky with 60% sureness would result in a shallow loss, less than 0.1. If the loss is small and the probability is high, training to make such predictions with even more certainty is useless - a loss would not become much smaller. The model would not try as hard to lower loss while predicting the sky and instead focuses on another "harder," unsure class (non-sky). In dataset v2, on which model 2 was trained, the amount of ground pictures is meager (3-4% of patches contain only ground, 20% contain both sky and ground). With the focal loss, model 2 focuses more on "hard" predictions (the horizon and non-sky objects in the middle of the sky, such as dirt on the lens or a fly), not big chunks of ground. While model 2 performs very well predicting sky and horizon and is sensitive to non-sky objects in the middle of the sky, it performs inferior predicting just ground 4.12. Therefore, this model would suit predicting images from Charles University's dataset, as they do not contain lower parts with the ground.

(a)



(b)

Figure 4.12: Model 2 at epoch 50 struggles to predict the ground as non-sky.

This model had initial weights of model 1. The assumption was that transfer learning from the model already trained on HDR input would be more beneficial than the model trained on the LDR ”imagenet” dataset. This model was trained for fifty epochs in total.

The model was stopped every 10 or 20 epochs. The gaps in figures represent these pauses.

(a)

(b)

(c) Model 2 detects 3 dots and avoids them.

Figure 4.8: Model 2 is very sensitive to small dirt dots on the sky.

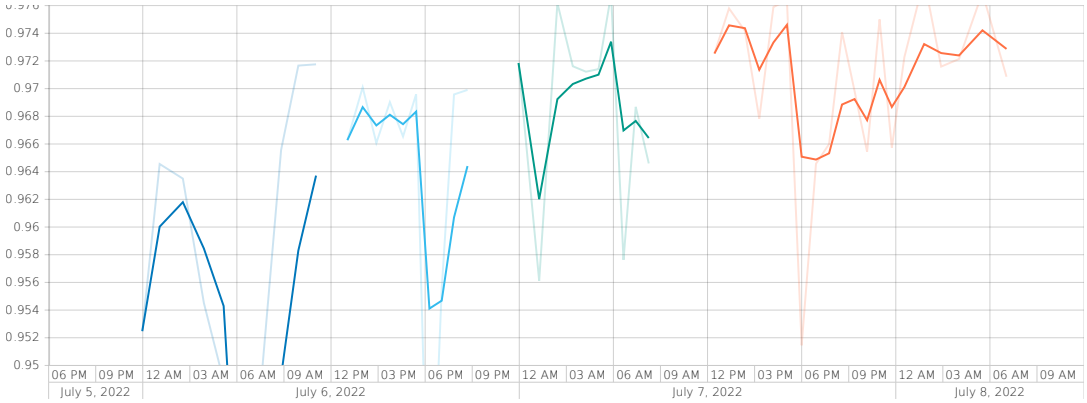


Figure 4.13: Model 2 logs. Validation IoU vs iterations.

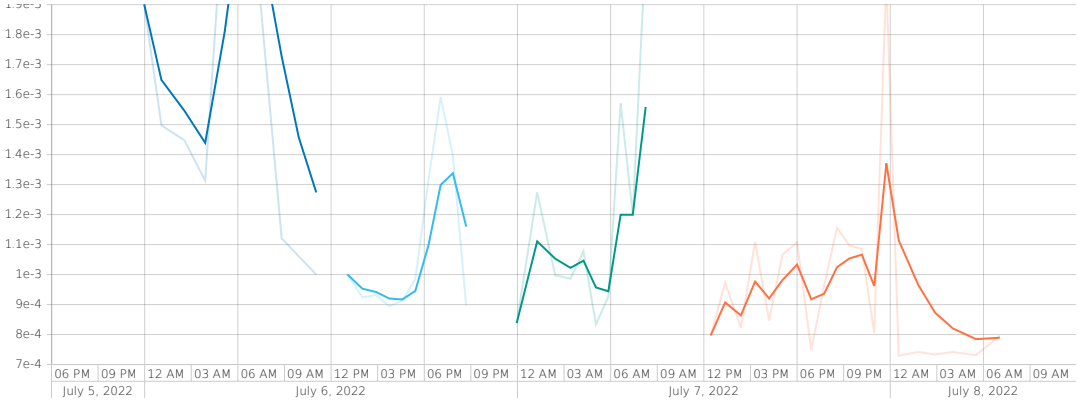


Figure 4.14: Model 2 logs. Validation loss vs iterations.

