Hand Fingers Based Digit Detection

Mac Shigeki Chong 2666136 | Anh Van Giang (V) 2658762 | Shahrin Rahman 2660701 | Stephen Kwan | Danna Shao 2663369

April 4, 2022

Abstract

In this project, we use various Convolutional Neural Networks (CNNs) to identify what number a hand gesture is trying to show. All models are trained on a dataset with about 21600 images of left or right hand fingers showing number 0 to 5. The goals are to find or formulate the best model to recognize the shown digits and attempt to apply the persistent homology tool from Topological Data Analysis (TDA) for data-preprocessing. The result shows that among AlexNet, DenseNet and our custom architecture IzolNet, IzolNet has the best time efficiency whereas the accuracy of these models are almost the same.

Introduction

Computer image recognition technology has made unprecedented breakthroughs in the last decade, which has been largely driven by the development of CNNs. Since the CNN architecture AlexNet[1] won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), known as the world cup of computer vision, by a landslide in 2012, CNNs have been dominating the field of computer vision (CV).

The aim of this research project is to develop the 'best' CNN architecture on recognizing the number that a hand is showing. The motivation for this topic is that its extensions have a wide range of promising applications in the field of human-computer interaction, such as commanding machines using hand gestures or sign language recognition. Starting with this simple task, we compared the performance of different models in the course of the study and attempted to build a custom one in an effort to improve accuracy, which would give us a guide to future applications in this area. In addition to the aforementioned aim, we observe how applying a tool called persistent homology from Topological Data Analysis (TDA) may assist us in data pre-processing and feature extraction.

Background

In this section we start with a brief review of the theory behind neural networks (NNs) and convolutional neural networks (CNNs) in general. Next we introduce the CNN architectures that were used in the code, namely AlexNet and DenseNet. Lastly, the concept of Topological Data Analysis (TDA) and Persistent Homology as well as how these were implemented in the context of this report shall be discussed.

2.0.1 Neural Network

A neural network (Figure 8 in appendix), to put it simply, is a directed multipartite graph with every layer as an independent set of nodes (or vertices). The first and last layers are called input and output layers, every other layer in between are called hidden layers. Every two contiguous layers (with respect to the direction) form

a complete bipartite graph with weights attached to each edge. Each node h in the hidden layers is calculated as

$$h = \sigma(W^T X) + b$$

where W, X, b are the weights, inputs, biases of the nodes connected to h and σ the non-linear function that helps our network learns non-linear features. Finally, the last hidden layer is connected to an output layer where the outputs are used with a loss function for backprobagation to minimize error so that the network can learn [2].

2.0.2 Convolutional Neural Network

Convolutional neural networks (CNNs) [3] are similar to regular neural networks in the sense that they are also made up of nodes with learnable weights and biases. The difference between the two is that CNN has a layer called convolution layer whose input is an image as opposed to regular neural networks. These convolution layers scale well for images because they take into account the two dimension spatial extent of the images, i.e. a neighborhood of pixels of the images for feature detection, and greatly reduce the number of parameters whereas regular neural networks would not.

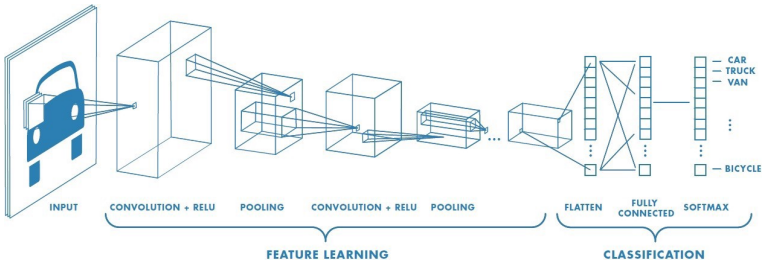

Figure 1: Example of a CNN [4]

There are more special layers to a CNN than a convolutional one but for the purposes of this report, allow us to redirect interested readers to a more comprehensive guide [3].

2.0.3 AlexNet

Although CNNs such as LeNet were used in the machine learning and computer vision communities before AlexNet, they were used on relatively small datasets of labeled images consisting of about tens of thousands of images [1]. This was not very useful as it did not represent realistic settings where variance could be largely affected. Growing technological capabilities in society (e.g. GPUs) as well as the possibility to collect larger datasets consisting of hundreds of thousands (LabelMe) to millions (ImageNet) of high-resolution images gave some hope to solve the above problem. However another issue was the expensive cost of applying CNNs in a large scale to high resolution images. Krizhevsky et al., built AlexNet to resolve these issues [1].

AlexNet is the first deep CNN to classify the high resolution images in ImageNet on GPUs and to have won the ILSVRC in 2012 by a large difference compared to the previous state of the art. It was mainly implemented to show that such a CNN could run on GPU hardware and thereby reduce training times while improving performance.

