UNIVERSITY OF SOUTHAMPTON

Faculty of Engineering and Physical Sciences
School of Electronics and Computer Science

Towards an Understanding of Generalisation in Deep Learning: An Analysis of the Transformation of Information in Convolutional Neural Networks

by

Dominic Belcher
ORCiD: 0009-0006-7712-808X

A thesis for the degree of
Master of Philosophy

June 2025

University of Southampton

Abstract

Faculty of Engineering and Physical Sciences
School of Electronics and Computer Science

Master of Philosophy

Towards an Understanding of Generalisation in Deep Learning: An Analysis of the Transformation of Information in Convolutional Neural Networks

by Dominic Belcher

Despite their enormous size, Deep Neural Networks are able to achieve exceptional performance across a wide variety of machine learning problems, and have become a de facto standard in many areas of machine learning. The ability of such large models to reliably achieve good generalisation is difficult to reconcile with conventional theory on machine learning, which bounds the generalisation capability based on the size of the model, implying that more complex models should not — but importantly not that they cannot — reliably generalise.

In this work, I investigate generalisation within the specific domain of Convolutional Neural Networks (CNNs) applied to image classification problems. I investigate the way in which the layers of a CNN transform the data, and how this may entail the good generalisation performance these models exhibit. I investigate how margins between classes manifest and change, showing that the different operations in the network can increase or decrease the margin, as well as change the shape of the data in relation to the margin. I combine this with a replication and extension of the use of hidden layer probes to investigate how the classification problem changes through the network, showing that linear separability emerges through the networks, to an extent that almost matches the full classification performance of the network. I show how this linear separability aligns with some of the patterns seen in the class margins, and how the convolutions and activations work in tandem to both increase the margin and the linear separability. Finally, I extend the existing work on hidden layer probes to investigate globally pooled features within the model, showing that the information distilled by the network at each stage is primarily in coarse features, rather than at the pixel level.

Contents

List of Figures

List of Tables

Declaration of Authorship

I declare that this thesis and the work presented in it is my own and has been generated by me as the result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at this University;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. Parts of this work have been published as: Belcher et al. 2020

Signed:... Date:..................

Acknowledgements

I would like to thank my supervisor, Prof. Adam Prugel-Bennett, for his unfailing support through all the challenges I encountered in the course of this project.

I acknowledge the use of the IRIDIS High Performance Computing Facility, and associated support services at the University of Southampton, in the completion of this work.

Chapter 1

Introduction

Generalisation is, in essence, the fundamental goal of machine learning. In any machine learning task, we aim to learn a model which can generalise effectively on unseen data, thereby yielding a model which is well suited to the task at hand. To understand generalisation is therefore to understand machine learning, and the better our understanding of generalisation, the more informed our decisions can be in designing solutions to machine learning problems.

We can frame the concept of understanding generalisation using a variety of related but subtly distinct questions, which examine the problem from different angles and with varying levels of specificity. For example, we may frame generalisation by the question *why do certain machine learning models generalise?*. While this may be the fundamental question we wish to answer, this question is exceptionally broad in covering all aspects of machine learning, and any answer broad enough to apply to any area, any task, or any model may do little to inform our choices in any specific case. We may ask a more specific question of *what does a machine learning model learn in order to exhibit good generalisation?*. This question may be applied to any specific model on any specific task, and may have starkly different answers depending on this. By "good generalisation" we generally mean that a model achieves good performance on unseen data, although this too will vary across different tasks, particularly between tasks of supervised and unsupervised learning. Our definition of "good" is also subjective, and may often be taken to mean "equal to or better than can be achieved through other methods" — or equivalently "state-of-the-art" — and so over time this will likely shift as methods exhibiting better generalisation are developed.

These factors make any investigation of generalisation, outside of the purely theoretical, highly context dependent, both in the context of the problems being investigated, and the temporal context in which it is undertaken, relative to an existing baseline of what may be considered "good". The context in which this research takes place is against a backdrop of more than a decade of renewed interest and rapid advancement in the field of deep learning, and the use of deep neural networks in solving machine learning problems (Choudhary et al. 2022). The use of deep learning and deep neural networks has yielded exceptionally impressive performance on a broad range of complex tasks, such as image classification, object detection, speech recognition and text generation, to name but a few. Despite the extraordinary progress seen with deep learning methods, generalisation within deep learning is far from well understood, and the question of what deep learning models learn that yields the exceptional generalisation they can exhibit is largely unanswered (Zhang et al. 2021). Indeed neural networks are often considered to be a "black box" (Alain et al. 2017), in which the learning of the model is largely not understood, and the reason for any particular output from the model is not possible to explain. While the operations that make up the model are often simple and easily understood in isolation, when taken together these form a complex connected system which cannot easily be explained, and can exhibit unintended or unexpected behaviour.

Understanding generalisation of deep learning models is therefore an exceptionally complex problem, however the potential benefits of understanding this are equally significant. Understanding what deep learning models learn in achieving good generalisation would yield the ability to make informed architectural choices in designing more effective models, and could most importantly allow for designing models with more predictable and explainable behaviour, which is of particular importance when utilising models in real world settings.

Deep learning also presents us with an additional problem in understanding generalisation, when compared with many other machine learning methods. A specific phenomenon often seen in using deep learning methods is that there is little, if any, correlation between the complexity of a model, and the ability of the model to achieve good generalisation. This appears to run contrary to the traditional understanding of generalisation, in which it is considered to be the case that the more complex a model is, the less certain it is possible to be that the model will generalise well.

The primary contributions of this work are in chapters 3 and 5. In chapter 3 I conduct an investigation into how the size of the class margins change through the layers of a CNN. In chapter 4 I replicate existing work on hidden layer probes (Alain et al. 2017) on the specific models and datasets under investigation, and relate the findings to the work in chapter 3. I also attempt to extend the use of hidden layer probes to use multilayer perceptron (MLP) probes. An earlier version of this work was accepted to the workshop "Deep Learning through Information Geometry" at NeurIPS 2020 (Belcher et al. 2020). In chapter 5 I conduct an experiment using hidden layer probes on globally pooled features of the latent representation, to investigate the significance of pixel-level vs. feature-level information, and relate this to the findings of chapters 3 and 4.

Chapter 2

Background and Prior Literature

The topic of generalisation has long been studied within the field of Machine Learning, with a significant body of work predating the recent rise in popularity of Deep Learning, as well as a wealth of more specific investigation of generalisation within the domain of Deep Learning.

The study of generalisation in Machine Learning is strongly linked to the theoretical foundations of the subject as a whole, and while the term generalisation was not always used, much of the early theoretical study of Machine Learning provides foundations for investigation of generalisation. Pinpointing the earliest discussions of generalisation as a specific field, and plotting an exact timeline of the evolution of the field, is somewhat tricky and not essential for this research. I aim instead to simply lay out key ideas relevant to the study of generalisation.

A core concept in generalisation is the problem of Risk Minimisation (Vapnik 1991), as this provides a mathematical framework for analysing generalisation, around which much of the relevant work is based. Risk minimization considers minimising the risk of a model over the class of functions that model represents, which is characterised by the variable weight parameters of the model, given a joint distribution of the data and target, i.e. the domain and co-domain of the function we attempt to learn. Formally the risk is defined as

$$R(w) = \int L(y, f(x,w))dP(x,y)$$

where we aim to minimise the risk $R(w)$ over the class of functions $f(x,w)$, where $F : X \times W \to Y$ is a function over the input X parameterised by weights $w \in W$, $L(\cdot,\cdot)$ is some loss function and $P(x,y) = P(y|x)P(x)$ is the joint probability distribution over X (the data space) and Y (the target space).

When learning on real data, this joint distribution is not known, and only a finite number of data points will be available. The risk must therefore be estimated using the empirical risk function

$$E(w) = \frac{1}{l} \sum_{i=i}^{l} L(y_i, f(x_i, w))$$

using the training data $\{(x_i, y_i) \in X \times Y\}_{i=0}^{l}$. This frames the learning process as a problem of empirical risk minimisation, where it is assumed that minimising the empirical risk will also (at least approximately) minimise the true risk.

The problem of risk minimisation can be seen as an analogue to generalisation. Good generalisation implies that the difference between the empirical risk and true risk is small, and in particular when we achieve low empirical risk that we also achieve low true risk.

Moreover generalisation is exactly the empirical risk minimisation (ERM) principle: that the function $f(x, w_l)$ which minimises $E(w)$ over the set $w \in W$ gives a true risk $R(w_l)$ close to the minimum , i.e. that learning on a training set should generalise to the underlying problem. It can be shown that the empirical risk must converge uniformly to the true risk as

$$Prob\{\sup_{w \in W} |R(w) - E(w)| > \epsilon\} \to 0 \quad \text{as} \quad l \to \infty$$

in order for the ERM principle to be consistent (Vapnik 1991).

Vapnik and Chervonenkis built on this theory further to establish bounds on the rate of convergence, based solely on a measure of the capacity of the learning machine, and independent of the joint distribution $P(x, y)$. The capacity of a learning machine is a somewhat abstract concept of the general expressiveness of the functions the machine can learn, however in their work Vapnik and Chervonenkis establish a quantifiable measure of this capacity, termed the VC dimension (Shalev-Shwartz et al. 2014).

The VC dimension is, informally, the maximum number of data points which can be shattered in all possible ways by the functions expressible by a learning machine. The formal definition of shattering and the VC dimension is often restricted to binary labelling problems for simplicity, that is learning a function from $X \to \{0, 1\}$. In this case a hypothesis class is said to shatter a set of points C if the hypothesis class is equivalent to the set of all functions $C \to \{0, 1\}$.

However, the bounds established by Vapnik and Chervonenkis require $\frac{l}{h}$, that is the ratio of the number of data points to the VC-dimension, to be large. Where this ratio is small, the bounds become weak, and so these bounds are not useful for high capacity machines. This is of specific relevance to deep learning, where the number of parameters in deep neural networks is so great that the capacity becomes almost unboundedly large. While many of the most notable and best performing deep neural networks that have been developed are trained on datasets that would usually be considered very large, these datasets are still dwarfed in size by the number of parameters, and thus the capacity, of these models. For example, second place in the current state of the art (*ImageNet Benchmark* 2024) for image classification on ImageNet (Deng et al. 2009) is, at time of writing, 91% top-1 accuracy, held by the CoCa model (Yu et al. 2022), which has over two billion parameters. This is several orders of magnitude larger than the number of data-points, which is around 14 million. First place is held by OmniVec (Srivastava et al. 2023), which does not state the number of parameters in the model.

Naively, deep learning appears to undermine this theory, since such large models are able to consistently achieve such impressive generalisation performance. However, it is important to note that the bounds established by Vapnik and Chervonenkis are just that — bounds — and that these do not preclude high capacity models from generalising, they merely indicate that generalisation cannot be guaranteed for high capacity models. Nonetheless, it is very much significant that deep learning models do seem to reliably generalise, as these bounds would indicate that generalisation should not be reliably achieved by high capacity models, even if such generalisation is possible. Moreover, deep learning techniques seem to be uniquely capable of achieving good generalisation with high capacity models, and prior to the advent of deep learning it was believed that high capacity models would not reliably generalise. This raises the question fundamental to much of the recent study on generalisation in Deep Learning, that is: why do deep learning models generalise?

Since the advent of deep learning, much of the study of generalisation has been focused on attempting to answer this question, and to reconcile the established theory with empirical results observable with deep learning.

It can be shown that the VC dimension of a neural network can be bounded by the width and depth of the network, specifically that the VC dimension is of order $O(|E|log(|E|))$, where E is the set of edges, or equivalently the weights, of the network (Shalev-Shwartz et al. 2014). Some research has focused on improving this bound by considering other factors. Neyshabur, Tomioka, et al. 2015 introduced a framework for controlling the capacity of networks based on l_p norm regularisation, showing that for deep ($d > 2$ where d is the number of layers) networks, l_p regularisation was not sufficient for controlling the capacity of a network without also controlling for width of the network, in contrast to linear models and two-layer networks.

Empirical investigation by Keskar et al. 2017 suggested that larger training batches in stochastic gradient descent led to sharper minima in the training and test losses, which would in turn lead to poorer generalisation (Keskar et al. 2017). However, further work by Neyshabur, Bhojanapalli, et al. 2017 showed that sharpness was insufficient to control the capacity of a network (Neyshabur, Bhojanapalli, et al. 2017).

It has been suggested that over-parameterisation may be an important factor in the generalisation of deep neural networks, in contrast to the classical view that models should be under-parameterised to avoid over-fitting. Neyshabur, Li, et al. 2018 developed new capacity bounds for two layer neural networks based on an alternative complexity measure, that involves decomposing the complexity of the network as a whole into the complexities of the hidden units within the network (Neyshabur, Li, et al. 2018). The authors showed that this capacity bound decreases with the increasing number of hidden units, i.e. the width of the network, however these bounds are still much larger than the amount of training data, and so are unlikely to be sufficient to fully explain the generalisation patterns in deep learning generally.

Belkin et al. 2019 conducted further work on over-parameterisation, investigating over-parameterisation beyond the point where a model can perfectly fit the training data. The authors showed that in certain scenarios, increasing the capacity of a model up to and then beyond the point of interpolating the training data, the empirical risk would follow a U-shape curve, decreasing to a minimum and then increasing as the capacity approached interpolation, however beyond the point of interpolation the risk would decrease again in a so-called “double descent”, potentially decreasing to a point lower than a minimum in the U-shaped curve. This behaviour could be observed in two layer neural networks, as well as with decision trees and random forest models. These results would suggest that a high degree of over-parameterisation is actually a beneficial quality in achieving good generalisation, inverting the conventional understanding of the importance of not over-fitting. This idea has been explored in more detail, with work by Nakkiran et al. 2019 showing that the double descent curve could be observed in commonly used neural networks, as as ResNets and transformers, and that the double descent curve could also occur as a function of training time, as well as model complexity (Nakkiran et al. 2019).

Much of the earlier work on generalisation — both in deep learning and machine learning more broadly — focused primarily on the models used in learning, and attempted to bound the generalisation capability based solely on properties of the model. Some more recent work however has focused more closely on the data used in machine learning problems. The work of Baldock et al. 2021 investigated the concept of ”example difficulty” for the data, a notion of how easily data points are classified by a model. In their work, the authors used k-nearest-neighbour (kNN) classifiers to examine how deep within networks individual data points are assigned to classes, by taking the earliest layer at which the kNN classifier gives the same class as the network for each data point. Their findings suggest that predictions occur in later layers for more difficult individual data points, and more difficult datasets. The use of kNN probes suggests that the networks group together data points from the same class, and that individual data points are more difficult to classify if they appear to be further from, and so more different to, other points within the same class.

One of the most notable investigations of generalisation in deep learning, and of particular significance to this work, is the work of Zhang et al. 2017. In this work the authors demonstrated that large CNN models, such as AlexNet (Krizhevsky, Sutskever, et al. 2012), could be trained to achieve 100% training accuracy on variants of the ImageNet classification problem, where either the training labels were randomised, or indeed the actual images were replaced with random noise. This is a particularly striking result, as it illustrates that such models have the capacity to memorise the entirety of large datasets, and could therefore achieve perfect training accuracy in these problems without learning anything about the underlying structure of the data, which would be expected, and likely required, for achieving good generalisation. As such this shows that there is no inherent reason why a large capacity model which achieves good or even perfect training performance should be able to generalise, and yet these models, and indeed many other models, are able to do exactly this. Moreover a random labelling of the data, or indeed labels attached to random noise, would give no correlation between any of the data, be it training or test data, and as such fitting these training labels would necessarily give a model with no generalisation capability whatsoever. It must therefore be impossible to bound the generalisation capability of any model capable of fitting random data, however as stated this directly contradicts the behaviour we can observe with such models.

We should note that while these models may not be considered large by the standards of state of the art models, at the time of their development, and the time of the authors publication, these models were much closer to the state of the art in size. In particular AlexNet is notable as being once a state of the art model for image classification on ImageNet, with its publication being commonly cited as the start of the Deep Learning era. (Alom et al. 2018). Moreover, since these models have long since been surpassed in both size and performance, this further illustrates the significance of these findings, as modern state of the art models would likely be able to memorise even larger datasets, and yet can achieve superior performance still.

Despite work showing that the capacity of models can be controlled in various ways, and that this can entail tighter generalisation bounds, we still see demonstrably high capacity models that are able to generalise. This motivates the question of how and why these models do achieve such good generalisation, despite their high capacity, and the clear fact that learning in a way which does generalise is in no way required to fit the training data.

Chapter 3

Empirically Estimating Class Margins in Convolutional Neural Networks

It appears from previous work, such as that of Baldock et al. 2021 and Zhang et al. 2017, that a holistic view of machine learning — which considers the whole problem, including both the models and the data — is important for fully understanding the nature of generalisation. The work of Baldock et al. 2021 suggests that not all data is equal, and some individual data points are more easily learned on than others, and so it would seem impossible to wholly capture the nature of generalisation without giving consideration to the data. Moreover, it would seem that the interplay of the model and data is critical to generalisation - we know that certain model architectures are better suited to certain problems and certain types of data - an example being the effectiveness of CNNs in image classification. Therefore we understand fundamentally that the data *is* important, and that some data can simply be more effectively learned on by particular models. Indeed if this were not the case, there would be no value in investigating different models in different settings, as any properties of the data could simply be ignored.

This re-frames the question of generalisation from being simply "*why do deep neural networks generalise so well?*" to "*why are certain deep neural networks able to generalise in certain problems?*", or perhaps more insightfully "*how do certain deep neural networks interact with certain data in a way which entails good generalisation?*".

Re-framing the question in this way motivates considering what function a model performs on the data. In the case of neural networks, data generally flows forward through the layers of the network, being transformed into a particular latent representation at each layer. Focusing specifically on CNNs, these tend to be composed of one or more convolutional layers, each paired with an activation function following the convolution. These convolutional layers then typically followed by multi-layer perceptron (MLP) which outputs predicted class logits. The MLP typically consists of one or more fully connected layers, with activations between them, and a final linear classifier, which outputs the logits used to train and evaluate the model.

It seems clear in this case that the convolutional portion of the network is significant, and that this performs some transformation on the data which makes it more easily classified by the MLP. Considering the work of Baldock et al. 2021, transforming *all* the data in such a way that it was more easily classified would simplify the entire classification problem, making it easier to achieve good generalisation.

The work of Zhang et al. 2017 showed that high capacity models are capable of memorising entire datasets, and that these models can learn random labellings or entirely random data. However, random labellings and random data are inherently complex problems, indeed they are infinitely complex, with no solution being possible. Therefore, a simple problem should be one in which the dataset need not be memorised, rather generalisable patterns can be learned which allow for generalisation without memorisation.

I hypothesise that the transformation of data by certain models, in this case CNNs in image classification, makes the data more easy to learn on, and thereby simplifies the underlying problem in a way which allows for learning more generalisable patterns. The question to ask is therefore: *"how does a CNN transform image data in a way which makes it easier to classify?"*. It is therefore essential to consider what can make data more easier to classify, and how such properties could be manifested by a CNN.

This principle of a simple problem being generalisable without memorisation can be illustrated by way of a trivial example. If we consider a dataset of pairs (x_i, y_i) where $x \in \{0,1\}$, $y \in \{0,1\}$, and $x_i = y_i$ for all i, where x_i is a feature and y_i the class label, this gives us the simplest problem we could consider, as the single feature of the data directly codes the class label. It is straightforward to see that any machine trained on this data has no need to memorise the entire dataset, as training on any number of examples is equivalent. It is also the case that any deterministic machine can only ever have one of the following functions: output the correct class label in every case, output the incorrect class label in every case, or output a single class label in every case. Therefore the training and test performance of any machine will always be equal, and so perfect performance can be achieved with a machine of any capacity.

This example is the extreme case of closeness between points within a class, and is contrived in the sense that all data points within a class are clustered together, rather than individual points being close to some others within the same class, but not them all.

We can consider a more interesting case in which properties of the data more closely match those in real world data. We consider a dataset (X, Y) where X is a p-dimensional feature vector and $y \in \{-1, 1\}$, where the data are distributed according to $p_{X,Y}(\boldsymbol{x}, y) = p_{X|Y}(\boldsymbol{x} \mid y)\, \mathbb{P}_Y[Y = y]$

$$p_{X|Y}(\boldsymbol{x} \mid y) = \mathcal{N}(\boldsymbol{x} \mid y\, \Delta\, \boldsymbol{x}^*, \mathbf{I}_p) \qquad \mathbb{P}_Y[Y = y] = \frac{1}{2}\left([\![y = 1]\!] + [\![y = -1]\!]\right) \tag{3.1}$$

where $[\![\text{predicate}]\!]$ is an indicator function, Δ is the distance along some $\boldsymbol{x}^* \in \mathbb{R}^p$ from the mean of each class to the origin, such that the means are separated by 2Δ, and $|\boldsymbol{x}^*| = 1$.

That is, we have two standard multivariate Gaussian distributions, each corresponding to the data points for one of the two classes, with the mean of each being some distance Δ from the origin, and the separation between the two means being 2Δ.

This dataset has the property that, in a sufficiently high number of dimensions, the expected distance between two randomly selected points from the same class, and two randomly selected points from different classes, both become dominated by the number of dimensions, and so the size of Δ becomes insignificant. The expected distance between points within the same class is exactly $\sqrt{2p}$, while the expected distance between points from different classes is exactly $\sqrt{2p + (2\Delta)^2}$. We should note that Δ does not need to be large in order to guarantee the complete separation of the classes, since the probability of a point from one class crossing the true decision boundary, that is the hyperplane intersecting the origin perpendicular to $\boldsymbol{x}^*$, is exactly the probability $p(Z > \Delta)$ for $Z \sim N(0, 1)$, regardless of the number of dimensions, which even for $\Delta = 4$ is approximately 0.003%.

