AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

KATEDRA TELEKOMUNIKACJI

PROJEKT INŻYNIERSKI

Porównanie Sieci Neuronowych ResNet i ODENet
Comparison of ResNet and ODENet neural network

Autor:	*Filip Gładzik*
Kierunek studiów:	*Elektronika i Telekomunikacja w j.ang*
Typ studiów:	*Stacjonarne*
Opiekun pracy:	*dr inż. Jarosław Bułat*

Kraków, 2019

Oświadczenie studenta

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Jednocześnie Uczelnia informuje, że zgodnie z art. 15a ww. ustawy o prawie autorskim i prawach pokrewnych Uczelni przysługuje pierwszeństwo w opublikowaniu pracy dyplomowej studenta. Jeżeli Uczelnia nie opublikowała pracy dyplomowej w terminie 6 miesięcy od dnia jej obrony, autor może ją opublikować, chyba że praca jest częścią utworu zbiorowego. Ponadto Uczelnia jako podmiot, o którym mowa w art. 7 ust. 1 pkt 1 ustawy z dnia 20 lipca 2018 r. —Prawo o szkolnictwie wższym i nauce (Dz. U. z 2018 r. poz. 1668 z późn. zm.), może korzystać bez wynagrodzenia i bez konieczności uzyskania zgody autora z utworu stworzonego przez studenta w wyniku wykonywania obowiązków związanych z odbywaniem studiów, udostępniać utwór ministrowi właściwemu do spraw szkolnictwa wyższego i nauki oraz korzystać z utworów znajdujących się w prowadzonych przez niego bazach danych, w celu sprawdzania z wykorzystaniem systemu antyplagiatowego. Minister właściwy do spraw szkolnictwa wyższego i nauki może korzystać z prac dyplomowych znajdujących się w prowadzonych przez niego bazach danych w zakresie niezbędnym do zapewnienia prawidłowego utrzymania i rozwoju tych baz oraz współpracujących z nimi systemów informatycznych.

Following work is dedicated to my familly and all good-willing people whom I met in my life

Contents

Introduction

The inspiration to the following work is the research paper "*Neural Ordinary Differential Equations*"[1] presented on the "32nd Conference on Neural Information Processing Systems" in Montreal, Canada. It has gained attention among many researchers and reinvigorated research on the subject of using differential calculus in neural networks.

This research paper introduces a new approach to the creation of deep neural network models. It is achieved by using well-established residual neural network models and treating them as elements of differential equations, what in turn allows optimizing them utilizing differential calculus. With the end effect of such operation being a working model of a new family of Ordinary Differential Equation Neural Network. Such an approach allows for greater flexibility in training models, substantially decreases the size of created neural network models and provides the possibility of creating arbitrarily deep models without specifying their depth. While their approach of using Ordinary Differential Equations in a neural network is not completely new, until previously mentioned work implemented *"adjoint sensitivity method"* for optimization of neural networks, there have not been any viable methods for optimizing the learning process of the differential equation-based neural networks.

The second chapter is dedicated to the overall basics of neural networks working principalities, with topics such as neurons learning process and combining them into a working example of shallow neural networks.

The third chapter will introduce a deep neural network family of Residual Neural Networks and answer a question: why they achieved such dominance as state-of-the-art models of Machine Learning.

The fourth chapter will present Ordinary Differential Equation Neural Networks, their structure, working principalities and try to explain previously mentioned *"adjoint sensitivity method"*.

The fifth chapter will describe the practical part of the implementation, the structure of used models, their training parameters, data processing methods and used datasets.

The sixth chapter will contain the results of experiments obtained by created implementation and their brief analysis.

Aim of work

The purpose of work is to discuss differences in structure and work principles of Neural Networks models from families of:

– Ordinary Differential Equation Neural Networks (ODENet).

– Residual Neural Networks (ResNet).

As well as the creation of an application which would allow for testing various models of Convolution Neural Networks from both families and comparing their results on various datasets such as MNIST, QMNIST, Fashion-MNIST, CIFAR10, and CIFAR100.

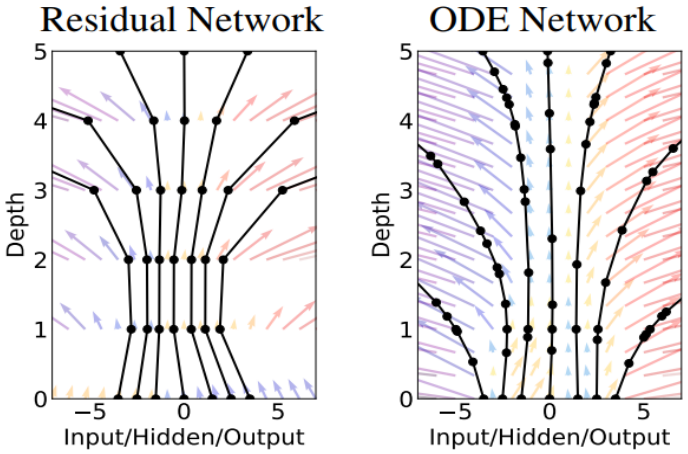

Figure 1. Both: Represent how a vector field gets evaluated by a neural network. Circles represent evaluation locations. Left: A Residual network defines a discrete sequence of finite transformations. Right: A ODE network defines a vector field, which continuously transforms the state. **Source:**[1]

1. Machine Learning

Machine Learning is an approach of problem-solving, which tries to solve problems by making algorithms "learn" method of solving a problem instead of explicitly programming them to do so. As a topic of research Machine Learning is intensively researched and implemented by companies, as with proper database and approach to creating "learning systems" many problems impossible to be solved by explicitly programmed algorithms, can be solved by implicit machine learning algorithms within a reasonable degree of error.

Among established methods in Machine Learning most notable are:

- Artificial neural networks
- Decision trees
- Support vector machines
- Bayesian networks
- Genetic algorithms

1.1. Artificial Neural Networks

Artificial Neural Networks, often referred to as *Universal Function Approximators* are one of Machine Learning most prominent branches, they owe their name to Biological Neural Networks after which they are inspired and modeled.

Neural Network consists of interconnected groups of neurons, which are created by using mathematical functions to emulate their biological counterparts.

1.2. Basic theory of Neural Networks

A typical neural network consists of blocks and/or layers, which can be presented as function F, which maps input vector x to output vector y. Output vector y can be presented in such case as F(x,θ), where θ is a weight vector for a given block

Artificial neuron structure can be presented as :

$$y = F(x, \theta) \tag{1.1}$$

$$y_{out} = \theta_t * x_{input} + b_t \tag{1.2}$$

where, x_{input} - input vector, θ_t - weight vector of neuron, b_t - bias vector of neuron

A neural network consisting of neurons presented in (1.2) would be simply a complex linear equation. Such a network would provide limited abilities to approximate more realistic non-linear functions. To improve the approximation capabilities of neural network functions called *"activation functions"* are introduced. Their purpose is to change the nature of groups of neurons from linear to non-linear equations. There are many popular activation functions, with most notable being: Sigmoid, ReLU, Leaky-ReLU and Swish. Each with unique advantages and drawbacks.

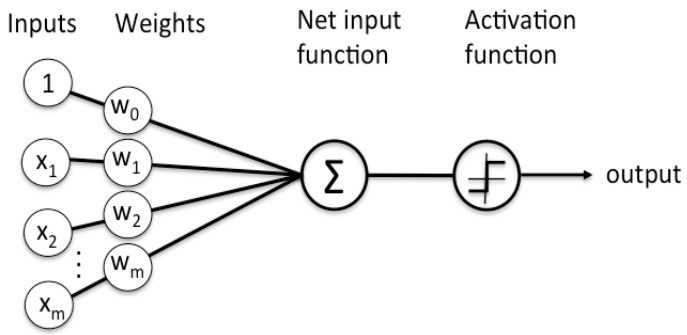

Figure 1.1. Neuron with activation function

Passing the input thought a layer of neurons combined with activation function, a good practice is to normalize output values before transferring them to the next neuron. The normalization goal is to decrease computational cost by reducing the range of output values as well as introduce some stochastic noise to the output[2].

1.2.1. Supervised learning process

A learning process for neural networks starts with calculating output vector Y for chosen neural network architecture. The output of the model is then compared with the target vector by calculating the loss value L, through loss function. Having metric measuring how far received output vector is from the desired target vector, the process of tweaking values of the weights θ and bias b vectors takes place. The tweaking process starts from the last layer, wherewith the help of the optimizer using gradient-based methods small value changes are back-propagated to previous layers. By completing such process throughout the whole network structure incremental steps towards approximating the desired function are made.

2. Residual Neural Networks family

When creating a neural network, creating deeper more complex models with more layers may seem like a good idea. It seems like the most obvious conclusion that more layers would provide significantly more parameters to tune, thus allowing the approximation of more complex function or approximating simple ones with higher precision. For a long time, exactly an such approach has been adopted, however, limitation at around 40 layers has been reached. After stacking that many layers adding more gives counter-productive results, and as it turns out they could not even reach results close to more fine-tuned models with a smaller amount of layers.

This problem has been solved mentioned problem by the introduction of so-called Residual Blocks, building blocks of Residual Neural Networks. Research paper "Deep Residual Learning for Image Recognition" [3] clearly show that "plain"(non-residual) network have limitation in its' depth, there are many possible reasons for this phenomenon; "vanishing" gradient problem, wrongly chosen optimization function, wrong initialization or simply too long path of propagation for information both in forward and backward passes through network.

