Université de Montréal

Predoc III

par

Vikram Voleti

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée en vue de l'obtention du grade de
Philosophiæ Doctor (Ph.D.)
en Computer Science

6 janvier 2021

Université de Montréal

Faculté des arts et des sciences

Cette thèse intitulée

Predoc III

présentée par

Vikram Voleti

a été évaluée par un jury composé des personnes suivantes :

Liam Paull
(président-rapporteur)

Christopher Pal
(directeur de recherche)

Ioannis Mitliagkas
(membre du jury)

(examinateur externe)

(représentant du doyen de la FESP)

Résumé

La génération d'images et la prédiction vidéo sont des problèmes difficiles qui ont attiré une attention accrue avec les progrès récents des réseaux de neurones profonds. Les Neural ODEs ont récemment gagné en importance pour incorporer des équations différentielles dans le pipeline d'entraînement basé sur la descente de gradient. Ils ont été utilisés efficacement comme flux de normalisation continus entre le bruit latent et les données d'image.

Dans ce travail, nous explorons la possibilité d'utiliser des Neural ODEs pour modéliser la dynamique latente en vidéo. Nous montrons les résultats sur des ensembles de données vidéo simples et explorons la possibilité de prédire les caractéristiques de haute dimension d'une vidéo, ainsi que de générer des images vidéo de retour à partir des caractéristiques prévues. Nous introduisons aussi une nouvelle méthode d'utilisation des flux de normalisation continus dans une architecture multi-échelles pour la génération d'images inconditionnelle, qui permet d'obtenir la probabilité meilleure en un minimum de temps sur CIFAR10 et ImageNet64.

Abstract

Image generation and video prediction are challenging problems that have gained increased attention with the recent advancements in deep neural networks. Neural ODEs have recently gained significance for incorporating differential equations into the pipeline of gradient descent-based training. They have been used effectively as Continuous Normalizing Flows between latent noise and image data.

In this work, we explore the possibility of using Neural ODEs to model latent dynamics in video. We show results on simple video datasets, and explore the possibility of predicting high-dimensional features of a video, as well as generating back video frames from the predicted features. We also introduce a novel method of using Continuous Normalizing Flows in a multi-scale architecture for unconditional image generation, which achieves state-of-the-art likelihood in the least time on CIFAR10 and ImageNet64.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to express my sincere gratitude to my advisor Prof. Christopher Pal for the continuous support of my PhD study and related research, for his motivation, immense knowledge, and flexibility. His guidance helped me in all the time of research and writing of this thesis.

I would also like to thank Prof. Graham Taylor for welcoming me to his lab at the University of Guelph. His constant support helped continue a sense of community and professional environment which were sorely needed especially in the times of COVID.

I would also like to thank my lab mates at Mila as well as in the MLRG group at Guelph, for the many chats, meetings, discussions, late nights, coding sessions that helped build my academic career so far. In particular, I would like to thank Samira Kahou for her valuable support throughout, Vincent Michalski (and their daughter Clara), David Kanaa, Anirudh Goyal, Florian Golemo, Krishna Murthy Jatavallabhula. I would also like to thank all my collaborators including Yoshua Bengio, Sanja Fidler, Guillaume Lajoie, Michael Mozer, Doina Precup, Florian Shkurti, Pascal Vincent.

Finally, I would like to thank my family: my parents and my wife, for supporting me in immeasurable and inconceivable ways throughout my life, and especially during my doctoral studies.

Chapter 1

Introduction

Generating realistic images and coherent video frames are among the most challenging problems for current machine learning systems. The generation of images requires an understanding of the composition of a scene in detail. A simple model that can generate small synthetic images is less likely to understand the intricacies in a scene as much as a model capable of generating large realistic images. Scaling up image generation to high resolutions is a challenging task. The underlying factors of video are even more complex, involving multiple objects in complex and interactive actions across time. These problems among others strongly motivate modeling image and video, such that the high resolution detail in images and object dynamics in video can be figured via a generative task.

Video prediction has seen a number of methods [**116, 121, 26, 2, 27, 56, 12**] that use an encoder-decoder-based architecture to condition on past frames and generate future frames. Chapter 2 describes a brief history of relevant works in this field. The encoded features are typically propagated forward in time using a recurrent neural network (RNN). This RNN models the dynamics of the objects in the scene, by using features that are disentangled into static or dynamic components, or into "content" and "pose", in an unsupervised fashion in many cases.

We introduce a new method for video prediction, where we replace the RNN with a Neural ODE [**15**] to model the object dynamics. Neural ODEs are deep parametric models that incorporate differential dynamics in the gradient-based training paradigm. They offer several advantages over RNNs in this context, including the fact that they model object

dynamics in continuous time. Chapter 3 describes the concept of Neural ODEs in detail, and explains their training formulation in the context of gradient descent.

Chapter 4 elaborates on our method that was recently published [**123**]. We use Neural ODEs to model the dynamics in simple videos. Our model is able to predict future frames of a video by decoding the predicted features from the Neural ODE.

Chapter 5 builds on Chapter 4, and expands the training paradigm to high resolution natural videos. We test whether Neural ODEs are capable of modeling the trajectory of high-dimensional latent features that capture the semantic information in a scene. We then test whether the frames of the video can be predicted by decoding the predicted high-dimensional features.

Over the years, generative modeling of images has seen some success using multi-scale architecture. Chapter 2 elaborates on previous methods of image generation, primarily those that used multi-scale architectures. Among GAN-based models, LAPGAN [**28**], ProGAN [**66**], etc. have incorporated the notion of generating an image at multiple scales with very good results. Recently, NVAE [**118**] introduced a hierarchical structure in VAEs. Normalizing flows [**29, 30, 71, 31**] operate in a multi-scale paradigm by sequentially decreasing the spatial resolution while assigning a part of the feature at each step to noise. Hence, multi-scale architectures have proven to be useful in recording and estimating hierarchical information in images.

Neural ODEs have also proven to be useful in making a likelihood-based generative model of images called Continuous Normalizing Flows [**15**], described in Section 3.2. In Chapter 6, we introduce a novel multi-scale architecture to generate images using Continuous Normalizing Flows. We use a Laplacian pyramid to map from latent noise to Laplacian images, and train to maximize the likelihood of the generated images. Our method achieves state-of-the-art likelihood in the least amount of time while training on images from CIFAR10 and ImageNet64.

Finally, Chapter 7 describes a nascent idea on the disentanglement of features in a deep neural network. This idea has the potential to be an easy-to-use plug-in to the existing frameworks of training of deep neural networks. We plan to pursue this idea in the near future.

Chapter 2

Literature Review

2.1. Image Generation: Past work

2.1.1. Autoregressive models

Autoregressive models try to model the generation of images pixel-wise, with later pixels conditionally dependent on earlier generated pixels. These models are likelihood-based, hence the likelihood of each image under the model can be efficiently calculated.

PixelRNN [**91**] uses an LSTM to model the conditional distribution of one pixel on the previous pixels. This paper also introduces the PixelCNN variant, which uses a CNN instead of an LSTM, and so is faster to train, but achieves lesser performance than PixelRNN. Conditional PixelCNN [**119**] introduces Gated PixelCNN which combines the best of both, and so is able to train faster as well as outperform PixelRNN. It also introduces a conditional variant, where the generation of images is conditioned on the class. [**104**] introduces a parallel inference mechanism that speeds up inference. Whereas in the earlier models inference occurred by sampling each pixel one after another, this paper introduces the notion of pixel groups being conditionally independent, and so inference can be parallelized among these groups. PixelCNN++ [**111**] improves upon the previous architectures by introducing several incremental changes, including computation at multiple resolutions.

Autoregressive models have several advantages such as exact density estimation. However, since the number of evaluations for sampling is linearly dependent on the number of pixels in the generated image, they have not gained as much success in the field of image or video generation as other methods.

2.1.2. Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) [**42**] were introduced in 2014 as a likelihood-free generative model of images. The core idea revolved around two adversaries, a generator and a discriminator, training themselves by competing with each other. The generator learns to map from noise $\mathbf{z}$ to an image prediction $\hat{\mathbf{x}}$ that should belong to the data distribution, while the discriminator learns to classify a real image $\mathbf{x}$ from the generated image $\hat{\mathbf{x}}$. GANs are likelihood-free models, they cannot be used to sample images based on their likelihood. There are some GAN architectures that proposed to tackle the generation of images from a multi-scale perspective.

One of the first GAN architectures to propose a multi-scale structure to the generation of images was LAPGAN [**28**]. It proposes to generate an image in multiple scales using the Laplacian Pyramid [**10**]. Generation can be performed by sequentially sampling noise at increasing resolutions, and converting those noise samples to Laplacians at each scale, which are then added to the upsampling of the image at the previous scale. The model is trained adversarially by using a discriminator at each resolution to classify a real Laplacian image from a generated Laplacian image.

ProGAN [**66**] tackled the problem by progressively generating finer and finer resolutions from the coarser ones. It proposed to train the generator and discriminator at multiple resolutions, by training to completion at each resolution and then increasing the capacity of both when proceeding to the next resolution. This was the first to produce high resolution images (1024×1024) using GANs.

BigGAN [**8**] changed the noise distribution to a hierarchical one. It built on top of SNGAN [**86**], and was able to produce state-of-the-art quality of images at 128×128 and 256×256. It also showed results on linear interpolations of noise samples.

Another GAN architecture that was popular for producing high resolution images of faces was StyleGAN [**67**]. It split the architecture into two parts : a style encoder, and a synthesis network. The style encoder would take a noise sample i.e. latent code, and embed it into a feature that represented style. This style feature would then be used as a condition to generate a new image of a face, while using hierarchical noise to introduce other details.

More recently, SinGAN [**113**] also follows a multi-scale architecture, where at each scale the generator takes as input the upsampled image from the lower scale, and a noise sample.

This architecture is trained using two losses: (1) an adversarial loss where a patch discriminator at each scale tries to classify the overlapping patches of the real and generated images, and (2) a reconstruction loss between the generated image at a scale with no noise and the real image at that scale. They show various applications of this architecture, including editing and super-resolution.

Although GANs have been successful in producing sharp images owing to the adversarial loss, they are not useful in density estimation since they lack a closed-form likelihood. Instead, the discriminator performs the job of estimating a version of the divergence between the real and generated data distributions, which is used as a supervisory signal by the generator.

2.1.3. Normalizing Flows

Normalizing Flows are likelihood-based generative models that map a series of invertible transformations from noise to data, and vice-versa if the transformation is invertible. Using invertible transformations, the data could be transformed to noise, and the likelihood of the data can be estimated from the likelihood of the noise using the change of variables formula:

$$\mathbf{x} = f(\mathbf{z}) \tag{2.1}$$

$$\implies p(\mathbf{x}) = p(\mathbf{z}).\left|\det \frac{\mathrm{d}\mathbf{z}}{\mathrm{d}\mathbf{x}}\right| \implies p(\mathbf{x}) = p(\mathbf{z}).\left|\det \frac{\mathrm{d}f}{\mathrm{d}\mathbf{z}}\right|^{-1} \tag{2.2}$$

$$\implies \log p(\mathbf{x}) = \log p(\mathbf{z}) - \log\left|\det \frac{\mathrm{d}f}{\mathrm{d}\mathbf{z}}\right| \tag{2.3}$$

Considering that $\mathbf{x}$ is the result of a series of transformations:

$$\mathbf{x} = f_K \circ f_{K-1} \circ \cdots \circ f_1(\mathbf{z}_0) \tag{2.4}$$

$$\implies \log p(\mathbf{x}) = \log p(\mathbf{z}_0) - \sum_{i}^{K} \log\left|\det \frac{\mathrm{d}f_i}{\mathrm{d}\mathbf{z}_{i-1}}\right| \tag{2.5}$$

Since $\mathbf{z}_i$ is noise, typically Gaussian noise, $p(\mathbf{z}_i)$ is easy to compute. Hence, $p(\mathbf{x})$ can be computed if the log-determinant of the Jacobians $\dfrac{\mathrm{d}f_i}{\mathrm{d}\mathbf{z}_{i-1}}$ are easy to compute. This model can be used as a generative model if the transformations f_i are invertible.

NICE [**29**] proposed f_i to be an additive coupling layer such that it performs an affine transformation of half its input based on the other half:

$$\begin{cases} \mathbf{b}_{1:d} = \mathbf{a}_{1:d} \\ \mathbf{b}_{d+1:D} = \mathbf{a}_{d+1:D} + m(\mathbf{a}_{1:d}) \end{cases} \tag{2.6}$$

RealNVP [**30**] proposed f_i to be an affine coupling layer:

$$\begin{cases} \mathbf{b}_{1:d} = \mathbf{a}_{1:d} \\ \mathbf{b}_{d+1:D} = \mathbf{a}_{d+1:D} \odot \exp(s(\mathbf{a}_{1:d})) + t(\mathbf{a}_{1:d}) \end{cases} \tag{2.7}$$

Both NICE and RealNVP have a multi-scale architecture that reduces the resolution in steps. At each step, they split the produced feature in half, and map one half to noise, making the noise hierarchical.

Glow [**71**] scales up RealNVP [**30**] to have simpler components, but much more in number. It generates much more realistic-looking samples than previous likelihood-based models. Notably, it replaces the simple permutation of dimensions in the previous flow-based models by a "shuffling" operation of dimensions using an invertible 1×1 convolution. More recently, RAD [**31**] proposed to use a combination of piece-wise invertible functions to map both real and discrete data to noise.

2.1.4. Continuous Normalizing Flows

Continuous Normalizing Flows were introduced in [**15**], and represent a body of work that perform a continuous mapping from latent code to data. Section 3.2 elaborates upon their functionality, and the various advances made in recent times on density estimation and image generation. Chapter 6 describes a novel way of using Continuous Normalizing Flows for image generation.

2.2. Video Generation: Past work

Earlier work on using deep learning algorithms to perform video prediction/generation tackled the problem in various ways. Most methods for video prediction of future frames from past frames use an encoder-decoder architecture. The past frames are encoded into a compact representation, and then decoded back into pixel space for future time steps. In

some cases, the compact representation is disentangles in content and motion, or in static and dynamic components.

2.2.1. Autoregressive models

Video Pixel Networks [**64**] build upon Conditional PixelCNN [**119**] for video. Time dependency of frames is captured using a ConvLSTM that preserves the spatial resolution of the input conditional frames. Each latent is then decoded using PixelCNN decoders, hence making the model autoregressive.

2.2.2. Recurrent methods

In a deterministic setting, [**101**] defined a recurrent network architecture for video prediction inspired from language modeling. Inspired from language modelling, they first made a vocabulary of image patches by clustering all 8×8 patches in all frames in all videos in the dataset to 10,000 centroids using k-means. Then, they used a Convolutional RNN to predict the image patches in future time steps. They argued that using ℓ_2 loss leads to blurry images, hence they circumvent this problem by predicting the indices of image patch centroids.

[**116**] proposed the Moving MNIST dataset, and used sequence-to-sequence LSTM [**53**] to encode conditional frames of a video. Then then decoded the learned representation in two branches: one produced future frames, and the other produced recent past frames. Then, the LSTM network was trained on reconstruction losses of both future and past frames. This helped set the context well for the feature to learn information relevant to both the past and the future. [**116**] argue against [**101**], and claim that ℓ_2 loss was sufficient for them to showcase good results.

ConvLSTM [**114**] is proposed to use convolutional layers in LSTM modules instead of fully-connected layers. This helped them model spatial and temporal variations jointly. 3D convolution [**96**] is also shown to be effective in extracting spatio-temporal information from sequences of images.

[**38**] uses stacked ConvLSTMS [**114**] to predict "motion" in a video, and thereby warps the previous frames according to this motion to generate new frames.

MCnet [**121**] proposes to disentangle "content" and "motion". It encodes only the last frame for a representation of the latest content in the scene. For motion, it recurrently

encodes the differences between consecutive images in the video. During prediction, a single content vector is shared across the frames. Then, that content and the next predicted motion (as image difference) are recurrently combined to predict the frames at future time steps.

DrNet [**26**] proposes to disentangle the representation into "content" and "pose". In addition to reconstruction loss, the content encoder is penalized for dissimilar content representation for frames within a video, and similar representations across videos, using an adversarial loss. The pose is recurrently estimated at each time step using an LSTM. To ensure that the pose encoder does not carry any information about the identity of the object, it is penalized for encoding similar semantic information either within or across videos, using an adversarial loss.

