Improved Neural Network Generalization using Channel-Wise NNK Graph Constructions

Degree Thesis submitted to the Faculty of the
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya
by

David Bonet Solé

In partial fulfillment
of the requirements for the degree in
Telecommunications Technologies and Services Engineering

Advisors:
Antonio Ortega (USC Viterbi)
Javier Ruiz-Hidalgo (UPC ETSETB)
Sarath Shekkizhar (USC Viterbi)

Barcelona, June 2020

Abstract

State-of-the-art neural network architectures continue to scale in size and deliver impressive results on unseen data points at the expense of poor interpretability. In the deep layers of these models we often encounter very high dimensional feature spaces, where constructing graphs from intermediate data representations can lead to the well-known curse of dimensionality. We propose a channel-wise graph construction method that works on lower dimensional subspaces and provides a new channel-based perspective that leads to better interpretability of the data and relationship between channels. In addition, we introduce a novel generalization estimate based on the proposed graph construction method with which we perform local polytope interpolation. We show its potential to replace the standard generalization estimate based on validation set performance to perform progressive channel-wise early stopping without requiring a validation set.

Resum

Les arquitectures de xarxes neuronals més avançades segueixen augmentant la seva mida i oferint resultats impressionants en noves dades a costa d'una escassa interpretabilitat. A les capes profundes d'aquests models ens trobem sovint amb espais de característiques de molt alta dimensió, en què la construcció de grafs a partir de representacions de dades intermèdies pot portar al conegut *curse of dimensionality*. Proposem un mètode de construcció de grafs per canal que treballa en subespais de menor dimensió i proporciona una nova perspectiva basada en canals, que porta a una millor interpretabilitat de les dades i de la relació entre canals. A més, introduïm un nou estimador de generalització basat en el mètode de construcció de grafs proposat amb el qual realitzem interpolació local en polítops. Mostrem el seu potencial per substituir l'estimador de generalització estàndard basat en el rendiment en un set de validació independent per a realitzar *early stopping* progressiu per canals i sense necessitat d'un set de validació.

Resumen

Las arquitecturas de redes neuronales más avanzadas siguen aumentando en tamaño y ofreciendo resultados impresionantes en nuevos datos a costa de una escasa interpretabilidad. En las capas profundas de estos modelos nos encontramos a menudo con espacios de características de muy alta dimensión, en los que la construcción de grafos a partir de representaciones de datos intermedias puede llevar al conocido *curse of dimensionality*. Proponemos un método de construcción de grafos por canal que trabaja en subespacios de menor dimensión y proporciona una nueva perspectiva basada en canales, que lleva a una mejor interpretabilidad de los datos y de la relación entre canales. Además, introducimos un nuevo estimador de generalización basado en el método de construcción de grafos propuesto con el que realizamos interpolación local en politopos. Mostramos su potencial para sustituir el estimador de generalización estándar basado en el rendimiento en un set de validación independiente para realizar *early stopping* progresivo por canales y sin necesidad de un set de validación.

Acknowledgements

First and foremost, I would especially like to thank Prof. Antonio Ortega for his exceptional guidance and advice during this research project. Despite the situation caused by the COVID-19 pandemic, he gave me the opportunity to conduct my thesis in his group, which I am truly grateful for. Special thanks to Prof. Javier Ruiz-Hidalgo and Sarath Shekkizhar for their advice and help during the whole project. Remote work has been a challenge, but all of you have made me feel very welcome and it has been a really enjoyable experience.

Finalment, voldria agrair i dedicar aquest treball a la meva família i amics, pel seu amor i suport incondicional. Amb especial menció als meus pares, per ajudar-me sempre que ho he necessitat, donant-me l'oportunitat de dedicar-me al que m'agrada sense haver de preocupar-me de res més.

Revision history and approval record

Revision	Date	Purpose
0	14/05/2021	Document creation
1	20/06/2021	Document revision
2	21/06/2021	Document approval

DOCUMENT DISTRIBUTION LIST

Name	e-mail
David Bonet Solé	davidbonetsole@gmail.com
Antonio Ortega	ortega@sipi.usc.edu
Javier Ruiz-Hidalgo	j.ruiz@upc.edu
Sarath Shekkizhar	shekkizh@usc.edu

Written by:		Reviewed and approved by:	
Date	14/05/2021	Date	21/06/2021
Name	David Bonet Solé	Name	Javier Ruiz-Hidalgo
Position	Project Author	Position	Project Supervisor

Contents

List of Figures

List of Tables

Acronyms

CW Channel-Wise

DNN Deep Neural Network

DR Distance Ratio

GPU Graphics Processing Unit

GSP Graph Signal Processing

ID Intrinsic Dimension

K-NN K-Nearest Neighbors

KRI Kernel Ratio Interval

LLE Local Linear Embedding

LOO Leave One Out

NNK Non-Negative Kernel

OS Overlap Score

ReLU Rectified Linear Unit

Chapter 1

Introduction

1.1 Motivation

Graphs play an important role in many machine learning applications, and are mainly used to model data structures and similarities in a dataset [2]. Powerful properties such as their compositional nature and ability to define relationships between information pieces make graphs the tool of choice for representing a wide variety of rich and complex data [3], e.g. chemical compounds, social networks, buying behaviours or 3D point clouds. This and the increasing availability of computational resources has motivated a recent surge in interest in processing graphs with deep learning models. Graph Neural Networks [4] and recently Graph Convolutional (Neural) Networks [5] are some of the predominant models that have overcome the big challenges of processing graphs in an adaptive fashion, including expressiveness and computational complexity, compared to the standard processing with vectorial data. Other works use graphs as a tool to understand the topology of intermediate data representations of neural networks. Specifically, they can be used for various tasks, such as, interpretation of how the model is learning [1, 6], enforcement of desirable properties on intermediate representations [7], improvement of knowledge distillation [8] or increased model robustness [9].

In the intermediate layers of deep neural networks we find feature vectors, i.e., learned representations of the input data that are suitable for the task at hand. In the specific case of convolutional neural networks, we encounter very high dimensional feature spaces, formed by the aggregation of lower dimensional channels. Graph construction methods based on node similarity such as local linear embedding (LLE) [10] and non negative kernel (NNK) regression graphs [11] can capture valuable information about the relative position of data points in these feature spaces, providing useful insights for the various above mentioned tasks. However, neural networks are being designed with an increased number of parameters in pursuit of higher accuracies [12, 13, 14]. This poses a problem because it becomes more difficult to extract relevant information from graph constructions in high dimensions, owing to the *curse of dimensionality* leading to poorer interpretability in these higher dimensional feature spaces.

A second problem in the deep learning field is that although state-of-the-art models have been shown to generalize very well to new data, our understanding of why this happens is somewhat limited [15, 16]. To achieve good generalization results, different explicit or implicit regularization methods are used to avoid overfitting the model to the training data [17], including the widely used early stopping method [18] which consists of stopping model optimization based on continuous monitoring of a selected metric performance on a separate validation set. However, this practice requires selecting part of the labeled data for a validation set, which may introduce biases or lead to lower performance when little labeled data is available [19].

1.2 Main Contributions

The main contributions of this thesis are:

1. We propose a novel channel-wise approach to graph construction based on NNK graphs, which we call CW-NNK graphs, that tackles the high dimensional graph construction problem by acting on well-defined feature subspaces of lower dimensionality.
2. We study the influence of dimension and the relationship between channels with our proposed channel-wise graph construction based on local polytope geometry, providing better interpretability with interesting geometric properties.
3. We develop a channel-wise extension of DeepNNK [1] (CW-DeepNNK), a non-parametric local interpolation framework that induces instance-based explainability for deep learning models.
4. We present a new generalization estimate to perform channel-wise progressive early stopping without the need for a validation set.

1.3 Overview

The structure of this thesis is as follows.

In Chapter 2 we introduce the basic concepts of Graph Signal Processing (GSP), and a review of the state-of-the-art graph construction methods based on node similarity. We also provide a brief introduction to Convolutional Neural Networks as they are the application focus of this work.

In Chapter 3 we discuss the *curse of dimensionality* and we present the first two main contributions. We develop an in-depth analysis of interesting properties of NNK graphs that provide a better understanding the low intrinsic dimensionality of real data and the relationships between channels.

In Chapter 4 we introduce a new channel-wise perspective of generalization, based on graph local geometry properties and the last two main contributions. We also compare our proposal with the standard method to perform early stopping.

In Chapter 5 we detail the budget to develop the project.

Finally, we conclude by discussing the main findings and open challenges for future work in Chapter 6.

1.4 Work plan

The work on this thesis was organized according to the following schedule:

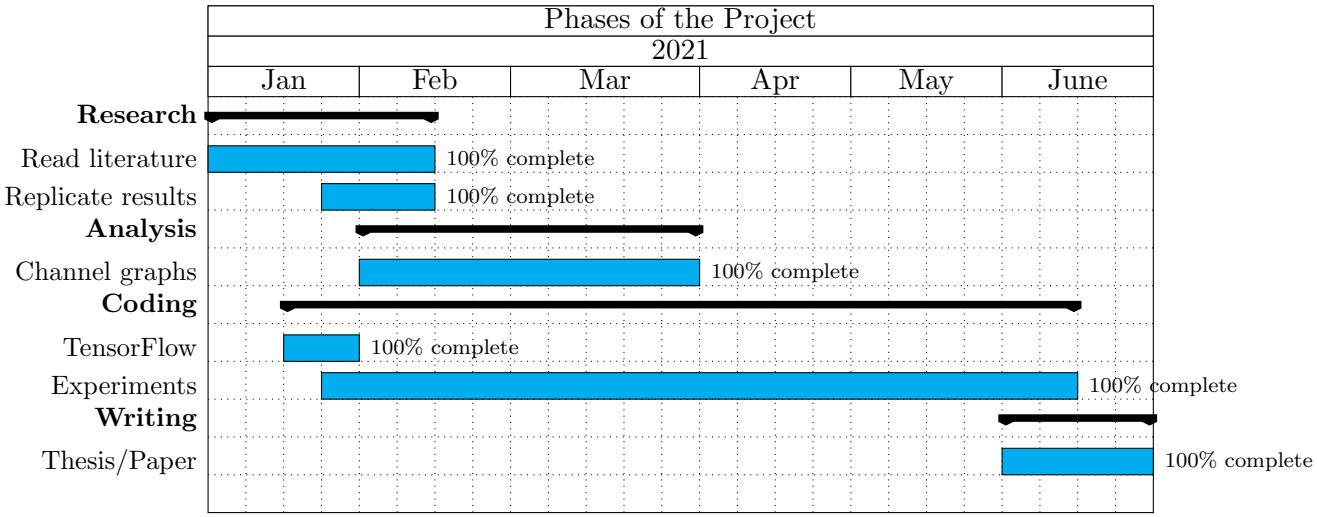


Figure 1.1: Gantt diagram of the project.

There have been no major changes with respect to the initial planning. The beginning of the project was very exploratory, understanding this new perspective of building channel-wise graphs in depth and analyzing the different paths it could open. At the experimental level, in the last three months we finished defining the final objectives of the project and how we could use this new method to improve interpretability and generalization in neural networks. Finally, the last month has been mostly devoted to writing this thesis report and preparing a paper submission for a signal processing conference.

Chapter 2

Fundamentals

2.1 Graph Signal Processing

We start by introducing basic concepts and notation of Graph Signal Processing (GSP) [20]. GSP is the study of how to analyze and process data associated with graphs.

2.1.1 Basic definitions

A **graph** $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a discrete structure that consists of a set of nodes, $\mathcal{V} = \{v_1, v_2, \dots, v_N\}$ and a set of edges, $\mathcal{E} = \{e_1, e_2, \dots, e_M\}$. The graph is **weighted** if real positive weights w_{ij} are associated with each edge e_{ij} that connects nodes i and j, or **unweighted** if all edges have weight equal to 1. If two nodes i and j are not connected, then $w_{ij} = 0$. A graph is **undirected** if e_{ij} exists whenever e_{ji} exists and $w_{ij} = w_{ji}$. Conversely, a graph is **directed** if e_{ij} or e_{ji} may not exist and in general $w_{ij} \neq w_{ji}$. The **adjacency matrix** $\boldsymbol{W}$ of the graph is an $N \times N$ matrix that captures all connectivity and edge weight information, with $\boldsymbol{W}_{ij} = w_{ij}$.

A graph with N nodes is considered **dense** if the number of edges per node is close to N, and **sparse** if the number of edges incident to any node is much smaller than N.

The **graph signal** is composed of N data points with feature vectors $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N\} \in \mathbb{R}^D$. Each feature vector $\boldsymbol{x}$ has dimension D and can be viewed as the aggregation of S lower-dimensional subvectors $\boldsymbol{x}_i^s \in \mathbb{R}^{D_s}$ where $\sum_{s=1}^{S} D_s = D$:

$$\boldsymbol{x}_i = \begin{bmatrix} \boldsymbol{x}_i^1 \\ \boldsymbol{x}_i^2 \\ \vdots \\ \boldsymbol{x}_i^S \end{bmatrix} \in \mathbb{R}^D \tag{2.1}$$

In this work, nodes represent data points that are part of a dataset, and the edges represent similarity between data points. In a supervised classification setting, each data point belongs to a category, i.e., it has a label associated to it. The goal is for a model to learn a function that best approximates to which category a new sample of data belongs to. As the feature vectors corresponding to each data point we can use either the features at the input of the model (e.g., original images of the training set) or the learned intermediate representations in convolutional layers, so that the separation of the features into channels is natural and complete.

2.1.2 Similarity-based Graph Construction Methods

We can define a notion of node similarity based on a pairwise similarity metric, e.g., distance, $d(i,j) = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|$. If good feature vectors have been chosen to represent the data, it is expected that the labels of points that are close to each other are more likely to be the same. This would result in a label signal that is smooth on the similarity graph, i.e., where large w_{ij} are generally associated with nodes that fall into the same category, i.e., $d(i,j)$ small. A common method to do this is to define edge weights based on the Gaussian kernel:

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{d(i,j)^2}{2\sigma^2}\right) \tag{2.2}$$

where σ is the bandwidth of the Gaussian Kernel. Another option is to use the range normalized cosine kernel:

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{1}{2}\left(1 + \frac{\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle}{\|\boldsymbol{x}_i\|\|\boldsymbol{x}_j\|}\right) \tag{2.3}$$

Note that, if we directly select all $w_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, both kernels will produce a complete weighted graph, since it is very unlikely to obtain exactly zero weights. In order to obtain a sparser graph with fewer connections we can apply different optimizations, which we discuss next.

Neighborhood optimization

Weighted K-Nearest Neighbor (K-NN) graphs [21] and ϵ-neighborhood graphs (ϵ-graphs) [22] are among the the most commonly used graph construction methods. K-NN graphs are constructed by connecting every node $v \in \mathcal{V}$ to its K most similar nodes in $\mathcal{V}$. Thus, only K pairwise distances $d(i,j)$ are needed. If (2.2) is used as the similarity metric, K and σ have to be chosen. In ϵ-graphs we also have to choose the parameter ϵ.

The choice of parameters usually depends on the dataset distribution or the task at hand. Defining $d_{\min} = \min_{i,j}\|\boldsymbol{x}_i - \boldsymbol{x}_j\|$, if σ is much larger than $d_{\min}$, edge weights will collapse to 1. Conversely, if σ is much smaller than $d_{\min}$, most weights will be very small. A common solution is to choose σ based on statistics of local distances (e.g. minimum or average distance for each point in the dataset). For some tasks, the choice of K, σ or ϵ is usually heuristic or based on a cross-validation strategy to obtain the best performance in a task [23].

A downside of this methods is that they do not consider the possibility of two nodes being redundant. Other approaches such as kernel learning [24] and adaptive edge weighting (AEW) [25] try to optimize the weights of K-NN/ϵ-graphs to the data without modifying the connectivity. Alternative techniques described below address this problem by taking into account relative distances.

