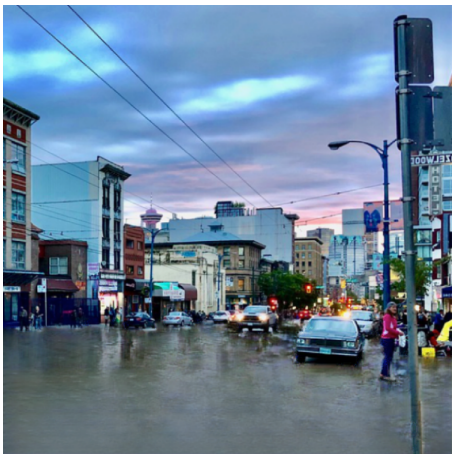
Using Artificial Intelligence to Contribute to Climate Change Mitigation: Generative Approaches and Practical Tools

Victor Schmidt



Predoctoral Thesis Proposal
Under the Supervision of Prof. Yoshua Bengio

Mila
Université de Montréal
December 2020

Contents

Abstract

It is difficult to downplay the importance of fighting climate change. Unfortunately, public awareness and concern about it often does not match the magnitude of its threat to humans and our environment. One reason for this mismatch is that it is difficult for people to mentally simulate the complex and probabilistic effects of climate change. To contribute to overcoming these challenges, we present work done towards producing visualizations of climate-related extreme events to help the public better understand - both rationally and emotionally - the consequences of not taking sufficient action against climate change. We focus on three events: floods, wildfires and smog. We show how we can easily obtain compelling visualizations of the latter events using pre-trained models of depth estimation and semantic segmentation. Visualizing floods however is much more challenging and we present a two-steps approach to generate such images, conditionally on an input image from Google StreetView: a Masker model to produce a pixel-wise binary mask of where water should be applied and a Painter model to generate water textures from a picture of scene and its associated flood mask. To train the Masker model, we created a Unity3D simulated world and employed Domain Adaptation techniques to transfer its masking abilities to the real world. We show that we can obtain realistic masks adapting to the angles and contents of reasonably complex scenes. The painter on the other hand is trained on real images of floods, using a variation of the GauGAN training procedure and architecture, leveraging Spatially Adaptive Denormalizations. We present results illustrating how the model is a able to produce realistic and context-aware water textures.

It is also important to reflect critically on our own impact as a community, not merely deflecting responsibility. We present work done towards quantifying the carbon emissions of machine learning, along with a set of actionable items for practitioners. We also introduce CodeCarbon, a Python package to systematically track the emissions of any Python script in general, in particular those of AI experiments. We believe that by providing automated tools to the public we can encourage awareness about carbon emissions and push for more transparency.

Finally, we present a contribution to climate modeling using conditional GANs. Clouds are a major component of our climate and understanding their dynamics is crucial to accurately and efficiently modeling our changing climate. A key way in which they contribute to the greenhouse effect is by reflecting the Sun's light, which is why modeling clouds' reflectance fields is of great importance. Using GANs conditioned on sparse physical measurements we are able to generate realistic samples, paving the way towards a data-driven approach to climate simulations.

Future work will be focused on improving the climate events visualizations. We envision a single model, OmniGAN, that could leverage all the data-sources currently available, be they real or simulated, ground-truth or inferred from pre-trained models, to produce flood masks, depth maps and/or segmentation maps. We believe leveraging additional tasks such as depth prediction and semantic segmentation would not only better inform the Masker to produce more realistic masks, but it would also allow us to produce images of all three events in a single forward pass through the model, making it much more computationally efficient in the context of a production environment with a public-facing website. Furthermore we aim at using this research as a potent tool to make climate change closer and contribute to research in behavioral psychology, being able to actually measure people's risk perception of climate-relate extreme events as a function of the event's distance.

Chapter 1

Introduction: From Machine Learning to Generative Adversarial Networks

Over the course of two decades, Deep Learning (DL) has risen to be an ubiquitous technological and scientific tool, enabling major advances in computer vision, speech and natural language processing, reinforcement learning, scientific discovery, generative modeling and many more. In this work, we investigate how DL can be used to help mitigating Climate Change especially through the lens of a special kind of DL-based generative models: Generative Adversarial Networks. We will also reflect critically on those methods' own impact on climate change through the emissions they create. Lastly we will focus on current challenges and planned future work.

The first chapter is dedicated to introducing the reader to the important concepts required to understand the content of subsequent chapters. We will introduce the notions at the core of Machine Learning and Deep Learning with a bias towards computer vision. Most of the content of those sections takes inspiration from Bishop (2006) and Goodfellow et al. (2016) which should be considered as capital references to dig deeper into Machine and Deep Learning. In the last section of Chapter 1, we will learn about Generative Adversarial Networks and focus on their visual applications.

In the second chapter, we will present work realized towards creating vivid and personalized visualizations of climate-related extreme events. We will show how GANs and other Deep Learning models for computer vision can be used to render realistic images of what a neighbourhood could look like if a flood, a wildfire or a smog episode happened there. Through these accurate, relatable and personalized visualizations of extreme climate events, we aim at raising the public's awareness and conceptual understanding of climate change by bringing the future closer.

In the third chapter, we will introduce work aimed at quantifying the carbon emissions of Machine Learning. Machine Learning is becoming more and more prevalent in an increasing variety of industries, with modern algorithms requiring more and more data and computational resources. It is therefore important to reflect critically on our own impact by quantifying it and encouraging available action items.

Chapter 4 will present an effort to use GANs to help improve cloud modeling and to accelerate General Circulation climate models. From an input tensor representing the spatial distribution of a set of physical measurements, can GANs recover the actual cloud reflectance fields matching those features?

Lastly in Chapter 5 we will present ongoing and future work with an emphasis on ways forward to improve the visualizations presented in Chapter 2. We will introduce OmniGAN, an architecture designed to leverage a variety of data sources and modeling tasks. We will also touch upon how we can evaluate the impact of our interactive climate events visualizations on people and the perception of the risk that climate-related extreme events represent to them.

1.1 Machine Learning

Machine Learning (ML) is the science of making computer algorithms perform certain tasks through an iterative learning process and exposure to data. To put it in the words of 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". This formulation emphasizes the three most important cogs in the machine's learning procedure: what do you want to learn (T), how do you measure success (P), and how do you learn (E)?

1.1.1 Tasks

There are many families of tasks one can perform using an ML approach. In this section, we will introduce the two most common types, for the reader to get more familiar with the kind of problems that ML can be used for. These families are not exclusive and in many cases one can frame the problem into an equivalent form using either one. The choice depends on the ultimate goal of the model but there is often several ways to look at a problem. Yet, Classification and Regression remain the most common and generic ways of addressing a data-driven problem.

Let us note $\boldsymbol{x} \in \mathbb{R}^d$ a data sample. In this vector form, $\boldsymbol{x}$ is a collection of *features* representing the sample. For instance, if $\boldsymbol{x}$ represents a 28×28 grey-scale image, then its vector representation is in $[0, 1]^{784}$. If $\boldsymbol{x}$ were a sentence, it could be represented as a vector of 0s and 1s, with 1 at position i meaning that the sentence contains the word i (with respect to a known vocabulary). In most cases, one has access to a dataset of many samples, which we'll note $\boldsymbol{X} = \{\boldsymbol{x}^{(k)}\}_{k=1..n}$.

Classification This type of problem arises when one aims at categorizing outcomes based on inputs. Given an input vector $\boldsymbol{x}$ the goal is to be able to predict a target value y which takes value in a bounded subset of $\mathbb{N}$: y is a label index from a known set of possible labels (or outcomes) for the data. For instance, one could try and classify images into digits. The goal is therefore to learn a mapping $f : \mathbb{R}^d \to \{1, ..., L\}$ such that $\forall \boldsymbol{x}^{(k)} \in \boldsymbol{X} : f(\boldsymbol{x}^{(k)}) = y^{(k)}$. When y can take multiple values at once (for instance a single text paragraph can have multiple topics), we call such tasks multi-label classification.

Regression Similarly to classification, regression aims at producing a value y from a given input $\boldsymbol{x}$. However in this situation, y's support is continuous over $\mathbb{R}$. For instance, one could want to predict the price of a house given a set of handcrafted features (surface, neighbourhood, age etc.) or predict the distance of a given pixel in an image to the camera (depth prediction). In this context, the output of the algorithm would be a mapping similar to that of the classification task, only with a different domain: $f : \mathbb{R}^d \to \mathbb{R}$ such that $\forall \boldsymbol{x}^{(k)} \in \boldsymbol{X} : f(\boldsymbol{x}^{(k)}) \approx y^{(k)}$.

1.1.2 Performance metrics

Choosing the right metric to evaluate the model (f) is paramount in obtaining a successful and robust procedure. When performing a classification task, the most common metric is accuracy, *i.e.* the ratio of successful classifications:

$$Acc_{\boldsymbol{X}}(f) = \frac{1}{n} \sum_{k=1}^{n} \mathbb{1}_{f(\boldsymbol{x}^{(k)})=y^{(k)}}. \tag{1.1}$$

For regression tasks, the most common metric is called the Mean Squared Error (MSE), *i.e.* the

average squared distance between predictions and targets:

$$MSE_{\boldsymbol{X}}(f) = \frac{1}{n}\sum_{k=1}^{n}(f(\boldsymbol{x}^{(k)}) - y^{(k)})^2. \tag{1.2}$$

Many more metrics exist and can be tailored to one's specific use-case. They should however all be understood with care: they depend on the *data* as much as they depend on the model f. If a model is fit to a limited set of data points one can usually expect that a problem called *overfitting* will arise: the model is too good on the training data, so much so that it picks up on statistical patterns specific to the dataset not to the data generation process itself, and will therefore likely not be able to generalize (see Fig. 1.1). To be able to counter that effect, one should split their full dataset into 3 exclusive parts: a training set to fit the model, a validation set to evaluate its performance on new data and tune the model's hyper-parameters (those which are defined prior to the fitting procedure, like a polynomial's order) and a test set for a final performance metric based on both optimization procedures (parameters on the train set and hyper-parameters on the validation set).

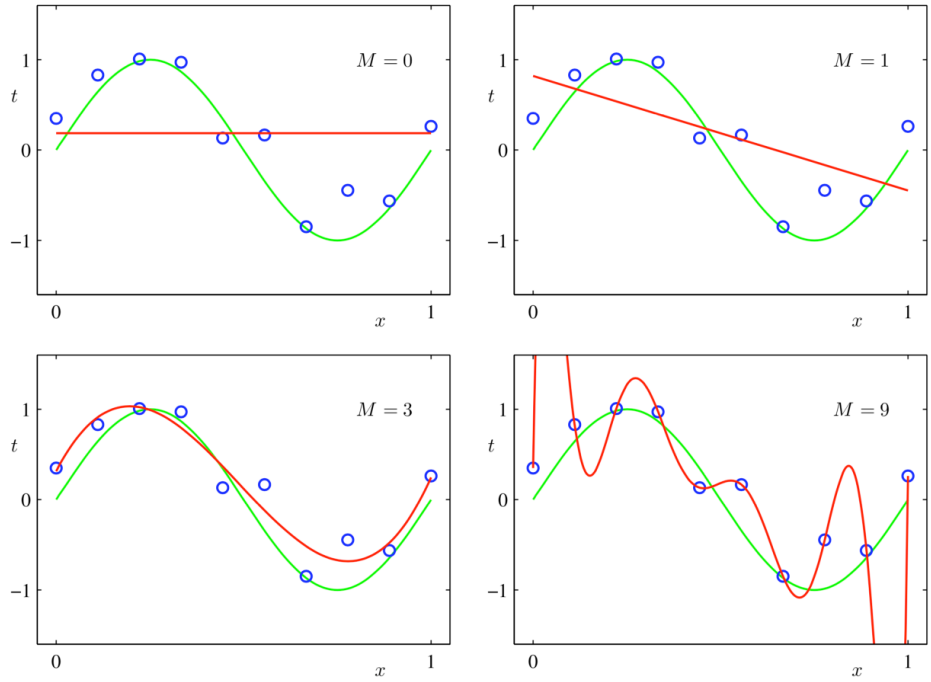


Figure 1.1: Over/Underfitting a polynomial. The green line is the true data generation process while blue dots are noisy samples collected. When trying to fit a polynomial of order M, one can see that as M grows, the optimal polynomial adjusts to the samples more than it adjusts to the true distribution. Extracted from Bishop (2006).

.

1.1.3 Training Paradigms

So far we have looked at the case when ground truth target values are available, be they regression targets or classification labels. In general, this set of ML problems is called *Supervised Learning*: a "teacher" supervises the training of the learning algorithm through these ground truth values $y^{(k)}$. Most often than not, the learned model f is trained not to output a "hard" and certain decision but rather a smooth probability distribution over possible labels given an input: $f : \boldsymbol{x} \rightarrow f(\boldsymbol{x}) = \tilde{y} = p(y|\boldsymbol{x})$. Looking at decisions in terms of probability distributions over outcomes creates a bridge

between Supervised Learning and Discriminative modeling, where the goal is to be able to draw decision boundaries in the target domain.

However, labels are not always available and we could still want to learn from the data: clustering similar samples together, denoising a signal, generating new samples, modeling the topics of a text etc. These use-cases are part of a set of ML problems called *Unsupervised Learning*. There is a wide range of applications and formulations but a common one is to learn about the data *itself* and is therefore focused on modeling the data generation density function p. This modeling of $p(\boldsymbol{x})$ may be explicit when one aims at estimating the likelihood $p(\boldsymbol{x})$ of each sample, or implicit, meaning that what is really of interest is to be able to sample new data from p (as we'll see later in Generative Adversarial Networks (Goodfellow et al., 2014)). These *Generative modeling* tasks are an important part of Unsupervised Learning algorithms, but many other forms exist.

In spite of the above very condensed presentation of the concepts, one should not understand them as exclusive categories, rather as complementary approaches to solve a given problem. For instance, the *a priori* unsupervised generative task of modeling $p(\boldsymbol{x})$ can be framed as a series of d discriminative tasks using the chain rule of probabilities:

$$p(\boldsymbol{x}) = \prod_{i=1}^{d} p(x_i|x_1, ..., x_{i-1}) \ , \qquad x \in \mathbb{R}^d. \tag{1.3}$$

On the other hand, if we wanted to generate new data samples *conditionally* on a category *i.e.* $p(\boldsymbol{x}|y)$ (think sampling images of specific digits, not just any digit) we would need some form of supervision as one should have access to the samples' labels, as we will see in Section 1.3.4.

1.1.4 Estimators

How do we actually train the models to best model the target distribution (of data or labels)? One way to look at this problem is to measure the likelihood of the data under the model's parameters and adjust those so as to maximize the likelihood of the data. This is called *Maximum Likelihood Estimation*. Let θ be our model's parameters, $p_{\text{model}}(\boldsymbol{x}|\theta)$ the likelihood of a sample $\boldsymbol{x}$ according to the model. If $\boldsymbol{X}$ is a set of identically and independently distributed samples, we have:

$$p_{\text{model}}(\boldsymbol{X}\,|\theta) = \prod_{k=1}^{n} p_{\text{model}}(\boldsymbol{x}^{(k)}|\theta). \tag{1.4}$$

We therefore look for θ^* such that:

$$\begin{aligned}
\theta^* &= \arg\max_{\theta} \prod_{k=1}^{n} p_{\text{model}}(\boldsymbol{x}^{(k)}|\theta) \\
&= \arg\max_{\theta} \sum_{k=1}^{n} \log\left(p_{\text{model}}(\boldsymbol{x}^{(k)}|\theta)\right) \qquad \text{(log does not the change argmax)} \\
&= \arg\min_{\theta} -\frac{1}{n} \sum_{k=1}^{n} \log\left(p_{\text{model}}(\boldsymbol{x}^{(k)}|\theta)\right) \qquad \text{This is called the Negative Log-Likelihood} \\
&= \arg\min_{\theta} -\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log\left(p_{\text{model}}(\boldsymbol{x}|\theta)\right)\right] \\
&= \arg\min_{\theta} -\left(\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log\left(p_{\text{model}}(\boldsymbol{x}|\theta)\right)\right] - \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log\left(p_{\text{data}}(\boldsymbol{x})\right)\right]\right) \\
&= \arg\min_{\theta} D_{KL}(p_{\text{data}}||p_{\text{model}}(\cdot|\theta)).
\end{aligned} \tag{1.5}$$

We can see in Equations 1.5 that maximizing the likelihood of the data under our parametric family of probability distributions is equivalent to minimizing the Kullback-Leibler divergence from the model to the data[1]. The learning procedure therefore turns into an optimization problem to find θ^*, maximizing the likelihood or, equivalently, minimizing the *negative log-likelihood* of the data under the model.

One problem in this formulation is that it considers all values of the parameter vector θ to be equally likely and feasible. This might however not be the case and we may want to be able to include prior knowledge on those parameters θ into the estimation. This procedure is called *Maximum A Posteriori* estimation because it operates on the posterior distribution $p_{\text{model}}(\theta|\,\boldsymbol{X})$ and can be formulated as follows:

$$\begin{aligned}\theta_{MAP} &= \arg\max_{\theta} p_{\text{model}}(\boldsymbol{X}\,|\theta)p(\theta) \qquad \big(= \arg\max_{\theta} p_{\text{model}}(\theta|\,\boldsymbol{X}),\ \text{Bayes Rule}\big)\\ &= \arg\max_{\theta} \log\big(p_{\text{model}}(\boldsymbol{X}\,|\theta)p(\theta)\big)\\ &= \arg\max_{\theta} \sum_{i=1}^{n} \log\big(p_{\text{model}}(\boldsymbol{x}^{(k)}|\theta)\big) + \log\big(p(\theta)\big).\end{aligned} \tag{1.6}$$

Equation 1.6 shows how we can interpret the prior as a regularizer (in other words, an extra constraint on the optimization problem).

Lastly, one might be interested in more than a point estimate for the parameters θ: full Bayesian inference would give $p_{\text{model}}(\theta|\,\boldsymbol{X})$'s probability density function and not just its arg-maximum. The difference, which makes Bayesian Inference often more difficult than MAP, is that we would now need to compute the *marginal likelihood* of the data (also called *evidence*) $p(\boldsymbol{X})$.

1.2 Deep Learning

In this section we will now introduce one of the most powerful and versatile tool in the modern Machine Learning modeling toolbox: Artificial Neural Networks (ANNs). An ANN is a differentiable parametric function f, composed of sub-functions which are themselves differentiable and parametric: $f = f^{(L)} \circ f^{(L-1)} \circ ... \circ f^{(1)}$, for L a positive integer. $f^{(i)}, i < L$ are called *hidden layers* and $f^{(L)}$ is called the output layer. The goal of an ANN is to approximate some function f^*, for instance a classifier $f^* : \boldsymbol{x} \to f^*(\boldsymbol{x}) = y$. To understand how the $f^{(i)}$ building blocks are chosen and trained to work together, let us start with introducing the most basic of them: linear regression.

1.2.1 Linear regression

Linear regression is a regression task where the target value y is predicted from a linear combination of the input's dimensions, or features. Let $\boldsymbol{x} \in \mathbb{R}^d$ be an input sample and $y \in \mathbb{R}$ its associated target value. Let $\boldsymbol{w} \in \mathbb{R}^d$ be a *weight* vector, *i.e.* the parameters of our linear model[2]:

$$f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{w}. \tag{1.7}$$

Given a dataset of n samples $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ and labels $\boldsymbol{Y} \in \mathbb{R}^n$ we want to learn $\boldsymbol{w}^*$ which minimizes

[1] The KL-divergence is a (non-symmetric) measure of similarity between two distributions. The smaller the divergence, the more similar the distributions are. Its formulation and behaviour will be further introduced in Section 1.3.1

[2] One can also define a linear model as $\boldsymbol{x} \to \boldsymbol{x}^\top \boldsymbol{w} + b$ with $b \in \mathbb{R}$ a bias term. The two definitions are in fact equivalent provided we concatenate a 1 to $\boldsymbol{x}$ in our formulation, increasing its effective dimension by 1.

Algorithm 1: Mini-batch Stochastic Gradient Descent for Linear Regression: minimizing mean squared error loss function $\mathcal{L} : \boldsymbol{w}, \boldsymbol{X}, \boldsymbol{Y} \rightarrow \frac{1}{2}|| \boldsymbol{X}\,\boldsymbol{w} - \boldsymbol{Y} ||_2^2$

Initialization: let $\alpha \in \mathbb{R}^{+*}$ be the learning rate, $\delta \in \mathbb{R}^{+}$ the tolerance, $\boldsymbol{w} \in \mathbb{R}^d$ an initial arbitrary vector and $m \in \mathbb{N}^*$ the mini-batch size;
while $|| \boldsymbol{X}\,\boldsymbol{w} - \boldsymbol{Y} ||_2^2 > \delta$ **do**
  Randomly sample m examples and targets from $\boldsymbol{X}$ and $\boldsymbol{Y}$ into $\hat{\boldsymbol{X}} \in \mathbb{R}^{m \times d}$ and $\hat{\boldsymbol{Y}} \in \mathbb{R}^m$
  $\nabla_{\boldsymbol{w}} L(\boldsymbol{w}, \hat{\boldsymbol{X}}, \hat{\boldsymbol{Y}}) = \hat{\boldsymbol{X}}^\top \hat{\boldsymbol{X}} \boldsymbol{w} - \hat{\boldsymbol{X}}^\top \hat{\boldsymbol{Y}}$
  $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}, \hat{\boldsymbol{X}}, \hat{\boldsymbol{Y}})$
end

the average distance between predictions and ground truth labels:

$$\begin{aligned} \boldsymbol{w}^* &= \arg\min_{\boldsymbol{w}} \frac{1}{n} \sum_{i=1}^{n} \left(y^{(i)} - \boldsymbol{x}^{(i)\top} \cdot \boldsymbol{w}\right)^2 \\ &= \arg\min_{\boldsymbol{w}} \frac{1}{2} || \boldsymbol{X}\,\boldsymbol{w} - \boldsymbol{Y} ||_2^2. \end{aligned} \tag{1.8}$$

The MSE objective function which we are trying to minimize being convex, one can find its unique global minimum by setting its gradient *w.r.t.* $\boldsymbol{w}$ to 0:

$$\nabla_{\boldsymbol{w}} f = \boldsymbol{X}^\top (\boldsymbol{X}\,\boldsymbol{w} - \boldsymbol{Y}) = \boldsymbol{X}^\top \boldsymbol{X}\,\boldsymbol{w} - \boldsymbol{X}^\top \boldsymbol{Y}\,. \tag{1.9}$$

If $\boldsymbol{X}^\top \boldsymbol{X}$ is invertible[3], then we can compute $\boldsymbol{w}^* = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{Y}$. If it is not, or if the inversion is too costly[4], one may turn to *Gradient Descent* to find $\boldsymbol{w}^*$ (Algorithm 1 presents a variant of Gradient Descent: mini-batch stochastic gradient descent).

If we model target values y as following a Gaussian distribution centered around some mean $\mu_\theta(\boldsymbol{x}) = f_{\boldsymbol{w}}(\boldsymbol{x})$ and constant standard deviation σ we can see that minimizing the MSE produces the same training algorithm as maximum-likelihood estimation:

$$\begin{aligned} & y \sim \mathcal{N}\left(y; f_{\boldsymbol{w}}(\boldsymbol{x}), \sigma^2\right) \\ \implies \boldsymbol{w}^* &= \arg\max_{\boldsymbol{w}} \sum_{i=1}^{n} \log p(y^{(i)} | \boldsymbol{x}^{(i)}, \theta) \\ &= \arg\max_{\boldsymbol{w}} - \sum_{i=1}^{n} \frac{||y^{(i)} - \boldsymbol{x}^{(i)\top} \boldsymbol{w}||^2}{2\sigma^2} - n \log(\sigma) - \frac{n}{2} \log(2\pi) \\ &= \arg\min_{\boldsymbol{w}} \frac{1}{n} \sum_{i=1}^{n} \left(y^{(i)} - \boldsymbol{x}^{(i)\top} \boldsymbol{w}\right)^2. \end{aligned} \tag{1.10}$$

To perform binary classification, we can use a similar process, modeling y as a Bernoulli variable $y \sim Bernoulli(p = f_{\boldsymbol{w}}(\boldsymbol{x}))$. Knowing that a Bernoulli distribution's parameter should be restricted to $p \in [0; 1]$ we can modify $f_{\boldsymbol{w}}$ slightly to guarantee this:

$$p_{\text{model}}(y = 1 | \boldsymbol{x}; \boldsymbol{w}) = \frac{1}{1 + e^{-\boldsymbol{x}^\top \boldsymbol{w}}} = \sigma(\boldsymbol{x}^\top \boldsymbol{w}). \tag{1.11}$$

[3] $\forall \boldsymbol{z} \in \mathbb{R}^d \neq 0, \boldsymbol{z}^\top \boldsymbol{X}^\top \boldsymbol{X}\,\boldsymbol{z} = || \boldsymbol{X}\,\boldsymbol{z}|| \geq 0$ so $\boldsymbol{X}^\top \boldsymbol{X}$ is positive *semi*-definite and it could be non-invertible

[4] Inversion is naively $\in O(d^3)$ and at best $\in O(d^{2.4})$ in terms of time complexity and $\in O(d^2)$ in terms of memory (Petković and Stanimirović, 2009)

In this context, σ is called the *sigmoid* function and this approach is known as *logistic regression*. Unlike linear regression (Equation 1.9), logistic regression does not have a closed-form solution and $\boldsymbol{w}^*$ is searched for by minimizing the negative log-likelihood (also called *cross-entropy* loss function) using gradient descent. In the case when there are more than 2 possible classes, we can generalize the sigmoid function:

$$\text{softmax}_i(\boldsymbol{x}) = \frac{e^{\boldsymbol{x}_i}}{\sum_j e^{\boldsymbol{x}_j}} = p_{\text{model}}(y = i). \tag{1.12}$$

Applying the softmax function to the output of a linear layer will normalize the resulting vector and make it a probability distribution, which we interpret as the model's current belief of $\boldsymbol{x}$ belonging to class y.

1.2.2 Composition of functions

A watchful eye might notice something about logistic regression: it is the composition of differentiable parametric functions $f^2 \circ f^{(1)}$ where $f^{(2)}$ is the sigmoid function and $f^{(1)}$ is a linear parametric function. It is the simplest form of ANN for classification there is.

So far we have restricted $\boldsymbol{w}$ to be a vector. But if we set it to be a matrix $\boldsymbol{W} \in \mathbb{R}^{p \times d}$ we obtain a vector output $\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} \in \mathbb{R}^p$. Using a second set of weights, say a vector $\boldsymbol{u} \in \mathbb{R}^p$ and applying σ to each element of $\boldsymbol{h}$ we can stack logistic regressions on top of each other:

$$f_{\boldsymbol{W},\boldsymbol{u}}(\boldsymbol{x}) = \sigma\big(\boldsymbol{u}^\top \sigma(\boldsymbol{W}\boldsymbol{x})\big). \tag{1.13}$$

Without the innermost σ, $f_{\boldsymbol{W},\boldsymbol{u}}$ could have been reduced to $f_{\boldsymbol{w}}$ with $\boldsymbol{w} = \boldsymbol{u}^\top \boldsymbol{W} \in \mathbb{R}^d$ but because σ is not linear, such a reduction is not possible and we can therefore create an intermediate *representation* $\boldsymbol{h}$ for $\boldsymbol{x}$. Through the learning process (most often with gradient descent), $\boldsymbol{W}$ and $\boldsymbol{u}$ can work together to best adjust the model to the task, giving it more learning capacity.