[**122**] propose to build a hierarchical framework to video prediction, by first predicting the high-level structure, and then the future frame from a single frame input. The high-level structure they choose to model is pose, which they obtain using Hourglass Networks [**88**] on the input images. The overall training procedure is: (1) encode the frames by estimating the pose for the conditional frames, (2) use an LSTM to predict pose at future time steps, (3) combine the most recent image with the desired pose in feature space, and decode to pixel space using a Visual Analogy Network (VAN) [**105**]. They are able to predict up to 128 time steps in the future using their pose predictor LSTM, and show good image quality compared to ConvLSTM [**114**], but do not perform well when the pose prediction LSTM fails.

[**128**] build a hierarchical model similar to [**122**] but in an unsupervised manner. While [**122**] imposed ground-truth pose as the high-level structure, [**128**] let the predictor LSTM and VAN figure out a high-level structure the helps the task of video prediction using the same architecture. In one variant of this structure, they add a reconstruction loss between the LSTM's high-level feature predictions at different time steps, and the respective embeddings of those frames by the high-level encoder. In another variant, they add an adversarial loss to the LSTM's predictions.

2.2.3. Stochastic Autoencoder-based methods

Stochasticity is introduced in other works [**20, 27, 2, 76**] to incorporate uncertainty over possible futures given then same past. They encode the conditioning frames into latent space similar to a prior distribution, and then sampled from it to predict the next frames.

[**45**] gives a detailed account of using recurrent neural networks for sequence generation, specifically for the task of handwritten text generation. In this work, the sequence is modeled as a series of strokes, and a probabilistic model is made of the offsets in each stroke and the chance of whether the stroke is made or not. The outputs of a recurrent neural network are modeled as the parameters of the distributions of these stroke offsets and presence.

VRNN [**20**] (Variational Recurrent Neural Network) instead models a variational version of the above. At each time step, the parameters of the distributions of the sequence are decoded from a latent variable. This latent variable is sampled from a prior distribution, which is parameterized by the output of a recurrent neural network. Thus, the trajectory of the underlying parameters of the latent variable distributions at different time steps are modeled by a recurrent neural network, while generation from and inference of this latent distribution are handled by a VAE.

Although the above two works don't use this method for video prediction/generation, later works build on these models.

[**130**] propose to predict a future frame from a past frame by modeling the "content" as feature maps, and "motion" as convolutional kernels. These are parameterized using normal distributions, and the parameters are learned using encoders via variational inference. The motion for the next image is sampled from this distribution as a difference image, and is added to the feature maps of the past image to decode to a future frame. In particular, this model is multi-scale - it combines the content and motion at different scales to generate a new frame.

SV2P [**2**] involves a VAE-style model for video prediction, where the encoder/inference network is conditioned on all frames at all time steps (past and future), and the decoder is conditioned on the past frames and a sample from a latent prior. The loss function involves a reconstruction term for the predicted frames, and a KL-divergence term for the inferred prior. Training occurs in three phases: (1) decoder is trained fixing the prior to a standard Gaussian, (2) the encoder is trained, again fixing the posterior on a standard Gaussian, (3) the KL-divergence term is introduced, and the VAE is trained end-to-end. The model architecture follows [**38**], with an added stochastic component.

SVG [**27**] replaces the convolutional network in SV2P [**2**] that uses all frames at all time steps, with LSTMs to be able to estimate a different posterior at every time step that is

dependent on only the previous frames for each time step. They also introduce a learned prior model SVG-LP, which estimates a posterior for every time step so that at inference time a latent can be sampled from this posterior, instead of a standard Gaussian prior.

DDPAE [**56**] manually disentangles the learned representation into "content" that is static all across the frames of a video, and "pose" that changes with the frames. It then follows the VAE-style architecture, it estimates each pose vector in the latent space using an RNN conditioned on previous pose vectors and frames. The encoder (and decoder) networks are also RNNs conditioned on the previous frames (and current pose).

Improved VRNN [**12**] uses a hierarchy of latent variables to improve the expressiveness of their generative model. They model the video prediction task using a hierarchical VRNN, which uses a hierarchy of latent variables per time step. The latent at each time step is in addition conditionally dependent on the latents at the previous scales as well.

[**120**] scale up SVG [**27**] to high quality video predictions. They replace LSTMs with ConvLSTMS [**114**], and use ℓ_1 loss instead of ℓ_2. They conduct a thorough study on various video prediction challenges, and show significant improvement in image quality.

2.2.4. Unconditional video generation using GANs

Video-LAPGAN [**82**] builds on top of LAPGAN [**28**], which used multiple generators at different scales to produce image Laplacians, and a discriminator at each scale. Video-LAPGAN first proposes to use LAPGAN's architecture to combine information from multiple scales, instead of the typical architectures of convolutions and pooling that may lose information due to the changes in resolution. Secondly, like in [**101**], it argues that ℓ_2 loss results in blurry images. Hence, there are three losses that are used to train the generator: (1) a proposed adversarial loss to match the generated images with possible images from the real dataset, (2) an ℓ_p loss to match the content of the generated images with the respective true images, (3) a proposed Gradient Difference Loss (GDL) that helps sharpen the images by penalizing disparity in horizontal and vertical differences in image pixels between the respective true and generated images.

VGAN [**124**] attempts to generate unconditional video from noise using GANs for video. They use a two-stream generator, i.e. the generator consists of two branches: one for a static "background" and another for a "foreground" with moving objects. The generator generates

all frames in the video jointly. They were not only able to generate 32 frames of 64×64 images, they also showed that the discriminator using 3D deconvolutional layers learned a representation of the video that was useful for downstream tasks such as action recognition.

TGAN [**109**] (Temporal GAN) adopted the WGAN [**1**] framework in making their discriminator K-Lipschitz. While VGAN [**124**] split the video into static "foreground" and moving "background", TGAN does not explicitly split the video;s components in pixel space. It instead models the video in latent space by sampling from a noise distribution first, $\mathbf{z}_0$, and then using a generator to transform from this latent point to different latent points corresponding to different time steps $[\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_T]$. It then uses another generator to transform each tuple $(\mathbf{z}_0, \mathbf{z}_t)$ into pixel space to make each frame independently. Finally, a spatio-temporal discriminator containing 3D convolutions is used to discriminate between real and generated videos.

[**125**] tries to estimate the relative transformation required to transform past frames into future frames. They train this using an adversarial loss.

MoCoGAN [**117**] presents an adversarial version of MCnet [**121**]. The video is represented as a sequence of latent points that are dimensionally split into "content" and "motion". The content is fixed for a video clip, while the motion varies within each clip. The variation of motion is modelled by an RNN, which takes a random Gaussian samples as input at each time step and maps them to a valid sequence of latent points. Each latent point is then decoded into a frame of the video by a generator. Two discriminators — one for each frame, and another for the entire video — are used to train the generator and RNN adversarially.

2.2.5. Combining adversarial loss with encoder-decoder framework

SAVP [**76**] also builds on SV2P [**2**], and adds an adversarial loss using a discriminator network that tries to classify generated frames from real frames. The adversarial loss is proposed since pure VAE loss leads to blurry samples, and adversarial losses have been shown to promote samples belonging to the true data manifold.

Dual Motion GAN [**78**] encodes the conditional frames, samples from the embedded distribution (like a VAE), and generates (1) an estimate of the flow for the future frame, and (2) an estimate of the future frame itself. It then "fuses" (2) and latest input frame warped by (1) into a better estimate of the future frame. The flow estimate, and the flow

between the latest frame and the future frame estimate are used as fake samples, while the true flow between the latest frame and true future frame are used as real samples for a flow discriminator. Equivalently, the future frame estimate, and the frame produced by the warping of the latest frame and the flow estimate are used as fake samples, while the true future frame is used as the real sample, to train a second discriminator on frames.

Vid2Vid [**126**] conditions on past frames as well as semantic segmentation maps in the future in a multi-scale fashion to generate single step future frame. While their results may be interesting, due to the conditioning, the model relies heavily on ground truth semantic segmentation.

2.2.6. Continuous Normalizing Flow-based methods

While some of the above methods have yielded state-of-the-art results, some still struggle to produce smooth motions. Those who do produce continuously smooth motion enforce it through temporal regularization in the optimisation objective, or through a specific training procedure.

Drawing from recent work on using parameterized ODE estimators [**15**] to model continuous-time dynamics, we choose to approach this problem with the intuition that we would like the video frames to be smoothly connected in latent space according to some continuous dynamics which we would learn. Continuously-defined dynamics should naturally allow to process observations occurring at non-uniform intervals and generate at any time. This is unlike recurrent neural networks and other purely auto-regressive approaches which require observations to occur at uniform intervals, and may require three different models to extrapolate forward and backwards, or interpolate. Thus, the same model can perform extrapolation and interpolation.

Notably, [**123, 131**] use Neural ODEs for video prediction. Chapter 4 elaborates on [**123**], and describes how Neural ODEs can be effectively used for video generation. Chapter 5 then describes our experiments in scaling up the results of Chapter 4 to realistic videos.

Chapter 3

Neural ODEs

In the recent past, Neural Ordinary Differential Equations (Neural ODEs) [**15**] were introduced as a differentiable way to perform approximate integration using neural networks.

Neural ODEs [**15**] represent a family of parameterized algorithms designed to model the evolution across time of any system with state $\mathbf{x}(t)$ at an arbitrary time t, governed by continuous-time dynamics satisfying a Cauchy (or initial value) problem:

$$\begin{cases} \mathbf{x}(t_0) = \mathbf{x}_0 \\ \dfrac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = f(\mathbf{x}(t), t) \end{cases} \tag{3.1}$$

By approximating the differential with an estimator $f_\theta \simeq f$ parameterized by θ, such as a neural network, these methods allow to learn such dynamics (or trajectories) from relevant data. This is a paradigm shift from earlier methods which typically defined a differential function by hand-design. Parameterization of the differential allows the learning of such dynamics from data through standard machine learning optimization such as gradient descent.

Thus formalized, the state $\mathbf{x}(t)$ of such a system is defined at all continuous times, and can be computed at any desired time using a numerical ODE Solver. This ODE Solver will evaluate the dynamics f_θ to determine the solution given the initial value $\mathbf{x}_0$, initial time step t_0, and the desired time steps:

$$(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \texttt{ODEsolver}(f_\theta, \mathbf{x}_0, t_0, (t_1, \ldots, t_n)) \tag{3.2}$$

For any single arbitrary time value t_i, a call to the `ODEsolver` computes a numerical approximation of the integral of the dynamics from the initial time value t_0 to t_i.

$$\mathbf{x}_i = \texttt{ODEsolver}(f_\theta, \mathbf{x}_0, t_0, t_i) \simeq \mathbf{x}_0 + \int_{t_0}^{t_i} f_\theta\left(\mathbf{x}(t), t\right) \mathrm{d}t = \mathbf{x}(t_i) \tag{3.3}$$

A plethora of algorithms exist in literature to perform numerical integration of differential equations. These include the simplest, Euler's method; higher order methods such as Runge-Kutta methods [**46, 74, 95**]; as well as multi-step algorithms like the Adams-Moulton-Bashforth methods [**11, 46, 99**]. More advanced algorithms have been proposed to provide better control over the approximation error and accuracy [**97**]. In their implementation [1], [**15**] use a variant of the fourth-order Runge-Kutta method with adaptive step size and monitoring of the local truncation error to ensure accuracy.

3.0.1. Adjoint method

It is possible to optimize for the parameters of the neural network by back-propagating through the iterations of the ODE Solver directly. However, it is highly memory-intensive, since the activations of the neural network at the end of every iteration must be saved. Moreover, if an adaptive-step ODE Solver is used, the number of iterations performed might be too high to save in memory.

Thus, the optimization of the Neural ODE is performed through the framework of adjoint sensitivity [**95**]. If the scalar-valued objective function $L(\theta)$ is provided:

$$L(\theta) = L\left(\mathbf{x}_0 + \int_{t_0}^{t_i} f_\theta\left(\mathbf{x}_0(t), t\right) \, dt\right) \tag{3.4}$$

the gradient of the objective with respect to the model parameters follows the differential system:

$$\mathbf{a}(t) = \frac{\partial L}{\partial \mathbf{x}(t)} \tag{3.5}$$

$$\frac{\partial \mathbf{a}(t)}{\partial t} = -\, \mathbf{a}(t)^\top \frac{\partial f_\theta(\mathbf{x}(t), t)}{\partial \mathbf{x}(t)} \tag{3.6}$$

$$\frac{\partial L}{\partial \theta} = \int_{t_i}^{t_0} -\mathbf{a}(t)^\top \frac{\partial f_\theta(\mathbf{x}(t), t)}{\partial \theta} dt \tag{3.7}$$

where $\mathbf{a}(t) = \partial L / \partial \mathbf{x}(t)$ is the adjoint.

1. https://github.com/rtqichen/torchdiffeq

Hence, while forward propagating through the Neural ODE, i.e. integrating forwards in time, there is no need to save any of the intermediate activations. At the end of the forward integration, the adjoint at the final time step is computed using equation 3.5.

The gradient of the loss with respect to the parameters can be computing using the backward integration of the expression in equation 3.7 from the final time step, say t_1, to the initial time step t_0. This involves the computation of $\mathbf{x}(t)$ and $\mathbf{a}(t)$ at intermediate time steps. Each of these expressions also follow a differential equation, the differential for $\mathbf{x}(t)$ being the same as in the forward integration, i.e. $f(\mathbf{x}(t), t)$, and the differential of $\mathbf{a}(t)$ is given by equation 3.6.

Hence, three differential equations need to be solved to compute $\frac{\partial L}{\partial \theta}$. These three ODE-Solves can be combined into one ODE Solve by augmenting the respective states, and integrating backwards from t_1 to t_0:

$$\begin{bmatrix} \mathbf{x}(t) \\ \frac{\partial L}{\partial \mathbf{x}(t_0)} \\ \frac{\partial L}{\partial \theta} \end{bmatrix} = \texttt{ODEsolver}\left(\begin{bmatrix} f_\theta(\mathbf{x}(t), t) \\ -\mathbf{a}(t)^\top \frac{\partial f_\theta(\mathbf{x}(t), t)}{\partial \mathbf{x}(t)} \\ -\mathbf{a}(t)^\top \frac{\partial f_\theta(\mathbf{x}(t), t)}{\partial \theta} \end{bmatrix}, \begin{bmatrix} \mathbf{x}(t_1) \\ \frac{\partial L}{\partial \mathbf{x}(t_1)} \\ \mathbf{0}_{|\theta|} \end{bmatrix}, t_1, t_0 \right) \tag{3.8}$$

Thus, using equation 3.8, $\frac{\partial L}{\partial \theta}$ can be computed without the need to store any activations in memory, and the weights of the neural network can be updated using any standard gradient-based optimization strategies such as gradient descent.

3.0.2. Later research on Neural ODEs

Since Neural ODEs were introduced, some work has been conducted to explore the nuances and expand their capacity. [**34**] identifies a simple case where Neural ODEs do not work, namely that a simple negative mapping from the real line to the real line is not possible since it would involve intersecting trajectories. To alleviate this, it proposes to augment the state of with extra dimensions, so that the flow is now a homeomorphism in the augmented space, but not in the original space. [**135**] provides guarantees on the modelling capability of homeomorphisms, in comparison with the capacity of Neural ODEs.

[**136**] expand the capacity by involving a second "weight network" Neural ODE to determine the parameters of the first "parent network" Neural ODE. [**47**] explores the robustness of Neural ODEs, and shows that it is better than typical Convolutional Neural Networks (CNNs) on the task of classification.

[**61**] and [**79**] incorporate stochasticity in the formulation of Neural ODEs by adding a stochastic component to the deterministic component in the differential. [**77**] provides a comprehensive framework of modeling stochasticity in Neural ODEs, and released the code [2].

3.1. Latent Variable Models

Neural ODEs can also be used to make latent variable encoder-decoder models like Variational Auto-Encoders [**70**]. An encoder is used to encode input data into a latent representation, preferably as parameters of a latent distribution. Typically the data used is time-series data, in which case a recurrent encoder could be used. This encoded latent distribution is then sampled from, the sample is used as the initial value to a Neural ODE. The Neural ODE is fed this initial latent value, and is used to perform numerical integration from the initial time step to a/multiple future time step(s). Thus, latent points at future time steps are estimated by the Neural ODE. Each latent point is then decoded into the original distribution. This Encoder-Neural_ODE-Decoder framework is trained using a reconstruction loss between the original input points and those predicted by the decoder.