Local linear embedding (LLE)

In local linear embedding [10], for each node i, a K-NN search is done and $\boldsymbol{X}_{\mathcal{S}}$ is the matrix containing in columns the feature vectors of the K nearest neighbors of $\boldsymbol{x}_i$ whose

indices are denoted by set $\mathcal{S}$. Then, LLE solves:

$$\min_{\boldsymbol{\theta}:\, \boldsymbol{\theta}\geq 0} \|\boldsymbol{x}_i - \boldsymbol{X}_{\mathcal{S}}\boldsymbol{\theta}\|_2^2 \tag{2.4}$$

where the solution $\boldsymbol{\theta}$ corresponds to the weights of the edges connecting node i and all the nodes in $\mathcal{S}$. If two nodes $j, k \in \mathcal{S}$ are very close, they may be redundant in terms of linear approximation (2.4), resulting in w_{ij} or w_{ik} being zero. State-of-the-art graph construction methods [26, 27] also aim to be locality-inducing using explainable regularization techniques.

Non Negative Kernel (NNK) regression graphs

Positive definite kernels $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ such as (2.2) and (2.3) correspond to a transformation of data points in $\mathbb{R}^D$ to a non linearly transformed feature space $\mathcal{H}$ referred to as the reproducing kernel Hilbert space [28], such that similarities can be interpreted as dot products in this transformed space (generally known as the *kernel trick*), i.e., $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi_i^T \phi_j$, where $\phi : \mathbb{R}^D \to \mathcal{H}$ and ϕ_i represents the transformed observation of $\boldsymbol{x}_i$.

Non-negative kernel graph construction [11] also starts finding the K nearest neighbors of each node denoted by the set $\mathcal{S}$. Then, the NNK optimization at each node solves:

$$\boldsymbol{\theta}_{\mathcal{S}} = \min_{\boldsymbol{\theta}:\, \boldsymbol{\theta}\geq 0} \|\phi_i - \boldsymbol{\Phi}_{\mathcal{S}}\boldsymbol{\theta}\|_2^2 \tag{2.5}$$

where $\boldsymbol{\Phi}_{\mathcal{S}}$ contains the transformed neighbors. Using the *kernel trick*, the NNK problem can be rewritten as:

$$\boldsymbol{\theta}_{\mathcal{S}} = \underset{\boldsymbol{\theta}:\, \boldsymbol{\theta}\geq 0}{\operatorname{argmin}} \; \frac{1}{2}\boldsymbol{\theta}^T \boldsymbol{K}_{\mathcal{S},\mathcal{S}}\boldsymbol{\theta} - \boldsymbol{K}_{\mathcal{S},i}^T\boldsymbol{\theta} \tag{2.6}$$

where $\boldsymbol{K}_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Finally, the i-th row of the adjacency matrix $\boldsymbol{W}$ is given by $\boldsymbol{W}_{i,\mathcal{S}} = \boldsymbol{\theta}_{\mathcal{S}}$ and $\boldsymbol{W}_{i,\mathcal{S}^C} = \boldsymbol{0}$.

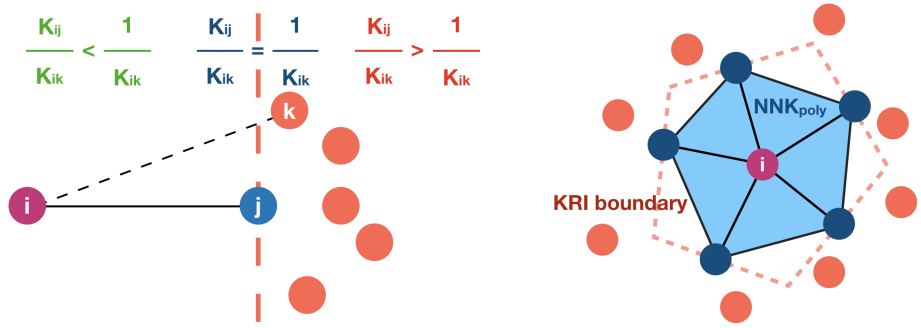
An advantage of NNK over other methods such as K-NN, which select the K largest inner products $\phi_i^T\phi_j$ and can be viewed as a thresholding-based representation, is its robustness to sparsity parameters such as K.

NNK also has a geometric interpretation based on the Kernel Ratio Interval (KRI) theorem: In a three-node scenario, for any positive definite kernel with range $[0, 1]$, the necessary and sufficient condition for two data points $\boldsymbol{x}_j$ and $\boldsymbol{x}_k$ to be connected to $\boldsymbol{x}_i$ in an NNK graph is

$$\boldsymbol{K}_{j,k} < \frac{\boldsymbol{K}_{i,j}}{\boldsymbol{K}_{i,k}} < \frac{1}{\boldsymbol{K}_{j,k}} \tag{2.7}$$

In words, the interval for both nodes to be connected to i is large if the two nodes, j and k, are dissimilar. But if the two nodes are very similar, the interval for both edges to exist is very small, and only one node of the two will be connected to the query node i.

In the case of the Gaussian kernel (2.2), considering the edge θ_{ij} connecting node i and node j, we can define a hyperplane with normal in the edge direction. As shown in Figure 2.1a this hyperplane divides the space in two, a region R_{ij} that contains $\boldsymbol{x}_i$, and



(a) (b)

Figure 2.1: (a) KRI hyperplane corresponding to connected neighbor $\boldsymbol{x}_j$. (b) KRI boundary associated to the convex polytope formed by NNK neighbors around $\boldsymbol{x}_i$. From [1], with permission.

its complement $\overline{R}_{ij}$. Then, the third node k will be connected to i only if $\boldsymbol{x}_k \in R_{ij}$. If $\boldsymbol{x}_k \in \overline{R}_{ij}$ in a three-node scenario, $\theta_{ik} = 0$ and we say that k has been eliminated by the hyperplane created by j.

Inductive application of the KRI connects to other points while producing a closed decision boundary around $\boldsymbol{x}_i$, i.e., the NNK optimization at each node constructs a convex polytope around node i disconnecting all the other points outside the polytope, see Figure 2.1b. The resulting set of NNK neighbors for each node is denoted by $\mathcal{N}$. Also note that $\mathcal{N} \subseteq S$.

2.2 Convolutional Neural Networks

Convolutional Neural Networks are one of the most popular types of Deep Neural Networks (DNN). They have multiple layers and they are often composed of convolutional layers, non-linearity layers, pooling layers and a fully-connected layer [29]. Input data of the model used usually consists of several channels, with each channel being the observation of a different quantity at some point in space or time [17]. Convolutional layers filter its multi-channel (aggregate of channels) input several times with multiple filters, commonly between 16 and 512, resulting also in a multi-channel output, where each channel captures different features of the input of the layer. Note that in each layer we encounter very high dimensional feature spaces, formed by the aggregation of large numbers of channels. In this work we use the terms "subvector" and "channel" interchangeably, since a convolutional channel is a subvector of a convolutional layer.

ReLU is the most used non-linear function, both for its function and gradient simplicity. Also for its capability to avoid "vanishing gradients" and to create sparser representations. Other activations functions such as *sigmoid* or tanh always obtain non-zero values, while ReLU collapse a lot of dimensions to complete zeros.

Chapter 3

CW-NNK graphs and their aggregation

Dealing with high dimensional data can lead to the well-known *curse of dimensionality* [30], which refers to the fact that volume in the space increases very fast with dimension, make available data sparse, unless the amount of data also increases significantly. Under these conditions, distances between points can become meaningless or not very informative [31], e.g., we might find that most of the points appear to be at similar distances with respect to each other. This is a problem to keep in mind when building graphs with edge weights based on node similarity, since if all the weights have similar values and we get a complete graph, it does not give us useful information about the data [20]. However, most state of the art machine learning models work with high dimensional data, and still obtain very competitive results [14]. This is because, in practice, the curse of dimensionality is effectively avoided [32, 33, 34]. The problem in this case is that it is not clear how the dimensionality problem is alleviated. Many theoretical results [35, 36] indicate that the data lies in a low-dimensional manifold and its *intrinsic dimension* (ID) is much lower than the nominal one, defining ID as the number of "degrees of freedom" that are necessary to generate the observed data. Other works try to estimate the ID to facilitate understanding and analysis in high dimensional spaces [37, 38, 39]. Nevertheless, we lack a clear understanding of how the manifold can be characterized in such high dimensional spaces. Most works analyze data feature vectors as a whole, i.e., as a single indivisible feature vector. In this work we seek a new perspective where we can understand the feature space as the aggregation of well-defined, low-dimensional feature subspaces. For example, we consider the activations of a convolutional layer of a neural network, where hundreds of outputs from different convolutional filters are aggregated to represent a single input image, as a concatenation of multiple subvectors, each associated with the output of one of the channels in the deep neural network.

In this thesis a feature *vector* corresponds to a complete representation of an item (e.g., an image). Each feature vector contains *subvectors*, each generated by a distinct computation in the feature extraction (e.g., the output of one of the channels comprising a convolution, non-linearity and pooling operations in a neural network). The feature vectors are contained in a feature *space*, were we can also refer to the independent *subspaces* induced by specific subvectors.

Shekkizhar and Ortega [11] observed the number of connections resulting from the NNK optimization, where redundant neighboring points are neglected, to be indicative of the ID of the data manifold. Thus, we take the average number of NNK neighbors as a proxy for ID. Constructing channel-wise NNK (CW-NNK) graphs we can study the ID in each subspace, which allows us to develop a better understanding of why the ID is much lower than what would have been predicted based on the overall dimension of the feature

vectors. In particular, CW-NNK graphs also allow us to derive overall interrelationships between independent feature subspaces from common neighboring NNK points.

The curse of dimensionality also implies a greater difficulty in terms of computation, since nearest neighbor methods can have an exponential dependence on dimensions when executing a search query [23]. We show how solving the NNK optimization per channel and combining the results allows skipping steps in the aggregate problem and reduce complexity. This holds not only in the case of convolutional layers, but in any scenario where we have defined channels or sub-features.

In real datasets or data representations not all dimensions have the same importance [40]: some of them play an important role for the specific task, while others are characterized by small variations that may be irrelevant to the analysis. In the specific case of neural networks, ID has been studied for full layer intermediate representations [33, 32]. This work, to the best of our knowledge, is the only one that studies ID by considering each of the channels separately and then aggregating information, to better explain the behaviour in individual channels. show how an in-depth analysis of the subvectors with CW-NNK graphs helps us to determine better the relative importance of the various dimensions, as compared to what can be achieved with a general approach. We analyze how there are dimensions that practically do not contribute to the task and how they imply a direct reduction of the ID.

Ultimately, our work tries to explain why even though systems of interest operate in high dimension we do not really suffer a performance penalty due to the *curse* of dimensionality.

3.1 *K*-NN analysis

From the channel-wise graph construction, we can have a better understanding of how it is possible for the intrinsic dimensionality of the data to be relatively low even in a high dimensional space. A theoretical analysis that builds on the information obtained from the NNK optimization in each of the independent channels allows us to better understand the relationship between the channels. From the sets of nearest neighbors and the NNK neighbors in the subvectors, we can study the relationship between the channels and infer relevant information and properties of the graph that we would obtain in the aggregate high dimensional space. For simplicity, we consider a scenario of two subvectors and their aggregate:

$$\boldsymbol{x}_i = \begin{bmatrix} \boldsymbol{x}_i^1 \\ \boldsymbol{x}_i^2 \end{bmatrix} \in \mathbb{R}^D$$

$$\boldsymbol{x}_i = (x_i(0), x_i(1), \ldots, x_i(D-1))^T = (x_i^1(0), \ldots, x_i^1(D_1-1), x_i^2(0), \ldots, x_i^2(D_2-1))^T$$

where $\boldsymbol{x}_i^1 \in \mathbb{R}^{D_1}$ and $\boldsymbol{x}_i^2 \in \mathbb{R}^{D_2}$ are the two defined subvectors of $\boldsymbol{x}_i$ and $D_1 + D_2 = D$, but all the results presented in this section can be extended to the multiple channel case.

The first step to build an NNK graph is to find the K nearest neighbors to node i (complexity of the algorithms we describe will be discussed in Section 3.3.5). As an

example, in the full space the query is $\boldsymbol{x}_i$ and $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\} \in \mathbb{R}^D$ are the training data points, excluding $\boldsymbol{x}_i$ from the training set. Given some norm $\|\cdot\|$ on $\mathbb{R}^D$, let $\boldsymbol{x}_{(1)}, \ldots, \boldsymbol{x}_{(K)}, \boldsymbol{x}_{(K+1)}, \ldots, \boldsymbol{x}_{(N)}$ be a reordering of the training data such that

$$\|\boldsymbol{x}_{(1)} - \boldsymbol{x}_i\| \leq \cdots \leq \|\boldsymbol{x}_{(K)} - \boldsymbol{x}_i\| \leq \|\boldsymbol{x}_{(K+1)} - \boldsymbol{x}_i\| \leq \cdots \leq \|\boldsymbol{x}_{(N)} - \boldsymbol{x}_i\|.$$

Note that

$$\|\boldsymbol{x}_{(1)} - \boldsymbol{x}_i\|^2 \leq \cdots \leq \|\boldsymbol{x}_{(K)} - \boldsymbol{x}_i\|^2 \leq \cdots \leq \|\boldsymbol{x}_{(N)} - \boldsymbol{x}_i\|^2$$

can also be written as

$$\|\boldsymbol{x}^1_{(1)} - \boldsymbol{x}^1_i\|^2 + \|\boldsymbol{x}^2_{(1)} - \boldsymbol{x}^2_i\|^2 \leq \cdots \leq \|\boldsymbol{x}^1_{(K)} - \boldsymbol{x}^1_i\|^2 + \|\boldsymbol{x}^2_{(K)} - \boldsymbol{x}^2_i\|^2 \leq \cdots \leq \|\boldsymbol{x}^1_{(N)} - \boldsymbol{x}^1_i\|^2 + \|\boldsymbol{x}^2_{(N)} - \boldsymbol{x}^2_i\|^2.$$

The indices of the K nearest neighbors to $\boldsymbol{x}_i$ in the aggregate space are denoted by set $\mathcal{S}_A$, while the the K nearest neighbors to $\boldsymbol{x}^1_i$ and $\boldsymbol{x}^2_i$ are denoted by $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively.

We now analyze which properties of the set $\mathcal{S}_A$ can be inferred from the sets $\mathcal{S}_1$ and $\mathcal{S}_2$. Then, we analyze the properties of the set of NNK neighbors $\mathcal{N}_A$ we can infer from $\mathcal{N}_1$ and $\mathcal{N}_2$. Assume $<$ instead of $\leq$ for simplicity.

Lemma 3.1.1. *If $j \notin \mathcal{S}_1$ and $j \notin \mathcal{S}_2$ then $j \notin \mathcal{S}_A$.*

Proof. Let us consider the edge case where for the sets $\mathcal{S}_1$ and $\mathcal{S}_2$, we have that $\boldsymbol{x}^1_k \in \mathcal{S}_1$ and $\boldsymbol{x}^2_k \in \mathcal{S}_2$ as the (K)th neighbor in both sets, while $\boldsymbol{x}^1_j \notin \mathcal{S}_1$ and $\boldsymbol{x}^2_j \notin \mathcal{S}_2$ is the $(K+1)$th neighbor in both sets:

$$\|\boldsymbol{x}^1_k - \boldsymbol{x}^1_i\|^2 < \|\boldsymbol{x}^1_j - \boldsymbol{x}^1_i\|^2$$

$$\|\boldsymbol{x}^2_k - \boldsymbol{x}^2_i\|^2 < \|\boldsymbol{x}^2_j - \boldsymbol{x}^2_i\|^2$$

In the aggregate:

$$\|\boldsymbol{x}^1_k - \boldsymbol{x}^1_i\|^2 + \|\boldsymbol{x}^2_k - \boldsymbol{x}^2_i\|^2 < \|\boldsymbol{x}^1_j - \boldsymbol{x}^1_i\|^2 + \|\boldsymbol{x}^2_j - \boldsymbol{x}^2_i\|^2$$

Let $\|\boldsymbol{x}^1_k - \boldsymbol{x}^1_i\|^2 = a$, $\|\boldsymbol{x}^1_j - \boldsymbol{x}^1_i\|^2 = a + \gamma$, $\|\boldsymbol{x}^2_k - \boldsymbol{x}^2_i\|^2 = b$ and $\|\boldsymbol{x}^2_j - \boldsymbol{x}^2_i\|^2 = b + \epsilon$,

$$a + b < a + \gamma + b + \epsilon$$

$$\gamma + \epsilon > 0$$

$$\text{where } \gamma, \epsilon > 0$$

□

This result leads to the following corollary.