Because all functions of which $f_{\boldsymbol{W},\boldsymbol{u}}(\boldsymbol{x})$ is a composition are differentiable, one can simply apply recursively the chain rule of calculus to each of them to compute the gradient of the loss function (MSE or negative log-likelihood) with respect to each weight and therefore iteratively update each of them towards the optimal set of parameters. This algorithm is called *back-propagation*.

Such a sequence of functional modules where parameters lie in linear matrix-vector operations followed by an *activation function* like σ is called a *Feed-forward Neural Network*. This name illustrates information flowing forward through layers, creating intermediate representations used by subsequent layers to eventually form a prediction $f_{\boldsymbol{\theta}}(\boldsymbol{x})$. The term "Neural Network" comes from a loose analogy with biological neurons which are layered and inter-connected. If we look at the value $\boldsymbol{h}_i$ and $\boldsymbol{h}_j$ of $\boldsymbol{h}$'s i^{th} and j^{th} dimensions we can see that they depend on two different combinations of the same input variables, thereby focusing on different properties of the input:

$$\begin{cases} \boldsymbol{h}_i = \sigma\big(\sum_{k=0}^{d-1} \boldsymbol{W}_{i,k}\boldsymbol{x}_k\big), \\ \\ \boldsymbol{h}_j = \sigma\big(\sum_{k=0}^{d-1} \boldsymbol{W}_{j,k}\boldsymbol{x}_k\big). \end{cases} \tag{1.14}$$

Recalling the notations from the beginning of the sub-section, the function which maps $\boldsymbol{x}$ (*i.e.* the input layer) to $\boldsymbol{h}$ is called a hidden layer, with parameter $\boldsymbol{W}$. $\boldsymbol{W}$'s first dimension p defines the *width* of the layer, represented by the number of vertical neurons in Figure 1.2. Theoretically, we can use an arbitrary number of layers of arbitrary width, provided we define weights with compatible dimensions. For instance we could have a second hidden layer parametrized by $\boldsymbol{V} \in \mathbb{R}^{p \times q}$. Hidden layers are followed by the output layer, with parameter $\boldsymbol{u}$ ($\in \mathbb{R}^q$ if using $\boldsymbol{V}$, $\in \mathbb{R}^p$ otherwise). A network's total number of layers is called its *depth* (hence, "deep" learning).

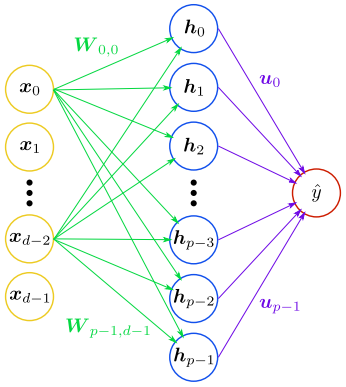


Figure 1.2: Illustration of the flow of information through a feed-forward neural network to obtain $\hat{y} = f_{\boldsymbol{W},\boldsymbol{u}}(x)$. Circles represent neurons and contain values (input features then intermediate representations and ultimately output prediction, from left to right) while arrows represent the weight of each connection between neurons. For readability only a few parameters are displayed alongside arrows.

In the end, whether the last function is the sigmoid function to model a Bernoulli variable or the identity function to perform a regression, however many intermediate layers there are, what matters is that feed-forward neural networks apply a linear model (the output layer parametrized by $\boldsymbol{u}$ in our example) to a *learnable feature extractor* (the hidden layers parametrized by $\boldsymbol{V}$ and $\boldsymbol{W}$). This, is the main difference between traditional Machine Learning and Deep Learning: instead of providing engineered features to the final decision model, we let it learn directly from the raw data the useful representations which will better allow it to solve its task.

1.2.3 Convolutional Neural Networks

Traditional linear layers in feed-forward neural networks (as introduced in the previous section) have the specificity of linking every output unit to every input unit, making the information flow very *dense* (for that reason they are also called dense or fully-connected layers). While this can be a useful feature in some cases, it also means that each part of the learned representation depends on the entire input data. In the case of images, this is however not desirable: in general salient and useful information is not spread uniformly across the image and objects tend to be local. Using *sparse* connections between units would be more appropriate.

Another specificity of images is that they often show translation invariance: the nature of an object's visual properties should not depend on its location. We should then design learning algorithms which are *equivariant* to translation, meaning that the result of the computations on a shifted image are the shifted results of the computations on the original image.

Lastly, some features of an image are generally useful and we should be able to re-use concepts like edge, shape or texture detection, as well as arbitrarily complex combinations of those. For instance to classify the image of a car as being one, having different parts of the model learn independently to detect the round shape of a wheel seems redundant. Yet, we also need to create an abstraction able to combine the concept of wheels with other feature detectors (for instance a trunk detector) to be able to differentiate cars from motorcycles. This can be achieved through *hierarchical parameter sharing*, in other words: reusing the same set of weights all over the image and stacking them one after the other.

Alongside inspiration from the visual cortex, these are the main motivations for *Convolutional Neural Networks*, or CNNs (LeCun, 1989), specialized in the analysis of grid-like data: 1D time series, 2D grey-scale images, 3D RGB images, 4D videos etc. The basic building block of a CNN is the convolution operation. Given an input signal $x(t)$ and some weighting function $w(t)$ to convolve $x(t)$

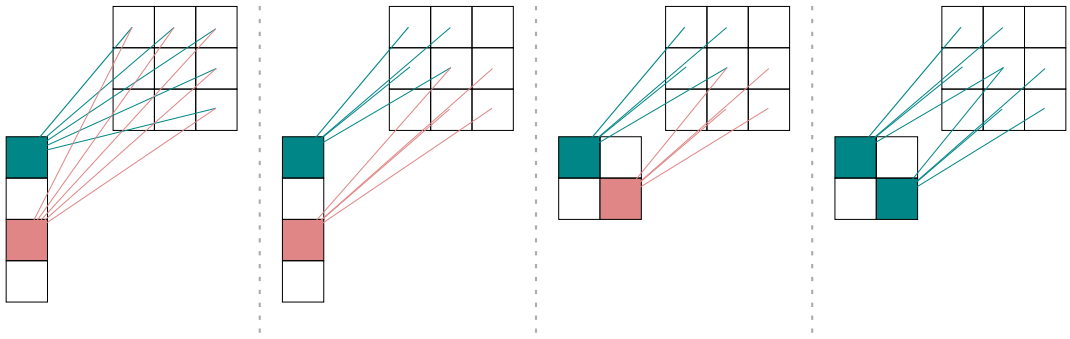


Figure 1.3: Incorporating assumptions into CNNs. **(a)** Dense connections: all neurons in the output layer (column, left) attend to all locations in the input image (grid, right) (some connections are omitted for clarity). **(b)** Sparse assumption: not all regions are important for a given output neuron. **(c)** Locality assumption: spatial distribution of neuron matters. **(d)** Weight sharing assumption: the same operation is performed across the whole image to detect one kind of feature anywhere ; a second (red) filter could be applied to the same locations in order to create a second feature map from the input.

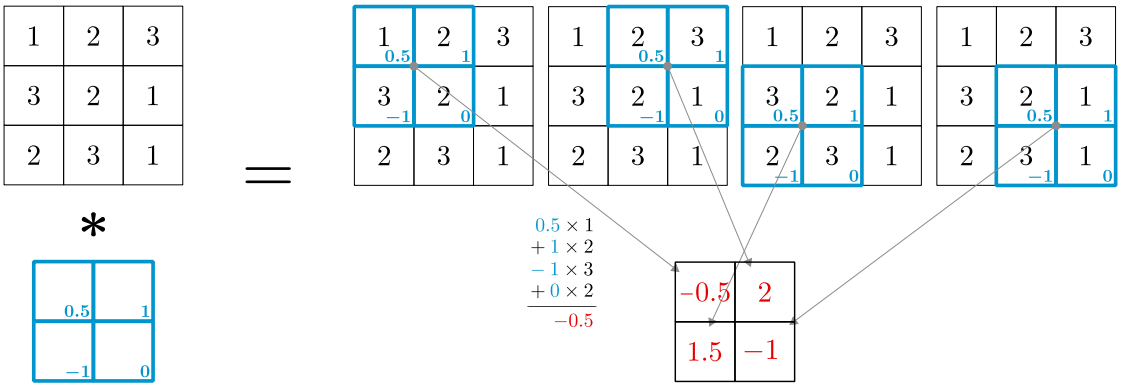


Figure 1.4: Illustration of the convolution operation of a 2×2 kernel K (blue) on a 3×3 feature map I (black) without padding, dilation and a stride of 1, resulting in a 2×2 output feature map (red).

with, then the result of the convolution operation is:

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da. \tag{1.15}$$

In most computerized applications, data is actually discretized and we use a weighting function $w(t)$ with the same dimensionality as $x(t)$. In this context, $w(t)$ is called a *kernel* or *filter* and $s(t)$ is called a *feature map*. For 2D data with input I and kernel K:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n). \tag{1.16}$$

For instance, to detect vertical edges, one may use a 3×3 filter such that $K[:,0] = 1$, $K[:,1] = 0$ and $K[:,2] = -1$. To detect horizontal edges, we can transpose this kernel: $K[0,:] = 1$, $K[1,:] = 0$ and $K[2,:] = -1$. We can then either choose to engineer a set of such filters which we hope will yield interesting features for a given image, or choose to *learn* filters. Just like in the previous section we explained that Deep Learning differed from traditional Machine Learning by the choice of learning

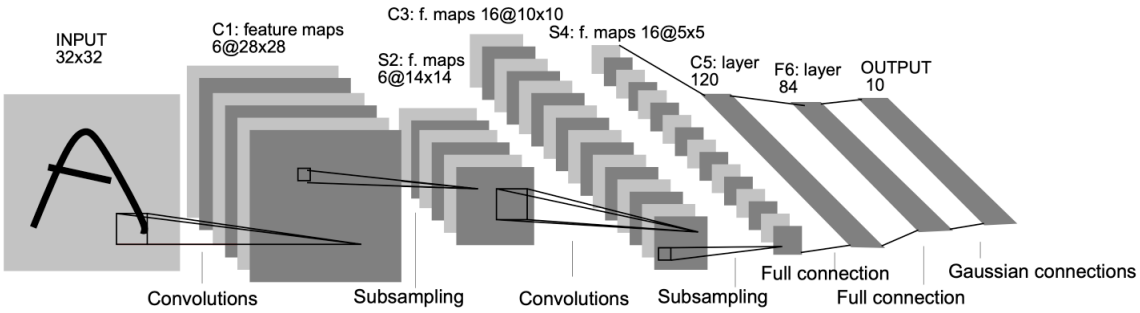


Figure 1.5: Convolutional then dense pipeline to classify documents in LeCun et al. (1998). Subsampling layers are analogous to pooling layers and Gaussian connections a special case of dense layer. One can understand the overall procedure as learning a vector representation $\boldsymbol{h} \in \mathbb{R}^{84}$ of any image fed through the network, and $\boldsymbol{h}$ is the input to a final classification layer with 10 classes.

representations instead of leveraging human-engineered features, Convolutional Neural Networks learn filters to create useful representations of an image.

To achieve this, we define a set of k filters to apply to an image, which we randomly initialize and learn through backpropagation (the convolution operation is differentiable). As illustrated in Figure 1.4, convolving a 2×2 kernel on a 3×3 input yields a 2×2 feature map. Using k of those will therefore yield a $k \times 2 \times 2$ 3D array, where each of the k feature map detects different properties of the input image. Those feature maps being analogous to images (they have k channels instead of 3), we can convolve each of them with a second layer of p 2×2 filters resulting in a $p \times 1 \times 1$ tensor, which we can reshape into a vector $\boldsymbol{h} \in \mathbb{R}^p$. Finally to make a prediction we can feed this representation $\boldsymbol{h}$ into a feedforward neural network to perform a classification or regression. This whole process of convolutional layers followed by dense layers was first introduced by LeCun et al. (1998) and is illustrated in Figure 1.5. Lastly, just like for feed-forward neural networks, we should apply an activation function after each convolutional layer to enable the model to learn non-linear representations of the data.

1.2.4 Practical Visual Deep Learning

In practice many improvements to the standard CNN described in the previous section are used to make learning more efficient and stable. Here is a non-exhaustive list of important concepts, some of which are illustrated in figure 1.6.

Advanced Convolutions To account for different resolutions at which features should be learned, there are many ways to perform a convolution differently from the vanilla setup illustrated in Figure 1.4. One can choose different kernel sizes (not only 2×2), shift kernels by more than 1 step left or down (called the *stride*), add *padding* around the feature map (with 0s or reflecting the nearest value etc.) or even dilate the kernel (*à trous* convolutions).

Pooling Layers Convolutional layers provide a good way of representing the spatial distribution of important features learned through the training process. This distribution might however be sensitive, by design, to the localization of objects. To limit this sensitivity, one can use pooling layers to downsample feature maps and aggregate features together. Most common downsampling strategies are 2×2 average or max pooling.

Activation functions There is a wide range of activation functions beyond the sigmoid function. One problem of the sigmoid function is that its gradient vanishes as $|x| \to \infty$, which slows down learning

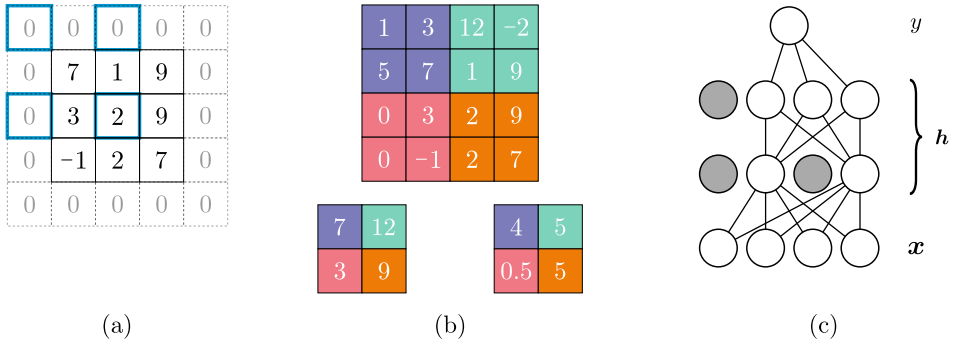


Figure 1.6: CNNs in practice: advanced convolutions, pooling and dropout. **(a)** Illustration of a dilated kernel (blue) applied on a feature map with zero-padding of 1. **(b)** Max-pooling (left) *versus* Average-pooling (right). **(c)** Dropout: grey nodes in hidden layers aggregated under the notation $\boldsymbol{h}$ are set to zero, so the computation of y from input nodes $\boldsymbol{x}$ does not take them into account.

as the backpropagation algorithm will factor σ's derivative in the overall gradient computation. The most used alternative is the Rectified Linear Unit ($\text{ReLU}(x) = \max(0, x)$) and variants (Exponential Linear Unit, Parametrized Rectified Linear Unit, Leaky Rectified Linear Unit etc.).

Dropout In Section 1.1.2 we introduced the concept of overfitting. In the special case of neural networks, a very efficient way of mitigating overfitting is to randomly zero-out neurons during training (Hinton et al., 2012). We can interpret this method in two interesting ways: because the zeroed-out neurons change at every training step, many architectures are tried thus making the final model (we do not drop neurons when evaluating it) an effective *ensemble* of models. Another way of understanding dropout is to say that because a given neuron cannot rely on its neighbours (input is noisy because some of it may be dropped, the effective number of neurons per layer changes all the time etc.), it has to focus on robust generic useful features instead of peculiarities of the data which may only be relevant when other neurons are present (Krizhevsky et al., 2012).

Batch Normalization Ioffe and Szegedy (2015) introduced a powerful technique to make the training of CNNs both more efficient and stable: Batch Normalization (BN). It can be understood as yet another type of layer in a CNN which does the following: first standardize values along the batch dimension (for instance, all pixel values in a batch of images are reduced by their mean and divided by their standard deviation) and then rescale them in a learnable way with trainable scale and shift parameters. Given a mini-batch $\hat{\boldsymbol{X}}$ of m inputs to the BatchNorm layer and trainable parameters γ and β [5]:

$$BN(\hat{\boldsymbol{X}}) = \gamma \frac{\hat{\boldsymbol{X}} - \mathbb{E}[\hat{\boldsymbol{X}}]}{\text{Var}[\hat{\boldsymbol{X}}]} + \beta. \tag{1.17}$$

While the empirical effects of BN are ubiquitous, its theoretical effects are not yet fully understood. It is currently believed that its performance comes from a smoothing effect on the loss function being optimized for (Santurkar et al., 2019).

Optimization Mini-batch Stochastic Gradient Descent, though a powerful improvement on Gradient Descent, suffers from a noisy procedure for which it is hard to find an appropriate learning rate and which can get stuck at saddle points of the loss function (Sun et al., 2019). To account for the former problem, AdaGrad (Duchi et al., 2011) scales the learning rate according to the squares of all historical gradients while Nesterov (Nesterov, 1983) accelerates the descent and accounts for saddle points by

[5] Mean, variance, γ, β are real numbers for feed-forward neural networks, vectors of size C in 4D tensors of size $m \times C \times H \times W$ like images or feature maps

accumulating the previous gradient as momentum. Many approaches combine those approaches, the most widely used are RMSProp (Hinton et al., 2012) and Adam (Kingma and Ba, 2014). More recent and promising work include RAdam (Liu et al., 2019) and Novograd (Ginsburg et al., 2019).

Residual Blocks A very important breakthrough in machine vision occurred with the invention of residual connections (He et al., 2015). The observation is that CNNs' performance as a function of depth saturates and then degrades as models grow deeper: in their experiments a 56-layer CNN performs worse than a 20-layer one on a classification task. Authors show that this is not a matter of overfitting and rather hypothesize that it is because the optimization problem itself gets much harder. To alleviate this, they introduce a simple yet powerful idea which is ubiquitous in vision models to this day: a residual connection from the input to the output of a layer. In other words, let F be a residual block (ResBlock), $\mathcal{F}$ be a set of 2 convolutional layers with constant spatial dimensions (typically 3×3 kernel and padding of 1) and $\boldsymbol{x}$ the input to F. The residual function is $F(\boldsymbol{x}) = \mathcal{F}(\boldsymbol{x}) + \boldsymbol{x}$. To see how this helps learning, notice all the weights of F are actually in $\mathcal{F}$, this means that $\mathcal{F}$ needs only to learn how to *transform* the input to F and not both include the useful information from $\boldsymbol{x}$ *and* how to create additional relevant features for downstream layers.

1.2.5 Some Applications

There is a very wide range of applications and use-cases which imply imagery and CNNs. This section will present some of them: tasks, datasets and concepts which will help in the context of this work.

ImageNet Started in 2010, the ImageNet (Russakovsky et al., 2014) project gathers more than 14 million labeled images spread across 20,000 classes. Of those, one thousand are used in the annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC). This competition has driven a lot of the recent advances in visual deep learning over the past decade, from the coronation of Deep Learning and first utilization of Graphical Processing Units (GPUs, Krizhevsky et al. (2012)) to human-level performance with residual connections (He et al., 2015) through the sanctuarization of small kernels, batch normalization and deep networks.

Semantic Parsing Beyond single-label image classification, another major task in modern applications of Deep Learning and CNNs is called *Semantic Segmentation*: its goal is to map every pixel in the input image to a class. Given a labeled dataset of $3 \times H \times W$ images and a set of L possible classes for each pixel, the model should output a probability map $L \times H \times W$ (typically using the softmax function in equation 1.12). A similar task directed towards understanding the scene captured by an image is called *Object Detection*. The goal is to detect and localize the presence of known objects. Putting those two ideas together leads to *Instance Segmentation* where the task is not only to classify pixels into classes, but also being able to separate different instances of the same class (two different cars on a road for example). Many other tasks exist in computer vision and in almost all of them CNNs are the leading technique.

Transfer Learning Not all datasets are as large and diverse as ImageNet. It can therefore become difficult for models to learn complex pixel interactions from few data samples. An entire sub-field of ML is dedicated to solving the problem of transfer learning: how can we design learning procedures which re-use "knowledge" of a task to a new one? The computer vision community has democratized an elegant way of (partially) answering this question: pre-trained models. As we have seen before, one can interpret the early layers of a model as feature extractors: they build useful representations of the data. When training a neural network on a new task, instead of initializing the network from randomly sampled weights, one may start with the weights of a network which was previously trained on the larger dataset of a "similar" task. The notion of similarity is vague on purpose, often only empirical studies will say whether the tasks transfer or not, yet if one trains a network to classify

specific kinds of butterflies chances are that the features learned by a network trained to perform well on ImageNet will work well and provide a good starting point. Depending on the task and amount of data available one might re-use all or only parts of the pre-trained network, allowing for some existing parts to change (*fine-tuning*) while not updating some other parts (*freezing*) (Yosinski et al., 2014).

1.3 Generative Adversarial Networks

The goal of Generative Modeling is to be able to learn (or *model*) a probability distribution p over some variables of interest $\boldsymbol{x}$: by sampling from such models it is possible to generate synthetic data points in the input space (Bishop, 2006). In 2014 Goodfellow et al. (2014) introduced a particularly successful and elegant way to use neural networks to learn p: Generative Adversarial Networks (GANs).

1.3.1 Measures

To understand GANs, their training and difficulties, we first need to introduce some relevant quantities.

Kullback-Leibler Divergence KLD is a measure of how a probability distribution diverges (is different) from a reference distribution.

$$\mathrm{D_{KL}}(p||q) = \mathbb{E}_{x\sim p}[\log(\frac{p(x)}{q(x)})] = \int_x p(x)\log(\frac{p(x)}{q(x)})dx. \tag{1.18}$$

It reads, somewhat counter-intuitively, "KL Divergence from q to p" and can be interpreted as the *relative entropy* of p and q or the *information gain* of using q instead of p. If p is the real-world data distribution and q is a model, then we can understand KLD as "how much the model still has to learn". It is always positive and is 0 for $p = q$, but it is not a metric as it is not symmetric and does not satisfy the triangle inequality. Note that $\mathrm{D_{KL}}(p||q)$ penalizes values where p puts mass but q does not, meaning that if it aims at minimizing KLD the model q should be able to account for all modes of p at the cost of maybe putting mass where p puts none. On the contrary *reverse* KLD $\mathrm{D_{KL}}(q||p)$ penalizes values where q puts mass but p doesn't, meaning that the model q should not put mass where there is no empirical evidence from p, at the cost of maybe missing some modes of the data.

Jensen-Shannon Divergence JSD is another measure of similarity (or dissimilarity) between probability distributions, and is constructed from KLD. Unlike the latter, JSD is symmetric and bounded in $[0;\ 1]$. It is still not a proper metric but its square root is (Endres and Schindelin, 2003).

$$\mathrm{JS}(p||q) = \frac{1}{2}\mathrm{D_{KL}}(p||\frac{p+q}{2}) + \frac{1}{2}\mathrm{D_{KL}}(q||\frac{p+q}{2}). \tag{1.19}$$

Because of the symmetric formulation, JSD has one noticeable downside: it is well behaved even where $p(x)$ is small so it does not penalize much samples drawn from q which are unlikely under p.

Wasserstein Distance WD is a proper distance metric between two different probability distributions. It is also called the *Earth-Mover* (EM) distance as it can be understood as the least amount of energy one has to spend to transform a pile of earth shaped as q into the shape of p. Its main conceptual components are therefore *how much* do you have to move and *how far* should you take it.

$$W(p,q) = \inf_{\gamma\sim\prod(p,q)} \mathbb{E}_{(x,y)\sim\gamma}[||x-y||]. \tag{1.20}$$

$\prod(p, q)$ is the set of all joint probability distributions $\gamma(x, y)$ which have p and q as marginals. One such probability distribution γ is called a transport plan: how much of x should we move to y so that it follows the same probability distribution. In particular, by definition, $\sum_x \gamma(x, y) = q(y)$: once the transport plan is executed on x we have re-distributed the mass adequately so as to match q. Similarly $\sum_y \gamma(x, y) = p(x)$. On the other hand, $||x - y||$ is the distance between x and y. Multiplying $\gamma(x, y)$ by $||x - y||$ we have "how much should we move from x to y and how far". Summing over possible values of (x, y) gives us the expected cost of the plan $\mathbb{E}_{(x,y)\sim\gamma}[||x-y||]$ and taking the $\inf_{\gamma\sim\prod(p,q)}$ gives us the best plan, minimizing the energy spent.

1.3.2 Vanilla GANs

In the original GAN framework, a *generator* network G is trained to map a sample from a prior distribution $\boldsymbol{z} \sim p_{\boldsymbol{z}}$ (typically uniform or Gaussian) to one following p_{G} so that the latter is as "close" as possible to the true data distribution p_{data} (in that it minimizes the Jensen-Shannon divergence between the two distributions). To do so, G should generate samples $\tilde{\boldsymbol{x}} = G(\boldsymbol{z})$ to fool a second network, the *discriminator* D, whose task (to the contrary of G) is to be able to discriminate (*i.e.* classify) between real samples $\boldsymbol{x}$ and fake samples $\tilde{\boldsymbol{x}}$.

Using back-propagation through the two differentiable functions that are D and G, the two models therefore play a two-player minimax game which theoretically converges (given enough capacity for D and G) to an equilibrium where $p_{\text{G}} = p_{\text{data}}$ and $\forall \boldsymbol{x} \sim p_{\text{data}}, \forall \boldsymbol{z} \sim p_{\boldsymbol{z}} : \; D(\boldsymbol{x}) = D(G(\boldsymbol{z})) = \frac{1}{2}$ (Goodfellow et al., 2014). The game they play is parametrized by the following value function $V(G, D)$[6]:

$$\min_G \max_D V(G, D) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}[\log(D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}[1 - \log(D(G(\boldsymbol{z}))]. \tag{1.21}$$

One may recognize in equation (1.21) that D should maximize the probability that it assigns to any $\boldsymbol{x}$ to be real while minimizing the probability that it assigns to any $\tilde{\boldsymbol{x}} = G(\boldsymbol{z})$ of being real. On the other hand G's sole goal is to fool D, meaning that it should maximize the probability D assigns to any $\tilde{\boldsymbol{x}}$.

1.3.3 Training GANs

Vanishing Gradient Images can be understood as vectors of a high dimensional space. For instance, a 100×100 RGB image lies in a 30,000-dimensional space. There is however theoretical and empirical evidence that this dimensionality is only artificially high: real data is not evenly distributed in those high-dimensional spaces, rather extremely concentrated near low dimensional manifolds (Arjovsky and Bottou, 2017). This makes sense intuitively: the 100×100 image of an object is very much constrained and restricted by its semantics: a face should have a mouth, a nose, eyes and a chin for instance. In other words the probability of obtaining a picture of a face by independently sampling each pixel is null, *i.e.* the set of faces in pixel space has a measure of 0 according to the lower dimensional manifold hypothesis. This means that it is very likely that a learned distribution will have no overlap, almost everywhere, with the real distribution. Theorem 2.3 in Arjovsky and Bottou (2017) shows that if p_{G} and p_{data} have their support lie in two manifolds that don't have full dimension and don't perfectly align and if it is further assumed that p_{data} and p_{G} are continuous in their respective manifolds, then:

$$\begin{aligned} &\text{D}_{\text{KL}}(p_{\text{G}}||p_{\text{data}}) = +\infty \\ &\text{D}_{\text{KL}}(p_{\text{G}}||p_{\text{data}}) = +\infty \\ &\text{JS}(p_{\text{G}}||p_{\text{data}}) = \log(2). \end{aligned} \tag{1.22}$$

[6]When used to train either D or G, this function is also noted as a loss function $\mathcal{L}(G, D)$ or $\mathcal{L}_{GAN}(G, D)$

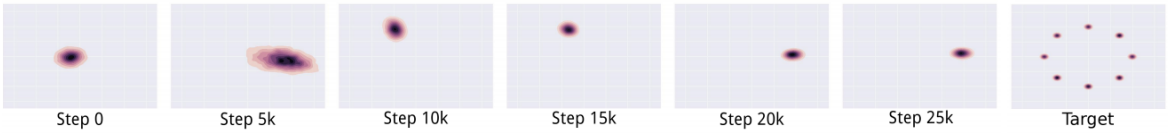


Figure 1.7: from Metz et al. (2017). Heat maps of the generator's empirical distributions after various training steps, with the actual target data distribution on the right. We can see that the generator is able to focus relatively well and fast on a given mode of the data but fails to encompass of all them even in such a toy example.