If we consider classification on this dataset using a perceptron with weight vector $\boldsymbol{w}_h \in S^{p-1}$ (the unit sphere in p dimensions) which predicts according to $h(\boldsymbol{X}) = \text{sign}(\boldsymbol{w}_h^\intercal \boldsymbol{X})$. we can then define a loss function for hypothesis h (the perceptron with weights $\boldsymbol{w}_h$) as

$$L_h(\boldsymbol{X}, Y) = [\![h(\boldsymbol{X}) \neq Y]\!]$$

The risk of a hypothesis, h, is then given by

$$R_h = \Phi(-\Delta\, \cos(\theta_h)) \tag{3.2}$$

where $\cos(\theta_h) = \boldsymbol{w}_h^\intercal \boldsymbol{x}^*$ and $\Phi(z)$ is the cumulative distribution function for a standard normal distribution.

If we assume we are given m training examples $\{(\boldsymbol{X}^\alpha, Y^\alpha) | \alpha = 1, 2, \ldots, m\}$ drawn independently from $p_{X,Y}$. We then consider a *Hebbian* classifier where we choose the weight vector $\boldsymbol{w} = \bar{\boldsymbol{w}}/|\bar{\boldsymbol{w}}|$ where

$$\bar{\boldsymbol{w}} = \sum_{\alpha=1}^{m} Y^{\alpha}\, \boldsymbol{X}^{\alpha}. \tag{3.3}$$

Since the risk of a weight vector with angle θ from $\boldsymbol{x}^*$ is given by $\Phi(-\Delta\,\cos(\theta))$, the expected risk using the Hebbian classifier can be approximated by

$$\bar{R} \approx \bar{R}_{approx} = \Phi\left(-\frac{\Delta}{\sqrt{1+\frac{p}{m\,\Delta^2}}}\right) \tag{3.4}$$

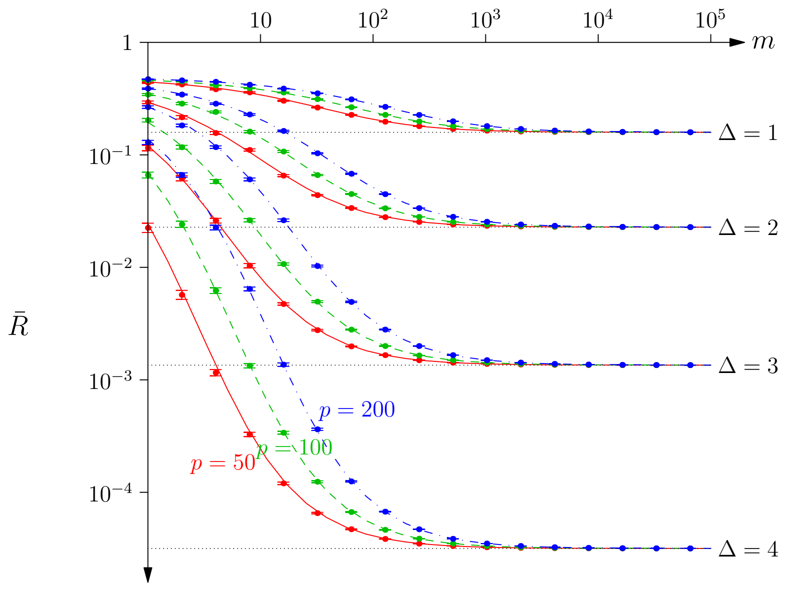


FIGURE 3.1: The expected risk of the simple learning machine versus the number of training examples, m (dashed lines), along with simulation results averaged over 100 runs (markers).

Figure 3.1 shows the expected risk versus the number of training examples m for feature vectors of length $p = 50, 100$ and 200, and for different values of the class separation $\Delta = 1, 2, 3$ and 4. Figure 3.1 also shows simulated results for the risk averaged over 100 runs. The simulated risks and the expected approximated risks are almost identical, indicating that this is a very good approximation. This also clearly illustrates that even for large numbers of features, as the class separation increases the risk becomes very small, for sufficiently large numbers of training examples.

Relating these insights to the case of classification in a convolutional neural network, I considered the hypothesis that the convolutional layers within the network induce an increase in the margins between classes, thereby transforming the raw image data into a latent space in which the classification task is more simple. I sought to test this hypothesis.

3.1 Methodology

I designed an experiment in which I can empirically investigate this hypothesis, using well established CNN model architectures. There may not always exist a directly measurable margin between the data in two separate classes, and it is almost certainly the case that the data cannot be separated by a maximum margin hyperplane, as this would imply that the data could be linearly classified, and so a deep neural network would be unnecessary. I therefore aimed to establish a proxy for the margin which we can quantify in a meaningful way. I propose a method of introducing noise to the latent representation of the data, and measure the tolerance of the network to varying levels of this noise as a proxy for the margin. It is a reasonable assumption that the more tolerant a classifier is to this noise, the greater the margin between the data at the point where the noise is introduced.

I did this by taking a k-layer network F, trained on some dataset $D = (X, Y)$, and split the network at some layer l, into two networks G_l and H_l, where H_l is the network composed of the layers of F up to and including layer l, and G_l is the network of layers of F after layer l, and so where $F = H_l \circ G_l \forall l \in \mathbb{Z}, 0 \leq l \leq k$. There are two special cases, $l = 0$ where G_l is the identity function and $H_l = F$, and $l = k$ where $G_l = F$ and H_l is the identity function. We then use G_l to project the data X into a latent representation Z_l, where $Z_l = G_l(X)$. I then added noise to this latent representation by generating noisy data points $\boldsymbol{z'} \in Z_l$ as

$$\boldsymbol{z'} = \boldsymbol{z} + \alpha \sum_{i=1}^{N} \eta_i \sqrt{\sigma_i} \boldsymbol{v_i}$$

Where $\boldsymbol{z} = G_l(x)$ is the latent representation of an image x, α is a parameter varied to control the magnitude of the noise, σ_i is the ith largest singular value and $\boldsymbol{v_i}$ the corresponding singular vector of the data in the latent space, N is the number of noise components, and $\eta_i \sim \mathcal{N}(0, \mathbf{I})$ is random uniform noise. The singular values and singular vectors were calculated through the singular value decomposition (SVD) of a subset of the data, using 2048 randomly sampled data points.

H_l is then used to predict labels for the corrupted data, and the classification accuracy measured against the true labels y. By varying the parameter α it is possible to control the amplitude of the noise, and measure the tolerance by the degree of noise needed to reduce the accuracy by a fixed amount, or alternatively the accuracy at a fixed degree of noise.

By introducing noise according to the most significant principal components of the dataset, I aimed to account for variations in the distribution of the data, since the distribution of the data is unknown. Further, by scaling the noise by the corresponding singular values, it is possible to normalise for variations in the scale of the data without having to re-scale the entire latent space.

I conducted experiments on models in the VGG family (Simonyan et al. 2015), those being well established convolutional neural networks which can achieve good performance in image classification, but which are sufficiently small enough that I was able to perform experiments using them without the need for excessively powerful hardware or unnecessary wasting of compute time. I used weights pre-trained on ImageNet, and performed experiments testing the noise tolerance on CIFAR-10 (Krizhevsky and Hinton 2009) and TinyImageNet (mnmoustafa 2017) by fine-tuning the weights for these datasets. I experimented with VGG models of varying depth, using VGG11, VGG13, VGG16 and VGG19, to investigate the significance of model depth on noise tolerance. I split each model after every operation, including activation functions, pooling and normalisations, in order to investigate the impact these operations have on the margins between classes. Table 3.1 shows the compositions of these models, with the layer number for each operation. I chose $N = 50$ singular vectors for our noise, and vary the noise between 0 and 5 (inclusive) in increments of 0.1. I average the results over 10 trials for greater accuracy. In each case I measured the accuracy over a test partition of the dataset not used in fine-tuning the model. Following the method used in training the pre-trained weights, the images are scaled up to 256x256 pixels, then centre-cropped to 224x224 pixels.

3.2 Results

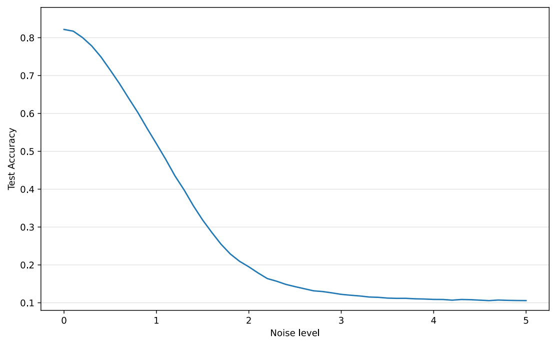
For any model/dataset/split combination, it is possible to plot a curve showing the relationship between introduced noise and test accuracy, to give an indication of the noise tolerance. Examples of some of these curves are shown in figure 3.2. We can see how the noise tolerance varies in each case, with the accuracy dropping off steeply in some, and more gradually in others, as the noise level increases. For example the plot for VGG11 at split 0 drops off quickly, showing less noise tolerance, while the plot for VGG16 drops off much more gradually, showing greater noise tolerance. We could plot these curves across different splits for the same model and dataset, to attempt to compare how the noise tolerance varies at different splits. However, these curves do not give us an easy way to compare the noise tolerance directly between the many layers of a model, or to visualise how the tolerance changes through the model. While we can compare curves for different layers, the more layers a model has the more difficult it becomes to use these curves to gain an understanding of the changes through the model.

Instead, we can approximate the area under these curves using Simpson's rule, and — since the curves should be monotonically decreasing — we should be able to use this

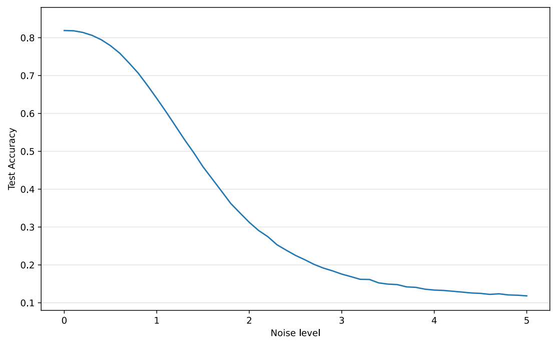
Layer	VGG11	VGG13	VGG16	VGG19
0	Raw input	Raw input	Raw input	Raw input
1	Conv2D	Conv2D	Conv2D	Conv2D
2	ReLU	ReLU	ReLU	ReLU
3	MaxPool2D	Conv2D	Conv2D	Conv2D
4	Conv2D	ReLU	ReLU	ReLU
5	ReLU	MaxPool2D	MaxPool2D	MaxPool2D
6	MaxPool2D	Conv2D	Conv2D	Conv2D
7	Conv2D	ReLU	ReLU	ReLU
8	ReLU	Conv2D	Conv2D	Conv2D
9	Conv2D	ReLU	ReLU	ReLU
10	ReLU	MaxPool2D	MaxPool2D	MaxPool2D
11	MaxPool2D	Conv2D	Conv2D	Conv2D
12	Conv2D	ReLU	ReLU	ReLU
13	ReLU	Conv2D	Conv2D	Conv2D
14	Conv2D	ReLU	ReLU	ReLU
15	ReLU	MaxPool2D	Conv2D	Conv2D
16	MaxPool2D	Conv2D	ReLU	ReLU
17	Conv2D	ReLU	MaxPool2D	Conv2D
18	ReLU	Conv2D	Conv2D	ReLU
19	Conv2D	ReLU	ReLU	MaxPool2D
20	ReLU	MaxPool2D	Conv2D	Conv2D
21	MaxPool2D	Conv2D	ReLU	ReLU
22		ReLU	Conv2D	Conv2D
23		Conv2D	ReLU	ReLU
24		ReLU	MaxPool2D	Conv2D
25		MaxPool2D	Conv2D	ReLU
26			ReLU	Conv2D
27			Conv2D	ReLU
28			ReLU	MaxPool2D
29			Conv2D	Conv2D
30			ReLU	ReLU
31			MaxPool2D	Conv2D
32				ReLU
33				Conv2D
34				ReLU
35				Conv2D
36				ReLU
37				MaxPool2

TABLE 3.1: Composition of each model being tested

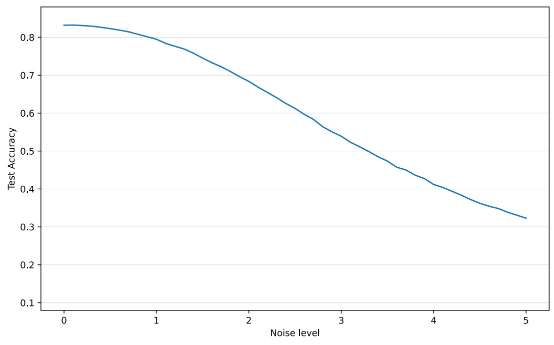
area as a proxy for the noise tolerance, as greater noise tolerance should give a curve which drops off more gradually, and so would have a greater area underneath than a curve which drops off more sharply. This area should then gives us a proxy for measuring the margins between classes at that point in the network. We should note that there is a minimum possible area under each graph, since we do not expect the accuracy to be below random performance, which for CIFAR-10 is 0.1 (as there are 10 classes), and for Tiny Imagenet is 0.005 (as there are 200 classes). This means the minimum area for CIFAR-10 is 0.5, while for Tiny Imagenet it is 0.025. We therefore subtract these values from the calculated areas to give us our approximated noise tolerance. Since we assume that accuracy should never increase as we increase the noise scale, we can calculate the maximum noise tolerance we expect to see for each network, which would be the area under a graph where the accuracy is constant for all noise values, which is equal to $(Acc_{max} - Acc_{baseline}) \times Noise_{max}$, where the maximum noise scale $Noise_{max}$ is 5.



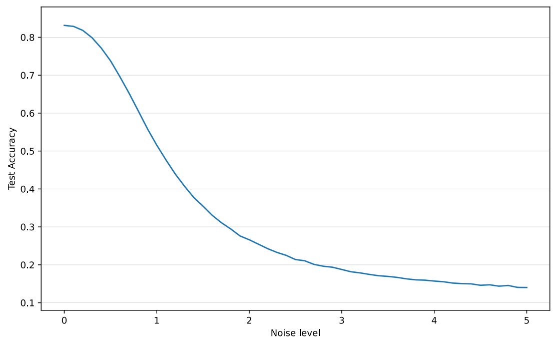
(A) Model=VGG11, Split=0



(B) Model=VGG13, Split=10



(C) Model=VGG16, Split=20



(D) Model=VGG19, Split=30

FIGURE 3.2: Accuracy vs. Noise plots for different models and splits on CIFAR-10 . Split indices correspond to the layers in table 3.1

We can illustrate this with an example of these curves. Figure 3.3 shows a plot of the accuracy vs. noise for VGG11 on CIFAR-10 , split at layer 11, with the area under the curve highlighted. The calculated area gives the noise tolerance value. The area is calculated only between the curve and the baseline of $y = 0.1$, since this is the minimum accuracy we expect to see on CIFAR-10 .

Table 3.2 shows the maximum accuracies and noise tolerances for each model/dataset combination. This theoretical maximum noise tolerance is the same for any layer, since it is proportional to the accuracy of the network when no noise is introduced.

Model	VGG11		VGG13		VGG16		VGG19	
Dataset	CIFAR10	Tiny Imagenet	CIFAR10	Tiny Imagenet	CIFAR10	Tiny Imagenet	CIFAR10	Tiny Imagenet
Acc_{max}	0.822	0.570	0.819	0.556	0.832	0.573	0.831	0.587
Max tolerance	3.610	2.825	3.595	2.755	3.660	2.840	3.655	2.910

TABLE 3.2: Maximum accuracy and theoretical maximum noise tolerance for each model and dataset combination

Figure 3.4 shows a plot of this noise tolerance proxy through the VGG11 network on CIFAR10, while Figure 3.5 shows the same plot for the VGG11 network on Tiny Imagenet. Complete results are shown in Appendix A, with plots for each model and dataset combination, in figs. A.1 to A.4 for CIFAR-10 and figs. A.5 to A.8 for Tiny Imagenet.

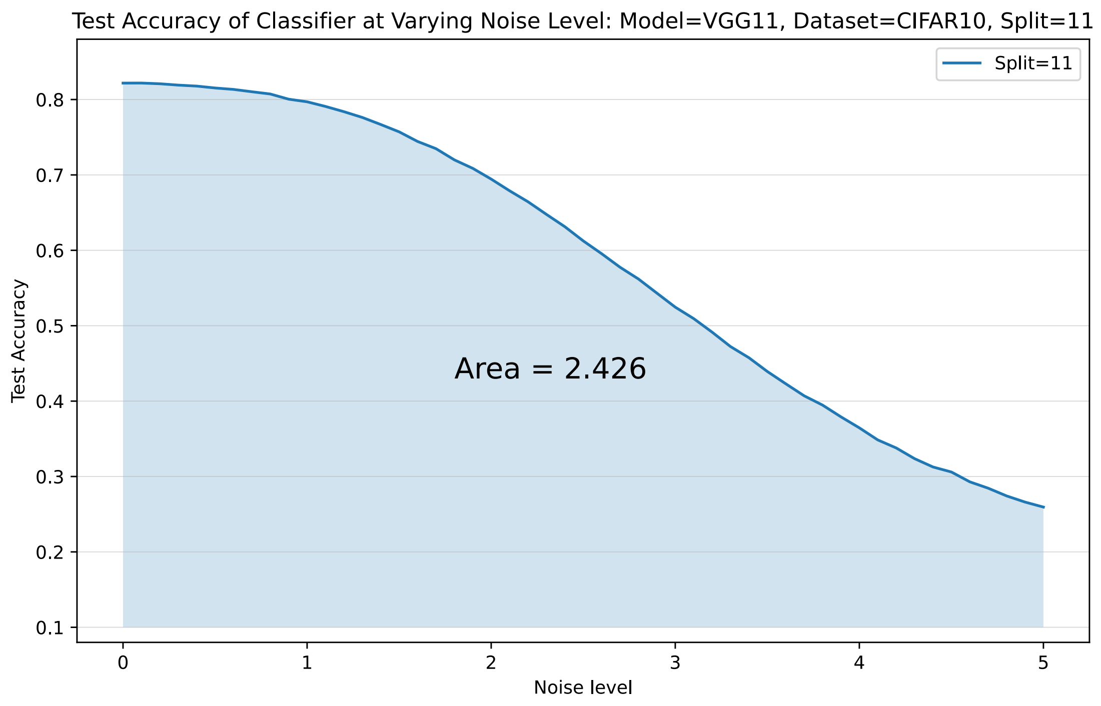


FIGURE 3.3: Accuracy vs. Noise for Model=VGG11, Split=11 on CIFAR-10 with area under graph showing noise tolerance

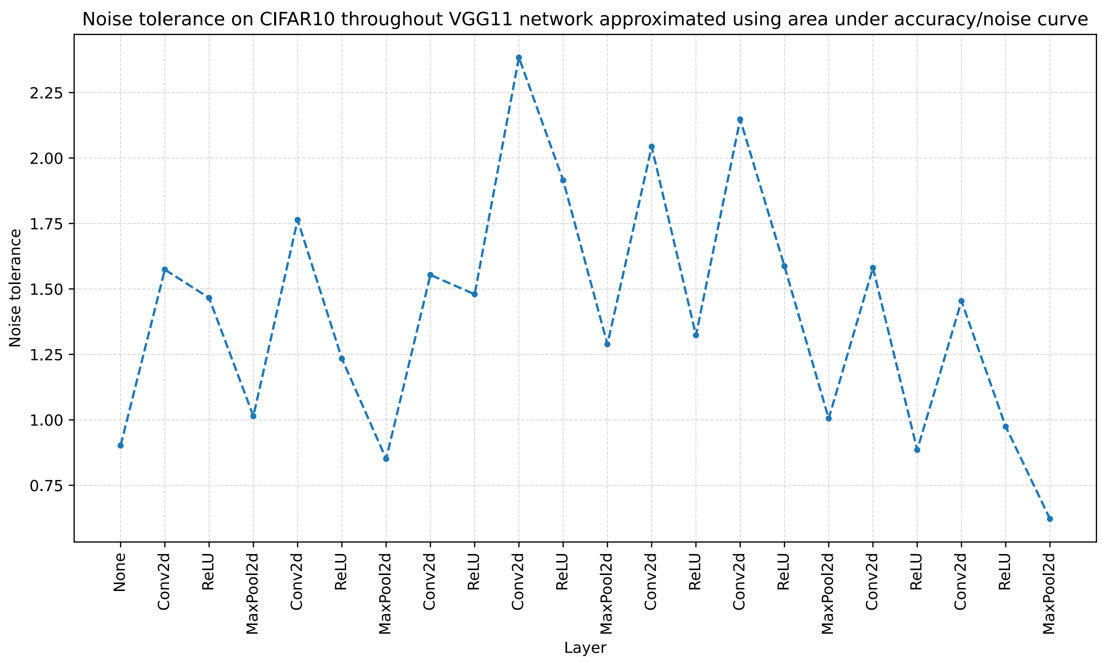


FIGURE 3.4: Noise tolerance through network approximated via Simpson's rule: Model=VGG11, Dataset=CIFAR10

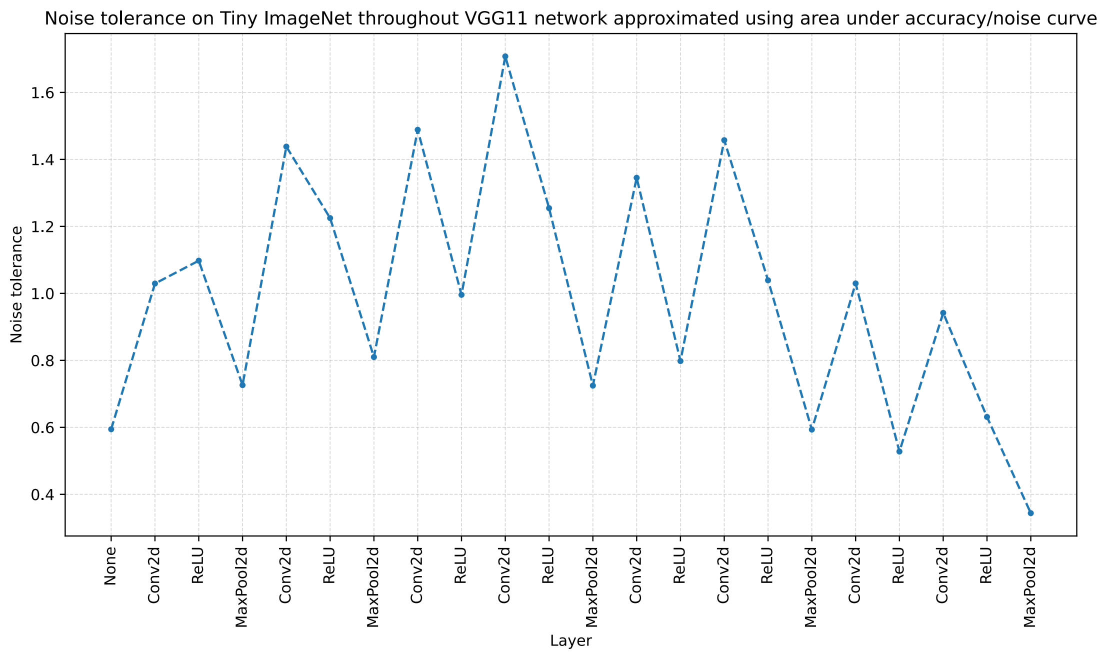


FIGURE 3.5: Noise tolerance through network approximated via Simpson's rule: Model=VGG11, Dataset=Tiny Imagenet

These plots illustrate the erratic behaviour of the noise tolerance, and this behaviour can be seen in all networks, and with both datasets. It is interesting that the noise tolerance appears to increase through the first half of the network, but then decrease through the second half, with the noise tolerance at the output being less than at the input. This appears to apply across all cases, with the noise tolerance seeming to peak at approximately halfway through each model.