Hence, the networks are trained via self-supervised learning, over the task of reconstruction. The advantage of using Neural ODEs in the latent space is that it is possible to integrate even further in time, and those latent points could be decoded to estimate time points in the future for free.

[**108**] improves this model for irregularly sampled data by incorporating a Neural ODE in the encoder framework where data is missing. [**131**] uses 2nd order Neural ODEs and bayesian neural network weights to train a generative model of videos.

Chapter 4 describes how we perform video generation using this model, as published in [**123**]. Chapter 5 describes how we expand this to video generation via object detection.

2. https://github.com/google-research/torchsde

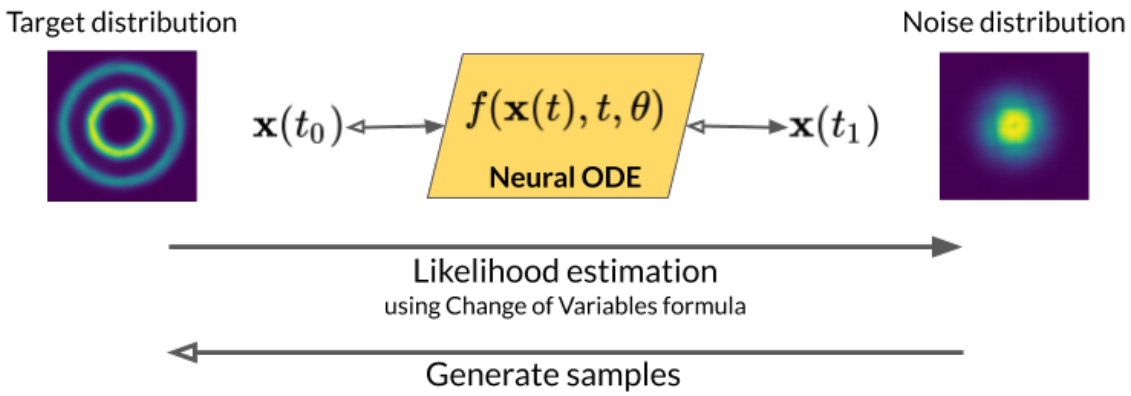


Figure 1. Continuous Normalizing Flows [**15**]

3.2. Continuous Normalizing Flows

Neural ODEs are reversible models. The direction of flow of information can be fixed by the direction of integration : forwards or backwards. Since Neural ODEs can transform one distribution to another using Ordinary Differential Equations (ODEs), they follow the same rules of transformation as ODEs. Primarily, Neural ODEs preserve dimensionality, and they form non-intersecting trajectories, i.e. they describe flows.

This implies that it is possible to map a target distribution, such as the real image manifold, to a noise distribution such as the Gaussian distribution using a Neural ODE, if the transformation can obey the above-mentioned rules, i.e. if the transformation is a flow. Since Neural ODEs operate in continuous dimensional space, these flows from noise to target are called Continuous Normalizing Flows [**15**] (Fig. 1).

The training of a Neural ODE as a Continuous Normalizing Flow takes place by maximizing the likelihood of data from the target distribution. This is calculated as the likelihood of the transformed data in the known noise distribution, times the change in likelihood due to the transformation by the Neural ODE. This can be expressed using the change of variables formula as:

$$\mathbf{x}_1 = g(\mathbf{x}_0) \implies \log p(\mathbf{x}_0) = \log p(\mathbf{x}_1) + \log \left| \det \frac{\partial g}{\partial \mathbf{x}_0} \right| \tag{3.9}$$

Here, $\mathbf{x}_1 = g(\mathbf{x}_0)$ represents equation 3.3, encapsulating the approximate integration, where $\mathbf{x}_0$ represents a data sample from the target distribution, and $\mathbf{x}_1$ is the transformed data point that is supposed to belong to the noise distribution. Hence, training involves maximizing $\log p(x_0)$, by maximizing the RHS of equation 3.9.

The log-determinant term is replaced with another differential equation using the Instantaneous Change-of-Variables formula [**15**]:

$$\frac{\partial \log p(\mathbf{x}(t))}{\partial t} = -\mathrm{tr}\left(\frac{\partial f}{\partial \mathbf{x}(t)}\right) \tag{3.10}$$

Generation of samples involves sampling a random point in the noise distribution $\mathbf{x}_1$, transforming it back into the target distribution by integrating the Neural ODE backwards.

3.2.1. Later research on Continuous Normalizing Flows

FFJORD [**43**] extends the concept of Continuous Normalizing Flows from [**15**] called Free-Form Jacobian , by presenting an efficient way of computing the log-determinant term in equation 3.9. It achieves state-of-the-art among exact likelihood methods with efficient sampling. It then presents results on generative modeling of images, and achieves state-of-the-art bits-per-dim on the generation of MNIST [**75**] and CIFAR-10 [**73**]. It also demonstrates results on high-dimensional density estimation and variational inference.

FFJORD-RNODE [**37**] builds on top of FFJORD [**43**], by introducing two regularization terms related to the kinetic energy of the flow, and its Jacobian norm. It achieves similar bits-per-dim as [**43**] but in much lesser time. In addition, it reports bits-per-dim for images from ImageNet [**25**] and CelebA [**80**].

SNODE [**98**] proposes to formulate the parametric function as a truncated series of Legendre polynomials, thereby constraining the dynamics of the flow. More recently, STEER [**40**] proposes a regularization scheme by making the final time step of integration a random parameter in a uniform range. TayNODE [**68**] proposes another regularization scheme by penalizing the higher-order Taylor series coefficients of the learned trajectory.

Chapter 6 describes a novel method that builds on top of FFJORD-RNODE in making a multi-scale generative model of images. This method achieves state-of-the-art bits-per-dim in the least time when trained on images from CIFAR10 and ImageNet64.

Chapter 4

Simple Video Generation using Neural ODEs

Video generation in a complex task involving the generation of frames of a scene typically using information from previous frames. Chapter 2.2 described past work in the field. In this chapter, we shall see the method adopted in the NeurIPS 2019 Workshop paper "Simple Video Generation using Neural ODEs" by Vikram Voleti, David Kanaa, Samira Kahou, Christopher Pal [**123**].

Our approach to the problem of video generation from conditional frames combines the familiar encoder-decoder architecture of neural network models with a Neural ODE that works in the latent space.

More formally, in accordance with established formulations of the task of video prediction, let us assume a set of m contextual frames $\mathcal{C} = \{(\mathbf{x}_i, t_i)\}_{i \in [0,m)}$. We seek to learn a predictive model such that, provided $\mathcal{C}$, we can make predictions $\mathcal{P}(\mathcal{C}) = \{(\mathbf{x}_j, t_j)\}_{j \in [m,n)}$ about the evolution of the video across time, arbitrarily in the future or past (extrapolation) or even in between observed frames (interpolation).

4.1. Method

A video is composed of spatial and temporal components. The temporal changes in the raw video can be interpreted as effects of temporal variations in the latent concepts embedded within it. For example, suppose we have a video of a ball moving. Any temporal change in the video will be observed only on pixels related to the latent notion of "moving ball". Hence, it is intuitive to model dynamics in latent space and capture spatial characteristics separately.

Thus we learn a predictor which (a) learns a latent representation of the observed discrete sequence of frames that captures spatial factors of variation, as well as (b) infers plausible latent continuous dynamics from which the aforementioned discrete sequence may be sampled i.e. which better explains the temporal variations within the sequence.

The proposed model follows the formalism of latent variable model proposed in [**15**]. The latent point at the initial time value $\mathbf{z}(t_0)$ is sampled from a distribution $\mathbb{P}_Z$. The latent generative process is defined by a Neural ODE that determines the trajectory followed in latent space from the initial latent $\mathbf{z}(t_0)$ to latent points at other time steps $\mathbf{z}(t_i)$. A conditional $\mathbb{P}_{X|Z}$ is used to independently sample predicted images from latent vectors predicted along the trajectory:

$$\mathbf{z}_0 \sim \mathbb{P}_Z(\cdot) \tag{4.1}$$

$$\mathbf{z}(t_i) = \mathbf{z}_0 + \int_{t_0}^{t_i} f_\theta(\mathbf{z}(s), s)\, \mathrm{d}s \qquad \forall\, t_i \in [t_0, t_n) \tag{4.2}$$

$$\hat{\mathbf{x}}(t_i) \sim \mathbb{P}_{X|Z}(\cdot \,|\, \mathbf{z}(t_i)), \quad \hat{\mathbf{x}}(t_i) \perp\!\!\!\perp \hat{\mathbf{x}}(t_j) \qquad \forall t_i, t_j \in [t_0, t_n) \tag{4.3}$$

In practice, we use an approximate posterior $q_\phi(\cdot \,|\, \mathcal{C})$ instead of $\mathbb{P}_Z$, and similarly, instead of $\mathbb{P}_{X|Z}$, we use an estimator $p_\psi(\cdot \,|\, \mathbf{z}(t_i))$. Together, these estimators function as an *encoder-decoder* pair between the space of image pixels and that of latent representations.

The variational encoder q_ϕ is based on a Recurrent Neural Network over the context $\mathcal{C} = \{(\mathbf{x}_i, t_i)\}_{i \in [0,n)}$. The decoder p_ψ is non-recurrent, following the hypothesis of independence between generated frames. The temporal dependencies are modelled by the Neural ODE. At training time, the entire estimator is optimized as a variational auto-encoder [**70, 106**] through the maximization of the Evidence Lower Bound (ELBO):

$$\mathcal{E}(\phi, \theta, \psi) = \underbrace{\sum_{[t_0, t_n)} -\mathbb{E}_{z_0 \sim q_\phi(\cdot \,|\, \mathcal{C})}[\log p_\psi(\hat{\mathbf{x}}_t \,|\, \mathbf{z}_t)]}_{\text{reconstruction term}} + \mathrm{D}_{KL}(q_\phi(\cdot \,|\, \mathcal{C}) \,\|\, \mathcal{N}(0, \boldsymbol{I})) \tag{4.4}$$

Hence, the mechanism can be described simply as:

(1) The input conditional frames are (recurrently) encoded into a latent distribution.

(2) This encoded latent distribution is sampled from, and the sample is fed to a Neural ODE as the "initial value" at time $t = 0$.

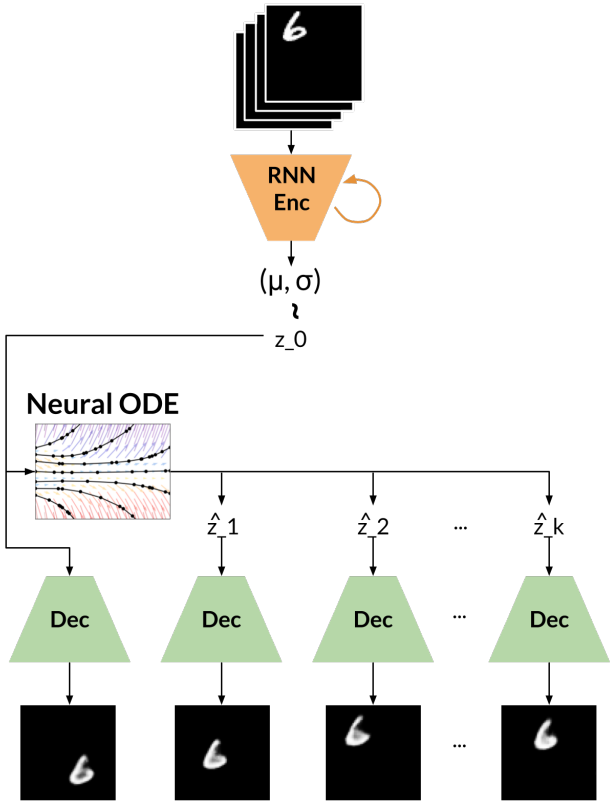


Figure 2. Model for Simple Video Generation using Neural ODEs [**123**]

(3) The Neural ODE is used to predict latent points corresponding to future time steps, by integrating from the initial value up to the time steps of the conditional frames, or beyond.

(4) The latent points are decoded into frames in pixel space at different time steps.

(5) To train the Encoder-Neural_ODE-Decoder framework, a reconstruction loss is computed between the decoded frames and true frames at the respective time steps.

In this way, the Encoder-Neural_ODE-Decoder framework is trained to reconstruct the input conditional frames, and predicted some time steps in future. Refer Fig. 2. Hence, training occurs in a self-supervised fashion. This helps exploit the vast repository of video data available. This is similar to a Variational Recurrent Neural Network [**21**], except here a Neural ODE handles the latent space.

Once trained, this Encoder-Neural_ODE-Decoder framework can then be used to predict future time steps beyond those which it was trained on, by simply using the Neural ODE to integrate further in time (extrapolation).

It can also be used to interpolate in between frames at fractional time steps by integrating the Neural ODE up to those fractional time steps and decoding the latent points (interpolation). Hence, we get interpolation and extrapolation for free. This could be helpful in, say, increasing the frame rate of videos, or constructing slow-motion videos.

4.2. Results

We explore the results for 1-digit and 2-digit Moving MNIST [**116**]. In each case, we use the first 10 frames as both input to the model and as ground truth for reconstruction, which is the output of the model. We then check how the model performs on the subsequent 10 frames.

4.2.1. 1-digit Moving MNIST

We feed the first 10 frames as conditioning input to a Recurrent Neural Network (RNN). This network outputs the mean and variance of a multivariate Gaussian distribution. We sample a latent point from this distribution, and feed this to a Neural ODE as the initial value of the latent variable at $t = 0$. The Neural ODE then predicts latent representations at the first 10 time steps, which we then decode independently to raw pixels. We compute a reconstruction loss between the predicted frames and the original frames in the first 10 time steps. We also add a KL-divergence loss between the predicted Gaussian distribution and the standard normal distribution, to constrain the latent representation to follow a standard normal prior.

The model architectures of the encoder (except the recurrent part) and the decoder are the same as in the previous model. We provide 10000 videos as training input, and use a batch size of 128. Figure 3 (a) shows the results using this architecture. We can see that the model has been able to capture both structural information and temporal information.

4.2.2. 2-digit Moving MNIST

We use the same architecture as for 1-digit Moving MNIST (Fig. 2) to try to reconstruct 2-digit Moving MNIST. We used the same model settings (number of layers, number of channels, etc.) and the same optimization settings. We stopped the training at 2000 epochs, same as that for 1-digit Moving MNIST.

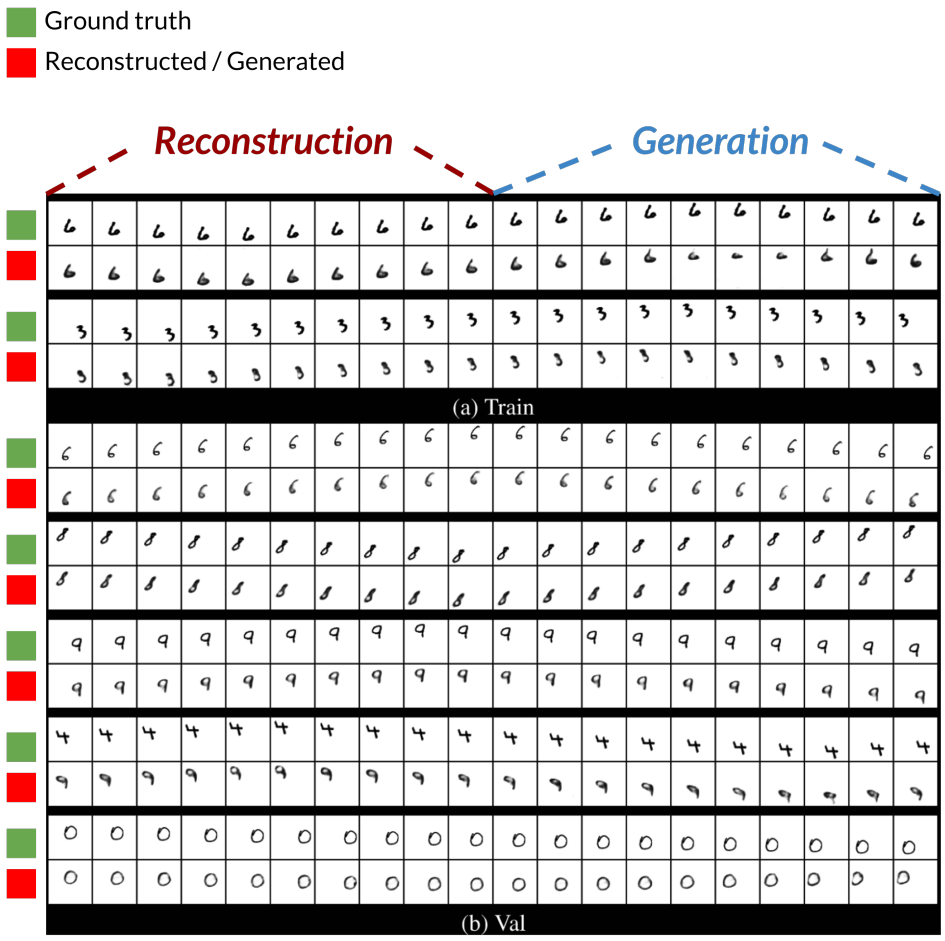
Figure 3 (b) shows some samples from a model trained on 2-digit Moving MNIST. We believe the spatial trajectories of each individual digit are being recorded very well by the Neural ODE. However it would take many more epochs for the encoder and decoder to reconstruct the images better. This phenomenon of the Neural ODE training earlier than the encoder-decoder was observed while training 1-digit Moving MNIST as well.