Figure 4.15: Model 2 was trained with augmentations. Results that model 1 was struggling with are easily achieved in model 2.

4.3 Model 3

4.3.1 Ground

Dataset v3 was used in training this model. Here, the attempt was made to solve the problem of model 2 predicting the ground pixels. The ground from Poly Haven images was not cut out in dataset v3, hoping it would help the model, but the lower part with nil information from the university images was cut out.

4.3.2 Patches and statistics

224x224 patches used and 100 pixels stride. **Training**:

- Total: 202904
- Only Sky: 128482 (63.35%)
- Only Ground: 40491 (19.95%)
- Both Sky and Ground (Horizon): 33931 (16.7%)

Validation:

- Total: 82292
- Only Sky: 53177 (64.6%)
- Only Ground: 15140 (18.4%)
- Both Sky and Ground (Horizon): 13975 (17%)

The batch size was 64 images.

4.3.3 Processing

In model 3, tanh function is not applied. After log transformation, the input is normalized with ”normalize” function to range of 0...1.

4.3.4 Results

Like in model 2, focal loss with gamma 2.0 was used. Model 3 initial weights were that of model 1. It was trained for 50 epochs in total. However, the first 30 epochs were enough for the model to converge. The last 20 epochs did not show improvement. On the contrary, it started being worse 4.16.

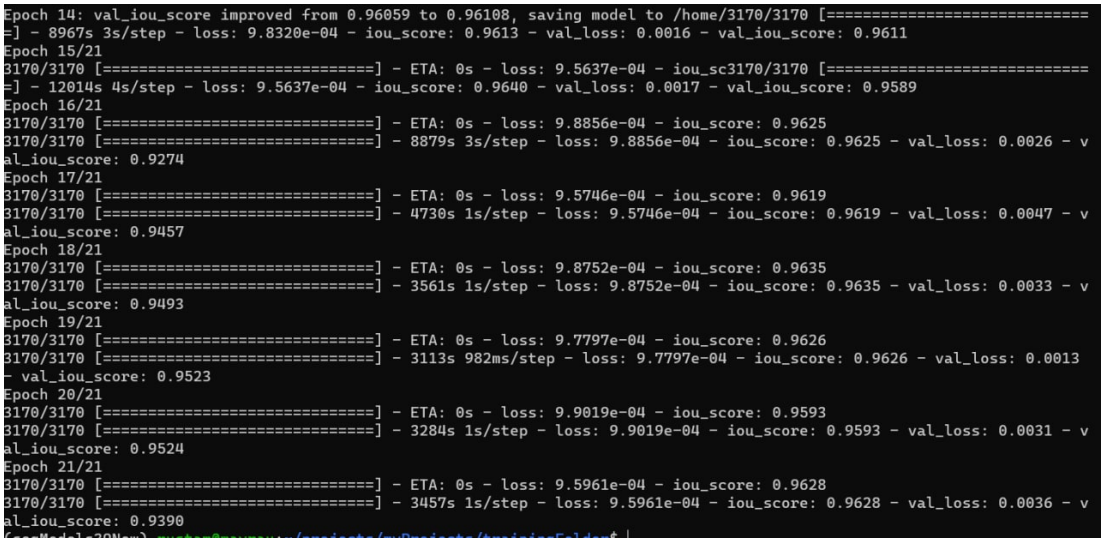


Figure 4.16: Model 3 last recorded logs. Loss and IoU score became worse.