Corollary 3.1.1.1. *If the number of neighbors K is the same for both subvectors and for the aggregate and if $\mathcal{S}_1 = \mathcal{S}_2$, then $\mathcal{S}_1 = \mathcal{S}_2 = \mathcal{S}_A$.*

Lemma 3.1.2. *If $j \in \mathcal{S}_1 \cap \mathcal{S}_2$, $k \notin \mathcal{S}_1$ and $k \in \mathcal{S}_2$, it is possible that $j \notin \mathcal{S}_A$ while $k \in \mathcal{S}_A$.*

Proof. Let i be the query, we know:

$$\|\boldsymbol{x}_j^1 - \boldsymbol{x}_i^1\|^2 < \|\boldsymbol{x}_k^1 - \boldsymbol{x}_i^1\|^2$$

Consider the edge case selecting the last neighbor (Kth neighbor) in $\mathcal{S}_A$. If j is selected, $k \notin \mathcal{S}_A$, and vice versa.

Since both $\boldsymbol{x}_j^2$ and $\boldsymbol{x}_k^2$ are in $\mathcal{S}_2$, there are two possibilities: if $\|\boldsymbol{x}_j^2 - \boldsymbol{x}_i^2\|^2 < \|\boldsymbol{x}_k^2 - \boldsymbol{x}_i^2\|^2$, $\boldsymbol{x}_j$ will be selected as a neighbor in the aggregate, $j \in \mathcal{S}_A$. But if $\|\boldsymbol{x}_j^2 - \boldsymbol{x}_i^2\|^2 > \|\boldsymbol{x}_k^2 - \boldsymbol{x}_i^2\|^2$, $\boldsymbol{x}_j$ will be selected as a neighbor only if $\|\boldsymbol{x}_j^1 - \boldsymbol{x}_i^1\|^2 + \|\boldsymbol{x}_j^2 - \boldsymbol{x}_i^2\|^2 < \|\boldsymbol{x}_k^1 - \boldsymbol{x}_i^1\|^2 + \|\boldsymbol{x}_k^2 - \boldsymbol{x}_i^2\|^2$. □

Corollary 3.1.2.1. *Every point $j \in \mathcal{S}_1 \cap \mathcal{S}_2$ will be selected in the aggregate if $\|\boldsymbol{x}_j^1 - \boldsymbol{x}_i^1\|^2 + \|\boldsymbol{x}_j^2 - \boldsymbol{x}_i^2\|^2 < \|\boldsymbol{x}_k^1 - \boldsymbol{x}_i^1\|^2 + \|\boldsymbol{x}_k^2 - \boldsymbol{x}_i^2\|^2 \;\; \forall k \in \mathcal{S}_1 \triangle \mathcal{S}_2$, where $\mathcal{S}_1 \triangle \mathcal{S}_2 = (\mathcal{S}_1 \cup \mathcal{S}_2) \setminus (\mathcal{S}_1 \cap \mathcal{S}_2)$.*

In words, a point $j \in \mathcal{S}_1 \cap \mathcal{S}_2$ can be eliminated of $\mathcal{S}_A$ by other points found in $\mathcal{S}_1 \triangle \mathcal{S}_2$. If the above condition is satisfied for all points $k \in \mathcal{S}_1 \triangle \mathcal{S}_2$, we can be sure that j will be selected in the aggregate. The main relevance of this result is that the complexity of the K-NN search in the aggregate could be reduced, since those points guaranteed to be selected can be added directly to the aggregate neighborhood.

3.2 NNK analysis

Proposition 3.2.1. *Gaussian kernel in the aggregate space is the product of Gaussian kernels in the subvectors.*

Proof. In the aggregate space, $\boldsymbol{K}_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2}\right)$. Where

$$\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 = \sum_{s=1}^{S} \|\boldsymbol{x}_i^s - \boldsymbol{x}_j^s\|^2 \tag{3.1}$$

and S is the number of subvectors. Then,

$$\boldsymbol{K}_{i,j} = \prod_{s=1}^{S} \boldsymbol{K}_{i_s,j_s} \tag{3.2}$$

□

In a two-subvector scenario:

$$\begin{gathered} \boldsymbol{K}_{i,j} = \exp\left(-\frac{\|\boldsymbol{x}_i^1 - \boldsymbol{x}_j^1\|^2 + \|\boldsymbol{x}_i^2 - \boldsymbol{x}_j^2\|^2}{2\sigma^2}\right) \\ \boldsymbol{K}_{i,j} = \boldsymbol{K}_{i_1,j_1} \boldsymbol{K}_{i_2,j_2} \end{gathered} \tag{3.3}$$

Theorem 3.2.2. *If $j \in \mathcal{N}_1$ and $j \in \mathcal{N}_2$ then $j \in \mathcal{N}_A$.*

Proof. Consider a three-node scenario with j, k, and query i. We know that $\theta_{i_1,j_1} > 0$ and $\theta_{i_2,j_2} > 0$. Based on KRI theorem (2.7):

$$\theta_{i_1,j_1} > 0 \iff \boldsymbol{K}_{j_1,k_1} < \frac{\boldsymbol{K}_{i_1,j_1}}{\boldsymbol{K}_{i_1,k_1}} \tag{3.4}$$

$$\theta_{i_2,j_2} > 0 \iff \boldsymbol{K}_{j_2,k_2} < \frac{\boldsymbol{K}_{i_2,j_2}}{\boldsymbol{K}_{i_2,k_2}} \tag{3.5}$$

Then, in the aggregate:

$$\theta_{i,j} > 0 \iff \boldsymbol{K}_{j,k} < \frac{\boldsymbol{K}_{i,j}}{\boldsymbol{K}_{i,k}} \quad \forall k \in \mathcal{S}_A \tag{3.6}$$

Using Proposition 3.2.1:

$$\boldsymbol{K}_{j,k} < \frac{\boldsymbol{K}_{i,j}}{\boldsymbol{K}_{i,k}}$$

can be expressed as

$$\boldsymbol{K}_{j_1,k_1}\boldsymbol{K}_{j_2,k_2} < \frac{\boldsymbol{K}_{i_1,j_1}\boldsymbol{K}_{i_2,j_2}}{\boldsymbol{K}_{i_1,k_1}\boldsymbol{K}_{i_2,k_2}}. \tag{3.7}$$

Let $\boldsymbol{K}_{j_1,k_1} = a$, $\boldsymbol{K}_{j_2,k_2} = b$, $\frac{\boldsymbol{K}_{i_1,j_1}}{\boldsymbol{K}_{i_1,k_1}} = a + \gamma$ and $\frac{\boldsymbol{K}_{i_2,j_2}}{\boldsymbol{K}_{i_2,k_2}} = b + \epsilon$. Then, we can substitute terms in (3.7):

$$ab < (a + \gamma)(b + \epsilon)$$

$$a\epsilon + b\gamma + \gamma\epsilon > 0 \iff \theta_{i,j} > 0$$

where $0 \leq a, b \leq 1$ and $\gamma, \epsilon > 0$ considering (3.4) and (3.5).

$$\boxed{j \in \mathcal{N}_A \iff j \in \mathcal{N}_1, j \in \mathcal{N}_2, j \in \mathcal{S}_A} \tag{3.8}$$

In words, if j is not eliminated by any other hyperplane created by a third point k in the subvectors (is an NNK neighbor in all subvectors), then $j \in \mathcal{N}_A$ ($\theta_{i,j} > 0$). The only condition in the aggregate is that j has to be selected in the initial set of neighbors $\mathcal{S}_A$.

There are no extra conditions for the initial sets of neighbors $\mathcal{S}_1$ and $\mathcal{S}_2$. We know that $j \in \mathcal{S}_1 \cap \mathcal{S}_2$. If a third point $k \in \mathcal{S}_1$ and $k \notin \mathcal{S}_2$, but $k \in \mathcal{S}_A$, we want to verify (3.6) for this third point but $\boldsymbol{K}_{j_2,k_2}$ is not known.

But we know that $0 \leq \boldsymbol{K}_{i,j} \leq 1$, and $\boldsymbol{K}_{i_2,k_2} < \boldsymbol{K}_{i_2,j_2}$ because $j \in \mathcal{S}_2$ and $k \notin S_2$. Then,

$$\boldsymbol{K}_{j_2,k_2} < \frac{\boldsymbol{K}_{i_2,j_2}}{\boldsymbol{K}_{i_2,k_2}}$$

so (3.5) is fulfilled and (3.4) is also fulfilled since $k \in \mathcal{S}_1$, therefore (3.6) is also fulfilled. Also, condition $j \in \mathcal{S}_A$ can be easily met by selecting $\mathcal{S}_A = \mathcal{S}_1 \cup \mathcal{S}_2$ □

Corollary 3.2.2.1. *$\mathcal{N}_1 \cap \mathcal{N}_2 \subseteq \mathcal{N}_A$ if $\mathcal{N}_1 \cap \mathcal{N}_2 \subseteq \mathcal{S}_A$.*

Any point $j \in \mathcal{N}_1$ verifies (3.4) and any point $j \in \mathcal{N}_2$ verifies (3.5). Thus, (3.6) is verified for j.

If $|\mathcal{N}_1 \cap \mathcal{N}_2|$ is large relative to $|\mathcal{N}_1|$ and $|\mathcal{N}_2|$, then $\mathcal{N}_1 \cap \mathcal{N}_2$ can be used as a good dictionary to approximate $\mathcal{N}_A$. Then, from the NNK solutions in the subvectors we could construct the NNK graph on the aggregate without having to solve the NNK regression in the aggregate, which could involve more complexity. We will discuss complexity in Section 3.3.5.

Theorem 3.2.3. *In a three-node scenario, if $j \in \mathcal{N}_1, j \in \mathcal{N}_2$ and $k \notin \mathcal{N}_1, k \notin \mathcal{N}_2$, then $k \notin \mathcal{N}_A$.*

Proof. If $\theta_{i_1,k_1} = 0$ because of the hyperplane created by $\theta_{i_1,j_1} > 0$, and $\theta_{i_2,k_2} = 0$ because of the hyperplane created by $\theta_{i_2,j_2} > 0$, from KRI theorem (2.7):

$$\theta_{j_1,k_1} = 0 \iff \frac{1}{\boldsymbol{K}_{j_1,k_1}} < \frac{\boldsymbol{K}_{i_1,j_1}}{\boldsymbol{K}_{i_1,k_1}} \tag{3.9}$$

$$\theta_{i_2,k_2} = 0 \iff \frac{1}{\boldsymbol{K}_{j_2,k_2}} < \frac{\boldsymbol{K}_{i_2,j_2}}{\boldsymbol{K}_{i_2,k_2}} \tag{3.10}$$

Then, in the aggregate:

$$\theta_{i,k} > 0 \iff \frac{1}{\boldsymbol{K}_{j,k}} > \frac{\boldsymbol{K}_{i,j}}{\boldsymbol{K}_{i,k}} \tag{3.11}$$

$$\frac{1}{\boldsymbol{K}_{j,k}} \overset{?}{>} \frac{\boldsymbol{K}_{i,j}}{\boldsymbol{K}_{i,k}}$$

$$\frac{1}{\boldsymbol{K}_{j_1,k_1}\boldsymbol{K}_{j_2,k_2}} \overset{?}{>} \frac{\boldsymbol{K}_{i_1,j_1}\boldsymbol{K}_{i_2,j_2}}{\boldsymbol{K}_{i_1,k_1}\boldsymbol{K}_{i_2,k_2}}$$

Let $\frac{1}{\boldsymbol{K}_{j_1,k_1}} = a$, $\frac{1}{\boldsymbol{K}_{j_2,k_2}} = b$, $\frac{\boldsymbol{K}_{i_1,j_1}}{\boldsymbol{K}_{i_1,k_1}} = a + \gamma$ and $\frac{\boldsymbol{K}_{i_2,j_2}}{\boldsymbol{K}_{i_2,k_2}} = b + \epsilon$. Then,

$$ab \overset{?}{>} (a+\gamma)(b+\epsilon)$$

$$a\epsilon + b\gamma + \gamma\epsilon \not< 0 \iff \theta_{i,k} = 0$$

where $0 \leq a, b \leq 1$ and $\gamma, \epsilon > 0$ considering (3.9) and (3.10).

In words, if a point k is eliminated by a hyperplane created by the same point j in all subvectors, k will not be connected as an NNK neighbor in the aggregate. □

Lemma 3.2.4. *If $k \notin \mathcal{N}_1$ and $k \notin \mathcal{N}_2$, but in a three-node scenario with i and j, $\theta_{i_1,k_1} = 0$ and $\theta_{i_2,k_2} > 0$, then it is possible that $k \in \mathcal{N}_A$.*

Proof. Consider node k which is not selected as an NNK neighbor in any of the two subvectors. In a three-node scenario with points i and j in subspace 1, $\theta_{i_1,k_1} = 0$ because of the hyperplane created by θ_{i_1,j_1}. But in a three-node scenario with the same points in subspace 2, $\theta_{i_2,k_2} > 0$, so k is eliminated in subspace 2, not because of the hyperplane created by θ_{i_2,j_2}, but by the hyperplane created by a fourth point q, denoted by its normal $\theta_{i_2,q_2} > 0$. Then, in the aggregate:

$$\theta_{i,k} = 0 \iff \frac{1}{\boldsymbol{K}_{j,k}} < \frac{\boldsymbol{K}_{i,j}}{\boldsymbol{K}_{i,k}} \tag{3.12}$$

$$\frac{1}{\boldsymbol{K}_{j,k}} \overset{?}{<} \frac{\boldsymbol{K}_{i,j}}{\boldsymbol{K}_{i,k}}$$

$$\frac{1}{\boldsymbol{K}_{j_1,k_1}\boldsymbol{K}_{j_2,k_2}} \overset{?}{<} \frac{\boldsymbol{K}_{i_1,j_1}\boldsymbol{K}_{i_2,j_2}}{\boldsymbol{K}_{i_1,k_1}\boldsymbol{K}_{i_2,k_2}}$$

Let $\frac{1}{\boldsymbol{K}_{j_1,k_1}} = a$, $\frac{1}{\boldsymbol{K}_{j_2,k_2}} = b + \epsilon$, $\frac{\boldsymbol{K}_{i_1,j_1}}{\boldsymbol{K}_{i_1,k_1}} = a + \gamma$ and $\frac{\boldsymbol{K}_{i_2,j_2}}{\boldsymbol{K}_{i_2,k_2}} = b$. Then,

$$a(b+\epsilon) \overset{?}{<} (a+\gamma)b$$

$$a\epsilon \overset{?}{<} b\gamma$$

where $0 \leq a, b \leq 1$ and $\gamma, \epsilon > 0$ considering (3.9) and (3.10).

Therefore, we cannot ensure that k is disconnected, and can be selected as an NNK neighbor in the aggregate. □

3.3 Experiments

In this section, we start by demonstrating empirically the effects of the curse of dimensionality when dealing with high dimensional data. We also show a better interpretation of why the ID of the data can be much lower than the nominal dimension of the feature vectors when data can be seen as a collection of subvectors that are related to each other, and how it is connected to several properties of the NNK graphs. Finally, we study the ID of the subvectors and their aggregation in the specific case of CNN data representations using NNK graphs. In our experiments we consider a 7 layer CNN model composed of 6 convolutional layers of 16 depth channels with ReLU activations, max-pooling and 1 fully connected softmax layer. A more detailed description of the architecture and the implementation can be found in Appendix A.