4.3. Reconstruction, not prediction

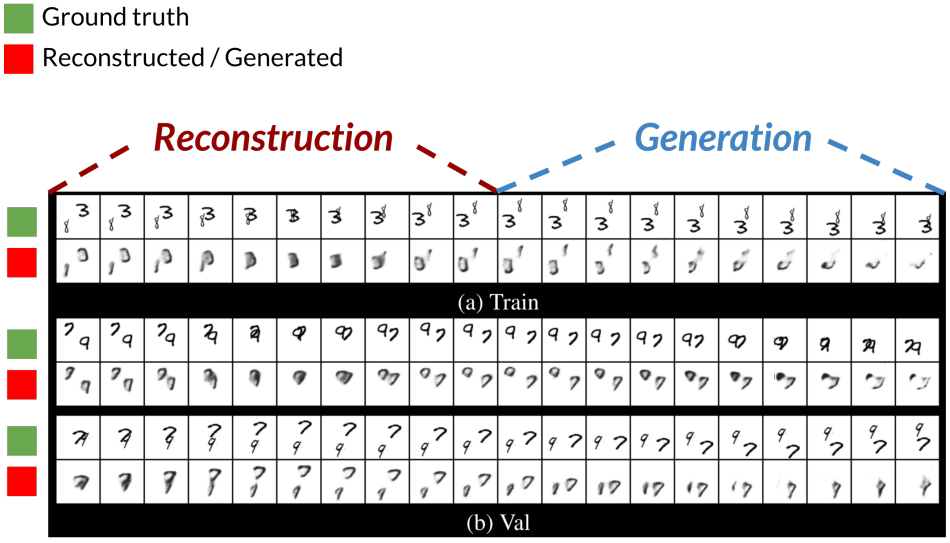
It is to be noted that there is a difference between our problem formulation, and the typical video prediction problem. The typical way of generating videos is to condition a model on the first few frames, and train it to predict some time steps in the future. Then in evaluation, the model is conditioned on the first few frames of unseen videos, and used to predict for time steps including and beyond those for which it was trained on.

In our training procedure, we condition our model on the first few frames, and then simply reconstruct those same frames. Hence, we formulate the training problem as *reconstruction* instead of prediction. Despite this, at evaluation, we are able to predict future frames very well. This is because the dynamics followed in the set of frames used at training are preserved throughout the subsequent time steps. And so by following the trajectory in latent space and decoding it, we are able to predict future frames.

This helps assert the fact that this framework could be used to construct a powerful model of videos that is able to capture information for the task of reconstruction that are useful for prediction. This is possible since the Neural ODE captures the temporal dynamics of the video, while the spatial components are handled by the encoder and decoder.



(a) Moving MNIST 1-digit



(b) Moving MNIST 2-digit

Figure 3. Results of Simple Video Generation using Neural ODEs [**123**]

Chapter 5

Modeling object dynamics using Neural ODEs

Building from Chapter 4, we can see that Neural ODEs are adept at modeling latent dynamics on a simple dataset (Moving MNIST). In this chapter, we scale up and explore the possibility of modeling pre-trained semantic information in realistic videos using the Berkeley Driving Dataset [**133**].

5.1. Modeling object dynamics

We tested whether semantic information in a scene can be modelled using temporal dynamics via Neural ODEs. Whereas the earlier model trained an end-to-end architecture of Encoder-Neural_ODE-Decoder, it did so by encoding the input frames into a latent space that was suitable for reconstruction via Neural ODE. Here, the images are first fed into an encoder that captures semantic information in the scene, such as an object detection model. We used the popular object detection model YOLO [**102**] as the feature extractor. We extract the pre-final layer features from YOLO, and use those as inputs to the Encoder-Neural_ODE-Decoder framework. This way, the Neural ODE is trained to model the temporal dynamics of the semantic information in the scene.

5.1.1. Method

Fig. 4 shows the overall framework:

(1) Extract features of video frames using a pre-trained feature encoder.

(2) Use the Neural ODE to predict the features in future time steps.

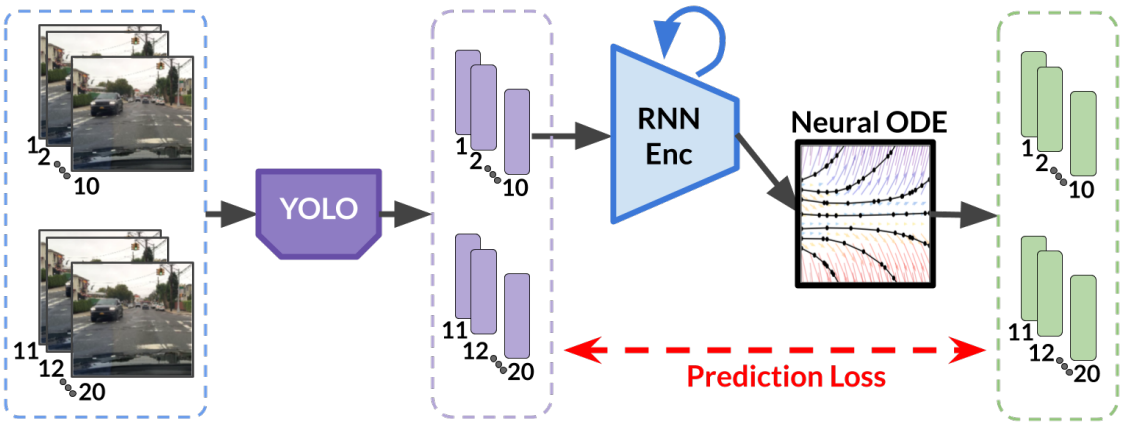


Figure 4. Model of ODE for YOLO

By modeling the YOLO features using a Neural ODE, we are modeling the flow of latent dynamics as described by the semantic information embedded in the YOLO features. It is to be noted that in practice we used YOLOv3 [**103**] [1] to extract features from the frames.

This is similar to ROLO [**90**] (Recurrent YOLO), as well as other methods [**39, 63, 72**] that track objects in time using RNNs. However, we replace the RNNs with a Neural ODE. As discussed earlier, the advantage of Neural ODEs is that latent dynamics are continuous in time.

5.1.2. Results of modeling object dynamics

Fig. 5 shows the results of our model on videos from the Berkeley Driving Dataset [**133**]. The Neural ODE was trained by conditioning on the YOLO features of 3 frames, to to predict the YOLO features of the next 3 frames. The trained Neural ODE was then used to generate the features 3 frames further in the future.

As can be seen from Fig. 5, the trained Neural ODE was able to capture the semantic information in the scene to such an extent as to not only be able to predict well, but also to generate in the future. In some cases, although the ground truth of YOLO predictions did not capture all the cars on the road, the predictions by the Neural ODE did. This indicates that (a) the YOLO feature space captures important semantic information in the scene that can be modeled using continuous dynamics, (b) the Neural ODE is able to model this well and predict object locations and classes that match those in the ground truth.

1. https://github.com/eriklindernoren/PyTorch-YOLOv3

The discrepancy in the predictions from YOLO and the Neural ODE are due to the suppression of spurious detections at the end of the prediction. Spurious detections are suppressed first by filtering the confidence of the predicted class, and then by erasing similarly-located predictions using non-max suppression. For the predictions in Fig. 5, YOLO used a class confidence threshold of 0.8, while the Neural ODE used 0.015. With these settings, the resulting mean Average Precision (mAP) was 11.83%. Despite the wide difference in class confidence, it can be seen that the Neural ODE-predicted feature captures much of the semantic information in the scene, such that the semantic information at future time steps can be predicted.

5.2. Video generation from object features

To address the original problem of video generation, we then tried to see if another reversible neural network could be applied to map between the pixel space of videos and the feature space of YOLO. The advantage with this model is that it could potentially be trained to directly predict both object-related semantic features such as object location, class, etc., as well as the video of the scene at future time steps.

5.2.1. Method

This model is illustrated in Fig. 6. We choose an i-RevNet [**60**] to map between the pixel space of frames in the videos, and the feature space of YOLO. i-RevNet is an invertible architecture that maps from one domain to another domain of a possibly different dimensionality (although there are some limitations on the final dimensionality). We could not use a Neural ODE itself since a Neural ODE preserves dimensions. i-RevNet has the capability of cleverly downsampling the spatial resolution of the image while performing convolutions on the intermediate tensors, all the while maintaining the entire operation invertible. In this way, the architecture is similar to RealNVP [**30**] and NICE [**29**]. Hence, once trained, it should be possible to either feed video frames from one end of the i-RevNet and get the corresponding YOLO features out the other, or feed YOLO features from the other end and get back the corresponding video frames.

The i-RevNet is separately trained to map between the pixel space of video frames and the feature space of YOLO features. The RNN Encoder and Neural ODE are trained as

before, to predict future YOLO features conditioned on past features. Finally, both models are joined as shown in Fig. 6.

(1) The conditional frames of a video are converted to features by YOLO.

(2) The YOLO features are input to the RNN Encoder and Neural ODE, which then predicts YOLO features in future time steps.

(3) The predicted YOLO features are then mapped back to pixel space using the i-RevNet.

5.2.2. Results of video generation

Fig. 7 shows the frame predictions of i-RevNet from the feature predictions of the Neural ODE. As can be seen, the i-RevNet fails to predict the frames of the video from the predicted features. We believe this is because the distribution of the ODE-predicted features is sufficiently different from the original YOLO features. This makes the i-RevNet predictions from seemingly intermediate features rather than the features themselves. The image predictions of i-RevNet at linearly interpolated points is shown in [**60**](Fig. 5). As can be seen, the i-RevNet architecture is adept at reconstructing images from known features, but the images from interpolated features are not realistic. Hence, we conclude that the i-RevNet is perhaps not the best architecture in this setting where the feature predictions by the Neural ODE are sufficiently distant from the original features.

5.3. Next steps

The goal of this project is to explore whether Neural ODEs can model the latent dynamics in realistic videos. We wish to be able to predict the semantic features of a video at future time steps, and then map those features back into a prediction of the video frames.

In Chapter 4, we confirmed the hypothesis that Neural ODEs are capable of modeling the temporal dynamics in video. In this chapter, we first tested whether Neural ODEs are capable of modeling pre-trained temporal dynamics, and then we tested whether the predicted features can be used for generating the video back. Although the first question has been answered in the affirmative, it remains to be seen how to make the Neural ODE more accurate. For the second question, perhaps a more suitable architecture than i-RevNet might prove to be better equipped to handle this situation. In the following sections, we propose various directions to take this project forward.

5.3.1. Improve feature predictions

Loss formulation: The critical component in this architecture is the fact that a Neural ODE is being used to model the temporal dynamics of semantic features. Currently, we use a variant of the mean squared loss (MSE) on the reconstructed video/feature as the training signal. However, as [**82**] showed, better loss formulations could lead to significant improvement in training. Whereas earlier we dealt with relatively typical data such as images, here we deal with features that are very high-dimensional data. MSE might not be the right choice to model such high dimensions. Minimizing MSE is effectively the same as maximizing the likelihood of a generated sample with respect to a Gaussian around the true sample. As shown in [**113**], a variant of MSE could be used that tunes the standard deviation of the Gaussian according to the realism of the generated image.

Adversarial loss: Another challenge was that the predictions of the Neural ODE did not match the original features enough for the i-RevNet to map them back properly. Adversarial loss has been shown to be effective in producing data that matches possible samples of the real data distribution rather than an average sample [**82, 28, 124, 109, 125, 117, 76, 126**]. Hence, an adversarial loss could help the Neural ODE be more accurate in high dimensions.

Time steps: In our current experiments, we conditioned on 3 frames and trained to predict 3 frames in the future. However, considering that not much changes in the span of 3-6 frames, it might be prudent to model farther changes, such as those in 10 frames. Since the Neural ODE operates in a continuous time setting, it is also possible to skip frames that mostly contain the same information. Instead of training on contiguous frames, we can choose to use only the "key frames" in a video for predicting features on. This way, the features of all the frames that were skipped are also captured reasonably well within the transitions of the features that were used for training.

Stochasticity: An important aspect in the current version of Neural ODEs is that once the initial feature is fixed, prediction is a deterministic process. However, recently there have been a number of works [**61, 79, 77**] that induce stochasticity in Neural ODEs. In the case of generative modeling, stochasticity might prove to be an important component in predicting multiple futures given the same past. However, in such a setting, the loss formulation needs to be carefully designed so that the one future of the ground truth does not limit the training of potential diverse futures.

5.3.2. Improve video predictions

In the experiments that we performed, we only used i-RevNet [**60**] as the invertible model that maps from features to image (and back). We can see from Fig. 7 that the i-RevNet is ill-equipped to deal with this situation. Hence, it may be prudent to either train the i-RevNet with data augmentation from both ends, or to replace the i-RevNet entirely with either another reversible decoder, or a separate decoder from features to frames.

Image optimization while fixing features: Some visualization methods [**36, 115, 81, 87, 132, 32**] fix the feature and optimize the image input to the neural network to minimize the loss between the produced feature and the fixed feature. We propose to adopt a similar approach, where we use a fixed encoder model (YOLO in this case) to optimize the input image to match the feature predicted by the Neural ODE. Recently, [**85**] provide a novel way of exploring the real image manifold that when downsampled produces the provided ground truth image, i.e. on the task of super-resolution. Instead, this method could be adopted to the task of image generation from a feature.

Residual flows: Recently, Residual Flows [**16**] were introduced that build on top of iResNets [**6**], which are capable of making an invertible model out of a generic Residual Network [**48**] as long as its Lipschitz constant is constrained. It is to be seen how these flexible architectures fare in the current situation.

End-to-end training: In the previous experiments, we trained the Neural ODE and the i-RevNet independently of each other. The Neural ODE was trained to predict future YOLO features based on past YOLO features. The i-RevNet was trained to map between YOLO features and video frames. It is to be seen how the Neural ODE and i-RevNet perform if they are trained jointly in an end-to-end manner. The loss function would then simply be on the reconstruction of video frames by the i-RevNet from the predicted YOLO features, as depicted in Fig. 6 with weighted "pixel loss" and "feature loss".

Semantic segmentation: We would also like to explore whether features useful for more complex tasks such as semantic segmentation provide better features in time than the one we used, i.e. object detection. Specifically, we propose to use a semantic segmentation network that follows U-Net structure [**107, 54**]: one that encodes input data into a bottleneck feature, and then decodes it by repeatedly combining with features from the encoder brought using skip connections. There are several models that use this architecture

for semantic segmentation [**22, 13, 129, 14**]. We then propose to model the temporally changing bottleneck features from this U-Net architecture in a video, using a Neural ODE. The predicted bottleneck features could then be decoded into the semantic segmentation map. Then, a model such as Vid2Vid [**126**] could be used to convert the predicted semantic segmentation maps back into video.

Cycle consistency: Among adversarial models, Pix2Pix [**59**] is a popular model that is used for paired image-to-image translation, while CycleGAN [**137**] is used for unpaired translation. Both these methods uses a cycle-consistency loss to control both transformations. We propose to use pix2pix or CycleGAN to map from video to features and back (since in this case we have paired features). If this is used in the end-to-end pipeline, this could help mitigate the effects of difference in data distribution of the real features and predicted features.