If we compare across different models, the overall noise tolerance seems to increase in deeper models, with the peak noise tolerance being greater the deeper the model. This also seems to be consistent in both datasets. For example, the peak noise tolerance for VGG11 on CIFAR10 is approx. 2.38, for VGG13 it is approx. 2.78, for VGG16 approx. 3.03, and for VGG19 approx. 3.08. Similarly on tiny ImageNet, the peak noise tolerances are approx. 1.71, 2.08, 2.24, and 2.31, for VGG11, VGG13, VGG16, and VGG19, respectively.

This is somewhat interesting, as the peak noise tolerance is also achieved at later layers in deeper models, being at layers 9, 13, 15, and 17 in the respective models on both datasets, except for VGG13 on Tiny Imagenet, where the peak is at layer 8, but the tolerance at layer 13 is only fractionally smaller. This indicates that the depth does play a role in improving the tolerance to noise. It is surprising, however, that the noise tolerance drops off after these peaks, seeming to decline steadily to a lower level than at the start of the network.

If greater depth were related to improved noise tolerance, we might expect to see the noise tolerance increasing through the entire network, rather than peaking in the middle. Additionally if noise tolerance were a direct indicator of improved generalisation, we would again expect to see the greatest noise tolerance at the end of the network.

It is possible the peak in the middle is related to the dimensionality of the data in the latent space, as this also increased and decreases throughout the network. We can plot the dimensionality alongside the noise tolerance to examine whether there may be a relationship between them. We plot the dimensionality on a log-scale, as in most cases (except in the first layer, where the 3 colour channels are mapped into 64 feature maps) the dimensionality changes by a power of two. Since images are input at a resolution of 224x224 pixels, the dimensionality at every point (other than the input where there are only 3 colour channels) will be a value equal to $7^2 \times 2^i$ for some $i \in \mathbb{Z}$.

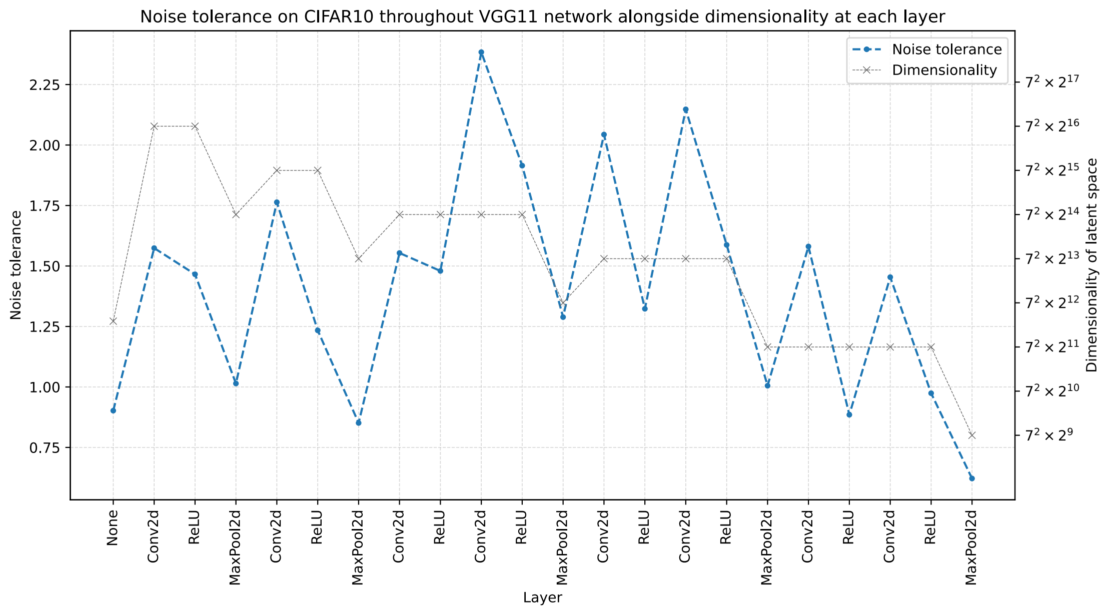


FIGURE 3.6: Noise tolerance & dimensionality through network: Model=VGG11, Dataset=CIFAR10

Figure 3.6 shows a plot of this noise tolerance proxy through the VGG11 network on CIFAR-10 , while figure 3.7 shows the same plot for the VGG11 network on Tiny Imagenet. As before complete results are shown in Appendix A, in figs. A.9 to A.12 for CIFAR-10 and figs. A.13 to A.16 for Tiny Imagenet.

It appears from these results that there may be some correlation between the dimensionality and the noise tolerance, with both generally first increasing and then decreasing through the network. However, there are significant caveats to this observation. Firstly in all cases the dimensionality peaks in the first few layers, and then drops off over the rest of the network, whereas the noise tolerance appears to peak in the middle of the networks, when the dimensionality has been reduced by several orders of magnitude. Secondly, the noise tolerance is erratic, spiking up and down from layer to layer,

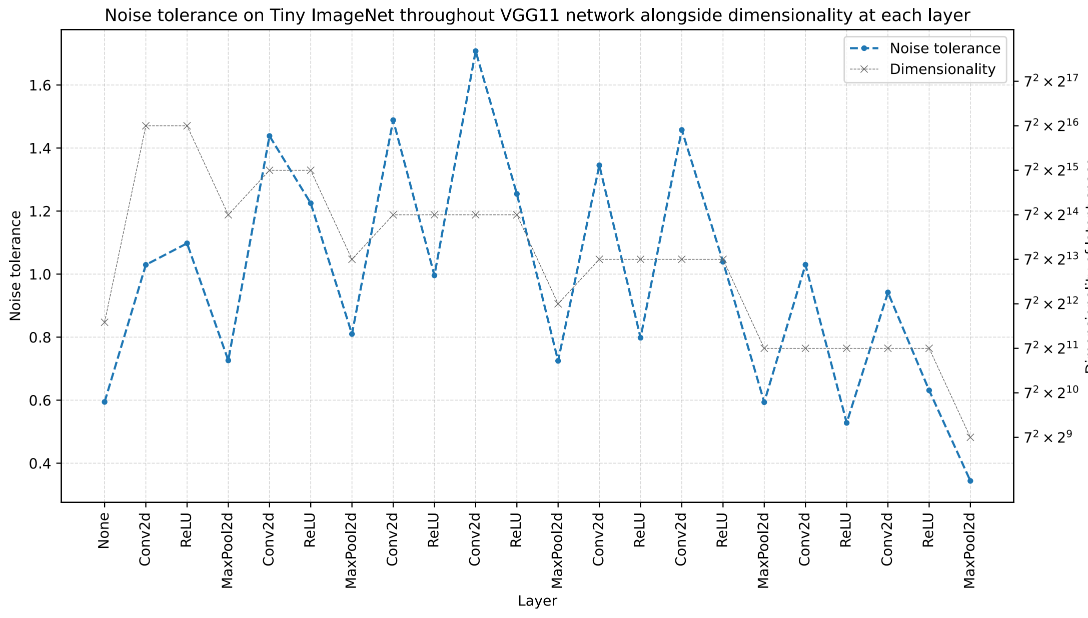


FIGURE 3.7: Noise tolerance & dimensionality through network: Model=VGG11, Dataset=Tiny Imagenet

whereas the dimensionality is constant for several layers, increasing on some convolutions, then decreasing on max pooling layers after one or two more convolutions and ReLU layers. Given these observations, we cannot infer that any apparent correlation implies any causation in relationship, and that it may simply be a coincidence.

We should instead focus on the changes that occur in the noise tolerance from layer to layer. In each case on our plots, the layer name given indicates that the model was split after that layer, i.e. that layer is the last to have been applied to the data before noise is introduced. As mentioned, we see an interesting pattern, where the noise tolerance is erratic and increases and decreases sharply between layers. In particular the noise tolerance seems to increase with each convolutional layer, and decrease with the ReLU and Max Pooling layers. This can be seen more easily if we plot the change in noise tolerance given with each layer, as shown in figures 3.8 and 3.9, with full results in Appendix A in figs. A.17 to A.20 for CIFAR-10 and figs. A.21 to A.24 for Tiny Imagenet. We refer to this change in noise tolerance as Δ.

These results are interesting, and there may be several possible explanations for the patterns we can observe. The convolutional layers increasing noise tolerance and Max Pooling layers decreasing the tolerance might indicate that noise tolerance is correlated with the dimensionality of the data, as we suggested previously, however we have seen that this does not apply in general. This would also not explain the decrease in noise tolerance given by the ReLU layers, as these do not change the dimensionality of the data, only modify the individual features. Both ReLU and Max Pooling layers by their nature throw away information, as pooling reduces the dimensionality, directly

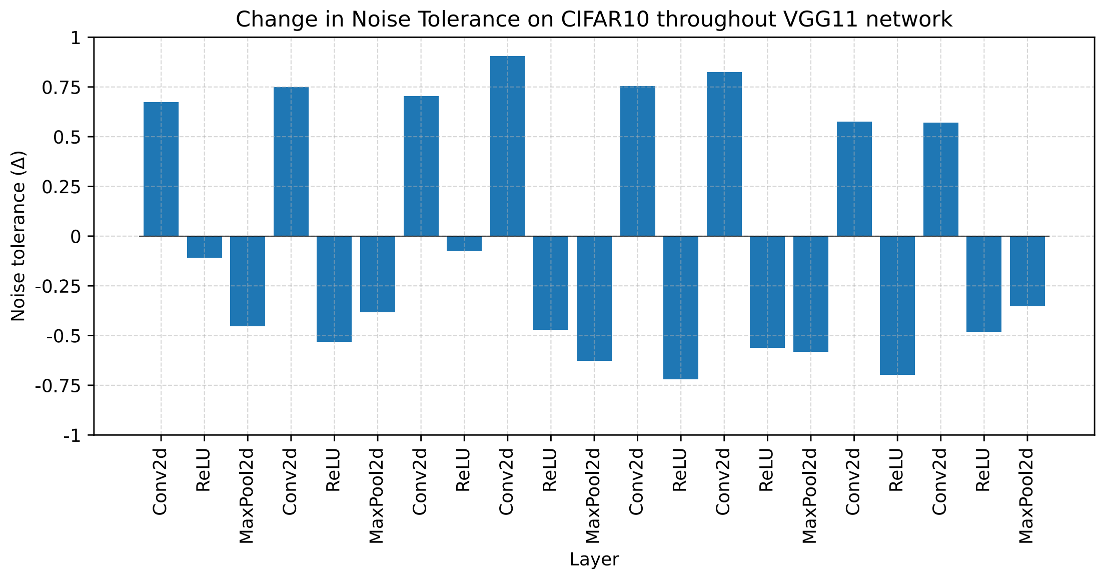


FIGURE 3.8: Change in noise tolerance through network: Model=VGG11, Dataset=CIFAR10

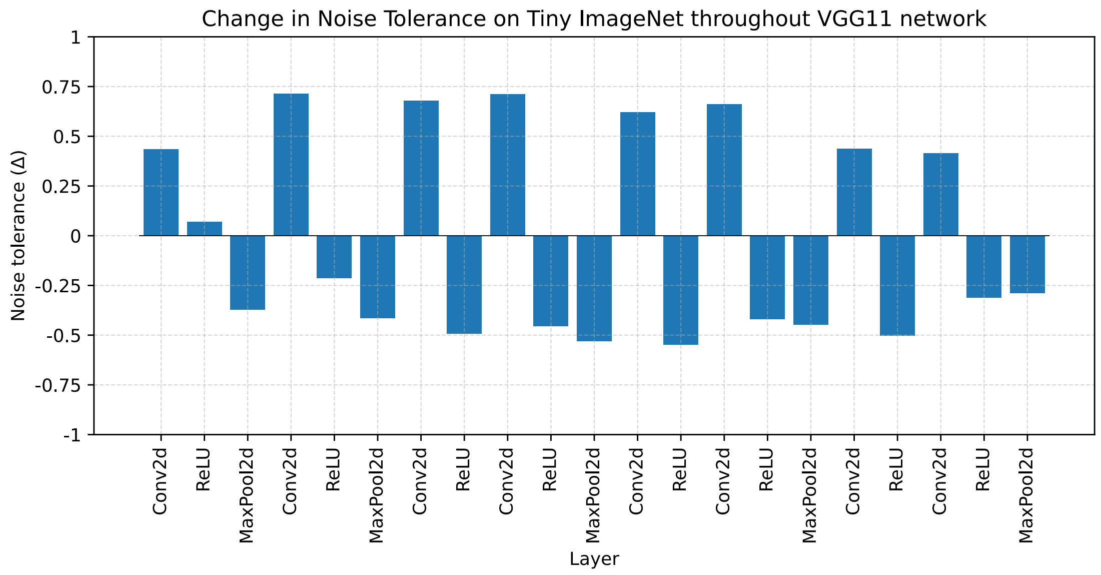


FIGURE 3.9: Change in noise tolerance through network: Model=VGG11, Dataset=Tiny Imagenet

eliminating features, and ReLU functions remove any negative values. This may indicate that reducing the information present in the data reduces the noise tolerance, however no layer can increase the quantity of information in the data, so the quantity of information alone also cannot account for the increase in noise tolerance. It may be that convolutional layers can in some cases separate salient information about the class of the data from spurious information unrelated to the class. This could explain why noise tolerance increases on the convolutional layers, and then decreases on the ReLU and pooling layers, as those eliminate some features from the data. However, not all of the convolutional layers increase the dimensionality of the data, but all the convolutions do seem to increase the noise tolerance.

3.3 Potential Flaws

The results so far seem to indicate that class margins do not increase through the network, despite the hypothesis that this could lead to improved generalisation. However, we observe that there may be a flaw in the methodology behind these experiments. The noise introduced to each data-point is calculated as the sum of the N most significant singular vectors of the data, each scaled by the product of the corresponding singular value, the fixed noise scale, and a random value $\in [0, 1)$ (as described in section 3.1). This has the effect of only perturbing the data in only the directions corresponding to the N most significant singular vectors, that is, the directions in which the data has most variation. This strategy was chosen to avoid perturbing in directions which were not present in the data, to better represent noise that might be observed in the data. However, this has two significant flaws. Firstly, the chosen value of N is significantly smaller than the dimensionality of the data, particularly in early hidden layers where the dimensionality is of the order $\approx 10^6$. Secondly, if the margins between classes increases, this could lead to greater variation in directions along which those margins lie, thereby meaning that we inadvertently increase the scale of noise in these directions, potentially cancelling out any increase in noise tolerance we would otherwise observe due to the larger margin. This could explain the somewhat surprising behaviour observed with convolutional layers, given the linear nature of convolutions. Since noise is generated following the principle components of the data, scaled to their relative size, a linear transformation should not change the distances between classes relative to this re-scaling. However, this could be due to the choice to use only the 50 most significant principle components, it may be that the linear re-scaling done by the convolutions changes which directions these 50 most significant principle components are, and so introducing noise following these new components could increase the noise tolerance. In other words, the re-scaling done by the convolution would make directions less correlated with the class boundaries more prominent in the data. Eliminating these directions would then have the curious effect of decreasing the apparent margin, but in

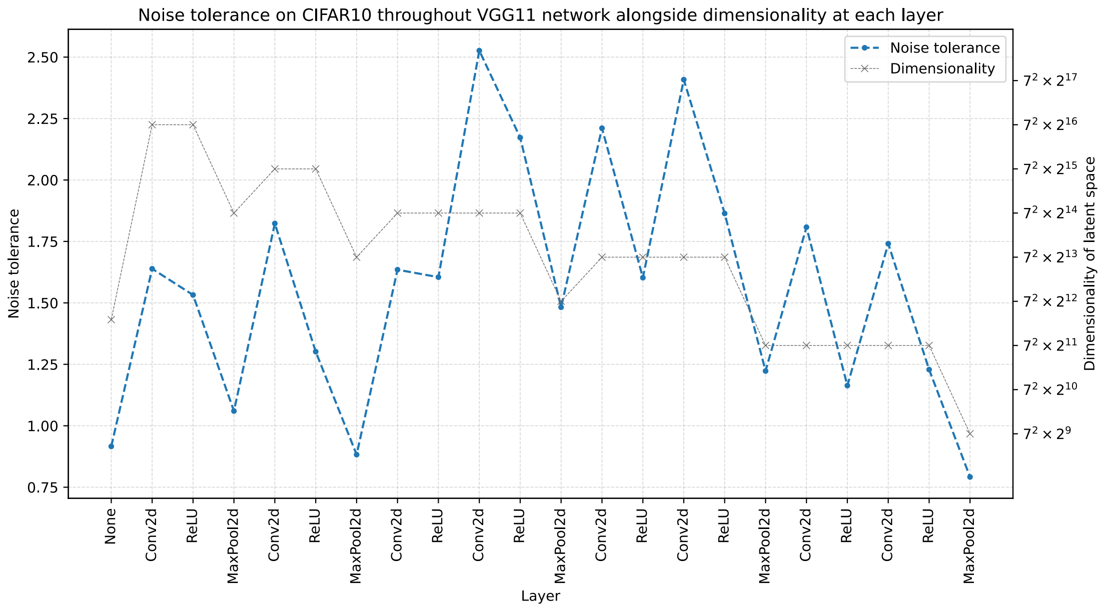


FIGURE 3.10: Noise tolerance through network with noise centered on class means: Model=VGG11, Dataset=CIFAR10

theory not reducing the salient information present in the data. This could explain why, under this framework, ReLU and max pooling layers give decreased noise tolerance.

3.4 Fixing Issues with the Methodology

I attempted to control for the second of these two factors by simply calculating the singular vectors for the noise based on data-points centred about the mean of the corresponding class. This should correct for any variation in the data caused by class margins, while retaining all other variation. I repeated the experiments generating noise in this way. In order to avoid using compute resources unnecessarily, I ran a trial experiment using only the VGG11 model on CIFAR10, to determine the value in repeating the experiment on the other models and datasets.

Figure 3.10 shows the results of generating noise in this way on the VGG11 model and CIFAR10. We can observe that, while the noise tolerances are slightly higher, the trend is essentially the same as in 3.4, where the noise is generated for singular vectors centred about the origin. It seems therefore that this correction does not affect the trends we observe in the results, and so I did not consider it worthwhile spending computing power on replicating these results for the other models and dataset. I therefore considered whether the other potential flaw affects the results, that is, that the chosen value of N is too small. It would be possible to try using a larger value of N, however for early layers of the network, where the dimension of the latent space is largest, computing the SVD quickly becomes intractable as N increases, both in time and memory. I therefore

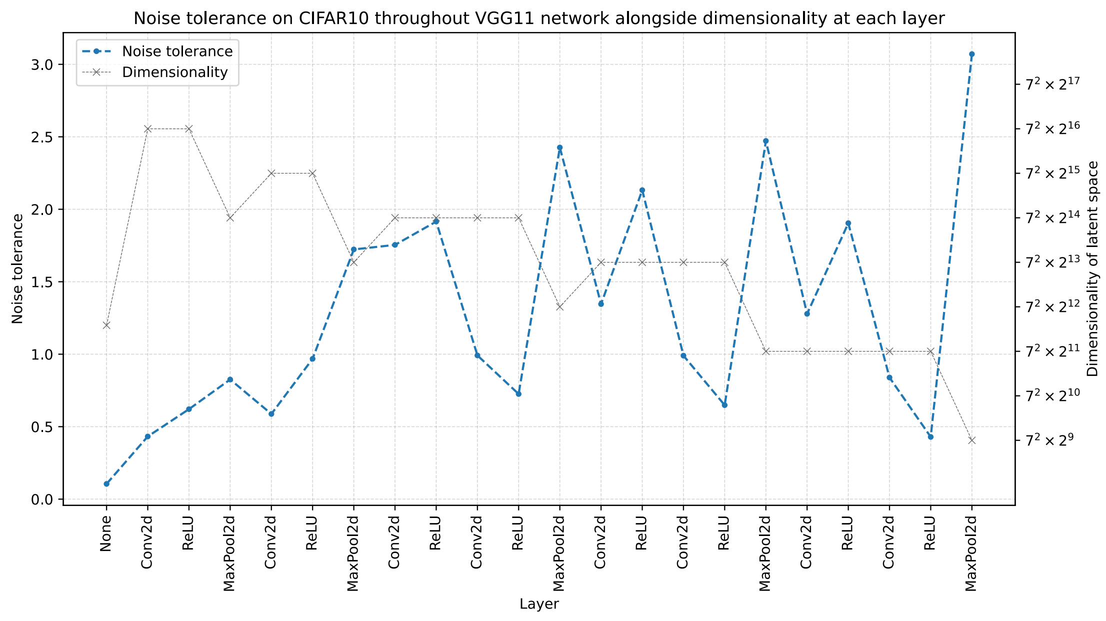


FIGURE 3.11: Noise tolerance through network with Gaussian noise: Model=VGG11, Dataset=CIFAR10

considered introducing noise in a more naive way, by simply adding uniform Gaussian noise to the data. I chose to scale this noise by the average norm of the data, divided by the square root of the number of dimensions, in order to control for both variations in the scale of the data. Dividing by $\sqrt{d}$ is necessary in order to correct for the dimensionality, since $|x|$ grows in proportion to $\sqrt{d}$. Formally, I generated noisy data-points z' as:

$$z' = z + \alpha\sqrt{N}\eta$$

where α is the noise scale we vary, $N = dim(z)$ and η is a vector sampled from $H \sim \mathcal{N}(\mathbf{0}, I_N)$.