In the lower dimensional manifold hypothesis, however good the samples from G might look, the divergences will be infinity, with arbitrarily close manifolds. This cues to the fact that using those quantities to measure the distance between those kinds of distributions may be problematic, especially when the goal is to perform gradient descent on them.

Furthermore, it will be very easy and fast for the discriminator to separate the manifolds. So if D doesn't train properly, the gradients flowing to G will be meaningless and it will not learn the real distribution, but if D is any good, the gradient quickly goes to 0 and G learns very slowly, if it does so at all. Corollary 2.1 shows that in the same conditions, for D^* the optimal discriminator:

$$\lim_{||D-D^*||\to 0} \nabla_\theta \, \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})} \Big[\log \big(1 - D(G(\boldsymbol{z}))\big) \Big] = 0. \tag{1.23}$$

Mode Collapse In practice, authors of the original GAN paper had noticed the very unstable training of the first formulation and advocated for the minimization of $\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[-\log D(G(\boldsymbol{z}))]$. In this case, Arjovsky and Bottou (2017) show in Theorem 2.5 that for D* the optimal discriminator and for a given θ_0:

$$\begin{aligned} D^* &= \frac{p_{\text{data}}}{p_{\text{data}} + p_{G_{\theta_0}}} \\ \implies \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})} \Big[-\nabla_\theta \log \big(D^*(G_\theta(z))\big)|_{\theta=\theta_0} \Big] &= \nabla_\theta \big[\mathrm{D_{KL}}(p_{\mathrm{G}_\theta}||p_{\text{data}}) - 2\mathrm{JS}(p_{\mathrm{G}_\theta}||p_{\text{data}}) \big]|_{\theta=\theta_0}. \end{aligned} \tag{1.24}$$

As noted in the paper, the first striking thing is that the JSD has a negative factor, meaning that it will push for the distributions to be different. Second, the KL term is actually the reverse KL which we know (see Section 1.3.1) will assign a high cost to any sample unlikely under the data distribution and low cost to mode-dropping. This explains how GANs are able to generate good looking examples but have a hard time encompassing all the modes of the real data distribution, a phenomenon called Mode Collapse and illustrated in Figure 1.7.

Wasserstein GANs To solve for the vanishing gradient and mode collapse problems, Arjovsky and Bottou (2017) argue for a "softer measure, that incorporates a notion of distance between the points in the manifolds". In a follow-up paper (Arjovsky et al., 2017) the authors introduced the Wasserstein distance as being such a measure. A toy example from the paper illustrates this: let $z \sim U[0,1]$ the uniform distribution over the unit interval. Let p_0 be the distribution of $(0, z) \in \mathbb{R}^2$ and p_θ be the distribution of $(\theta, z) \in \mathbb{R}^2$. p_0 and p_θ are both 2D-distributions which are uniform along the y axis

and constant on the x axis ($x = 0$ or $x = \theta$). Then we can see that:

$$\mathrm{D_{KL}}(p_0||p_\theta) = \mathrm{D_{KL}}(p_\theta||p_0) = \begin{cases} 0 & \text{if } \theta = 0, \\ +\infty & \text{if } \theta \neq 0, \end{cases}$$

$$\mathrm{JS}(p_0||p_\theta) = \begin{cases} 0 & \text{if } \theta = 0, \\ \log 2 & \text{if } \theta \neq 0, \end{cases} \tag{1.25}$$

$$W(p_0, p_\theta) = |\theta|.$$

This means that as $\theta_t \to 0$, the sequence $(p_{\theta_t})_{t\in\mathbb{N}}$ converges to p_0 under the EM/W distance but not under either the KL, reverse KL or JS divergences. To use the Wasserstein distance into our optimization problem, Arjovsky et al. (2017) transform Equation 1.20 into a more tractable form using the Kantorovich-Rubinstein duality (Villani, 2008):

$$W(p_{\mathrm{G}_\theta}, p_{\text{data}}) = \sup_{||f||_L \leq 1} \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}[f(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{x}\sim p_{\mathrm{G}_\theta}}[f(\boldsymbol{x})]. \tag{1.26}$$

The supremum is taken over the set of all 1-Lipschitz functions. It is important to note that in the WGAN formulation, f's role is to estimate the EM distance, not discriminate between real and fake samples. It is therefore called a *critic* and not a discriminator. Because the critic estimates the EM distance, it should be trained for multiple steps on its own before updating the generator. Authors enforce the critic's Lipschitzness by clipping its weights to $[-0.01; 0.01]$.

Gradient Penalty Authors of Arjovsky et al. (2017) acknowledge that "Weight clipping is a clearly terrible way to enforce a Lipschitz constraint. If the clipping parameter is large, then it can take a long time for any weights to reach their limit, thereby making it harder to train the critic till optimality. If the clipping is small, this can easily lead to vanishing gradients when the number of layers is big, or batch normalization is not used" and "actively encourage interested researchers to improve on this method".

To solve for this, Gulrajani et al. (2017) introduced a regularizer to the WGAN loss, by noting that "a differentiable function is 1-Lipschitz if and only if it has gradients with norm at most 1 everywhere, so we consider directly constraining the gradient norm of the critic's output with respect to its input". This constraint is softly implemented as an extra term to the WGAN loss instead of imposing it directly on the critic itself. Because enforcing this constraint everywhere is intractable, it is sufficiently computed along straight lines from data to generated samples with $\hat{\boldsymbol{x}} = \epsilon\boldsymbol{x} + (1 - \epsilon G(\boldsymbol{z}))$, $\epsilon \sim \mathrm{U}[0, 1]$:

$$\mathcal{L}_{\text{WGAN-GP}} = \underbrace{\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[D(G(\boldsymbol{z}))] - \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[D(\boldsymbol{x})]}_{\text{WGAN Loss}} + \lambda \underbrace{\mathbb{E}_{\hat{\boldsymbol{x}}\sim\hat{p}(\boldsymbol{x})}\left[(||\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})|| - 1)^2\right]}_{\text{Gradient Penalty}}. \tag{1.27}$$

Spectral Normalization WGAN-GP introduces a powerful yet computationally expensive way of constraining the critic to be 1-Lipschitz. An alternative introduced by Miyato et al. (2018) called *Spectral Normalization* acts directly on the weights of f instead of relying on a soft constraint on the whole model. To understand how this works, let us consider a critic with structure $f(\boldsymbol{x}, \theta) = W^{L+1} \underbrace{a_L}_{\text{Activation function}}(W^L a_{L-1}(\cdots(a_1(\underbrace{W^1}_{\text{Weight parameters}}\boldsymbol{x}))\cdots))$.

Let $\sigma(W)$ be the spectral norm of a matrix W (which is also its larger singular value) and $g : \boldsymbol{x} \to W\boldsymbol{x}$ a linear function then its Lipschitz constant is:

$$||g||_{\text{Lip}} = \sup_{\boldsymbol{h}} \sigma(\nabla g(\boldsymbol{h})) = \sup_{\boldsymbol{h}} \sigma(\nabla W\boldsymbol{h}) = \sup_{\boldsymbol{h}} \sigma(W) = \sigma(W). \tag{1.28}$$

Given that activations in the ReLU family have a maximum slope of 1, their Lipschitz constant is at most 1 and we have:

$$\begin{aligned} ||f||_{\text{Lip}} &\leq ||\boldsymbol{h}_L \to W^{L+1}\boldsymbol{h}_L||_{\text{Lip}} \cdot ||a^L||_{\text{Lip}} \cdots ||a_1||_{\text{Lip}} \cdot ||\boldsymbol{h}_0 \to W^1\boldsymbol{h}_0||_{\text{Lip}} \\ &\leq \prod_{l=1}^{L+1} ||\boldsymbol{h}_{l-1} \to W^l\boldsymbol{h}_{l-1}||_{\text{Lip}} \\ &= \prod_{l=1}^{L+1} \sigma(W^l). \end{aligned} \tag{1.29}$$

We can therefore see that by transforming weights W into $\tilde{W}_{SN} = W/\sigma(W)$ we can enforce $||f||_{\text{Lip}} = 1$. Using the power iteration method we can efficiently approximate $\sigma(W)$. This has become a standard in training GANs.

1.3.4 Conditional GANs

In many cases, we want to not only sample from the distribution over all the data indiscriminately but rather apply some form of conditioning: sampling from a particular class, getting a specific visual feature or making the output resemble another image in some way.

DCGAN The basic model structure which shaped modern GANs for visual applications is called DCGAN: Deep Convolutional Generative Adversarial Network (Radford et al., 2016). The model illustrated in Figure 1.8 starts from a sampled latent vector $\boldsymbol{z} \in \mathbb{R}^{100}$ which is reshaped into a grid-like shape and passed through a series of transposed convolutions to learn greater spatial features. This form of upsampling has been found to generate images with so-called "checkerboard artifacts" (Odena et al., 2016), and it has been replaced in recent years with a standard non-learned bi-linear upsampling followed by learnable dimensionality-preserving convolutions (typically a 3×3 kernel with padding 1). Nonetheless, DCGAN was a cornerstone empowering future advances in (conditional) adversarial image generation.

cGAN If one has access to labels, a simple yet effective idea (Oord et al., 2016) to generate samples conditionally on a specific class is to simply feed the label to the generator alongside noise so that it knows what it should output. Now that the generation is conditional, the discriminator should not only distinguish real *vs* fake, but also assess how realistic a sample is *within* its class. This is achieved by concatenating the image input (real data or generated sample) with a label descriptor (a 1-hot vector for instance). This leads to a very simple modification of the GAN objective:

$$\min_G \max_D V(G, D) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log(D(\boldsymbol{x}|\boldsymbol{y})] - \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z}|\boldsymbol{y})))]. \tag{1.30}$$

Info-GAN In the absence of labels, one can still hope to be able to discover structure in $p(\boldsymbol{x})$. The idea behind InfoGAN (Chen et al., 2016) is that if we want to discover C classes in the data and be able to generate images conditionally on those discovered clusters, we should maximize the Mutual Information between a label c and the generated image $G(\boldsymbol{z}|c)$, which is noted $\boldsymbol{I}(c; G(\boldsymbol{x}|c))$. If $\boldsymbol{I}$ is low, then knowing c is irrelevant to the content if the image. On the other hand, if $\boldsymbol{I}$ is large

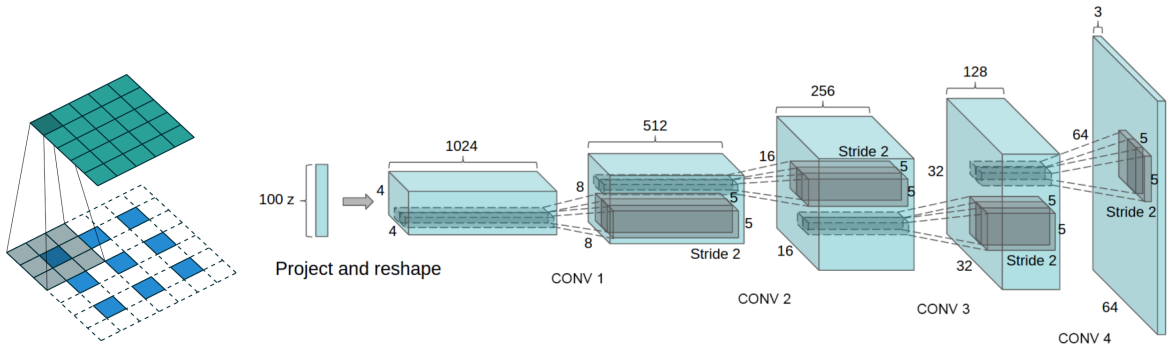


Figure 1.8: **Left** (from Dumoulin and Visin (2016)): Illustration of a *fractionally strided convolution*, also called *transposed convolution* used to learn an increased spatial resolution. Input is the lower blue squares and output is the upper green feature map. **Right** (from Radford et al. (2016)): DCGAN generator. A 100 dimensional uniform distribution $\boldsymbol{z}$ is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions then convert this high level representation into a 64×64 pixel image. Notably, no fully connected or pooling layers are used.

then knowing c is a good predictor for the image's content. In practice, since $\boldsymbol{I}$ is hard to estimate, authors maximize a lower bound on $\boldsymbol{I}$ by making the discriminator output not only a number $D(\boldsymbol{x})$ but also a probability distribution $q(c|\boldsymbol{x})$ using an additional layer in D: it should be able to recover the initial class c which conditioned the generation of $\boldsymbol{x}$. Sampling c from a categorical distribution with 10 classes, authors are able to recover without any supervision the 10 MNIST classes (LeCun, 1998). Sampling c from a continuous distribution (factored gaussians for instance), they are able to control for the orientation and thickness of digits generated. As we have already emphasized multiple times, the great challenge of Deep Learning is to create useful representations of the data which can be leveraged to perform specific tasks. The process by which InfoGAN learns how to perform unsupervised conditional generation enables it to create *disentangled representations*, meaning that even though the data is gathered indiscriminately in the dataset $\boldsymbol{X}$, the representations InfoGAN creates are able to isolate specific features and structure within the data distribution to create high-level control variables (the classes C).

AC-GAN The core idea behind InfoGAN is that the discriminator should be able to reconstruct the label c which conditioned the generation of a sample $G(\boldsymbol{z}, c)$. This idea can be implemented to obtain great performance when labels are available: this is the idea behind Auxiliary Classifier GANs (Odena et al., 2017). Instead of maximizing the mutual information of a generated sample and its class, we can simply maximize the likelihood of the class because we have access to labels. Similarly to InfoGAN, the discriminator outputs a distribution $q(c|G(z, c))$ in addition to its prediction of a given sample being real or fake. Authors of AC-GAN therefore derive two losses $\mathcal{L}_{source\ is\ real\ or\ fake} = \mathcal{L}_S$ and $\mathcal{L}_{class} = \mathcal{L}_C$, G should maximize $\mathcal{L}_C - \mathcal{L}_S$ and D should maximise $\mathcal{L}_C + \mathcal{L}_S$:

$$L_S = \mathbb{E}_{c\sim p(c),\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x}|c)}[\log(D(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z}),c\sim p(c)}[\log(D(G(z,c)))] \tag{1.31}$$

$$L_C = \mathbb{E}_{c\sim p(c),\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x}|c)}[\log(q(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z}),c\sim p(c)}[\log(q(G(z,c)))]. \tag{1.32}$$

In this formulation, both models work together to make classes as predictable as possible but they work against each other to generate real-looking samples.

SPADE An alternative way of conditioning generated samples is to *denormalize* weights in layers according to some learned rule. The idea is that normalization techniques such as Batch Norm (others

include Instance Normalization (Ulyanov et al., 2017), Layer Normalization Ba et al. (2016) etc.) are unconditional, they simply operate on the feature statistics of the data and learn bias and gain terms depending on the task at hand. The idea introduced in de Vries et al. (2017) and Huang and Belongie (2017) is to make those parameters functions of some other input to *condition* the denormalization on some external data. This idea was improved upon by Perez et al. (2017) and later successfully utilized and generalized by Park et al. (2019) to generate images conditioned on a segmentation map. To do so, they turned the learned denormalization parameters into small CNNs which take a segmentation map as input and produce a tensor of gains a biases per location and per channel in a feature map. At a given layer of shape $C \times H \times W$ and for a batch size of N, let $h_{m,c,y,x}$ be the activation before normalization at site $(m \in N, c \in C, y \in H, x \in W)$. Let $\boldsymbol{s}$ be a segmentation map. Then the Spatially Adaptive (De)normalization of $h_{m,c,y,x}$ is:

$$\hat{h}_{m,c,y,x} = \gamma_{c,y,x}(\boldsymbol{s})\frac{h_{m,c,y,x} - \mu_c}{\sigma_c} + \beta_{c,y,x}(\boldsymbol{s})$$

with

$$\mu_c = \frac{1}{NHW}\sum_{m,y,x} h_{m,c,y,x} \tag{1.33}$$

$$\sigma_c = \sqrt{\frac{1}{NHW}\sum_{m,y,x}(h_{m,c,y,x})^2 - (\mu_c)^2}.$$

1.3.5 Image to Image Translation

A specific task in machine vision is image to image translation (IIT). The goal is to transform an image into another one, from a domain to another, just like verbal translation systems translate from one language to another. In this section, we will look at examples of such IIT tasks and the models developed to achieve them with an exclusive focus on GAN methods. Because the image we seek to generate (the translated image) depends on another one (the image to be translated), this task is inherently a conditional generation task and we will expand on ideas from the previous section to understand it.

Tasks There are multiple families of IIT task. The first thing to consider is the availability of ground truth data (which would call for a supervised approach) or not (framing the task as an unsupervised learning problem). Supervised tasks include the translation of an aerial image of a city to its equivalent in a map rendering, transforming a segmentation map into a realistic photographic view of the scene, colourizing black and white images[7], etc. In the absence of labels, meaning there is no ground-truth transformation available, we can still aim at transforming apples into oranges, horses into zebras, photographs into paintings etc. In general, the unity of IIT tasks is that the model should learn a pixel to pixel mapping.

Pix2Pix Isola et al. (2017) introduced the first GAN architecture able to perform well on a variety of supervised IIT tasks. Their model is based on a series of convolutions to downsample the input image followed by ResBlocks and a series of transposed convolution to upsample back to the original image's dimension. They train the model not only with a standard conditional GAN loss but also with a regularization L_1 term, inciting the model to generate images close to the ground truth. Let $\boldsymbol{x}$ be from one domain and $\boldsymbol{y}$ the target for $\boldsymbol{x}$ in the other domain:

[7] One might think that there usually is no ground truth data for B&W photos, and this is true for older pictures or those which were actually taken in B&W. A simple trick to get ground truth data is to look at the problem the other way around: it is very easy to turn an RGB image into a B&W equivalent and use the latter to reconstruct the former.

Figure 1.9: from Isola et al. (2017). **Left**: example supervised IIT tasks. **Right**: illustration of Pix2Pix's training procedure.

$$\mathcal{L}_{Pix2Pix}(G,D) = \underbrace{\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}}[\log(D(\boldsymbol{y}))] + \mathbb{E}_{\boldsymbol{x},\boldsymbol{z}}[\log(1-D(G(\boldsymbol{x},\boldsymbol{z})))]}_{\mathcal{L}_{cGAN}} + \lambda \underbrace{\mathbb{E}_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}}[|\boldsymbol{y}-G(\boldsymbol{x},\boldsymbol{z})|]}_{\mathcal{L}_{L_1}}. \tag{1.34}$$

CycleGAN Another milestone in IIT was the invention of CycleGANs Zhu et al. (2017), allowing for the unsupervised learning of two mappings $G : X \to Y$ and $F : Y \to X$ such that F and G are respectively trained with standard GAN losses, and a *cycle-consistency loss* to structure the domains of each model, enforcing $F(G(x)) \approx x$ and $G(F(x)) \approx y$. Concretely, if F translates horses into zebras and G translates zebras into horses then a zebra translated into a horse should look like the original zebra when the resulting horse is translated back into a zebra. Inherently this formulation requires two full GAN models (generators and discriminators) trained together to learn both translations directions:

$$\begin{aligned}\mathcal{L}_{\text{CycleGAN}}(F,G,D_X,D_Y) =& \mathbb{E}_y[\log(D_Y(y))] + \mathbb{E}_x[\log(1-D_Y(G(x)))] \\ &+ \mathbb{E}_x[\log(D_X(x))] + \mathbb{E}_y[\log(1-D_X(F(y)))] \\ &+ \lambda\big(\mathbb{E}_x[|F(G(x))-x|] \\ &+ \mathbb{E}_y[|G(F(y))-y|]\big) \\ =& \mathcal{L}_{GAN}(G,D_Y) + \mathcal{L}_{GAN}(F,D_X) + \lambda\,\mathcal{L}_{cyc}(G,F).\end{aligned} \tag{1.35}$$

CycleGAN was the first GAN-based model able to learn translation without supervision on a variety of tasks: horses to zebras, apples to oranges, paintings to photos, summer to winter, iPhone to DSLR[8]. While these results were impressive, a major limitation of this approach is that it gives very little control over the generation process and tends to change high level features globally in addition to the actual object which should be changed. For instance, the model is very impressive at translating pictures to paintings and the other way around because it should change the whole *style* of the image, without altering its *content*. On the other hand, translating horses to zebras or more drastically sheep to giraffes, requires a model to actually change part of the image's content, not it's global style. For instance, even though zebras are most often pictured in the Savannah and horses in meadows, the translation of a horse into a zebra should not change the grass's color or the shape of trees. Also, while the losses do not encourage "hallucinations" they do not explicitly prevent them: F can hallucinate content (include a new element in the generated image) provided it is realistic and G is able to remove it (to satisfy the reconstruction criterion). Strengths and weaknesses of CycleGAN are illustrated on the horse to zebra task in Figure 1.10.

InstaGAN A strategy to make translation models focus on content and shape instead of the entire image's composition is to provide them with masks. For instance Attention-Guided CycleGAN (Alami Mejjati et al., 2018) learns an attention map $A \in [0;1]^{H\times W}$ to only translate the model's

[8]Digital Single-Lens Reflex cameras used with a shallow depth of focus have a very sharp foreground and a blurred background

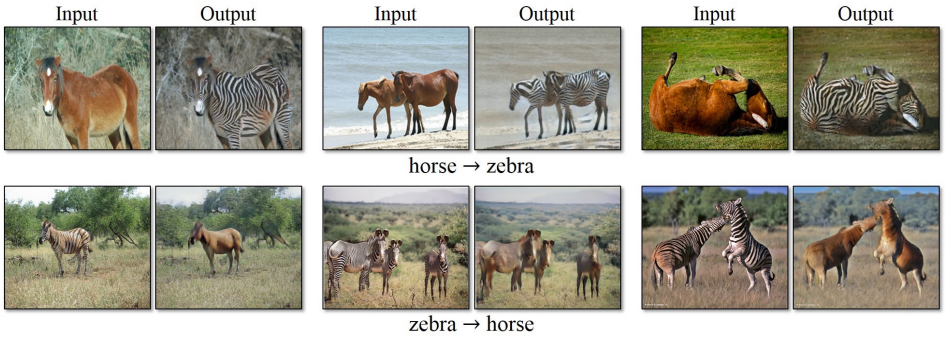


Figure 1.10: from Zhu et al. (2017). Example translations from a CycleGAN pair of models on the horse to zebra task. Strengths: the models are able to paint or remove stripes from the bodies of the animals, there are very few artifacts and the images are quite sharp. Weaknesses: in some areas the model is not able to focus exclusively on the bodies of the animals (e.g. top middle translation), it changes the saturation and contrast of images, and alters other salient objects (e.g. the tree in the bottom left translation).

zone of focus: $\hat{y} = A \odot G(x) + (1 - A) \odot x$[9]. This strategy, while efficient to protect erroneous translations of backgrounds, does not inherently encourage (or empirically allows) the models to alter shapes. This is why Mo et al. (2018) introduced the idea of having a second generator network whose sole goal is to learn shape translations, while a second network would be in charge of combining the original image with the translation of its mask into a final translated image. The efficiency of this strategy however relies on the availability of "instance masks", a binary supervision signal making the approach semi-(un)supervised.

1.3.6 Domain Adaptation

In section 1.2.5 we introduced the concept of Transfer Learning, a subfield of machine learning focused on learning procedures aimed at transferring knowledge from one problem to another. A specific instance of Transfer Learning is called Domain Adaptation, where the task to solve is the same but the data sources used are different (Pan and Yang, 2010). Even models trained on large generic datasets can have a hard time transferring knowledge to a related but different data distribution (Donahue et al., 2013). The difference between the data distributions of both problems we are trying to solve is called the *domain shift* or *domain gap*. The domain for which we have knowledge to transfer from is called the *source* domain $(\tilde{\boldsymbol{X}}, \tilde{\boldsymbol{Y}})$, while the domain to which we want to transfer knowledge is called the target domain $(\boldsymbol{X}, \boldsymbol{Y})$. Domain Adaptation deals with situations where $p(\boldsymbol{Y}|\boldsymbol{X}) \approx p(\tilde{\boldsymbol{Y}}|\tilde{\boldsymbol{X}})$ but $p(\boldsymbol{X}) \neq p(\tilde{\boldsymbol{X}})$. For instance we might want to perform object recognition on data gathered by a different device, allow a robot to explore the outdoors while trained indoors or train a segmentation model on synthetic data and deploy it in the real world (Gopalan et al., 2011; Hoffman et al., 2017). The availability of labels in the target domains conditions the family of domain adaptation tasks, in this section, we will explore how GANs can be used to perform unsupervised domain adaptation in the context of visual tasks, meaning when no labels are available in the target domain.

Domain Classifier An idea very similar to GANs – where a generator network attempts to fool a discriminator networks – is that of domain classifiers with a gradient reversal layer, introduced in Ganin and Lempitsky (2015). In this paper, a standard classification model (feed-forward or convolutional) is split into a feature extractor h which produces features $\boldsymbol{f} = h(\boldsymbol{x})$ and a label predictor which classifies this intermediate representation $y = G(\boldsymbol{f})$. The idea is to add a domain classifier to the training procedure C which takes features as inputs and should output the domain label for the features' initial

[9]Where $\odot$ symbolizes an element-wise multiplication, or Hadamard product.

domain, $h(\boldsymbol{x}) = \boldsymbol{f}$ *vs* $\tilde{\boldsymbol{f}} = h(\tilde{\boldsymbol{x}})$. Instead of back-propagating the gradient from C through h directly (as is the one from G), it is first "reversed" by multiplying it by a negative constant. Effectively, this means that the loss is minimized when C is wrong, entailing that the intermediate representations created by h are *domain-agnostic* because they cannot be (easily) separated. Such techniques focusing on intermediate representation are called *feature-level* domain adaptation methods.

Translate to Adapt Given the previous section on using GANs to translate images, one might seek to bridge the domain gap by translating target images of the source domain where labels are available to make them look like images of the target domain where they are not. This straightforward idea is developed in Bousmalis et al. (2016): a conditional GAN (G, D) is trained to transform source samples into the target domain $G(\tilde{\boldsymbol{x}}, \boldsymbol{z})$ and the classifier is trained on both the fake target data $\{G(\tilde{\boldsymbol{x}}, \boldsymbol{z}), \tilde{y}\}$ and the source data $\{\tilde{\boldsymbol{x}}, \tilde{y}\}$. Authors argue that while T could be trained solely on fake target data, training also on source data stabilizes training and prevents it from learning a permutation of labels[10]. The overall model is trained on a weighted average of a standard GAN loss and a task-specific loss for T. Such techniques focusing on the pixels of visual data are called *pixel-level* domain adaptation methods.

CyCADA While feature-level domain adaptation is able to control for high-level marginal distribution matching, it is not able to account for semantics, nor can it model aspects of low-level appearance variance which are crucial for the end visual task. On the other hand, pixel-level domain adaptation is harder to implement efficiently on larger images and while it is able to efficiently transform style (as CycleGAN can for instance) it should also preserve content and semantic information (no hallucination) (Hoffman et al., 2017). To leverage the strengths and mitigate the weaknesses of both families, Hoffman et al. (2017) introduce Cycle-Consistent Adversarial Domain Adaptation (CyCADA). The idea is to leverage the cycle consistency loss introduced by CycleGAN and augment it with a semantic consistency loss to perform domain adaptation of semantic segmentation models. More specifically, the goal is to train a semantic segmentation on simulated data with ground truth labels and adapt it to perform well on real images without real labels. To achieve so, $G_{S\to T}$ transforms a synthetic image into a real one, while $G_{T\to S}$ should reconstruct it (cycle consistency). The model they aim at training is F_T, the semantic segmentation model in the target (real) domain. To do so, it is trained with the source labels of the adapted source image and its features are adapted to match those inferred from real target images. In addition, the semantic maps of $\boldsymbol{x}$ and $G_{S\to T}(\boldsymbol{x})$ predicted by a pretrained source segmentation model F_S should be the same (semantic consistency). The outline of the CyCADA model and example inferences are presented in Figure 1.11.