The architecture of AlexNet (Figure 9 in the appendix) is similar to LeNet and is composed of an 8 layer CNN: 5 convolutional layers where some were followed up with max-pooling layers and 3 fully connected layers that were followed up with a 1000-way softmax classifier. The ReLU nonlinearity was then applied after all the convolution and fully connected layers. Furthermore in order to reduce overfitting, they employed two methods: dropout and data augmentation techniques such as generating image translations and reflections as well as using Principle component analysis (PCA) on the RGB pixel values. Overall AlexNet has 60 million parameters.

On top of the properties mentioned above, the model that won the competition implemented a few more details such as using normalization layers, a batch size of 128, stochastic gradient descent with momentum as the learning algorithm and ensembling. The use of the ReLU nonlinearity, overlapping max pool layers, as well as the dropout method were what made AlexNet so interesting at the time.

2.0.4 DenseNet

As it is known both theoretically and experimentally that increasing depth of a neural network is more effective in improving the network's complexity than width, we chose to use DenseNet 121 [5] by using Tensorflow. However, as the depth increases, training becomes more difficult due to the multi-layer back-propagation of the error signal can easily lead to gradient dispersion (too small a gradient can make the learned training error extremely weak) or explosion (too large a gradient can cause a "NaN" in the training). On the other hand, the biggest advantage of DenseNet is its optimization of the gradient flow. The key idea of DenseNet is to add a separate

shortcut to each preceding layer so that any two layers can "communicate" directly with each other, receiving a "collective knowledge". Therefore, DenseNet has a higher computational and memory efficiency.

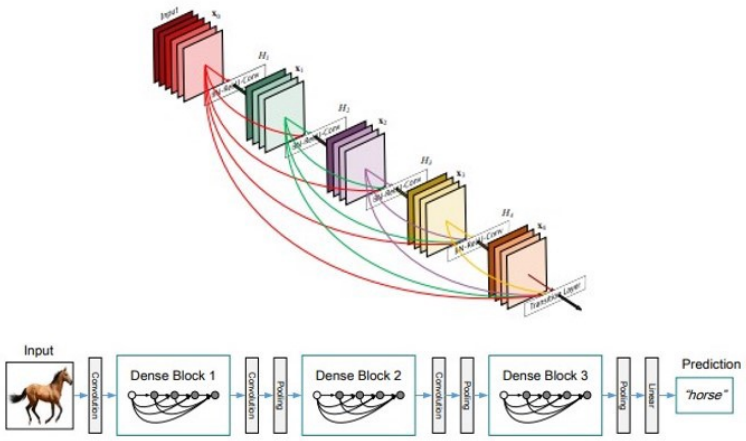

Figure 2: DenseNet

Each layer in the network is connected to all other levels deeper in the network, as shown in the figure above. For example, the first layer is connected to the 2nd, 3rd, 4th, and so on, while the second layer is connected to the 3rd, 4th, 5th, and so on. This is done to ensure that the maximum amount of data can move between the network's layers. To maintain the feed-forward character of the system, each layer receives input from all previous layers and passes on its own feature maps to all subsequent layers.

2.0.5 Topological Data Analysis

One way of utilizing Topological Data Analysis (TDA) is to apply techniques from algebraic topology to extract features from images for the data pre-processing in a machine learning task. For the purpose of this project, we are going to use a tool called persistence homology. For simplicities' sake, we shall only introduce the bare minimum amount of information needed to grasp said topic and explain why such methods are useful.

Definition 2.0.1. A k-dimensional simplex σ, or k-simplex, spanned by a set $P = \{p_1, p_2, ..., p_k\} \in \mathbb{R}^d$ of affinely independent points is the set of convex combinations

$$\sum_{i=0}^{k} \lambda_i p_i, \text{ with } \sum_{i=0}^{k} \lambda_i = 1 \text{ and } \lambda_i \geq 0.$$

A face of σ is a non-empty subset of P.

Definition 2.0.2. A (finite) simplicial complex $K \in \mathbb{R}^d$ is a (finite) collection of simplices such that:

1. Any face of a simplex of K is a simplex of K.
2. The intersection of any two simplices of K is either empty or a common face of both.

The underlying space of K, denoted by $|K| \subset \mathbb{R}^d$ is the union of the simplices of K.

Intuitively, homology is a way of associating a sequence of algebraic objects with topological space, with an original motivation of distinguishing two shapes by its holes. This method can be applied to data by constructing a chain of k-simplex on it and derive the underlying topological features.

Example 2.0.1.

- A 0-hole is a connected component.
- A 1-hole is a loop or circular holes, a circle has one 0-hole and one 1-hole.
- A 2-hole is a cavity or void, a torus has one 2-hole, two 1-holes and one 0-hole.