I then repeated the experiments again with this revised method of introducing noise. As before, I ran an initial trial experiment using the VGG11 model on CIFAR10.

3.5 Further Results

Figure 3.11 shows the results of the trial experiment of generating uniformly scaled Gaussian noise. The results of the trial experiment are promising, we see a much more marked increase in noise tolerance through the network, with some interesting trends. The noise tolerance in the final layer is now much larger than in the first layer, instead of being lower as in our previous experiments. The noise tolerance in the first layer is close to zero, while the noise tolerance in the last layer is close to the maximum possible

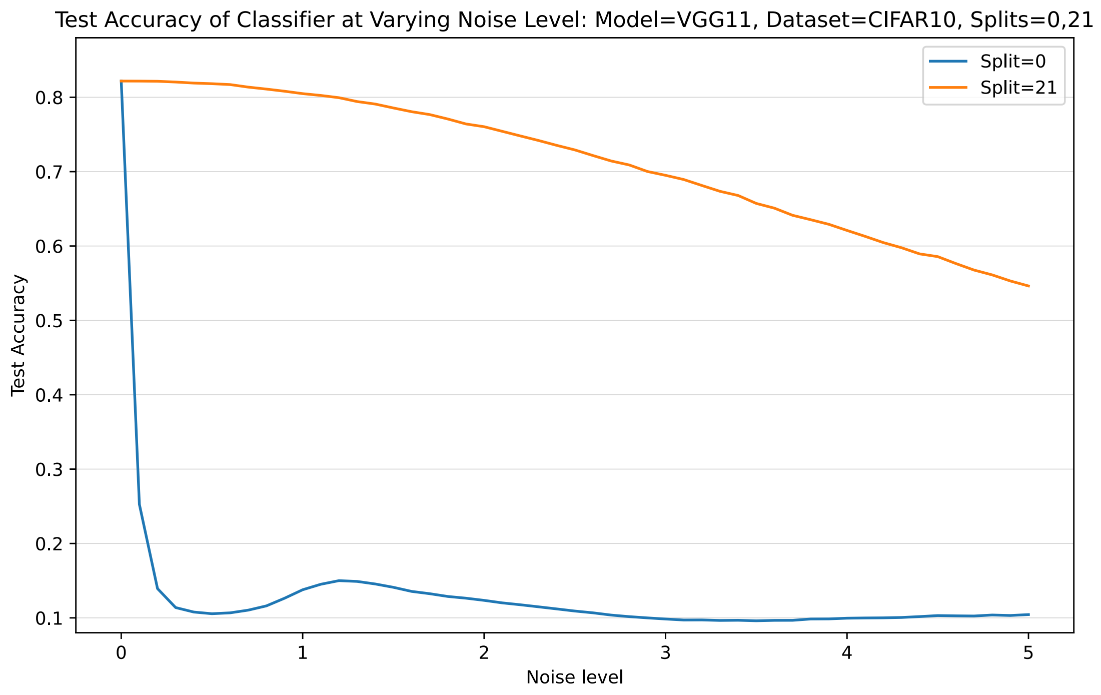


FIGURE 3.12: Accuracy vs Noise plots using Gaussian noise: Model=VGG11, Dataset=CIFAR10

3.6. If we plot accuracy over noise graphs for these layers, we can see how pronounced the difference of these values is.

Figure 3.12 shows the accuracies for these layers. We can see that the accuracy in the first layer drops off very quickly, reaching the minimum value of 0.1 with only a noise scale of around 0.5. Conversely the accuracy in the final layer drops off very slowly, reaching only around 5.5 at a noise scale of 5, and still being close to the accuracy of the uncorrupted model. Interestingly there is a small increase in accuracy in the first layer after the initial drop off, around a noise level of 0.5 - 1.5, with the accuracy then tailing off again. This is very unexpected, as a greater degree of data corruption should only hurt performance. I have no good hypothesis for this, although investigating it may be interesting for future work.

We see other interesting trends in 3.11. While the noise tolerance increases from the first to the last layer, this increase is distinctly not monotonic. The tolerance generally increases in the first few layers, with one small dip, but then begins to vary drastically in the middle and later layers. We can see that many of the largest drops come after convolutional layers, while the largest increases come from max pooling layers, with the ReLU layers sometimes increasing and sometimes decreasing the tolerance, as well as some convolutions seeming to have minimal effect.

The increase in noise tolerance in the max pooling layers is particularly interesting. While these layers decrease the number of dimensions, and so would decrease the norm of the data, this was explicitly corrected for in the method of introducing noise, so the dimensionality itself should not account for this. This suggests there must be some important factor in the dimensions that are eliminated or retained by the pooling layers which would account for the increase in noise tolerance, beyond simply the reduction in number, and that the pooling layers lead to an increase in the margins between classes.

The decrease in noise tolerance from the convolutional layers is also interesting, as this reverses the trends we observed in the earlier results using noise generated from the singular vectors of the latent space. This would suggest that the convolutional layers skew the data significantly in directions orthogonal to the class boundaries, making these directions more significant in the data. The effect that this would have on overall performance of the model is uncertain, but this is also a question worth further exploration.

I considered these results to be promising, and worth investigating further, so I repeated the experiment on the other VGG models, and on the Tiny ImageNet dataset. Full results of these experiments are again shown in appendix A, in figs. A.25 to A.28 for the CIFAR-10 results and figs. A.29 to A.32 for the Tiny Imagenet results. We can see these results match with the trends in the trial experiment, with the noise tolerances increasing between the first and final layers, but with large peaks and troughs in layers in between. We see similar trends in the individual operations, with max pooling layers giving large increases in tolerance, and the convolutional and ReLU layers giving both increases and decreases, with the largest decreases being in convolutional layers. It appears that before each max pooling operation, the preceding convolution and ReLU both yield significant decreases in noise tolerance, with these ReLU layers being the main ReLU layers to give decreases in tolerance, where ReLU layers preceding convolutions tend to give increases in tolerance. This is interesting as it suggests a difference in the way the convolution and ReLU layers operate in conjunction when depending on the layer following them.

3.6 Summary

In this chapter I conducted experiments to attempt to measure the margins between classes through the layers of CNN models in the VGG family, by introducing noise to the data and measuring the accuracy at varying noise levels. Two methods of introducing noise were used: introducing noise in only the most significant principle components of the data, and introducing uniform Gaussian noise. When introducing principle component noise, the noise tolerance appeared to peak around the middle of the networks, and to drop off in later layers, whereas when introducing Gaussian noise the noise tolerance was highest at the end of the networks, with peaks and troughs throughout. Convolutional layers appeared to be more tolerant to principle component noise and less tolerant to uniform Gaussian noise, while ReLU and max pooling layers appeared to be more tolerance to Gaussian noise, and less tolerant to principle component noise. This suggests that convolutions skew the data in directions orthogonal to the class boundaries, but that the class margins may still be small, while ReLU and max pooling layers may eliminate variation in directions orthogonal to the margin, making the size of the margin larger relative to the overall size of the data. This would suggest that ReLU and max pooling layers eliminate information not relevant to classification, and possibly that the convolutional layers transform the data in such a way as to allow for this.

Chapter 4

Using Hidden Layer Probes to Explore the Transformation of Data Through CNN Models

As demonstrated in the previous experiments, CNNs do appear to induce an increase in the margin between classes with successive layers of the network. The increase in the margin may be significant in the improved generalisation exhibited by the CNN, however we cannot simply assume this to be the case. Following the initial hypothesis that a larger margin implies an easier classification problem, and so better generalisation, it should be the case that where the margins in the data are greater, the data can be more easily classified.

I therefore aimed to relate the size of the margin to the complexity of classification on the data. I aimed to quantify the complexity in a way which allows for meaningful measurements to be calculated, to give quantifiable values to relate to the margins. I therefore chose a straightforward method of training a relatively simple classifier on the data, and take the test accuracy of this classifier, with the assumption that the complexity of classifying the data is inversely proportional to the classification score. This assumption naturally follows from the assumption that less complex data should be more easy to classify, and therefore should yield greater classification accuracy than more complex data. This is illustrated by the trivial example from chapter 3, where the least complex data — that is data with one feature, which directly codes the label — can be classified perfectly by any deterministic classifier. In the extreme opposite case, data which is of maximum complexity, i.e. random data, can never yield greater accuracy than random chance. Applying a classifier in this way to the latent representation of the data at a point partway through the flow of the network is akin to the method of hidden layer probes introduced by Alain et al. 2017.

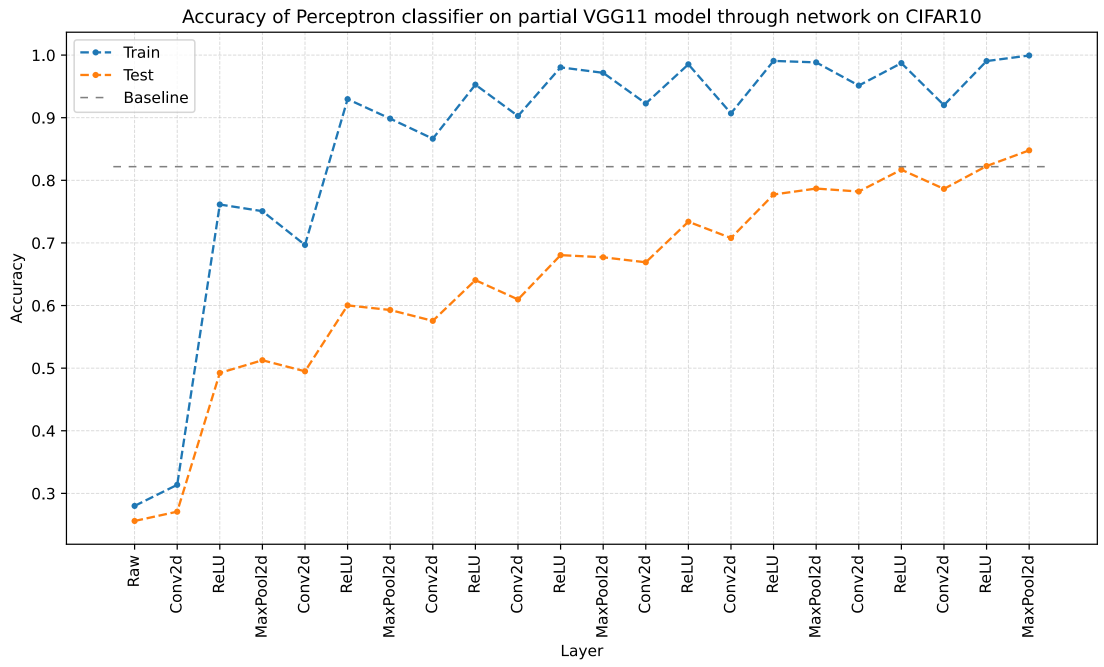


FIGURE 4.1: Test & Train accuracy of perceptron probes throughout network: Model=VGG11, Dataset=CIFAR10

4.1 Methodology

As in the previous experiment, I took a pre-trained convolutional neural network F, composed of n distinct layers, and choose a layer $l \in \{1, \ldots, n\}$ at which to split the network, giving partial networks G_l and H_l. As before $F = H_l \circ G_l$. I then created an untrained classification model, C_l, and using this created a composite model $M_l = C_l \circ G_l$. I trained this model on a dataset D, fixing the weights of G_l so as to only learn the weights of C_l. I then measured the classification accuracy of M_l on the test data, giving a complexity score.

As before, I used models in the VGG family, namely VGG-11, VGG-13, VGG-16 and VGG-19, all pre-trained on ImageNet, and train the composite models on CIFAR-10 and Tiny ImageNet. Following the methods of Alain et al. 2017, I investigated the use of linear classifiers for the classification model C_l. Each CNN is again split after every operation, including activations and pooling, to investigate the significance of these, and provide a direct comparison with our experiments measuring the class margins. In each case I trained the classifiers for a maximum of 100 epochs, but with early stopping used if the validation accuracy - measured on a separate validation set - does not increase for five epochs.

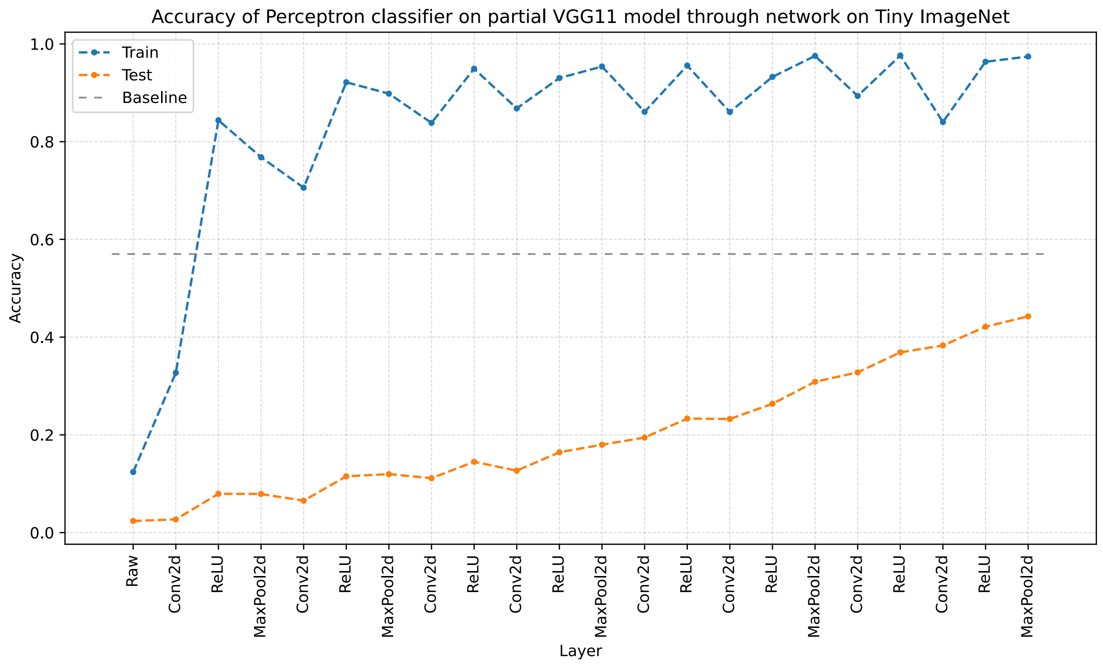


FIGURE 4.2: Test & Train accuracy of perceptron probes throughout network: Model=VGG11, Dataset=Tiny Imagenet

4.2 Results

Figures 4.1 and 4.2 show the results of training perceptron probes on the hidden layers of the VGG11 model on CIFAR-10 and Tiny Imagenet. Full results are shown in appendix B, in figs. B.1 to B.4 for CIFAR-10 and figs. B.5 to B.8 for Tiny Imagenet. In each case we plot a baseline accuracy of the test accuracy we obtained using the whole model, fine-tuned on the particular dataset, to compare the results against. We can see that on both datasets the train accuracy is very good, quickly approaching 100%, even on the 200 class Tiny Imagenet. If we compare with the test data, the accuracy increases through the network much more gradually, but reaching reasonably good accuracy at the end of the model. Indeed on CIFAR-10 , the test accuracy reaches around the same level as the baseline accuracy on each model, indicating that the simple perceptron classifier can be as effective, at least on simpler problems, as the more complex MLP classifier used in the full model. On Tiny Imagenet, the test accuracy is still relatively good - for the problem at least - achieving around 45-50% accuracy in each case, compared with the approx. 55-60% accuracy achieved using the full model. The accuracy is considerably lower than on CIFAR-10 due to the much larger number of classes, however achieving even 50% accuracy on a 200 class problem, with no explicit regularisation or data augmentation techniques - which are frequently used in achieving state of the art performance - is still good performance. The gap between the baseline and the probe accuracy on Tiny Imagenet does indicate that the perceptron classifier is less effective on more complex problems than the full MLP used in the complete model.

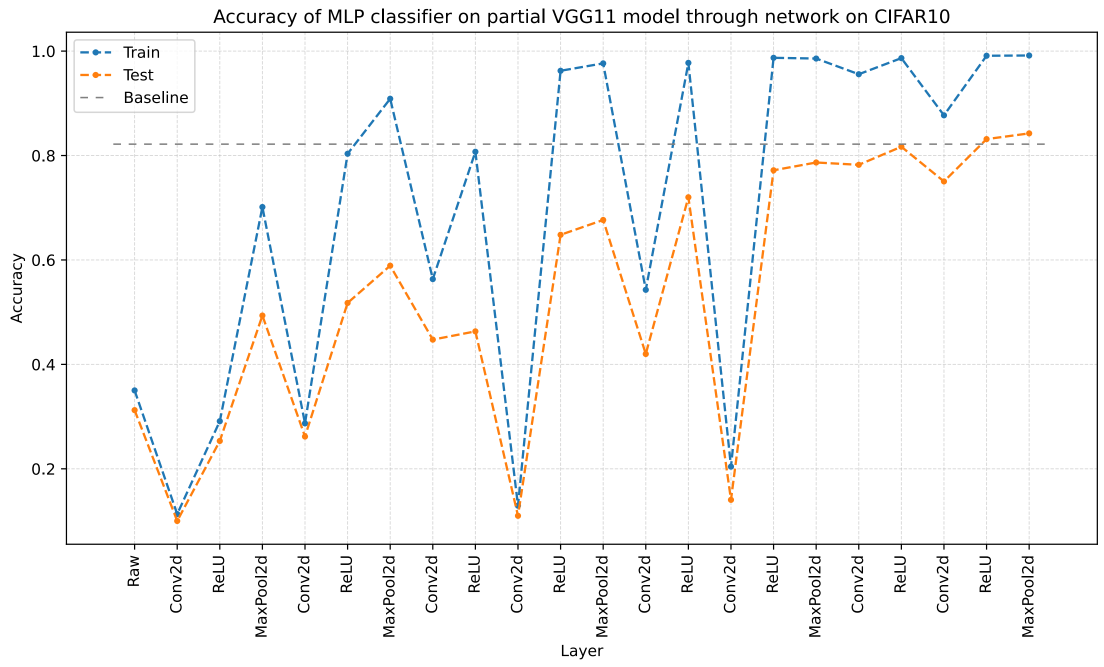


FIGURE 4.3: Test & Train accuracy of MLP probes throughout network: Model=VGG11, Dataset=CIFAR10

The fact that accuracy increases through each network, on both datasets, is an interesting observation. The use of a perceptron probe, which is a linear classifier, indicates that the data becomes progressively more linearly separable through the network. Indeed in the case of CIFAR-10 , the data appears to be as easily classified by a linear classifier as by the MLP used in the full model. This suggests that the linear separability is an important factor in the generalisation performance of the network, and fits with the hypothesis of the network simplifying the problem being integral to yielding good generalisation. Moreover the fact that this linear separability appears to emerge throughout the network indicates that this simplification of the problem is a gradual effect through the network, being the cumulative result of many layers, rather than only a few specific layers contributing to this effect.

While I hypothesise that information is filtered gradually by each layer, with salient information being slowly linearised, and spurious information discarded, it may be the case that salient information is distilled into features more quickly, but in a feature space that cannot be linearly separated. In order to draw accurate conclusions about the nature of this information filtering, it is important to investigate this possibility. I therefore repeated these experiments, but in place of the perceptron classifier used a two-layer MLP, which should be capable of capturing more complex patterns within the feature space than a perceptron, and therefore give an indication of whether salient information is distilled into non linearly separable features.