We can simplify this to the fact that "deep" plain neural networks are extremely hard to learn and have problems with approximating simple functions, thus creating limitation in depth of the created model.

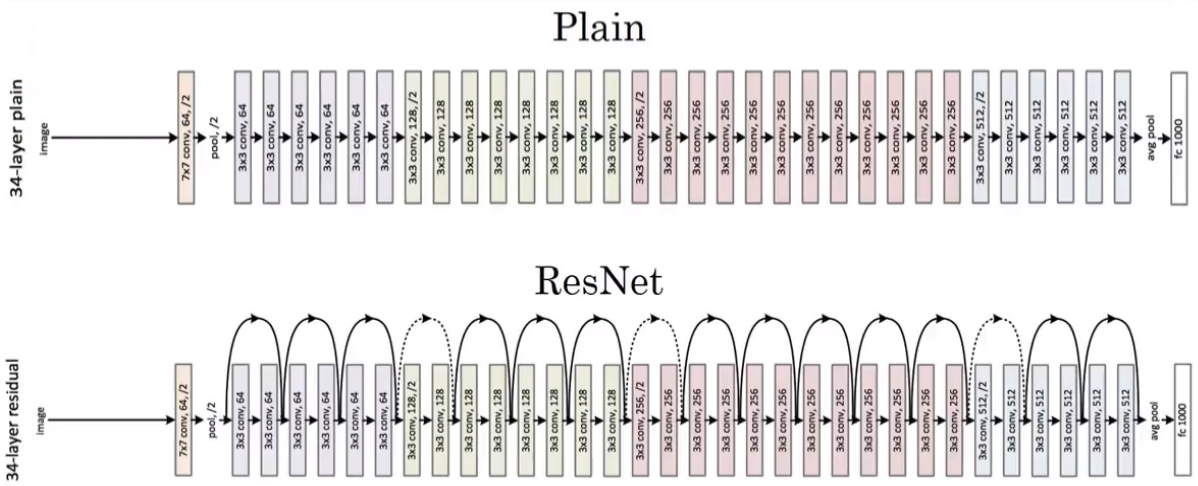

Figure 2.1. Top: 34 layer architecture of plain network. **Bottom:** 34 layer architecture of residual network. **Source:**[3]

2.1. Residual learning block

Residual neural network solves the limitation of depth in a very simple and elegant manner. The first layer of architecture building block forwards not only its' output into the next inner layer but also its' input is forwarded to the output of block via "skip connection". At the output of block, it is summed with the result of the last layer thus, creating so-called Residual Blocks, to which this architecture owes its' name.

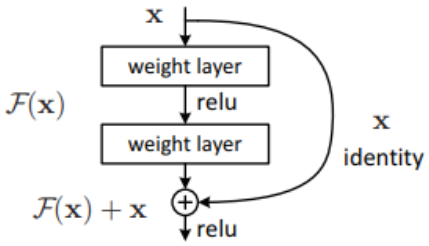

Figure 2.2. Residual block. **Source:** [3].

Such Residual Block can be represented as:

$$y_l = F(x_l, \theta_l) + h(x_l) \tag{2.1}$$

where:

x_l - is input vector of l-th block, θ_t - weight vectors for l-th residual block, F - being combination of weighted, activation and normalization layers, possibly multiple times [4].

In following work 2-layer residual blocks with h being simple identity mapping $h(x_l) = x_l$ are used. Thus, allowing us to simplify expression into the following form:

$$x_{l+1} = F(x_l, \theta_l) + x_l \tag{2.2}$$

In recursive form:

$$x_{L+1} = x_0 + \sum_{i=1}^{L} F(x_i, \theta_i) \tag{2.3}$$

In contrast to the form of plain neural network which we can expressed as:

$$x_{L+1} = \prod_{i=0}^{L} \theta_i x_0 \tag{2.4}$$

From the comparison of those two equations, we can observe that Residual Neural Networks deal with vanishing gradient problem by forwarding initial input as well as all intermediate results throughout the whole network. Thus increasing robustness and amount of information available to each block.

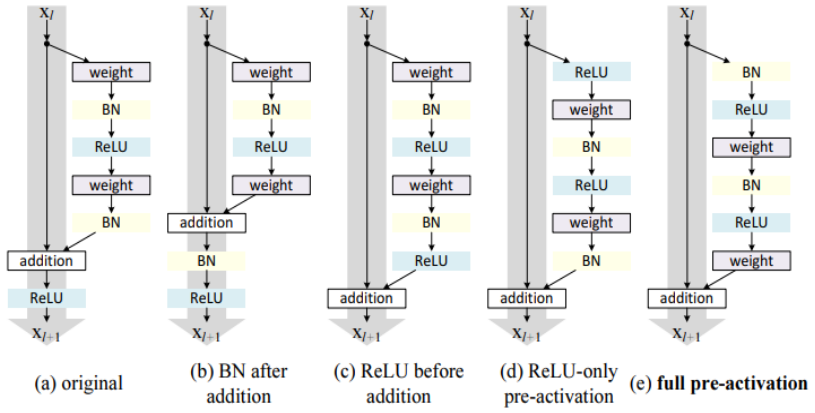

Figure 2.3. Examples of residual block variations proposed in [5]

2.2. ResNet architecture variations

Residual Neural Networks have many variations, they can have two or three layers in between of skip connections or those skip connections might be multiplied by some particular weight vector. Most notable ones being:

– ResNet - Skip connection is identity matrix [3]

– HighwayNet - Skip connection is multiplied by an additional weight matrix [6]

– DenseNets - Having several parallel skips [7]

– ResNext - Network having branching structure of multiple parallel layers with additional identity skip connection [8]

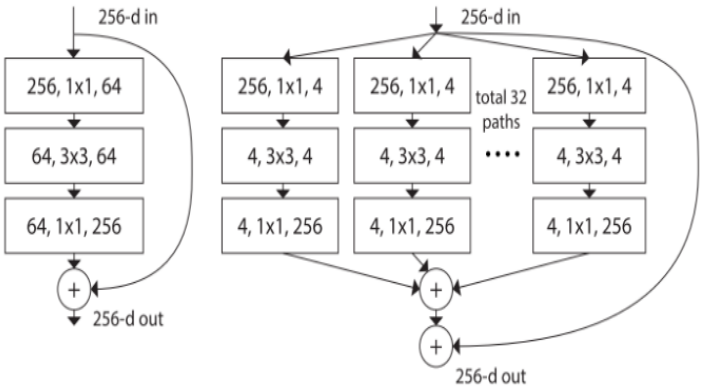

Figure 2.4. Left: A block of ResNet. **Right:** A block of ResNeXt. **Source:** [8]

3. Ordinary Differential Equation Neural Networks family

Early research into neural networks suggested continuous architecture based on biological neural networks as a natural progression. Due to the high complexity and unstable memory cost of optimizing continuous models, this approach has not been well researched.

The currently designing neural networks requires specifying the number of discrete layers at the initial stages of design. To change the nature of such a discrete neural network one of the layers is additionally parameterized with depth parameter t. The behavior of the whole network changes from explicitly deep discrete to infinitely deep, where the number of layers hidden inside the parameterized layer is determined by its solver. Treating it as black-box allows calculation of its' output using differential equation solver.

Such an approach has a few benefits:

- Constant size of created model
- Adaptive computation trade-off between time and precision
- Robustness to data fluctuations in time-series models

3.1. Similarities to Residual Neural Networks

Ordinary Differential Equation Neural Networks (ODENets) family can be understood as a continuous equivalent of discrete Residual Neural Networks (ResNets).

First of their similarities is the overall learning process. For ResNet, it requires first obtaining a mapping of x to output y by forward pass throughout the network. After calculating loss function for obtained y, changes to weight vector θ are back-propagated through the network so that next forward pass for the same input provides result closer to y_{true}. For ODENets learning, we first obtain a mapping of x to y by solving ODE with ODESolver. The next step is to calculate loss and adjust dynamics of the system such that the ODE transforms in a way that minimizes the loss, thus bringing output of network closer to y_{true} [9]

To understand their similarity closer look at the mathematics behind both families is required. If autonomous ODE is defined as:

$$\frac{\partial z}{\partial t} = f(z(t), \theta) \tag{3.1}$$

where $z(t = 0) = z_0$ and $z_1 = z(t = 1)$, where t is time.

Solution of equation (3.1) for z_1 can be expressed as:

$$z_1 = z_0 + \int_0^1 f(z_0, \theta)dt \tag{3.2}$$

Solving this equation (3.2) with the help of the Euler method for unit time step results in:

$$z_1 = z_0 + f(z_0, \theta) \tag{3.3}$$

For the current example we express ResNet equation (2.2) from previous chapter as:

$$z_1 = z_0 + f(z_0, \theta) \tag{3.4}$$

Looking at equations (3.3) and (3.4) it is possible to explain why ResNets and ODENets are compared together. After discretization of a forward pass of Ordinary Differential Equation based networks with the Euler method for unit time step, the result is a Residual neural network.[10]

3.2. Ordinary Differential Equation(ODE) Solvers

One of the most crucial parts of ODENet is ODESolver. They are used as black-boxes providing vital part for output calculation of ODENet. To better understand them it is necessary to take a look at their role. Let z be a vector that changes with time. Its time-derivative can be then presented in the form of the equation (3.5):

$$\frac{dz}{dt} = f(z(t), t) \tag{3.5}$$

A typical problem for which ODESolver is used: Given $z(t_0)$ find $z(t_1)$. The simplest way of solving it would be Euler method(3.6):

$$z(t + h) = z(t) + hf(z, t) \tag{3.6}$$

In place of Euler, it is possible to use Runge–Kutta methods for solving differential equations, giving a wide range of algorithms to choose from.