Unfortunately, there is no rigorous definition of a hole within topology so to accurately compute the number of holes created when we build our simplicial complex on the data, we need some extra definitions.

Definition 2.0.3. A group is a set G equipped with a binary operation, usually called "+", that follows the following axioms:

- For all $a, b \in G$, $a + b \in G$.
- For all $a, b, c \in G$, $(a + b) + c = a + (b + c)$.
- There exists a unique element $e \in G$ such that, for every $a \in G$, $a + e = e + a = a$. Such an element is called the identity.
- For each $a \in G$, there exists a unique element $b \in G$ such that $a + b = e = b + a$ with e being the identity. Such an element b is called the inverse of a.

If for every $a, b \in G$, $a + b = b + a$ then G is called an abelian group.
The identity of a group G is often denoted as 0.

Definition 2.0.4. Given two groups $(G, +)$ and $(H, *)$, a group homomorphism from G to H is a function $f : G \to H$ such that for all $u, v \in G, h$, we have

$$h(u + v) = h(u) * h(v).$$

The kernel of a homomorphism f is denoted as

$$\ker f = \{g \in G \mid f(g) = e\}.$$

The image of a homomorphism f is denoted as

$$\operatorname{Im} f = \{f(g) \in H \mid g \in G\}.$$

A subgroup of G is a subset of G that is itself a group.

Definition 2.0.5. A subgroup $N \subset G$ is normal if and only if $gng^{-1} \in N$ for all $g, n \in G, N$.

Proposition 2.0.1. Given two groups G, H and a homomorphism $f : G \to H$, the kernel $\ker f$ and the image of f, $\operatorname{Im} f$ form normal subgroups of G.

Proof. [6] □

Definition 2.0.6. Let N be a normal subgroup of a group G. The set

$$G/N = \{aN \mid a \in G\}$$

together with the operation

$$(aN)(bN) = (ab)N$$

for $b \in G$, the identity element N and the inverse $a^{-1}N$ forms a group, the quotient group of G by N. It can be easily verified that G/N with the aforementioned operations form a group.

Definition 2.0.7. A group G can be called a free abelian group if it is an abelian group that has a basis i.e every element of G can be uniquely expressed as a linear combination of the basis elements.

Definition 2.0.8. A k-chain c is a finite formal sum of k-simplices in K written as

$$c = \sum_i a_i \sigma_i$$

with $a_i \in \mathbb{Z}_2$. Let c_o, c_1 be two k-chains then

$$c_0 + c_1 = \sum_i a_i \sigma_i + \sum_i b_i \sigma_i = \sum_i (a_i + b_i)\sigma_i.$$

Let $\sigma = [v_0, v_1, \ldots, v_k]$ be a k-simplex. The boundary of this simplex is the $(k-1)$-chain of σ. Denote ∂_k as

the k-boundary of a simplex then

$$\partial_k \sigma = \sum_{i=0}^{k} (-1)^i [v_0, ..., \hat{v_i}, ..., v_k]$$

where $\hat{v_i}$ is the point being excluded from the sequence $v_0, ..., v_k$.

Example 2.0.2. Given [7] a triangle with vertex set $\{v_0, v_1, v_2\}$ with a counter clockwise orientation, its boundary is

$$\partial_k [v_0, v_1, v_2] = [v_1, v_2] - [v_0, v_2] + [v_0, v1]$$

Note that since we defined the coefficients of a k-chain to be in $\mathbb{Z}_2$, $-1 \equiv +1$.

Proposition 2.0.2. Let K be a simplicial complex and C_k the set of k-chains over K. The set C_k with the operator "+" forms a free abelian group.

Proof. [7] □

Definition 2.0.9. Given a simplicial complex K, the boundary map $\partial_k : C_k \to C_{k-1}$ is a homomorphism that assigns each simplex to its boundary.

Lemma 2.0.1. Fundamental lemma of homology The boundary of a boundary is 0 i.e

$$\partial_k(\partial_{k+1}(\sigma)) = 0.$$

Proof. One can opt for a more general approach [8] for k-simplex or apply induction with a simple observation that for $\sigma = [v_0, v_1, v_2]$,

$$\begin{aligned} \partial_2(\partial_3(\sigma)) &= \partial_2([v_1, v_2] - [v_0, v_2] + [v_0, v_1]) \\ &= [v_2] - [v_1] - [v_2] + [v_0] + [v_1] - v[0] \\ &= 0. \end{aligned}$$

□

Proposition 2.0.3. The boundary map ∂_k is a homomorphism

Proof. It is trivial to see that ∂_k is a homomorphism if one follows the example given above with the fact that C_k is a free abelian group. □

A k-cycle z is a k-chain with boundary zero. It is easy to see that the set of all the k-cycles forms a subgroup of C_k, denote said group as $Z_k = \ker(\partial_k)$ and the group of the k-boundaries of K as $B_k = \mathrm{Im}(\partial_{k+1})$ then we can define the k^{th} homology group of K as [9]

$$H_k(K) = Z_k / B_k.$$

Given this homology group, we can find the topological features using a number called Betti number.