AdvEnt One of the main drawbacks of CyCADA's approach is its complexity: it requires 2 generators, 2 discriminators and 2 semantic segmentation models at each time step. To circumvent this, the authors of AdvEnt (Vu et al., 2018) start from a simple observation: models trained only on source domain tend to produce over-confident, i.e., low-entropy, predictions on source-like images and under-confident, i.e., high-entropy, predictions on target-like ones. To bridge the domain gap between synthetic and real datasets, they propose to enforce high-confidence (low entropy) on the target domain as well. Let P be the semantic segmentation model, and $P(\boldsymbol{x})$ a prediction in the target (real) domain. Then the Shannon entropy (Shannon, 1948) is minimized by adding a MinEnt loss:

$$\mathcal{L}_{ent}(P, \boldsymbol{x}) = \underbrace{\sum_{h,w}}_{\text{sum over locations } (h,w)} \underbrace{-\frac{1}{\log C} \sum_{c=1}^{C} P(\boldsymbol{x})^{(h,w,c)} \log P(\boldsymbol{x})^{(h,w,c)}}_{\text{Shannon entropy of a location}}. \tag{1.36}$$

[10] The model could learn

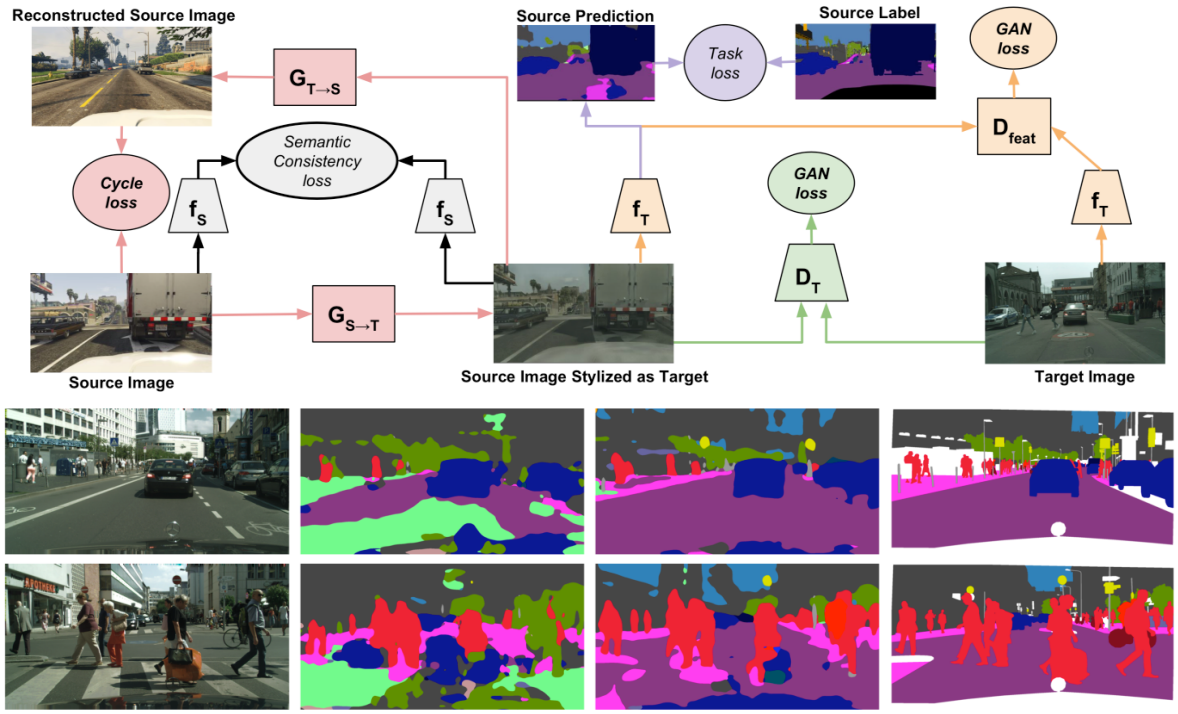


Figure 1.11: from Hoffman et al. (2017). **Top**: outline of the CyCADA procedure. **Bottom**: for 2 different target (real) images, comparison between the source-only segmentation model, CyCADA's prediction and the ground-truth labels.

However, as we have seen in CycADA's argument, a mere pixel-level adaptation as MinEnt is not enough to enforce structural dependencies between semantics of the adapted domains. To do so, they aim at aligning the weighted self-information distributions of target and source domains, thereby indirectly minimize the entropy of target predictions. Self-information (Tribus, 1961) is defined as $-\log P(\boldsymbol{x})^{(h,w,c)}$ and the weighted self-information[11] for a location is a vector $\boldsymbol{I} \in \mathbb{R}^C$ such that $\boldsymbol{I}(\boldsymbol{x})^{(h,w)} = -P(\boldsymbol{x})^{(h,w)} \odot \log P(\boldsymbol{x})^{(h,w)}$. Considering $\boldsymbol{I}(\boldsymbol{x}) \in \mathbb{R}^{H\times W\times C}$ as a "fake image" and $\boldsymbol{I}(\tilde{\boldsymbol{x}})$ as a "real image", one can use a discriminator network to adversarially bridge the gap between the domains. An illustration of this simple model (at least compared to CyCADA) can be found in Figure 1.12

[11] In that regard, weighted self information can be seen as a "disentangled entropy" as it provides the full distribution over classes instead of collapsing all the information into a single real number $E(\boldsymbol{x})^{(h,w)} = \mathbb{E}_c[-\log P(\boldsymbol{x})^{(h,w)}]$

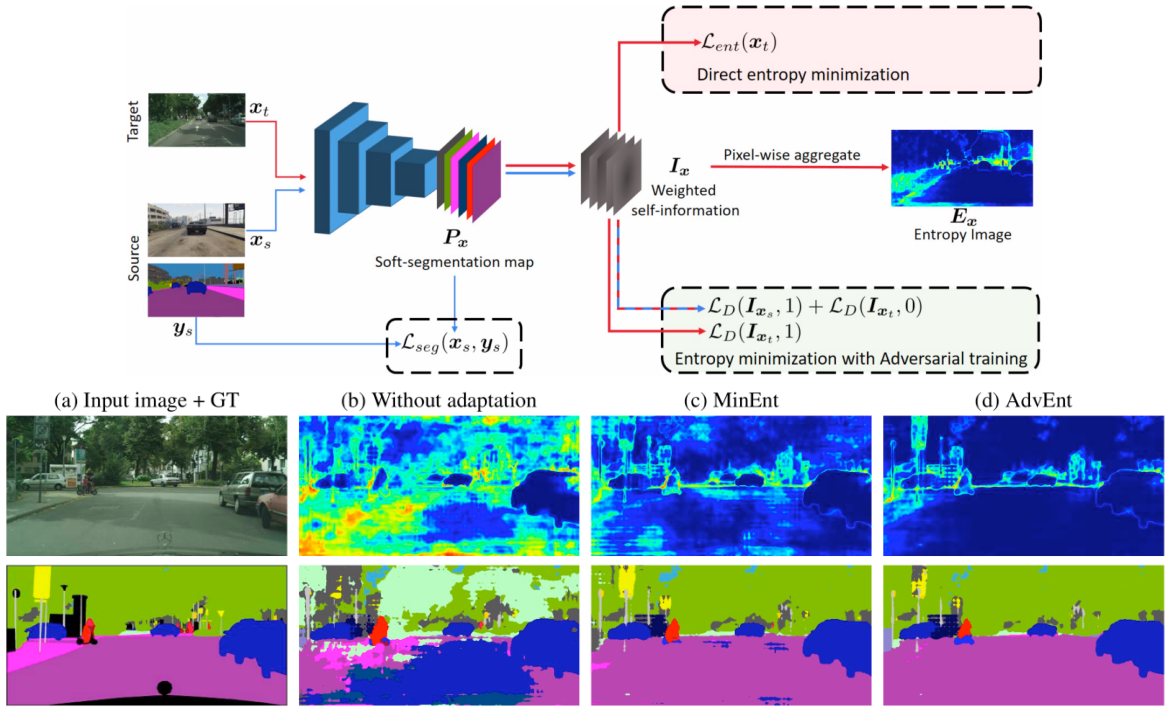


Figure 1.12: from Vu et al. (2018). **Top**: overview of the MinEnt+AdvEnt training procedure. **Bottom**: comparison of entropy maps (top row) and segmentation maps (bottom row) depending on the losses used.

Chapter 2

Visualizing the Impacts of Climate Change

Content presented in this chapter is the direct consequence of work published at ICLR 2019's AI for Social Good Workshop (Schmidt et al., 2019) and ICLR 2020's Machine Learning in Real-Life Workshop (Cosne et al., 2020).

2.1 Motivation

Content from a viewpoint article to be published in Jan. 2021 in the IEEE Computer Graphics and Applications Journal.

Historically, Climate Change has been an issue around which it is hard to mobilize collective action, despite the threat that it presents to our species and life on our planet. Researchers studying this phenomenon have systematically observed that effective communication on the subject of climate change arises from messages that are both emotionally charged and personally relevant, and that *images* in particular are key in increasing the awareness and concern regarding the issue of climate change (Wang et al., 2018a). Despite this assertion, many of the traditional forms of communication that are used by experts to communicate to the public are often based on scientific reports, which can fail to communicate the urgency and importance of this monumental phenomenon to the general public (Chapman et al., 2016). The aim of the project, VICC, is to use Artificial Intelligence to contribute to bridging this gap, and to create a tool to raise awareness with regards to the impacts of climate change.

2.1.1 Images as communications vector

Research in behavioral science has systematically shown that people's behaviors and attitudes can be affected by interactions with images that visually depict the future consequences of actions taken today. For instance researchers have found that young people tend to save significantly more money when presented with a decision calculator that project their pension amount, accompanied with a age-progressed photograph of themselves at retirement age (Hershfield et al., 2011). Likewise, middle-aged and older employees commit to putting aside more money for their retirement when the total amount is made more vivid and concrete, as a monthly amount to live on, with concrete examples of the objects that they can and cannot purchase (Goldstein et al., 2016).

The above studies show that when people are given evocative depictions of how actions today affect outcomes tomorrow, they are more likely to change their behavior. In particular, they often behave more like people who have thought about the issues at length. This emotionally grounded "decision aiding" approach maintains credibility by drawing on unbiased projections of the future and leaving freedom of choice to the (now better informed) individual. Conscious of the fine line between educating (i.e., by providing transparent and truthful information) and manipulating (i.e., using advertising or misinformation to influence outcomes), we propose to harness this approach, allowing viewers to time-travel into an imagined future to and visualize the severe impacts that climate change is likely to have by using an AI technology.

2.1.2 VICC: Overview

We focus our work on 3 common and visually striking climate-relate extreme events ("events" from now on): floods, wild fires and smog[1]. The goal is to create a website allowing users to query an address, of which we can fetch a first-person picture using Google's StreetView API and transform this image to illustrate events. By showing people what those events look like, we aim at creating empathy towards regions which already suffer from an increase in frequency and intensity of these events, and towards future generations which will have to handle a drastic change in our ways of life.

2.2 Generating Flooding Images

2.2.1 Overview of the Approach

Floodings are projected to occur mainly inland because of precipitations and faster snow melts or on costs because of sea-level rises (Dottori et al., 2016; Vousdoukas et al., 2018). It is therefore paramount to understand their consequences and help the public visualize what a flood would actually mean for them.

Inspired by the works of InstaGAN (Mo et al., 2018) we decided to split the architecture simulating floods in two: 1/ a model producing a binary mask, called the *Masker*, in charge of describing where the water should go, and 2/ a model focusing solely on rendering water given a mask and an image, called the *Painter*.

A key difference with the InstaGAN setting is that the latter focuses on transforming *instances* of a specific class into instances of another. But in the case of floods, what is the class which would need to be transformed? If we simply select the "ground" class or classes[2] of a semantic segmentation model, we'd only be able to simulate a flood of a few centimeters, hardly communicating the urgency of Climate Change. Moreover, in a picture where the ground is not horizontal, a sensible flood might not cover the whole ground but only the lowest part. Another challenge for the masker is that *instances* should often be only partially altered by our transformation: a flood-level of around $1m$ would hide parts of cars, buses, bicycles etc. but not completely hide those "objects". Lastly, because approaches like InstaGAN or CycleGAN (Zhu et al., 2017) rely on the reconstruction of a translated image, it discourages any loss of information such as hiding a major part of an image under water (we could see that cycle consistency-based models tended to generate transparent water in our early iterations). This is why, while taking inspiration from InstaGAN, we resorted to a somewhat more classical task: binary (pixel-wise) classification for the Masker.

Given a mask, the painter should create contextualized water: it is very important for the credibility of an image that water contains the appropriate reflections of surrounding objects, of the sky, and is coloured conditionally on the environment (light, exposure, presence of terrain in the original image etc.). The Painter's task is therefore to generate a new image *conditionally* on an input image and a binary mask. Because a binary mask (of where the water should go) is simply a semantic segmentation with 2 classes, we built our Painter model from the GauGAN (Park et al., 2019) architecture using SPADE conditioning modules.

2.2.2 Masker

Domain Adaptation The greatest challenge we initially faced developing the Masker was data: existing pictures of floods are rarely in 1^{st} person view, whereas at test-time images queried from Google StreetView (GSV) will be. Another limitation is that images collected online very rarely contain

[1] A "portmanteau" word made of the contraction of "smoke" and "fog", a phenomenon mainly due to air pollution by fossil fuels combustion.

[2] Some models may have several classes of "ground" such as "terrain", "grass", "road" etc.

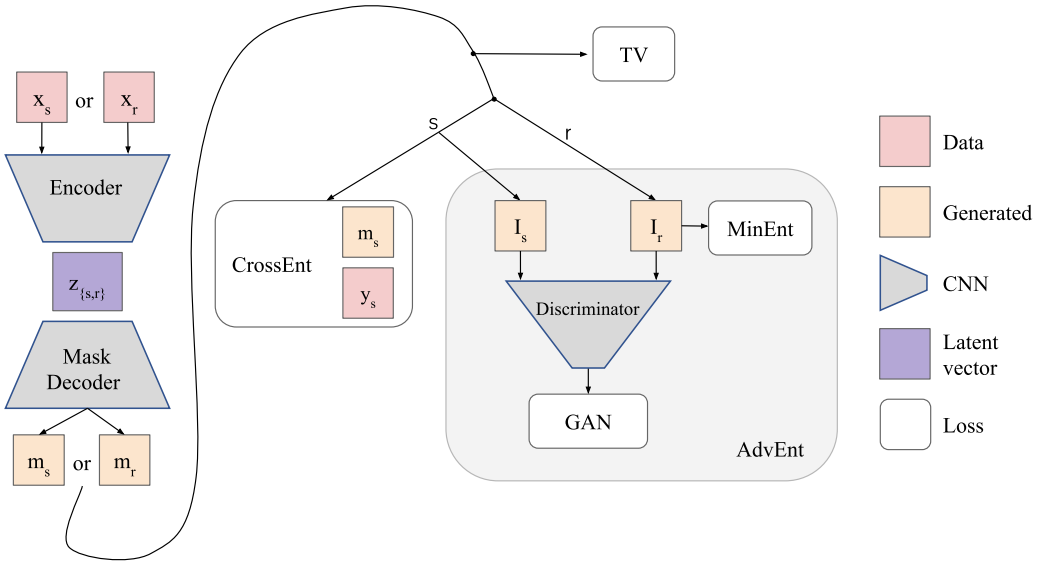


Figure 2.1: The Masker's training procedure. Input data (1^{st} person point of view of a street) $\boldsymbol{x}_r$ and $\boldsymbol{x}_s$ are passed through the Masker to produce $\boldsymbol{m}_r$ and $\boldsymbol{m}_s$, which are used to compute self-information maps I_r and I_s. The Masker is trained with a supervised Cross-Entropy loss in the simulated domain with ground-truth flood mask $\boldsymbol{y}_s$, and with a GAN objective in the simulated domain. The entropy of $\boldsymbol{m}_r$ is also minimized to encourage confident predictions and for both domains holes in masks are penalized with a Total Variation loss.

additional information we could want to leverage during training such as water height, geometry and semantic labels of the scene. Lastly, collecting pairs of images of the exact same location before and after a flood is not feasible, such pairs are very scarce.

All these reasons led us to endeavour to create a simulation allowing for all of this: 1^{st} person images, paired data before/after flood, multiple water levels for the same exact location, semantic labels and depth information. To be as close as possible to test-time, we set our simulated world's camera's parameters and height to match GSV's.

As emphasized in Section 1.3.6 we cannot simply train on simulated data and infer on real images without bridging the domain gap between those distributions. This is why we adapt the AdvEnt (Vu et al., 2018) strategy to our use-case: given a simulated image, the Masker should output a flood mask matching the ground-truth simulated flood (binary cross-entropy loss) while the self-information map ($\boldsymbol{I}$ in 1.3.6) of a real image's predicted flood mask is adversarially compared to that of a simulated image's prediction's.

Model Let $\boldsymbol{x}$ be an input image, $\boldsymbol{m}$ be a mask prediction with float values between 0 and 1, $\boldsymbol{y}$ a ground truth mask with values exactly 0 or 1, and I be $\boldsymbol{m}$'s self-information. Subscripts s mean that a given quantity comes from the synthetic world or is inferred from such an input, and subscripts r identify the real domain. Let E be an encoder network, M a decoder network and D a discriminator network. Both are convolutional neural network architectures. We then have $\boldsymbol{m}_s = M(E(\boldsymbol{x}_s)), \boldsymbol{m}_r = M(E(\boldsymbol{x}_r))$, $I_s = -\boldsymbol{m}_s \odot \log \boldsymbol{m}_s$ and $I_r = -\boldsymbol{m}_r \odot \log \boldsymbol{m}_r$. The model and its losses are illustrated in Figure 2.1.

Notations As per previous notations, the first dimension N is the batch dimention, the second C is the number of channels for an image input, H and W are respectively its height and width. To

simplify notations, sums over lower-case letters are taken over the full dimensions corresponding to their upper-case letters, *e.g.* $\sum_{h,w} \coloneqq \sum_{h=0}^{h=H-1} \sum_{w=0}^{w=W-1}$.

Losses For a batch of inputs $\boldsymbol{x}_s$ and $\boldsymbol{x}_r$, we start by computing $\boldsymbol{m}_d$ and I_d for $d \in \{r, s\}$. Depending on the canonical formulation of a given loss, masks $\boldsymbol{m}_d$ will be considered to be either 3D $N \times H \times W$ or 4D $N \times 2 \times H \times W$ tensors. Those shapes are equivalent: $\boldsymbol{m}_d$ being a binary mask, when a 4D tensor is needed we simply add 1 channel containing $1 - \boldsymbol{m}_d$ to represent the probabilities of the class "do not flood".

To enforce smoothness and discourage holes in the masks, we compute the Total Variation Loss of the masks, as argued for in Johnson et al. (2016):

$$\mathcal{L}_{TV}(\boldsymbol{m}) = \frac{1}{NHW} \sum_h \sum_w |\boldsymbol{m}_{h+1,w} - \boldsymbol{m}_{h,w}| + |\boldsymbol{m}_{h,w+1} - \boldsymbol{m}_{h,w}|. \tag{2.1}$$

In the simulated domain we have access to ground truth flood masks so we can compute the average pixel-wise cross-entropy to provide supervision to the Masker:

$$\mathcal{L}_{CE}(\boldsymbol{m}_s) = -\frac{1}{NHW} \sum_{n,h,w} y_s^{(n,h,w)} \log \boldsymbol{m}_s^{(n,1,h,w)} + (1 - y_s^{(n,h,w)}) \log(1 - \boldsymbol{m}_s^{(n,0,h,w)}). \tag{2.2}$$

Then to encourage confidence in the real domain predictions, we compute the MinEnt loss to minimize entropy (C=2 for binary classification):

$$\mathcal{L}_{ME}(\boldsymbol{m}_r) = -\frac{1}{NHW \log(C)} \sum_{n,c,h,w} \boldsymbol{m}_r^{(n,c,h,w)} \log \boldsymbol{m}_r^{(n,c,h,w)}. \tag{2.3}$$

We also add a Ground Intersection loss to encourage predicted masks in the real domain to at least cover the ground $\boldsymbol{g}_r$ as labeled by a pre-trained model:

$$\mathcal{L}_{GI}(\boldsymbol{m}_r, \boldsymbol{g}_r) = \frac{1}{NHW} \sum_{n,h,w} 1.0 \times ((\boldsymbol{g}_r^{(n,h,w)} - \boldsymbol{m}_r^{(n,h,w)}) > 0.5). \tag{2.4}$$

Lastly, we adversarially train the Masker to produce indistinguishable self-information maps using D and a WGAN loss with Spectral Normalization:

$$\mathcal{L}_{GAN}(M, D, \boldsymbol{x}_r, \boldsymbol{x}_s) = \begin{cases} -\frac{1}{N} \sum_n D(I_r^{(n)}) & \text{when updating } G, \\ -\frac{1}{N} \sum_n (D(I_s^{(n)}) - D(I_r^{(n)})) & \text{when updating } D. \end{cases} \tag{2.5}$$

The final loss is a weighted sum of these:

$$\mathcal{L}_{Masker} = \mathcal{L}_{GAN} + \lambda_{CE}\, \mathcal{L}_{CE} + \lambda_{MinEnt}\, \mathcal{L}_{ME} + \lambda_{TV}\, \mathcal{L}_{TV} + \lambda_{GI}\, \mathcal{L}_{GI}\,. \tag{2.6}$$

Data Our real dataset is a collection of images from various sources, mainly collected from GSV and Cityscapes (Cordts et al., 2016). We selected images representative of urban, suburban and rural areas with an emphasis on images from Cityscapes which resembled GSV's viewpoints. In total we have around 2900 real images. Examples can be found in the 'before' images of the Results section.

Our simulated dataset is collected from a $1km^2$ world created using Unity3D with a custom reflective water shader so that the rendered simulated water is as close to real world flooded scenarios as possible. Similarly to the real dataset, the simulated world contains urban, suburban and rural areas. Example images can be found in Figure 2.2. We selected around 2000 individual viewpoints in the world from which we performed 10x ground-truth 3D data augmentation: for each image captured at a view point, we also captured 9 other images with small random variations of angle and field of view.

(a) Bird's eye view of the simulated world

(b) Example urban flood rendering

Figure 2.2: The simulated world we built with Unity3D to train the Masker. It contains around 1500 viewpoints from which we extract ground-truth segmentation maps, depth maps and flood masks. We also perform ground-truth 3D data augmentation by capturing additional data from small perturbations around those viewpoints.

Results We train the Masker using the ExtraAdam optimizer (Gidel et al., 2018) with learning rates 10^{-5} for the Masker and 10^{-4} for the discriminator. Data is randomly cropped and resized in order to be 640×640 as GSV's API would give us. We can fit a maximum batch size of 8 in an NVIDIA RTX8000 with 48GB of memory.

We ran a random hyper parameter search to look for the best set of scaling factors: $\lambda_{CE} = 1$, $\lambda_{MinEnt} = 0.5$, $\lambda_{TV} = 1$ and $\lambda_{GI} = 0.05$. To compare models we compare the cross-entropy loss in the simulated domain, consider the Ground Intersection loss and *look* at validation inference in the real domain. In the context of our task, in absence of any real-world label, we cannot rely entirely on any specific metric (or combination of metrics) to compare real inferences. This is why the constitution of a good validation set is paramount. Understanding the Masker's failures on our validation set lead us to create a improve our initially urban-only synthetic world to include suburban areas, lawns, fences and large-sized trucks.

In Figure 2.3, we show successful inferences by the Masker. We can see that it is able to selectively flood specific objects: it masks out small objects entirely but only parts of taller objects, going appropriately around a phone booth, a car or a house. It is also able to understand the perspective and angle of an image, not merely drawing horizontal lines but rather following walls and the vanishing point of a road. The contrast with ground segmentation (HR-Net (Li, 2017) trained on the Cityscapes dataset (Cordts et al., 2016)) is striking and shows that even in the best-case scenario (the pre-trained model is actually very noisy on some StreetView images due to image wrapping and incongruous angles) the ground label is not sufficient to render a proper flood. What's more, the masks are dense as expected, without holes and with clear-cut edges.

In Figure 2.4 we show examples of the Masker's failure cases. The first picture shows that fences are hard to deal with and that the Masker might not be able to flood through wire meshes or create

non-connected components. The second and last rows illustrate how it can sometimes predict uneven flood levels. In the third to 5th row we can see the Masker was not able to delimit precisely pedestrians nor was it able to flood the road's right lane where cars are waiting when it can perfectly flood the empty left lane.

Multiple flood heights To strengthen our message and make a more compelling case for floods, we modified the Masker to make it handle multiple flood levels we could show people. To do so, we created more simulated data with a second, lower flood level of 50cm. To incorporate this difference in how the Masker processes images, we concatenate to the intermediate feature maps $\boldsymbol{z}_d = E(\boldsymbol{x}_d)$ a simple bit $b \in \{0, 1\}$, encoding which flood level the masker should produce. To ensure that the bit does in fact condition the mask generation we add an AC-GAN loss as per Equations 1.31 with a second discriminator which takes as input a mask, the flood-label bit and the context vector z_d so that it is able to judge whether the mask makes sense in the context of the original image. This new discriminator should output whether a mask looks "real" or not[3] on the one hand, and what the conditioning bit was on the other. At test-time we are not restricted to the 2 training values (0 and 1) for the flood-level bit. By interpolating linearly between 0 and 1, we can simulate a rising flood in the form of a GIF. This process and example results are illustrated in Figure 2.5.

[3]In this specific context, we do not mean "real" as in "simulated or real", but rather as in "generated by the masker or ground truth".