The training was interrupted after the first 29 epochs. After the training was renewed for extra 21 epochs, logs recorded only the first nine epochs. The reason for that is unclear. However, the checkpoint saved midsts training (a model checkpoint after some epoch with the highest-so-far IoU score) at renewed epoch 14 showed a validation loss of 0.001619 and IoU 96.1%. The validation results of the final epoch (50th total, or 21st renewed) degraded: loss of 0.0036 and IoU 93.9%. Therefore, the model's checkpoint at total epoch 43 will be used to show the results.

In general, model 3 performs better than model 1 in every way. It is more mindful of thin wires and other non-sky objects, and its edges are smoother and closer to non-sky than those of model 1.

Compared to model 2, its results are not as sensitive for the horizon and non-sky objects in the sky.

Nonetheless, it predicts the ground generally well, meaning this problem is fixed. Some mistakes happen, though 4.18b. It predicts ground worse than model 1, but much better than model 2.

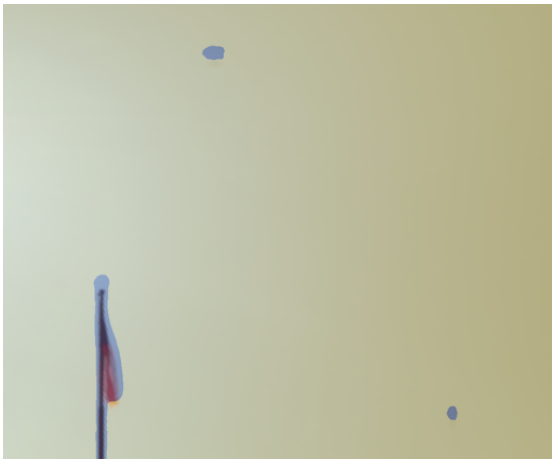
(a) Model 3 successfully detects ground.

(b) Model 3, however, struggles predicting ground in some images. But amount of mistakes is much less than in model 2.

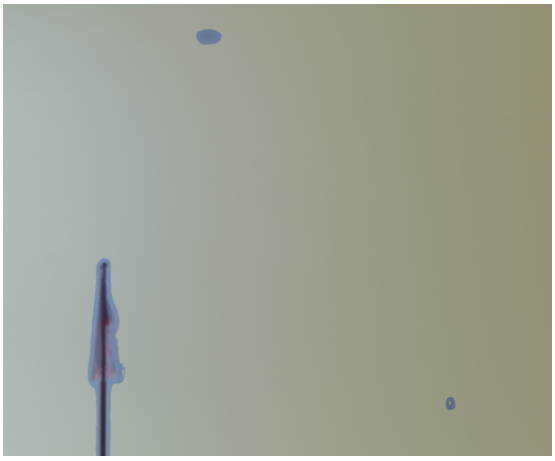
Figure 4.18: Model 3 results predicting ground.

Model 3 is better at predicting foggy images. It is less sensitive to it than model 2 and shows better results than model 1 4.19.

However, it was noticed that model 3 sometimes shows strange results predicting images from the Charles University dataset 4.20. It correctly detects edges and even dirt stains or dots, but outputs prediction slightly higher than all of them are located. Model 2 on the same image shows much better results.



(a) Model 3 makes strange mistakes.



(b) Model 2 predicts this part of image perfectly.

Figure 4.20: Model 3 shows worse results than model 2.

Model 3 is recommended for prediction in images containing a large portion of ground, such as ones from the Poly Haven.

(a)

(b)

Figure 4.17: Model 3 is not as sensitive to wires as model 2.

(a)

(b)

Figure 4.19: Model 3 results predicting foggy parts.

(a)

(b)

Figure 4.21: Different model 3 predicting results.

(a)

(b)

Figure 4.22: Other different model 3 predicting results.

(a)

(b)

Figure 4.23: Illustration of how worse results are in model 3 at epoch 50. Possibly, after 40th epochs it started overfitting.