3.3. Backpropagation in ODENet and adjoint method

Knowing how to calculate forward pass of ODENet, the only remaining question is the method of backpropagation thought hidden layers embedded into a parameterized layer of black-box. For standard neural network backpropagation can be expressed in the equation(3.7):

$$\frac{\partial z_t}{\partial z_{t+1}} = \frac{\partial L}{\partial z_{t+1}} \frac{\partial f(z_t, \theta)}{\partial z_t} \tag{3.7}$$

Then loss with respect to weights vector θ_t can be formulated into equation(3.8):

$$\frac{\partial L}{\partial \theta_t} = \frac{\partial L}{\partial z_t} \frac{\partial f(z_t, \theta)}{\partial \theta_t} \tag{3.8}$$

For ODENet loss function can be presented as:

$$L(z(t_1)) = L(z(t_0) + \int_{t_0}^{t_1} f(z(t), t, \theta)dt) \tag{3.9}$$

$$L(z(t_1)) = L(ODESolver(z(t_0), f, t_0, t_1, \theta)) \tag{3.10}$$

Such reverse-mode differentiation incurs high memory costs as all intermediate values of differentiation have to be stored in memory, numerical error and instability. The solution to this problem is to treat ODESolver as a black-box (3.10), and to compute gradients with the help of the *adjoint sensitivity method*[11]. Such approach allows to approximate gradients of by solving second, augmented ODE backward in time[1].

The adjoint sensitivity method starts with the introduction of additional value called *adjoint* (3.11).

$$a(t) = \frac{\partial L}{\partial z(t)} \tag{3.11}$$

Its' dynamics are described by another ODE(3.12).

$$\frac{da(t)}{dt} = -a(t)^T \frac{\partial f(z(t), t, \theta)}{\partial z} \tag{3.12}$$

Which allows us to obtain derivative of loss function for ODE:

$$\frac{dL}{d\theta} = -\int_{t_1}^{t_0} a(t)^T \frac{\partial f(z(t), t, \theta)}{\partial \theta} dt \tag{3.13}$$

Having this value allows for training the ODENet model with the help of standard optimizers.

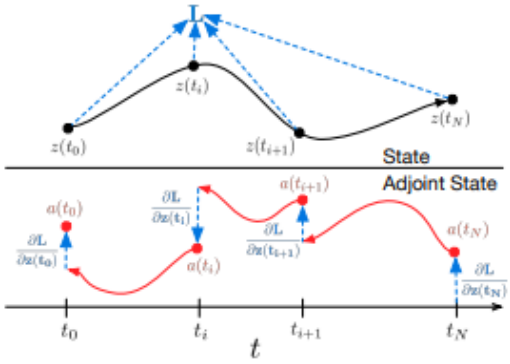

Figure 3.1. Reverse-mode differentiation of an ODE solution. The adjoint sensitivity method solves an augmented ODE backward in time. The augmented system contains both the original state and the sensitivity of the loss with respect to the state. If the loss depends directly on the state at multiple observation times, the adjoint state must be updated in the direction of the partial derivative of the loss with respect to each observation. **Source:** [1]

4. Project Implementation

The main focus of this project was to create neural networks of ODENet and ResNet families. Then to train and test them on bigger and more complex data sets than in [1]. To maximize the reliability of conducted experiments, selected data sets are widely available and well-known.

4.0.1. Software

The project has been implemented in Python 3 using the following libraries:

PyTorch	- 1.3.0	- overall neural network architecture
Torchvision	- 0.4.1	- pre-processing images
Numpy	- 1.17.3	- calculations on matrices
Bokeh	- 1.3.4	- graph creation
Torchdiffeq	- 0.0.1	- ODENet implementation

4.1. Neural network structure

4.1.1. General Structure

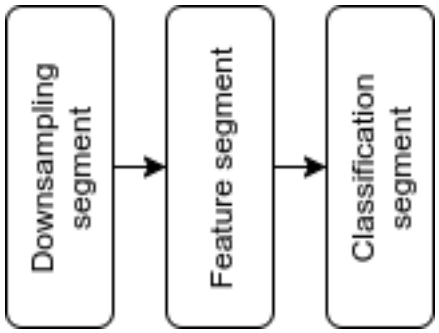

Figure 4.1. Generalized structure of used neural networks

The structure of the designed networks can be split into 3 segments, which will be referred to as the Downsampling segment, Feature segment and Classification segment. All implemented models are created with the following segments, without any alterations, unless specified in this chapter. Feature

segment being the ODE block or chain of residual blocks depending on the used family of neural networks.

4.1.2. Downsampling segment

The downsampling segment is used to decrease data size before introducing it to feature layers and to start extracting features of input data for the learning of the model. It is the same for all learned models with variations only in first convolution layers as implementation is capable of processing both RGB and greyscale images.

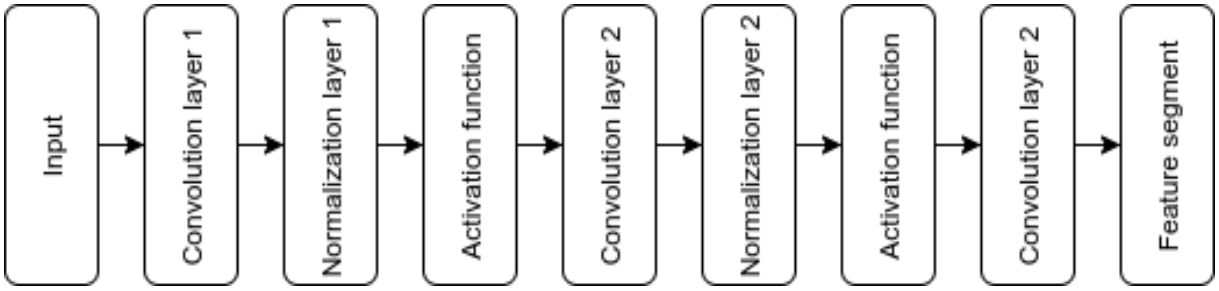

Figure 4.2. Structure of downsampling segment

Convolutional layer 1

- input planes = 1 or 3, depending on the input channels in a dataset
- output planes = 64
- kernel size = 3
- stride size = 1
- padding = 0

Convolutional layer 2 and 3

- input planes = 64
- output planes = 64
- kernel size = 4
- stride size = 2
- padding = 1

Normalization layers

Both provide group normalization over the whole batch of data.

- input planes = 64
- output planes = 64

Activation layers

ReLU is used as the activation function.

4.1.3. Residual block

The feature segment for residual neural networks consist of a chain of residual blocks. They are of the full pre-activation [5] type with the following structure:

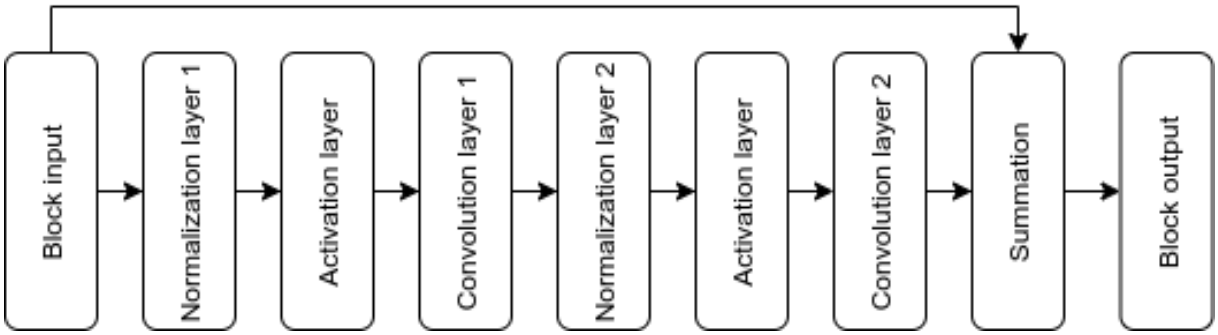

Figure 4.3. Structure of residual block

Convolutional layers

Parameters of layers:

– input planes = 64
– output planes = 64
– kernel size = 3
– stride size = 1
– padding = 1

Normalization layers

Both provide group normalization over the whole batch of data.

– input planes = 64
– output planes = 64

Activation layers

ReLU is used as the activation function.

Summation

Addition operation of the skip connection with the output of the last convolutional layer.