3.3.1 Curse of dimensionality illustration with Distance Ratio

In order to have a clearer and more visual understanding of the *curse of dimensionality*, we use the Distance Ratio (DR) [41], a dimensionality metric described in Algorithm 1. For each point i we find the distance to the farthest and closest points in the dataset and

DR is the average of the ratio of these two distances. Thus DR is small when all pairwise distances in the dataset are similar, which could be the case in high-dimensional scenarios where distances are not informative. We show empirical results for both real and synthetic datasets.

Algorithm 1 Distance Ratio metric

Input: Data points $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\} \in \mathbb{R}^D$

$\boldsymbol{D} \leftarrow pairwiseDistances(\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\})$ ▷ pairs i, j with $i \neq j$, $\boldsymbol{D} \in \mathbb{R}^{N \times (N-1)}$

for $i = 1 : N$ **do**

  $\boldsymbol{m}_i \leftarrow \max_{1 \leq j \leq N-1} \boldsymbol{D}_{ij}$

  $\boldsymbol{n}_i \leftarrow \min_{1 \leq j \leq N-1} \boldsymbol{D}_{ij}$

  $\boldsymbol{r}_i \leftarrow \boldsymbol{m}_i / \boldsymbol{n}_i$

end for

$DR \leftarrow mean(\boldsymbol{r})$

Output: Distance Ratio DR

Generating 1000 10-dimensional random data points uniformly sampled from $\mathcal{N}(\mathbf{0}, \mathbf{I})$, we get $DR = 4.1$, where ID ≈ 10. In contrast, if we generate 1000 100-dimensional data points sampled from the same distribution, $DR = 1.5$, where ID ≈ 100. This means that on average, the farthest data point from each other point is only $\sim 50\%$ farther than the closest data point. The DR converges to 1 as the ID of the data increases. Also, it depends on the sample size, a larger dataset is prone to higher DR.

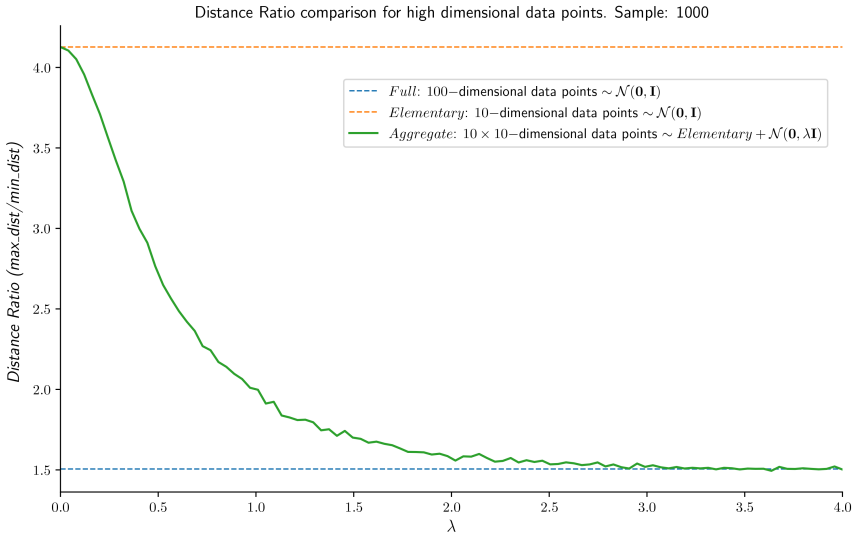


Figure 3.1: Distance Ratio as a function of relationship between subvectors controlled by λ, bounded by edge cases. Elementary: all subvectors are identical (min ID). Full: no relationship between subvectors (max ID).

Moving to a scenario with an aggregation of subvectors, we generate 10 10-dimensional subvectors to create a 100-dimensional aggregate. Again we use a sample set with 1000 data points and, for each data point, we generate random subvectors by first finding a subvector $\boldsymbol{x}_i^{\text{init}} \in \mathbb{R}^{10}$, sampled from $\mathcal{N}(\mathbf{0}, \mathbf{I})$, and then assigning to each subvector

$$\boldsymbol{x}_i^s = \boldsymbol{x}_i^{\text{init}} + \boldsymbol{w}_i^s \tag{3.13}$$

where $\boldsymbol{w}_i^s \sim \mathcal{N}(0, \lambda \mathbf{I})$ and λ is a parameter that controls the variance of the gaussian noise added to each subvector independently. Thus, all subvectors are related and they are closer to each other for smaller values of λ. If $\lambda = 0$ the subvectors are identical, while a large λ leads the subvectors to be virtually independent. In Figure 3.1 we can see how the DR of an aggregate of subvectors depends on the relationship between subvectors, which is bounded by the DRs mentioned above. If all the subvectors are initialized with the same 10-dimensional primary random vector ($\lambda = 0$), ID $\approx$ 10. As more noise is added to each subvector independently, DR decreases, reaching no relationship between subvectors, where ID $\approx$ 100.

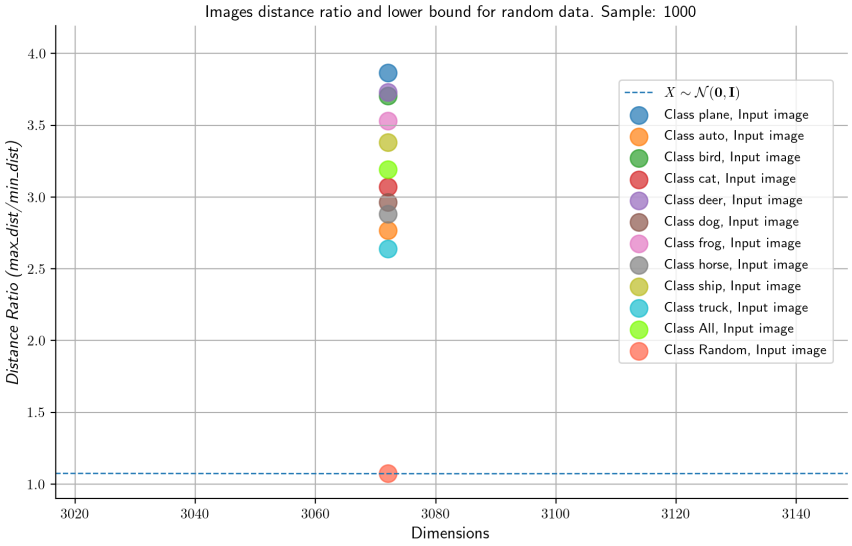


Figure 3.2: Distance Ratio for CIFAR-10 classes, random generated class and lower bound.

We can also compute DR for real datasets and random examples. In Figure 3.2 we show the DR obtained for each of the image classes in the CIFAR-10 dataset ($D = 32 \times 32 \times 3$), together with the lower bound of completely random data (ID $\approx D$). We also add an additional class ("Random") of generated random examples that have the CIFAR-10 images size and where each pixel value is sampled from $\mathcal{U}(0, 1)$. We see how the DR of the CIFAR-10 classes is much higher than that of completely unstructured random noise, and it is also different between classes.

Going one step further, we analyze the dimensionality in the intermediate representations of a CNN. Note that in Figure 3.2 original images of the classes "plane", "truck" and "random" produce very different DR, i.e., the ID is different between classes. In Figure 3.3, we show the DR obtained across the CNN layers, where the first intermediate representations are the ones with higher dimension. We can see how the ranking of ID between classes is preserved end-to-end: from input space (right) to last layer feature space (left), across all layers of the model. Also note that the obtained DR for the generated class of random examples is minimal for the original images, whereas the DR for their intermediate representation vectors in the model is higher than that obtained with completely random data (lower bound).

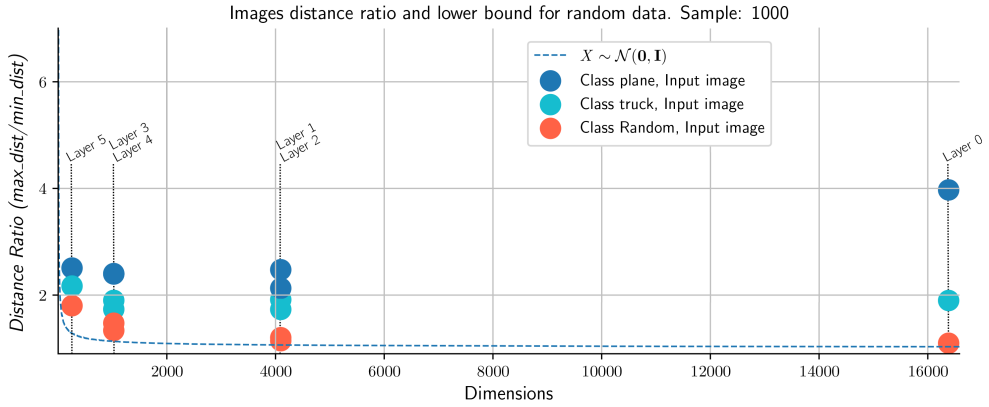


Figure 3.3: Distance Ratio of deep representations across CNN layers. Difference between classes is preserved end-to-end.

3.3.2 NNK neighbors overlap between channels

In Section 3.2 we have seen what information we can extract when different subvectors share the same NNK neighbors. However, we do not know how common this is and the effect it has on the dimensionality of the full space. To quantify the number of neighbors appearing in various subvectors, we compute the subvector neighbor **Overlap Score** (OS):

$$OS = \sum_{s=2}^{S} \frac{s}{S} \frac{P(s)}{T} \tag{3.14}$$

where S is the number of subvectors, $P(s)$ is the number of points overlapping in s subvectors, and T is the total union of neighbors in all subvectors. If the features are the same between subvectors, the neighbors are the same in all subspaces and $OS = 1$. If the features are completely independent between subspaces and there is no overlap of neighbors, $OS = 0$.

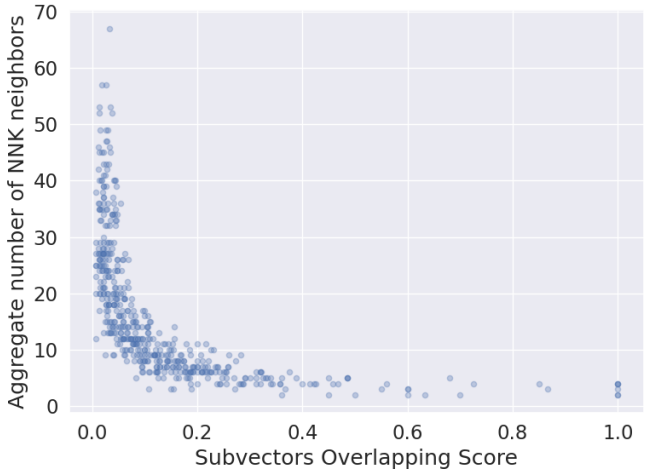


Figure 3.4: Number of NNK neighbors in aggregate space as a function of the subvector OS for synthetic data.

We perform an experiment with synthetic data to see how the OS is related with the number of NNK neighbors in the aggregate. We generate 150 data points, in 500 different realizations. Each data point is composed of 10 subvectors of dimension 2. Each subvector is constructed as in (3.13), with $\boldsymbol{x}_i^{\text{init}} \in \mathbb{R}^2 \sim \mathcal{U}(0,1)$ and $\boldsymbol{w}_i^s \sim \mathcal{N}(0, \sigma_{\text{realization}})$ where $\sigma_{\text{realization}} \sim \text{Exp}(\lambda = 5)$. In Figure 3.4 we can see how the NNK dimension in the aggregate feature space is a function of the relationship between subvectors. When the NNK neighbors are the same in all subvectors, the dimension of the full NNK graph is minimal. On the other hand, when the channels are more independent, OS is much lower, while the number of neighbors in the NNK graph and the ID of the aggregate space both increase.

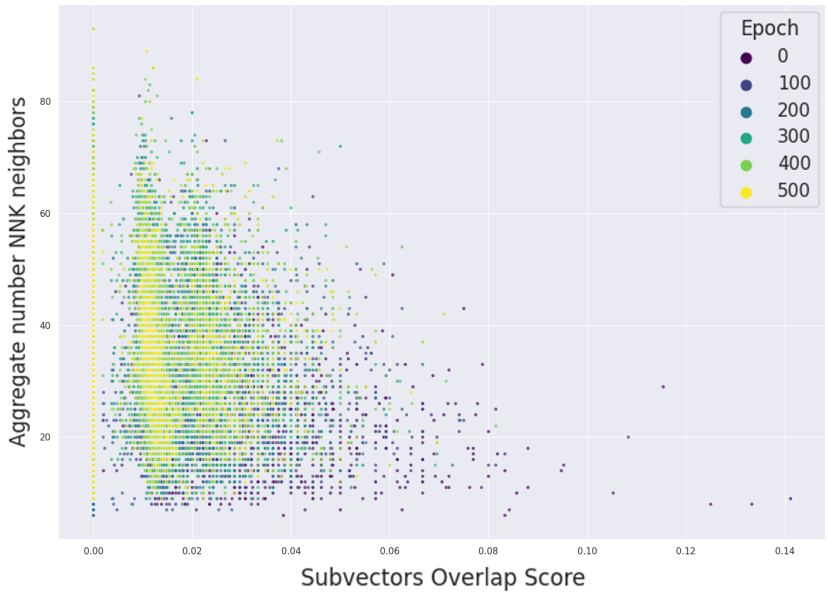


Figure 3.5: Number of NNK neighbors in the full last layer of a CNN as a function of the channel OS during a training of 500 epochs.

We also quantify the level of neighbor set overlap between subvectors in a practical setting for the CNN representations from the CIFAR-10 dataset. We focus on the last layer channels and their aggregation (full layer). While the data is different from the previous random examples (in particular, it is non-negative because of ReLU, and many dimensions are zero), we verify that the *maximum* number of NNK neighbors obtained in the aggregate is also directly related to the amount of neighbor set overlap in the channels, see Figure 3.5. This is an interesting scenario because even though every subspace is generated by different filtering, pooling and other nonlinear operations, there is overlap of neighbors. Also, we can observe how the maximum overlap cases appear in the first epochs of training. Then, as the training proceeds, each channel specializes to some specific features, resulting in a lower overlap and a higher maximum of NNK neighbors in the full layer.

3.3.3 Sufficient K to construct the NNK polytope

As described in Section 2.1.2, NNK graphs construct a convex polytope around each query point $\boldsymbol{x}_i$ using the Gaussian kernel (2.2), where the only point in the intersection of all R_{ij} regions is $\boldsymbol{x}_i$. When the decision boundary is *closed*, no additional points can be connected to $\boldsymbol{x}_i$ and we consider the graph construction procedure completed. But there is the possibility that the decision boundary may not be completely closed, due to the hyperplanes created by the edges θ_{ij} not being sufficient to enclose R_{ij}. In this scenario, the NNK graph could continue to grow if new points were added.

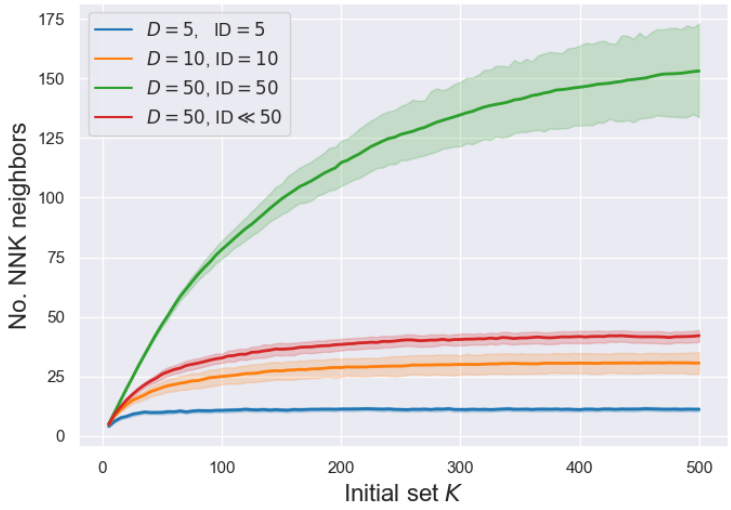


Figure 3.6: Number of NNK neighbors as a function of $\bar{K}_{\text{suf}}$. 20 realizations with 1000 train points per case and per dimension, where $\boldsymbol{x}_i \in \mathbb{R}^D \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, except for "$D = 50$, ID $\ll 50$" where there are 5 subvectors with $\boldsymbol{x}_i^{\text{init}} \in \mathbb{R}^{10} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and each subvector $\boldsymbol{x}_i^s = \boldsymbol{x}_i^{\text{init}} + \boldsymbol{w}, \boldsymbol{w} \sim \mathcal{N}(\mathbf{0}, 0.1\mathbf{I})$.