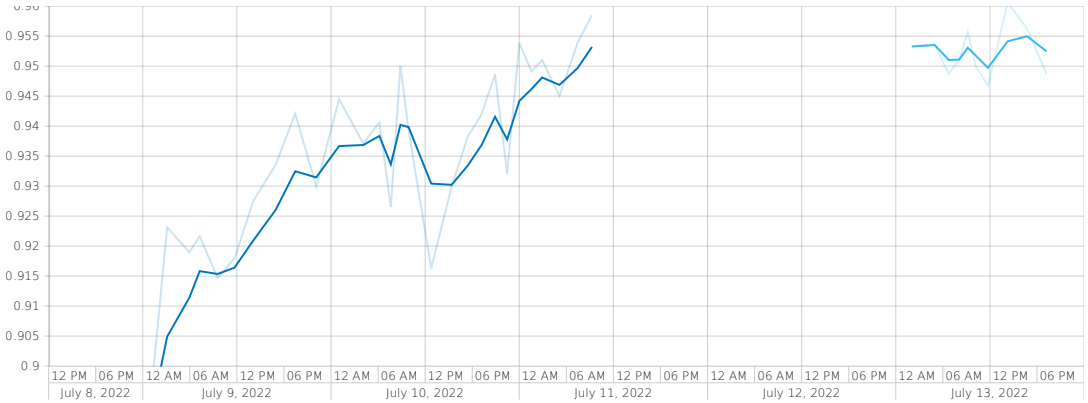


Figure 4.24: Model 3 logs. Validation IoU score vs iterations.

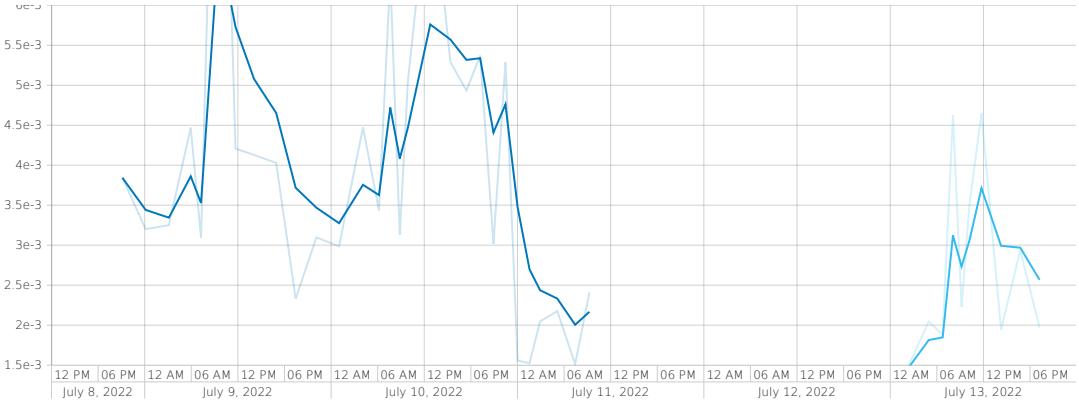


Figure 4.25: Model 3 logs. Validation loss vs iterations.

4.4 Model LDR

4.4.1 Ground

Images were not cut.

4.4.2 Patches and statistics

Patches of size 512x512 were created in the LDR dataset.
Training:

- Total: 7890
- Only Sky: 2710 (34.35%)
- Only Ground: 3772 (47.8%)
- Both Sky and Ground (Horizon): 1408 (17.85%)

Validation:

- Total: 2630
- Only Sky: 967 (36.8%)
- Only Ground: 1200 (45.6%)
- Both Sky and Ground (Horizon): 463 (17.6%)

Batch size was 128. Initial training images were also of 8K resolution.

4.4.3 Processing

Input was normalized to the range of 0...1. LDR images were divided by 255.