Definition 2.0.10. The rank of a group G is the smallest cardinality of a generating set for G,

$$\mathrm{rank}(G) = \min\{|X| \mid X \subset G, X \text{ generates } G\}$$

The k^{th} Betti number β_k is the rank of the k^{th} homology group H_k.

One of the reason why the Betti number is important is that it is a topological invariant which means that it is invariant under homeomorphism (bijective continuous map) [10]. In application, this can mean that under perturbations, the Betti number is less affected. Thus far, we have only discussed the theoretical aspect of persistent homology without the practical parts, the following will shed some light on it.

Definition 2.0.11. Given a simplicial complex K, a filtration of K is a finite sequence

$$\emptyset = K_0 \subset K_1 \subset ...K_{n-1} \subset K_n = K$$

such that for each i, K_i is a valid simplicial subcomplex of K.

In practice [**TDA**], a filtering function f is often used to induce a filtration on K.

Definition 2.0.12. Let $m \in \mathbb{N}$, a filtering function [11] is a function $f : K \to \{1, 2, ..., m\}$ such that, for each k-simplex σ and each $0 \leq i \leq k$, it holds that $f(\hat{\sigma}_i) \leq f(\sigma)$ with $\hat{\sigma}_i$ the convex hull of all the vertices of σ_i but v_i. For each $0 \leq p \leq m$, the simplicial subcomplex K^p is defined by

$$K^p := \{\sigma \in K \mid f(\sigma) \leq m.\}$$

Persistent homology is a tool that describes the change in homology that occur to an object which evolves with respect to a parameter [12] so given a dataset, we can build a filtration of simplicial complexes and determine the changes in homology through the building process.
Given $p \leq q \in \mathbb{N}$, the (p, q)-persistent k-homology group $H_k^{p,q}(K)$ consists of the k-cycles included from $C_k(K^p)$ into $C_k(K^q)$ modulo the boundaries. It can be formally defined as

$$H_k^{p,q}(K^f) := \mathrm{Im}(i_k^{p,q})$$

where $i_k^{p,q}$ denotes the linear map between $H_k(K^p)$ and $H_k(K^q)$ induced by the inclusion of complexes between K^p and K^q. With this definition, persistent homology allows for the retrieval of cycles that are non-boundary in a certain step of the filtration and that will turn into boundaries in some subsequent step. The persistent of a cycle i.e "how long a cycle lasts" during the filtration gives quantitative information about the relevance of the cycle itself for the shape [9]. The persistent homology can be often visualized using a tool called persistent diagram.

Example 2.0.3.

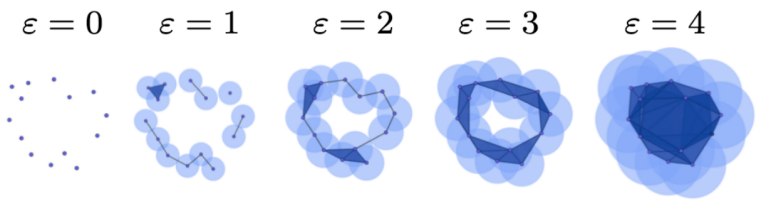

Figure 3: Example of building a simplicial complex

Given a set of points shown in the figure, for each point we construct a ball of radius ϵ around it and gradually increase ϵ. Then we can construct simplicial complex using different methods such as the Vietoris-Rips complex [13] then find the persistent homologies.

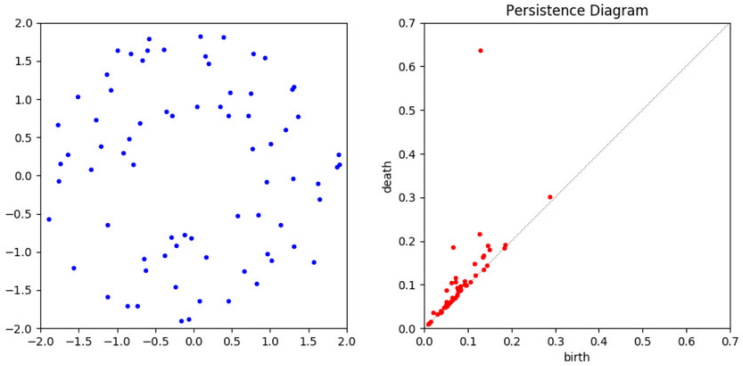

Figure 4: Example of a persistent diagram [14]

A persistent diagram is a set of points $\{(x, y) \mid x, y \in \mathbb{R}^2, y > x\}$ where each point corresponds to a topological feature in a corresponding family of simplicial complexes. In particular, each point (x, y) denotes a topological feature being born at radius x and dying at radius y.
Given two different sets of points with two different persistent diagrams, it is guaranteed by the stability theorem [15] i.e persistence diagrams are stable under possibly irregular perturbations of small amplitude. This shows

the viability of persistent homology methods in feature extraction for ML methods.