4.1.4. ODE block

The feature segment for the ODENet consists of a single block, which is based on the full pre-activation residual block[5]. During run-time hidden layers are created based on this block structure, resulting in construct similar to Recurrent Neural Networks. The number of hidden layers is determined during backpropagation with help of ODESolver, and they are later accessed with help of parameter t. It has the following structure:

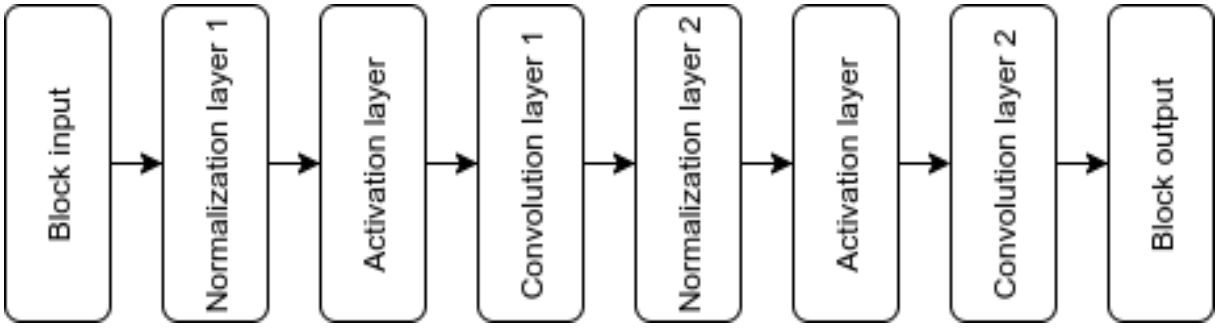

Figure 4.4. Structure of ODE block

Convolutional layers

Additional input plane is introduced by depth parameter t, which is determined by the ODESolver with respect to the error tolerance.

- input planes = 65
- output planes = 64
- kernel size = 3
- stride size = 1
- padding = 1

Normalization layers

Both provide group normalization over the whole batch of data.

- input planes = 64
- output planes = 64

Activation layers

ReLU is used as the activation function.

4.1.5. Classification segment

The function of this part is to process a feature map created in the previous segments and to classify the results into proper amount of classes.

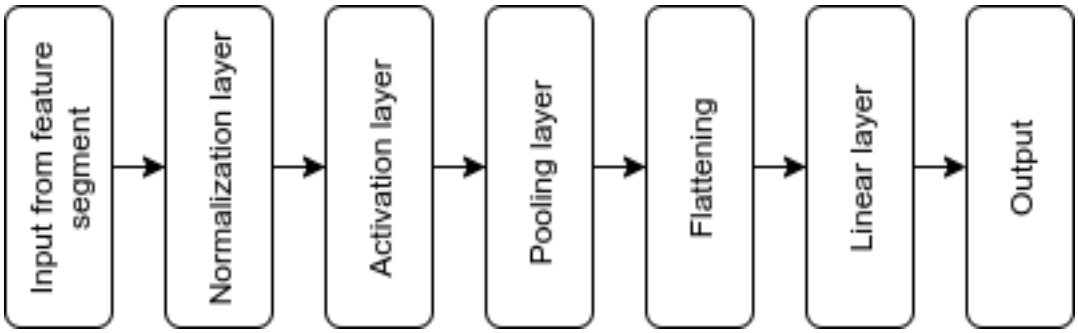

Figure 4.5. Structure of classification segment

Normalization layers

Provides group normalization over the whole batch of data.

– input planes = 64
– output planes = 64

Activation layers

ReLU is used as the activation function.

Pooling layer

The pooling layer applies a 2D adaptive average pooling over the input composed of several input planes.

Flattening

Flattens input into one-dimensional output.

Linear layer

– input planes = 64
– output planes = 10 or 100, depending on the number of classes in a dataset

4.2. Used Datasets

Datasets are obtained with the help of the PyTorchs *util.data* module, which provides various training and testing sets for the learning of models. Among those the following has been chosen for the project:

MNIST

Set of 10 classes of hand-written digits in the greyscale images.
One of the most used basic sets for machine learning.
Considered over-used and unbalanced by the research community.
Consists of 60,000 training images and 10,000 test images.

Source: http://yann.lecun.com/exdb/mnist/

QMNIST

Set of 10 classes of hand-written digits in the greyscale images.
The reconstructed version of the MNIST dataset.
More balanced than the MNIST, with small complexity of the challenge.
Consists of 60,000 training images and 60,000 test images.

Source:[12]

Fashion-MNIST (FMNIST)

Set of greyscale images of 10 types of clothing.
The more complex and realistic challenge for machine vision algorithms than the MNIST digits.
Consists of 60,000 training images and 10,000 test images.

Source:[13]

CIFAR10

Set of natural color images, representing animals and vehicles of 10 classes.
Widely used to evaluate the recognition capability of deep neural networks.
Consists of 50,000 training images and 10,000 test images.

Source: https://www.cs.toronto.edu/ kriz/cifar.html

CIFAR100

Set of a natural color image, representing animals and vehicles of 100 classes.
Significantly more complex than the CIFAR10.
Consists of 50,000 training images and 10,000 test images.

Source: https://www.cs.toronto.edu/ kriz/cifar.html

4.3. Training approach

Proper training of neural networks is a form of art as it requires fine-tuning many hyper-parameters. The process of selecting proper values of those parameters takes huge amounts of resources and is time-consuming due to the high computational costs of training networks.

4.3.1. Adaptive learning rate

Most influential of hyper-parameters is learning rate, with recommended range of values from less than 1.0 to greater than 1^{-6}. Select too big learning and network may never converge, too low and there is a risk of not finding optimal values of weights. Selecting the right value determines the successful learning of the model.

With such a large range of possible values and limited resources, the solution is to use an adaptive learning rate. For the purpose of this project learning rate with step decay has been selected. It decreases the initial learning rate by order of magnitude after every $0.3*$ specified max epochs.

4.3.2. Optimizer

Another important aspect of training is the used optimizer. It is a tool used for finding proper weights of a model to minimize the loss function and increase the accuracy of predictions.

Following two have been selected based on their popularity and efficiency respectively:

– **Stochastic gradient descent with momentum**

– **Adam**

4.4. Data preparation

Input data has to be pre-processed before it is fed to a neural network. Following good practices, all data is normalized to assure balanced training and testing dataset.
Pre-processing for training data involves the following steps:

1. Normalization
2. Up-scaling of image to 32x32
3. Randomly cropping 28x28 image

Pre-processing of validation data consists of resizing an image to 28x28 size.

After pre-processing images are split into batches of 128 images and are ready to be fed into the input of a neural network.

5. Results evaluation

5.1. Evaluation methods

For evaluation criteria presented below only the result with the highest certainty is taken into account. The presented results are for the evaluation phase, which takes place after the training model, it will show a percentage value for the number of correct predictions per epoch. The dataset used for training and validation is the same training dataset, it is preprocessed as described in chapter 4.4.

5.2. Environment

All models have been trained on Google Cloud platform, on instances with the following properties:

- SSD Hard Drive
- 4 core CPU
- 16 GB Ram
- OS Linux image, specially optimized for Deep Learning

ODENet family does not have GPU acceleration, what excluded the possibility of using GPUs for tensor-based calculations. All training has been done on CPUs.

5.3. Results

5.3.1. MNIST

The first step in the project was to create an ODENet model capable of converging on the MNIST dataset. The model has been created according to the schematic in 4.1.1, by combining the Downsampling segment, ODEBlock and Classification segment into sequentional model.

Run settings:

- initial learning rate = $1e-1$
- max number of epochs = 30
- relative error tolerance = $1e-3$
- optimizer = SGD with momentum = 0.9

After creating converging model not using adjoint method, model with its implementation as described in chapter 3.3 has been implemented. Comparison of their validation results in shown on figure 5.2.

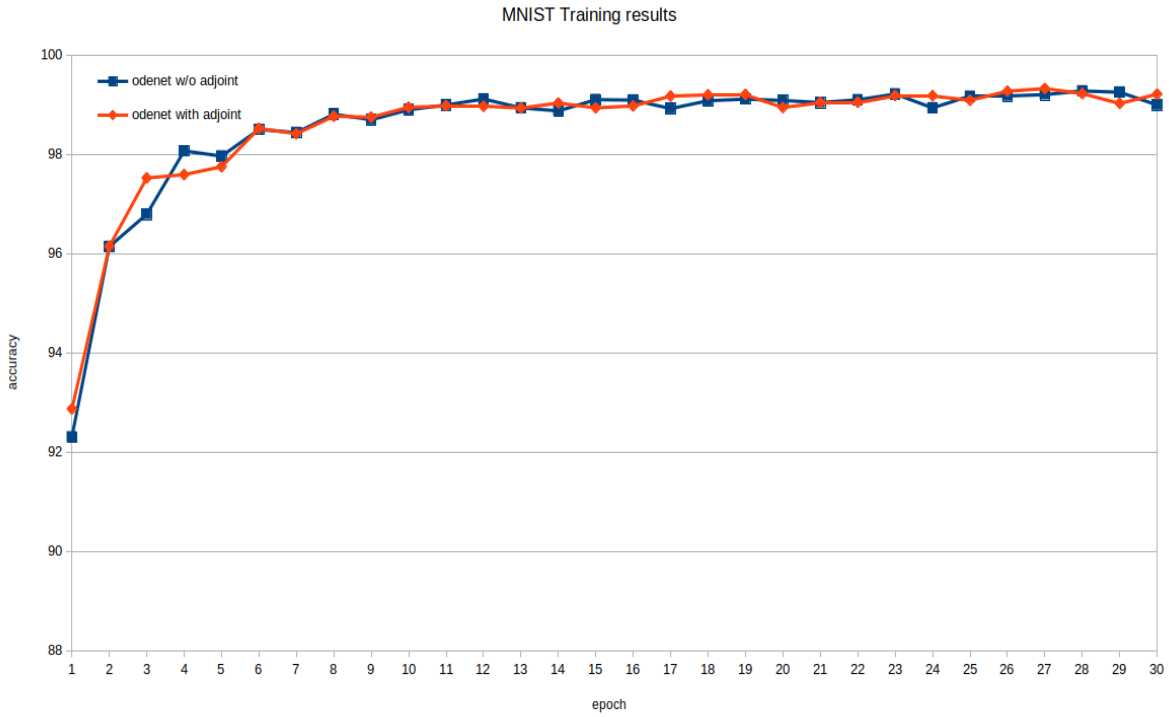

Figure 5.1. Example of evaluation results for ODENets with and without adjoint method on MNIST dataset

Network	ODENet without adjoint	ODENet with adjoint
Average time per epoch [sec]	1826	2270
Created hidden layers	26	26
Number of model parameters	208266	208266

Final comments on dataset

Both networks achieved comparable results of 99%, confirming that prototype is able to converge properly. Obtaining such result is not surprising as MNIST dataset is simple and selected learning methods allowed for quick and precise conversion of models.