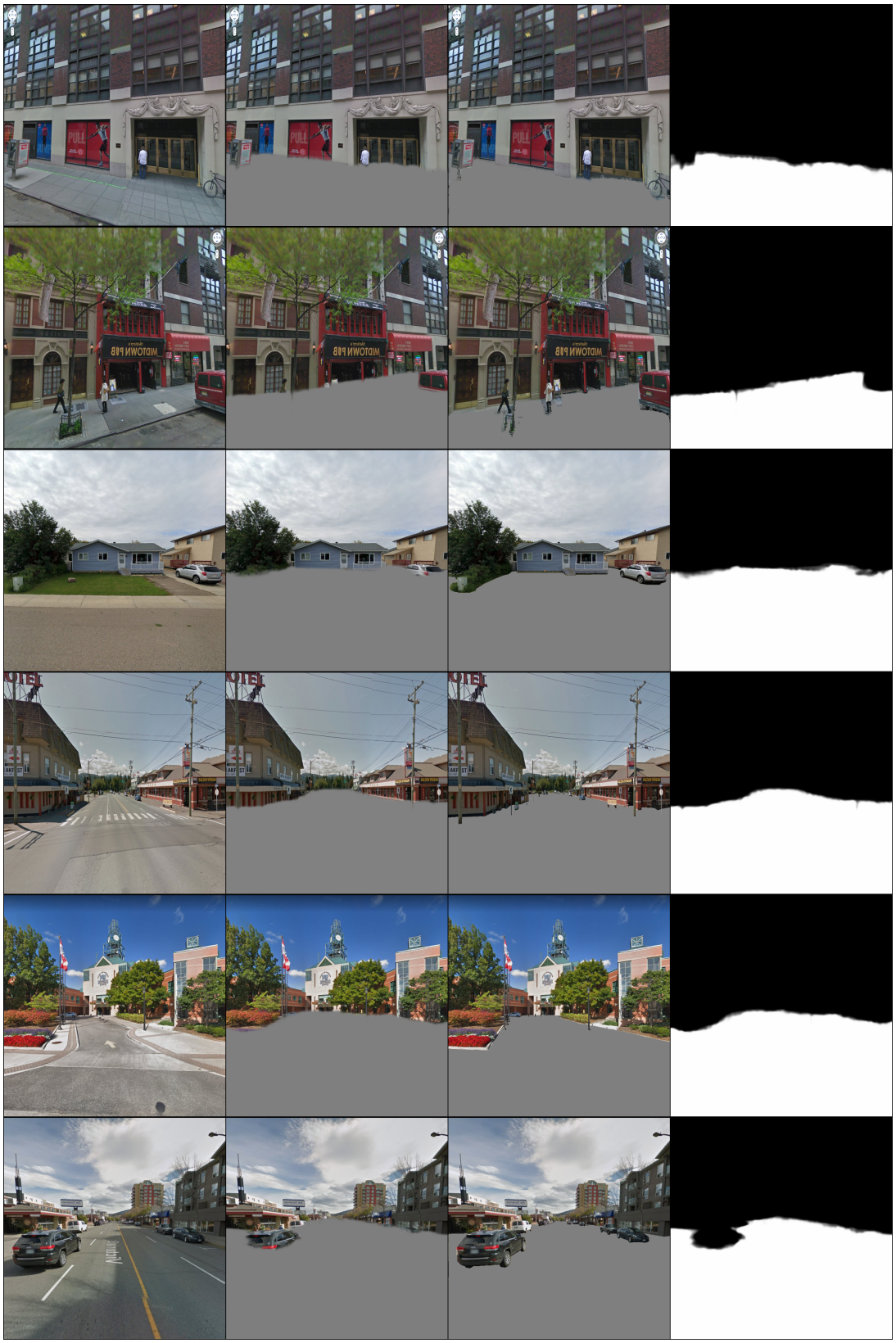


Figure 2.3: Example successful inferences from the Masker. Left to right: input image, predicted mask on the image, ground label from pre-trained model for reference, predicted mask alone. We can see it is able to accurately capture the angle of an image, circumvent objects and adapt to perspective.

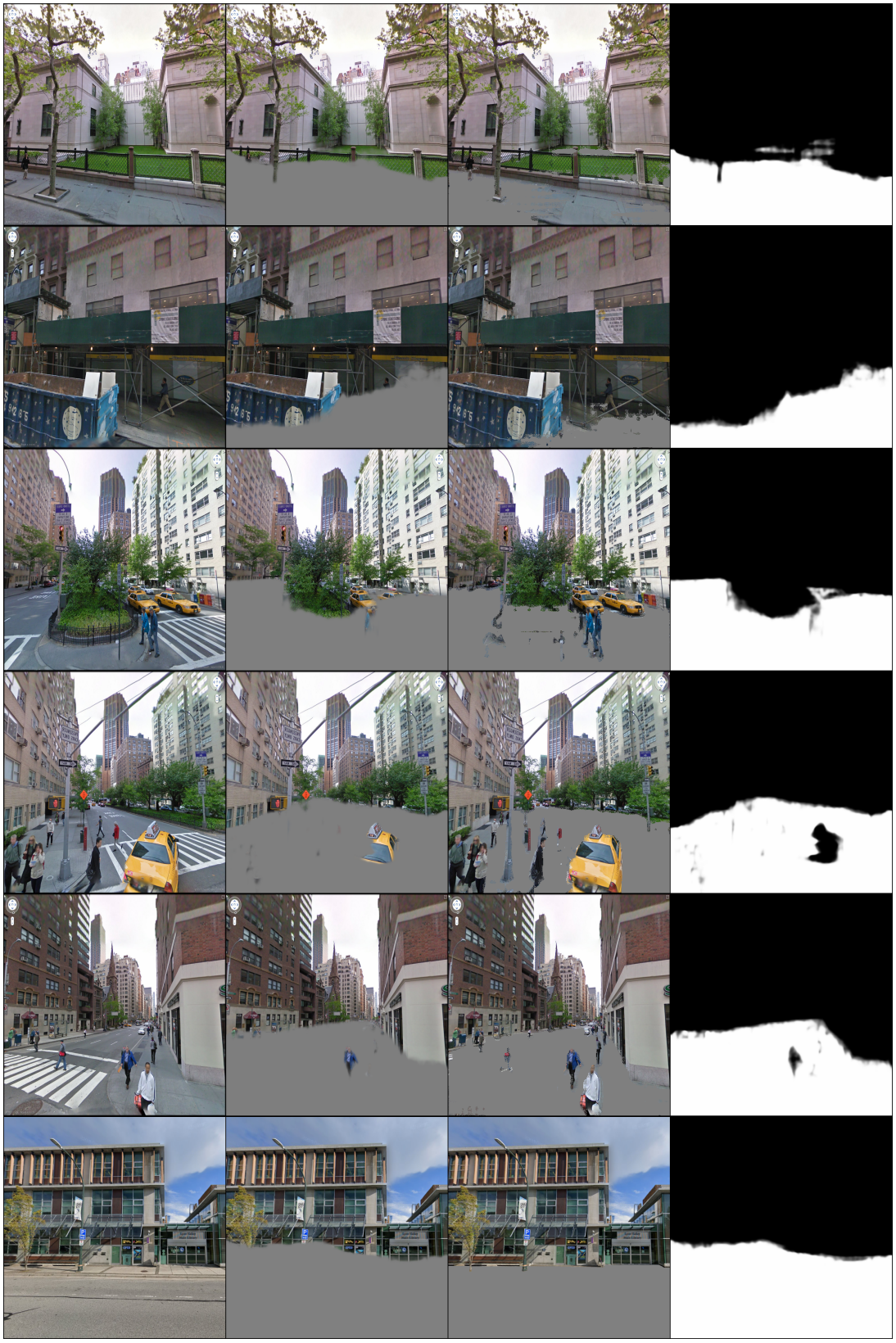


Figure 2.4: Example problematic inferences from the Masker. Left to right: input image, predicted mask on the image, ground label from pre-trained model for reference, predicted mask alone. In complex scenes, the Masker fails to precisely circumvent objects, flood through a fence and bluntly fails when presented with an out-of-distribution scene such as a construction site (this scenario is not present in the simulated world).

Figure 2.5: Multi-level masker. **Top**: the model's architecture, simplified to illustrate mostly the changes with respect to Figure 2.1. In particular, we illustrate the AC-GAN approach by emphasizing the double output of the discriminator, and its conditioning on the latent vector. We use $\boldsymbol{z}_i$ as it is less domain-dependant than the input image. **Bottom**: Results of linearly interpolating the flood-level bit between 0 and 1 with 10 values, yielding a continuum of flood-levels.

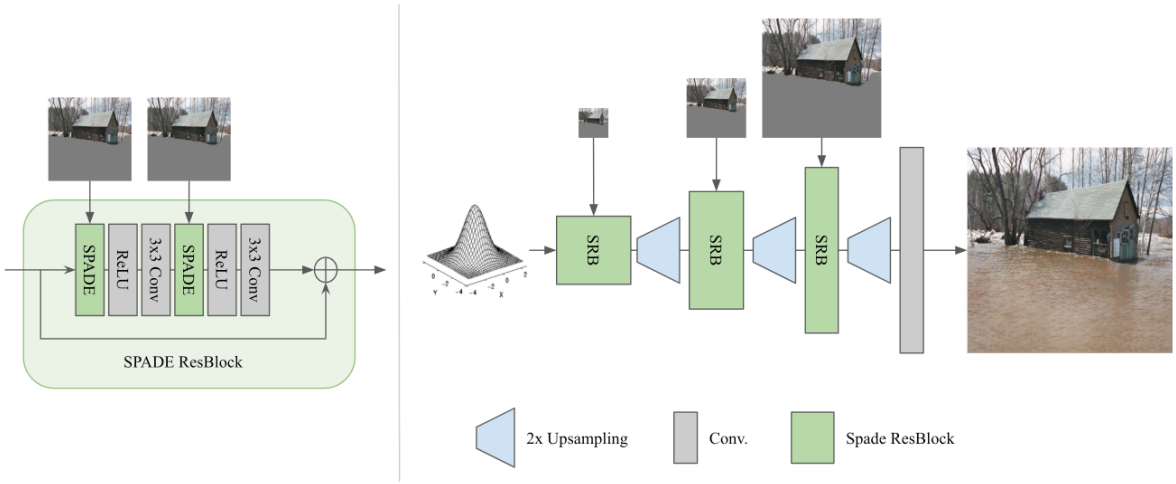


Figure 2.6: the Painter's architecture. **Left**: a SPADE-based residual block. In addition to illustrated layers, inputs to SPADE blocks are batch-normalized and convolutional layers are spectral-normalized. **Right**: the Painter's generator's architecture. Only 3 SPADE residual blocks and upsampling layers are illustrated, in practice we use 7 of these. The noise input to the generator is sampled with a shape $N \times 512 \times 5 \times 5$ so that, when upsampled it matches our target resolution of $640 = 5 \times 2^7$. The generator gets a masked image of a flood as input and should reconstruct the water.

2.2.3 Painter

Model As explained in Section 2.2.1, the Painter model is based off of the GauGAN (Park et al., 2019) architecture, leveraging the SPADE conditioning blocks. The core idea is that these blocks showed to be able to generate photo-realistic images based solely on segmentation maps. We therefore hypothesized that it would be able to do so on the mix of picture and segmentation that is a masked input. Since we have access to a binary mask, the final output of the Painter is pasted onto the input image to only change areas with generated water and keep the original context. This pasting procedure frees the Painter from learning the identity function outside the mask.

More specifically, the generator is a sequence of 7 SPADE-denormalized residual blocks followed by an upsampling with scale factor 2. At each level, an interpolation of the input with the appropriate size (height $\times$ width: $5 \times 5 \rightarrow 10 \times 10 \rightarrow ... \rightarrow 640 \times 640$) is given as conditioning image. The input is a masked image of a flooded scene: the water is hidden to the network which should reconstruct it. The generator's architecture is illustrated in Figure 2.6. Following GauGAN's implementation, we use a multi-resolution PatchGAN-based discriminator (Isola et al., 2017).

Losses Following the GauGAN implementation, the Painter has a combination of losses: a standard GAN loss, a perceptual VGG loss and a discriminator feature-matching loss. In addition, we introduce a masked reconstruction loss. In the following, let P be the painter, $\boldsymbol{z}$ a noise input, $\boldsymbol{x}$ a real image of a flood and $\boldsymbol{m}$ a binary mask (with 1s at locations where there is water in $\boldsymbol{x}$, indicating where the painter should generate water). Each of those input tensors have a batch dimension of N. For clarity, let us also note $\tilde{\boldsymbol{x}} = P(\boldsymbol{z}, (1-\boldsymbol{m}) \odot \boldsymbol{x}) \odot \boldsymbol{m} + \boldsymbol{x} \odot (1-\boldsymbol{m})$ an output image with generated water and original context.

The GAN loss is the same as in Equation 1.21. The VGG loss was introduced in Ledig et al. (2016) to complement pixel-level losses. The idea is that generated and real samples should not only look the same to a discriminator, but also to a model pre-trained on a visual task. Authors show that VGG (Simonyan and Zisserman, 2014) trained on ImageNet is a good choice for this perceptual loss. In practice, features maps at different layers ϕ_i (with output height and width H_i and W_i) in a VGG network computed from a real or a generated image should be close:

$$\mathcal{L}_{VGG}(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \frac{1}{N} \sum_i \frac{1}{W_i H_i} ||\phi_i(\boldsymbol{x}) - \phi_i(\tilde{\boldsymbol{x}})||_2^2. \tag{2.7}$$

In Salimans et al. (2016), authors advocate for another loss term called the Feature Matching Loss. The idea is that the real and generated distributions shouldn't only be indistinguishable to the output layer of the discriminator but also to its intermediary layers, so the generator is trained to match the discriminator's intermediate features' distribution on real data. Let h_i be an intermediate layer of the discriminator of output height and width H_i and W_i:

$$\mathcal{L}_{FeatMat}(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \frac{1}{N} \sum_i \frac{1}{W_i H_i} ||h_i(\boldsymbol{x}) - h_i(\tilde{\boldsymbol{x}})||_1. \tag{2.8}$$

Given our specific application where we care more about water quality than diversity, we introduce an L_1 loss encouraging generated water to match the ground-truth water. It also encourages the generated water to include relevant context such as sky color and reflections. We only apply this to the masked area of the input (the part which should be painted with water):

$$\mathcal{L}_{Reconstruction}(\boldsymbol{x}, \boldsymbol{m}, \tilde{\boldsymbol{x}}) = \frac{1}{NHW} ||\boldsymbol{m} \odot (\boldsymbol{x} - \tilde{\boldsymbol{x}})||_1. \tag{2.9}$$

The painter's overall loss function is a weighted average of those losses:

$$\mathcal{L}_{Painter} = \mathcal{L}_{GAN} + \lambda_{VGG}\, \mathcal{L}_{VGG} + \lambda_{FeatMat}\, \mathcal{L}_{FeatMat} + \lambda_{Reconstruction}\, \mathcal{L}_{Reconstruction}\,. \tag{2.10}$$

Results Example water generations are presented in Figure 2.7. We can see that the Painter is able to not only create realistic-looking water through ripples but also contextualize it realistically by creating object reflections and adapting the colour to the lighting conditions. Interestingly it is also able to-inpaint objects and ignore border artifacts as shown in Figure 2.8. On the other hand, though we find them to be quite rare, failure examples are presented in Figure 2.9. We can see in those generated images that water can sometimes have cartoonish colors and repeating ripple patterns making it obviously fake to a human eye. When it comes to validation, GANs are known to be hard to evaluate and compare. We have unfortunately found no correlation between human quality perception and standard metrics used to measure image quality. An example of this is shown in Figure 2.10: while the validation FID score of a model can be much smaller than that of another one, we can *see* that the images generated by the latter are much better. Yet again, we resorted to a manual and visual evaluation of models to select the final architecture and hyper-parameters.

2.3 Other Events

In addition to illustrating the potential consequences of floods, we endeavoured to create visualizations of two other climate change-related extreme events: smog and wildfires. For reference, real-world examples of what those events may look like are presented in Figure 2.11

2.3.1 Smog

Smog is a type of intense air pollution. The word "smog" was coined in the early 20^{th} century, and is a contraction of the words "smoke" and "fog" to refer to smoky fog due to its opacity, and odor. One of its main components is Ozone, which is present both naturally in the atmosphere and as a byproduct of two pollutants (nitrogen oxides and volatile organic compounds) that react in the presence of heat and sunlight at the ground-level. Emissions from chemical and industrial plants, electric utilities, refineries, exhaust from cars and trucks, and increasingly wildfire smoke and oil and gas extraction are sources of these pollutants. Ground-level ozone doesn't rise into the stratosphere, but builds up at the Earth's surface (Climate Central, 2019). On days with intense smog events, which happen most often in urban areas, a particular scene will look foggy, with a restricted depth of field of view as illustrated in Figure 2.11.

To illustrate the potential consequences of a smog event in a scene, it is paramount to make further objects more blurry and faded than objects in the foreground. The rate at which this should happened is modeled in Zhang et al. (2017) as a pixel's transmission $\boldsymbol{t}$. If we assume the smog's haze and the airlight[4] $a = 0.76$ to be uniform and using MiDaS (Ranftl et al., 2020) to predict a depth map $\boldsymbol{d}$:

$$\begin{aligned} \boldsymbol{t} &= e^{-\beta \boldsymbol{d}} \\ \boldsymbol{x}_{smog} &= \boldsymbol{t} \odot \boldsymbol{x} + (1 - \boldsymbol{t})a. \end{aligned} \tag{2.11}$$

In Equation 2.11 the parameter β is inversely proportional to the smog's intensity through the visible range R and we set it to match $R = 1km$. The process and example inferences are presented in Figure 2.12

2.3.2 Wildfires

This paragraph is an extract from Berger et al. (2020).

Wildfires are becoming more frequent and severe as a result of climate change, as forests in arid parts of the world become hotter and drier. In the mid-1980s, wildfires in the United States consumed just under two million acres a year on average; now, eight million acres burn each year. We can expect to see more wildfires in the future, especially in regions with a Mediterranean climate and low rainfall in the summer, such as the western U.S. and Canada, Spain and Portugal, Chile, and parts of coastal Australia.

Wildfire-prone regions are warming — the western U.S. has warmed as much as 1.9°C since 1912. This leads to more droughts that dry out forests. In California, more than 147 million trees have died due to extreme droughts since 2010. Dead plants and trees catch fire more easily than live plants, and provide more fuel for fires to burn. Climate change is also expanding the range of some invasive insects and plant diseases, creating more dead plants for fires to consume.

And there's a feedback loop: wildfires intensify climate change by releasing large amounts of carbon dioxide (CO2) and other greenhouse gases into the atmosphere. California's 2018 wildfire season released nearly as much CO2 as did the state's electric power plants that year. During Australia's 2020 bushfire season, wildfires pumped out more CO2 than the 116 least-polluting countries emit in a year.

[4]Part of a scene's ambient illumination is scattered by the haze particles into the camera as *airlight* that increases the intensity of the image

In light of the recent California wildfires, we decided to illustrate wildfires not with dire, dramatic and unlikely houses on fire, but rather with the more generic and widespread effect of red foggy skies as illustrated in Figure 2.11. Given an input image to transform, we first increase the contrast and the red colors. Then using a pre-trained segmentation model (HR-Net Wang et al. (2019)) we isolate the sky to make it uniformly red or orange. Because the segmentation may be problematic at its edges, we increase its range by a few pixels and finally apply Gaussian blur. Process and examples are presented in Figure 2.13

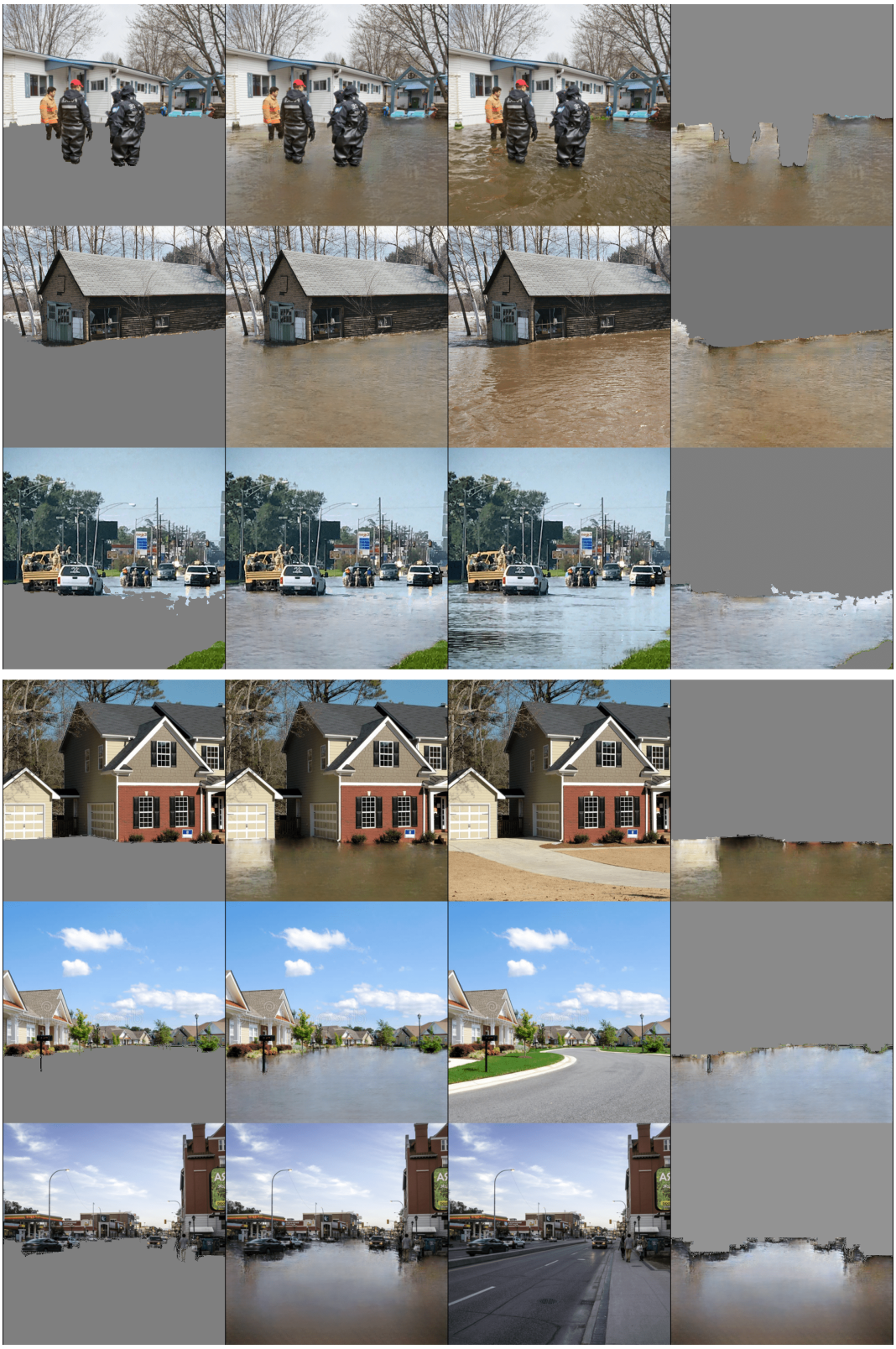
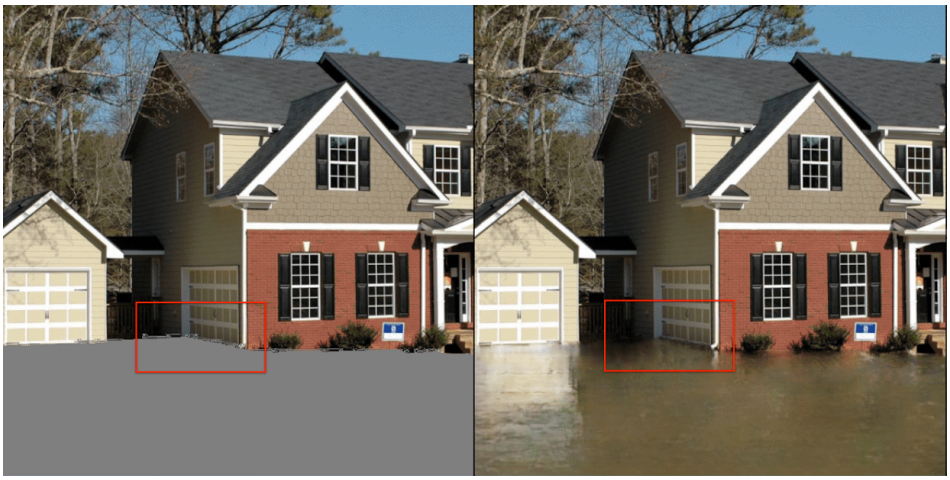


Figure 2.7: Example Painter inferences. **Top** three rows are from the training set (flooded images), **bottom** three rows are from the validation set (non-flooded images). The first column contains the masked input image; the second is the painter's generated water added to the input; the third is the non-masked input image and the last column shows the generated water only, without the original context. We can see in those non-cherry picked examples that the model is able to generate relevant and contextualized water, with the appropriate reflections and ripples.



(a)

(b)

Figure 2.8: Inpainting. The Painter is able not only to create realistic and contextualized water, it is also able to adjust its edges to match the actual image when masks have hard edges or artifacts. **(a)** Shows how the Painter chooses to in-paint edge pixels instead of adding water. **(b)** Illustrates a mask artifact in the middle of an objects which, again, the Painter in-paints instead of flooding it.

Figure 2.9: Examples of Painter failures. These are not prohibitively bad, and quite rare, but they show how the Painter sometimes repeats identical patterns and can be biased by a strong color: the yellow shop signs clearly have an impact on the generated water resulting in unrealistic colouring.

(a)

(b)

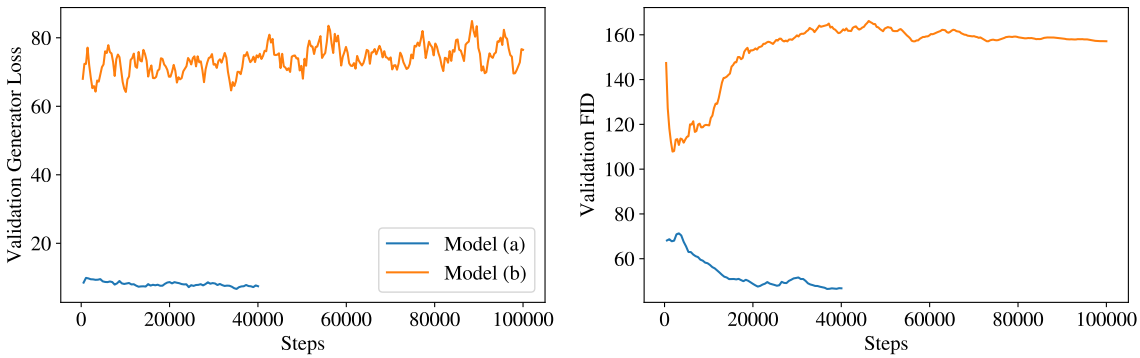
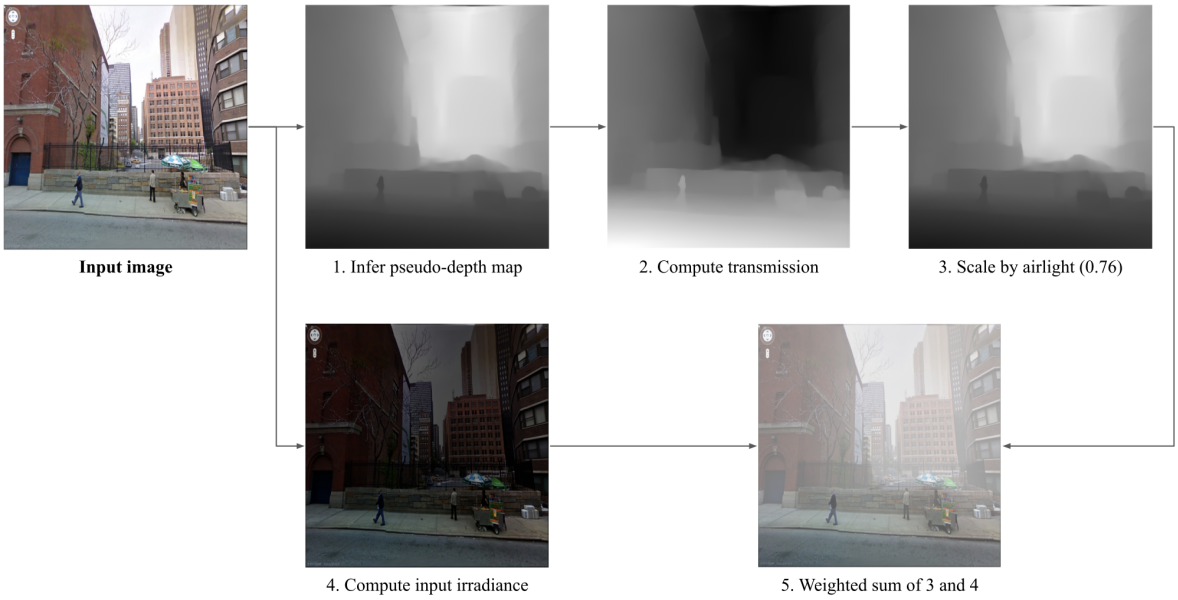


Figure 2.10: Comparison of the validation FID score and generator loss of two models. In both plots model (a) is the lower, blue curve while model (b) is the higher, blue curve. While one could expect better water quality from Model (a) based on the FID score and the loss, we can see that it is empirically not the case comparing actual images - only 2 are displayed here but we did verify this fact on more than 50 inferences. Models (a) and (b) differ in a number of ways, including losses, scaling factors and other hyperparameters. We also verified that one cannot simply reverse those criteria and consider for instance the largest FID as an indicator of the Painter's performance. These empirical observations make it very difficult to trust any metric other than visual quality of samples.)