Methods

3.1 Dataset

We use the Fingers dataset from Kaggle [16, 17], with the first dataset containing both left and right hand images distributed into a training set of 18,000 images and a test set of 3600 images. Each image shows a certain number of fingers with the respective hand in the same orientation, described by the last two characters of the file name. L/R indicates the left or right hand and $0, 1, 2, 3, 4, 5$ indicates the number of fingers.
Additionally, in the second dataset [17], there are roughly 1000 training images and 100 test images, each image is 128x128 pixels with a grayscale filter, a consistent noise pattern in the background and the hands are within the center of the images. In [16], there are more noises as the colors in the background are somewhat different and the location of the hands changes from image to image.
For the purpose of this report, we do not take into account the orientation of the hand but only the number of fingers showing.

3.2 Data Pre-processing

For data pre-processing, we leverage the DataSet class from PyTorch to create a custom class that handles the loading and preprocessing of the data. First, the raw images are converted into numpy arrays and then torch tensors. Each image is accompanied with a label so the tensors for the images and labels have the shape of $(n \times h \times w)$ and $(n \times 1)$ where n is the number of images, h, w represent the height and width respectively. Our dataset contains both the images and the labels. Following the convention given in [18], we normalized the images to train using a pre-trained DenseNet model. The normalization is done as follows: [19]

$$\text{output[channel]} = \frac{(\text{input[channel]} - \text{mean[channel]})}{\text{std[channel]}}.$$

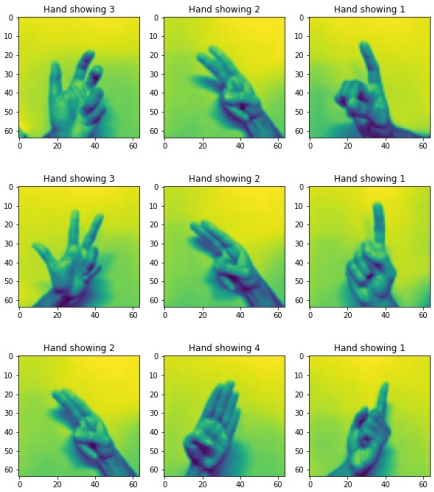

Figure 5: Dataset 1 after preprocessing

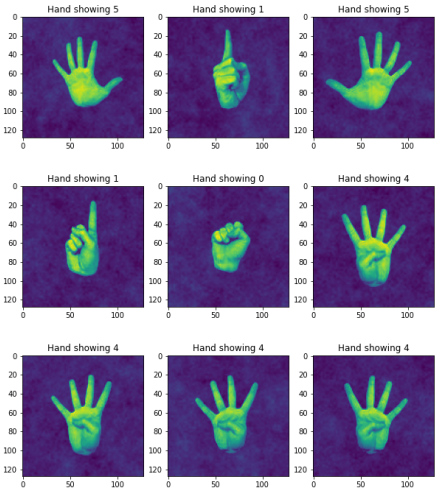

Figure 6: Dataset 2 after preprocessing

3.3 Implementation

Now that the data is processed, we can start building the models. In addition to Section 2, we discuss how the three CNN models were implemented with the addition of persistence homology and observe the performance of each model.

3.3.1 Models

Since we are using pre-trained models of AlexNet and DenseNet from PyTorch, the implementations for both of them are pretty much the same, so we shall combine them into one section. For all models, optimization algorithm used is Adam. The results of the models with different learning rates are shown in the experimentation section.

AlexNet and DenseNet

The models were loaded in using torchvision with their non-fully connected layers frozen. The last fully connected layers of both were replaced with another fully connected layer because of the difference in the number of output.

IzolNet

It is important to note that both AlexNet and DenseNet were made with the complexity of the ImageNet dataset which has 1000 classes and is undoubtedly significantly more complex than the current one. Thus, it is quite impractical to only use either of those and conclude that they are the best models. It is imperative that a smaller model be constructed that can roughly yield the same accuracy as the other two and with that objective in mind, a relatively simple CNN called IzolNet (named after a colleague) was made.
The architecture of this model is relatively simple. It composes of (through trial and error) four "Sequential" layers which are just a sequence of a convolutional layer, ReLU, batch normalization, max pool and drop out repeated four times. To use this layer, the input image has to be center cropped to the size of 64×64.
Note that this particular architecture is by no mean the optimal one time-performance wise but given the satisfactory results obtained by it, we chose to keep it as is. Every sequence of layers starting with the convolution one as shown in Figure 10 ends with a dropout layer with a parameter of 0.2 to prevent overfitting.

