Homework 2: Neural ODEs

Task 1. Please load the following code into Google Colab and make it runnable in Google Colab. This code is for classifying MNIST images.

Using the pip command, I installed '`torchdiffeq`' (a package that wasn't pre-installed), changed a few lines of code to make it Colab-runnable, commented on the relevant details, and then changed the code to run for various adjoint methods. I also commented code into different runnable sections accordingly.

Task 2. What are the structure and role of the ODEfunc class? Write several sentences.

The ODEfunc class is a type of dynamic (time-dependent) differential equation that models the behavior of layers in traditional neural networks, particularly ResNets, by describing how the data, in this case, the MNIST data, evolves through a network. The multiple layers include Norm and Activation, ConcatConv2d, and the calculation of the Number of Function Evaluations (nfe). Norm and Activation are provided by norm1, norm2, and norm3 with ReLU, which aid in stabilizing and normalizing the learning process through normalization and nonlinear activation functions. The ConcatConv2d layer replicates the function of time in differential equations by taking a concatenated input of time t and data x, making the 'dynamic' equation possible by capturing the data's evolution across time. Finally, the Number of Function Evaluations (nfe) is crucial for understanding the computing cost and complexity when utilizing neural ODEs such as this one, as it calculates the number of times the function has been evaluated.

Task 3. What are the structure and role of the ODEblock class? Write several sentences.

Over a specific period of time, the integration process is managed by the ODEBlock, which wraps the ODEfunc. These are some of its main features: Integration Time, where the depth of processing in a network can be viewed as the amount of time over which the ODE is solved, represented by the parameter 'integration_time'; ODE Solver, where the ODE defined by ODEfunc is numerically solved using the odeint function and one of several adaptive or fixed step-size solvers, including Euler, DOPRIS5 or 8, and RK4. This solver propagates input via layers by solving the ODE's continuous transformation, similar to a conventional network. Additionally, Properties and Setters for nfe include the previously mentioned nfe and its setter, which allow for management and monitoring of the computing load during training by accessing and resetting the number of function evaluations.

Task 4. The entire architecture consists of the following three layers: Downsampling -> Neural ODE -> Classifier. Please describe the role of each layer. For each layer, write several sentences.

The entire architecture consists of the following three layers: Downsampling, Neural ODE, and Classifier. The roles of each layer are interconnected and can be described as follows.
First, Downsampling decreases the spatial dimension of input images, which is crucial for efficient processing in subsequent layers. Our experiment employs two techniques for downsampling: convolutional (conv) and residual block (res). The residual block technique maintains gradient flow in the network while reducing resolution, ensuring that information is preserved during the downsampling process. On the other hand, the convolutional technique extracts essential features from input images while reducing their spatial dimensions through a sequence of convolutional layers with strides larger than one. The goal of both techniques is to convert raw pixel data into a format that emphasizes higher-level features, making them more valuable for the classification tasks handled by later layers.Next, the Neural ODE layer plays a critical role by transferring data continuously from one layer to the next, rather than utilizing

discrete stages as traditional methods do. This continuous data transformation is achieved by treating the data as a differential equation, hence the name Neural ODE. Within this layer, the ODEfunc is encapsulated in the ODEBlock, which allows for the solving of classification tasks using various ODE solvers. This continuous transformation enables more flexible and potentially more accurate modelling of the data as it flows through the network. Finally, the Classifier layer takes the processed features and converts them into class probabilities, which are the final predictions of the network. This process involves several steps: activation functions are applied to the features, normalization ensures consistent scale, and adaptive pooling reduces spatial dimensions to single values. The multidimensional result is then flattened into a flat vector, which is mapped to a single MNIST class through linear transformation. This structured approach ensures that the raw input is effectively transformed and interpreted, leading to accurate classification outcomes.
Thus, each layer builds upon the previous one, creating a cohesive and efficient architecture that enhances the accuracy and performance of the classification tasks.

Task 5. Set adjoint=True and train. You can see values for each epoch during training. Describe the meaning of each value with several sentences. In particular, NFE values are important.