4.4.4 Results

LDR model was trained for 100 epochs with a combined loss of Binary Cross-Entropy and Jaccard. Augmentations were used. Unfortunately, final result of 100th epoch was not saved due to an error. But there was a model checkpoint created at epoch 98. That is what will be used to show the results.

Model trained on LDR shows the worst results out of all models.

It makes mistakes with ends of branches 4.26.

(a)

(b)

Figure 4.26: Model LDR does does not detect thin edges of branches.

It does not notice wires 4.27.

(a)

(b)

Figure 4.27: Model LDR fails to notice wires.

(a)

(b)

Figure 4.28: Other examples of model LDR failingto notice wires.

It does not notice dirt.

Figure 4.29: Model LDR cannot detect tiny dots and stains on images. It predicts them as sky.

It is less prone to noticing fog and smoke as well. In this case it works as a benefit as it makes less mistakes.

(a)

(b)

Figure 4.30: Model LDR results on predicting foggy parts.

Some random mistakes predicting ground may happen as well.

Figure 4.31: Model LDR predicts ground as a sky.

However, in general predictions can be considered not that bad 4.32.

(a)

(b)

Figure 4.32: General examples of predictions made by model LDR.

4.4.5 Comparison between LDR and HDR models

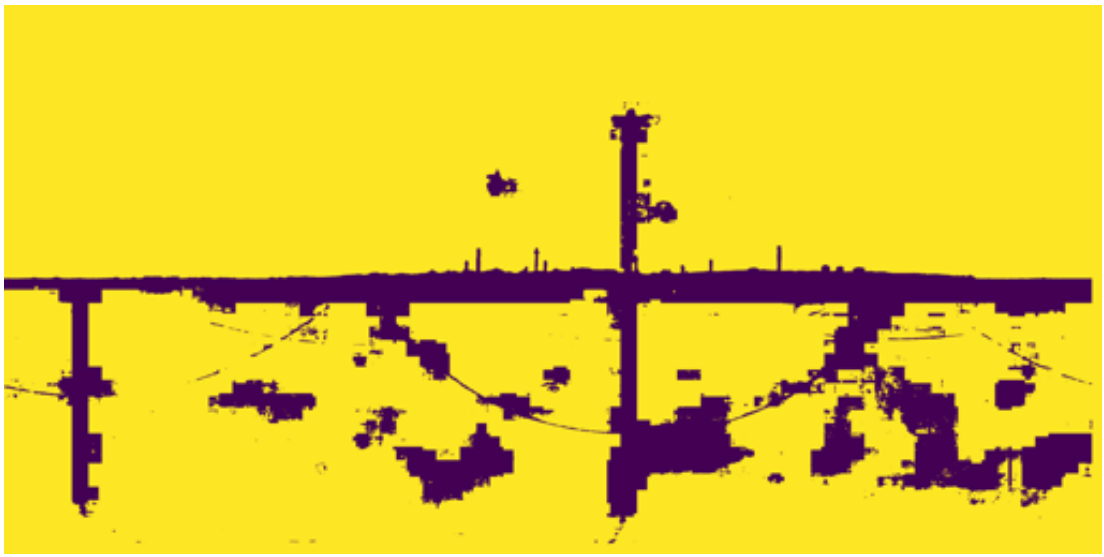
Some experiments were done by feeding processed HDR image to LDR model and feeding processed LDR image to HDR model. General suggestion is to not make such predictions. Models are very sensitive to the input they get. Both models produce a lot of mistakes in these cases.

LDR model in general can adequately predict HDR input normalized to the range of 0...1. However, such results cannot be used in practice. HDR models can also predict LDR image if it is processed correctly. Usually the result is "clearer" than that of LDR model.

(a)

(b)

Figure 4.33: HDR model 1 predicting LDR input.



(a)

(b)

Figure 4.34: HDR model 2 predicting LDR input.

(a)

(b)

Figure 4.35: LDR model predicting HDR input.