3.3.2 Persistent homology supervised learning pipeline

This implementation is inspired by [20] and [21]. We define a simple machine learning pipeline as follows

1. Binarize the image: convert each pixel to either 0 or 1.
2. Apply a filtration function.
3. Build a persistence diagram and rescale it.
4. Apply metrics (such as the heat kernel) to get the amplitude using L_1 or L_2 norms.

The amplitude of a persistence diagram is defined as its distance from an empty one i.e only contain diagonal points [20]. The reason we apply the heat kernel is that the persistent diagram with the Wasserstein distance only forms a metric space thus making them unsuitable for ML algorithms such as PCA or SVM, which operates in Hilbert space [22] ($\mathbb{R}^n$ is a Hilbert space with the usual notion of inner products and Euclidean distance). Interested readers can find the definition of a Hilbert space in [23] but as a short summary, a Hilbert space is a vector space with a norm and an inner product. The results from each of these steps can be found in the Appendix 11. The extracted features from the pipeline are then fed to a RandomForest classifier. The results are shown in Section 4 (Results).

Results

In this section we report the accuracy for all models presented earlier, the accuracy after using TDA tools and our choice of hyperparameters. For the sake of brevity, we made a simplifying assumption. Given a neural network model and two somewhat similar datasets with the same target classes, we assume that the model's performance using the dataset with less noise and more uniform data is going to be higher than the other one. Based on this intuition, we are going to test the model on the second dataset [17] and use that performance as our lower bound.

4.0.1 Neural network based models

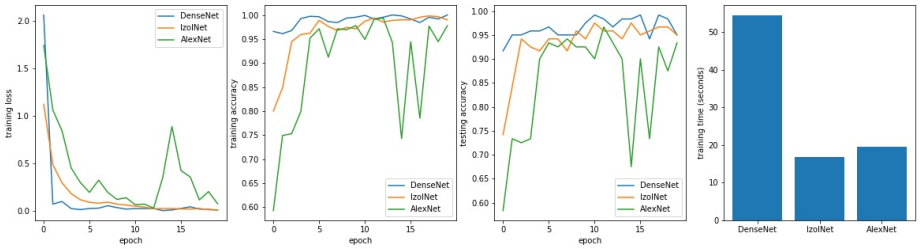

Figure 7: Comparison of different models

From Figure 7, it can be seen that the most viable and practical model is IzolNet because its performance is almost as good as the best performing one, DenseNet, but with significantly less training time (using CUDA with Tesla K8 GPU). The chosen hyperparameters are:

Hyperparameter	Value
Training batch size	64
Testing batch size	32
Learning rate	0.001
Drop out rate (IzolNet)	0.2
Optimizer	Adam

We found that these hyperparameters are (somewhat) the optimal ones and any major changes would not necessarily improve the performance.

4.0.2 Persistence homology based model

Using a pipeline as proposed in [21] which combines different filtration functions and metrics, for each (preprocessed) image of size 64×64, the derived features is a vector of size (1×476). For experimental purpose, we only used the first 500 images for the pipeline which, after feature extractions, lead to a training set of size (500×476) and after being fed to a Random Forest classifier, the obtained accuracy was 0.76, a relatively high score. It is important to note that not all the 476 derived features are informational, so it might be necessary to assign weights to each of them but for the purpose of this report, we neglected such tasks with regret.

Conclusion

Out of all the models tested, IzolNet had the best performance time-accuracy wise for the simple reason that it is just complicated enough to capture the important features of the data unlike DenseNet and AlexNet. Our topological data analysis approach to this using persistence homology (PH) yielded a satisfactory result with an accuracy score of 0.76 as this method is relatively new and rarely applied in practical settings. Also, it is clear that in this context, the theoretical part of PH is much more interesting as it offers a new perspective on these deep learning related tasks. On the other hand, the application side is not so interesting because the libraries that these methods are implemented on, utilize CPU instead of GPU thus making them significantly slower than neural nets.

Further Research

Although adaptive gradient optimizers such as Adam are regularly used for training feed forward and recurrent neural networks, it had been observed that they did not generalize as well as methods that use stochastic gradient descent with momentum such as AdamW for many deep learning tasks [24]. Researchers found that a reason may be due to the implementation of L2 regularization in many deep learning libraries when rather weight decay should have been used for adaptive gradient optimizers [25].