A *sufficient* K, which we call K_{suf}, is the number of initial neighbors with which the number of connections in the NNK graph saturates. That is, when we reach K_{suf}, the decision boundary is normally closed if there exist data points in all directions, and even if we increase the initial set of neighbors, the NNK graph will not grow any more. Since K_{suf} is specific for each data point, we use $\bar{K}_{\text{suf}}$ as the average K_{suf} for a particular set of data points. In Figure 3.6, we show how $\bar{K}_{\text{suf}}$ grows with the ID of the data manifold. Starting with a small K and increasing it, we reach a point where the number of NNK neighbors converges, reaching $\bar{K}_{\text{suf}}$. The number of NNK neighbors converges at different $\bar{K}_{\text{suf}}$ for ID ≈ 5 and ID ≈ 10. But for ID ≈ 50, $\bar{K}_{\text{suf}} \gg 500$, i.e., $\bar{K}_{\text{suf}}$ is related to ID and quantifying $\bar{K}_{\text{suf}}$ can help us estimating ID. In addition, experiments show that even in a high dimensional space, if ID $\ll D$, $\bar{K}_{\text{suf}}$ will be much lower, proportional to the ID.

In general, $\sum_{c=1}^{C} K_{\text{suf},c} < K_{\text{suf},A}$, i.e., we need more neighbors than the union of channel neighbors to build the NNK graph in the aggregate space. As an example, Figure 3.7 shows a specific case where $\mathcal{S}_A = \mathcal{S}_1 \cup \mathcal{S}_2$ is not sufficient to build the full NNK graph in the aggregate.

Generating synthetic random data where each data point $\boldsymbol{x}_i \in \mathbb{R}^D \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, ID $\approx D$. In Figure 3.8, we can see how the number of NNK neighbors also provides information

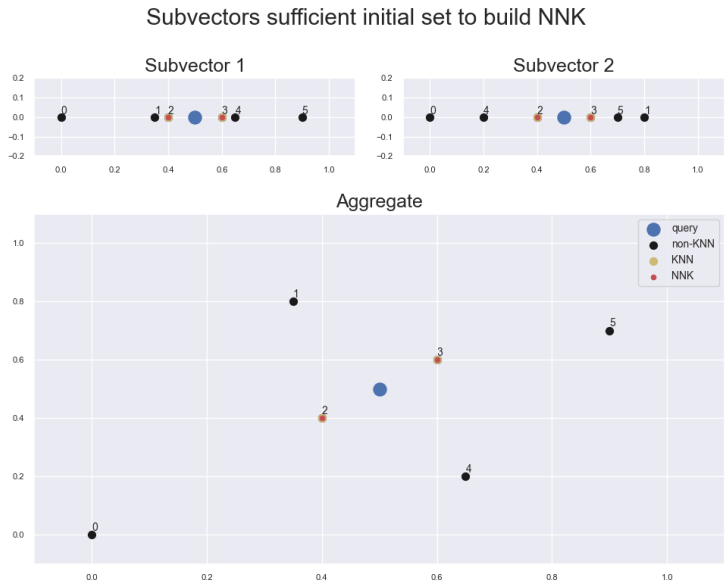


Figure 3.7: NNK graph on the aggregate using the union of sufficient sets $\mathcal{S}_A = \mathcal{S}_1 \cup \mathcal{S}_2$. Points 1 and 4 are not selected in the subvectors but should be selected in the aggregate to build the full NNK graph.

about the ID. To obtain the maximum number of NNK neighbors in each graph construction, we need a large enough K, which is related to the ID as previously described. Otherwise, we obtain a result similar to the cases of $K = 50$ or $K = 100$, where we see that the number of NNK neighbors starts to converge to K, since $K < \bar{K}_{\text{suf}}$ and we need more initial neighbors to build the NNK polytope. When K is large enough, the number of NNK neighbors increases proportionally with the ID.

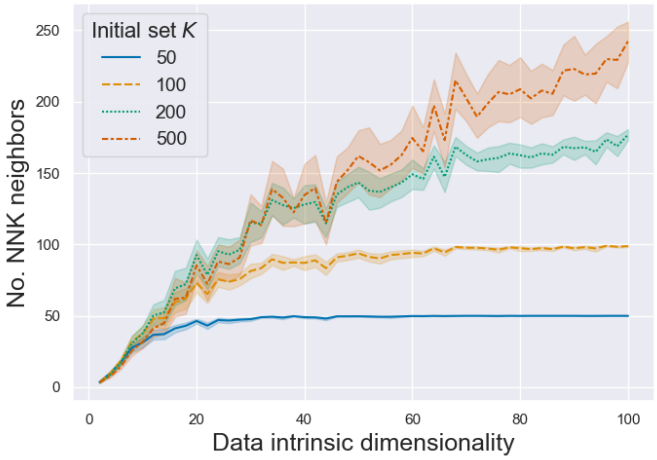


Figure 3.8: Number of NNK neighbors as a function of the ID of the data, for different K in the initial search of neighbors. 20 realizations per dimension and per K, with 1000 training points $\boldsymbol{x}_i \in \mathbb{R}^D \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, where ID $\approx D$.

3.3.4 Dimension significance and overall dimensionality reduction effect

The activations of CNN layers are usually high dimensional due to the large number of channels in each layer. Each feature map represents certain features extracted with filtering from the layer input, after going through different nonlinear operations. However, not all dimensions contribute equally to the final output [42, 43]. Only a portion of the activations are constantly contributing to the classification, while the rest of the dimensions are noisy, or are activated in very isolated cases.

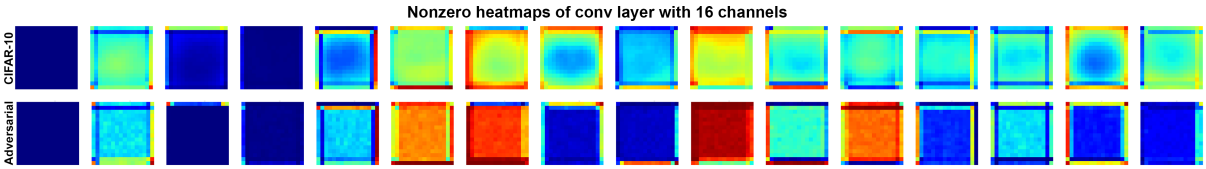


Figure 3.9: Nonzero heatmaps of the feature maps in the third layer of the CNN. In the first row, all CIFAR-10 test set is used as input, while in the second row we use random examples $\sim \mathcal{U}(0,1)$.

Especially in the case of CNNs where the ReLU is commonly used [17], we may find many dimensions that are completely zero, and in certain cases, full zero channels independently of the input, see Figure 3.9. This leads to a direct reduction of the intrinsic dimensionality of the data representations and can be found in a wide variety of models, particularly in overparametrized or non-regularized models. Moreover, this is a special case to take into account when constructing the NNK graph: we find multiple data points that have the same feature vector (full zero) in some channels. In this case, we keep only one data point of these characteristics (it can also be the query), since the rest of the points would not provide information in new orthogonal directions.

It has been demonstrated that adversarial examples have activations that are distributed more uniformly among channels, and have higher magnitude [44]. In Figure 3.9 we show that this is true, but only in the significant channels. Note that the artifacts generated on the edges are caused by zero padding. Conversely, in the rest of the dimensions, we see that the results are very similar to those obtained with normal examples: the dimensions fail to activate and undergo very little variation. Nonzero heatmaps for the different CIFAR-10 classes for both regularized and non-regularized models are provided in Appendix B.

In Figure 3.10 we show how the NNK dimension (number of connected neighbors) depends on the number of zeros in the activations. One of the main reasons for having fewer NNK neighbors in some cases is that the number of common zeros reduces the dimension of the space. Also, the number of neighbors in the aggregate of channels is very close to the channel-wise case, although the crude dimension of the activation is much larger. This demonstrates that channels are highly dependent and in the overall feature space the ID is very low.

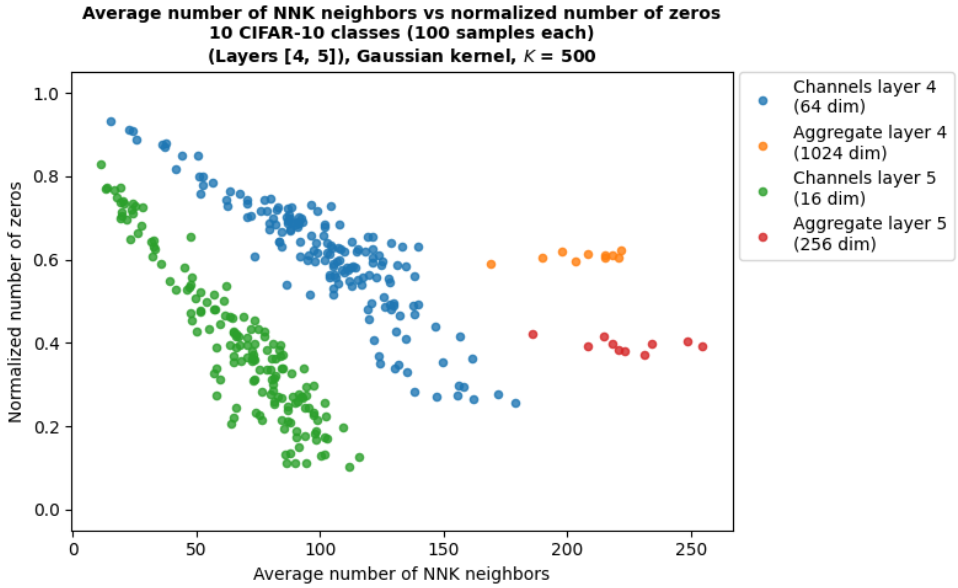


Figure 3.10: Average number of NNK neighbors and normalized number of zeros per CIFAR-10 class and per activation in layers 4 and 5 of the model.

.

3.3.5 Complexity

The NNK graph construction consists of two steps. First, finding the set $\mathcal{S}$ of K nearest neighbors out of N training data points. Although there exist some efficient algorithms that find an approximate solution in $O(N^{1.14})$ [21], the exact solution can always be found by brute force and it requires $O(N^2KD)$ for each query. Second, solving a non-negative kernel regression (2.6) that runs in $O(K^3)$.

With our proposed CW-NNK graphs method, we could alleviate complexity when constructing the graph in the full space by combining the channel results. For example, we could skip the first step in the aggregate, and use the union of the obtained K-NN in the channels as the aggregate neighbor initial set $\mathcal{S}_A$. In the full space, the first step is $O(N^2K_AD)$, and in the channel-wise case $O(N^2K_CD_CC)$. Since $D_CC = D$, the complexity is lower in the channel-wise approach only if $K_C < K_A$, i.e. we use a lower K in the individual channels than in the aggregate. As described in Section 3.3.3, we should always select $K_C \ll K_A$, since $\bar{K}_{\text{suf}}$ to build the NNK graph is proportional to the ID of the data, which will be higher in the full feature space than in the lower dimensional channels. Therefore, by selecting a suitable K, we can reduce the complexity of NNK graph construction with the channel-wise approach. We also have to ensure that when selecting $\mathcal{S}_A$ as the union of channel neighbors, $|\mathcal{S}_A|$ is of the same order of magnitude as $\bar{K}_{\text{suf}}$ for the full space or lower, thus maintaining or reducing the complexity in the second step of NNK (2.6). Also, in a practical application, (e.g., label interpolation from the neighboring nodes, which we will address in the next Chapter) using NNK graphs or related algorithms, we can still get many of the benefits of these methods without having to compute the exact solution, so that much faster approximate approaches can still be very useful.

3.4 Discussion

We presented a channel-wise graph construction approach as an extension to NNK graphs. To the best of our knowledge, no prior work had formally studied this setting. Although encountering very high dimensional data, we can often divide the feature vectors into subvectors in a consistent and interpretable way, as is the case for the channels in convolutional layers. We studied how properties of the NNK solution in the aggregate can be inferred from the solutions in each of the channels, which leads us to a better understanding of the information relationship between channels. We also analyzed the complexity of the proposed approach, and by doing so we were able to see how we could find an approximate solution in the aggregate from the channels with a lower computational cost.

We illustrated the effects of the curse of dimensionality and how it is significantly eased in real data scenarios. From the construction of NNK graphs, we showed that some polytope properties such as the required number of initial neighbors and the number of NNK neighbors are directly related to the intrinsic dimensionality of the data, which is usually much lower than the actual dimension of the vector in real data or deep representations in neural networks. Finally, with the channel-wise graph construction approach we obtain a better estimates of the ID through finding the number of neighbors.

Chapter 4

CW-DeepNNK generalization estimate without validation set

During the training of a neural network, the goal is to minimize a loss function given a finite training dataset. One of the main challenges in this optimization is for the model to be able to generalize and perform well on unseen data. The problem is that there will come a point in the training where the model may stop generalizing and will start to learn the statistical noise of the finite training dataset, which could lead to decreasing performance on new data, even though the loss function continues to be decreased by additional training [17]. But how can we detect when this happens in order to avoid it, and save the model that generalizes better? The most common approach is to hold out part of the training set (a validation set) with which we will not train, but on which we will evaluate the performance of the model so as to obtain a generalization estimate. Then, we can perform early stopping [18], which consists in stopping the training when we detect that the generalization on the validation set does not improve after some given epochs. Finally, we keep the copy of the model in the iteration where we obtained the best generalization performance.

Although these methods are very effective in practice and lead to state-of-the-art results, there are some drawbacks [19]. The choice of validation set size carries with it a trade-off: a small validation set has a large stochastic error and may introduce biases, which can result in a poor generalization estimate. On the other hand, a larger validation set yields a more robust generalization estimate but deprives the model of valuable information by reducing significantly the amount of training data. If there is scarcity of labeled data, the selection of the validation set is critical and its use would be more valuable if all the data could be used to train the model. In addition, while validation strategies can be used in practice, there are still difficulties in achieving a good understanding and interpretation of why large neural networks generalize well in practice [15].

Motivated by the previously proposed DeepNNK approach [1], we introduce a novel approach for channel-wise generalization estimation (CW-DeepNNK) that allows us to perform channel-wise early stopping effectively without the need for a validation set. This method is based on leave one out estimation using local polytope label interpolation. In addition, it can be easily integrated with an existing training setup, replacing the existing generalization estimate, e.g., validation accuracy.

We propose a generalization estimate that can be decomposed into multiple estimates, in this case by convolutional channels, rather than having a single metric as in most standard methods, such as performance on a validation set. We also show that the point at which additional training can worsen generalization occurs at different stages in different channels. Thus, when detecting that generalization performance decreases in some channels it is possible to stop training some channels while continuing to train the others.

A comparison with other state of the art methods is carried out, showing how this channel-wise monitoring can be equally or more effective in detecting generalization in some scenarios. Finally, we discuss different options for reducing the complexity of our algorithm while maintaining a good estimate of generalization.

4.1 Related work

Most successful deep learning models include some kind of regularization in their architectures to ensure a small generalization error [15]. Among them, we find data augmentation, weight decay [45], dropout [46] and batch normalization [47]. Regularization may also be implicit as in the case of early stopping.

Many early stopping criteria have been proposed [18, 48, 49, 19], most of which are based on the validation set performance, usually using the loss or accuracy curve. The most used criterion is to stop training the model when the validation performance has not improved over the best one recorded for a given number of epochs, usually called *patience*. Other stopping conditions focus on an absolute change in performance, an average change over a given number of epochs, or the worsening of performance in consecutive epochs. An alternative is to stop training when the validation performance is under the best one recorded by a given threshold, while the model error on the training set no longer improves much [18]. But again, these are all different stopping criteria that are generally constructed around the performance curve of an independent validation set, which is the state-of-the-art generalization estimator.