5. Code

Python was chosen as the main programming language for this work. It is one of the most popular languages for machine learning and has vast amount of libraries, frameworks and APIs. One of the main frameworks here was Tensorflow TensorFlow [2022a] and Keras Keras [2022]. Jupyter notebook Jupyter [2022] is the main IDE used in writing the code and working on experiments. The provided in attachments code is in the Jupyter notebook. The very useful used library was "segmentation models" "qubvel" Iakubovskii) [2022]. Conda Conda [2022] was used to create python virtual environments.

The code is available in the gitlab: `https://gitlab.mff.cuni.cz/fadeevr/semantic-segmentation-sky-hdr`

5.1 User guide

This is a guide how to replicate the experiments and prepare the same programming environment as used in this thesis. While most the given commands can be run in Windows, note that some suggestions are written for UNIX OS.

1. Install Conda
   - Installation guide [1]
   - Create virtual environment with Python 3.9

     ```
     conda create -n envName python=3.9
     ```

   - Active the environment

     ```
     conda activate envName
     ```

   - Make sure you use Python from the environment.

     ```
     pip -V
     which pip
     which python
     ```

     It should show path to conda envName environment
   - To forbid pip seeing global Python librarise use this:

     ```
     export PYTHONNOUSERSITE=1
     ```

2. Install CUDA and Tensorflow
   - CUDA is required for running the code on GPU.
   - Installation of CUDA is a very difficult process in most cases. It will require some tuning from the user's side.

[1] `https://docs.conda.io/projects/conda/en/latest/user-guide/install/index.html`

- Follow this instruction on TensorFlow website. [2]. It also contains instruction on how to install CUDA.
- The instruction requires exporting LD_LIBRARY_PATH to conda's path. That can cause this problem "libtinfo.so.6: no version information available (required by /bin/bash)". This is a solution:

```
rm ${CONDA_PREFIX}/lib/libtinfo*
ln -s /lib/x86_64-linux-gnu/libtinfo.so.6
    ${CONDA_PREFIX}/lib/libtinfo.so.6
```

3. Jupyter Notebook
   - Install Jupyter notebook.

```
pip install notebook
```

   - Create Jupyter Kernel connected to Conda environment.

```
pip install ipykernel
python -m ipykernel install --user --name
    myenvKernel --display-name "Python
    (myenvKernel)"
```

   - Go to the folder with Jupyter Notebook and open it:

```
jupyter notebook
# If on server through ssh, use this instead:
jupyter notebook --port=31888 --ip=0.0.0.0
    --no-browser
```

   - Web page with Jupyter Notebook should open.
4. Open the code
   - Click on the .ipynb file.
   - Choose Kernel that was created for Conda environment. It should be visible in the menu on top of the notebook: Kernel - Change Kernel.
   - Uncomment pip commands in the first cell (Choose the lines and press CTRL+/). It is necessary libraries that need to be installed via pip.
   - To execute the cell, press Shift+Enter.
   - Next two cells define functions and a custom generator.
   - The cell after that is the place where different paths should be defined depending on the user's system. Parameters, such as patch size, can be changed. Flags should be set to enable training, prediction or creation of the dataset.
   - Next cells depend on the flags (booleans). They execute training, prediction or creation of the dataset.

[2] https://www.tensorflow.org/install/pip

Please note, that while this instruction should fully satisfy all requirements to run the code fully, some problems may arise which are specific to the user's system.

5.2 TensorBoard and logs

TensorBoard should be installed together with TensorFlow. To open logs, use this command:

```
tensorboard --logdir=path/to/logs
```

The logs folder contains many logs from different experiments. When refered to the whole folder, all of them will be opened. However, some sub folders were created with compilations of only selected logs. For example, "only_model2" which contains only logs made during training of model 2.