Additionally, Smith et al. reported a phenomenon called 'super-convergence' where training a neural network with one learning rate cycle and a large maximum learning rate could improve training times [26]. Applying this with Adam [26] allows adaptive gradient optimizers to solve the problem mentioned earlier and allows one to use these optimizers to their full potential [27].

It would be interesting to implement such methods as well as observe the difference when using Adam with super-convergence against AdamW with super-convergence.

On the side of topological data analysis, additional research into minimizing the number of features and applying other methods such as Mapper [28] must be made to fully utilize the potential of TDA.

Bibliography

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012).

[2] Michael Nielsen. *How the backpropagation algorithm works*. 2019. URL: http://neuralnetworksanddeeplearning.com/chap2.html.

[3] *CS231n Convolutional Neural Networks for Visual Recognition*. URL: https://cs231n.github.io/convolutional-networks/.

[4] Sumit Saha. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. 2018. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[5] Gao Huang et al. "Densely connected convolutional networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708.

[6] David S. Dummit and Richard M. Foote. *Abstract algebra*. 3rd ed. Wiley, 2004.

[7] Marc Khoury. *Introduction to Simplicial Homology*. URL: http://web.cse.ohio-state.edu/~wang.1016/courses/788/Lecs/lec6-marc.pdf.

[8] Samuel Eilenberg. *Foundations of Algebraic Topology*. 1952.

[9] S. Scaramuccia U. Fugacci. *Persistent homology: a step-by-step introduction for newcomers*. 2016. URL: https://www.math.uri.edu/~thoma/comp_top__2018/stag2016.pdf.

[10] Margherita Barile and Eric W Weisstein. *Betti number*. URL: https://mathworld.wolfram.com/BettiNumber.html.

[11] Federico Iuricich. *Persistent homology: An introduction via interactive examples*. URL: https://iuricichf.github.io/ICT/filtration.html.

[12] Herbert Edelsbrunner and John Harer. *Computational Topology - an Introduction*. American Mathematical Society, 2010, pp. I–XII, 1–241. ISBN: 978-0-8218-4925-5.

[13] Afra Zomorodian. "Fast construction of the Vietoris-Rips complex". In: *Computers Graphics* 34.3 (2010). Shape Modelling International (SMI) Conference 2010, pp. 263–271. ISSN: 0097-8493. DOI: https://doi.org/10.1016/j.cag.2010.03.007. URL: https://www.sciencedirect.com/science/article/pii/S0097849310000464.

[14] Omer Bobrowski and Matthew Kahle. "Topology of random geometric complexes: a survey". In: *Journal of Applied and Computational Topology* 1 (June 2018). DOI: 10.1007/s41468-017-0010-0.

[15] Edelsbrunner Cohen-Steiner D. *Stability of Persistence Diagrams*. 2007, pp. 103–120. DOI: https://doi.org/10.1007/s00454-006-1276-5. URL: https://link.springer.com/article/10.1007/s00454-006-1276-5.

[16] *Starter: Finger Signs Detection Dataset*. URL: https://www.kaggle.com/code/maneesh99/starter-finger-signs-detection-dataset/data.

[17] Pavel Koryakin. *Fingers*. URL: https://www.kaggle.com/datasets/koryakinp/fingers.

[18] *Pytorch Vision DenseNet*. URL: https://pytorch.org/hub/pytorch_vision_densenet/.

[19] *Normalize*. URL: https://pytorch.org/vision/main/generated/torchvision.transforms.Normalize.html.

[20] Ad'elie Garin and Guillaume Tauzin. "A Topological "Reading" Lesson: Classification of MNIST using TDA". In: *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)* (2019), pp. 1551–1556.

[21] *Classifying handwritten digits*. URL: https://giotto-ai.github.io/gtda-docs/0.5.1/notebooks/MNIST_classification.html.

[22] Jan Reininghaus et al. *A Stable Multi-Scale Kernel for Topological Machine Learning*. 2014. DOI: 10.48550/ARXIV.1412.6821. URL: https://arxiv.org/abs/1412.6821.

[23] Eric W Weisstein. *Hilbert Space*. URL: https://mathworld.wolfram.com/HilbertSpace.html.

[24] Ashia C. Wilson et al. *The Marginal Value of Adaptive Gradient Methods in Machine Learning*. 2017. DOI: 10.48550/ARXIV.1705.08292. URL: https://arxiv.org/abs/1705.08292.

[25] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. 2017. DOI: 10.48550/ARXIV.1711.05101. URL: https://arxiv.org/abs/1711.05101.

[26] Leslie N. Smith and Nicholay Topin. *Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates*. 2017. DOI: 10.48550/ARXIV.1708.07120. URL: https://arxiv.org/abs/1708.0o7120.