Other generalization estimates to perform early stopping have been proposed without the need for a validation set. Duvenaud et al. [49] proposed an interpretation of stochastic gradient descent in the variational inference framework. This motivated a generalization estimate that can be used to construct a stopping rule. It is based on estimating the marginal likelihood, by tracking the change in entropy of the posterior distribution of the network parameters at each optimization step. However, this method requires computing the Hessian diagonals, which may be impractical for large neural networks. Mahsereci et al. [19] proposed an estimate based on fast-to-compute local statistics of the computed gradient, aimed at detecting when it represents statistical noise of the finite training set, instead of an informative gradient direction. These two proposed methods for early stopping are based on gradient-related statistics, thus, they are sensitive to hyperparameter selection, e.g., learning rate, batch size or optimizer selection. In addition, they are also sensitive to neural network parameters, i.e., weights and biases, converging at very different speeds during optimization.

4.2 DeepNNK: generalization estimate using polytope interpolation

DeepNNK [1] is a non parametric interpolation framework based on local polytopes obtained using NNK graphs [11] on neural network data representations. Other methods

such as K-NN-based interpolation can be biased if the data density is different in different directions in space. In contrast, NNK selects only relevant points for interpolation, eliminating redundant points that do not provide new (orthogonal) information.

To integrate this interpolation framework with an existing neural network setup, we replace the last classification layer with the DeepNNK interpolator framework, using as input the features of the transformed space of the penultimate layer. We can continue using the loss obtained with the last layer to perform backpropagation, while the framework can be used to evaluate during training or testing. In this way, we perform label interpolation based on the relative positions of the training data in the output transformed space, instead of defining a classification boundary in the space. As previously mentioned, we do not have to set aside part of the training set to create a validation set to evaluate the model during training.

Now, the input data $\boldsymbol{x}_i$ is transformed by a non linear mapping $\boldsymbol{h}$ denoting the deep neural network. We can rewrite the Gaussian kernel (2.2) as

$$k_{\text{DNN}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\|\boldsymbol{h}(\boldsymbol{x}_i) - \boldsymbol{h}(\boldsymbol{x}_j)\|^2}{2\sigma^2}\right) \tag{4.1}$$

Given K nearest neighbors of a sample $\boldsymbol{x}$, $\mathcal{S} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2) \dots (\boldsymbol{x}_K, y_K)\}$ the unbiased NNK interpolation estimate is defined as

$$\hat{\eta}(\boldsymbol{x}) = \mathbb{E}(Y|X = \boldsymbol{x}) = \sum_{i=1}^{\hat{K}} \frac{\boldsymbol{\theta}_i}{\sum_{j=1}^{\hat{K}} \boldsymbol{\theta}_j} y_i \tag{4.2}$$

where $\boldsymbol{\theta}$ are the $\hat{K}$ non zero recomputed NNK weights obtained from the minimization of (2.6). Most of the initial K nearest neighbor weights are set to zero and we end up performing the label interpolation with the stable set of NNK neighbors.

In order to evaluate the performance of the estimator, we perform the leave one out (LOO) procedure, which is unbiased and widely used [50]. Given the training data $\mathcal{D}_{\text{train}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2) \dots (\boldsymbol{x}_N, y_N)\}$, the NNK interpolation estimator for $\boldsymbol{x}_i$ is based on the set containing all training points except $\boldsymbol{x}_i$, which we denote by $\mathcal{D}^i_{\text{train}}$. Formally,

$$\mathcal{R}_{\text{LOO}}(\hat{\eta}|\mathcal{D}_{\text{train}}) = \frac{1}{N} \sum_{i=1}^{N} l(\hat{\eta}(\boldsymbol{x}_i)|\mathcal{D}^i_{\text{train}}, y_i) \tag{4.3}$$

where $l(\hat{y}_i, y_i)$ is the error associated in regression or classification. Shekkizhar and Ortega [1] demonstrated that LOO performance can be a better indicator of generalization than the empirical model performance on training data.

4.2.1 CW-DeepNNK

The interpolation framework just described aims at estimating generalization error with the LOO procedure. In this work, continuing with the channel-wise spirit of Chapter 3, we formulate the local polytope label interpolation in individual channels (CW-DeepNNK).

Instead of using the transformed data representations of the full last convolutional layer $\boldsymbol{h}_{\text{last}}(\boldsymbol{x})$, which consists of the aggregation of outputs of C convolutional channels, we suggest dividing the feature space into channels:

$$\boldsymbol{h}_{\text{last}}(\boldsymbol{x}_i) = \begin{bmatrix} \boldsymbol{h}^1_{\text{last}}(\boldsymbol{x}_i) \\ \boldsymbol{h}^2_{\text{last}}(\boldsymbol{x}_i) \\ \vdots \\ \boldsymbol{h}^C_{\text{last}}(\boldsymbol{x}_i) \end{bmatrix} \in \mathbb{R}^{D_{\text{last}}} \tag{4.4}$$

which are well defined and have a better interpretability on their own. Then, the first step for the CW-DeepNNK LOO procedure is to perform the K-NN search in each channel, obtaining $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_C$ for each train data point $\boldsymbol{x}_i$. The second step is to use the subvectors $\boldsymbol{h}^c_{\text{last}}(\boldsymbol{x}_i)$ to construct the similarity matrix $\boldsymbol{K}_{\mathcal{S}_c}$ with the Gaussian kernel (4.1), and solve the NNK regression (2.6) obtaining $\boldsymbol{\theta}_{\mathcal{S}_c}$ in each channel c. Then, we perform the NNK interpolation (4.2) per instance and per channel. Finally, we compute the LOO estimation (4.3) per channel, obtaining the CW-DeepNNK label interpolation errors $\mathcal{R}^1_{\text{LOO}}, \mathcal{R}^2_{\text{LOO}}, \ldots, \mathcal{R}^C_{\text{LOO}}$.

By computing the CW-DeepNNK procedure at each training epoch we obtain a CW-DeepNNK label interpolation error curve for each channel. Using these curves to monitor the generalization of the model during training, we propose a novel channel-wise early stopping method, which does not require a validation set and the stopping is performed in stages. Starting from the standard *patience* criterion, we monitor the generalization performance in the last convolutional layer channels and we use a *patience* parameter in each channel. When a channel stops generalizing we freeze the model parameters of the channel and stop training it. The rest of the model continues learning until each of the channels stops generalizing, where we consider that we have reached the optimal point and the overall generalization of the model no longer improves. Finally, we save the model parameters where the last minimum generalization error is detected.

4.3 Experiments

In this section, we focus on binary classification using 2 classes of CIFAR-10: "plane" and "ship". We use a CNN architecture of 4 conv layers with 5 depth channels, ReLU and max-pooling and a last fully connected layer. Details of the model architecture and implementation are provided in Appendix A. We compare the model performance with the DeepNNK interpolation on train data, using the data representations from the full last convolutional layer or the individual channels. Some channels learn features more valuable than others for the classification task, and we study how this can be detected based on the NNK polytope local geometry and activation patterns. We also analyze the behaviour of the proposed generalization estimate when we do and do not apply explicit regularization to the model.

Additionally, we compare the NNK-based generalization estimates with the standard method [18] to perform early stopping and we discuss their complexity.

4.3.1 Interpretation of channel-wise generalization estimates

A convolutional layer is composed of various channels that are outputs of different filtering operations. Each channel defines a feature subspace where we should be able to quantify how useful the information from that channel is for the classification task, and have a better interpretation of the captured features in each channel. We can assume that each filter captures different features, although they are not completely independent between channels.

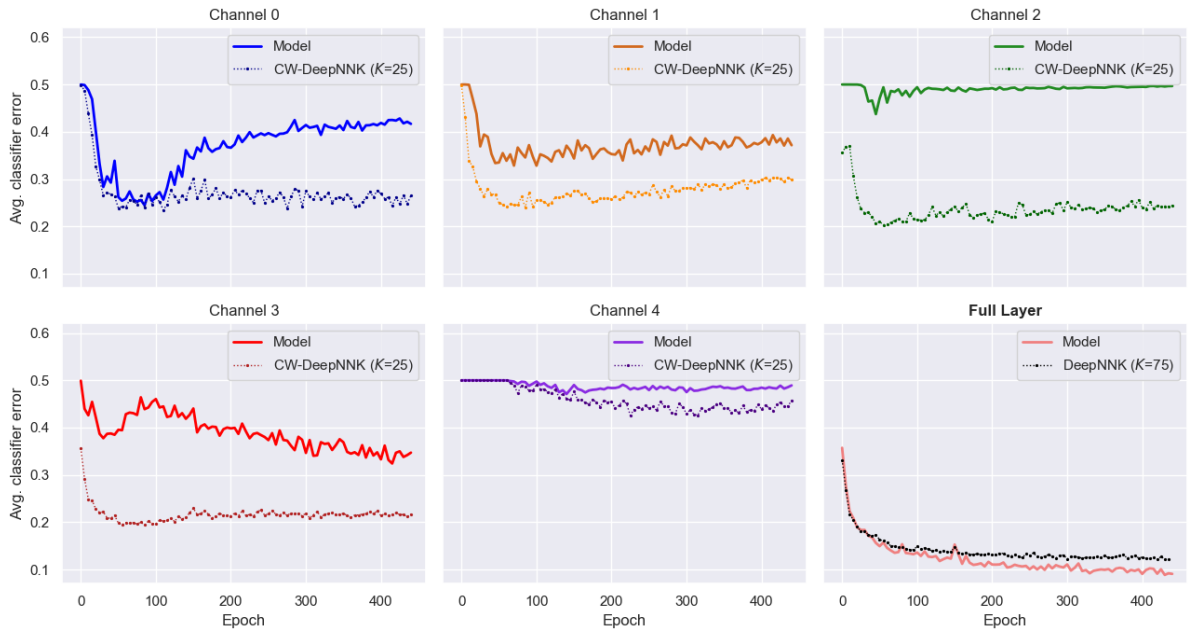


Figure 4.1: DeepNNK, CW-DeepNNK and model error on the train set during training with no regularization.

Figure 4.1 shows a comparison between model error on training data, DeepNNK and CW-DeepNNK label interpolation error with LOO estimation. In this case, the model is trained with no regularization. In the standard case using all channels (full layer), the error gap between the model on train data and LOO DeepNNK increases with the epochs, indicating that the generalization performance is worsening and the model starts to overfit to the train data. We can see how CW-DeepNNK has a much better performance than the model when only a single channel is activated. Note that an error of 0.5 in a binary classification is as bad as doing random classification. We also observe how the interpolation error in the channels soon reaches a minimum, and then the classification error increases again. This minimum may indicate the optimal point of generalization in each channel, from which the learned features begin to fit the training data noise.

Although the last fully connected layer of the model is trained to use the full combination of features from all channels, we wanted to see what happens if the model has to perform classification when relying only on partial information. We demonstrate how our method is able to perform much better in each independent channel subspace, and the model is not capable of performing at a decent level when some feature channels are

deactivated. Thus, we now have a new point of view, where we can estimate properly how useful each of the individual channels is for the task.

Generalization estimation in regularized vs. non-regularized models

With this new channel generalization estimate based on LOO label interpolation, we can analyze how it performs in different scenarios. With the same setup as the one used for experiments in Figure 4.1, in Figure 4.2 we compare the previous results obtained with no regularization with the results obtained adding a dropout layer after each convolutional block, with a dropout rate of 0.2.

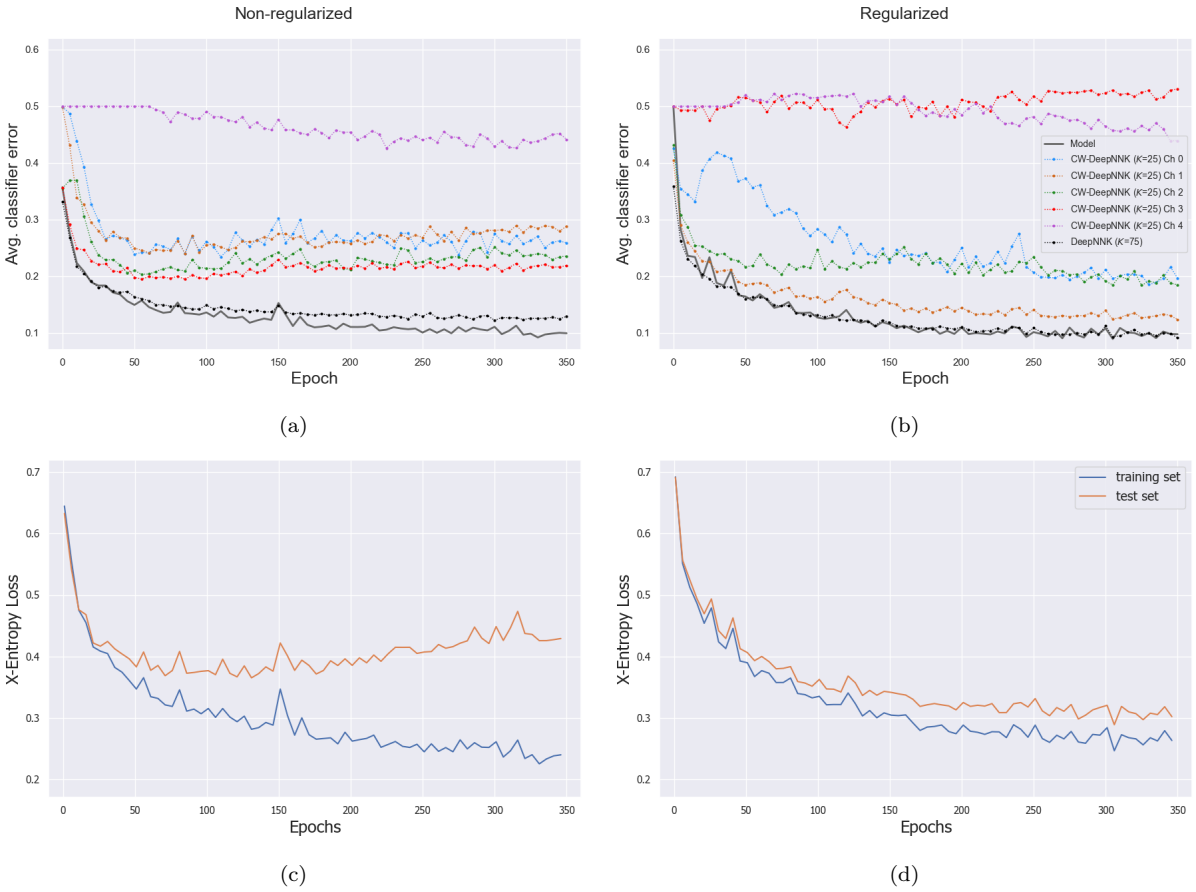


Figure 4.2: Metrics shown every 5 epochs, binary classification. (a) DeepNNK, CW-DeepNNK and model error on the train set with no regularization. (b) DeepNNK, CW-DeepNNK and model error on the train set using dropout. (c) Loss on train and test data with no regularization. (d) Loss on train and test data using dropout.

In both cases we see individual channels that fail to learn features of the data relevant to the task, obtaining almost maximum error in binary classification $\mathcal{R}_{\text{LOO}} \approx 0.5$. In the following study we focus on the channels with an error $\mathcal{R}_{\text{LOO}} < 0.4$.