5.4. Optical Flow estimation using Neural ODEs

Another direction we would like to pursue following this work is to explore how Neural ODEs can be used to estimate optical flow. Whereas in the earlier sections we dealt with modeling object dynamics using features learnt from images, here we would like to model the transition of pixels in a scene.

An ordinary differential equation describes the change in a variable across, say, time using a function of the variable. Let us assume that the ODE Solver is the simple Euler integration, and the number of time steps is 1. The state variable of the Neural ODE at the initial time step t_0 is $\mathbf{x}(t_0)$. The numerical integration of $\mathbf{x}_0$ for one time step using the Euler method is:

$$\mathbf{x}(t_1) = \mathbf{x}(t_0) + (t_1 - t_0)\ f(\mathbf{x}(t_0), t_0) \tag{5.1}$$

where f is the function of the differential, parameterized in our case by a neural network.

In our case, the state variable is the image itself. The change in the pixel values from one image to the next in a video can be modeled as a translation of those pixel values in space, called "optical flow". Hence, we propose that the optical flow in a video could be modelled using a Neural ODE.

Hence, equation 5.1 can be modified by replacing the addition operation with an image warping from the initial frame $x(t_0)$ to the frame at the next time step $x(t_1)$:

$$\mathbf{x}(t_1) = \text{Warp}(\mathbf{x}(t_0), f(\mathbf{x}(t_0), t_0)) \tag{5.2}$$

The "Warp" function warps the current image using the predicted optical flow, by defining an affine transformation from $\mathbf{x}(t_0)$ to $\mathbf{x}(t_1)$:

$$\begin{cases} u, v = f(\mathbf{x}(t_0), t_0) \\ \mathbf{x}(t_1) = x(t_1, i, j) = x(t_0, i + u, j + v) = \text{Warp}(\mathbf{x}(t_0), u, v) \end{cases} \tag{5.3}$$

This new ODE Solver can now be used to predict frames in a video, and a reconstruction loss on the frames could be used to train the Neural ODE.

Moreover, the Neural ODE could integrate using multiple steps between two video frames, while the loss could be calculated only on the ground truth frames of the video. This would enable the trained model to produce the optical flow at fractional times steps, which could be used for upsampling of videos.

Optical flow estimation is a challenging problem to tackle [**55, 5, 3, 127, 33, 134, 58, 19, 100, 57, 4, 62**]. We believe this novel approach could prove to be a significant contribution to the field.

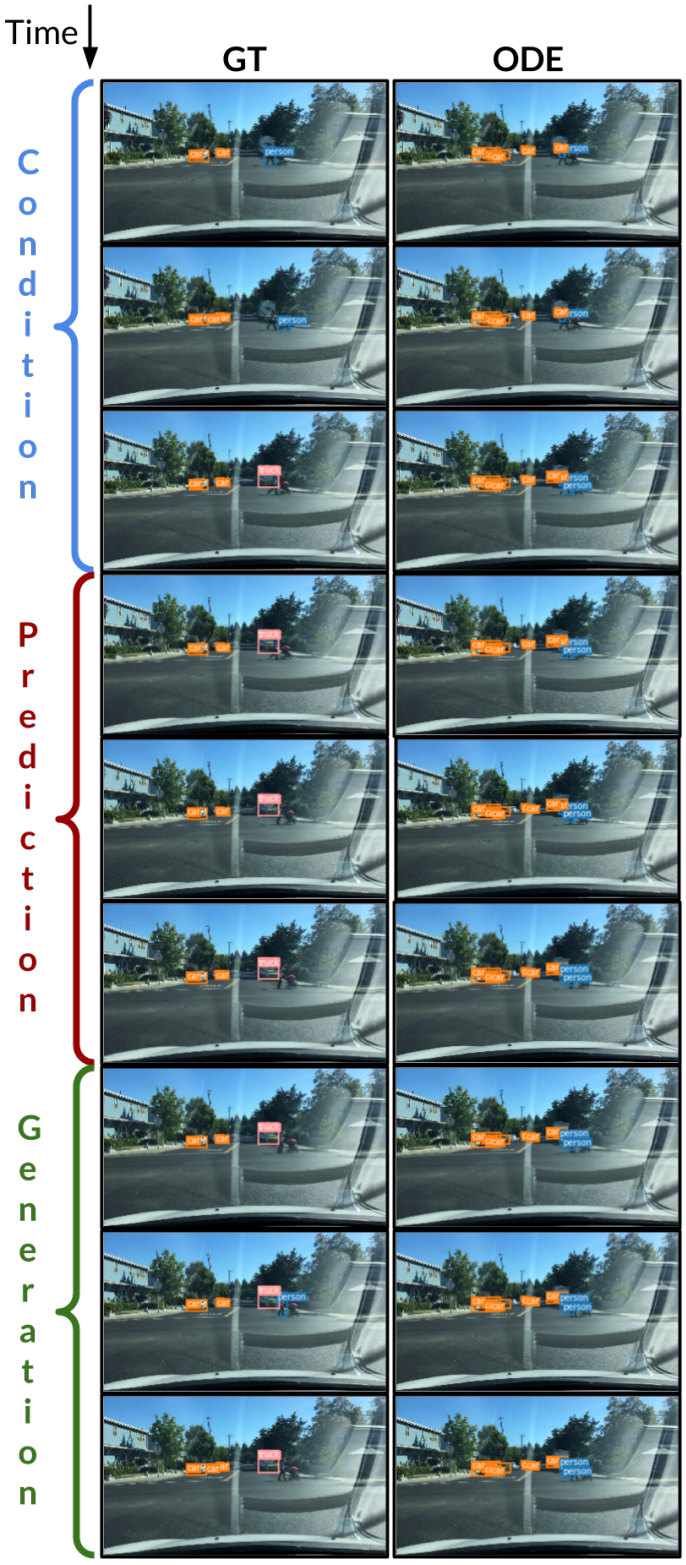


Figure 5. Results of using Neural ODE to predict YOLO features

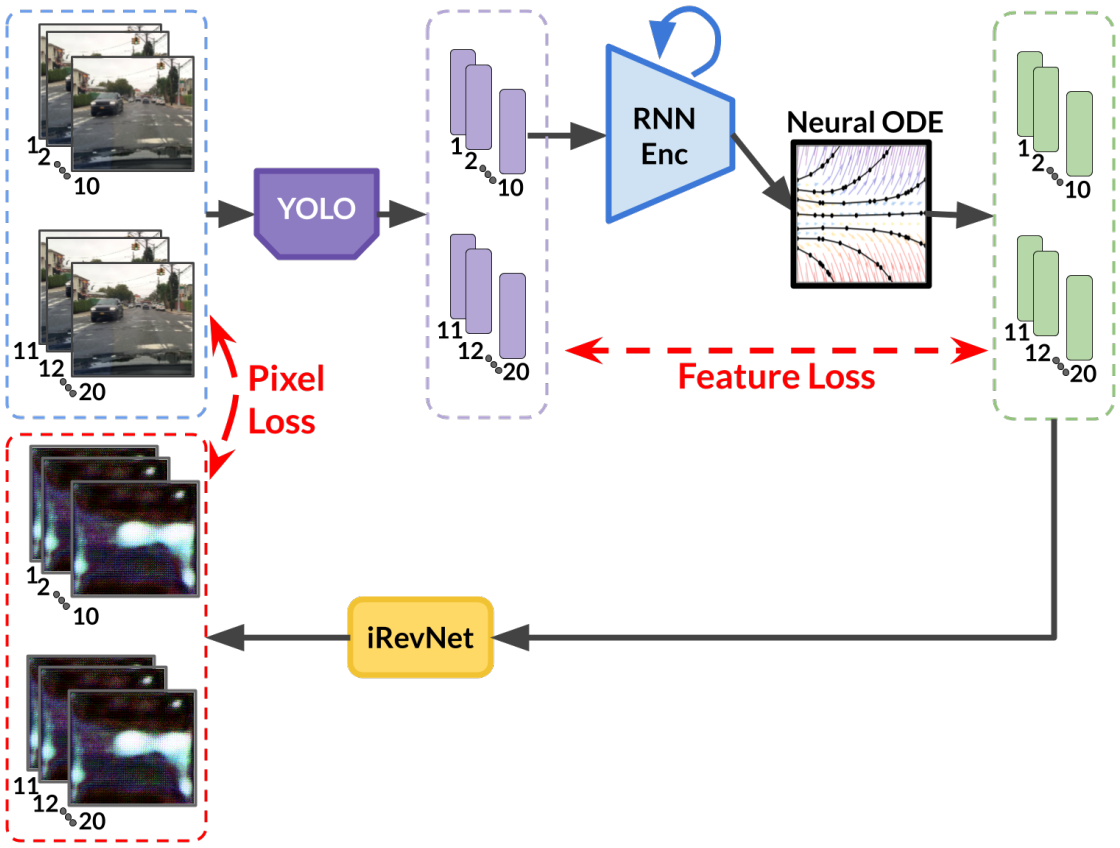


Figure 6. Model of ODE for YOLO with i-RevNet for video generation

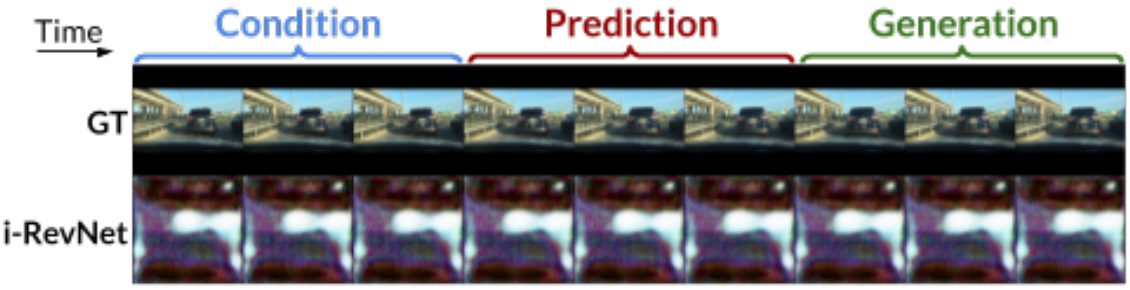


Figure 7. Predictions of i-RevNet for video generation

Chapter 6

Multi-Scale Continuous Normalizing Flow (MSFlow)

In this chapter, we introduce a novel method of generating images using Continuous Normalizing Flows. Previous methods [**43, 37, 40, 46**] involved training a flow between latent noise and data samples, and then introducing a relevant regularization term to speed up training. Our method introduces the notion of generation at multiple scales in the context of Continuous Normalizing Flows. Our method achieves state-of-the-art likelihood values in the least time compared to all existing normalizing flow-based methods.

6.1. Method

Suppose we are given a set of images of the *same* scene at progressively finer scales (or resolutions) $\mathbf{x} := \{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_S\}$. This is sometimes called an image pyramid. Here $\mathbf{x}_0$ is the image at the coarsest scale, and $\mathbf{x}_S$ is at the finest scale.

Each image can be upsampled to finer scales using an interpolation (upsampling) function $\mathcal{U} : \mathbb{R}^{d\times d} \mapsto \mathbb{R}^{2d\times 2d}$, and downsampled to a coarser scale using a restriction (downsampling) function $\mathcal{D} : \mathbb{R}^{2d\times 2d} \mapsto \mathbb{R}^{d\times d}$. It is common to upsample using a combination of oversampling and a Gaussian blur, and to downsample using a combination of a Gaussian blur and subsampling.

Suppose we are given a sample of N image pyramids $\{\mathbf{x}^{(i)}\}$, $i = 1, \ldots, N$. We want to perform density estimation on this set: find a probability distribution $p(\mathbf{x})$ from which the samples are drawn. We will model the scales as a Bayesian network, in that we will assume

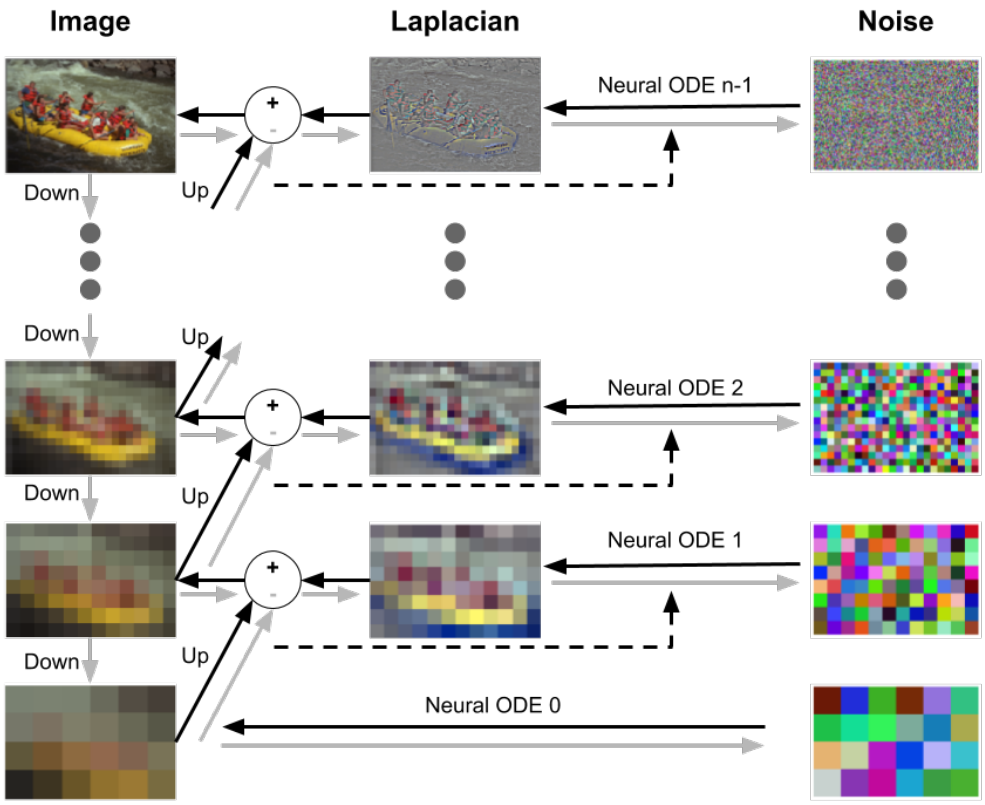


Figure 8. Multi-Scale Continuous Normalizing Flow

coarser scales do not depend on finer scales. Moreover we will assume that one scale only directly depends on its immediate coarser scale.

$$p(\mathbf{x}_S) = p(\mathbf{x}_0) \prod_{s=1}^{S} p(\mathbf{x}_s \mid \mathbf{x}_{s-1}) \tag{6.1}$$

so that the log-density is:

$$\log p(\mathbf{x}_S) = \log p(\mathbf{x}_0) + \sum_{s=1}^{S} \log p(\mathbf{x}_s \mid \mathbf{x}_{s-1}) \tag{6.2}$$

We will estimate these log-probabilities using continuous normalizing flows, i.e. Neural ODEs. Let f_s be the Neural ODE at scale s. We map the Laplacian of the image at each scale to Gaussian noise using f_s, i.e. $f_s : Lap(\mathbf{x}_s) \mapsto \mathbf{y}_s \mid \mathbf{x}_{s-1}$, where $\mathbf{y}_s \sim \mathcal{N}(\mathbf{y}_s; \mathbf{0}, \mathbf{I})$ ($\mathcal{N}$ is the probability density function of the standard normal distribution), and Lap is the laplacian operator:

$$Lap(\mathbf{x}_s) = \mathbf{x}_s - \mathcal{U}(\mathbf{x}_{s-1}) \tag{6.3}$$

Each Neural ODE f_s is trained such that the corresponding Laplacian is mapped to noise of the same dimensions as the Laplacian (or image) at that scale. For example, an image of size $x_S(32 \times 32)$ is downsampled twice to make 3 scales of the same image: $x_0(8 \times 8), x_1(16 \times 16), x_2(32 \times 32)$. The Laplacian at each scale is mapped to noise of the same shape: $y_0(8 \times 8), y_1(16 \times 16), y_2(32 \times 32)$.