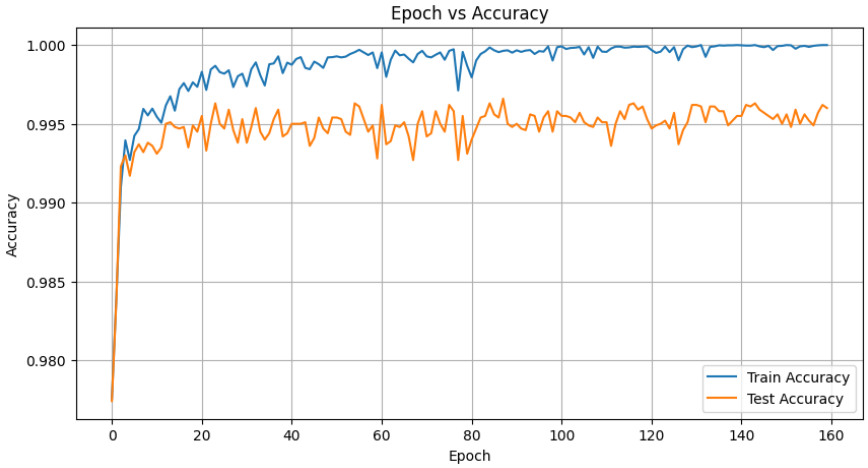
Below are the described different values: Time, Test Accuracy, and Training Accuracy. Time refers to the amount of wall-clock time needed for a single training epoch, indicating the real-time (in seconds) required to process the training set of data through the network, including both forward and backward passes. The values in this case are in the range of 50 seconds. Test Accuracy is a useful measure of the model's generalization ability, representing the percentage of accurate predictions the model produced on the unseen test dataset. Comparing test accuracy with training accuracy can help identify issues such as overfitting. In our instance, the test accuracy is approximately 99.7%. Training Accuracy refers to the proportion of accurate predictions the model produced using the training dataset, serving as a measure of how well the model is able to learn and adapt to the training data. During our session, we observed the training accuracy to be nearly 100%.

The additional nfe values discovered are NFE-B and NFE-F. NFE-B represents the number of evaluations during the backwards pass, particularly during the gradient calculation and backpropagation phase, with B standing for backward. In our case, the backward evaluation differs from the forward pass because the adjoint method was used to compute the gradients, resulting in a value of around 20.0. NFE-F represents the number of times the ODEfunc was evaluated for each epoch during the forward pass of the training phase, with F standing for forward. This value indicates the extent to which the model works to solve the ODE during forward propagation, and we obtained it to be around 26.2.

Task 6. Describe the differences between adjoint=False and adjoint=True in a paragraph. Train with both options and report their results in terms of training speed and accuracy. When adjoint=True, you should test Euler, RK4, and DOPRI. You should draw a chart of epoch vs. training/testing accuracy and report the average wall-clock time for each epoch in all the four cases, i.e., adjoint=False and adjoint=True with Euler, RK4, and DOPRI.

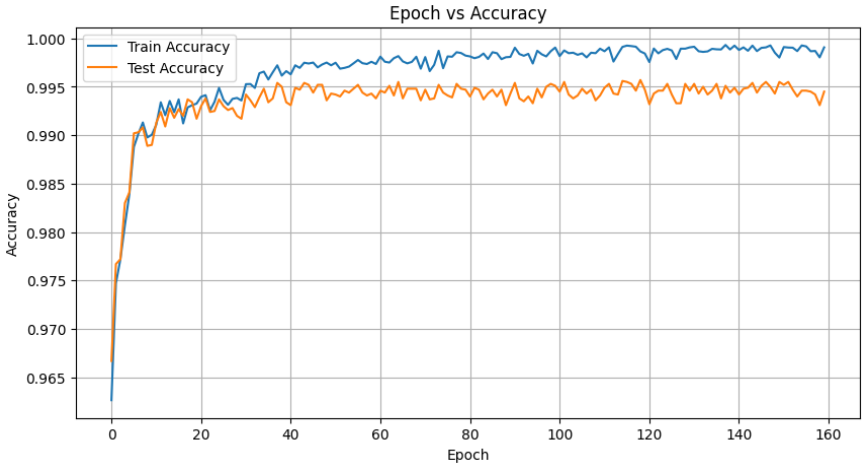
The following results were obtained:

1. **adjoint = False**



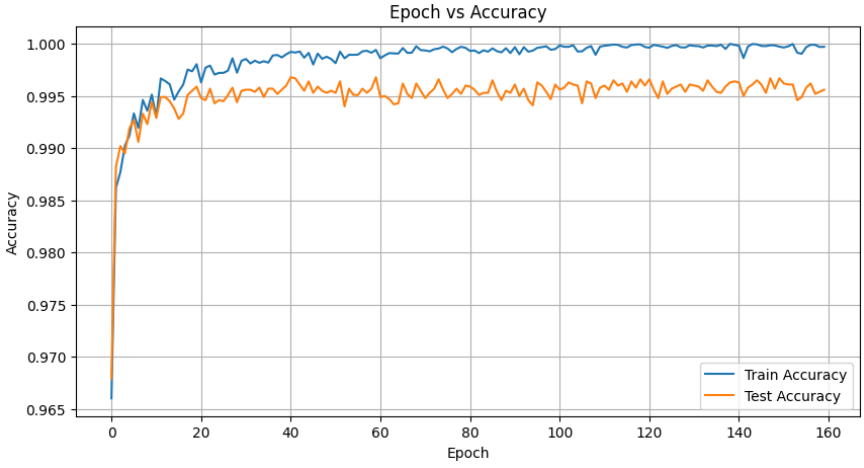
Average Wall-Clock Time per Epoch: 33.063 seconds
NFE-F: 26.2, NFE-B:0.0

2. adjoint = True, 'euler'



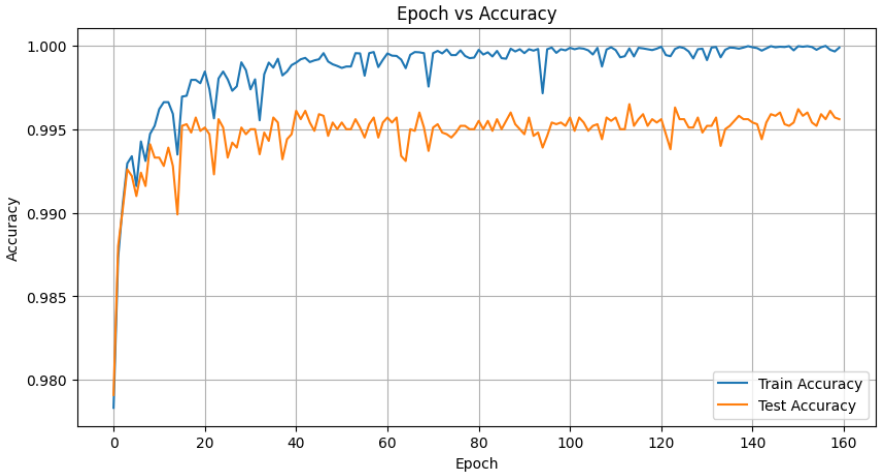
Average Wall-Clock Time per Epoch: 6.621 seconds
NFE-F: 1.0, NFE-B: 1.0

3. adjoint = True, 'rk4'



Average Wall-Clock Time per Epoch: 8.924 seconds
NFE-F: 4.0, NFE-B: 4.0

4. **adjoint = True, ‘dopri5’**



Average Wall-Clock Time per Epoch: 45.219 seconds
NFE-F: 26.2, NFE-B: 20.0

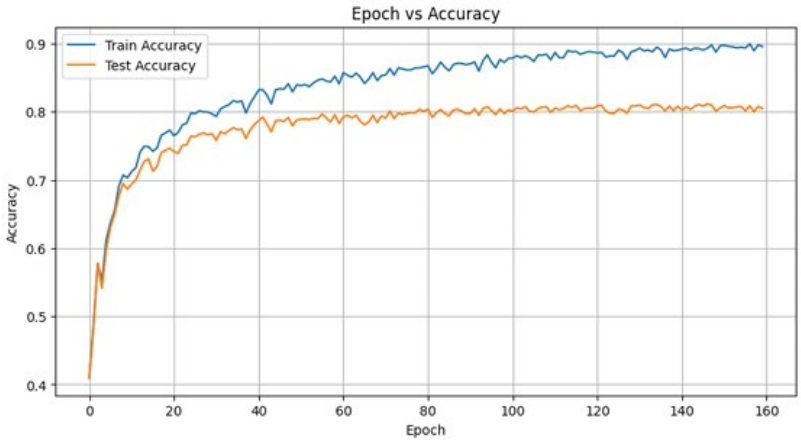
When we compare the True and False conditions of the adjoint, the main difference is that when adjoint = False, the NFE-B value was 0, which indicates that no function evaluations were performed in this case during the backward pass.
It doesn't seem to have much of an impact on accuracy because all the other four ‘True’ tests' accuracies are similar, with dopri5 being a bit higher. We could reason that there is a difference in the training times; dopri5 is just very slow, averaging 45.2 s per epoch, rk4 is roughly 8.9 s, and euler is about 6.6 s per epoch, whereas adjoint = False returns an average of 33 s per epoch.