Figures 4.3 and 4.4 show the results of training the two-layer MLP classifier probes on

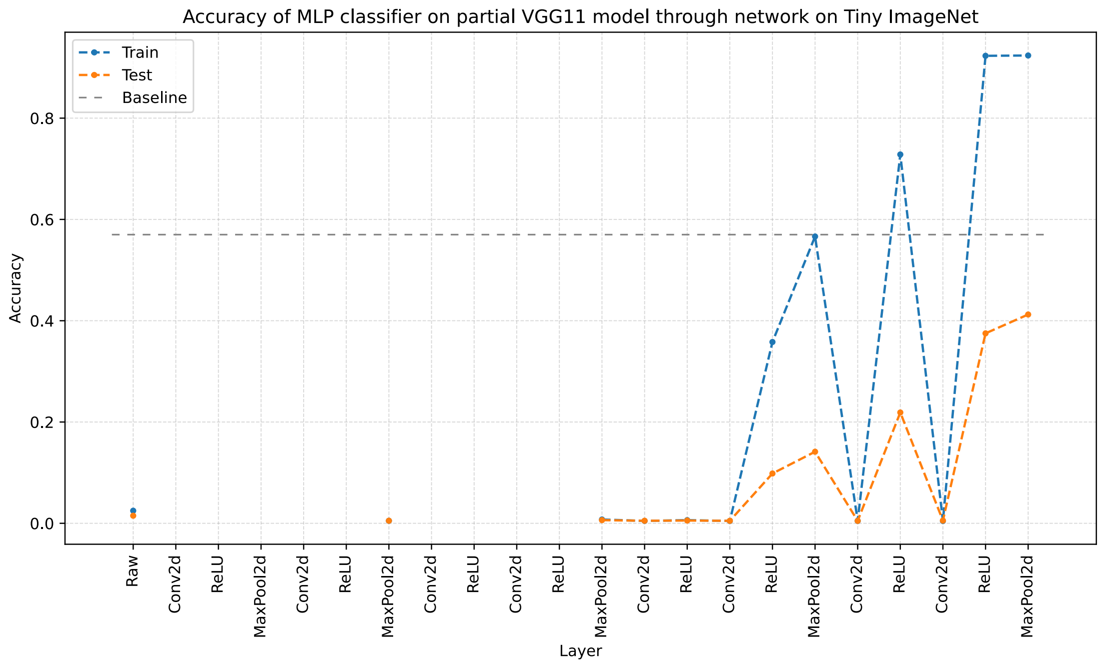


FIGURE 4.4: Test & Train accuracy of MLP probes throughout network: Model=VGG11, Dataset=Tiny Imagenet

the hidden layers of VGG11, on CIFAR-10 and Tiny Imagenet. Full results for all models are shown in appendix B, in figs. B.9 to B.12 for CIFAR-10 and figs. B.13 to B.16 for Tiny Imagenet. I encountered significant difficulty in training these MLP classifiers, particularly on Tiny Imagenet. For early layers of the networks, the number of parameters required to train an MLP is extremely large, and is unfortunately beyond the capabilities of the hardware available. Where training was not possible, no values are plotted. Even in later layers where the training is tractable, I still had significant difficulties, with the models for certain layers seemingly not learning, and being stuck at essentially random accuracy, despite being able to run the training loop. In these cases, the values are plotted as untrainable layers. It is of some interest that this only appears to occur at certain layers, however there is no clear reason for this, as it does not appear to simply be due to dimensionality, since some layers of equal dimensionality do learn as normal.

From the results I could produce, we do see some interesting results. Both the training and test accuracies do still seem to increase through the network, however both accuracies are less consistent, with more variation in both test and train accuracies throughout the networks. This is difficult to be certain of, however, due to the difficulties encountered in training.

We see that the performance in many cases does not significantly increase when using a multi-layer perceptron in place of a linear classifier. Indeed by comparing these results directly, we can see how closely these results match. Figures 4.5 and 4.6 show

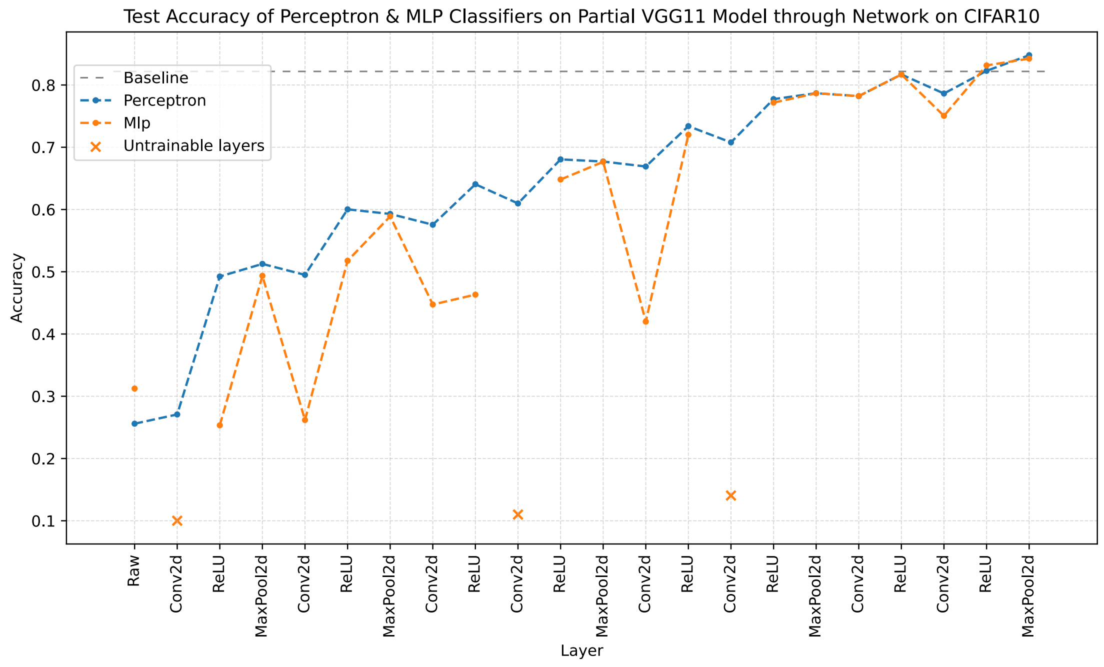


FIGURE 4.5: Test accuracy of Perceptron & MLP probes throughout network: Model=VGG11, Dataset=CIFAR10

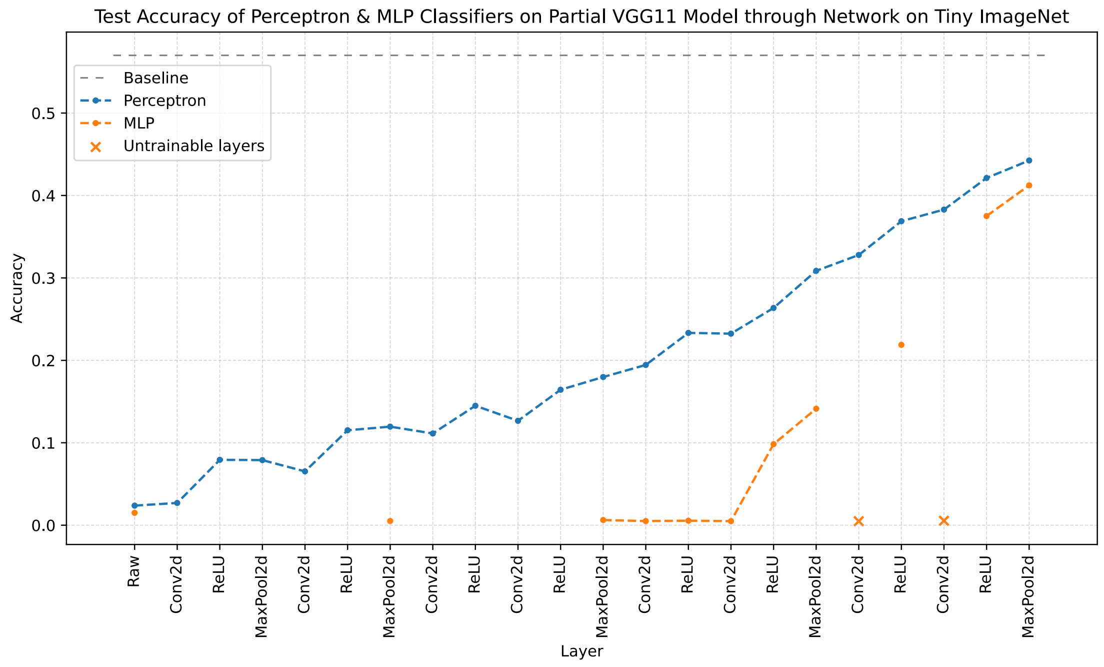


FIGURE 4.6: Test accuracy of Perceptron & MLP probes throughout network: Model=VGG11, Dataset=Tiny Imagenet

comparisons of the perceptron and MLP test accuracies on CIFAR-10 and Tiny Imagenet respectively, through the VGG11 network. Layers where the classifier appears to not be trainable are marked, while layers where no training was possible are omitted. As before, full results are shown in appendix B, in figs. B.17 to B.20 for CIFAR-10 and figs. B.21 to B.24 for Tiny Imagenet. In all cases, the accuracy does not appear to be much improved, if at all, using the MLP classifier, where the classifier was able to learn. This suggests that where the network filters salient information, it does so in a way that is at best linearly separable, and these features are not encoding information in a more complex way. This supports the hypothesis that the generalisation ability using the CNN, and so reduction in complexity of the data, comes from the data becoming increasingly linearly separable through the network. This is a striking result, as there is no inherent reason for the data to become more linearly separable until prior to the very last layer, at which point the remaining layer of the network is a linear classifier.

This suggests that the network naturally induces linear separability in the data as the data flows through the network, despite even the partial networks being capable of learning extremely expressive functions on data which is not linearly separable. This shows a far more significant level of structure emerging in the data as it passes through the network than simply an increase in the margin, as it implies that this margin approaches a linear hyperplane, despite the extremely high dimensionality of the latent space.

This motivates the question of how this linear separability emerges, and whether particular parts of the network, or particular operations within the network, may be driving this. From the previous results we can investigate the role of the different operations. We can highlight the change in linear separability after each type of operation, that being convolutions, activations (ReLU), and pooling, to see how these each affect the separability. Due to the issues encountered in training the MLP models, we focus primarily on the results using linear classifiers, as these are complete and of more interest.

Figures 4.7 and 4.8 show the change in train and test accuracy using perceptron classifiers in VGG11, on CIFAR-10 and Tiny Imagenet. Once again full results are shown in appendix B, in figs. B.25 to B.28 for CIFAR-10 and figs. B.29 to B.32 for Tiny Imagenet. We can observe that there is a very clear trend in which, with only a few exceptions, the convolutional layers lead to decreases in both the training and test accuracy, while the ReLU layers lead to increases in both accuracies. This is rather interesting when we consider how these layers transform the data. Convolutional layers are linear, and so we would not expect these to have much effect on linear classification. However, it should be noted that a convolution is not necessarily invertible, and that it may reduce the dimensionality of the data, either explicitly by having a smaller dimensional output than input, or implicitly by projecting into a linear subspace of the co-domain. It is therefore

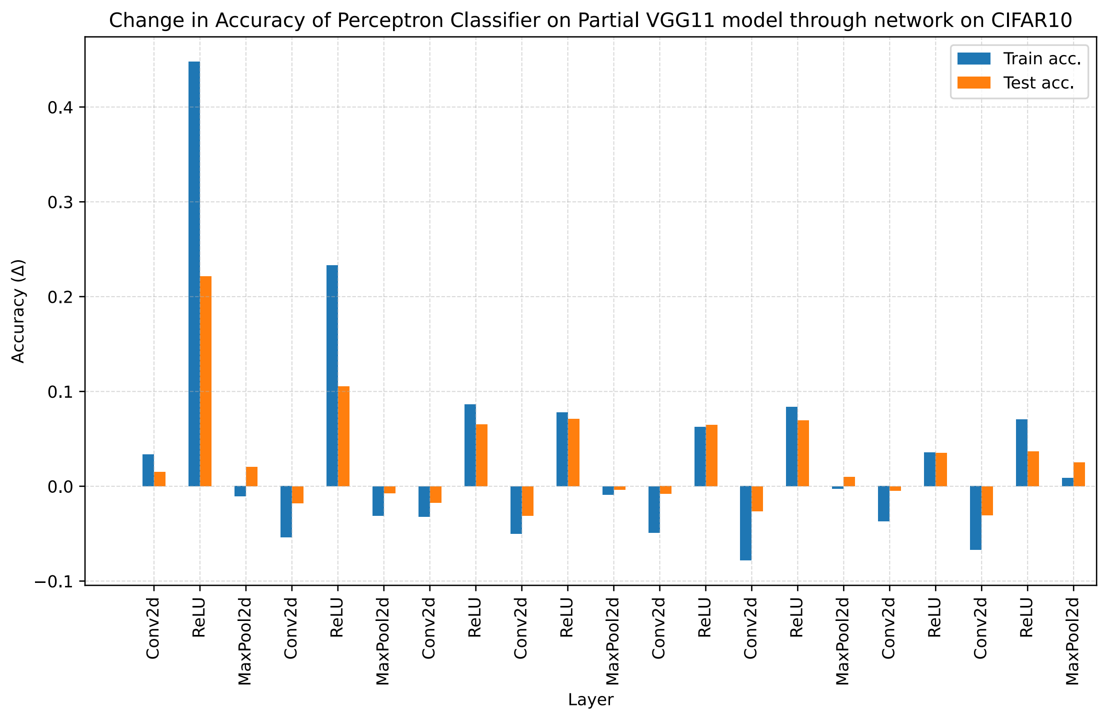


FIGURE 4.7: Change in train and test accuracy of Perceptron probes throughout network: Model=VGG11, Dataset=CIFAR10

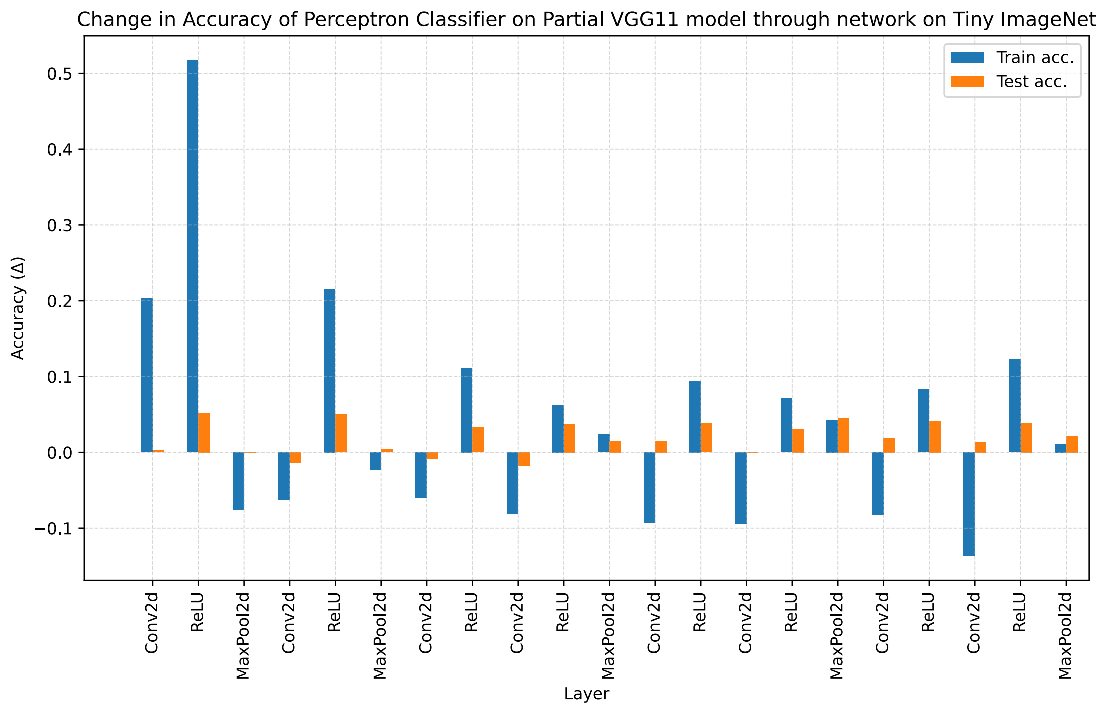


FIGURE 4.8: Change in train and test accuracy of Perceptron probes throughout network: Model=VGG11, Dataset=Tiny Imagenet

somewhat expected that convolutional layers would not increase the training performance, as this should not be possible, but that they might decrease both training and test performance, if reducing the dimensionality of the data, and thereby discarding information. While we do see some examples of convolutions increasing the training accuracy, this may simply be an artifact of the randomness of training the classifiers.

ReLU layers however are non-linear, and so it is possible for these to increase the performance of the classifier. However, ReLU layers do discard a significant amount of information, clamping any negative values to zero, and so it is interesting that these do increase the accuracy in almost all cases, both in training and test. This indicates that the information discarded by the ReLU layers is not salient to the classification problem, but the information retained is. Again we see evidence of the convolutional and non-linear layers working in tandem, since the ReLU layers in many cases give improvements in accuracy over the level lost in the preceding convolution, indicating that as a pair they lead to an increase in accuracy.

The max pooling layers are more curious, in most cases giving only minimal changes in test and train accuracy, sometimes increasing and sometimes decreasing the accuracy. Moreover there are some cases where the max pooling layers give a decrease in train accuracy but an increase in test accuracy, although both changes are often small. Whether this pattern is indicative of a facet of the model is uncertain, however if it is it would indicate that, at least in these instances, the max pooling layers are able to filter some spurious information, while retaining and amplifying salient information. It does at least appear to be the case that the max pooling layers do not remove salient information, as they do not seem to decrease test accuracy, even if they do not directly make the features more linearly separable. This is still an interesting observation, given that the max pooling layers discard (in this instance) $\frac{3}{4}$ of the information by cutting down the number of features by a factor of four. The fact that this appears to have little to no impact on the test accuracy, and so the content of salient information, is certainly interesting, and rather surprising.

4.3 Summary

In this chapter I conducted experiments using hidden layer probes to investigate how the data and complexity of the problem changes through CNN models. I experimented with both linear and non-linear MLP classifiers, and while only the linear classifiers were able to yield complete results, there is a strong indication that linear probes are close to or as effective as the non-linear probes. Moreover the linear probes also appear to be close to as effective as the complex MLP classifier used in the full model, indicating a strong degree of linear separability in the data, close to the total level of information distilled by the CNN. The linear separability also appears to emerge through the networks incrementally, with significant increases to linear separability being given by ReLU layers, countering decreases from convolutions. This gives an indication of the convolutional and ReLU layers working in tandem to linearise the data and distil salient information.

Chapter 5

Investigating the Transformation of Pixel-Level Information into Coarse Features in CNNs

We can see from the results of the investigations in chapters 3 and 4 that the different operations within the networks have differing impacts on the performance and the learning of the network. In analysing these results, I have proposed the hypothesis that these effects are related to the way in which these operations transform, discard or retain information. Indeed, as previously discussed, the overall goal of any machine learning model is to extract salient information from data, in a way which is pertinent to the problem at hand. This necessarily involves filtering and discarding spurious information, at least in order to achieve good generalisation. Focusing on our specific line of enquiry, the idea of discarding information is particularly relevant to the ReLU and MaxPooling operations, which each discard a significant amount of information, while the convolutional layers primarily transform information. While convolutions can discard information, we can assume that generally the amount of information discarded by these is far less than by the directly destructive operations of the ReLU and max pooling.

ReLU operations seem to reliably improve performance, increasing noise tolerance when weighting directions equally, and by assumption the class margins, as well as directly improving both training and test performance in the partial models. Max pooling layers however are more varied. When investigating the margins, the max pooling layers appear to give decreases in margins in a few principle directions, however appear to have the opposite effect when weighting directions equally, giving large increases in the margin. In the partial models, the max pooling layers are particularly intriguing. The effect of pooling layers appeared to be very small, generally giving by far the smallest changes in train and test accuracy. Notably however, in some cases the pooling layers seemed to give small increases in test accuracy, but at the expense of small decreases in train accuracy. This is in contrast to the ReLU layers, which generally increased both test and train accuracy, and the convolutional layers, which generally decreased both test and train accuracy.

As discussed, this is still rather striking, as it suggests that the max pooling layers discard large amounts of spurious information — by virtue of reducing the number of features by a factor of four — in a way that does not appear to affect salient information. This is an effect directly necessary for generalisation, indeed a model which generalises perfectly would discard all spurious information, and retain all salient information, as perfectly salient information fundamentally would be equivalent to the target or class label.

If we consider what function pooling layers perform, this may give us some insight into the nature of this behaviour. A max pooling operation takes the maximum channel value within a fixed window, and discards the other values in that window, and thereby reduces the size of the feature space. If we think of our latent representation as a series of feature maps, where each feature map is a two dimensional grid, each value in a feature map is a measure of how strong the presence of a particular feature is at that position. Deep in the network these features will be complex non-linear representations of a large area of the image around that point, composed of the result of multiple convolutions, activations, and possibly pooling operations applied to that region of the image. Taking a max pool across these features is therefore equivalent to capturing the strongest of these representations in a region of the image, and so will discard some fine-grained positional information of the location of that feature, but will retain the broader information on the degree to which a feature is present within the image.

If we think of this in a specific context, for example classifying an image of a cat, early layers in the network may give latent representations with strong features for things such as eyes, ears, and whiskers, while later layers may give strong features for the whole of a cat's face or head. The pixel-specific locations of these features is not so important for classifying the image, or for constructing the larger representations deeper in the network — as if you were the edit an image of a cat to move it's ears one pixel apart, it would not change the fact it was an image of a cat — however the presence of these features in roughly the correct position is important.

It is possible that max pooling operations naturally yield this sort of behaviour in a network, by discarding fine-grained positional information, while retaining coarse positional and feature information. This would likely contribute to the generalisation capability of the network, due to this innate property of filtering out spurious information, while retaining salient information. If this is the case, and the transformation of fine-grained positional information into coarse information occurs through the network, it may explain some aspect of the generalisation capabilities of convolutional neural networks.

This idea motivates investigating whether this effect occurs in the networks I have studied previously. My hypothesis is for it to occur with max pooling layers, however it may occur elsewhere as well, or may not be observable at any point. It is worth noting that a combination of a convolution and a ReLU can yield an effect similar to that of a max pooling operation, as a ReLU layer will eliminate any negative values, and so a convolution which maps points in a window to be all negative, except for one point which is mapped to a positive, applying the ReLU would leave only this positive value within that window. This is however dependent on the specific features, and the weights of the convolution, but illustrates that this combination of operations can to some extent filter fine-grained positional information, while retaining more coarse feature information.

5.1 Methodology

My hypothesis is that through the layers of the network, the information encoded by the features of the latent space is transformed from the pixel-level information in the input image, where the position of each individual pixel is significant, to complex features of representations of more complex elements of the images, where the position of these features is less significant than their presence or absence. I refer to these as coarse features, in contrast to the fine-grained features that are the individual pixels of the input. These may be representations of specific elements, such as wheels or faces, however they may be more abstract. The specific content of these features is not under investigation.