Figure 2.11: Real images of smog and wildfires in urban and suburban areas. **Left**: 2020 smog pictures taken in Beijing, China and New Delhi, India. **Right**: 2020 California, USA wildfires.



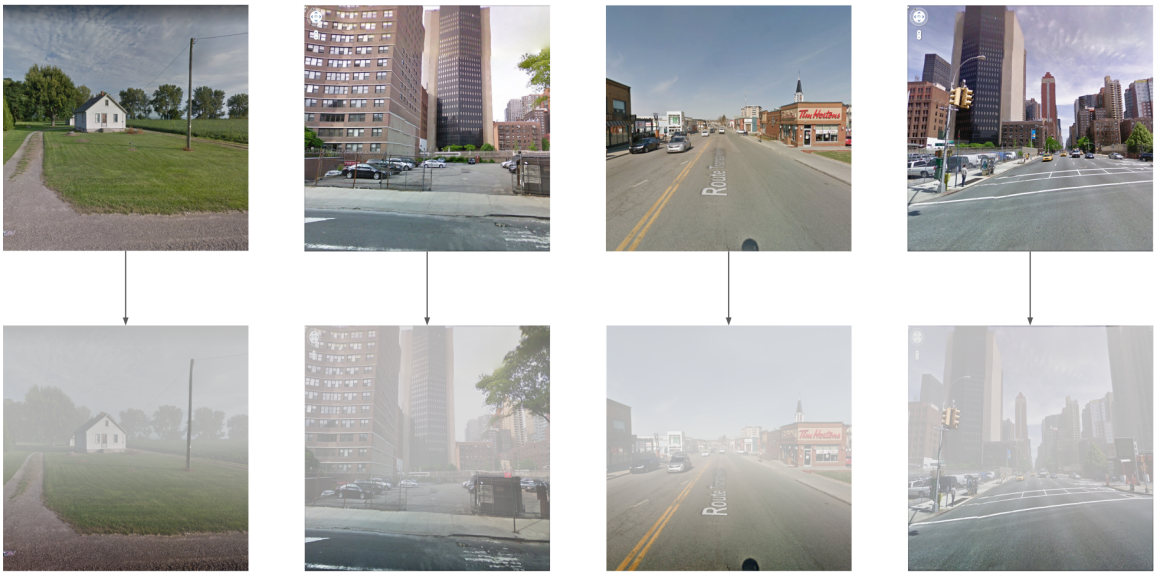


Figure 2.12: Creating smog illustrations using pseudo-depth maps inferred using MiDaS. **Top**: step-by-step illustration of the creation of a smog scene from an input image, using a pre-trained monocular depth estimation model. **Bottom**: example results of this process on various urban and sub-urban scenes. We can control the opacity of the smog by changing the β parameter in 2.11.

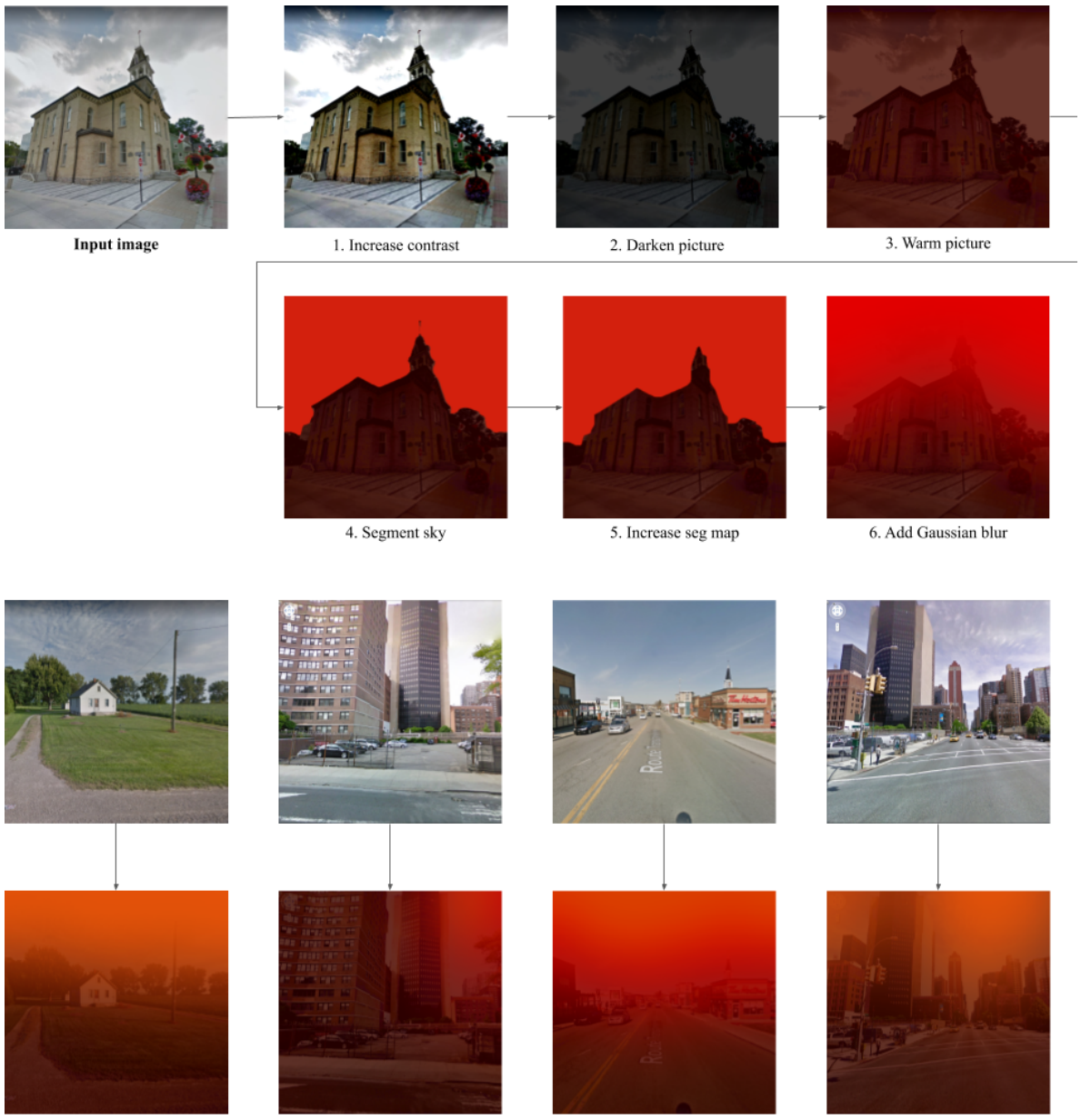


Figure 2.13: Creating wildfire simulations. **Top**: step-by-step process of transforming a street-view image in the real world into its potential equivalent if wildfires were to happen in its area. Using a pre-trained segmentation model, we are able to isolate the sky and treat it independently from the rest of the image. **Bottom**: illustration of the transformation on a variety of scenes with different shades of red.

Chapter 3

Quantifying the Carbon Emissions of Machine Learning

This chapter is an adaptation of work published at NeurIPS 2019's Climate Change AI Workshop (Lacoste et al., 2019).

3.1 Introduction

While a decade ago, only a few ML pioneers were training neural networks on GPUs (Graphical Processing Units), in recent years powerful GPUs have become increasingly accessible and used by ML practitioners worldwide. Furthermore, new models often need to beat existing challenges, which entails training on more GPUs, with larger datasets, for a longer time. This expansion brings with it ever-growing costs in terms of the energy needed to fuel it. This trend has been the subject of recent studies aiming to evaluate the climate impact of AI, which have predominantly put the focus on the environmental cost of training large-scale models connected to grids powered by fossil fuels (Strubell et al., 2019; Schwartz et al., 2019). While these models are not necessarily representative of common practice, we believe that it is important to continue this conversation further and work towards defining the tools and steps that we need to assess the carbon emissions generated by the models we train, as well as to propose ways to reduce those emissions.

In this work, we present our Machine Learning Emissions Calculator (https://mlco2.github.io/impact/), a tool for our community to estimate the amount of carbon emissions produced by training ML models. We accompany this tool with a presentation of key concepts and an explanation of the factors impacting emissions. Finally, we end our article with some recommendations of best practices for the overall ML research community, as well as for individual researchers.

3.2 Quantifying Carbon Emissions in Neural Network Training

In order to quantify carbon emissions, we use *CO_2-equivalents* (CO_2eq), which is a standardized measure used to express the global-warming potential of various greenhouse gases as a single number, i.e. as the amount of CO_2 which would have the equivalent global warming impact (Eggleston et al., 2006). We will use this single metric to compare the factors and choices that impact overall amount of emissions produced by training an ML model in the sections below.

3.2.1 Type of Energy Used

Practically speaking, it is hard to estimate exactly the amount of CO_2eq emitted by a cloud server in a given location because the information regarding the energy grid that it is connected to is rarely publicly available. However, if we assume that all servers are connected to local grids at their physical location, we are able to make an estimation of the amount of CO_2eq that they emit using public data

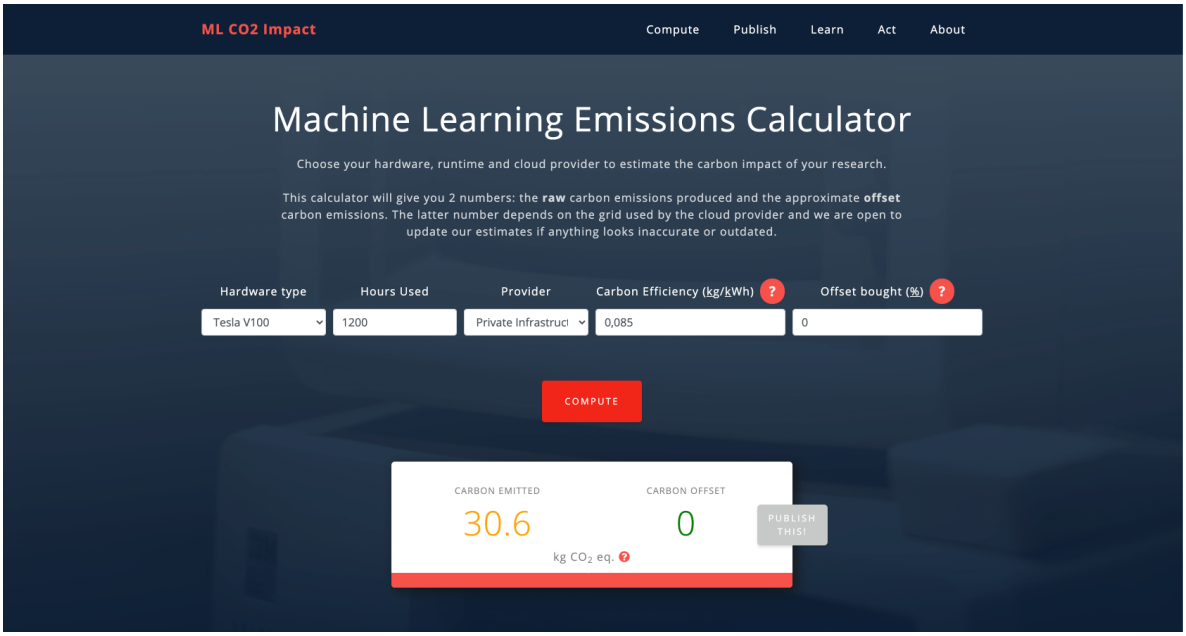


Figure 3.1: The MLCO2 Carbon Impact calculator taking as inputs time, hardware and either a Cloud Provider Region or a private infrastructure's grid efficiency.

```
\usepackage{hyperref}

\subsection{CO2 Emission Related to Experiments}

Experiments were conducted using a private infrastructure, which has a carbon efficiency of 0.(

Total emissions are estimated to be 30.6 kgCO$_2$eq of which 0 percents were directly offset.

%Uncomment if you bought additional offsets:
%XX kg CO2eq were manually offset through \href{link}{Offset Provider}.

Estimations were conducted using the \href{https://mlco2.github.io/impact#compute}{MachineLearn

@article{lacoste2019quantifying,
  title={Quantifying the Carbon Emissions of Machine Learning},
  author={Lacoste, Alexandre and Luccioni, Alexandra and Schmidt, Victor and Dandres, Thomas},
  journal={arXiv preprint arXiv:1910.09700},
  year={2019}
}
```

Figure 3.2: Automatically generated LateX snippet to encourage the reporting of emissions.

sources (To and Lee, 2017; Brander et al., 2011). Therefore, in order to create our emissions calculator, we gathered data regarding CO_2eq emissions of different grid locations and cross-referenced them with known GPU server locations from the three major cloud providers: Google Cloud Platform, Microsoft Azure and Amazon Web Services [1]. Our aim in doing this is to illustrate the degree of variability that exists depending on the location of a given server. For instance, in Figure 3.3, we show the distribution and variation in carbon emissions depending on geographical region. It can be noted that a large amount of variation can be found within a single region; for instance, servers located in North America can emit anywhere between 20g CO_2eq/kWh in Quebec, Canada to 736.6g CO_2eq/kWh in Iowa, USA (Brander et al., 2011).

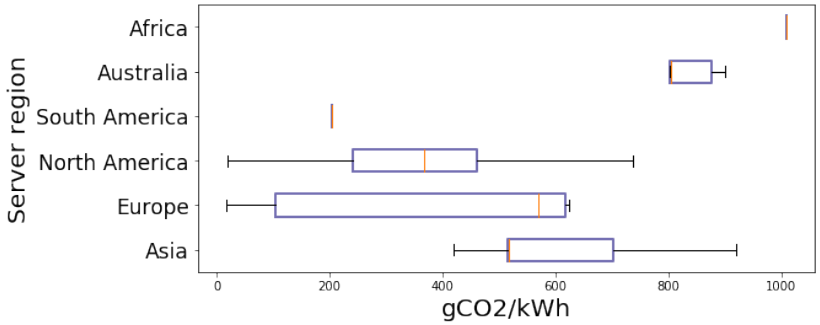


Figure 3.3: Variation of the Average Carbon Intensity of Servers Worldwide, by Region. (Vertical bars represent regions with a single available data point.)

3.2.2 Computing Infrastructure and Training Time

Another, more subtle, factor in carbon emitted by a neural network is the computing infrastructure used and training time of the model. In terms of performance the number of floating point operations per second (FLOPS) of GPUs has been steadily increasing in recent years, from 100 Giga FLOPS per second in 2004 to up to 15 Tera FLOPS per second in recent hardware (Harju et al., 2012). However, with neural network architectures becoming deeper and more complex, recent state-of-the-art models are often trained on multiples GPUs for several weeks (or months) to beat benchmark performance, requiring more and more energy.

Finally, when it comes to defining a training procedure for ML architectures, there are several elements to consider: for starters, whether it is necessary to train a model from scratch or whether fine-tuning is adequate for the task at hand. Notably, recent research has shown that using pre-trained models with task-specific fine-tuning performs as well as training from scratch, while being more robust, for tasks in image recognition (Tajbakhsh et al., 2016; Tsoumakas, 2019) and NLP (Howard and Ruder, 2018). Furthermore, when it comes to hyperparameter search, it has been proven both empirically and theoretically that random hyperparameter search is more efficient than grid search for hyperparameter optimization (Bergstra and Bengio, 2012), and there is much research being done on ways to improve the efficiency of hyperparameter optimization (Falkner et al., 2018; Feurer and Hutter, 2019), which makes it possible to continue choosing the right hyperparameters for new models without incurring superfluous computing and energy costs.

3.3 ML Emissions Calculator and Actionable Items

It is difficult to provide clear-cut guidelines for ML researchers to follow in order to reduce the carbon emissions, or specific benchmarks for the training time that a given model or task warrants. Nonethe-

[1]The data can be found at https://github.com/mlco2/impact/tree/master/data

less, we think that there are certain best practices and actionable items that can be adopted by our community to reduce environmental impact of the ML domain. We present some of these, along with our ML emissions calculator, in the current section.

Quantify Your Emissions Being informed regarding the factors that impact the quantity of carbon emissions produced by ML research is the first step to making positive changes. It is for this reason that we created our ML Emissions Calculator. This tool, currently in its alpha version, takes as input the details regarding the training of an ML model: the geographical zone of the server, the type of GPU, and the training time, and gives as output the approximate amount of CO_2eq produced. We collected publicly available data for the 4 main variables of this computation: (i) the energy consumption of hardware (see "Choose More Efficient Hardware" below), (ii) the location of providers' regions of compute – which we assumed to be connected to their local grid, (iii) the region's CO_2eq emissions per kWh and (iv) potential offsets bought by the provider.

We intend to adopt an open and transparent approach: the data we used is publicly available, debatable and editable through Github issues and pull requests. We are therefore open to updating data as more information becomes available. Since this paper's core goal is to raise awareness around the carbon emissions of ML, we have also included two educational sections in the website: one about learning the main notions and concepts related to this domain (e.g. RECs, carbon neutrality, etc.), the other about actionable items an individual or an organization can leverage to mitigate their carbon impact.

Choose Cloud Providers Wisely In recent years, many cloud providers have defined ambitious sustainability goals and are offsetting their emissions through Renewable Energy Certificates (RECs) in an effort to become carbon neutral, a term used to indicate a net zero carbon footprint of an organization. Each REC bought attests that 1 MWh of renewable energy has been added to the energy grid and can be used to offset an equivalent amount of non-renewable energy. For instance, Google Cloud Platform is certified carbon neutral and funds solar and wind farms directly on local grids through RECs (Google, 2018). Microsoft Azure is also certified carbon neutral and 44% of its electricity consumption directly comes from renewable energy, according to a 2016 estimate (Microsoft, 2018). Finally, to the best of our knowledge, while not yet 100% carbon neutral on an organizational level, Amazon Web Services is also funding renewable energy projects and some of their data centers are powered by renewable energy (Amazon Web Services, 2019).

Another major energy consumption factor of server installations is the power usage effectiveness (PUE) of the centers where the GPUs are hosted, which represents the percentage of energy consumption that is used for cooling, power conversion, and other auxiliary tasks, and can vary immensely. For example, Google Cloud Services has an average PUE of 1.1, meaning that only 11% of their total energy usage is not used for the servers themselves, a ratio that they have been steadily reducing using Reinforcement Learning (Gao, 2014; Google, 2018). Finally, if you rely on a local private compute infrastructure, it is also possible to engage with administrators about quantifying and offsetting the emissions produced, as well as improving the efficiency of your grid – this may help bring your organization toward carbon neutrality and have a significant impact at scale.

Select Data Center Location While many cloud providers are carbon neutral, some of their data centers may still be carbon intensive due to the local grid that they are connected to, whereas others will be low carbon and powered solely by renewable energy sources. Hence, selecting the data center location where an algorithm will be trained has a large impact on its direct carbon emissions. This choice can be achieved by consciously selecting the server location before dispatching your jobs. As we illustrated in previous sections, this single choice can make the direct emissions of an algorithm vary by a factor of 40, from 20g CO_2eq/kWh in a location that uses renewable energy sources to 820g CO_2eq/kWh in a location that solely relies on fossil fuels (Brander et al., 2011). For a model such as VGG (Simonyan and Zisserman, 2014) or BERT (Devlin et al., 2019), which are trained on multiple

GPUs for several weeks, this can correspond to avoiding emitting several hundreds of kilograms of CO_2eq by training on a server powered by hydroelectricity instead of fossil fuels.

Reduce Wasted Ressources Grid search is still often used in practice, in spite of its low efficiency both in terms of model performance and environmental impact. However, it has been shown that random search (and others) not only is a straightforward replacement but also has potential to significantly accelerate hyperparameter search (Bergstra and Bengio, 2012; Li et al., 2016; 2018; Falkner et al., 2018), consequently reducing carbon emissions. Also, while failed experiments are a common part of ML research and are sometimes unavoidable, their number can often be reduced with careful design such as unit tests, integration tests, and extensive and early debugging. Uninformative experiments are also frequent (sometimes unknowingly) – they can be caused by unstable learning algorithms requiring averaging results over many random seeds. Taking the time to carry out a literature review and to understand the potential sources of noise before launching large-scale hyperparameter searches increases the chance of obtaining reproducible and statistically significant results. Hence, reducing the need to extend the experiment cycles.

Choose More Efficient Hardware The choice of computing hardware can also have a major impact on ML emissions. To perform a comparison between different devices, their compute efficiency can be estimated in FLOPS/W. This estimation is based on their theoretical peak performance with respect to their Thermal Design Power (TDP)[2]. Using this approach, it can be found that CPUs can be 10 times less efficient than GPUs while TPU 3 can be 4 to 8 times more efficient than GPUs (Teich, 2018) (refer to Lacoste et al. (2019)'s Appendix B for details). Interestingly, in contexts where low power consumption and efficiency are important, e.g., for embedded applications, GPUs such as the Jetson AGX Xavier can be 10 to 20 times more efficient than traditional GPUs.

3.4 Discussion

The factors that we discussed in the current work give ML practitioners a certain amount of control over the environmental impact produced by the training of their models. We are aware that these choices are not always possible to make in practice – for instance, the choice of server location can be limited due to privacy considerations in the case of applications in the medical or financial domain, and large amounts of data may be needed to produce most robust models. However, we find that our emissions calculator is a good starting point to estimate the impact that small choices in model training can have on direct carbon emissions resulting from ML research.

Despite our best efforts, our calculator remains simply an approximation of the true emissions produced by ML training for several reasons: to start with, there is the issue of global load balancing, i.e. if a majority of practitioners choose to run their models in a low-carbon location, the servers will get saturated and other servers will still need be used. In that perspective, the global gain will not be a 40-fold reduction of emissions, but much smaller. Furthermore, there is a lack of transparency with regards to the true quantity of emissions produced by organizations, so while we use the current best publicly-available sources, there is still a large margin of error with regards to the exact quantity of energy consumed and carbon produced – we remain open to additional data sources and numbers. Finally, while in the current version of our tool, we focus on quantifying the emissions of training ML models, there is still the issue of deploying them, since the inference process is also energy-expensive, especially if done continuously and on a large scale. This is something that should be taken into account by ML practitioners in their products that are deployed in real-world settings, for instance by using energy-efficient architectures (Cai et al., 2017) and computing infrastructure.

There are also more far-reaching discussions to be had regarding the environmental value of scientific

[2]Empirical measurement of GFLOPS/W on various ML architecture would provide more accurate numbers but we are only interested in approximate values to compare classes of devices.

knowledge in general and of ML research in particular. On one hand, there is valuable research to be done in ML especially with regards to tackling climate change (Rolnick et al., 2019; Schmidt et al., 2019), whereas on the other hand, the emissions of the field of ML are growing quickly (Strubell et al., 2019). We do not propose the solution to this problem, but we believe that there are steps to be taken, for instance by using efficiency as an evaluation criterion (as proposed by (Schwartz et al., 2019)) or by taking concrete steps to reduce emissions (as proposed by the current paper). We hope that our work, along with others, will open the door for these conversations and debates to take place, to quantify the environmental impact of our field, and for positive changes that can be made to reduce it.

3.5 Code Carbon

As a follow-up on the above work published at NeurIPS 2019's Climate Change AI workshop, we have since worked on a tool to make reporting easier and more systematic. CodeCarbon is a Python package which tracks GPU energy consumption, maps it to a region and can therefore give effortlessly the user an estimate of their carbon emissions. Emissions are aggregated in a single CSV file allowing for an easy and instantaneous overview of an entire project's emissions. In order to encourage this systematic tracking and reporting of carbon emissions, we have also partnered with the dashboard company Comet.ml to make emissions tracking default for their users. Moreover, to make numbers more accessible and meaningful to the general public and AI practitioner we include a dashboard with exemplary equivalents as illustrated in Figure 3.4.

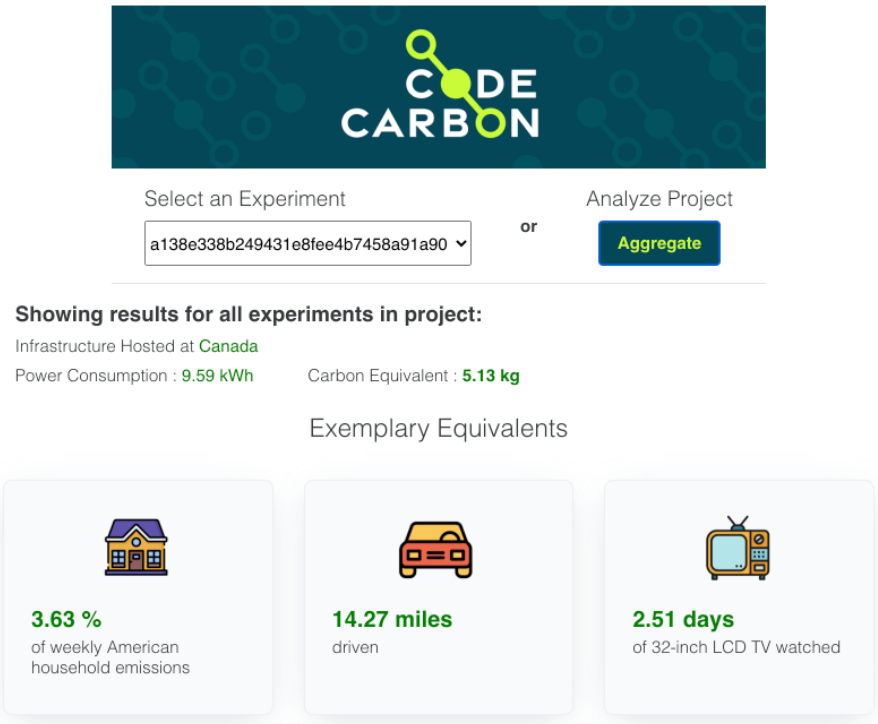


Figure 3.4: Illustration of the panel integrated with Comet.ml's dashboard.

Chapter 4

Modeling Cloud Reflectance Fields using Generative Adversarial Networks

This chapter is an adaptation of work published at ICLR 2020's Climate Change AI Workshop (Schmidt et al., 2020).

4.1 Introduction

Global Climate Models (GCMs) are one of the most important tools available to understand and anticipate the consequences of climate change, including changes in precipitation, increases in temperatures, and acceleration in glacial melting (Stocker et al., 2013). One of the key physical principles these models rely on is the Earth's energy balance (North et al., 1981): in short, the difference between how much energy the earth receives and how much it emits. In this context, it is paramount to model clouds accurately as they both reflect energy coming to the Earth and the infrared radiations it radiates (Ramanathan et al., 1989). However, as physical processes at play in cloud composition and evolution typically range from 10^{-6} to 10^{6}m, direct simulation of their behavior can consume up to 20% of a GCM's computations - depending on their time and spatial scales (Arakawa, 2004; Bretherton, 2015; Schneider et al., 2017). Various efforts have tried to address this challenge. This includes traditional approaches that incorporate domain knowledge to build and validate model hypotheses using observations as well as sub-grid cloud modeling (known as super-parameterization). Alternatively, recent machine learning approaches use meteorological variables to model sub-grid clouds, thereby reducing the computational cost of super-parameterization (Brenowitz and Bretherton, 2018; Rasp et al., 2018; O'Gorman and Dwyer, 2018; Yuan et al., 2019).