$$\begin{cases} \mathbf{y}_0 = f_0(\mathbf{x}_0) \\ \mathbf{y}_s = f_s(Lap(\mathbf{x}_s) \mid \mathbf{x}_{i-1}) = f_s(\mathbf{x}_s - \mathcal{U}(\mathbf{x}_{s-1}) \mid \mathbf{x}_{s-1}) \quad \forall\ s > 0 \end{cases} \tag{6.4}$$

The change-of-variables formula describes the change in probability due to a transformation of $\mathbf{x}$ to $\mathbf{y}$ as:

$$p(\mathbf{x}) = p(\mathbf{y}) + \log\det\left|\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}\mathbf{x}}\right| \tag{6.5}$$

Thus, the conditional log-probability at each scale can be expressed using the change-of-variables formula as:

$$\begin{cases} \log p(\mathbf{x}_0) = \log\mathcal{N}(\mathbf{y}_0; \mathbf{0}, \mathbf{I}) + \log\det\left|\frac{\mathrm{d}\mathbf{y}_0}{\mathrm{d}\mathbf{x}_0}\right| \\ \log p(\mathbf{x}_s|\mathbf{x}_{s-1}) = \log\mathcal{N}(\mathbf{y}_s; \mathbf{0}, \mathbf{I}) + \log\det\left|\frac{\mathrm{d}\mathbf{y}_s}{\mathrm{d}\mathbf{x}_s}\right| \quad \forall\ s > 0 \end{cases} \tag{6.6}$$

Each $\mathbf{y}_s$ is given by equation 6.4, hence:

$$\begin{cases} \log p(\mathbf{x}_0) = \log\mathcal{N}(f_0(\mathbf{x}_0); \mathbf{0}, \mathbf{I}) + \log\det\left|\frac{\mathrm{d}f_0(\mathbf{x}_0)}{\mathrm{d}\mathbf{x}_0}\right| \\ \log p(\mathbf{x}_s|\mathbf{x}_{s-1}) = \log\mathcal{N}(f_s(Lap(\mathbf{x}_s) \mid \mathbf{x}_{s-1}); \mathbf{0}, \mathbf{I}) + \log\det\left|\frac{\mathrm{d}f_s(Lap(\mathbf{x}_i) \mid \mathbf{x}_{s-1})}{\mathrm{d}\mathbf{x}_s}\right| \quad \forall\ s > 0 \end{cases} \tag{6.7}$$

Thus, each Neural ODE f_s is trained to maximize the log-probability of the image given by equation 6.2. Equivalently, each f_s is trained to minimize the bits-per-dim of the image at its scale. If D_s is the number of pixels at scale s,

$$\text{bpd}(\mathbf{x}_s) = \frac{-\log_2 p(\mathbf{x}_s)}{D_s} = \frac{-1}{D_s \log 2} \log p(\mathbf{x}_s) \tag{6.8}$$

$$= \frac{-1}{D_s \log 2} \left[\log p(\mathbf{x}_0) + \sum_{i=1}^{s} \log p(\mathbf{x}_i \mid \mathbf{x}_{i-1}) \right] \tag{6.9}$$

$$= \frac{-1}{D_s \log 2} \left[\left(\log \mathcal{N}(\mathbf{y}_0; \mathbf{0}, \mathbf{I}) + \log \det \left| \frac{\mathrm{d}\mathbf{y}_0}{\mathrm{d}\mathbf{x}_0} \right| \right) + \sum_{i=1}^{s} \left(\log \mathcal{N}(\mathbf{y}_i; \mathbf{0}, \mathbf{I}) + \log \det \left| \frac{\mathrm{d}\mathbf{y}_i}{\mathrm{d}\mathbf{x}_i} \right| \right) \right] \tag{6.10}$$

where each y_i is given by equation 6.4.

We call our method Multi-Scale Flow (MSFlow). To train the network, we can either:

(1) train all the f_s's jointly, in which case we minimize $\text{bpd}(\mathbf{x}_S)$, or

(2) train each f_s independently to minimize each $\text{bpd}(\mathbf{x}_s)$ step by step from the coarsest scale ($s = 0$) to the finest scale ($s = S$), having frozen $f_j : j \neq s$.

In addition, we provide the image in logit space to each Neural ODE f_s. We use equation 27 in [**93**] to compensate for the change from image space to logit space. If $\mathbf{x}_s$ is the logit image at scale s, and $\mathbf{z}_s$ is the corresponding image in [0,256] image space of D_s pixels, each corresponding pixel in the two spaces $z_s^{(p)}$ and $x_s^{(p)}$ are related by:

$$z_s^{(p)} = \frac{256}{1 - 2\lambda} \left(\sigma(x_s^{(p)}) - \lambda \right) \tag{6.11}$$

where $\sigma(.)$ is the sigmoid function. Hence,

$$p(z_s^{(p)}) = p(x_s^{(p)}) \left(\frac{\mathrm{d}z_s^{(p)}}{\mathrm{d}x_s^{(p)}} \right)^{-1} = \; p(x_s^{(p)}) \frac{1 - 2\lambda}{256} \left(\sigma(x_s^{(p)})(1 - \sigma(x_s^{(p)})) \right)^{-1} \tag{6.12}$$

$$\implies p(\mathbf{z}_s) = p(\mathbf{x}_s) \left(\frac{1 - 2\lambda}{256} \right)^{D_s} \left(\prod_p \sigma(x_s^{(p)})(1 - \sigma(x_s^{(p)})) \right)^{-1} \tag{6.13}$$

Hence, the bits-per-dim of $\mathbf{z}_s$ in [0,256] image space can be computed from $\mathbf{x}_s$ in logit space as:

$$\text{bpd}(\mathbf{z}_s) = \frac{-\log p(\mathbf{z}_s)}{D_s \log 2} \tag{6.14}$$

$$= \frac{-\log p(\mathbf{x}_s)}{D_s \log 2} - \log_2(1-2\lambda) + 8 + \frac{1}{D_s}\sum_p \Big(\log_2(\sigma(x_s^{(p)})) + \log_2(1-\sigma(x_s^{(p)}))\Big) \tag{6.15}$$

where $-\log p(\mathbf{x}_s)$ can be computed as in equation 6.8.

Assuming each f_s is invertible, then we may then generate images by sampling $\mathbf{y}_s$'s from the standard normal distributions, and inverting equation 6.4:

$$\begin{cases} \mathbf{x}_0 = f_0^{-1}(\mathbf{y}_0) \\ \mathbf{x}_s = f_s^{-1}(\mathbf{y}_s \mid \mathbf{x}_{s-1}) + \mathcal{U}(\mathbf{x}_{s-1}) \quad \forall\ s > 0 \end{cases} \tag{6.16}$$

Rather than computing each $\mathbf{x}_s$ during training, the Laplacians $Lap(\mathbf{x}_s)$ can be computed and stored off-line. This would speed up training time. If the pyramid is computed with Gaussian smoothing, then the variables define a *Laplacian pyramid*.

Although we used Neural ODEs [**15**] as f_s, it is possible to replace them with any mapping from data to noise.

For easier computation, [**15**] and [**43**] avoid the calculation of the log-determinant term in equation 6.4. Instead they use the instantaneous change of variables formula to replace the term with an integration of a trace operation. We follow the same steps.

In addition, [**37**] introduce two regularization terms to speed up the training of FFJORD models by stabilizing the learnt dynamics:

$$\mathcal{K}(\theta) = \int_{t_0}^{t_1} \|f(\mathbf{x}(t), t, \theta)\|_2^2 \, \mathrm{d}t \tag{6.17}$$

$$\mathcal{B}(\theta) = \int_{t_0}^{t_1} \|\epsilon^\top \nabla_z f(\mathbf{x}(t), t, \theta)\|_2^2 \, \mathrm{d}t, \quad \epsilon \sim \mathcal{N}(0, I) \tag{6.18}$$

Here, $\mathcal{K}(\theta)$ represents the kinetic energy of the flow learned by the Neural ODE, and $\mathcal{B}(\theta)$ represents its the Jacobian norm.

We build on top of the code provided in [**37**] [1].

	CIFAR 10			ImageNet64		
	BPD	Time	FID	BPD	Time	FID
RealNVP [**30**]	3.49	-	-	3.98	-	-
Glow [**71**]	3.35	-	-	3.81	-	-
FFJORD, ORIGINAL [**43**]	3.40	$\geq$ 5 days	-	-	-	-
Residual Flow [**16**]	3.28	-	-	3.76	-	-
FFJORD VANILLA + STEER [**40**]	3.40	86.34	-	-	-	-
FFJORD RNODE [**37**]	3.38	31.84	-	3.83*	64.1*	-
FFJORD RNODE + STEER [**40**]	3.397	22.24	-	-	-	-
Multi-scale FFJORD RNODE **(Ours)**	**3.28**	**13.97**	143.14	**3.48**	**44.8**	249.4

Table 1. Unconditional image generation metrics. *FFJORD RNODE [**37**] used 4 GPUs train on ImageNet64. Ours was trained on 1 GPU in all cases.

6.2. Results

Fig. 9 shows qualitative results on the CIFAR10 dataset (32×32). Fig. 11 shows qualitative results on the ImageNet64 dataset, i.e. the ImageNet dataset at resolution 64×64.

Table 1 lists previous flow-based methods, the Bits-per-dim (BPD) achieved by each of them on CIFAR10 and ImageNet64, the time taken to train the respective models to achieve the BPD reported, and the Fréchet Inception Distance (FID) [**49**] of the generated images. BPD is the average negative log likelihood of images under the model, divided by the number of pixels in the image. FID is a popular image quality metric used to test the realism of generated images for datasets with natural images.

It can be observed that by incorporating a multi-scale architecture, the time taken to train drastically decreases. It is to be noted that many of the flow-based models require multiple GPUs to train, and still take in the order of days to train on complex datasets. For example, FFJORD-RNODE [**37**] train on ImageNet64 using 4 GPUs. In contrast, we used only 1 GPU in all our experiments, and still gained significant improvement in the BPD value in each dataset. We achieved **state-of-the-art** BPD for *both* CIFAR10 and ImageNet64 in the *least* time.

1. https://github.com/cfinlay/ffjord-rnode

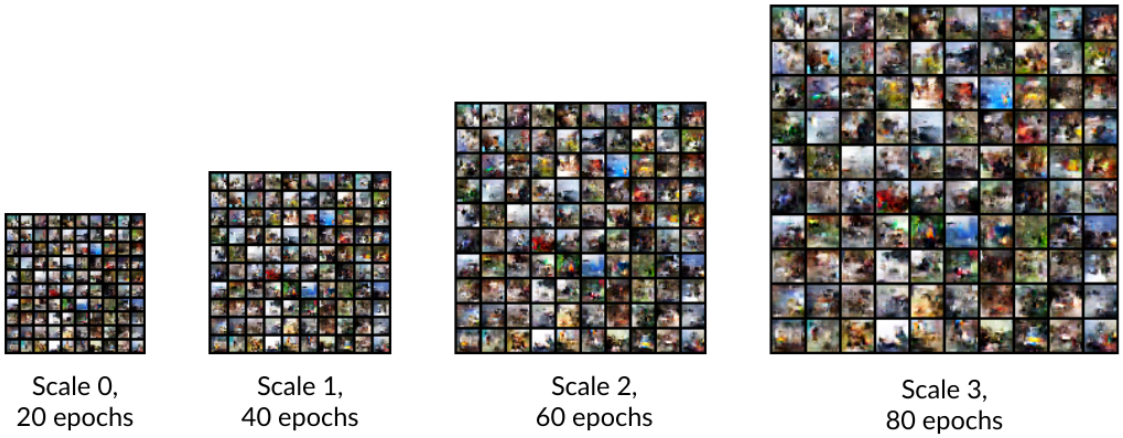
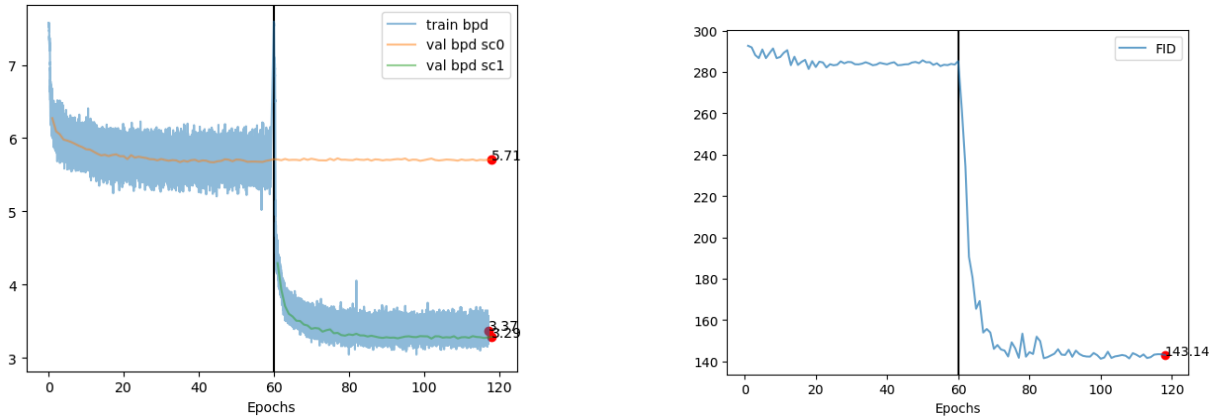


Figure 9. MSFlow results on unconditional CIFAR10 image generation



(a) CIFAR10 BPD

(b) CIFAR10 FID

Figure 10. BPD and FID while training MSFlow on CIFAR10

6.2.1. BPD vs FID

However, it can be verified from the qualitative results that a low BPD does not correspond to good generation of images. As can be seen from Fig. 10 and Fig. 12, a significant decrease in BPD does not correspond well with decrease in FID. Hence, FID seems to be a better indicator of image quality than BPD, although BPD i.e. log likelihood has been used to train the models.

It is to be noted that most of the previous Continuous Normalizing Flow-based publications [**43, 40**] *do not* provide generated images of models trained on ImageNet. Hence, it is possible that this dichotomy between BPD and FID is known in the research community but has not been tackled yet.

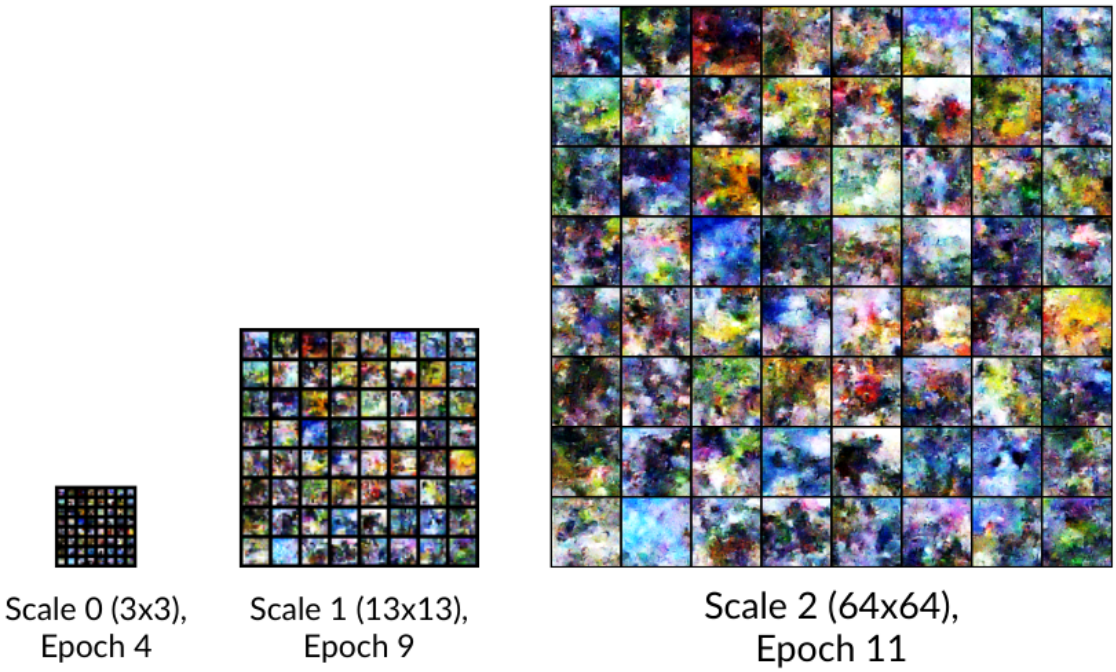
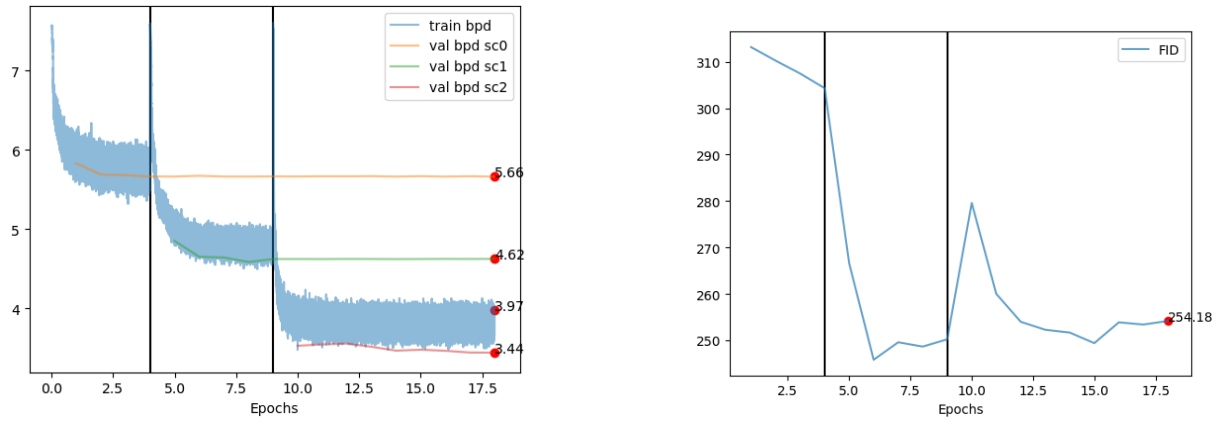


Figure 11. MSFlow results on unconditional ImageNet image generation



(a) ImageNet64 BPD

(b) ImageNet64 FID

Figure 12. BPD and FID while training MSFlow on ImageNet64

6.3. Conclusion

We propose a novel multi-scale version of the Continuous Normalizing Flow, that uses a Laplacian pyramid to make a generative model of images. We trained it on CIFAR10 and ImageNet64, and achieved state-of-the-art bits-per-dim in the least time among comparable methods.