Conclusion

Three semantic segmentation models trained on HDR input were created. One of this models (model 2) showed great results at predicting the sky and hard non-sky elements of pictures. All three trained models showed much better results than trained LDR model. This proves initial hypothesis, that HDR input does improve the network's abilities. Also, experiments showed that HDR and LDR models are not interchangeable. Predicting LDR images with HDR models does not improve results. It is better to use the model that was trained specifically for such input. Further experiments and other improvements of HDR models are possible. For example, another optimizer may improve predictions even more. Also, a learning scheduler, which changes learning rate throughout the training, may be a good idea. Experiments showed that the content of the dataset is of high importance. All models struggled correctly predicting images with fog or smoke in them. Adding more such images into training may solve the problem. On the other hand, the dataset contained a lot of images with wires, but they were one of the main problems for all models. This may indicate that bigger dataset could be needed.
In this work also the pipeline was provided that allows to point to the HDR image of .exr format, process it for the model and predict.

Bibliography

Albumentations. Albumentations documentation, perspective, 2022. URL https://albumentations.ai/docs/api_reference/augmentations/geometric/transforms/#albumentations.augmentations.geometric.transforms.Perspective. Accessed: 2022-07.

Y. Bengio and Yann Lecun. Convolutional networks for images, speech, and time-series, 11 1997. URL http://yann.lecun.com/exdb/publis/pdf/lecun-bengio-95a.pdf.

Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A. Kalinin. Albumentations: Fast and flexible image augmentations. *Information*, 11(2), 2020. ISSN 2078-2489. doi: 10.3390/info11020125. URL https://www.mdpi.com/2078-2489/11/2/125.

Cityscapes. Cityscapes website, labeling policy, 2022. URL https://www.cityscapes-dataset.com/dataset-overview/#labeling-policy. Accessed: 2022-07.

Conda. Conda website, 2022. URL https://docs.conda.io/en/latest/. Accessed: 2022-07.

COCO Consortium. Coco dataset, data format, 2022. URL https://cocodataset.org/#format-data. Accessed: 2022-07.

Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding, 2016. URL https://arxiv.org/abs/1604.01685.

et al. David Duque-Arias. On power jaccard losses for semantic segmentation, 2021. URL https://hal.archives-ouvertes.fr/hal-03139997.

Li Deng and Dong Yu. Deep learning: Methods and applications, May 2014. URL https://www.microsoft.com/en-us/research/publication/deep-learning-methods-and-applications/.

Soumyabrata Dev, Florian M. Savoy, Yee Hui Lee, and Stefan Winkler. High-dynamic-range imaging for cloud segmentation, 2018. URL https://arxiv.org/abs/1803.01071.

Gabriel Eilertsen, Joel Kronander, Gyorgy Denes, Rafał K. Mantiuk, and Jonas Unger. Hdr image reconstruction from a single exposure using deep cnns. *ACM Trans. Graph.*, 36(6), nov 2017. ISSN 0730-0301. doi: 10.1145/3130800.3130816. URL https://doi.org/10.1145/3130800.3130816.

Rongyu Zhang et al. Comparison of backbones for semantic segmentation network, 2020. URL https://iopscience.iop.org/article/10.1088/1742-6596/1544/1/012196/pdf.

Chen Feng, Zihao Qi, Duolikun Danier, Fan Zhang, Xiaozhong Xu, Shan Liu, and David Bull. Enhancing hdr video compression through cnn-based effective bit depth adaptation, 2022. URL https://arxiv.org/abs/2207.08634.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Mark Hamilton, Zhoutong Zhang, Bharath Hariharan, Noah Snavely, and William T. Freeman. Unsupervised semantic segmentation by distilling feature correspondences, 2022. URL https://arxiv.org/abs/2203.08414.

Poly Haven. Poly haven website, hdris, 2022. URL https://polyhaven.com/hdris. Accessed: 2022-07.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. URL https://arxiv.org/abs/1512.03385.

Inc. Heartex. Label studio website, 2022a. URL https://labelstud.io/. Accessed: 2022-07.

Inc. Heartex. Label studio, semantic segmentation with polygons, 2022b. URL https://labelstud.io/templates/image_polygons.html. Accessed: 2022-07.