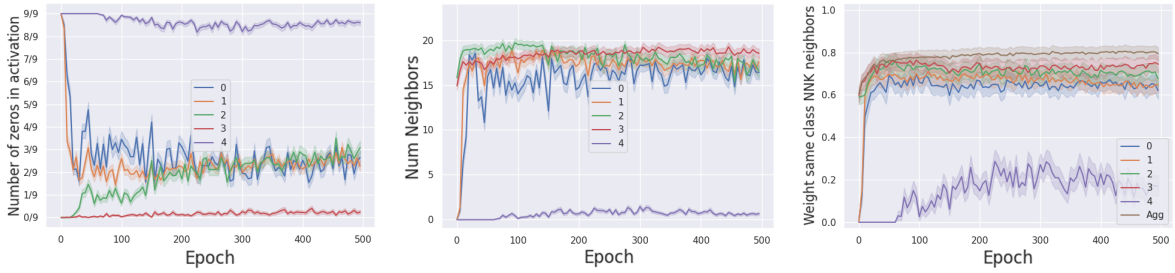
Figure 4.2a shows how in a non-regularized model, our CW-DeepNNK generalization error estimate finds a minimum and from then on it gets worse with more training iterations. If we compare it with the test performance in Figure 4.2c, we see that channel-wise LOO performances peak at a similar place to the peak of the test loss, where early stop-

ping would occur using the standard method based on validation set performance. On the contrary, by monitoring this generalization estimator in a well regularized network that does not overfit, our estimator keeps improving, as well as the test performance, as shown in Figures 4.2b and 4.2d. All this indicates that this metric is a good estimator of generalization and could replace other estimators such as validation accuracy for performing early stopping. In addition, channel minimum error detection in different points in time suggests that we could stop training each channel when it has reached its best generalization performance, and stop training all the network when we have reached best performance in all channels.

We can also detect that in the non-regularized case, the active channels have a very similar, but very poor, performance with an error between 0.2 and 0.3. Instead, in the regularized case we can see that in the main channels we reach an error below 0.2.

4.3.2 Relevant channel detection based on NNK polytope local geometry

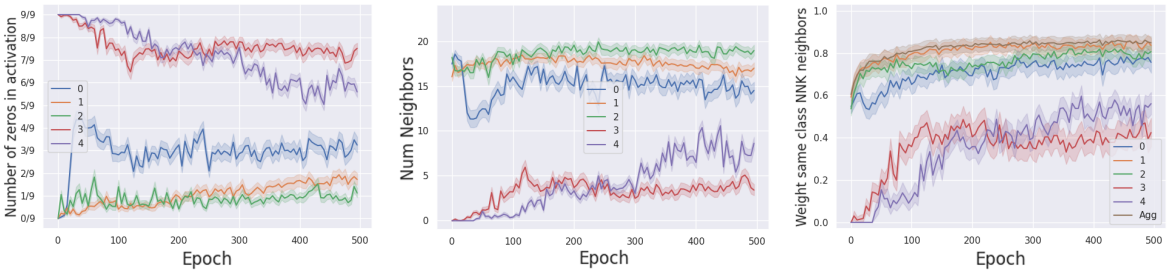
As seen in Section 4.3.1, only some channels of a convolutional layer concentrate the data features that are key for the task, and we can detect these channels before performing the LOO interpolation based on the NNK polytope local geometry and zero patterns in the activations.



(a) No. zeros per channel, non-regularized case.

(b) No. NNK neighbors per channel, non-regularized case.

(c) Weight same class NNK neighbors, non-regularized case.



(d) No. zeros per channel, regularized case.

(e) No. NNK neighbors per channel, regularized case.

(f) Weight same class NNK neighbors, regularized case.

Figure 4.3: Number of zeros in activation, number of NNK neighbors and weight of same class neighbors per channel in the last convolutional layer of a CNN, for both regularized and non-regularized scenarios.

In Figure 4.3 we can see how in the non-regularized case (first row), there is a very noticeable binary behavior. 4 out of 5 channels present very few zeros in their activations

due to ReLU, i.e. in most dimensions we will find positive values. In these same channels, we connect with a high number of NNK neighbors, obtaining a high weight of NNK neighbors of the same class to the instance. In the remaining channel, we observe that most dimensions are always zero so the ID is very low, we connect with very few neighbors and the weight of neighbors of the same class is very low, since the information in that channel is very poor. These properties will lead to a bad label interpolation. In the regularized case (second row) we observe an heterogeneous behavior: there are channels with less zero dimensions than others, in which we will connect with more NNK neighbors and we will obtain a higher weight of neighbors of the same class. What is relevant in this case is that, in the channels with higher NNK dimensionality we obtain a higher weight of neighbors of the same class, surpassing the weight obtained in the non-regularized case, thus leading to lower NNK label interpolation error, which is indicative of a better generalization in the regularized case.

In short, we can predict the channel LOO label interpolation performance of 4.2 by analyzing the dimensionality and NNK graph local geometry in each channel. In channels with fewer zero dimensions we will obtain a higher dimensional NNK graph with higher same-class weights, which will lead to better interpolation. Besides, the non-regularized networks have a more homogeneous behavior between channels but with worse performance in general, while for regularized models, the most relevant channels drive a better overall result.

4.3.3 Comparison of generalization estimates for early stopping

We compare our generalization estimate with the standard and state-of-the-art estimate, i.e., validation set performance, as well as with the DeepNNK estimate on the full layer [1], to perform early stopping. In this case, we use a *patience* parameter as stopping rule for the validation-based estimate and for the full layer LOO DeepNNK interpolation. For our estimate, we use a *patience* stopping rule in each channel, as described in Section 4.2.1. Note that other proposed stopping criterions [18, 48] could be applied to our generalization estimate as well, replacing the validation curve, but we leave these experiments for future work.

Figure 4.4 shows the results obtained using the different generalization estimates for early stopping, with a patience of 20 epochs. We can see how both NNK methods obtain test accuracies comparable to that of the standard validation method. Although the best models are obtained in the full layer case, it requires a lot of epochs to find the optimal stopping point whereas the other methods are able to detect overfitting in a much earlier stage. In the two proposed methods based on NNK interpolation, using a patience parameter that is too small can lead to not finding the global minimum of generalization error, leading to premature stopping and lower test accuracy, as in the case of the outliers. Therefore, choosing a higher patience can ensure a high test accuracy, but at the cost of a longer training, often unnecessary. The proposed channel-wise generalization estimate is the alternative that obtains the best trade-off between performance and training iterations. This method may be the most effective and the preferred one when dealing with small datasets or when test performance is a priority, since we can train the model

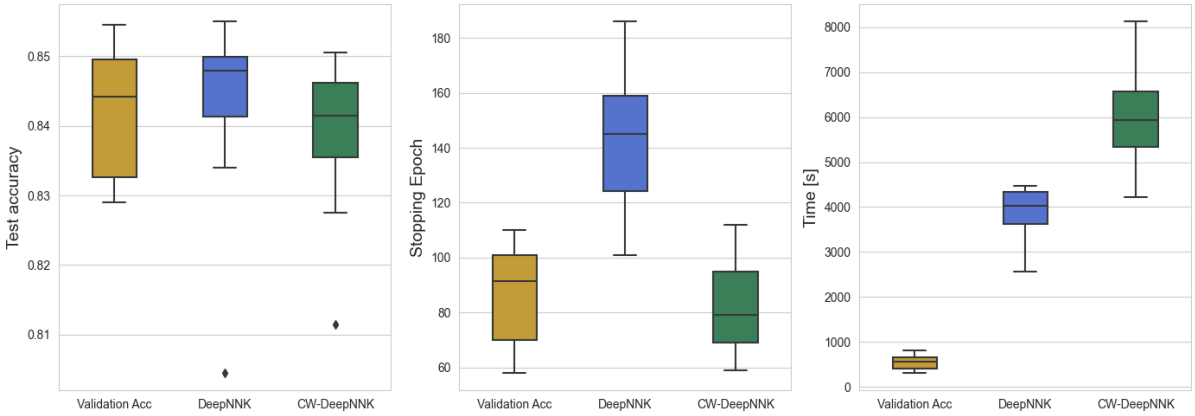


Figure 4.4: Test accuracy, stopping epoch and training elapsed time for different early stopping methods, using 20 epochs of patience and 10 different initializations. Note that test accuracy refers to the accuracy obtained in the test set with the best model according to each criterion, i.e., where we find the last minimum generalization error, not the last version where we stop training. Parameters: $K = 15$ for NNK-based methods, and validation set consisting of 20% of the original train set for the standard method.

with all labeled data available without the need of separating data for a validation set. However, the channel-wise approach *baseline* entails a higher complexity, which may not be desirable in certain scenarios where lightweight training is a priority. In the following section we discuss several different ways to speed up our proposed method and reduce complexity while maintaining good results.

Even if in the channel-wise method we stop training weights of last layer convolutional channels, the number of epochs is comparable to that of the other criterions, since the fraction of parameters that we stop training in intermediate stages is not significant. An extension of this method to the rest of the layers of the model could speed up the training iterations.

4.3.4 Complexity

The channel-wise leave one out label interpolation *baseline* is computationally expensive, since it requires constructing an NNK graph per training instance, per channel, and per epoch. Nonetheless, several optimizations can be applied to it to create a more efficient yet equally competitive early stopping method.

Performing NNK independently in every epoch is very costly, but features learned in the channels are similar between consecutive epochs. Thus, we could reduce the complexity by half by performing the leave one out estimation every other epoch. Another option is to monitor if the dictionary of neighbors is deviating from a good dictionary and only recompute the search when needed.

Other improvements in efficiency are in the direction of reducing the number of queries and channels with which the interpolation is performed. We could perform the interpolation only on the most relevant channels with the lowest error, described in Section 4.3.2. We could also perform random subsampling of training data points instead of doing

the full LOO procedure. As a result, computing time would be drastically reduced, but the generalization estimate would become more stochastic. This could be helped by augmenting the patience parameter, thus avoiding curve oscillations and finding the global generalization error minimum with higher probability.

Additionally, in the channel-wise method itself, we already find some complexity benefits over the whole-layer approach. $\bar{K}_{\text{suf}}$ is much larger in the aggregate space than in the subspaces of each channel $K_A \gg K_C$. This leads to a remarkably faster NNK graph construction starting point in the channel-wise case, since the NNK regression (2.6) runs in $O(K^3)$. Also, while selecting K for the experiments we observed that both NNK-based methods are very robust with respect to the selection of this single hyperparameter. Therefore, K could be chosen as the minimum K that yields stable results, significantly reducing the complexity. Exploring all these ideas for further efficiency improvement of the proposed method is left for future work.

4.4 Discussion

We introduced a new generalization estimate based on channel-wise local polytope interpolation that can be used to perform early stopping without the need for a validation set. We provided an interpretation for the channel-wise approach, that focuses on selecting relevant information on multiple well-defined feature subspaces. By doing so, we were able to find a new and more precise perspective on the generalization of neural networks.

From an in-depth analysis of the polytope local geometry and data representation patterns, we showed how to detect the channels that are the most relevant for the task and obtain best interpolation performance. Our experiments suggest that a lot of computation can be saved based on this observations. Key insights can be obtained from the overall channel interpolation error during training, detecting very different behaviors between regularized models that generalize well and models that overfit.

Our proposal may be the preferred for early stopping in situations where very little labeled data is available, since we could use it all for training without the need to save data for validation. Also, when using small size neural networks where test performance is key, e.g. embedded systems. However, our method may not be as useful for large scale problems, because if we already have a lot of labeled data, it is not a big impact on training to have to separate a validation set.

Chapter 5

Budget

For the student we assume a salary of 9 €/h, which is the standard undergraduate internship salary, and we estimate a salary of 30 €/h for the supervisors.

We estimated the computation cost based on Google Cloud rates[1] for the NVIDIA Tesla T4. Overall GPU usage has been approximately 60 hours per week, with a project duration of 25 weeks.

	Amount	Cost/hour	Time	Total
Junior engineer	1	9,00 €/h	625h	5.625,00 €
Senior engineer	3	30,00 €/h	25h	2.250,00 €
GPU usage	1	0,34 €/h	1.500h	510,00 €
			Total	**8.385,00 €**

Table 5.1: Project budget.

[1] https://cloud.google.com/compute/gpus-pricing

Chapter 6

Conclusions and Future Work

The first goal of this project was to tackle the high dimensional graph construction problem while having a better understanding of the intrinsic dimensionality of the data. We achieved this with a channel-wise approach, as an extension of the existing NNK graphs method. We performed an in-depth study of the benefits of this new method and the insights we can extract about the dimensionality of the data from the local geometry of the polytope and the relationship between channels.

Secondly, we employed the proposed graph construction method to derive a new generalization estimator in convolutional neural networks, based on channel-wise local polytope interpolation. Then we used it to detect the best generalization performance using only train data, and stop the training before converging to zero train error. We shown how this estimator can replace the standard generalization estimator (i.e. validation set performance) to perform early stopping and thus prevent overfitting. This method may be the preferred in cases of small datasets or small size neural networks where test performance is crucial.

Future work should focus on developing new efficient implementations of this generalization estimator, taking full advantage of the information between consecutive epochs and between channels, since on both axes we have detected a great opportunity to reduce complexity while maintaining a good generalization estimate. This could lead our method to standardize in other scenarios of different nature. Future research could also be in the direction of neural network pruning based on the CW-DeepNNK interpolation error, since we have seen that in certain channels we have practically no useful information to perform the interpolation, and those channels could be pruned, resulting in a more compact network with less computational cost.

Another line of research could involve a *progressive* early stopping of the full model, achieving a significant saving of gradient computation and backpropagation throughout the training. It would be interesting to investigate this new approach of improving generalization in stages, and the hypothesis that the low-level features learned in the first layers are very general and are learned quickly in the first few epochs of training, while the higher-level features learned in the deeper layers that are more specific to the training set reach their optimal point later. Obviously, all weights continue to update as a whole throughout training, but excessive training can overfit each group of weights to the training data without realizing it. By performing this progressive early stopping of the full model we could stop at the point of highest generalization each channel of each layer, perhaps thus avoiding the overfitting of the model to the training data in a more dedicated way and with a much better interpretation than the standard black box approach.

Finally, the work reported in this thesis reflects the main contributions of a scientific publication under progress that will be submitted to a signal processing conference.

Bibliography

[1] Sarath Shekkizhar and Antonio Ortega. Deepnnk: Explaining deep models and their generalization using polytope interpolation. *arXiv preprint arXiv:2007.10505*, 2020.

[2] Antonio Ortega, Pascal Frossard, Jelena Kovačević, José MF Moura, and Pierre Vandergheynst. Graph signal processing: Overview, challenges, and applications. *Proceedings of the IEEE*, 106(5):808–828, 2018.

[3] Davide Bacciu, Federico Errica, Alessio Micheli, and Marco Podda. A gentle introduction to deep learning for graphs. *Neural Networks*, 2020.

[4] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[5] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[6] Vincent Gripon, Antonio Ortega, and Benjamin Girault. An inside look at deep neural networks using graph signal processing. In *2018 Information Theory and Applications Workshop (ITA)*, pages 1–9. IEEE, 2018.

[7] Carlos Lassance, Vincent Gripon, and Antonio Ortega. Representing deep neural networks latent space geometries with graphs. *Algorithms*, 14(2):39, 2021.

[8] Carlos Lassance, Myriam Bontonou, Ghouthi Boukli Hacene, Vincent Gripon, Jian Tang, and Antonio Ortega. Deep geometric knowledge distillation with graphs. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8484–8488. IEEE, 2020.

[9] Carlos Lassance, Vincent Gripon, and Antonio Ortega. Laplacian networks: Bounding indicator function smoothness for neural networks robustness. *APSIPA Transactions on Signal and Information Processing*, 10, 2021.

[10] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[11] Sarath Shekkizhar and Antonio Ortega. Graph construction from data using non negative kernel regression (nnk graphs). *arXiv preprint arXiv:1910.09383*, 2019.

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

[13] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[14] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[15] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[16] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do cifar-10 classifiers generalize to cifar-10? *arXiv preprint arXiv:1806.00451*, 2018.

[17] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*. MIT press Cambridge, 2016.

[18] Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 1998.

[19] Maren Mahsereci, Lukas Balles, Christoph Lassner, and Philipp Hennig. Early stopping without a validation set. *arXiv preprint arXiv:1703.09580*, 2017.

[20] Antonio Ortega. *Introduction to Graph Signal Processing*. Cambridge University Press, 2021.

[21] Wei Dong, Charikar Moses, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web*, pages 577–586, 2011.

[22] Václav Chvátal and PL Hammer. Aggregations of inequalities. *Studies in Integer Programming, Annals of Discrete Mathematics*, 1:145–162, 1977.

[23] George H Chen, Devavrat Shah, et al. *Explaining the success of nearest neighbor methods in prediction*. Now Publishers, 2018.