[27] Sylvain Gugger and Jeremy Howard. 2018. URL: https://www.fast.ai/2018/07/02/adam-weight-decay/#appendix-full-results.

[28] Gurjeet Singh, Facundo Memoli, and Gunnar Carlsson. "Topological Methods for the Analysis of High Dimensional Data Sets and 3D Object Recognition". In: *Eurographics Symposium on Point-Based Graphics*. Ed. by M. Botsch et al. The Eurographics Association, 2007. ISBN: 978-3-905673-51-7. DOI: 10.2312/SPBG/SPBG07/091-100.

[29] *Neural Network*. URL: https://databricks.com/glossary/neural-network.

Appendix

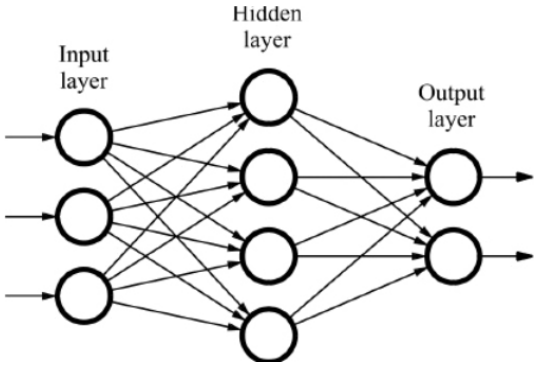

Figure 8: Example of a neural network [29]

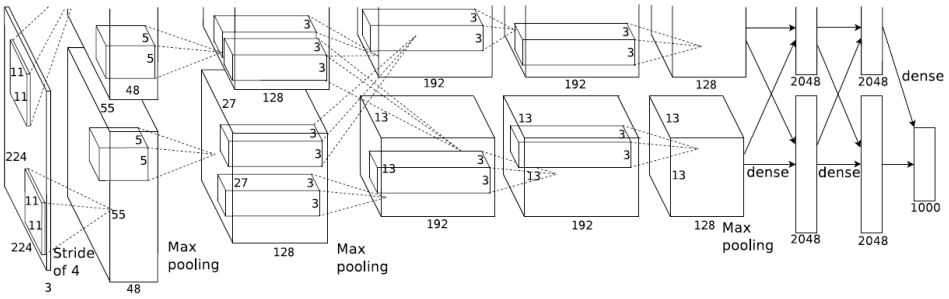

Figure 9: AlexNet Architecture

```
Sequential(
  (0): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (1): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (2): Sequential(
    (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (3): Sequential(
    (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (4): Flatten(start_dim=1, end_dim=-1)
  (5): Linear(in_features=512, out_features=128, bias=True)
  (6): Sigmoid()
  (7): Linear(in_features=128, out_features=6, bias=True)
```

Figure 10: IzolNet

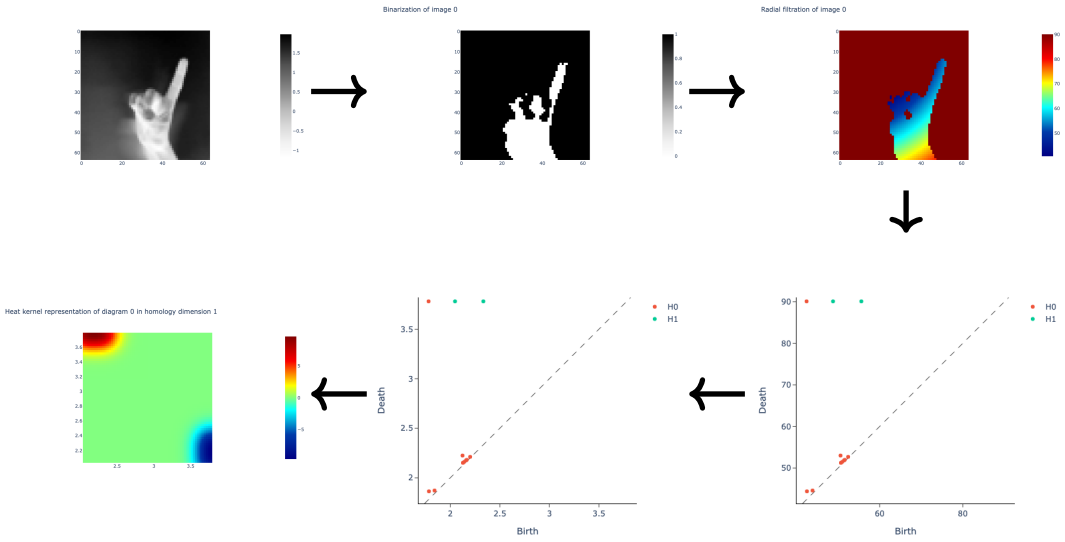

Figure 11: Persistent homology pipeline