Task 7. Test with other image datasets, e.g., cifar-10, svhn, tiny imagenet, etc. You can choose one dataset that you want. If your selected dataset is too large, you can subsample images for finishing homework quickly. In this step, you should tune your model with the following techniques: modifying your model architecture if you want, batch norm, testing various optimizers and ode solvers with adjoint=False and True, drop out after adding FC layers, learning rate scheduling, etc.

To compare the model's performance on the CIFAR-10 dataset, which consists of coloured images sized 32x32, as opposed to MNIST's grayscale images of 20x20, a baseline was established by running the best model from MNIST, configured with adjoint=True and method='euler' due to its speed.

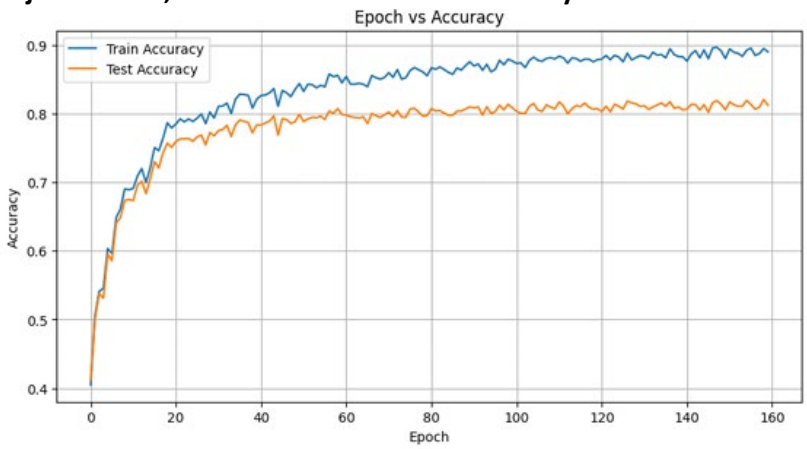
The only adjustments made to the code were changing the input channels for the convolution layers from 1 (black and white) to 3 (coloured) and modifying the random crop size from 28 to 32.

1. **adjoint = True, ‘euler’ but with a few minor changes here and there (Baseline)**



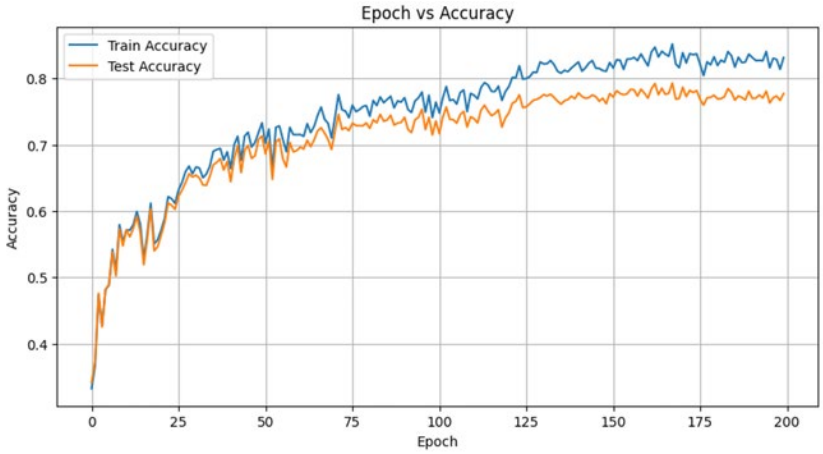
The graph above depicts that the highest test accuracy was limited to approximately 80%. In order to maintain spatial dimension, another layer was added, a normalization layer, one dropout layer (0.1), and downsampling with extra layers and a stride of 1 (and padding=1).

2. **adjoint = True, 'euler' but with an additional layer and conducted with light dropout**