In this paper, we extend Yuan et al. (2019), a data-driven approach to contribute to cloud modeling, focusing on one of the main features used in energy balance calculations: reflectance fields. We use Conditional Generative Adversarial Networks (Isola et al., 2017) to generate these reflectance fields conditioned on meteorological variables[1]. We suggest using these generated images to extract important cloud parameters such as optical depth. We believe our approach is a step towards building a data-driven framework that can reduce the computational complexity in traditional cloud modeling techniques.

Our goal is to model reflectance fields, which in turn could be used as a proxy for cloud optical depth, a major component of GCMs' energy balance computations (Hartmann et al., 1992; Corti et al., 2005). To do so, we leverage 3100 aligned sample pairs $\mathcal{X} = \{r_i, m_i\}$, where meteorological data m_i are collocated with reflectances r_i. Each m_i is a $44 \times 256 \times 256$ matrix, representing 42 measurements from MERRA-2 (Gelaro et al., 2017) (see Schmidt et al. (2020) Table 1) along with longitude and latitude to account for the Earth's movement relative to the satellite[2]. On the other hand r_i, is a $3 \times 256 \times 256$ matrix representing each location's reflectance at RGB wavelengths (680, 550 and 450 nm) as measured by the Aqua dataset (Platnick et al., 2016). One could consider working in a Supervised Learning setting to learn a deterministic mapping $f : m_i \mapsto r_i$; however given the chaotic nature of climate, we need not a point estimate of the potential cloud distribution on earth, but rather an ensemble of likely scenarios given initial conditions. This motivates a generative approach using conditional GANs.

[1]The code is available on Github: https://github.com/krisrs1128/clouds_dist

[2]As the earth rotates, the actual geographical locations on Earth change pixel position in the data.

4.2 Modeling reflectance fields

4.2.1 Network

Architecture : motivated by Ronneberger et al. (2015), we use a U-Net as conditional generator. The U-Net architecture helps our generator capture global context, and skip connections allow localization. All of the convolution modules in our U-Net implementation consist of the same building blocks: a 3x3 convolutional layer followed by padding - which eliminates the need for cropping - followed by batch normalization, leaky ReLU, and a dropout layer with 0.25 probability.

Source of stochasticity : we introduce stochasticity in the generator only through the dropout layers at both training and test times, i.e we do not use noise input vectors. As observed by Isola et al. (2017) dropout introduces sufficient diversity in the output of conditional GANs

Checkerboard artifacts : a direct implementation of this generator results in checkerboard artifacts, a result of the use of transposed convolutions to upsample the feature maps in the U-Net expansion path (Odena et al., 2016). We solve this problem by replacing transposed convolution with a resize operation of the feature maps using 2d nearest neighbor interpolation, followed by a convolution as proposed in Odena et al. (2016).

Discriminator : we use a multi-scale discriminator as proposed by Wang *et al.* in Wang et al. (2018b). This introduces 3 discriminators with identical network structure operating on different input scales: one discriminator operates on the raw input image, while the other two operate on the raw image downsampled by factors of 2 and 4, using average pooling with a stride of 2. The motivation behind using discriminators at different scales is to provide the generator with better guidance both in the scale of global context and finer details in the image.

4.2.2 Training

Training objectives To train our generator, we use a weighted objective function composed of two losses: a non-saturating GAN loss and a matching loss:

1. Non-saturating adversarial loss. We experimented with two types of adversarial loss: the hinge loss of Lim and Ye (2017) and the least square loss (LSGAN) of Mao et al. (2017). We observe better performance with LSGAN. In Schmidt et al. (2020)'s Figure 6 (Appendix B, not reproduced here), we also see that least squares loss is more stable during training.
2. L^1 matching loss. We use L^1 loss between generated and true reflectance images, which encourages the generator to produce outputs close to the observed images from a regression perspective. The L^1 loss has been found to produce less blurry outputs than L^2 loss Isola et al. (2017).

Optimizer As we explored various optimization strategies and regularization methods, we observed significant improvement both in terms of convergence and in the quality of the generated output by using the Extra-Adam[3] method proposed by Gidel et al. (2018) compared to Adam and SGD, see Schmidt et al. (2020)'s Figure 7 (Appendix B, not reproduced here).

[3]See code at https://github.com/GauthierGidel/Variational-Inequality-GAN

4.3 Results and Discussion

4.3.1 Visual analysis

Our U-Net generator, trained against a multi-scale discriminator and optimized by Extra-Adam, is able to generate visually appealing CRFs that are difficult to distinguish from true samples. On a validation set of ground truth images, we obtain an ℓ^2 loss ~ 0.027. Figure 4.1 shows 4 different pairs $(G(m_i), r_i)$: we can see that the model is able to pick up large-scale cloud structures as well as the continents and oceans beneath them. Although not as precise as the ground-truth r_i, the generated samples exhibit similar global composition as well as local structures.

In Figure 4.2, we generate 3 reflectance fields from the same conditioning measurements. We notice a consistent global pattern in the three samples, with variations visible in finer details. In order to quantitatively measure diversity across generations, we fix the validation set to 5 samples that are selected manually to capture different regions of the rotating earth and generate 15 samples in total: 3 for each validation sample. For each set, we compute 3 metrics: pixel-wise mean, standard deviation and inter-quartile range across samples. Figure 4.2 shows that the model can obtain high image quality and proximity to the original distribution, but only at the cost of low diversity.

Our model still has limitations, such as blurriness and small size checkerboard artifacts. We believe the reasons for this are:

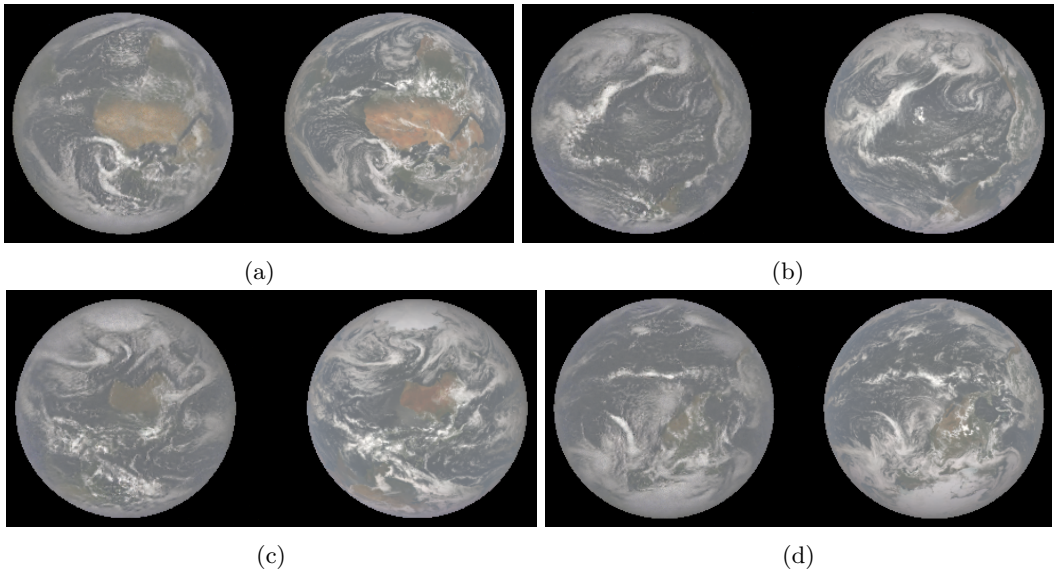
1. More training samples are needed to represent such a high dimensional distribution, i.e 3100 samples are not enough to train a deep U-Net generator ($\sim$ 1.4 million parameters) and discriminator ($\sim$ 8.3 million parameters).
2. More hyperparameter tuning, including architectural choices of the generator and discriminator to ensure the right capacity balance that lead to a long lasting GAN training game and avoid prematurely saturated learning.
3. Further training – we can see that the discriminator loss still slightly oscillates after saturation points and eventually decreases with number of steps as shown in Schmidt et al. (2020)'s Figure 10.c (Appendix B, not reproduced here).

4.3.2 Spectral analysis

Although visual inspection techniques can give insight into GAN performance, it is an expensive, cumbersome, and subjective measure (Borji, 2019). We address this issue by comparing the frequency spectrum of true and generated samples using 2D Discrete Fourier transform (DFT). This allows us to compare the images' geometric structures by examining the contribution of frequency components (Gonzalez and Woods, 1991). We compare the the magnitudes of the 2D DFT calculated from the grayscale versions of the true and generated images, and compare the histograms of the calculated magnitudes, their means, variances and the logarithmic average L^2 distances. In figure 4.4 we observe that our generated images have consistent and similar DFT distributions to those of their corresponding true images, with a very small average L^2 distance.

4.4 Conclusion and future work

We show that using conditional GANs to model CRFs can be an effective approach towards building a data-driven framework. We think our approach could significantly help improve the computation time of clouds modeling in global climate models.
Future work includes increasing the size of the dataset and exploitation of the temporal structure in our



(a) (b)

(c) (d)

Figure 4.1: 4 inferences obtained from our trained model. Generated images are on the left and the corresponding true images on the right. We can see how composition is preserved, most large cloud fields have similar shapes but differ in the details.

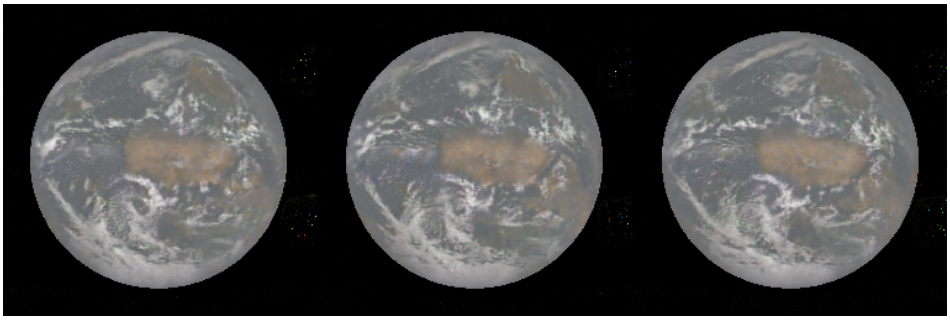


Figure 4.2: Reflectance fields generated by conditioning on the same input (noise comes from dropout, which is kept at test time).

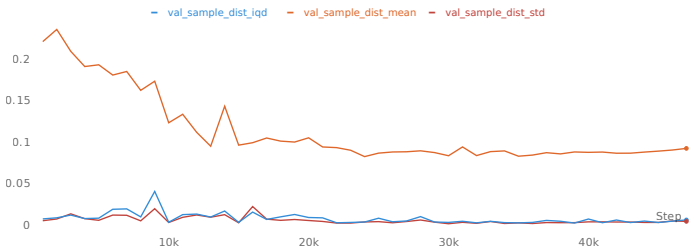


Figure 4.3: Plot of the inter-quartile distance, mean, and standard deviation of 15 generated samples at different steps during the model's training.

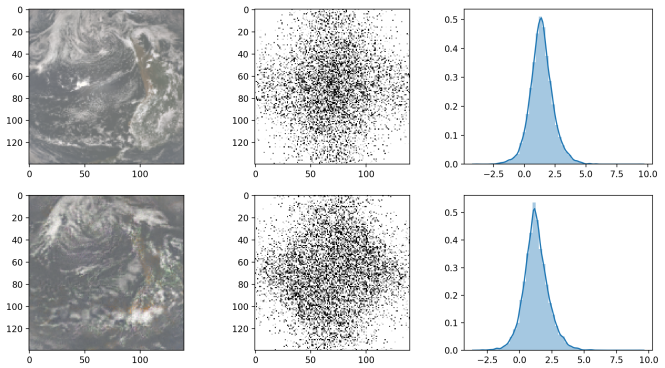


Figure 4.4: Comparison of DFTs (from left to right: image, frequency magnitudes, histogram of magnitudes), with real data in the top row and a generated reflectance field (from the real data's associated measurements)in the bottom one

data in two ways: by adding date and time as extra labels to the input variable, and by using temporal cross validation (Varma and Simon, 2006) to validate our generator's ability to predict possible changes in cloud distribution over time. We also plan to increase the diversity in the generated ensembles by incorporating input noise channels as an extra source of stochasticity. To address what we suspect to be mode collapsing in our network (the matching loss discourages the exploration of other potential modes in the data) we suggest using staged training where an adaptive weight for the matching loss encourages the generator to regress onto true images during early stages of the training, eventually decreasing to zero as training progresses.

Chapter 5

VICC: Ongoing and Future Work

5.1 OmniGAN

We are currently working on a model called OmniGAN, which we envision will leverage all possible and available data, creating images of all of the extreme events we want to display in a single forward pass through a unified model. This would not only save inference time and be more elegant, but we also believe that combining specific tasks together would improve the performance of the overall model. The goal of this section is to present ongoing work to bring together ideas and models presented in Chapter 2 in order to leverage simulated and real data, images of flooded and non-flooded areas, pre-trained models and pseudo-labels, and eventually combining the masking, painting, segmentation and depth prediction tasks.

5.1.1 Pseudo Supervision for GANs

As we have emphasized in section 2.2.2, understanding the geometry of a scene is paramount in yielding appropriate flood masks. This is also an observation that the authors of Depth-Aware Domain Adaptation (DADA, Vu et al. (2019)) make and leverage to improve the performance of their segmentation model trained under the AdvEnt paradigm, augmented with a depth-prediction head. The idea is that by forcing the model to perform depth prediction in addition to semantic segmentation using shared weights for most of the 2 prediction heads, it will encode relevant information for both tasks and therefore make depth information available to the segmentation head. Through an operation they call "feature fusion", authors merge the features of the depth head into the segmentation head for it to pay a stronger attention to closer pixels.

We want to push these ideas forward in three ways:

1. Incorporating the DADA idea into our Masker model, we can hope to improve its performance by making depth information available to it
2. We could actually include a third decoder to predict a segmentation map: now geometric and semantic information would be available to the masker. This would imply training the semantic segmentation decoder with DADA and incorporating the result of those two surrogate decoders (depth and segmentation) into the flood-mask decoder.
3. A third idea to improve the Masker is to leverage pre-trained models: MiDaS (Ranftl et al., 2020) to obtain depth maps and DeepLab v3+ (Chen et al., 2018). While we have mostly leveraged ideas from *unsupervised* domain adaptation, we believe that there is knowledge to be extracted from pre-trained models, framing our work in a "pseudo-supervised" paradigm. We mean here to use inferences from pre-trained labels as a source of *noisy* supervision since our domain (Google StreetView images) is related but different from these models' training distributions. A first idea to leverage pseudo-labels is to only use them in the beginning of the training to drive the model in a somewhat correct initial direction, but stopping the supervision at some point since pseudo-labels are too noisy to be mimicked exactly.

5.1.2 End-to-End Training

So far we have presented the Masker and the Painter as different entities, two independent models which do not share any weights or data. While it makes sense that the Painter should not depend on the Masker (as the input of the masker is a non-flooded image while the painter's input is a flooded image and its mask), the masker should predict flood masks which allow the painter to represent accurately a flood. To that end, our current idea is to back-propagate the gradient from the painter's discriminator to the masker: the painter is trained to create realistic water given a mask and an image so a natural idea is to paint an image given a predicted mask instead of a regular ground-truth mask. Because both the painter and the masker need to give sensible results before this kind of loss makes sense, we expect this new loss to only come into play after a certain number of epochs.

In other words, let $\boldsymbol{x}_r$ be a real image of a non-flooded scene, M be the masker, P the painter, $\boldsymbol{z}$ the noise used to sample from P and D_P the painter's discriminator. For this particular update, we should freeze the weights of P and D_P (just like one freezes the discriminator when training the generator in a regular GAN model)[1]:

$$\mathcal{L}_{Makser} \mathrel{+}= \lambda_{Painter}\, \mathbb{E}_{\boldsymbol{x}_r, \boldsymbol{z}} \Big[- D_P\Big(P(\boldsymbol{z}, M(\boldsymbol{x}_r) \odot \boldsymbol{x}_r)\Big)\Big]. \tag{5.1}$$

If M is implemented as described in the previous section with a segmentation and a depth prediction head, then we can train this entire model, which we call OmniGAN, in an end-to-end fashion, from the Painter to the depth, segmentation and mask decoders. Furthermore, in a single forward pass through this model we could also obtain a depth map and the sky segmentation required to illustrate the consequences of wildfires and smog, in addition to the visualization of a flood. The overall target procedure is described in Figure 5.1.

5.2 Implementation

While we aim at integrating the features of the depth prediction into the segmentation and flood-mask decoders and possibly the segmentation head's features into the flood-mask decoder too, we have not yet implemented a functioning model. The current state of our work is a lighter version of the OmniGAN idea, where there is simply no feature fusion or interaction at all between the heads (*a.k.a* decoders) beyond their shared encoder network. This means the model we will describe in this section is the same as the one illustrated in Figure 5.1 except for the feature fusion pointers (arrows with a square head).

5.2.1 Depth estimation

To train the depth decoder, we will use ground-truth data from the simulated world and explore whether MiDaS pseudo-labels could help. In addition to the original masker's loss we will therefore compute a depth decoder-specific loss which updates the weights of the decoder D_{epth} and the encoder's E, hopefully encoding geometrical information in the shared representations. Following DADA, we propose to use the Reverse Huber Loss ($c = 1/5$) with $\boldsymbol{x}_i$ being an input image and $\boldsymbol{d}_i$ being a

[1] $+=$ with respect to Equation 2.6

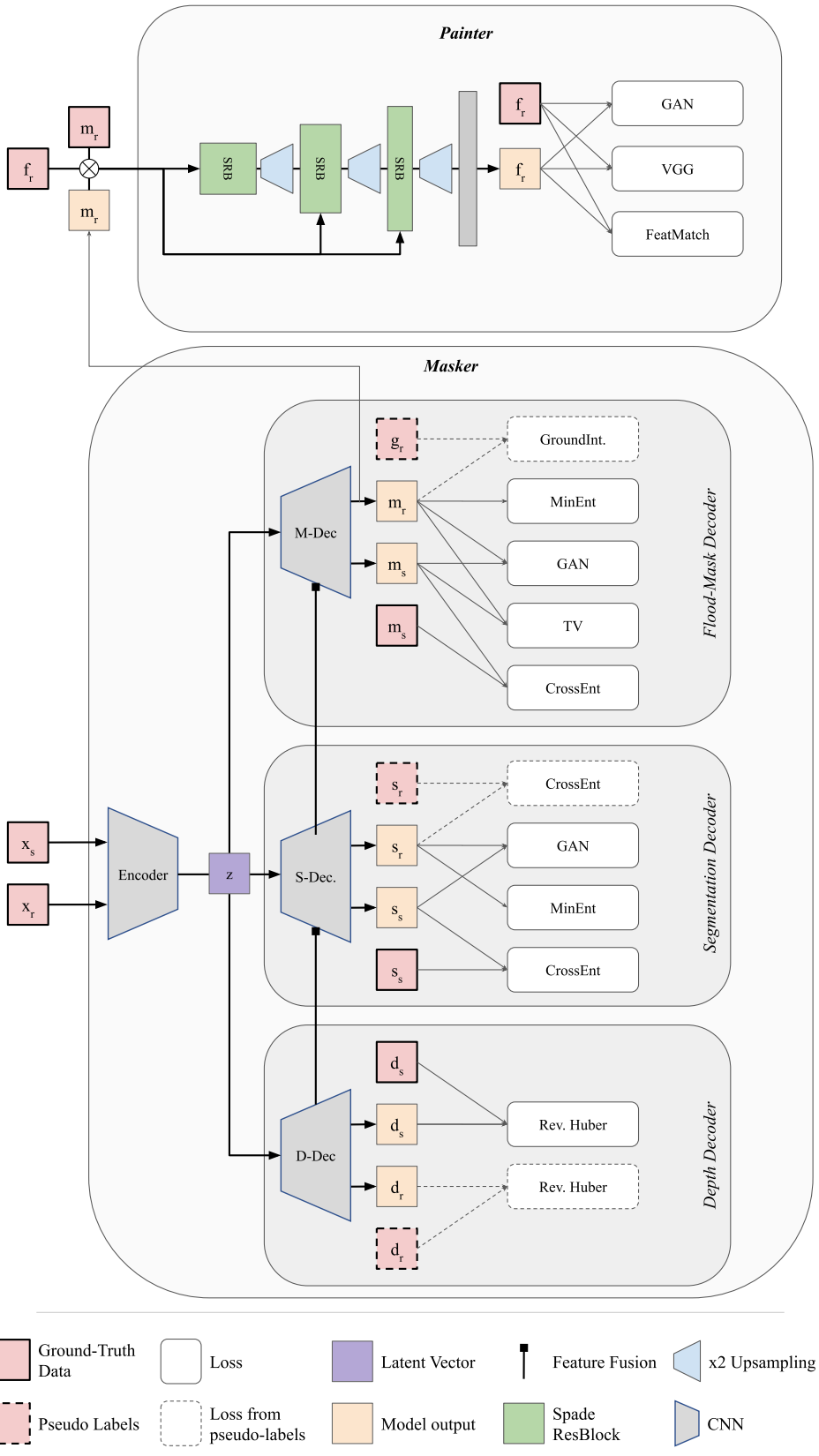


Figure 5.1: OmniGAN. $\boldsymbol{x}_r$ and $\boldsymbol{x}_s$ are real and simulated input non-flooded images, while $\boldsymbol{f}_r$ is a real input image of a flooded scene. We mark signal coming from pre-trained models in dashed lines and borders. The Painter is trained as described in Section 2.2.3. For the Masker, after an image is encoded using DeepLabV3+'s MobileNet backbone, it is first decoded into a depth map. The features of this predicted depth map are used to perform feature fusion into the Segmentation and Mask decoders as per the DADA procedure. We include a weak signal from pseudo labels in the three decoders (for the mask decoder, $\boldsymbol{g}_r$ is the union of segmentation labels which can be interpreted as "ground".)

ground-truth or pseudo depth map depending on domain i:

$$\begin{aligned}
\text{berHu}(e_z) &= \begin{cases} |e_z| & \text{if } |e_z| < c \\ \frac{e_z^2+c^2}{2c} & \text{otherwise} \end{cases} \\
\mathcal{L}_{berHu}(i)(\boldsymbol{x}_i, \boldsymbol{d}_i) &= \frac{1}{NHW} \sum_{n,h,w} \text{berHu}(D_e(E(\boldsymbol{x}_i^{(n,h,w)})) - \boldsymbol{d}_i^{(n,h,w)}) \\
\mathcal{L}_{masker} &\mathrel{+}= \lambda_{depth}\, \mathcal{L}_{berHu}(s) + \lambda_{depth}^{pseudo}\, \mathcal{L}_{berHu}(r).
\end{aligned} \tag{5.2}$$

5.2.2 Semantic Segmentation

Similarly to the depth decoder, we introduce a new segmentation decoder network S which takes the encoder's output as input and produces a segmentation map. We will implement the sequence $S \circ E$ to match Deeplabv3+'s architecture and train it with the AdvEnt procedure described previously (as it was originally designed for unsupervised domain adaptation for semantic segmentation) (see Equation 2.6 for reference):

$$\begin{aligned}
\mathcal{L}_{seg} &= \mathcal{L}_{GAN} + \lambda_{CE}^{seg}\, \mathcal{L}_{CE} + \lambda_{MinEnt}^{seg}\, \mathcal{L}_{ME} + \lambda_{TV}^{seg}\, \mathcal{L}_{TV} \\
\mathcal{L}_{masker} &\mathrel{+}= \lambda_{seg}\, \mathcal{L}_{seg} + \lambda_{seg}^{pseudo}\, \mathcal{L}_{CE}\,.
\end{aligned} \tag{5.3}$$

5.2.3 Pre-training on Virtual Kitti 2

Virtual Kitti 2 (Cabon et al., 2020) is a self-driving car dataset comprised, amongst other things, of RGB inputs along ground truth segmentation and depth maps. Because it consists of around 30 000 additional images compared to our simulated world's 10 000, we believe it could greatly improve convergence and performance of the Segmentation and Depth decoders in OmniGAN. The dataset (not surprisingly) does not however include flooded scenes and we cannot therefore use it along the flood-mask predictor. This is why we envision a pre-training use-case described previously.

5.2.4 Challenges

Training Dynamics One of the greatest challenges we face is to appropriately scale the numerous losses which train independent decoders and the shared encoder. These losses have very different "natural" ranges[2] and dynamics, which makes it very hard to get good convergence. This also makes the Full Masker very sensitive to those hyper-parameters. In addition to the losses' dynamics, the various heads and encoder could also benefit from individual learning rates to better match the specifics of each individual task.

Model Size Adding decoding heads not only introduces noticeable instability, it also makes the whole procedure more computationally expensive. In Mila's RTX8000 GPUs with 48GB memory, we can at most fit a batch size of 3 when training a Full Masker and a single step takes around 5s, making any training loop very long and early-stopping difficult as we have to wait for at least 12 or 24h to have an idea of the model's performance. Furthermore, it is difficult to isolate individual heads' contributions and compare models based on their validation performance metrics. As emphasized before, because performance on the simulated world is not a perfect predictor for performance in the real world, we still have to *look* at masker inferences to choose models and hyper parameters.

[2] the TV loss is often around 10^{-3} while the berHu loss is around 10^{1}

Production considerations In the end, the goal of this research project is to deploy the models in production to a user-facing web interface. This brings many efficiency considerations into our challenges and we will also be looking at Mixed-Precision training, distillation and pruning to optimize the number of images per second the model is able to process.

5.3 Measuring the impact of personalized images

In Chapter 2 we presented our work as being aimed at the general public to increase awareness about climate change's consequences and empathy towards present and future victims. While this is grounded in behavioral psychology and climate communications research, our modeling work can actually *contribute* to these research areas. This would not only be the opportunity to validate our premises and methods, but also to advance the scientific understanding of human perception and psychology in the context of environmental awareness. We are currently working with a professor in political sciences at Université de Montréal to perform a series of studies on risk perception related to climate change events: using our tool to personalize consequences of climate-related extreme events is a unique opportunity to measure and verify a series of hypothesis.

While this work does not, in itself, contribute to research in Artificial Intelligence, it is a great opportunity to create bridges between our community and the social sciences, making personalized systems at the service of human-centered research. The following paragraphs are adapted extracts from work in progress towards this objective.

Study Citizens tend to perceive climate change as a distant threat (or high-level construal[3]), which influences the degree to which they are willing to act in order to reduce the potential runaways of the environmental crisis (Leiserowitz, 2006; O'Neill and Nicholson-Cole, 2009; Trope and Liberman, 2010). In order to increase engagement and support toward policies, scholars have tried to increase the salience of climate change by proximizing this issue (as a low - concrete - construal). This proximization, however, has rarely been able to reduce this psychological distance because the treatments are unrealistic (or not realistic enough) and do not represent an object for which respondents *really* care. These results and measurements make the identification of an underlying mechanism causing the distance particularly difficult because Construal Level Theory (CLT) (Trope and Liberman, 2010) is, by itself, less likely to change opinions and engagement. Proximization on the other hand will alter information processing and decision-making and increase, by making climate change concrete, the probability to engage and adopt pro-environmental habits.

To ensure that participants care about the treatments, we provide *personalized* images of potential climate-induced floods on (1) a typical house, (2) a district landmark, and (3) a provincial landmark. No other studies, to our knowledge, have done such work. Generative Adversarial Networks allow us to flood images from Google Map in every electoral district. Regarding these images, we hypothesize that:

- H_1: Images that are more proximate to people will have stronger effects than distant ones.
- $H_{1.a}$: Typical house > District and Provincial landmark
- $H_{1.b}$: District > Provincial landmark
- $H_{1.c}$: National > International (Flood in Canada ¿ Flood in Bangladesh for Canadians).