In order to investigate this hypothesis, I modified the previous experiment into partial models, however in this case instead of training a classifier on the entire latent representation of our images, I applied a global pooling operation to these latent representations, to reduce them down to one value per feature map.

This global pooling operation is a simple sum of the features in each feature map of the latent representation. Specifically, for a $d \times d \times n$-dimensional latent representation, composed of n distinct $d \times d$ feature maps, the global pooling operation outputs an n-dimensional feature vector where each element z_i is the sum of all elements in the ith feature map. This pooling operation eliminates all positional information at that point in the network, leaving only the coarse feature information.

I ran this experiment as before, using models in the VGG family, evaluated on the CIFAR-10 and Tiny ImageNet datasets. The global pooling greatly reduces the number of features in the latent space, making training the classifiers much more viable, so it

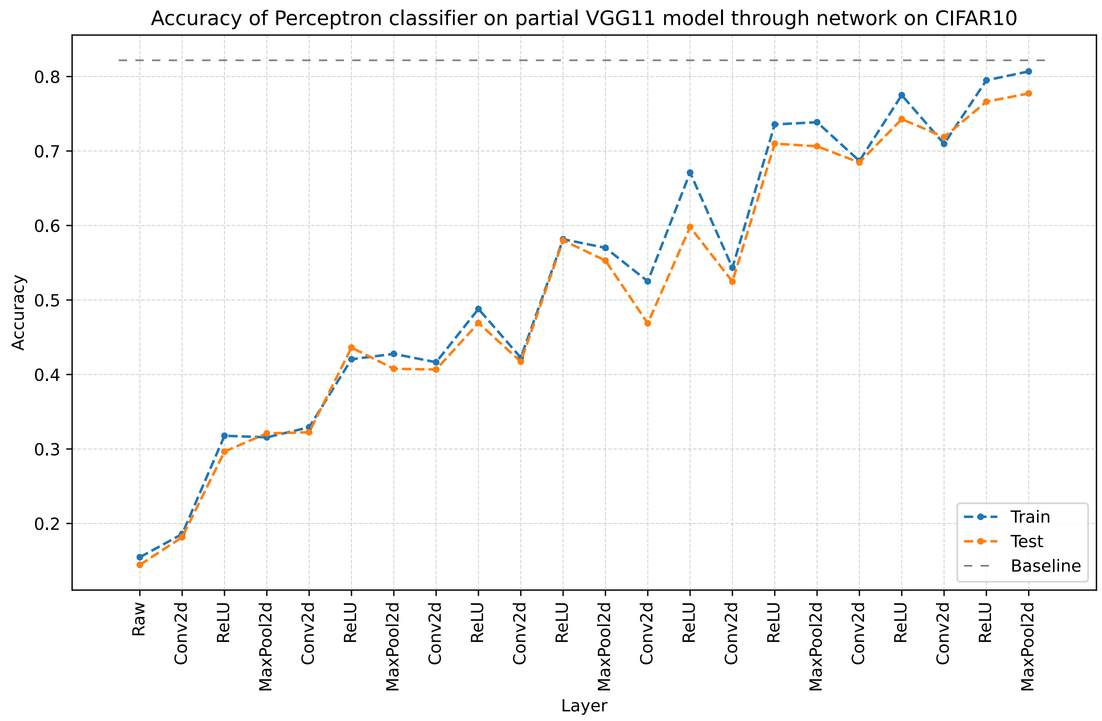


FIGURE 5.1: Test & Train accuracy of perceptron probes throughout network with global pooling: Model=VGG11, Dataset=CIFAR10

should be possible to produce good results using both a perceptron and an MLP. As before, I first investigated using a perceptron classifier.

5.2 Results of Global Pooling with Linear Classifier

Figures 5.1 and 5.2 show the results of training perceptron probes on globally pooled features through the VGG11 network, on CIFAR-10 and Tiny Imagenet respectively. Full results are shown in appendix C, in figures figs. C.1 to C.4 for CIFAR-10 and figs. C.5 to C.8 for Tiny Imagenet. The most notable feature of these results is how good the classification performance is. On both datasets, the accuracies, particularly on the test data, are close to those achieved in the previous experiment, training a classifier on the entire latent representation. Indeed the test accuracies are still relatively close to the baseline accuracies using the entire models. This is an extraordinary result, as the use of global pooling throws away the vast majority of the features, and we assume the vast majority of the information they contain. Moreover the train accuracy - which in our previous experiment generally exceeded the test accuracy by a significant amount, often approaching 100% - now much more closely matches the test accuracy, in some cases even almost on par with it. This suggests that the information retained by global pooling is extremely salient, and is strongly correlated with the underlying classification problem.

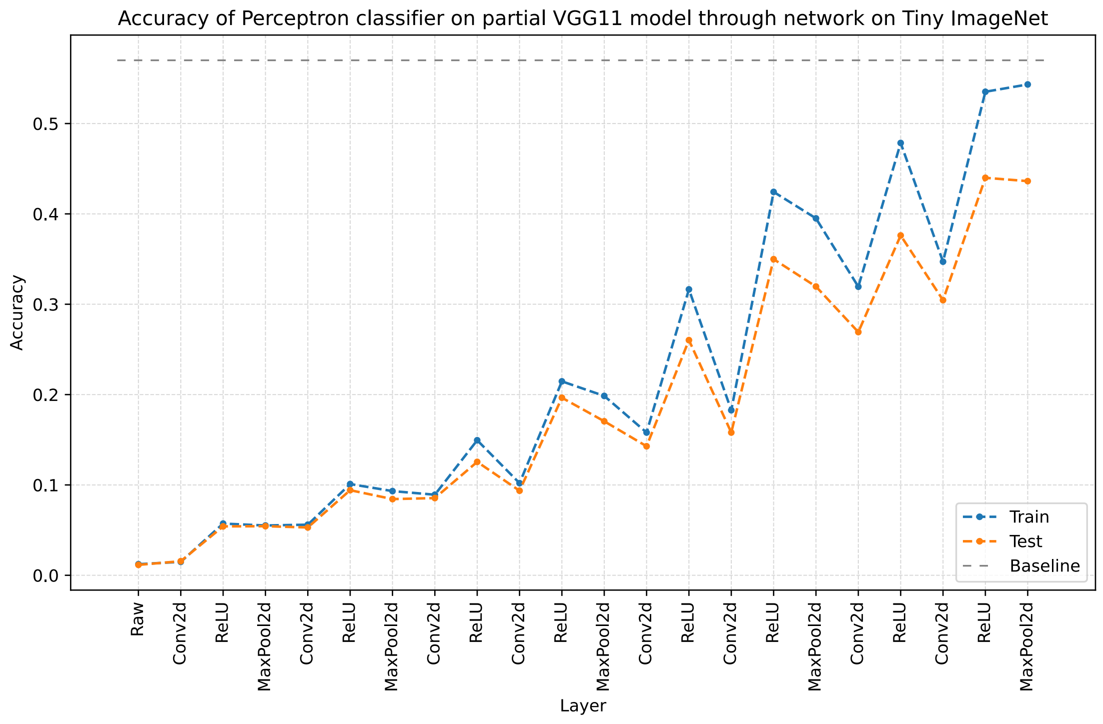


FIGURE 5.2: Test & Train accuracy of perceptron probes throughout network with global pooling: Model=VGG11, Dataset=Tiny Imagenet

5.3 Comparison with Results from Chapter 4

Moreover, the test accuracy when using global pooling seems close in many cases to that achieved when training classifiers on the entire latent representation. By comparing directly with the results of these experiments, we can see this more clearly. Figures 5.3 and 5.4 show these results side by side for the VGG11 model, on CIFAR-10 and Tiny Imagenet respectively. Full results are shown in appendix C, in figs. C.9 to C.12 for CIFAR-10 , and figs. C.13 to C.16 for Tiny Imagenet.

By comparing these sets of results, we can see just how impressive the performance on the pooled latent representation is. In all cases the test accuracy is close to, or even on par with, that achieved when training on the full latent representation. In the case of CIFAR-10 , there is a noticeable gap, of around 5-10% accuracy at the end of the model. Interestingly there is a larger gap in earlier layers, and this gap shrinks through the model. On Tiny Imagenet however, there is almost no difference between the test accuracy with and without pooling, at all stages of the model. The main difference we can observe is that on the pooled features the increase in performance is not as smooth, spiking up and down from layer to layer, but still increasing generally through the model, in line with the unpooled results. This further suggests that the information retained through global pooling is extremely salient, being seemingly as good for classification as the full latent representation.

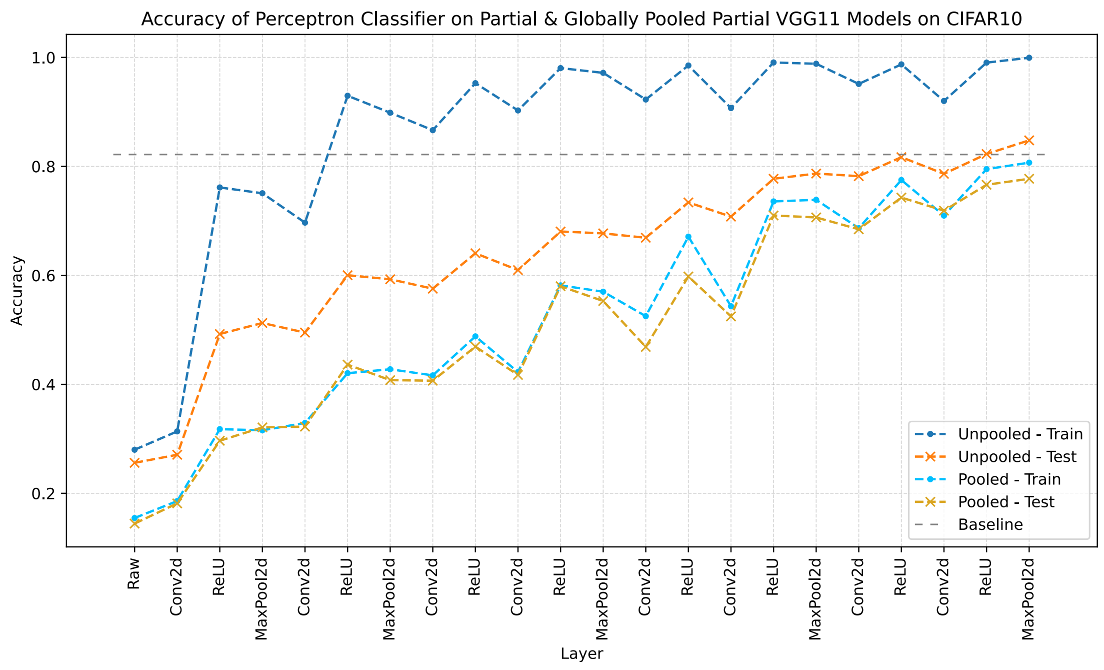


FIGURE 5.3: Test & Train accuracy of perceptron probes throughout network with and without global pooling: Model=VGG11, Dataset=CIFAR10

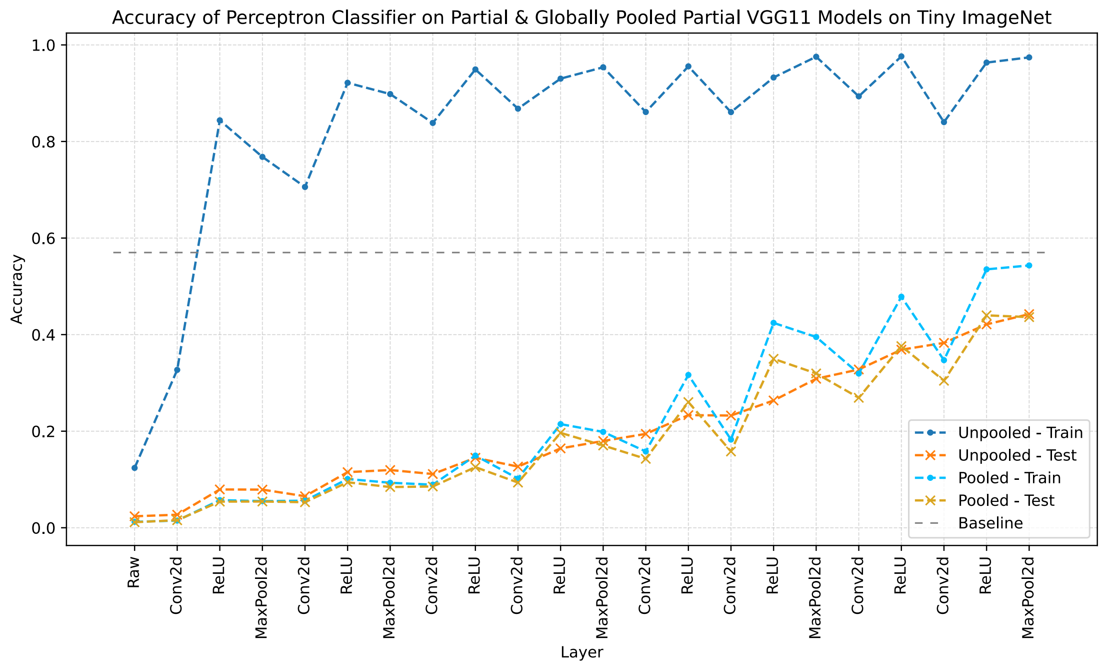


FIGURE 5.4: Test & Train accuracy of perceptron probes throughout network with and without global pooling: Model=VGG11, Dataset=Tiny Imagenet

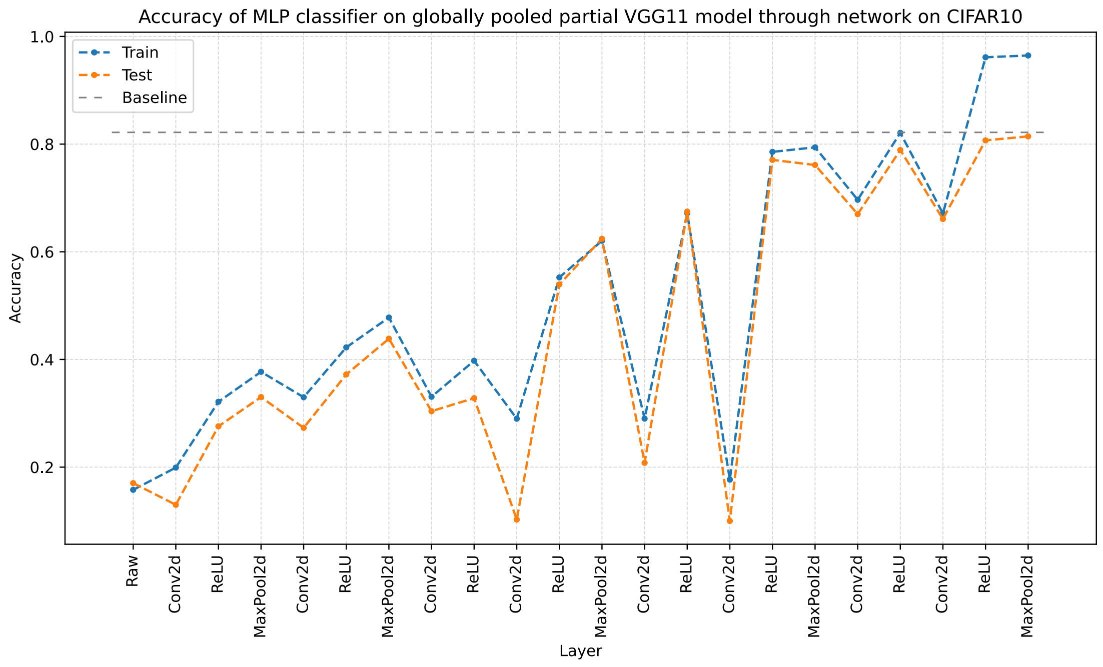


FIGURE 5.5: Test & Train accuracy of MLP probes throughout network with global pooling: Model=VGG11, Dataset=CIFAR10

We can see also that the layer-to-layer variations in accuracy on the pooled features match with those on the full features, with convolutional layers generally giving decreases, ReLU layers giving increases, and Max Pooling layers having only small effects one way or another.

When we consider the numbers of features in question, the significance of the similarities in performance becomes even more apparent. In the final layer, the full model has a latent representation composed of over 25,000 features (25,088 to be exact), while the pooled representation is only 512 features. The full MLP classifiers used in these model have around 120 million weights (119,578,624 for CIFAR-10 , 120,356,864 for Tiny Imagenet), whereas a linear classifier on the globally pooled feature space has around 250,000 for CIFAR10, and around 5 million for Tiny Imagenet. This further supports the hypothesis that the transformation performed by the CNN is far more crucial to generalisation than the over-parameterisation of the classifier.

5.4 Results of Global Pooling with MLP Classifier

Following on from these results, I repeated the same experiments, this time using an MLP classifier, as in chapter 4. Figures 5.5 and 5.6 show these results on VGG11. Full results are shown in appendix C, in figs. C.17 to C.20 for CIFAR-10 and figs. C.21 to C.24 for Tiny Imagenet.

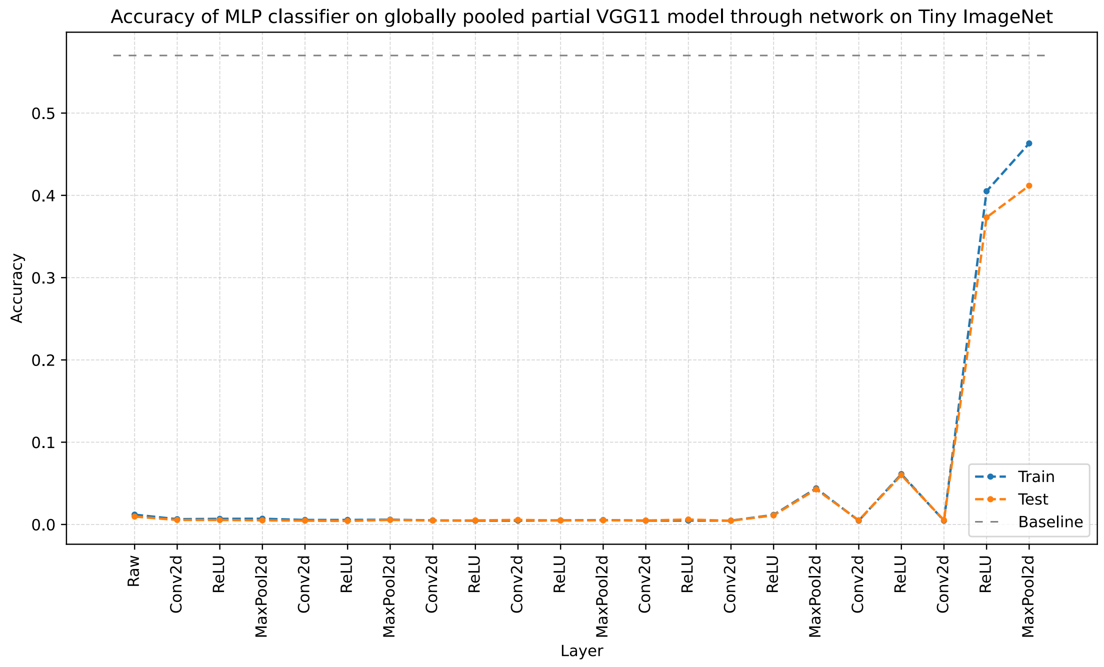


FIGURE 5.6: Test & Train accuracy of MLP probes throughout network with global pooling: Model=VGG11, Dataset=Tiny Imagenet

We can see that again there is significant difficulty in training the MLP models, despite the greatly reduced number of features, particularly on Tiny Imagenet. While the reduced number of features makes it feasible to run the training loop, in many cases, and indeed almost all with Tiny Imagenet, the classifier appears to not learn. This is rather surprising given the relatively small number of features and parameters, given that the MLP models each had only 256 hidden units. Whether it is significant that the models only learn in a few layers — at the end of the model, and only on ReLU or Max Pooling layers — is uncertain. If we focus on the results on CIFAR-10 , where the models do mostly seem to learn, we can see again that by the end of the model the classifiers achieve test accuracy very close to the baseline of the full models. This further supports the hypothesis that model transforms information from fine-grained pixel-level information into coarse, feature level information. It is interesting that the performance, both in training and test, is so erratic. Even in the cases where the accuracy drops very low, the classifiers do appear to learn, as the training performance is not as low as random performance would be (an accuracy of 10%), although the test performance is close to this. It is interesting also that in all cases the most erratic performance is seen in the middle layers, with large spikes between convolutional and ReLU layers. This also seems to occur specifically between the third and fourth max pooling layer in each model. The fact that this trend is seen in each model is interesting, but there is no obvious explanation for this.

5.5 Summary

In this chapter I experimented with training hidden layer probes on globally pooled feature maps through CNN models, to investigate the potential transformation of pixel-level information into coarse features through the networks. Using a linear classifier on the pooled features yielded performance similar to that achieved on the entire feature maps, as in chapter 4, indicating that not only does the data become more linearly separable, but that this linear separability is present in the coarse features of the latent space, rather than location-specific features. Similar trends could also be observed to those in chapter 4, with convolutional layers seeming to decrease separability, and ReLU layers recovering this decrease and improving beyond it, indicating that the layers work in tandem.

Chapter 6

Discussion & Conclusions

If we compare results across the different experiments, we see some particular trends. There appears to be a clear pattern of different layers transforming the data in different, but consistent, ways. Convolutional layers appear to transform the data in a way that makes the data more difficult to classify, decreasing the class margin and linear separability, and in some cases reducing the strength of coarse features. In contrast, ReLU layers appear to transform the data such that it becomes more easy to classify, increasing the class margin and linear separability, and increasing the strength of coarse features. In concert, the convolutional and ReLU layers appear to form a pair which makes the data easier to classify. As such, the penalty to classification ability that is incurred in the convolutional layers does not reduce the salient information in the data, merely encode it in a way that can be extracted by the ReLU. Moreover this process is gradual, with each of the convolution-ReLU pairs providing a small improvement over the last. This indicates that rather than the data being entangled in a complex way through the whole network, before being disentangled all at once in the final layer(s), the layers slowly disentangle the information bit by bit. It may be that this is a reason for the good generalisation performance, in either that these layer pairs are only capable of incremental disentanglement of this information, or that doing so in small increments reduces the likelihood of disentangling spurious information, and thereby distils more salient information from the data, yielding better generalisation. This would give some answer as to why depth may be preferred over width, as widening the network would not provide the incremental transformation of the data that is achieved with depth.