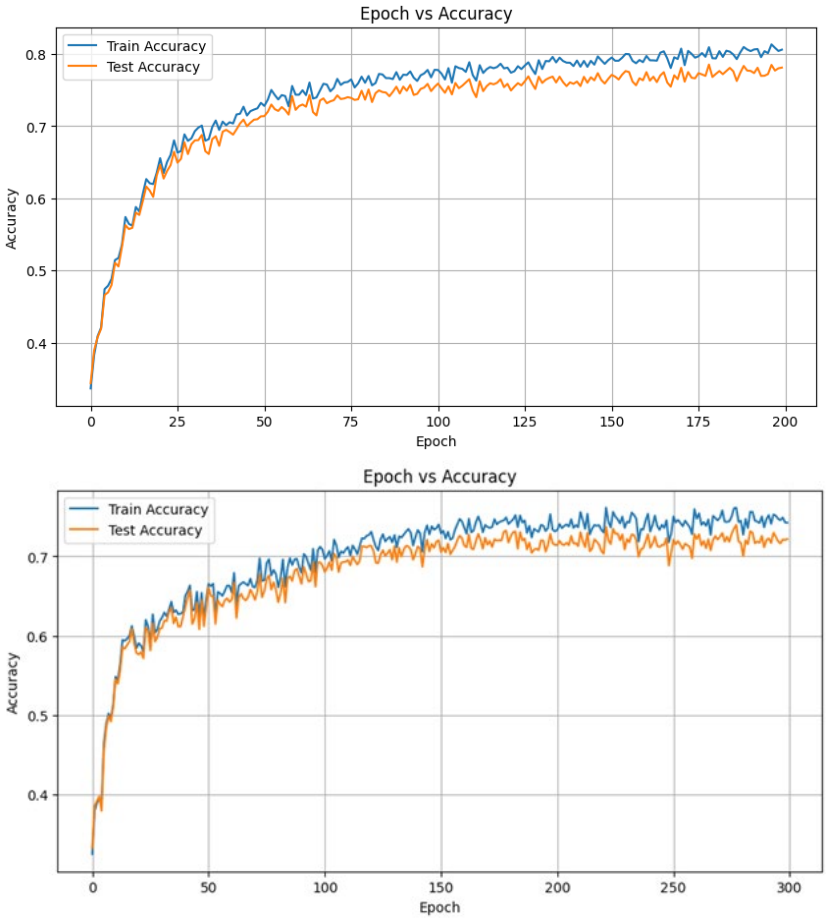
The graph above indicates that accuracy improved, but only somewhat, and the improvement was gradual. Consequently, the decision was made to try using Adam as the optimizer instead of SGD, maintaining the same learning rate, and adding one more convolutional layer. Additionally, the number of epochs was increased to 200.

3. **adjoint = True, 'euler' but with additional layers and conducted with light dropout, Adam (lr = 0.001) as the optimizer, and ran on 200 epochs**



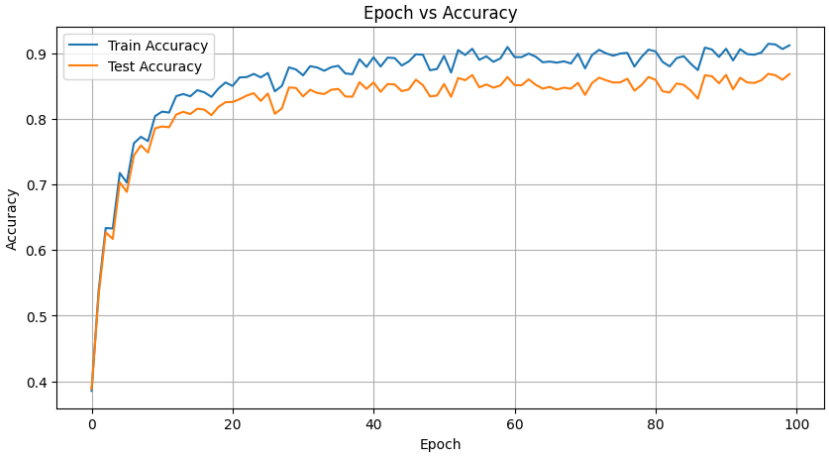
The graph above shows that there wasn't much of a difference, which may indicate that the model didn't perform as well as it had previously. Consequently, the decision was made to return to SGD as the preferred optimizer, while retaining the additional layers and 200 epochs. Data augmentation techniques such as rotation, horizontal flipping, colour jitter, and normalization (of test and train data) were also added.

4. **adjoint = True, 'euler' but with additional layers and conducted with light dropout, ran on 200 epochs and augmentation**



Sadly, this initial run performed poorly and took the longest, approximately 30 seconds per epoch, although it showed signs of improvement. As a result, training continued for an additional 100 epochs, with the outcomes presented in the second graph. An extra 100 training epochs were carried out in an attempt to surpass baseline A results, but no improvements were observed. After reviewing some literature and drawing insights, modifications to the training strategy were implemented. These included adding specific normalization values ((0.4824, 0.4713, 0.4357), (0.1862, 0.1703, 0.1875)) and then applying data augmentation techniques such as random cropping, random horizontal flipping, and colour jitter for hue and saturation at 0.05 each. The model was then trained with a weight decay of 0.0001 for an additional 100 epochs. Architectural changes were also made, including reducing the number of fully connected layers, using 'dopri5' as the solver, and downsampling with the 'res' method.