With such, promising results the next step is to try models on a more complex database of QMNIST.

5.3.2. QMNIST

Run settings:

- initial learning rate = $1e-1$
- max number of epochs = 60
- relative error tolerance = $1e-3$
- optimizer = SGD with momentum = 0.9

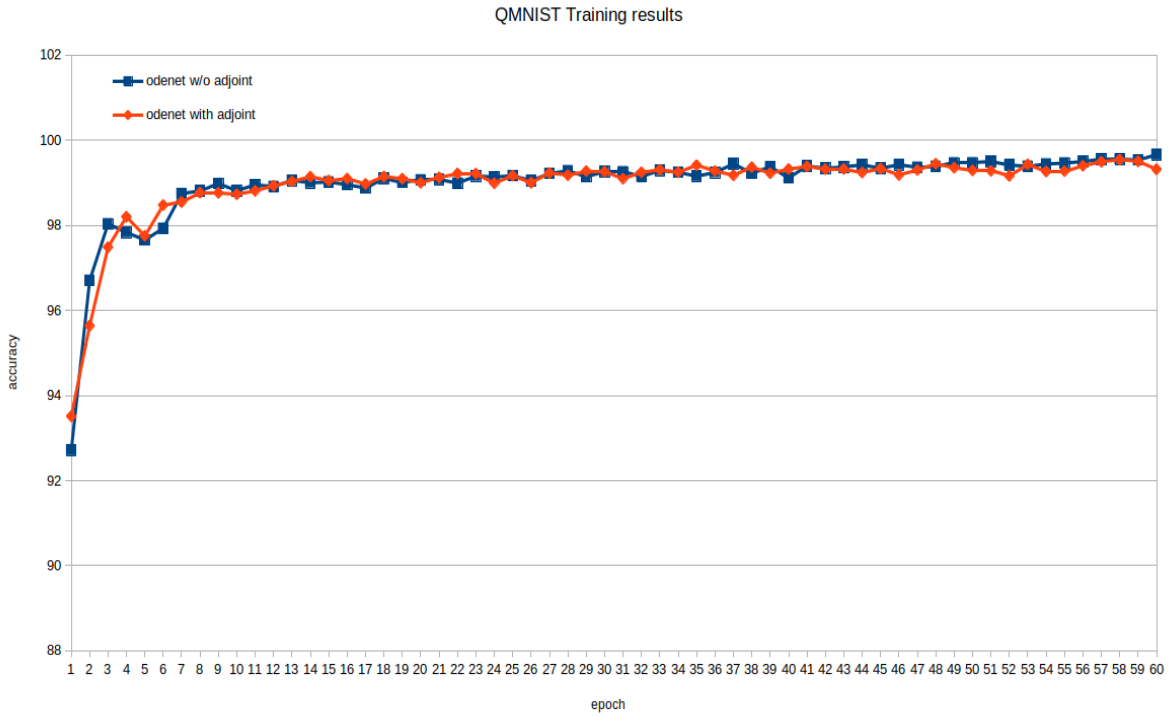

Figure 5.2. Example of evaluation results for ODENets with and without adjoint method on QMNIST dataset

Network	ODENet without adjoint	ODENet with adjoint
Average time per epoch [sec]	1883	2470
Created hidden layers	26	26
Number of model parameters	208266	208266

Final comments on dataset

Again comparable to results for the MNIST dataset, good results of 99,5% have been achieved for validation of training. It has been run significantly longer to check if memory instability issues can be present on low complexity dataset, however, no such issues have been found. The time difference between adjoint and non-adjoint method has been significant at this point, suggesting that current adjoint implementation has to be optimized.

Such results suggest need for even more complex dataset to train ODENet model against ResNet

5.3.3. FMNIST

Fashion-MNIST has been determined as the next database in terms of the complexity of the challenge.

For Fashion-MNIST Dataset I decided to initially train 5 models:

- ResNet with 2 residual blocks
- ResNet with 4 residual blocks
- ResNet with 8 residual blocks
- ODENet with adjoint sensitivity method
- ODENet without adjoint sensitivity method

Run settings:

- initial lr = $1e-1$
- optimizer = SGD with momentum = 0.9
- relative error tolerance for ODENets = $1e-3$
- max number of epochs = 50

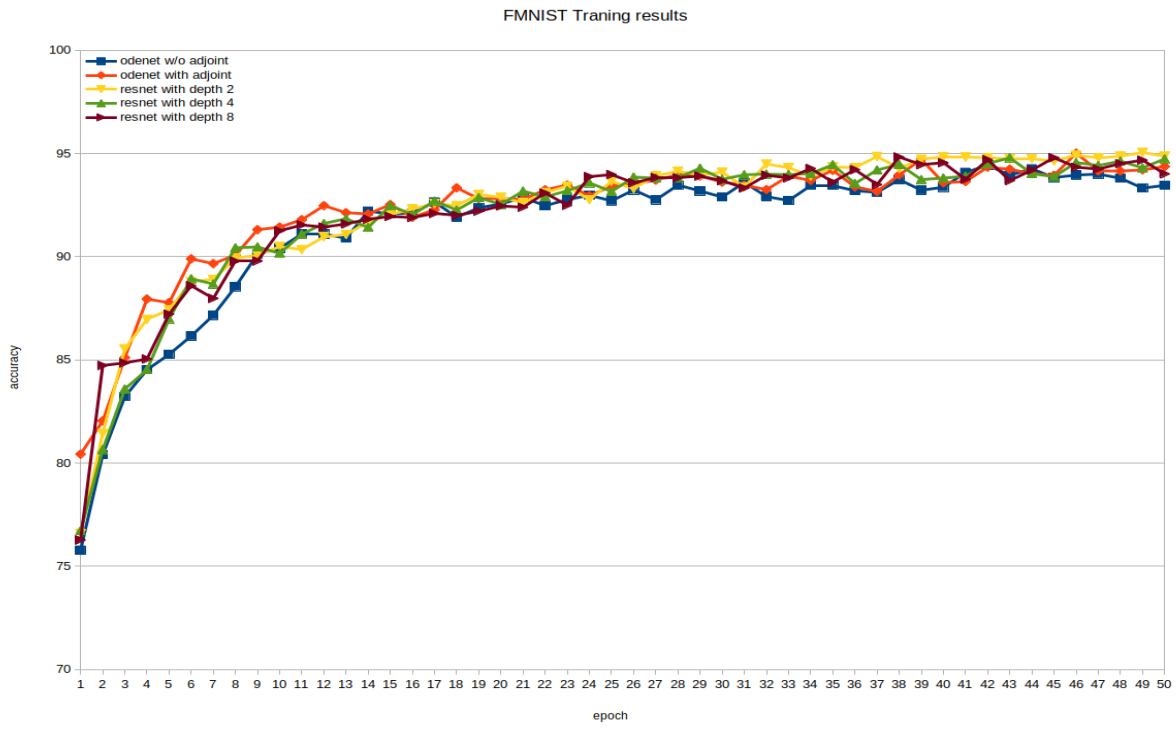

Figure 5.3. Evaluation results for models trained on FMNIST dataset

Network	ODENet w/o adjoint	ODENet adjoint	Resnet 2	ResNet 4	ResNet 8
Avg time per training [sec]	1346	1913	232	248	368
Avg time per validation [sec]	541	540	87	103	145
Avg time per epoch [sec]	1887	2453	320	352	513
Created hidden layers	26	26	N/A	N/A	N/A
Number of model parameters	208266	208266	280842	428810	724746

At this point, an interesting idea appeared. *What if we could stack few ODE blocks in a chain?* With such an idea experiment has been conducted in which ODENet consisting of two ODEBlocks has been created to be tested against ones with a single block.

Run settings: They have been run with the same parameters as in the previous part.