We hypothesize that three mechanisms will moderate and enhance engagement and opinion when climate change will be proximate (Brügger et al., 2015; Nilsson et al., 2004; Smith and Leiserowitz, 2014). Beliefs, values and emotions about climate change will moderate the effects of the treatments:

[3]In social psychology, construals are how individuals perceive, comprehend, and interpret the world around them, particularly the behavior or action of others towards themselves.

$H_{2.a}$: Ideology hypothesis (is the right side of the political spectrum more permeable?)

$H_{2.b}$: Worry Hypothesis (does more worry about a subject yield more engagement?)

$H_{2.c}$: Emotions hypothesis - (what are the emotions triggered by the treatments? *e.g.* skepticism, fear, hope ...)

Experimental design Our design is conceived to test the causal effect of climate change flood induced images on the support of environmental policy, perceptions, and engagement toward climate change. We will run an online survey experiment on a representative sample of the Canadian population. After asking for their consent and their postal code, we will measure participant's socio-economic status, values, partisan identification, ideology, and their general attitudes toward climate change (causes, optimism, harm, worry). Every participant will be matched to their electoral district and province, with the postal code they provided, to assign a personalized treatment.

Participants will then be randomly assigned to one of five groups. Firstly, a control group will not receive any information. The first treatment group (distant) will see a visual depiction of a flood in Bangladesh.[4] The second treatment (personalized - typical household) will depict a flooded typical house from their electoral district. A third group (proximal place of interest) will receive an image of a close flooded public place in their electoral district. The last treatment (proximal place of interest - province) will show the respondent an image of a flooded known provincial landmark.

Following the treatment, we will measure three dependent variables. First, the willingness to pay higher taxes (on buildings or oil) in order to prevent floods. Second, the behavioural intentions to address climate change (personal measures to reduce my carbon footprint, to protect my home from flooding, and to convince the government to address climate change). Additionally, we will ask the respondent if they want more information on how to reduce their carbon emission and how to protect their home from floods. If they do, we will send them an e-mail with the information. Thirdly, we will estimate the engagement toward fighting climate change by asking them if they want to send a letter to their Member of Parliament to urge more government action on climate change. If respondents click "yes," we will provide them a prepared letter they could sign in their name. Finally, we will measure three moderators: beliefs (ideology and climate change), emotions (Anxiety-related, Shame-focused, Sadness-related, Hope-related), and values.

[4]This image will depict the "best" method in climate change communication according to agencies (Climate Visuals).

Chapter 6

Conclusion

Climate Change may very well be one of the greatest challenges to modern science and society. And the Artificial Intelligence community can do something about it, be it through contributing to advancing climate science, create cleaner technologies or help societies adapt and mitigate its consequences. In this ongoing work, we show how state-of-the-art representation learning and generative modeling can be put to use to improve the public's understanding of the challenges and consequences of Climate Change as well as to contribute to the modeling of cloud reflectance fields used in climate simulations. We also present how those considerations can be built into practical tools to both reflect critically on the AI community's own carbon impact and empower the research community with a new way to measure human risk perception, in the context of addressing climate change.

The main focus of our research is to create visualizations of the potential impacts of climate change from Google StreetView images. In order to generate images of floods, we successfully combine on the one hand domain adaptation techniques to adversarially train a binary mask predictor and on the other hand a conditional image generation process to paint realistic water within the mask. We show that in most scenes of reasonable complexity, the masker is able to produce realistic flood masks, yet struggles with images containing many objects or out-of-distribution angles and contents. We also demonstrate how our adaptation of the GauGAN procedure produces contextualized water textures and reflections. We also discover that the model is able to in-paint small missing details to compensate for a mask's erratic edges. Additionally, leveraging pre-trained depth estimation and semantic segmentation models, we produce images of smog and wildfire. All in all, our approach allows for the generation of personalized and vivid imagery of the potential impacts climate change could have on our daily lives.

In future work, we aim at unifying methods developed to create individual climate event visualizations into a single model, OmniGAN. We believe that maximizing the interactions between different kinds of data and tasks will not only yield a more robust but also more compute-efficient model to deploy in the real world. To do so, we will add two decoders to the mask predictor making information about geometry (via depth prediction) and semantics (via segmentation) available to it through the shared encoder. As a byproduct, we will be able to obtain depth and segmentation maps to create images of smog and wildfire at minimal additional cost compared to full forward passes in pre-trained models. Furthermore, there is a great opportunity to leverage this work in order to contribute to behavioural psychology research, by customizing treatments and therefore better understand how to efficiently communicate the risks related to climate change to the public.

Subsequently, we present work dedicated to quantifying the carbon emissions of our community. We provide actionable insights into the sources of carbon emissions and how to decrease them. Our main take-away is that location of compute matters a lot, with regional electricity grids' carbon intensities varying with a factor up to almost 40, from 20g CO_2eq/kWh in Quebec, Canada to 736.6g CO_2eq/kWh in Iowa, USA. A simple lever available to AI practitioners using cloud providers is therefore to choose (when possible) the location of their compute according to its carbon intensity. We also introduce a new tool to track emissions across experiments and teams to encourage transparency and systematic tracking: CodeCarbon. Future work on this topic will be focused on developing CodeCarbon's features. We hope that this can eventually lead to more self-regulation and provide policy makers with more accurate information.

Finally, we introduce preliminary work to include GANs in the modeling of clouds' reflectance fields. Formulating the problem of predicting reflectance fields given physical measurements in a conditional

image generation framework, we present a proof of concept able to generate visually compelling samples from a relatively small number of training pairs. Bringing insights from the computer vision community into climate sciences allows us to conceive a data-driven approach to address a physical problem. Similarly, we envision bringing more physical insights into the training procedure could help both better inform the predictions and better measure performance.

Bibliography

Alami Mejjati, Y., Richardt, C., Tompkin, J., Cosker, D., and Kim, K. I. (2018). Unsupervised Attention-guided Image-to-Image Translation. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, Advances in Neural Information Processing Systems 31, pages 3693–3703. Curran Associates, Inc.

Amazon Web Services (2019). AWS & Sustainability.

Arakawa, A. (2004). The cumulus parameterization problem: Past, present, and future. Journal of Climate, 17(13):2493–2525.

Arjovsky, M. and Bottou, L. (2017). Towards Principled Methods for Training Generative Adversarial Networks. arXiv:1701.04862 [cs, stat]. arXiv: 1701.04862.

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein GAN. arXiv:1701.07875 [cs, stat]. arXiv: 1701.07875.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer Normalization. arXiv:1607.06450 [cs, stat]. arXiv: 1607.06450.

Berger, A., Susskind, J., and Gribkoff, E. (2020). MIT Climate Explainers: Wildfires.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. Journal of Machine Learning Research, 13(Feb):281–305.

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

Borji, A. (2019). Pros and cons of gan evaluation measures. Computer Vision and Image Understanding, 179:41–65. Publisher: Elsevier.

Bousmalis, K., Silberman, N., Dohan, D., Erhan, D., and Krishnan, D. (2016). Unsupervised Pixel-Level Domain Adaptation with Generative Adversarial Networks. arXiv:1612.05424 [cs]. arXiv: 1612.05424.

Brander, M., Sood, A., Wylie, C., Haughton, A., and Lovell, J. (2011). Electricity-specific emission factors for grid electricity. Ecometrica, Emissionfactors. com.

Brenowitz, N. D. and Bretherton, C. S. (2018). Prognostic validation of a neural network unified physics parameterization. Geophysical Research Letters, 45(12):6289–6298. Publisher: Wiley Online Library.

Bretherton, C. S. (2015). Insights into low-latitude cloud feedbacks from high-resolution models. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 373(2054):20140415. Publisher: The Royal Society Publishing.

Brügger, A., Dessai, S., Devine-Wright, P., Morton, T. A., and Pidgeon, N. F. (2015). Psychological responses to the proximity of climate change. Nature Climate Change, 5(12):1031–1037.

Cabon, Y., Murray, N., and Humenberger, M. (2020). Virtual KITTI 2. arXiv:2001.10773 [cs, eess]. arXiv: 2001.10773.

Cai, E., Juan, D.-C., Stamoulis, D., and Marculescu, D. (2017). Neuralpower: Predict and deploy energy-efficient convolutional neural networks. arXiv preprint arXiv:1710.05420.

Chapman, D. A., Corner, A., Webster, R., and Markowitz, E. M. (2016). Climate visuals: A mixed methods investigation of public perceptions of climate images in three countries. Global Environmental Change, 41:172 – 182.

Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F., and Adam, H. (2018). Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. arXiv:1802.02611 [cs]. arXiv: 1802.02611.

Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., and Abbeel, P. (2016). InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. arXiv:1606.03657 [cs, stat]. arXiv: 1606.03657.

Climate Central (2019). Climate Change Is Threatening Air Quality Across The Country.

Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., and Schiele, B. (2016). The Cityscapes Dataset for Semantic Urban Scene Understanding. In Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Corti, T., Luo, B., Peter, T., Vömel, H., and Fu, Q. (2005). Mean radiative energy balance and vertical mass fluxes in the equatorial upper troposphere and lower stratosphere. Geophysical research letters, 32(6). Publisher: Wiley Online Library.

Cosne, G., Juraver, A., Teng, M., Schmidt, V., Vardanyan, V., Luccioni, A., and Bengio, Y. (2020). Using Simulated Data to Generate Images of Climate Change. arXiv:2001.09531 [cs, eess]. arXiv: 2001.09531.

de Vries, H., Strub, F., Mary, J., Larochelle, H., Pietquin, O., and Courville, A. (2017). Modulating early visual processing by language. arXiv:1707.00683 [cs]. arXiv: 1707.00683 version: 3.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs]. arXiv: 1810.04805.

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. (2013). DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition. arXiv:1310.1531 [cs]. arXiv: 1310.1531.

Dottori, F., Salamon, P., Bianchi, A., Alfieri, L., Hirpa, F. A., and Feyen, L. (2016). Development and evaluation of a framework for global flood hazard mapping. Advances in Water Resources, 94:87 – 102.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research, 12:2121–2150.

Dumoulin, V. and Visin, F. (2016). A guide to convolution arithmetic for deep learning. arXiv:1603.07285 [cs, stat]. arXiv: 1603.07285.

Eggleston, S., Buendia, L., Miwa, K., Ngara, T., and Tanabe, K. (2006). 2006 IPCC guidelines for national greenhouse gas inventories, volume 5. Institute for Global Environmental Strategies Hayama, Japan.

Endres, D. and Schindelin, J. (2003). A new metric for probability distributions. IEEE Transactions on Information Theory, 49(7):1858–1860.

Falkner, S., Klein, A., and Hutter, F. (2018). BOHB: Robust and efficient hyperparameter optimization at scale. arXiv preprint arXiv:1807.01774.

Feurer, M. and Hutter, F. (2019). Hyperparameter optimization. In Automated Machine Learning, pages 3–33. Springer.

Ganin, Y. and Lempitsky, V. (2015). Unsupervised Domain Adaptation by Backpropagation. ICML, page 10.

Gao, J. (2014). Machine Learning Applications for Data Center Optimization. Google Research.

Gelaro, R., McCarty, W., Suárez, M. J., Todling, R., Molod, A., Takacs, L., Randles, C. A., Darmenov, A., Bosilovich, M. G., Reichle, R., and others (2017). The modern-era retrospective analysis for research and applications, version 2 (MERRA-2). Journal of Climate, 30(14):5419–5454.

Gidel, G., Berard, H., Vignoud, G., Vincent, P., and Lacoste-Julien, S. (2018). A Variational Inequality Perspective on Generative Adversarial Networks. arXiv:1802.10551 [cs, math, stat]. arXiv: 1802.10551.

Ginsburg, B., Castonguay, P., Hrinchuk, O., Kuchaiev, O., Lavrukhin, V., Leary, R., Li, J., Nguyen, H., Zhang, Y., and Cohen, J. M. (2019). Stochastic Gradient Methods with Layer-wise Adaptive Moments for Training of Deep Networks. arXiv:1905.11286 [cs, stat]. arXiv: 1905.11286.

Goldstein, D. G., Hershfield, H. E., and Benartzi, S. (2016). The Illusion of Wealth and Its Reversal. Journal of Marketing Research, 53(5):804–813. _eprint: https://doi.org/10.1509/jmr.14.0652.

Gonzalez, C. and Woods, E. (1991). Digital Image Processing, Addison-Wesley Publishing Company. Pearson.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep Learning. MIT Press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, Advances in Neural Information Processing Systems 27, pages 2672–2680. Curran Associates, Inc.

Google (2018). Google Environmental Report 2018.

Gopalan, R., Ruonan Li, and Chellappa, R. (2011). Domain adaptation for object recognition: An unsupervised approach. In 2011 International Conference on Computer Vision, pages 999–1006, Barcelona, Spain. IEEE.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. (2017). Improved Training of Wasserstein GANs. arXiv:1704.00028 [cs, stat]. arXiv: 1704.00028.

Harju, A., Siro, T., Canova, F. F., Hakala, S., and Rantalaiho, T. (2012). Computational physics on graphics processing units. In Proceedings of the 11th international conference on Applied Parallel and Scientific Computing, pages 3–26. Springer-Verlag.

Hartmann, D. L., Ockert-Bell, M. E., and Michelsen, M. L. (1992). The effect of cloud type on Earth's energy balance: Global analysis. Journal of Climate, 5(11):1281–1304.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs]. arXiv: 1512.03385.

Hershfield, H. E., Goldstein, D. G., Sharpe, W. F., Fox, J., Yeykelis, L., Carstensen, L. L., and Bailenson, J. N. (2011). Increasing Saving Behavior Through Age-progressed Renderings Of The Future Self. JMR, Journal of marketing research, 48:S23–S37.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. arXiv:1207.0580 [cs]. arXiv: 1207.0580.

Hoffman, J., Tzeng, E., Park, T., Zhu, J.-Y., Isola, P., Saenko, K., Efros, A. A., and Darrell, T. (2017). CyCADA: Cycle-Consistent Adversarial Domain Adaptation. arXiv:1711.03213 [cs]. arXiv: 1711.03213.

Howard, J. and Ruder, S. (2018). Universal Language Model Fine-tuning for Text Classification. arXiv:1801.06146 [cs, stat]. arXiv: 1801.06146.

Huang, X. and Belongie, S. (2017). Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization. arXiv:1703.06868 [cs]. arXiv: 1703.06868.

Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv:1502.03167 [cs]. arXiv: 1502.03167.

Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1125–1134.

Johnson, J., Alahi, A., and Fei-Fei, L. (2016). Perceptual Losses for Real-Time Style Transfer and Super-Resolution. arXiv:1603.08155 [cs]. arXiv: 1603.08155.

Kingma, D. P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs]. arXiv: 1412.6980.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, Advances in Neural Information Processing Systems 25, pages 1097–1105. Curran Associates, Inc.

Lacoste, A., Luccioni, A., Schmidt, V., and Dandres, T. (2019). Quantifying the Carbon Emissions of Machine Learning. arXiv:1910.09700 [cs]. arXiv: 1910.09700.

LeCun, Y. (1989). Generalization and network design strategies. Connectionism in perspective.

LeCun, Y. (1998). The MNIST database of handwritten digits. http://yann. lecun. com/exdb/mnist/.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324.

Ledig, C., Theis, L., Huszar, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., and Shi, W. (2016). Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. arXiv:1609.04802 [cs, stat]. arXiv: 1609.04802 version: 5.

Leiserowitz, A. (2006). Climate Change Risk Perception and Policy Preferences: the Role of affect, Imagery, and Values. Climatic Change, 77:45–72.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2016). Hyperband: A novel bandit-based approach to hyperparameter optimization. arXiv preprint arXiv:1603.06560.

Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Hardt, M., Recht, B., and Talwalkar, A. (2018). Massively parallel hyperparameter tuning. arXiv preprint arXiv:1810.05934.

Li, Y. (2017). Deep Reinforcement Learning: An Overview. arXiv:1701.07274 [cs]. arXiv: 1701.07274.

Lim, J. H. and Ye, J. C. (2017). Geometric gan. arXiv preprint arXiv:1705.02894.

Liu, L., Jiang, H., He, P., Chen, W., Liu, X., Gao, J., and Han, J. (2019). On the Variance of the Adaptive Learning Rate and Beyond. arXiv:1908.03265 [cs, stat]. arXiv: 1908.03265.

Mao, X., Li, Q., Xie, H., Lau, R. Y., Wang, Z., and Paul Smolley, S. (2017). Least squares generative adversarial networks. In Proceedings of the IEEE International Conference on Computer Vision, pages 2794–2802.

Metz, L., Poole, B., Pfau, D., and Sohl-Dickstein, J. (2017). Unrolled Generative Adversarial Networks. arXiv:1611.02163 [cs, stat]. arXiv: 1611.02163.

Microsoft (2018). Beyond Carbon Neutral. White paper.

Mitchell, T. M. (1997). Machine Learning, volume 1 of 1. McGraw-Hill Science/Engineering/-Math.

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. (2018). Spectral Normalization for Generative Adversarial Networks. arXiv:1802.05957 [cs, stat]. arXiv: 1802.05957.

Mo, S., Cho, M., and Shin, J. (2018). InstaGAN: Instance-aware Image-to-Image Translation. arXiv:1812.10889 [cs, stat]. arXiv: 1812.10889.

Nesterov, Y. (1983). A method for unconstrained convex minimization problem with the rate of convergence $O(1/kˆ2)$. Doklady Akademii Nauk SSSR, 269:543–547.

Nilsson, A., Borgstede, C. v., and Biel, A. (2004). Willingness to accept climate change strategies: The effect of values and norms. Journal of Environmental Psychology, 24(3):267–277.

North, G. R., Cahalan, R. F., and Coakley Jr, J. A. (1981). Energy balance climate models. Reviews of Geophysics, 19(1):91–121. Publisher: Wiley Online Library.

Odena, A., Dumoulin, V., and Olah, C. (2016). Deconvolution and Checkerboard Artifacts. Distill, 1(10):e3.

Odena, A., Olah, C., and Shlens, J. (2017). Conditional Image Synthesis With Auxiliary Classifier GANs. arXiv:1610.09585 [cs, stat]. arXiv: 1610.09585.

O'Gorman, P. A. and Dwyer, J. G. (2018). Using machine learning to parameterize moist convection: Potential for modeling of climate, climate change, and extreme events. Journal of Advances in Modeling Earth Systems, 10(10):2548–2563. Publisher: Wiley Online Library.

O'Neill, S. and Nicholson-Cole, S. (2009). "Fear Won't Do It": Promoting Positive Engagement With Climate Change Through Visual and Iconic Representations. Science Communication, 30(3):355–379.

Oord, A. v. d., Kalchbrenner, N., Vinyals, O., Espeholt, L., Graves, A., and Kavukcuoglu, K. (2016). Conditional Image Generation with PixelCNN Decoders. arXiv:1606.05328 [cs]. arXiv: 1606.05328.

Pan, S. J. and Yang, Q. (2010). A Survey on Transfer Learning. IEEE Transactions on Knowledge and Data Engineering, 22(10):1345–1359.

Park, T., Liu, M.-Y., Wang, T.-C., and Zhu, J.-Y. (2019). Semantic Image Synthesis with Spatially-Adaptive Normalization. arXiv:1903.07291 [cs]. arXiv: 1903.07291.

Perez, E., Strub, F., de Vries, H., Dumoulin, V., and Courville, A. (2017). FiLM: Visual Reasoning with a General Conditioning Layer. arXiv:1709.07871 [cs, stat]. arXiv: 1709.07871.

Petković, M. D. and Stanimirović, P. S. (2009). Generalized matrix inversion is not harder than matrix multiplication. Journal of Computational and Applied Mathematics, 230(1):270–282.

Platnick, S., Meyer, K. G., King, M. D., Wind, G., Amarasinghe, N., Marchant, B., Arnold, G. T., Zhang, Z., Hubanks, P. A., Holz, R. E., and others (2016). The MODIS cloud optical and microphysical products: Collection 6 updates and examples from Terra and Aqua. IEEE Transactions on Geoscience and Remote Sensing, 55(1):502–525. Publisher: IEEE.

Radford, A., Metz, L., and Chintala, S. (2016). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. arXiv:1511.06434 [cs]. arXiv: 1511.06434.

Ramanathan, V., Cess, R., Harrison, E., Minnis, P., Barkstrom, B., Ahmad, E., and Hartmann, D. (1989). Cloud-radiative forcing and climate: Results from the Earth Radiation Budget Experiment. Science, 243(4887):57–63. Publisher: American Association for the Advancement of Science.

Ranftl, R., Lasinger, K., Hafner, D., Schindler, K., and Koltun, V. (2020). Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer. arXiv:1907.01341 [cs]. arXiv: 1907.01341 version: 3.

Rasp, S., Pritchard, M. S., and Gentine, P. (2018). Deep learning to represent subgrid processes in climate models. Proceedings of the National Academy of Sciences, 115(39):9684–9689. Publisher: National Acad Sciences.

Rolnick, D., Donti, P. L., Kaack, L. H., Kochanski, K., Lacoste, A., Sankaran, K., Ross, A. S., Milojevic-Dupont, N., Jaques, N., Waldman-Brown, A., Luccioni, A., Maharaj, T., Sherwin, E. D., Mukkavilli, S. K., Kording, K. P., Gomes, C., Ng, A. Y., Hassabis, D., Platt, J. C., Creutzig, F., Chayes, J., and Bengio, Y. (2019). Tackling Climate Change with Machine Learning. arXiv:1906.05433 [cs, stat]. arXiv: 1906.05433.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In International Conference on Medical image computing and computer-assisted intervention, pages 234–241. Springer.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A., and Fei-Fei, L. (2014). ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision, 115.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved techniques for training gans. In Advances in neural information processing systems, pages 2234–2242.

Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. (2019). How Does Batch Normalization Help Optimization? arXiv:1805.11604 [cs, stat]. arXiv: 1805.11604.

Schmidt, V., Alghali, M., Sankaran, K., Yuan, T., and Bengio, Y. (2020). Modeling Cloud Reflectance Fields using Conditional Generative Adversarial Networks. arXiv:2002.07579 [physics]. arXiv: 2002.07579.

Schmidt, V., Luccioni, A., Mukkavilli, S. K., Balasooriya, N., Sankaran, K., Chayes, J., and Bengio, Y. (2019). Visualizing the Consequences of Climate Change Using Cycle-Consistent Adversarial Networks. arXiv:1905.03709 [cs]. arXiv: 1905.03709.

Schneider, T., Teixeira, J., Bretherton, C. S., Brient, F., Pressel, K. G., Schär, C., and Siebesma, A. P. (2017). Climate goals and computing the future of clouds. Nature Climate Change, 7(1):3–5. Publisher: Nature Publishing Group.

Schwartz, R., Dodge, J., Smith, N. A., and Etzioni, O. (2019). Green AI. arXiv preprint arXiv:1907.10597.

Shannon, C. E. (1948). A Mathematical Theory of Communication. The Bell System Technical Journa, 27:379–423.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

Smith, N. and Leiserowitz, A. (2014). The Role of Emotion in Global Warming Policy Support and Opposition. Risk Analysis, 34(5):937–948.

Stocker, T. F., Qin, D., Plattner, G.-K., Tignor, M., Allen, S. K., Boschung, J., Nauels, A., Xia, Y., Bex, V., Midgley, P. M., and others (2013). Climate change 2013: The physical science basis. Contribution of working group I to the fifth assessment report of the intergovernmental panel on climate change, 1535. Publisher: Cambridge university press CambridgeNew York.

Strubell, E., Ganesh, A., and McCallum, A. (2019). Energy and Policy Considerations for Deep Learning in NLP. arXiv preprint arXiv:1906.02243.

Sun, S., Cao, Z., Zhu, H., and Zhao, J. (2019). A Survey of Optimization Methods from a Machine Learning Perspective. arXiv:1906.06821 [cs, math, stat]. arXiv: 1906.06821.

Tajbakhsh, N., Shin, J. Y., Gurudu, S. R., Hurst, R. T., Kendall, C. B., Gotway, M. B., and Liang, J. (2016). Convolutional neural networks for medical image analysis: Full training or fine tuning? IEEE transactions on medical imaging, 35(5):1299–1312. Publisher: IEEE.

Teich, P. (2018). Tearing Apart Google's TPU 3.0 AI Coprocessor.

To, W. and Lee, P. K. (2017). GHG emissions from electricity consumption: A case study of Hong Kong from 2002 to 2015 and trends to 2030. Journal of cleaner production, 165:589–598. Publisher: Elsevier.

Tribus, M. (1961). Thermostatics and thermodynamics. Van Nostrand Reinhold.

Trope, Y. and Liberman, N. (2010). Construal-Level Theory of Psychological Distance. Psychological review, 117(2):440–463.

Tsoumakas, G. (2019). A survey of machine learning techniques for food sales prediction. Artificial Intelligence Review, 52(1):441–447. Publisher: Springer.

Ulyanov, D., Vedaldi, A., and Lempitsky, V. (2017). Instance Normalization: The Missing Ingredient for Fast Stylization. arXiv:1607.08022 [cs]. arXiv: 1607.08022.

Varma, S. and Simon, R. (2006). Bias in error estimation when using cross-validation for model selection. BMC bioinformatics, 7(1):91. Publisher: Springer.

Villani, C. (2008). Optimal transport: old and new, volume 338. Springer Science & Business Media.

Vousdoukas, M. I., Mentaschi, L., Voukouvalas, E., Verlaan, M., Jevrejeva, S., Jackson, L. P., and Feyen, L. (2018). Global probabilistic projections of extreme sea levels show intensification of coastal flood hazard. Nature Communications, 9(1):2360.

Vu, T.-H., Jain, H., Bucher, M., Cord, M., and Pérez, P. (2018). ADVENT: Adversarial Entropy Minimization for Domain Adaptation in Semantic Segmentation. arXiv:1811.12833 [cs]. arXiv: 1811.12833.

Vu, T.-H., Jain, H., Bucher, M., Cord, M., and Pérez, P. (2019). DADA: Depth-aware Domain Adaptation in Semantic Segmentation. arXiv:1904.01886 [cs]. arXiv: 1904.01886.

Wang, J., Sun, K., Cheng, T., Jiang, B., Deng, C., Zhao, Y., Liu, D., Mu, Y., Tan, M., Wang, X., Liu, W., and Xiao, B. (2019). Deep High-Resolution Representation Learning for Visual Recognition. arXiv:1908.07919 [cs]. arXiv: 1908.07919 version: 2.

Wang, S., Corner, A., Chapman, D., and Markowitz, E. (2018a). Public engagement with climate imagery in a changing digital landscape. WIREs Climate Change, 9(2):e509. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/wcc.509.

Wang, T.-C., Liu, M.-Y., Zhu, J.-Y., Tao, A., Kautz, J., and Catanzaro, B. (2018b). High-resolution image synthesis and semantic manipulation with conditional gans. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 8798–8807.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K. Q., editors, Advances in Neural Information Processing Systems, volume 27, pages 3320–3328. Curran Associates, Inc.

Yuan, T., Song, H., Hall, D., Schmidt, V., Sankaran, K., and Bengio, Y. (2019). Artificial intelligence based cloud distributor (AI-CD): probing clouds with generative adversarial networks. AGU Fall Meeting 2019.

Zhang, Y., Ding, L., and Sharma, G. (2017). HazeRD: An outdoor scene dataset and benchmark for single image dehazing. In 2017 IEEE International Conference on Image Processing (ICIP), pages 3205–3209, Beijing. IEEE.

Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. (2017). Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. arXiv:1703.10593 [cs]. arXiv: 1703.10593.