6.4. Next steps

The final goal of this project is to be able to effectively use continuous normalizing flows to map between latent code and realistic images. Presently, our model is able to train on multiple image datasets by maximizing an accumulation of likelihoods from multiple scales.

However, there is some way to go before the generated images look more realistic, both qualitatively and quantitatively. As mentioned above, this is probably a known problem in the research community, as other papers that use a variant of FFJORD [**43**] do not show images or report any image quality-related metrics. Other recent likelihood-based models such as [**52**] report Inception Score [**110**] and FID [**49**] in addition to likelihood for their generated images. We propose the following to overcome these challenges, as well as take this forward in newer directions:

6.4.1. Neural ODE variants and regularization methods

In the recent past, variants [**47, 98, 40, 68**] of Neural ODEs have been proposed (apart from FFJORD-RNODE [**37**], which we used). TisODE [**47**] proposes to regularize the flow on perturbed data via the time-invariant property and the imposition of a steady-state constraint, and is shown to be adversarially robust. SNODE [**98**] proposes to formulate the parametric function in a Neural ODE as a truncated series of Legendre polynomials, thereby restricting the flow in higher order terms. STEER [**40**] proposes to regularize by randomly sampling the end time of the ODE during training. [**68**] propose to formulate the parametric function as a truncated Taylor series. We plan to incorporate these methods into our multi-scale architecture and perform an ablation study.

6.4.2. BPD vs FID

As seen above from Fig. 10 and Fig. 12, there is a dichotomy between BPD and FID in terms of visual quality of generated images. A lower BPD does not guarantee an image of good quality, as can be seen from the high FID values of the generated images. We plan on investigating this aspect. To confirm whether earlier flow-based models are able to achieve good FID values along with their BPD values, we plan to check the FID of the generated images for each of them. We also plan on setting FID as a potential loss to be minimized, in isolation of or in addition to BPD.

6.4.3. Adversarial loss

We also plan to incorporate an adversarial loss to better generate real images, as evidenced to work in earlier models [**82, 121, 124, 125, 76**]. As described in [**82**], adversarial loss helps to generate realistic images because it forces the generator to pick one of many real possible outcomes, whereas ℓ_2 loss, or in this case log likelihood, tends to promote the average of the possibilities. Hence, adding an adversarial loss in this setting might prove beneficial.

6.4.4. Joint training

So far, we have been training the Neural ODE at each scale independently, one after another from the lowest resolution to the highest. This way, we fix the generative models at all lower/coarser resolutions while training at any scale. Using this scheme, we were able to train at a decent resolution using 1 GPU only. However, this means that the likelihood of the final image is constrained by the trained likelihood of the images at the lower scales.

It is worth exploring the other option of jointly training all scales using the likelihood of the image at the final resolution. In our experiments with 1 GPU, we found that this caused Out-of-Memory issues due to the size and number of scales used. Hence, using multiple GPUs we could instead train it on all scales jointly. Perhaps this would lead to improvement in the FID of the generated images, since the coarser scales have an additional guiding signal from the finer scales.

6.4.5. Multi-scale video generation

Once this model is capable of generation realistic-looking images for multiple datasets, it is worth looking at the problem of generating videos from this perspective. The spatial relationships in the content of each frame of a video would be handled by the multiple flows at different scales. The temporal dynamics between frames, as seen in Chapters 4 and 5 can also be modelled using a flow. Hence, it may be possible to then combine these two methods to make a video generation model that generates each frame using MSFlow, and also models the temporal relationship between frames as another flow at each scale.

The advantage of this method is that the entire video could be seen as an accumulation of likelihoods at multiple scales and in time. Hence, it could be a complete likelihood-based

video generation model. Typically, such models suffer from blurring artifacts, or need to be trained using a lower bound on the likelihood. However, the proposed model could potentially generate sharp images by incorporating an adversarial loss, and shall directly maximize the likelihood of the video.

Moreover, the model would inherently be able to generate the frames at fractional time steps as well for free, since the approximate integration in the temporal Neural ODE could be made to any real value. We believe this could be a significant contribution to the field of image and video generation.

Chapter 7

MiMIc: Minimization of Mutual Interaction

While the previous chapters describe work performed over several months with publications and potential submissions to conferences, the following is a description of an as yet unexplored idea. We plan on designing experiments to test out the idea in the near future.

Mutual Information [**24**] has been used in recent times as a measure of disentanglement among random variables, especially in the context of the features of a deep neural network. Several works [**18, 83, 50, 69, 17, 9, 89**] try to disentangle features by maximize the mutual information between the input and the learned features, typically at a bottleneck so that only the relevant information about the input is preserved. More recently, some works [**94, 112**] split the latent code into a component with shared information and another component with exclusive information with respect to different inputs, and then *minimize* the mutual information among the two groups of latents. This results in the two groups being disentangled.

In the context of deep neural networks, we propose to learn disentangled representations by minimizing the mutual information among all channels of a learned feature. Suppose a neural network is to be trained on a supervised task such as classification. By minimizing the mutual information among the channels of the feature at, say, the pre-final layer of this neural network, it could potentially lead to easier training for classification since the pre-final feature is already disentangled. If the channels of the feature at the first layer of a CNN are disentangled, it could potentially lead to more intuitive filters that look for independent aspects of the input image.

The estimation of mutual information can be carried out using another neural network [**7, 51, 92**]. In particular, MINE [**7**] estimates the mutual information between two random variables X and Z using the following formula:

$$I_\Theta(X, Z) = \sup_{\theta \in \Theta} \mathbb{E}_{\mathbb{P}_{XZ}}[T_\theta] - \log(\mathbb{E}_{\mathbb{P}_X \otimes \mathbb{P}_Z}[e^{T_\theta}]) \tag{7.1}$$

where T_θ represents a family of functions $T_\theta : \mathcal{X} \times \mathcal{Z} \to \mathbb{R}$ parameterized by a neural network. The first term on the right is an expectation over the joint, and the second term contains an expectation over the marginals. The estimate of the mutual information between X and Z is the supremum given in equation 7.1. Hence, to find the supremum among all T_θ, the neural network can be optimized by gradient ascent to maximize the expression in equation 7.1. Given a dataset of features, the expectation over the joint can be estimated using the empirical samples in the dataset, and the expectation over the marginal can be estimated by shuffling the samples in the batch dimension.

In our proposal, we wish to minimize the mutual information among all the channels of a feature, which are typically more than 2 in number. The mutual information among multiple variables is called "mutual interaction" [**84**]. To estimate the mutual interaction among the channels of a feature, the same formulation as in equation 7.1 can be used.

Hence, given some features from a neural network, the mutual interaction among the channels of the features can be estimated by *maximizing* a quantity computed from the output of an auxiliary neural network using equation 7.1.

However, our objective is to *minimize* the mutual interaction among the channels of a feature of the original neural network N_ϕ parameterized by ϕ. During training, the feature would constantly evolve according to the training paradigm of the supervised task of N_ϕ. To incorporate disentanglement, a penalty from the auxiliary neural network T_θ corresponding to its estimate of the mutual interaction can be added to the loss function of the original neural network $\mathcal{L}(\mathcal{N}_\phi)$, and the two neural networks could be trained simultaneously. Hence, the original network would try to minimize the quantity on the RHS of equation 7.1, while the auxiliary neural network would try to maximize it. The overall objective function can be expressed as:

$$\min_{N_\phi} \max_{T_\theta} \mathcal{L}(\mathcal{N}_\phi) + \mathbb{E}_{\mathbb{P}_{X_1 X_2 \dots X_f}}[T_\theta] - \log(\mathbb{E}_{\mathbb{P}_{X_1} \otimes \mathbb{P}_{X_2} \otimes \dots \otimes \mathbb{P}_{X_f}}[e^{T_\theta}]) \tag{7.2}$$

Some of the features of this formulation are:

(1) **Plug-in MiMIc**: Such auxiliary classifiers could be plugged in at any layer to disentangle the features of that layer. Indeed, auxiliary network could be attached to multiple layers to try to disentangle multiple features, especially of a deep neural network. An ablation study could be performed to check the effect on disentanglement and on the supervised task, of one auxiliary network at different locations, as well as multiple auxiliary networks.

(2) **GANs**: The min-max term in the loss formulation in equation 7.2 seems similar to the adversarial loss formulation in GANs [**42**]. Hence the wide literature of training GANs could extend to training for minimizing mutual interaction.

(3) **Disentanglement**: Various metrics exist in literature [**41, 44, 65, 50, 69, 17, 35**] to check the quality of disentanglement. An ablation study could be conducted whereby the quality of disentanglement in the respective features with and without minimizing mutual interaction are recorded.

(4) **Transfer learning**: More importantly, disentangled features could help learn domain-independent characteristics that transfer to other datasets. Hence, an ablation study could be performed on checking how, say, a Residual Network trained on ImageNet transfers to other image datasets for classification. A semantic segmentation network such as DeepLabV3+ [**14**] could be trained on the images of, say, the CityScapes [**23**] dataset while minimizing the mutual interaction at the bottleneck layer, and then tested on the Berkeley Driving Dataset [**133**].

This idea is at a nascent stage at this point. Experiments need to be carefully designed to conduct each ablation study thoroughly. We believe this idea has the potential to evolve into an easy plug-in to the training of neural networks, and would like to pursue it in the near future.

Chapter 8

Conclusion

The problems of video prediction and image generation are complex and challenging. In this work, we propose to tackle them using a recently introduced dynamical model called Neural ODE (described in Chapter 3), and a variant of it which is used to map between image data and latent noise called Continuous Normalizing Flows (described in Section 3.2).

Many of the previous models for video prediction (elaborated upon in Chapter 2) use an encoder-decoder structure to encode conditional frames and predict the latent variables at future time steps using RNNs. In Chapter 4, we introduced a new method of video prediction by replacing the RNN in the latent space with a Neural ODE. We first tested whether a Neural ODE is able to model the trajectory of latent variables in a video. We showed that the Neural ODE is able to learn the dynamics of multiple objects in a scene independently, and visualized the results on a simple video dataset.

In Chapter 5, we scaled it up to test whether a Neural ODE is able to model the trajectory of high-dimensional pre-trained features that embed important semantic information in a natural scene. We used the popular object detection model YOLO to encode the frames into feature space. Then, we used the Neural ODE architecture described in Chapter 4 to model the trajectory of YOLO features of a video. We verified that the Neural ODE is able to predict YOLO features to a reasonable degree, and tested the quality of predicted object detections qualitatively and quantitatively. We identified the shortcomings in the current state, and proposed various steps to take this forward.

We then tested whether the predicted high-dimensional features could be used to generate video frames. To help with the overall pipeline, an invertible neural network called i-RevNet

was trained to map between the pixel space of video frames and the latent space of YOLO features. We found that the discrepancy between the actual features and the predictions of the Neural ODE were enough to make the i-RevNet fail to predict legible frames. We then listed potential steps to correct this problem.

Along similar lines, we also proposed a new way of handling video prediction, by modeling the optical flow between frames using a Neural ODE. A variant of the Euler's method for numerical integration was formulated that replaced the approximate integration operation with an affine transformation of the input image using the optical flow prediction by the neural network in the Neural ODE. Experiments need to be designed in this direction to test how this method works on different videos.

Many previous methods of image generation (Chapter 2) take advantage of a multi-scale framework to generate images from coarser to finer resolutions. In Chapter 6, we introduced a novel likelihood-based method of unconditional image generation, by incorporating a multi-scale architecture on Continuous Normalizing Flows. We compared the log likelihood of images using our model with previous methods, and found that our method achieved state-of-the-art likelihood in the least amount of training time.

We identified that the likelihood values did not necessarily correspond with image quality, and proposed various steps to improve the realism of generated images. We also proposed a large-scale idea of likelihood-based video generation by combining the multi-scale image generation framework with the video prediction pipeline in the previous chapters.

Finally, in Chapter 7 we introduced a nascent idea of a new plug-in to neural networks that can improve disentanglement in intermediate features by minimizing the mutual interaction among the channels of the features. We described how the existing methods of estimating mutual information using neural networks can be used in this direction. Various experiments have been proposed to delve into the details and potential applications of this method.

As is evident, there are several immediate steps as well as long-term directions that have been identified for each of the three projects discussed: (1) video prediction, (2) image generation, (3) mutual interaction minimization. We aim to complete each of these threads by exploring the best ways of achieving the respective goals in a thorough and scientific way, and potentially publish quality papers at respectable conferences along the way.

References

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[2] Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H. Campbell, and Sergey Levine. Stochastic variational video prediction. *International Conference on Learning Representations*, 2018.

[3] Simon Baker, Daniel Scharstein, JP Lewis, Stefan Roth, Michael J Black, and Richard Szeliski. A database and evaluation methodology for optical flow. *International journal of computer vision*, 92(1):1–31, 2011.

[4] Aviram Bar-Haim and Lior Wolf. Scopeflow: Dynamic scene scoping for optical flow. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7998–8007, 2020.

[5] Steven S. Beauchemin and John L. Barron. The computation of optical flow. *ACM computing surveys (CSUR)*, 27(3):433–466, 1995.

[6] Jens Behrmann, Will Grathwohl, Ricky TQ Chen, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks. In *International Conference on Machine Learning*, pages 573–582, 2019.

[7] Mohamed Ishmael Belghazi, Aristide Baratin, Sai Rajeshwar, Sherjil Ozair, Yoshua Bengio, Aaron Courville, and Devon Hjelm. Mutual information neural estimation. In *International Conference on Machine Learning*, pages 531–540, 2018.

[8] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *International Conference on Learning Representations (ICLR)*, 2019.

[9] Christopher P Burgess, Irina Higgins, Arka Pal, Loic Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding disentangling in β-vae. *arXiv preprint arXiv:1804.03599*, 2018.

[10] Peter Burt and Edward Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on communications*, 31(4):532–540, 1983.

[11] John Charles Butcher. *Numerical methods for ordinary differential equations*. John Wiley & Sons, 2016.

[12] Lluis Castrejon, Nicolas Ballas, and Aaron Courville. Improved conditional vrnns for video prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 7608–7617, 2019.

[13] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[14] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818, 2018.

[15] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 2018.