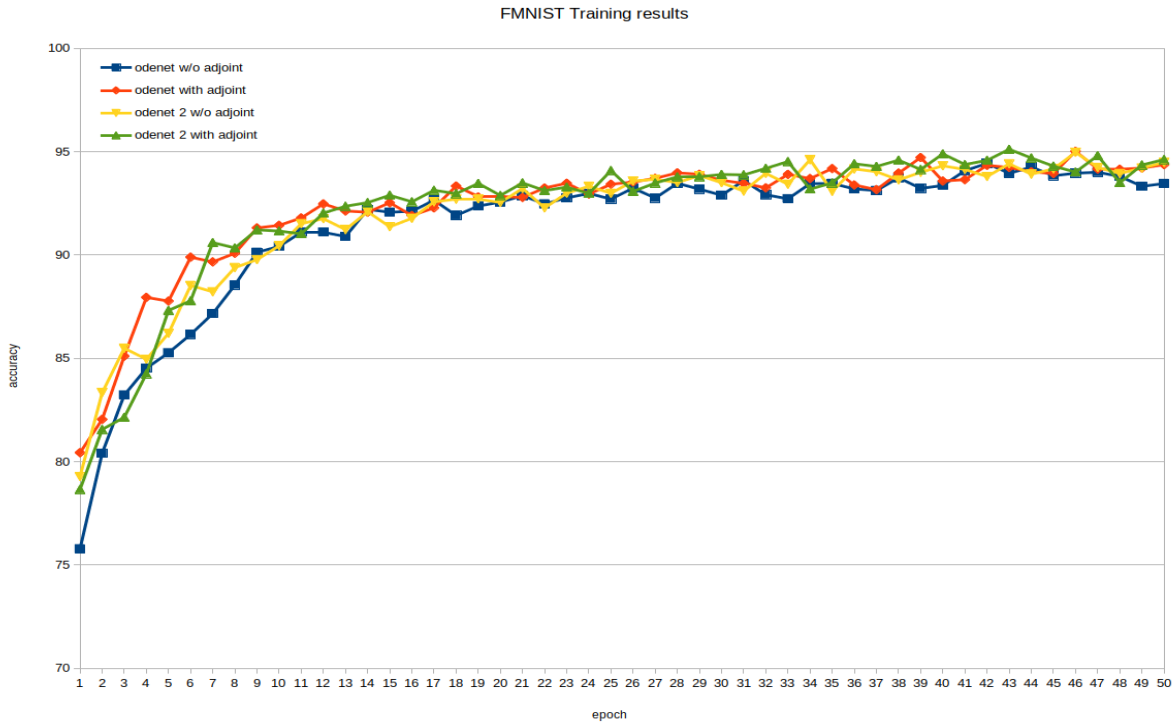

Figure 5.4. Evaluation results for models trained on FMNIST dataset

Number of ODE Blocks	1 block w/o adj	1 block with adj	2 blocks w/o adj	2 blocks with adj
Avg time per training [sec]	1346	1913	2731	1958
Avg time per validation [sec]	541	540	945	898
Avg time per epoch [sec]	1887	2453	3677	2857
Created hidden layers	26	26	20	20
Number of model parameters	208266	208266	283658	283658

Comments on increasing number of blocks for FMNIST

ODENet with an increased number of ODE blocks kept the ability to learn features of a dataset and reach satisfactory levels of accuracy on a dataset. However increasing of blocks does not seem to bring significant benefits while increasing learning time enormously, due to poorly optimized learning algorithms of ODENets. Interestingly models with additional ODEBlock created less hidden layers.

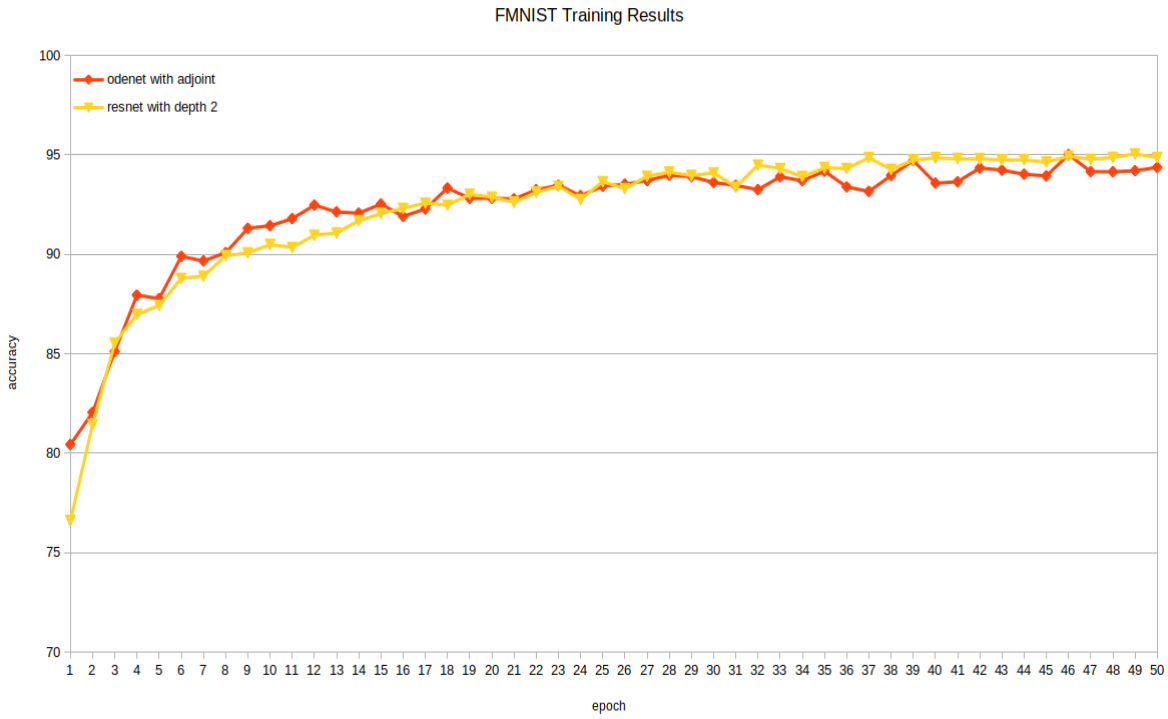

Figure 5.5. Resnet with 2 ResBlocks and ODENet with adjoint comparison

Final comments on dataset

Overall performance on the FMNIST dataset for all networks has been promising, all models managed to reach a satisfactory level of 90%+ accuracy of validation. While reaching similar results, the run time for both training and validation for ODENets has been substantially bigger than their discrete counterparts.

Surprisingly ResNet with the depth of only two residual blocks outperforms all other architectures while having the number of parameters comparable to that of ODENets. Such a result suggests that for datasets of smaller complexity ODENets should be compared with it, not with its bigger siblings.

Fact that nothing unexpected happened to this point, suggested the necessity for a more complex dataset. As next in terms of complexity I have selected the CIFAR10 database. It consists of images presenting various animals and vehicles in color. They are not only much more challenging as color increases task complexity significantly, but are also more realistic challenge than greyscale digits or cloths.

5.3.4. CIFAR10

For CIFAR10 Dataset I decided to initially train 4 models:

- ResNet with 4 residual blocks
- ResNet with 8 residual blocks
- ODENet with adjoint sensitivity method
- ODENet without adjoint sensitivity method

Run settings: As the complexity of the database increased significantly, the number of epochs on which models are allowed to learn has been increased to 100. Such operation had as a goal also establishing the number of epochs for further models.

- initial lr = $1e-1$
- type of learning rate = adaptive with drop of magnitude every 30 epochs
- optimizer = SGD with momentum = 0.9
- relative error tolerance for ODENets = $1e-3$
- max number of epochs = 100

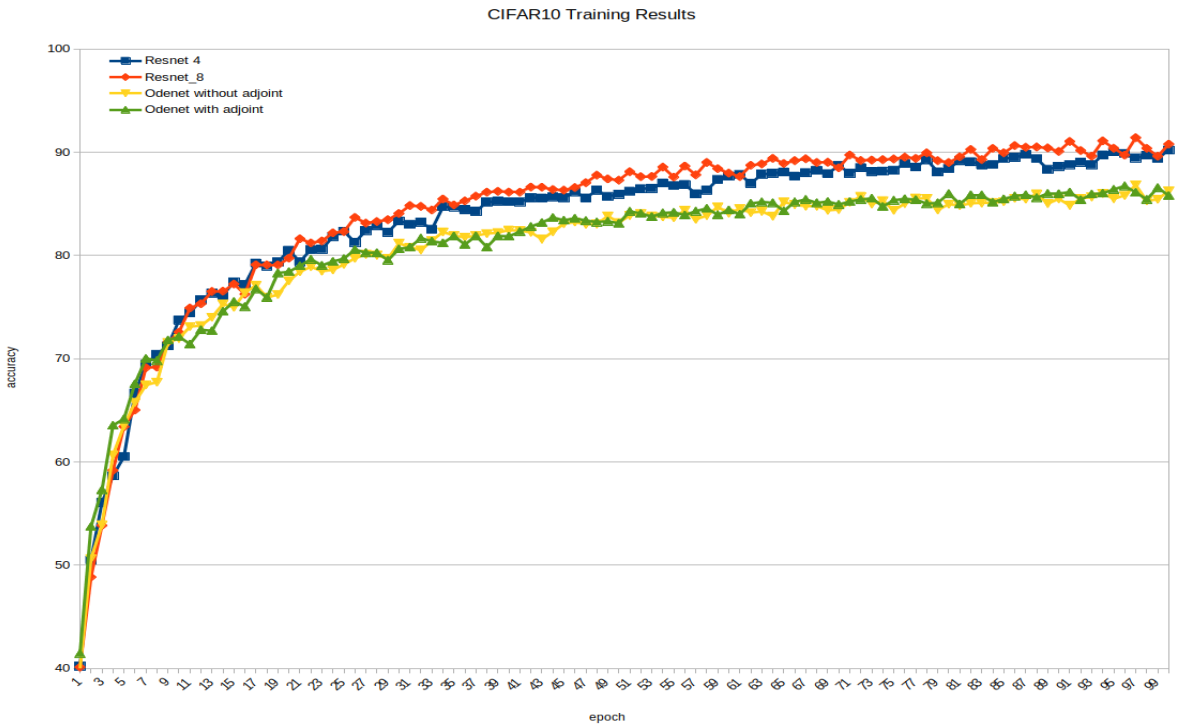

Figure 5.6. Evaluation results for initial run on CIFAR10 dataset

Network	ODENet w/o adjoint	ODENet adjoint	Resnet 4	ResNet 8
Avg time per training [sec]	978	1191	186	360
Avg time per validation [sec]	382	378	75	123
Avg time per epoch [sec]	1361	1570	261	484
Created hidden layers	26	26	N/A	N/A
Number of model parameters	209418	209418	429962	725898

Comments on initial run

As seems on the chart above interesting pattern emerged here, both ODENets achieve much worse results than their Residual counterparts. As a selection of wrong hyper-parameters might have been a reason, I decided to try various combinations of hyper-parameters.