[24] Ashish Kapoor, Hyungil Ahn, Yuan Qi, and Rosalind Picard. Hyperparameter and kernel learning for graph based semi-supervised classification. *Advances in neural information processing systems*, 18:627–634, 2005.

[25] Masayuki Karasuyama and Hiroshi Mamitsuka. Adaptive edge weighting for graph-based learning algorithms. *Machine Learning*, 106(2):307–335, 2017.

[26] Vassilis Kalofolias and Nathanaël Perraudin. Large scale graph learning from smooth signals. *arXiv preprint arXiv:1710.05654*, 2017.

[27] Vassilis Kalofolias. How to learn a graph from smooth signals. In *Artificial Intelligence and Statistics*, pages 920–929. PMLR, 2016.

[28] Nachman Aronszajn. Theory of reproducing kernels. *Transactions of the American mathematical society*, 68(3):337–404, 1950.

[29] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6. Ieee, 2017.

[30] Richard E Bellman. *Adaptive control processes: a guided tour*. Princeton university press, 2015.

[31] Charu C Aggarwal, Alexander Hinneburg, and Daniel A Keim. On the surprising behavior of distance metrics in high dimensional space. In *International conference on database theory*, pages 420–434. Springer, 2001.

[32] Stefano Recanatesi, Matthew Farrell, Madhu Advani, Timothy Moore, Guillaume Lajoie, and Eric Shea-Brown. Dimensionality compression and expansion in deep neural networks. *arXiv preprint arXiv:1906.00443*, 2019.

[33] Alessio Ansuini, Alessandro Laio, Jakob H Macke, and Davide Zoccolan. Intrinsic dimension of data representations in deep neural networks. *arXiv preprint arXiv:1905.12784*, 2019.

[34] Ryumei Nakada and Masaaki Imaizumi. Adaptive approximation and estimation of deep neural network with intrinsic dimensionality. *arXiv preprint arXiv:1907.02177*, 2019.

[35] Matthias Hein and Markus Maier. Manifold denoising. In *NIPS*, volume 19, pages 561–568, 2006.

[36] Jose A Costa and Alfred O Hero. Determining intrinsic dimension and entropy of high-dimensional shape spaces. In *Statistics and Analysis of Shapes*, pages 231–252. Springer, 2006.

[37] Elizaveta Levina and Peter J Bickel. Maximum likelihood estimation of intrinsic dimension. In *Advances in neural information processing systems*, pages 777–784, 2005.

[38] Claudio Ceruti, Simone Bassis, Alessandro Rozza, Gabriele Lombardi, Elena Casiraghi, and Paola Campadelli. Danco: An intrinsic dimensionality estimator exploiting angle and norm concentration. *Pattern recognition*, 47(8):2569–2581, 2014.

[39] Francesco Camastra and Antonino Staiano. Intrinsic dimension estimation: Advances and open problems. *Information Sciences*, 328:26–41, 2016.

[40] Elena Facco, Maria d'Errico, Alex Rodriguez, and Alessandro Laio. Estimating the intrinsic dimension of datasets by a minimal neighborhood information. *Scientific reports*, 7(1):1–8, 2017.

[41] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In *International conference on database theory*, pages 217–235. Springer, 1999.

[42] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[43] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[44] Yang Bai, Yuyuan Zeng, Yong Jiang, Shu-Tao Xia, Xingjun Ma, and Yisen Wang. Improving adversarial robustness via channel-wise activation suppressing. *arXiv preprint arXiv:2103.08307*, 2021.

[45] Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.

[46] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[47] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[48] Justin K Terry, Mario Jayakumar, and Kusal De Alwis. Statistically significant stopping of neural network training. *arXiv preprint arXiv:2103.01205*, 2021.

[49] David Duvenaud, Dougal Maclaurin, and Ryan Adams. Early stopping as nonparametric variational inference. In *Artificial Intelligence and Statistics*, pages 1070–1077. PMLR, 2016.

[50] André Elisseeff, Massimiliano Pontil, et al. Leave-one-out error and stability of learning algorithms with applications. *NATO science series sub series iii computer and systems sciences*, 190:111–130, 2003.

[51] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[52] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Appendix A

Experiment Details

A.1 Section 3.3 model

Group name	Operation	Number of filters	Filter size	Stride size	Padding size	Output size
Input image		–	–	–	–	$32 \times 32 \times 3$
Layer 0	Convolution	16	$3 \times 3 \times 3$	1×1	1×1	$32 \times 32 \times 16$
	ReLU	–	–	–	–	$32 \times 32 \times 16$
Layer 1	Convolution	16	$3 \times 3 \times 16$	1×1	1×1	$32 \times 32 \times 16$
	ReLU	–	–	–	–	$32 \times 32 \times 16$
	Max pooling	1	2×2	2×2	0	$16 \times 16 \times 16$
Layer 2	Convolution	16	$3 \times 3 \times 16$	1×1	1×1	$16 \times 16 \times 16$
	ReLU	–	–	–	–	$16 \times 16 \times 16$
Layer 3	Convolution	16	$3 \times 3 \times 16$	1×1	1×1	$16 \times 16 \times 16$
	ReLU	–	–	–	–	$16 \times 16 \times 16$
	Max pooling	1	2×2	2×2	0	$8 \times 8 \times 16$
Layer 4	Convolution	16	$3 \times 3 \times 16$	1×1	1×1	$8 \times 8 \times 16$
	ReLU	–	–	–	–	$8 \times 8 \times 16$
Layer 5	Convolution	16	$3 \times 3 \times 16$	1×1	1×1	$8 \times 8 \times 16$
	ReLU	–	–	–	–	$8 \times 8 \times 16$
	Max pooling	1	2×2	2×2	0	$4 \times 4 \times 16$
Output	Fully connected	–	–	–	–	No. classes
	Softmax	–	–	–	–	No. classes

Table A.1: CNN architecture used in Section 3.3.

A.2 Section 4.3 model

Group name	Operation	Number of filters	Filter size	Stride size	Padding size	Output size
Input image		–	–	–	–	$32 \times 32 \times 3$
Layer 0	Convolution	5	$5 \times 5 \times 3$	1×1	0	$28 \times 28 \times 5$
	ReLU	–	–	–	–	$28 \times 28 \times 5$
Layer 1	Convolution	5	$5 \times 5 \times 5$	1×1	0	$24 \times 24 \times 5$
	ReLU	–	–	–	–	$24 \times 24 \times 5$
	Max pooling	1	2×2	2×2	0	$12 \times 12 \times 5$
Layer 2	Convolution	5	$5 \times 5 \times 5$	1×1	0	$8 \times 8 \times 5$
	ReLU	–	–	–	–	$8 \times 8 \times 5$
Layer 3	Convolution	5	$3 \times 3 \times 5$	1×1	0	$6 \times 6 \times 5$
	ReLU	–	–	–	–	$6 \times 6 \times 5$
	Max pooling	1	2×2	2×2	0	$3 \times 3 \times 5$
Output	Fully connected	–	–	–	–	No. classes
	Softmax	–	–	–	–	No. classes

Table A.2: CNN architecture used in Section 4.3.

A.3 Hyperparameters

Parameters	Description
Data split	20% of the train set is held out for validation (only when not using NNK interpolation framework)
Weight initialization	He uniform [51]
Regularization	Dropout with rate = 0.2 in each layer (if *non-regularized model* not stated in experiment)
Loss	Cross-entropy loss
Batch size	50
Epochs	20 (if not stated in experiment)
Learning rate	0.001
Optimizer	Adam [52] with $\beta_1 = 0.9, \beta_2 = 0.999$

Table A.3: Hyperparameters for thesis experiments, using CIFAR-10 dataset.

Appendix B

Nonzero heatmaps of CNN activations

This appendix contains a graphical representation of the behavior of convolutional layers using nonzero heatmaps of their feature maps, both for a regularized model and a non-regularized model, using the same weight initialization and trained for the same epochs as detailed in Appendix A.1. In the regularized case, there is a large proportion of zero dimensions in the activations. But we also see that as expected, the patterns of active dimensions are different between classes. In the non-regularized case, we encounter an extreme binary behaviour: regardless of the input class, certain channels are always active, i.e. non-negative due to ReLU, while other channels are completely deactivated at zero.

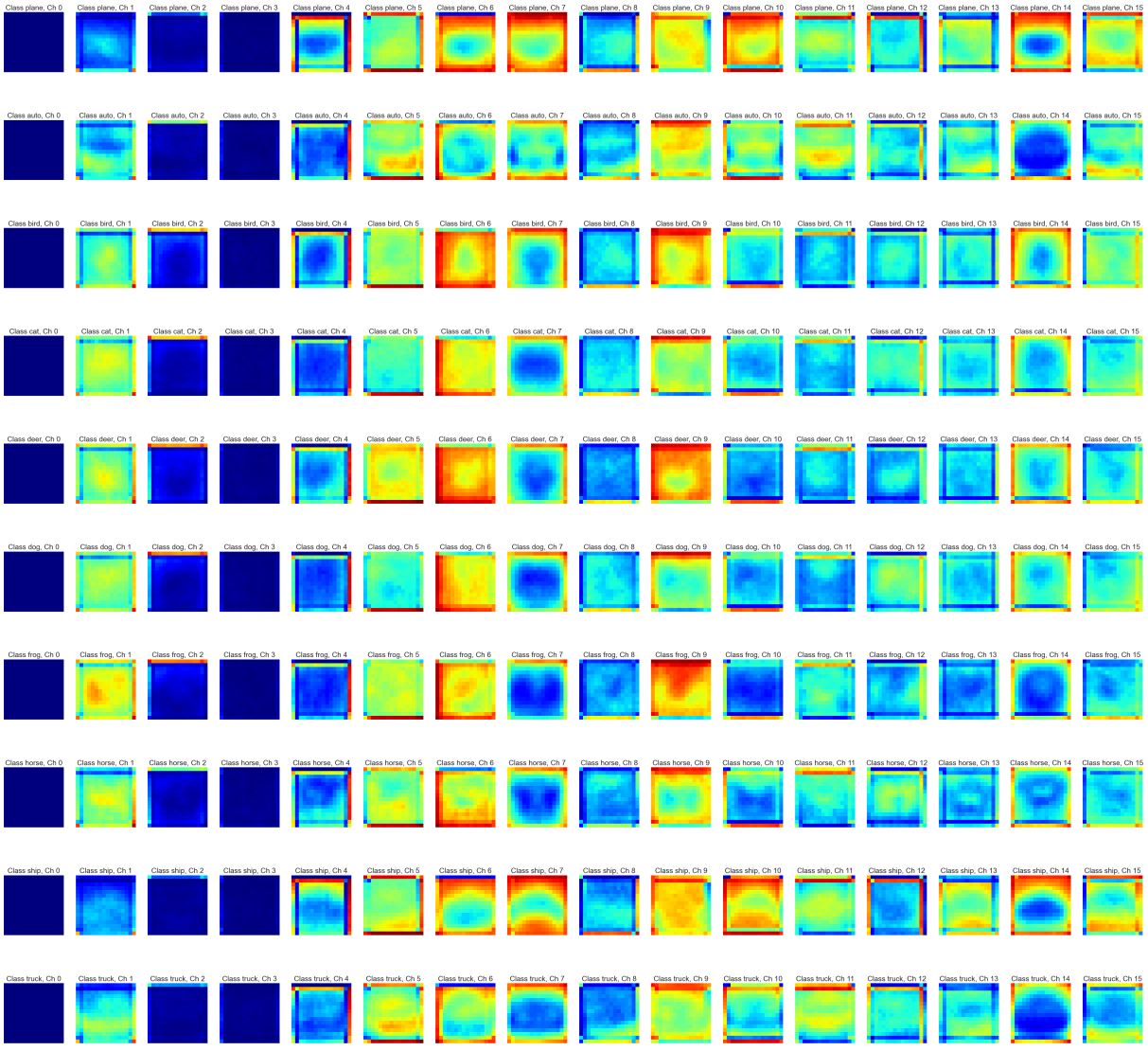


Figure B.1: Nonzero heatmaps of layer 2 of model A.1 using regularization during training, for the different CIFAR-10 classes.

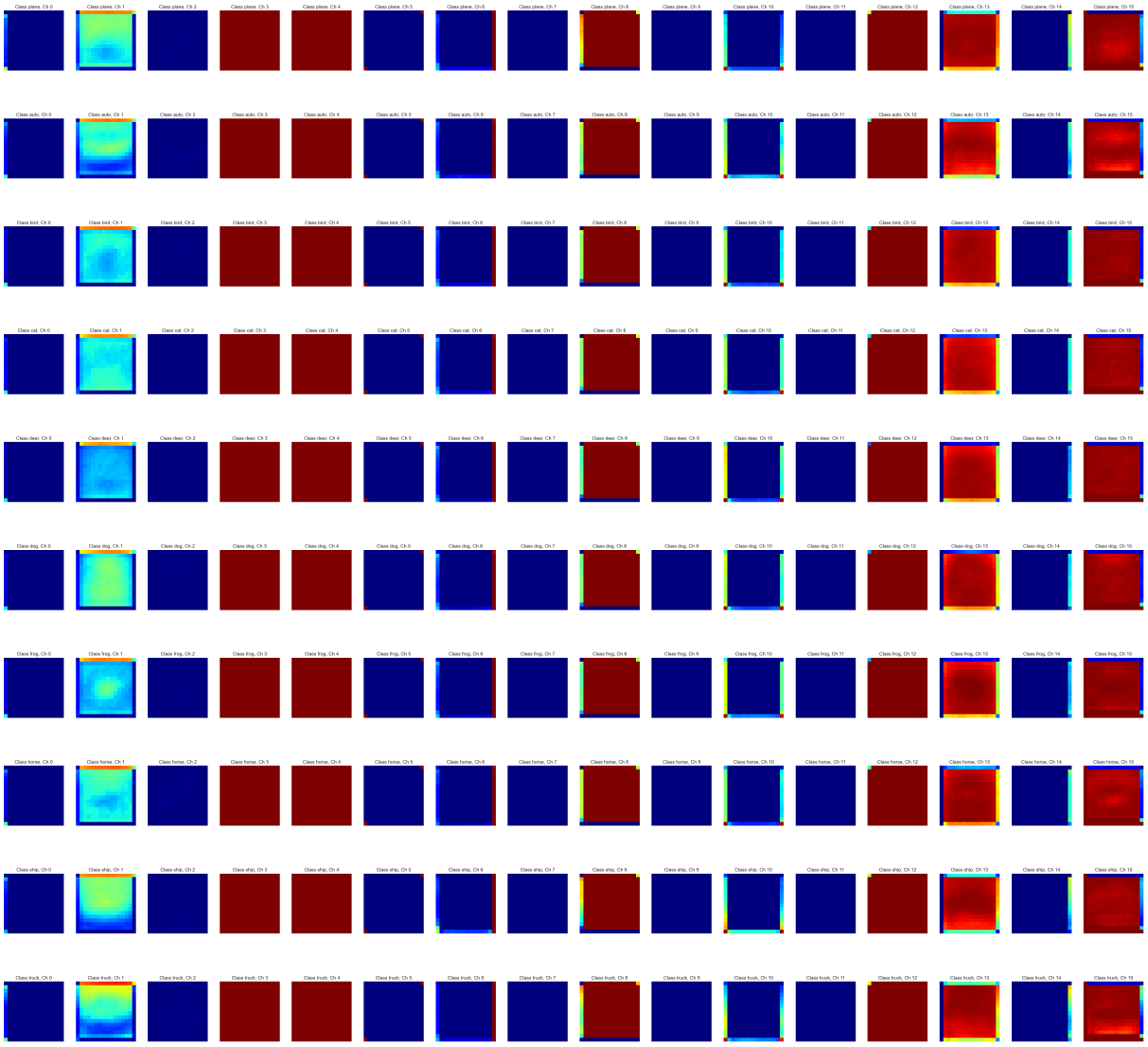


Figure B.2: Nonzero heatmaps of layer 2 of model A.1 using no regularization during training, for the different CIFAR-10 classes.