[16] Ricky TQ Chen, Jens Behrmann, David K Duvenaud, and Jörn-Henrik Jacobsen. Residual flows for invertible generative modeling. In *Advances in Neural Information Processing Systems*, pages 9916–9926, 2019.

[17] Ricky TQ Chen, Xuechen Li, Roger B Grosse, and David K Duvenaud. Isolating sources of disentanglement in variational autoencoders. In *Advances in Neural Information Processing Systems*, pages 2610–2620, 2018.

[18] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180, 2016.

[19] Jingchun Cheng, Yi-Hsuan Tsai, Shengjin Wang, and Ming-Hsuan Yang. Segflow: Joint learning for video object segmentation and optical flow. In *Proceedings of the IEEE international conference on computer vision*, pages 686–695, 2017.

[20] Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In *Advances in neural information processing systems*, pages 2980–2988, 2015.

[21] Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C. Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. *Neural Information Processing Systems*, 2015.

[22] Özgün Çiçek, Ahmed Abdulkadir, Soeren S Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d u-net: learning dense volumetric segmentation from sparse annotation. In *International conference on medical image computing and computer-assisted intervention*, pages 424–432. Springer, 2016.

[23] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[24] T. M. Cover and J. A. Thomas. *Elements of information theory*. John Wiley & Sons, 1991.

[25] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. Ieee, 2009.

[26] Emily Denton and Vighnesh Birodkar. Unsupervised learning of disentangled representations from video. In *Advances in neural information processing systems*, pages 4414–4423, 2017.

[27] Emily Denton and Rob Fergus. Stochastic video generation with a learned prior. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1174–1183, 2018.

[28] Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in neural information processing systems*, pages 1486–1494, 2015.

[29] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. In *International Conference on Learned Representations Workshop*, 2015.

[30] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. In *International Conference on Learned Representations*, 2017.

[31] Laurent Dinh, Jascha Sohl-Dickstein, Razvan Pascanu, and Hugo Larochelle. A rad approach to deep mixture models. In *International Conference on Learned Representations Workshop*, 2019.

[32] Alexey Dosovitskiy and Thomas Brox. Inverting visual representations with convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4829–4837, 2016.

[33] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2758–2766, 2015.

[34] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes. In *Advances in Neural Information Processing Systems*, pages 3140–3150, 2019.

[35] Cian Eastwood and Christopher KI Williams. A framework for the quantitative evaluation of disentangled representations. In *International Conference on Learning Representations*, 2018.

[36] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.

[37] Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam M Oberman. How to train your neural ode: the world of jacobian and kinetic regularization. In *International Conference on Machine Learning*, 2020.

[38] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in neural information processing systems*, pages 64–72, 2016.

[39] Quan Gan, Qipeng Guo, Zheng Zhang, and Kyunghyun Cho. First step toward model-free, anonymous object tracking with recurrent neural networks. *arXiv preprint arXiv:1511.06425*, 2015.

[40] Arnab Ghosh, Harkirat Singh Behl, Emilien Dupont, Philip HS Torr, and Vinay Namboodiri. Steer: Simple temporal regularization for neural odes. *arXiv preprint arXiv:2006.10711*, 2020.

[41] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *International Conference on Machine Learning*, 2011.

[42] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[43] Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *International Conference on Learning Representations*, 2019.

[44] Will Grathwohl and Aaron Wilson. Disentangling space and time in video with hierarchical variational auto-encoders. *arXiv preprint arXiv:1612.04440*, 2016.

[45] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[46] E Hairer, SP Norsett, and G Wanner. Solving ordinary, differential equations i, nonstiff problems/e. hairer, sp norsett, g. wanner, with 135 figures, vol.: 1. Technical report, 2Ed. Springer-Verlag, 2000, 2000.

[47] YAN Hanshu, DU Jiawei, TAN Vincent, and FENG Jiashi. On robustness of neural ordinary differential equations. In *International Conference on Learning Representations*, 2019.

[48] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[49] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in neural information processing systems*, pages 6626–6637, 2017.

[50] I Higgins, L Matthey, A Pal, C Burgess, X Glorot, M Botvinick, M. Botvinick, S. Mohamed, and A Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2017.

[51] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. In *International Conference on Machine Learning Representations*, 2019.

[52] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *arXiv preprint arXiv:2006.11239*, 2020.

[53] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[54] Sina Honari, Jason Yosinski, Pascal Vincent, and Christopher Pal. Recombinator networks: Learning coarse-to-fine feature aggregation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5743–5752, 2016.

[55] Berthold KP Horn and Brian G Schunck. Determining optical flow. In *Techniques and Applications of Image Understanding*, volume 281, pages 319–331. International Society for Optics and Photonics, 1981.

[56] Jun-Ting Hsieh, Bingbin Liu, De-An Huang, Li F Fei-Fei, and Juan Carlos Niebles. Learning to decompose and disentangle representations for video prediction. In *Advances in Neural Information Processing Systems*, pages 517–526, 2018.

[57] Tak-Wai Hui, Xiaoou Tang, and Chen Change Loy. Liteflownet: A lightweight convolutional neural network for optical flow estimation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8981–8989, 2018.

[58] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2462–2470, 2017.

[59] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[60] Jörn-Henrik Jacobsen, Arnold Smeulders, and Edouard Oyallon. i-revnet: Deep invertible networks. In *International Conference on Learning Representations*, 2018.

[61] Junteng Jia and Austin R Benson. Neural jump stochastic differential equations. In *Advances in Neural Information Processing Systems*, pages 9847–9858, 2019.

[62] Rico Jonschkowski, Austin Stone, Jonathan T Barron, Ariel Gordon, Kurt Konolige, and Anelia Angelova. What matters in unsupervised optical flow. In *European conference on computer vision*, 2020.

[63] Samira Ebrahimi Kahou, Vincent Michalski, Roland Memisevic, Christopher Pal, and Pascal Vincent. Ratm: recurrent attentive tracking model. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1613–1622. IEEE, 2017.

[64] Nal Kalchbrenner, Aäron Oord, Karen Simonyan, Ivo Danihelka, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu. Video pixel networks. In *International Conference on Machine Learning*, pages 1771–1779, 2017.

[65] Theofanis Karaletsos, Serge Belongie, and Gunnar Rätsch. Bayesian representation learning with oracle constraints. In *International Conference on Learning Representations*, 2016.

[66] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *International Conference on Learned Representations*, 2018.

[67] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4401–4410, 2019.

[68] Jacob Kelly, Jesse Bettencourt, Matthew James Johnson, and David Duvenaud. Learning differential equations that are easy to solve. *arXiv preprint arXiv:2007.04504*, 2020.

[69] Hyunjik Kim and Andriy Mnih. Disentangling by factorising. *arXiv preprint arXiv:1802.05983*, 2018.

[70] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014.

[71] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in neural information processing systems*, pages 10215–10224, 2018.

[72] Adam Kosiorek, Alex Bewley, and Ingmar Posner. Hierarchical attentive recurrent tracking. In *Advances in neural information processing systems*, pages 3053–3061, 2017.

[73] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *University of Toronto*, 2009.

[74] Wilhelm Kutta. Beitrag zur naherungsweisen integration totaler differentialgleichungen. *Z. Math. Phys.*, 46:435–453, 1901.

[75] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[76] Alex X. Lee, Richard Zhang, Frederik Ebert, Pieter Abbeel, Chelsea Finn, and Sergey Levine. Stochastic adversarial video prediction. *arXiv preprint arXiv:1804.01523*, 2018.

[77] Xuechen Li, Ting-Kam Leonard Wong, Ricky TQ Chen, and David Duvenaud. Scalable gradients for stochastic differential equations. *International Conference on Artificial Intelligence and Statistics*, 2020.

[78] Xiaodan Liang, Lisa Lee, Wei Dai, and Eric P Xing. Dual motion gan for future-flow embedded video prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1744–1752, 2017.

[79] Xuanqing Liu, Tesi Xiao, Si Si, Qin Cao, Sanjiv Kumar, and Cho-Jui Hsieh. Neural sde: Stabilizing neural ode networks with stochastic noise. *arXiv preprint arXiv:1906.02355*, 2019.

[80] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[81] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5188–5196, 2015.

[82] Michael Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. In *International Conference on Learning Representations*, 2016.

[83] Loic Matthey, Irina Higgins, Demis Hassabis, and Alexander Lerchner. dsprites: Disentanglement testing sprites dataset. *URL https://github. com/deepmind/dsprites-dataset/.[Accessed on: 2018-05-08]*, 2017.

[84] William McGill. Multivariate information transmission. *Transactions of the IRE Professional Group on Information Theory*, 4(4):93–111, 1954.

[85] Sachit Menon, Alexandru Damian, Shijia Hu, Nikhil Ravi, and Cynthia Rudin. Pulse: Self-supervised photo upsampling via latent space exploration of generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2437–2445, 2020.

[86] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018.

[87] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going deeper into neural networks. *https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html*, 2015.

[88] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *European conference on computer vision*, pages 483–499. Springer, 2016.

[89] Weili Nie, Tero Karras, Animesh Garg, Shoubhik Debhath, Anjul Patney, Ankit B Patel, and Anima Anandkumar. Semi-supervised stylegan for disentanglement learning. In *International Conference on Machine Learning*, 2020.

[90] Guanghan Ning, Zhi Zhang, Chen Huang, Xiaobo Ren, Haohong Wang, Canhui Cai, and Zhihai He. Spatially supervised recurrent convolutional neural networks for visual object tracking. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4. IEEE, 2017.

[91] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *International Conference on Machine Learning*, 2016.

[92] Sherjil Ozair, Corey Lynch, Yoshua Bengio, Aaron Van den Oord, Sergey Levine, and Pierre Sermanet. Wasserstein dependency measure for representation learning. In *Advances in Neural Information Processing Systems*, pages 15604–15614, 2019.

[93] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. *Neural Information Processing Systems*, 2017.

[94] Xingchao Peng, Zijun Huang, Ximeng Sun, and Kate Saenko. Domain agnostic learning with disentangled representations. In *International Conference on Machine Learning*, 2019.

[95] LS Pontyagin, VG Boltyanskii, RV Gamkrelidze, and EF Mishchenko. The mathematical theory of optimal processes. *Interscience, NY*, 1962.

[96] Mr Prabhat, Evan Racah, Jim Biard, Yunjie Liu, Mayur Mudigonda, Karthik Kashinath, Christopher Beckham, Tegan Maharaj, Samira Kahou, Chris Pal, et al. Deep learning for extreme weather detection. In *AGU Fall Meeting Abstracts*, 2017.

[97] William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[98] Alessio Quaglino, Marco Gallieri, Jonathan Masci, and Jan Koutník. Snode: Spectral discretization of neural odes for system identification. In *International Conference on Learning Representations*, 2020.

[99] Alfio Quarteroni, Riccardo Sacco, Fausto Saleri, and P Gervasio. Matematica numerica 2a edizione, 2000.

[100] Anurag Ranjan and Michael J Black. Optical flow estimation using a spatial pyramid network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4161–4170, 2017.

[101] MarcAurelio Ranzato, Arthur Szlam, Joan Bruna, Michael Mathieu, Ronan Collobert, and Sumit Chopra. Video (language) modeling: a baseline for generative models of natural videos. *arXiv preprint arXiv:1412.6604*, 2014.

[102] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[103] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[104] Scott Reed, Aäron van den Oord, Nal Kalchbrenner, Sergio Gómez Colmenarejo, Ziyu Wang, Dan Belov, and Nando De Freitas. Parallel multiscale autoregressive density estimation. In *International Conference on Machine Learning*, 2017.

[105] Scott E Reed, Yi Zhang, Yuting Zhang, and Honglak Lee. Deep visual analogy-making. In *Advances in neural information processing systems*, pages 1252–1260, 2015.

[106] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

[107] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[108] Yulia Rubanova, Ricky TQ Chen, and David Duvenaud. Latent odes for irregularly-sampled time series. *Advances in neural information processing systems*, 2019.

[109] Masaki Saito, Eiichi Matsumoto, and Shunta Saito. Temporal generative adversarial nets with singular value clipping. In *Proceedings of the IEEE international conference on computer vision*, pages 2830–2839, 2017.

[110] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, 2016.

[111] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. In *International Conference on Learning Representations*, 2017.

[112] Eduardo Hugo Sanchez, Mathieu Serrurier, and Mathias Ortner. Learning disentangled representations via mutual information estimation. In *European conference on computer vision*, 2020.

[113] Tamar Rott Shaham, Tali Dekel, and Tomer Michaeli. Singan: Learning a generative model from a single natural image. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4570–4580, 2019.

[114] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems*, pages 802–810, 2015.

[115] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *International Conference on Machine Learning*, 2014.

[116] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using lstms. In *International conference on machine learning*, pages 843–852, 2015.

[117] Sergey Tulyakov, Ming-Yu Liu, Xiaodong Yang, and Jan Kautz. Mocogan: Decomposing motion and content for video generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1526–1535, 2018.

[118] Arash Vahdat and Jan Kautz. Nvae: A deep hierarchical variational autoencoder. *arXiv preprint arXiv:2007.03898*, 2020.

[119] Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in neural information processing systems*, pages 4790–4798, 2016.

[120] Ruben Villegas, Arkanath Pathak, Harini Kannan, Dumitru Erhan, Quoc V Le, and Honglak Lee. High fidelity video prediction with large stochastic recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 81–91, 2019.

[121] Ruben Villegas, Jimei Yang, Seunghoon Hong, Xunyu Lin, and Honglak Lee. Decomposing motion and content for natural video sequence prediction. In *International Conference on Learning Representations*, 2017.

[122] Ruben Villegas, Jimei Yang, Yuliang Zou, Sungryull Sohn, Xunyu Lin, and Honglak Lee. Learning to generate long-term future via hierarchical prediction. In *International Conference on Machine Learning*, 2017.

[123] Vikram Voleti, Samira Kanaa, Davidand Kahou, and Christopher Pal. Simple video generation using neural odes. In *Workshop on Learning with Rich Experience, Advances in Neural Information Processing Systems*, volume 32, 2019.

[124] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Generating videos with scene dynamics. In *Advances in neural information processing systems*, pages 613–621, 2016.

[125] Carl Vondrick and Antonio Torralba. Generating the future with adversarial transformers. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1020–1028, 2017.

[126] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. Video-to-video synthesis. *arXiv preprint arXiv:1808.06601*, 2018.

[127] Philippe Weinzaepfel, Jerome Revaud, Zaid Harchaoui, and Cordelia Schmid. Deepflow: Large displacement optical flow with deep matching. In *Proceedings of the IEEE international conference on computer vision*, pages 1385–1392, 2013.

[128] Nevan Wichers, Ruben Villegas, Dumitru Erhan, and Honglak Lee. Hierarchical long-term video prediction without supervision. *International Conference on Machine Learning*, 2018.

[129] Zbigniew Wojna, Vittorio Ferrari, Sergio Guadarrama, Nathan Silberman, Liang-Chieh Chen, Alireza Fathi, and Jasper Uijlings. The devil is in the decoder. In *British Machine Vision Conference 2017, BMVC 2017*, pages 1–13. BMVA Press, 2017.

[130] Tianfan Xue, Jiajun Wu, Katherine Bouman, and Bill Freeman. Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks. In *Advances in neural information processing systems*, pages 91–99, 2016.

[131] Cagatay Yildiz, Markus Heinonen, and Harri Lahdesmaki. Ode2vae: Deep generative second order odes with bayesian neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 13412–13421. Curran Associates, Inc., 2019.

[132] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. In *International Conference on Machine Learning Workshop*, 2015.

[133] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving dataset for heterogeneous multitask learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2636–2645, 2020.

[134] Jason J. Yu, Adam W. Harley, and Konstantinos G. Derpanis. Back to basics: Unsupervised learning of optical flow via brightness constancy and motion smoothness. In *Computer Vision - ECCV 2016 Workshops, Part 3*, 2016.

[135] Han Zhang, Xi Gao, Jacob Unterman, and Tom Arodz. Approximation capabilities of neural odes and invertible residual networks. In *International Conference on Machine Learning*, 2020.

[136] Tianjun Zhang, Zhewei Yao, Amir Gholami, Joseph E Gonzalez, Kurt Keutzer, Michael W Mahoney, and George Biros. Anodev2: A coupled neural ode framework. In *Advances in Neural Information Processing Systems*, pages 5151–5161, 2019.

[137] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *International Conference on Computer Vision (ICCV)*, 2017.