Max pooling layers are more curious. It appears that max pooling layers can give substantial increases in the size of the class margin, however they do not appear to have much effect on linear separability of the data or the coarse features within the latent representations. However, given the amount of information they discard, by drastically reducing the dimensionality, the fact that they do not appear to decrease linear separability or reduce the strength of coarse features suggests that the information they discard is not salient. It therefore seems they do have an important purpose in discarding spurious information.

There is certainly evidence to indicate that the CNN models being investigated perform gradual transformations to the data which simplify the problem, thereby making the problem easier, and that this could account for some of the generalisation capabilities these models exhibit. The class margins and linear separability of the data both appear to increase through the models, in clearly identifiable trends across different operations. These transformations make the data much easier to classify in a way which generalises, as shown by the ability to achieve performance close to the full network using only a linear classifier. Moreover, the coarse features learned by the network become more easy to classify, and similar performance can again be achieved by classifying only the globally pooled features, with no fine-grained positional information, and only a fraction of the total representation. Again this increases gradually layer by layer, and clearly indicates a simplification of the problem, with individual features which are much more salient to the classification than the pixel-specific features that are the input to the model. These findings indicate that over-parameterisation is not such a core component of generalisation as some other work would suggest, and that instead the compositional structure of gradually building complex representations, slowly distilling salient information and filtering out spurious information, is more significant in explaining the generalisation capability of deep models.

Chapter 7

Future Work

There is a wealth of possibility for future work building on these findings, which could provide even greater insight into the nature of deep networks, and further the goal of understanding exactly how and why deep networks are able to generalise so well. It would be extremely interesting to repeat the investigations of this work on a broader class of deep learning models, both those used in image classification, and in other tasks. Models such as ResNets (He et al. 2016) and transformers (Kolesnikov et al. 2021) have in recent years become more prominent in image classification problems, and investigating the behaviour of these would provide more valuable insight in the study of generalisation. Outside of the image classification space, transformer models (Vaswani et al. 2017) have become particularly prevalent in the field of natural language processing, with very impressive results seen from a variety of models, many of which have extremely large numbers of parameters.

While this work gives an indication of *how* CNN models are able to generalise, i.e. the convolutional layers seem to transform the data in a way which filters out spurious information, and in doing so distil out linearly separable features, the question of *why* this is the case is very much open. Indeed as noted, there is no explicit reason for this process to occur, and the fact that it does is surprising. Whether this is a property unique to the convolution and/or ReLU operations, specifics of the training process, the particular problems, any combination of these, or something else entirely, is not something we can yet answer. Experimenting in a wider variety of settings, such as with different model architectures, different problems, or different methods of training may help to expand these findings and paint a more detailed picture of the nature of generalisation. Additionally, it may be possible to use these findings in driving further theoretical exploration of generalisation.

This work is only a small piece of the greater puzzle of generalisation in deep learning, however it does give some insight into the nature of generalisation in the specific domain of image classification using CNNs. Until the question of generalisation is fully solved, a goal which may be forever out of reach, there will always be further work to do in exploring generalisation, as deep learning continues to push the frontiers of machine learning, both in existing problem domains, and ones as yet unexplored.

Appendix A

Complete Results from Chapter 3

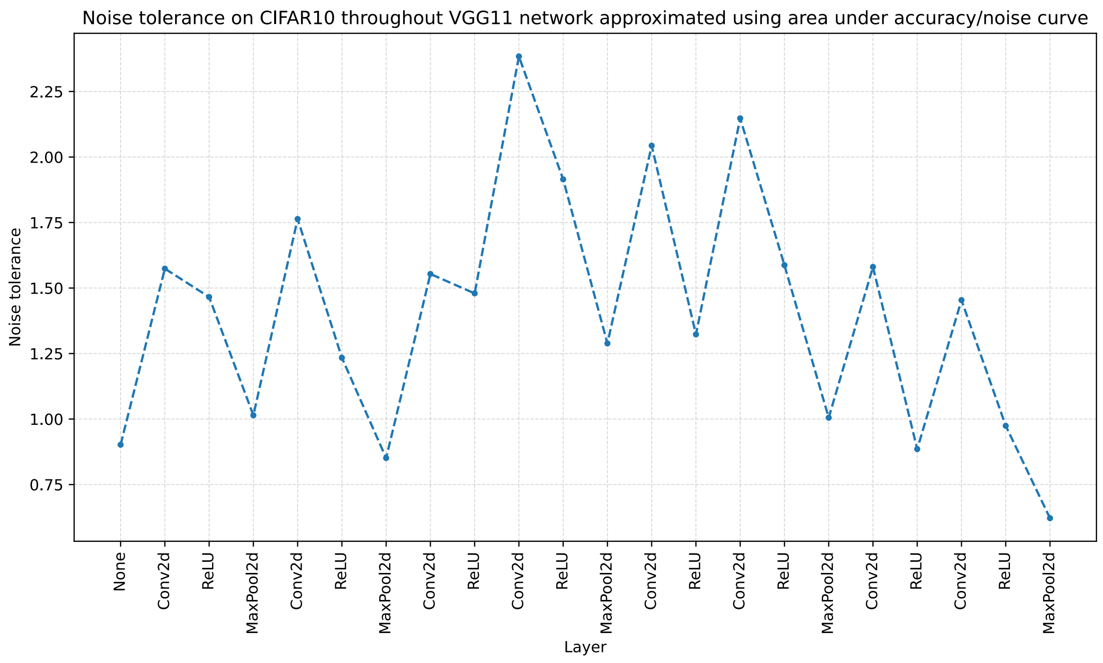


FIGURE A.1: Noise tolerance through network approximated via Simpson's rule: Model=VGG11, Dataset=CIFAR10

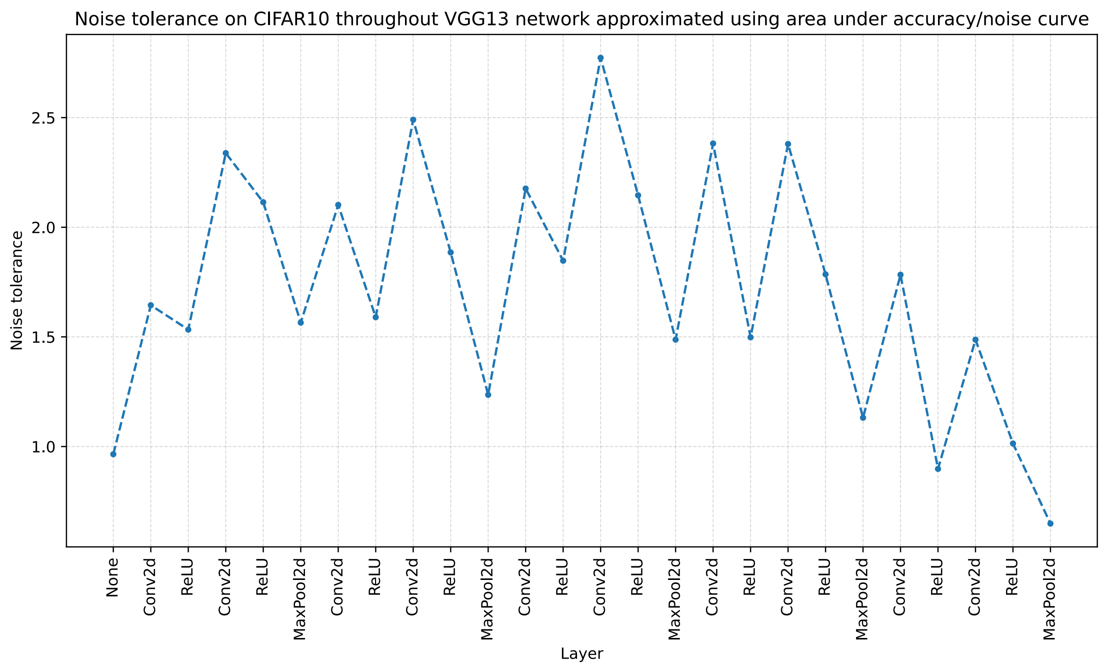


FIGURE A.2: Noise tolerance through network approximated via Simpson's rule: Model=VGG13, Dataset=CIFAR10

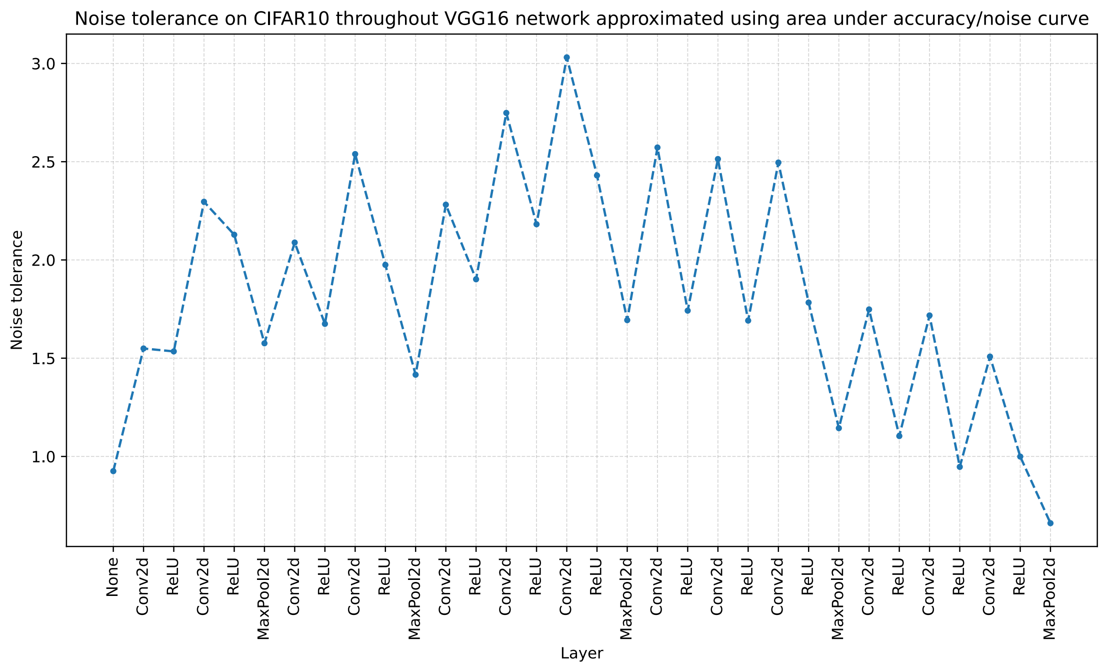


FIGURE A.3: Noise tolerance through network approximated via Simpson's rule: Model=VGG16, Dataset=CIFAR10

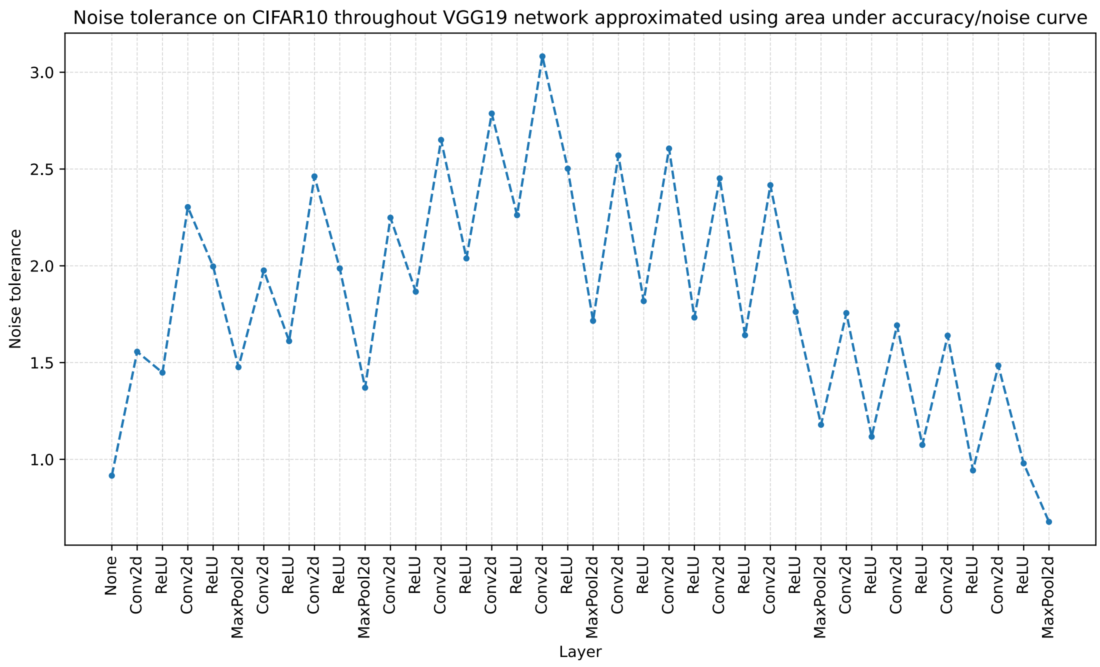


FIGURE A.4: Noise tolerance through network approximated via Simpson's rule: Model=VGG19, Dataset=CIFAR10

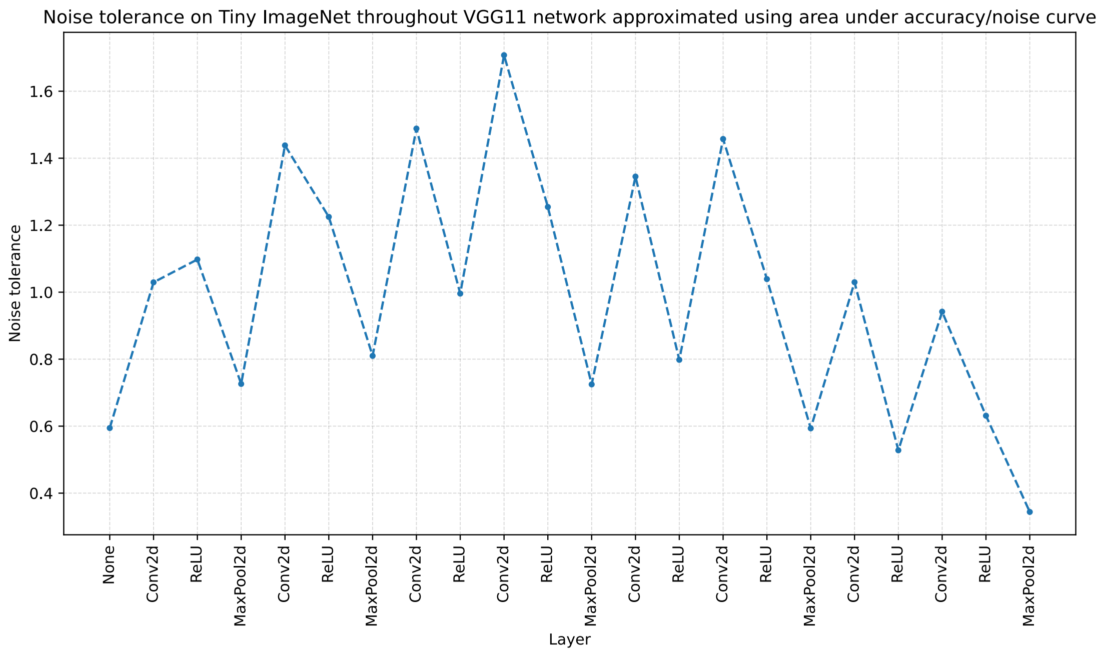


FIGURE A.5: Noise tolerance through network approximated via Simpson's rule: Model=VGG11, Dataset=Tiny Imagenet

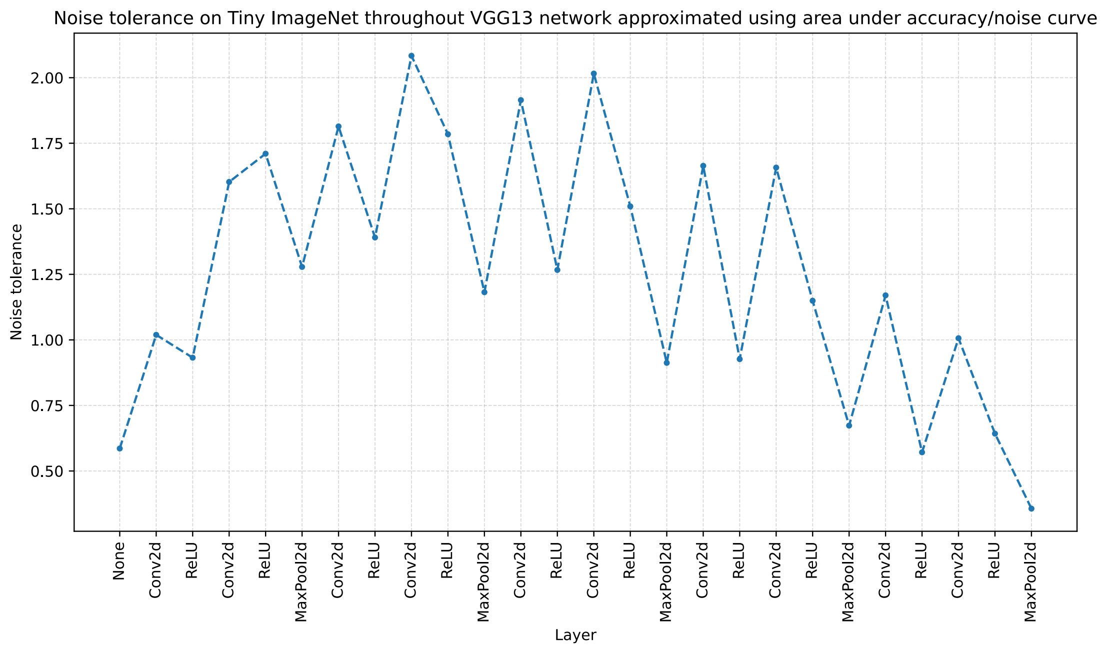


FIGURE A.6: Noise tolerance through network approximated via Simpson's rule: Model=VGG13, Dataset=Tiny Imagenet

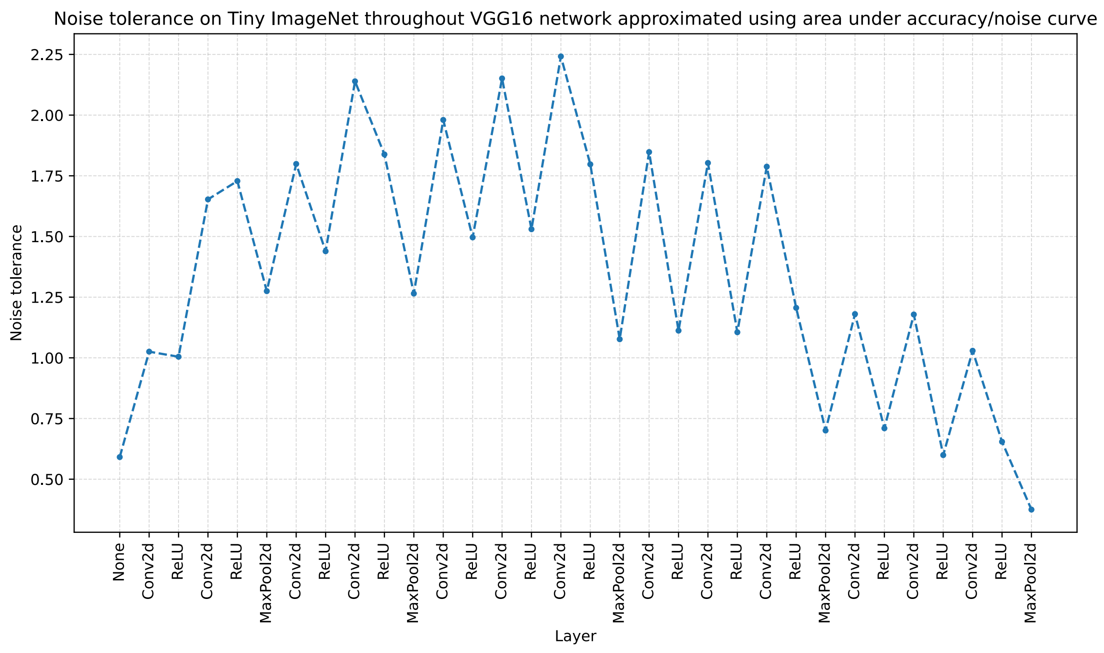


FIGURE A.7: Noise tolerance through network approximated via Simpson's rule: Model=VGG16, Dataset=Tiny Imagenet