Run settings: Error tolerance variation has been most interesting of hyper-parameters to conduct experiments on. Due to the significant increase in learning time, the number of epochs has been reduced to 25, as any clear difference should have become apparent after such many epochs.

- initial lr = $1e-3$
- type of learning rate = constant, changed to see clearly influence of difference in tolerance
- optimizer = ADAM, selected to counter-act decrease in learning rate
- relative error tolerance for ODENets = $1e-4$ and $1e-5$
- max number of epochs = 25, decreased due to lack of resources

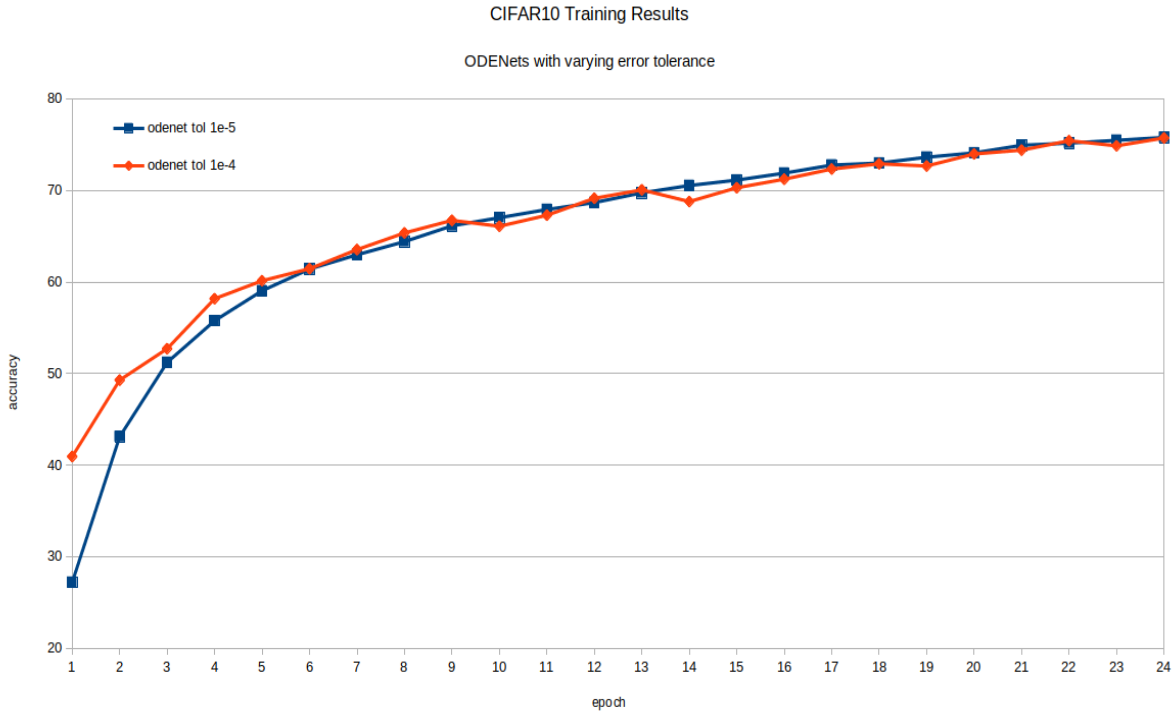

Figure 5.7. Evaluation results for models with varying tolerance on CIFAR10 dataset

ODENet tolerance	1e-4	1e-5
Average time per epoch [sec]	3252	4536
Created hidden layers	34	52

Comments on run with varying tolerance

As seems in the chart above, the selected learning rate has been too small as the model learned slowly. At this point, one complete epoch took around 1h limiting the number of models that have been learned

with such parameters. Interestingly initially bigger tolerance seemed to perform better, however, due to not enough test runs no proper conclusion can be reached.

Final comments on CIFAR10 dataset

Among those I tried various learning rates, using ADAM optimizer and tweaking tolerance rate, however, for ODENets I have been unable to outperform models trained initially.

5.3.5. CIFAR100

Having observed an interesting pattern on CIFAR10 dataset with limitation in a convergence of ODENets. Further investigation of the topic seemed like a reasonable idea. After taking previous observations into considerations neural networks with the following settings have been created.

- ResNet with 4 residual blocks
- ResNet with 8 residual blocks
- ODENet with adjoint sensitivity method, optimized by SGD
- ODENet without adjoint sensitivity method, optimized by SGD
- ODENet with adjoint sensitivity method, optimized by ADAM
- ODENet without adjoint sensitivity method, optimized by ADAM

Run settings:

- initial lr = $1e-1$
- type of learning rate = adaptive with drop of magnitude every 30 epochs
- optimizer = SGD with momentum = 0.9 and ADAM
- relative error tolerance for ODENets = $1e-3$
- max number of epochs = 100

Network	ODENet w/o adjoint SGD	ODENet adjoint SGD	Resnet 4	ResNet 8
Avg time per training [sec]	982	1364	195	371
Avg time per validation [sec]	429	429	83	133
Avg time per epoch [sec]	1411	1793	279	504
Created hidden layers	26	26	N/A	N/A
Number of model parameters	215268	215268	435812	731748

Two additional ODENet models have been created and trained for 200 epochs

Comments on CIFAR100 dataset

ResNets have substantially outperformed ODENets, which have again converged in a similar pattern of convergence to some limit. The observed pattern probably is based on the continuous nature of created models and their inherent limitations. More on the subject can be found in the article [9].