Cambridge in Colour. Bit depth tutorial. https://www.cambridgeincolour.com/tutorials/bit-depth.htm, 2022a. Accessed: 2022-07.

Cambridge in Colour. Dynamic range in digital photography. https://www.cambridgeincolour.com/tutorials/dynamic-range.htm, 2022b. Accessed: 2022-07.

Shruti Jadon. A survey of loss functions for semantic segmentation. In *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*. IEEE, oct 2020. doi: 10.1109/cibcb48159.2020.9277638. URL https://doi.org/10.1109%2Fcibcb48159.2020.9277638.

Jupyter. Jupyter website, 2022. URL https://jupyter.org/. Accessed: 2022-07.

Keras. Keras website, 2022. URL https://keras.io/. Accessed: 2022-07.

LeeJayHui. Github forum discssion in "segmentation models", "about preprocessing", 2019. URL https://github.com/qubvel/segmentation_models/issues/277. Accessed: 2022-07.

Martin Mirbauer, Tobias Rittig, Tomáš Iser, Jaroslav Křivánek, and Elena Šikudová. SkyGAN: Towards Realistic Cloud Imagery for Image Based Lighting. In Abhijeet Ghosh and Li-Yi Wei, editors, *Eurographics Symposium on Rendering*. The Eurographics Association, 2022. ISBN 978-3-03868-187-8. doi: 10.2312/sr.20221151.

Andreas Mischok. Poly haven, image "liliensteina", 2021. URL https://polyhaven.com/a/liliensteina. Accessed: 2022-07.

Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

Michal Nazarczuk, Sibi Catley-Chandar, Ales Leonardis, and Eduardo Pérez-Pellitero. Self-supervised hdr imaging from motion and exposure cues, 2022. URL https://arxiv.org/abs/2203.12311.

OpenEXR. Openexr website. https://www.openexr.com/, 2022. Accessed: 2022-07.

Paperswithcode. Paperswithcode on semantic segmentation. https://paperswithcode.com/task/semantic-segmentation, 2022. Accessed: 2022-07.

Pavel "qubvel" Iakubovskii). "segmentation models" library, 2022. URL https://github.com/qubvel/segmentation_models. Accessed: 2022-07.

Erik Reinhard, Greg Ward, Sumanta Pattanaik, and Paul Debevec. *High dynamic range imaging : acquisition, display, and image-based lighting*. Morgan Kaufman, 2006. ISBN 978-0-12-585263-0.

et al. S. Dev, F. M. Savoy. Singapore high-dynamic-range whole sky imaging segmentation database, 2018. URL http://vintage.winklerbros.net/shwimseg.html.

Stackoverflow. Loss plot figure. https://stackoverflow.com/questions/64940632/how-to-illustrate-a-3d-graph-of-gradient-descent-using-python-matpl 2022. Accessed: 2022-07.

TensorFlow. Tensorflow website, 2022a. URL https://www.tensorflow.org/. Accessed: 2022-07.

TensorFlow. Tensorflow documentation, resnet50, 2022b. URL https://www.tensorflow.org/api_docs/python/tf/keras/applications/resnet50/ResNet50. Accessed: 2022-07.

Michael Weiher. Domain adaptation of hdr training data for semantic road scene segmentation by deep learning, 2019. URL https://www.bit-ts.com/wp-content/uploads/2020/02/domain_adaptation_of_hdr_training_data_for_semantic_segmentation_by_deep_learning-michael_weiher.pdf.

Hanning Yu, Wentao Liu, Chengjiang Long, Bo Dong, Qin Zou, and Chunxia Xiao. Luminance attentive networks for hdr image and panorama reconstruction, 2021. URL https://arxiv.org/abs/2109.06688.

List of Figures

A. Attachments

A.1 Code

Jupyter notebook .ipynb with code.

A.2 Logs

A folder with different logs from TensorBoard.

A.3 Predictions

A folder with full-sized predictions from different models