5. **adjoint = True, 'dopri5' but with augmentation, residual downsampling, weight dropping, and ran on 100 epochs**



The experiment produced findings that exceeded expectations. The average wall-clock time during the training process was approximately 40 seconds per epoch. Notably, the model outperformed the baseline, achieving a peak accuracy of around 85%, which is an improvement of about 6%. Additionally, the model achieved convergence early in the 100-epoch period. In light of these results, the model's performance proved to be satisfactory.

Task 8. In our lecture, I didn't prove the backward and the gradient of neural odes but just compared them with those of ResNets. Derive the backward and the gradient of neural odes on your own. You can refer to the following paper or other online materials. I want to let you know that there are several ways to prove. Some are very complicated while others are surprisingly simple.

The backward pass and gradient for neural ODEs can be derived as shown below, using the cited study and an article (mentioned in the references):

We know that the forward pass: $\frac{dx(t)}{dt} = \delta(x(t), t, \theta)$ at an intial state of x(t_0)

Solving the ODE in order to find out the state at a later time T, we get

$$x(T) = x(t_0) + \int_{t_0}^{T} \delta(x(t), t, \theta)\ dt$$

Let us define the loss function as L(x(T)), where we want to compute the gradient $\frac{dL}{d\theta}$. Thus, we will introduce the adjoint state a(t), which is the gradient of the loss with respect to the state at time t: $a(t) = \frac{dL}{dx(t)}$

Hence, the adjoint state dynamics are given by, $\frac{dL}{d\theta} = \frac{dL}{dx(t)} * \frac{dx(t)}{d\theta}$ (from chain rule)

⇨ $\frac{dL}{d\theta} = \int_{t_0}^{T} a(t) * \frac{d\delta(x(t),t,\theta)}{d\theta}\ dt$ since x(T) depends on x(t) and θ

Now, to compute a(t), we need to solve the adjoint equation, which includes backward in time:

$$\frac{da(t)}{dt} = -a(t) * \frac{d\delta(x(t), t, \theta)}{dx(t)}$$

Here, $\frac{d\delta(x(t),t,\theta)}{dx(t)} = J$, the Jacobian of δ with respect to x(t).

At the final time T, the adjoint state is initialised as $a(t) = \frac{dL}{dx(t)}$

⇨ We obtain the backward pass as $a(t) = a(T) - \int_t^T a(\tau) * \frac{d\delta(x(\tau),\tau,\theta)}{dx(\tau)}\, d\tau$

And gradient of the loss $\frac{dL}{d\theta} = \int_{t_0}^T a(t) * \frac{d\delta(x(t),t,\theta)}{d\theta}\, dt$

References:

1. https://github.com/rtqichen/torchdiffeq/blob/master/examples/odenet_mnist.py

2. https://github.com/rtqichen/torchdiffeq/blob/master/examples/latent_ode.py

3. Neural ODE from scratch and revisit backward propagation | by Fei Cheung | Medium

4. [1806.00736] Complete 72-Parametric Classification of New and Old Kinds of Surface Plasmon Waves (arxiv.org)

5. GitHub - fabiocarrara/neural-ode-features: Pytorch code to train image classifiers based on ODE Nets on MNIST and CIFAR-10, extract features and test robustness to adversarial examples

6. Carrara, F., Amato, G., Falchi, F. and Gennaro, C., 2019, September. Evaluation of Continuous Image Features Learned by ODE Nets. In International Conference on Image Analysis and Processing (ICIAP '19) (pp. 432-442). Springer, Cham.

7. Carrara, F., Amato, G., Falchi, F. and Gennaro, C., 2020, June. Continuous ODE-defined Image Features for Adaptive Retrieval. In Proceedings of the 2020 International Conference on Multimedia Retrieval (ICMR '20) (pp. 198-206). ACM.

8. Carrara, F., Caldelli, R., Falchi, F. and Amato, G., 2019, December. On the robustness to adversarial examples of neural ode image classifiers. In 2019 IEEE International Workshop on Information Forensics and Security (WIFS '19) (pp. 1-6). IEEE.

9. Additionally, I received help from one of my seniors in understanding the working and explanation behind this topic, both conceptually and in coding, and how to resolve when troubleshooting.