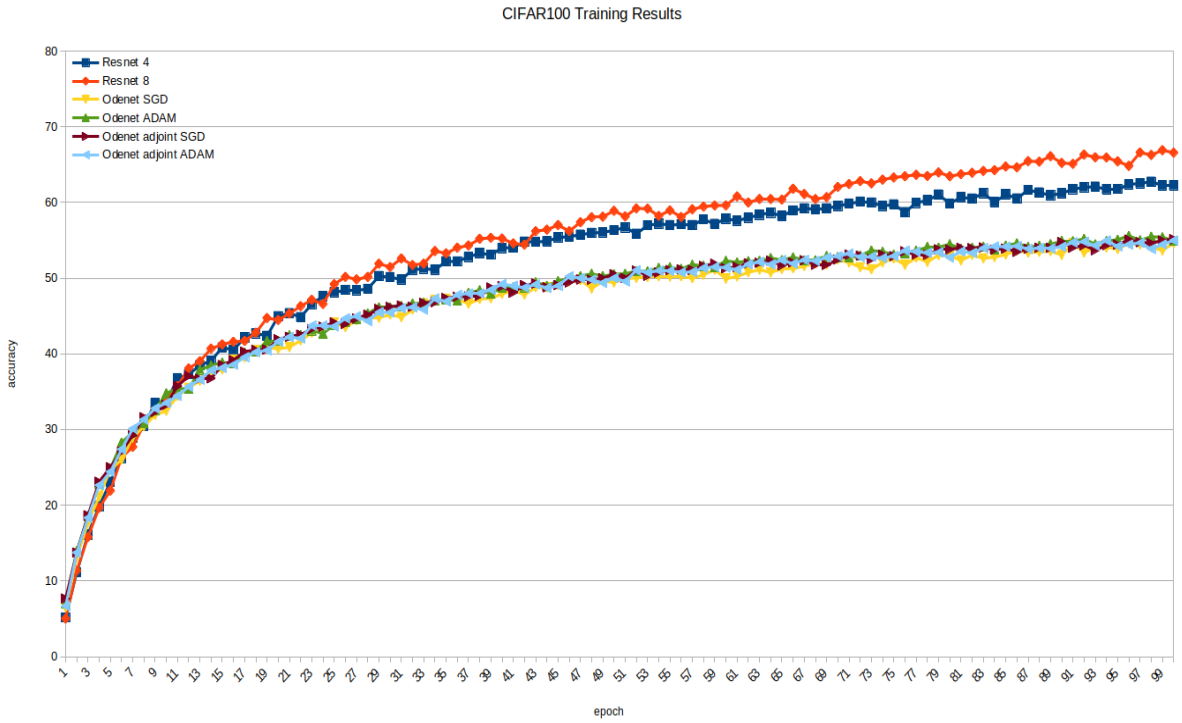

Figure 5.8. Evaluation results for CIFAR100 dataset

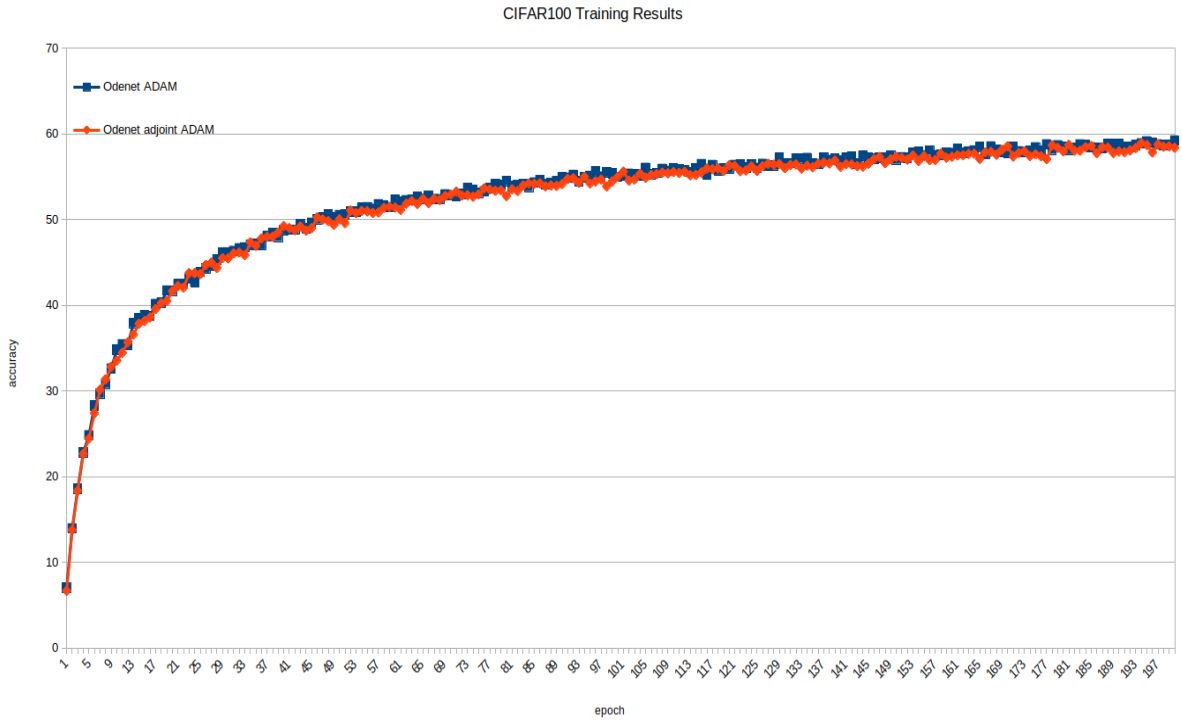

Figure 5.9. Evaluation results for 2nd CIFAR100 dataset

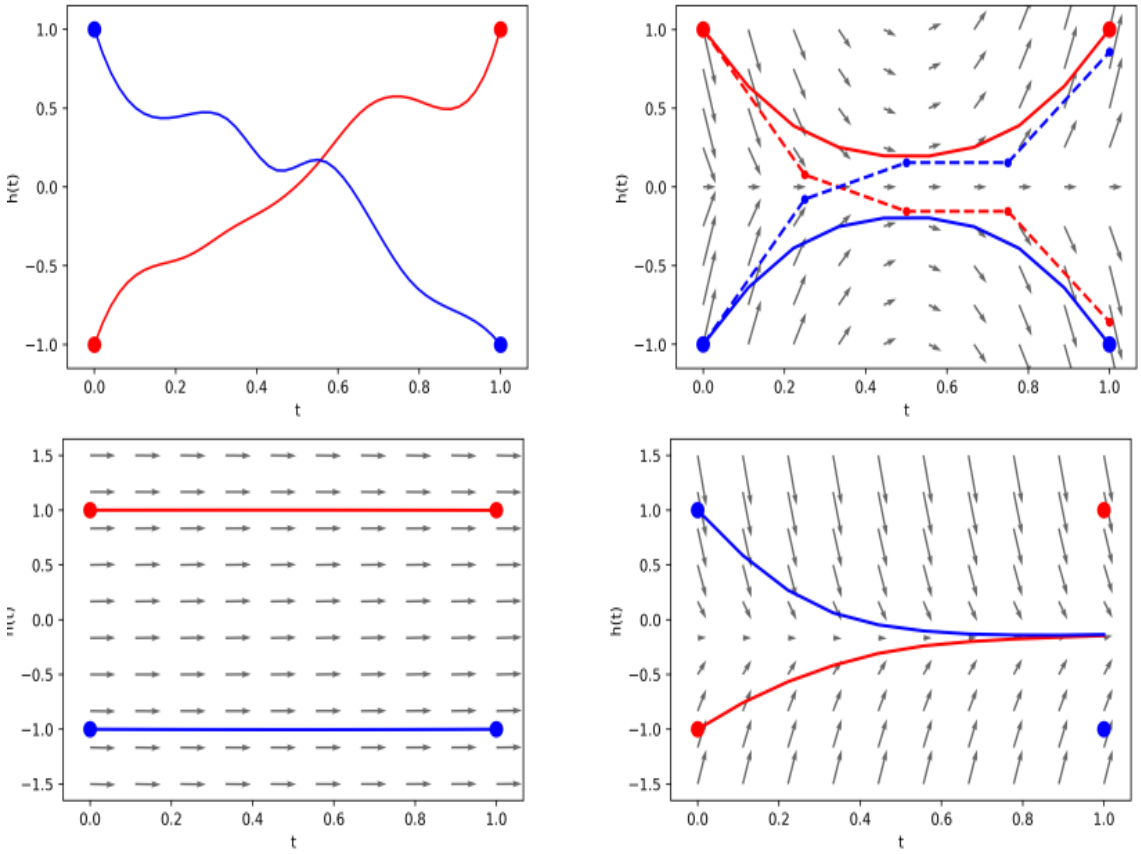

Figure 5.10. (Top left) Continuous trajectories mapping -1 to 1 (red) and 1 to -1 (blue) must intersect each other, which is not possible for an ODE. (Top right) Solutions of the ODE are shown in solid lines and solutions using the Euler method (which corresponds to ResNets) are shown in dashed lines. As can be seen, the discretization error allows the trajectories to cross. (Bottom) Resulting vector fields and trajectories from training on the identity function (left) and $g_{1d}(x)$ (right)
Source:[9]

Conclusion

During this project, the neural networks of ResNet and ODENet families have been implemented, then trained on MNIST, QMNIST, FMNIST, CIFAR10 and CIFAR100 datasets. Models have been trained with supervised learning methodology. For the ODENet family, both models with and without adjoint method[11] have been implemented. After training them their validation results have been compared.

Memory usage for training ODENets without adjoint method has been on average higher than those with adjoint method(7.4GB vs 5.8GB). For training and validation, both have been achieved comparable results. In terms of time spent per training not using adjoint has been more effective. The author of the project believes that adjoint method should be used as it provides a more stable and reliable algorithm for training ODENets, and probably in the future, its optimization will get significantly better.

Comparing learning ResNets and ODENets in the field of image classification has not been favorable for ODENets. They have been outperformed by their discrete counter-parts on every plane except for the size of the created model (864kB for ODENet vs 2,9MB for ResNet trained on CIFAR100 database). For medium-sized models size cost of ResNets is not problematic, and for big-sized models, the time-cost of training ODENets would prove them not viable. In the whole project, they have been compared to the most basic of Residual Neural Networks, simple and shallow one, without any Dropout-like augmentations. One might say that comparing ODENets to ResNet consisting of only 8 residual blocks is unfair, as the main strength of ResNets is their ability to reach depths of 100+ easily. On the other side specifying tolerance rate as $1e-6$ and smaller is problematic. It yields the inherent risk of wrongly rounding small numbers and requires enormous precision in calculations inflicting a heavy increase in training time.

ODENets might be worth considering as a competitor to Recurrent Neural Networks(RNN) in the field of time-series data processing, as they provide robustness to missing samples and share mechanic of recurring through the same weights. However, in the field of supervised learning of image classification, they cannot be compared to the ResNet family at this point.

Bibliography

[1] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. "Neural Ordinary Differential Equations". In: *arXiv:1806.07366* (2019).

[2] Tim Salimans and Diederik P. Kingma. *Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks*. Feb. 25, 2016. arXiv: *cs.LG/1602.07868* [cs.LG].

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition". In: *arXiv:1512.03385* (2015).

[4] Han Zhang, Xi Gao, Jacob Unterman, and Tom Arodz. "Approximation Capabilities of Neural Ordinary Differential Equations". In: *arXiv:1907.12998* (2019).

[5] Kaiming He et al. "Identity Mappings in Deep Residual Networks". In: *arXiv:1603.05027* (2016).

[6] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. "Highway Networks". In: *arXiv:1505.00387* (2015).

[7] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. "Densely Connected Convolutional Networks". In: *arXiv:1608.06993* (2016).

[8] Saining Xie et al. "Aggregated Residual Transformations for Deep Neural Networks". In: *arXiv:1611.05431* (2017).

[9] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. "Augmented Neural ODEs". In: *arXiv:1904.01681* (2019).

[10] Amir Gholami, Kurt Keutzer, and George Biros. "ANODE: Unconditionally Accurate Memory-Efficient Gradients for Neural ODEs". In: *arXiv:1902.10298* (2019).

[11] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishechenko. *The Mathematical Theory of Optimal Processes*. 1962.

[12] Chhavi Yadav and Léon Bottou. "Cold Case: The Lost MNIST Digits". In:

[13] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. Aug. 28, 2017. arXiv: *cs.LG/1708.07747* [cs.LG].