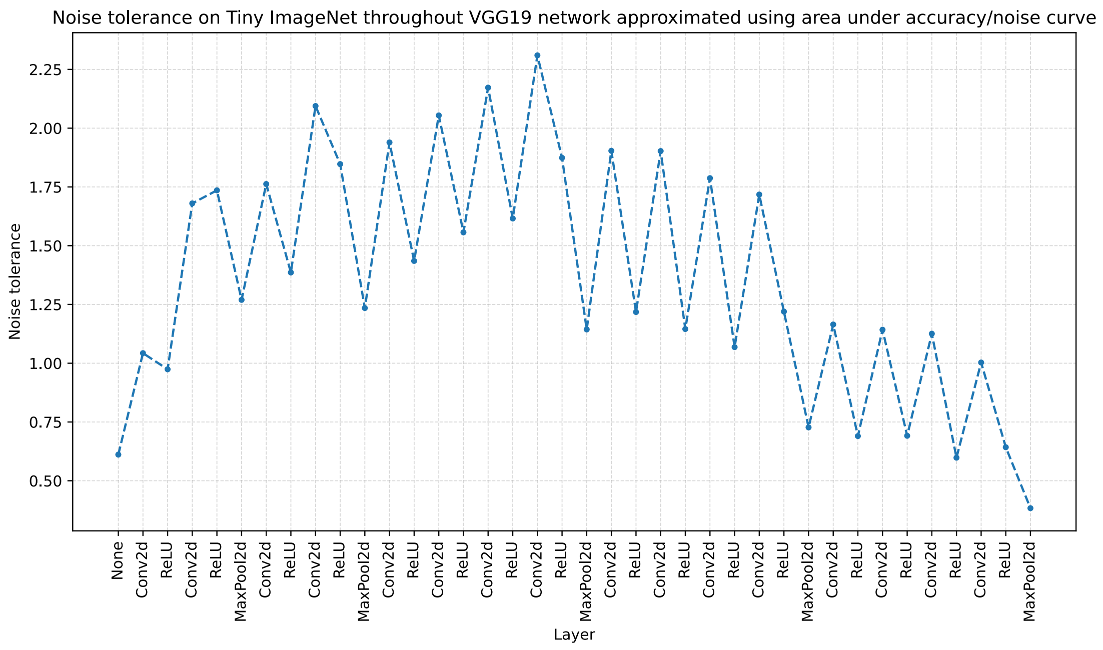


FIGURE A.8: Noise tolerance through network approximated via Simpson's rule: Model=VGG19, Dataset=Tiny Imagenet

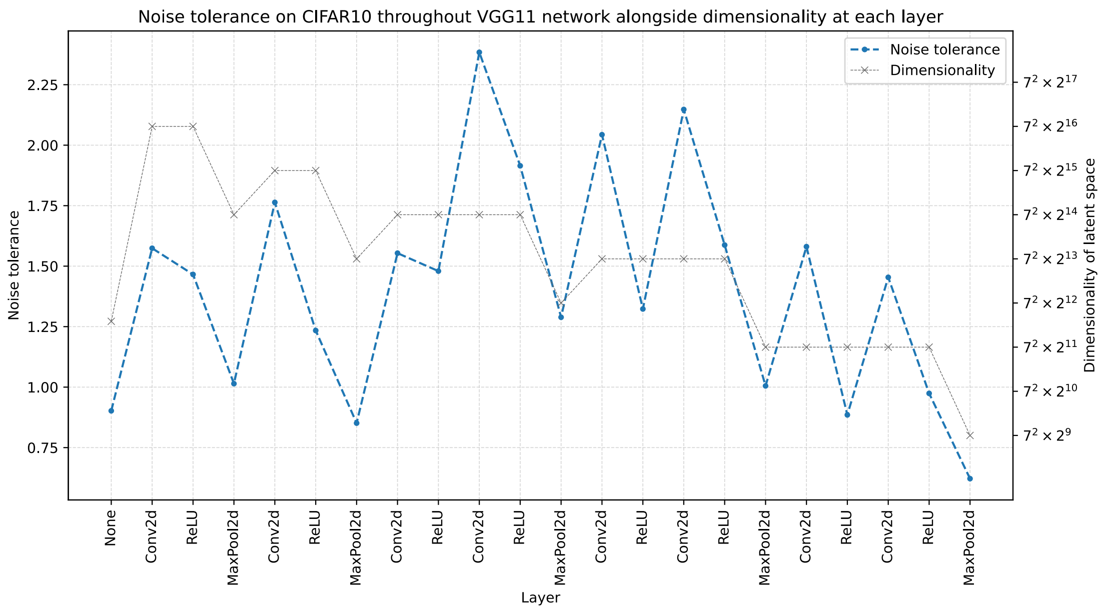


FIGURE A.9: Noise tolerance & dimensionality through network: Model=VGG11, Dataset=CIFAR10

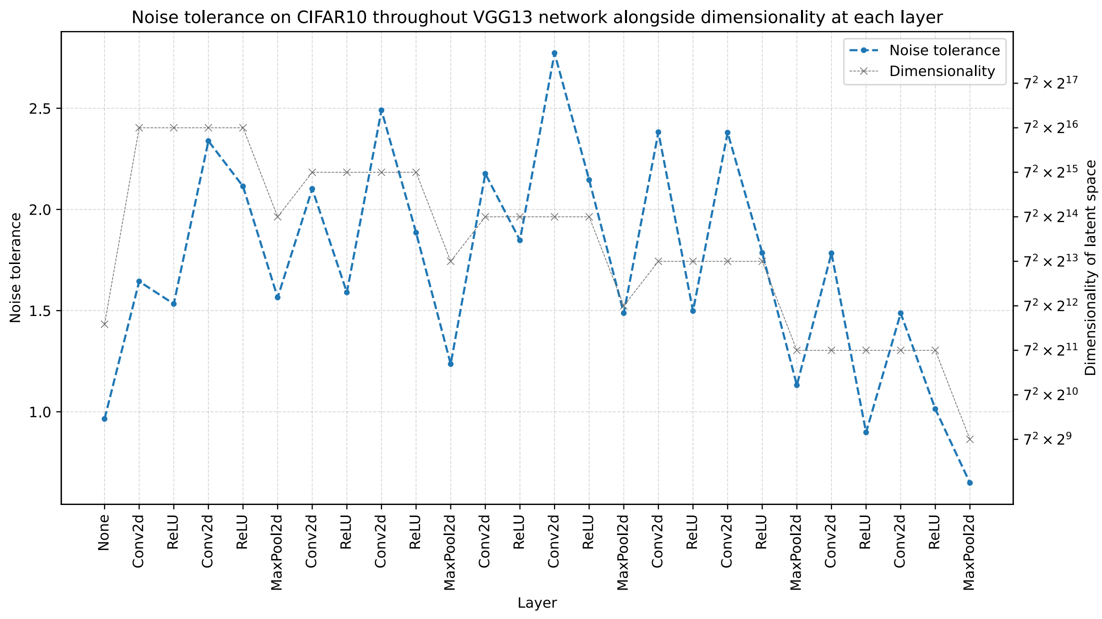


FIGURE A.10: Noise tolerance & dimensionality through network: Model=VGG13, Dataset=CIFAR10

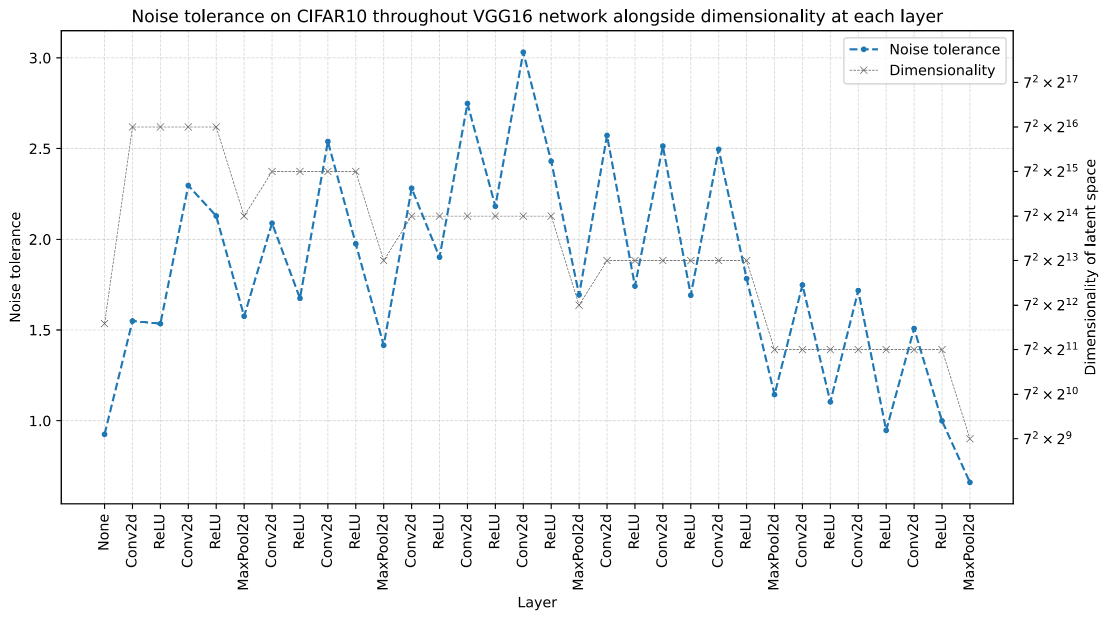


FIGURE A.11: Noise tolerance & dimensionality through network: Model=VGG16, Dataset=CIFAR10

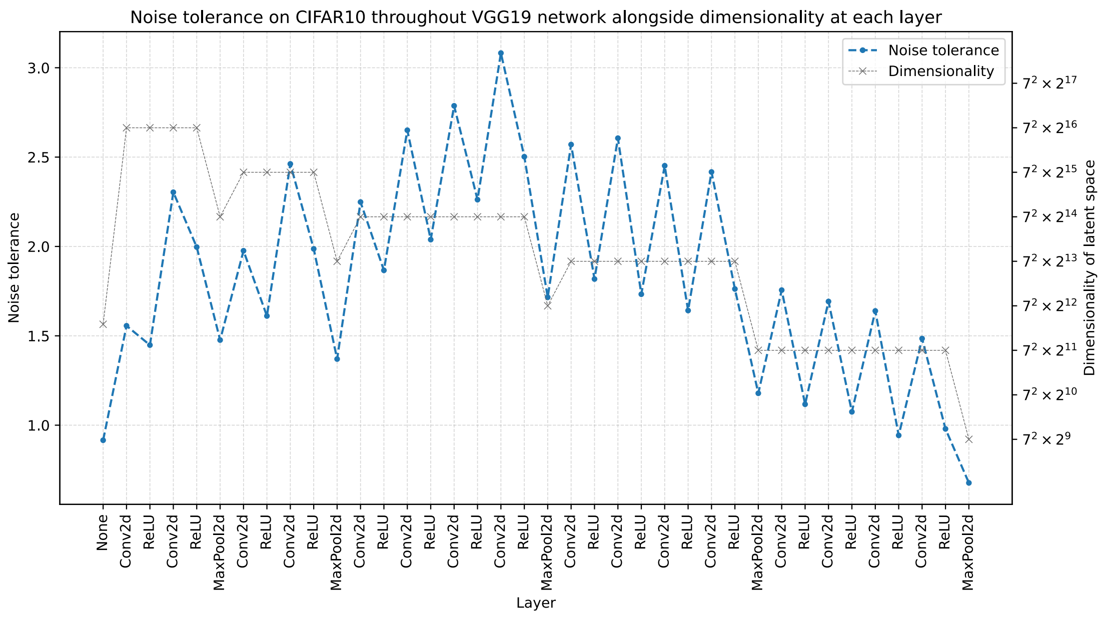


FIGURE A.12: Noise tolerance & dimensionality through network: Model=VGG19, Dataset=CIFAR10

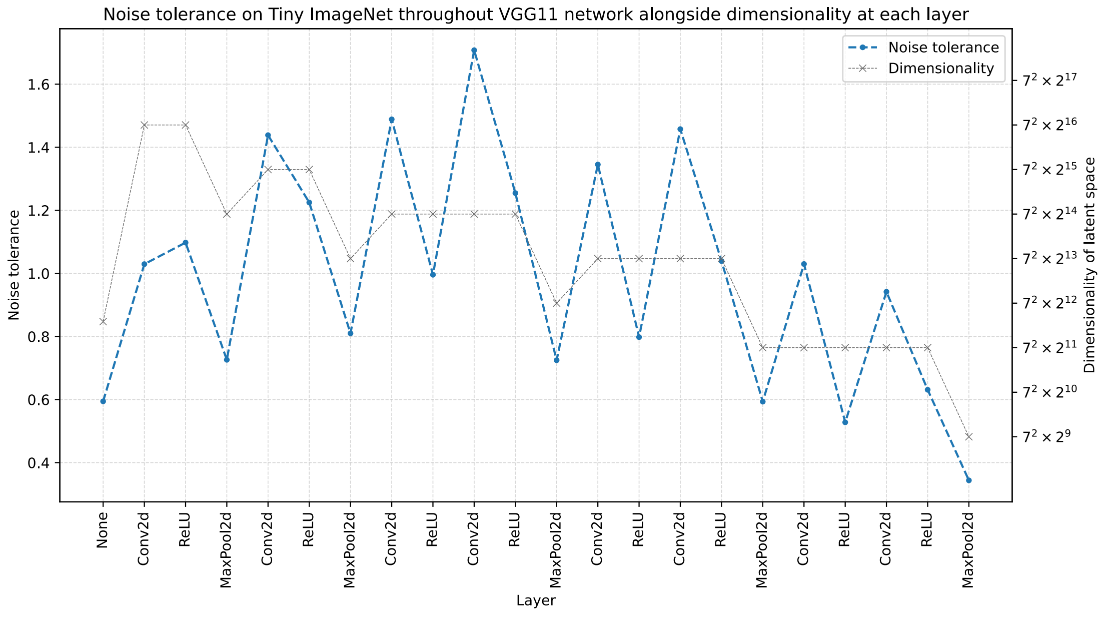


FIGURE A.13: Noise tolerance & dimensionality through network: Model=VGG11, Dataset=Tiny Imagenet

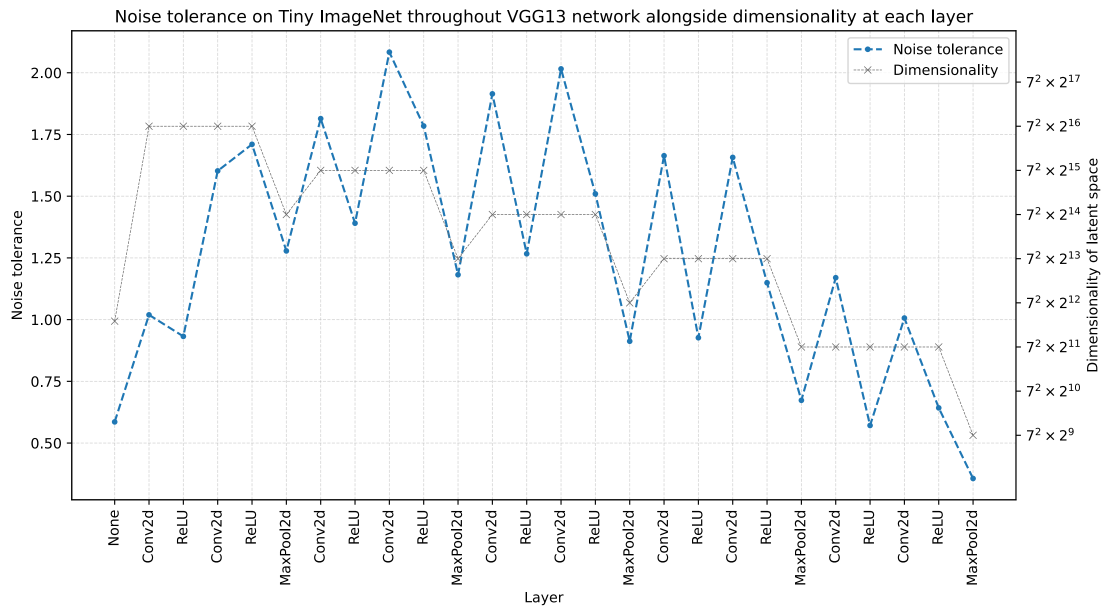


FIGURE A.14: Noise tolerance & dimensionality through network: Model=VGG13, Dataset=Tiny Imagenet

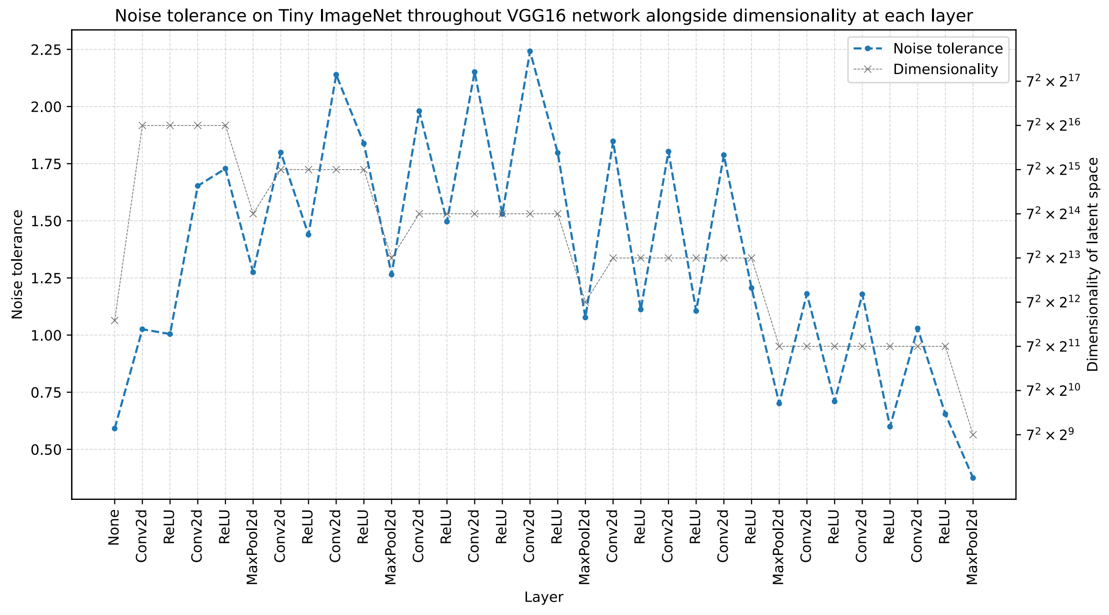


FIGURE A.15: Noise tolerance & dimensionality through network: Model=VGG16, Dataset=Tiny Imagenet

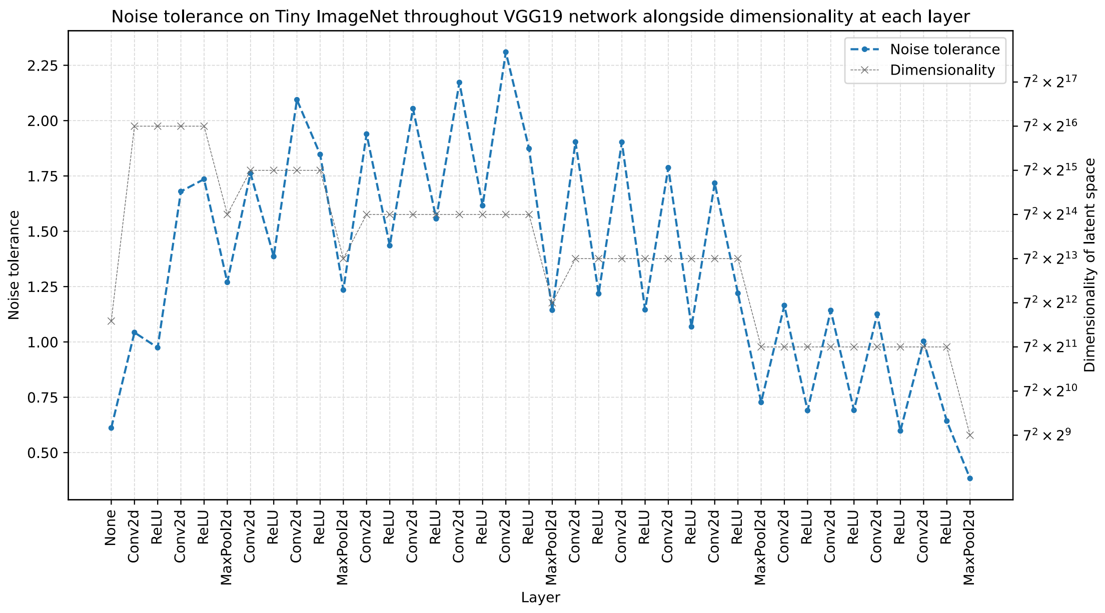


FIGURE A.16: Noise tolerance & dimensionality through network: Model=VGG19, Dataset=Tiny Imagenet

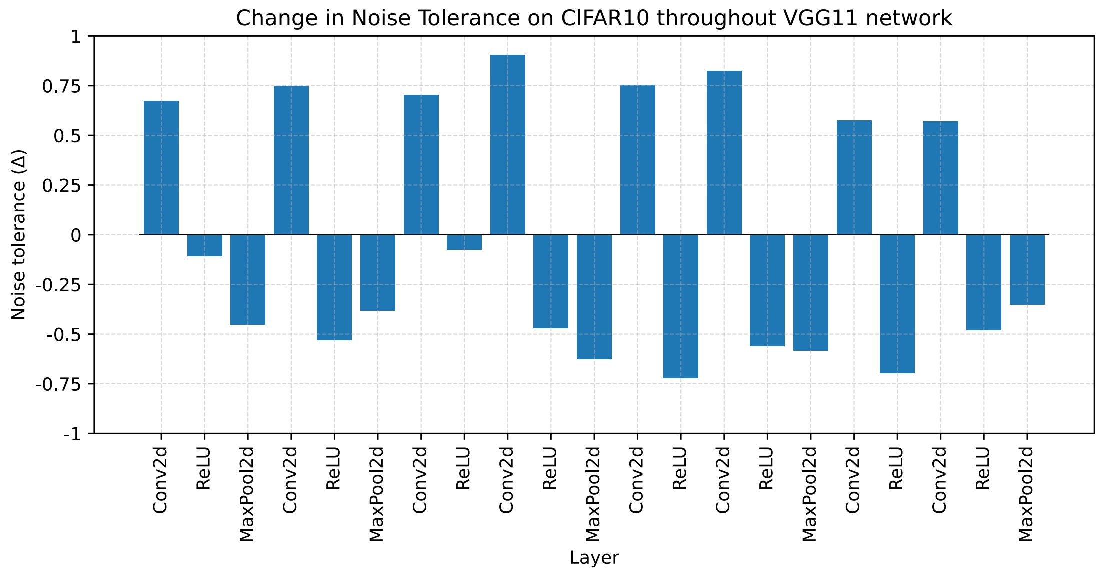


FIGURE A.17: Change in noise tolerance through network: Model=VGG11, Dataset=CIFAR10

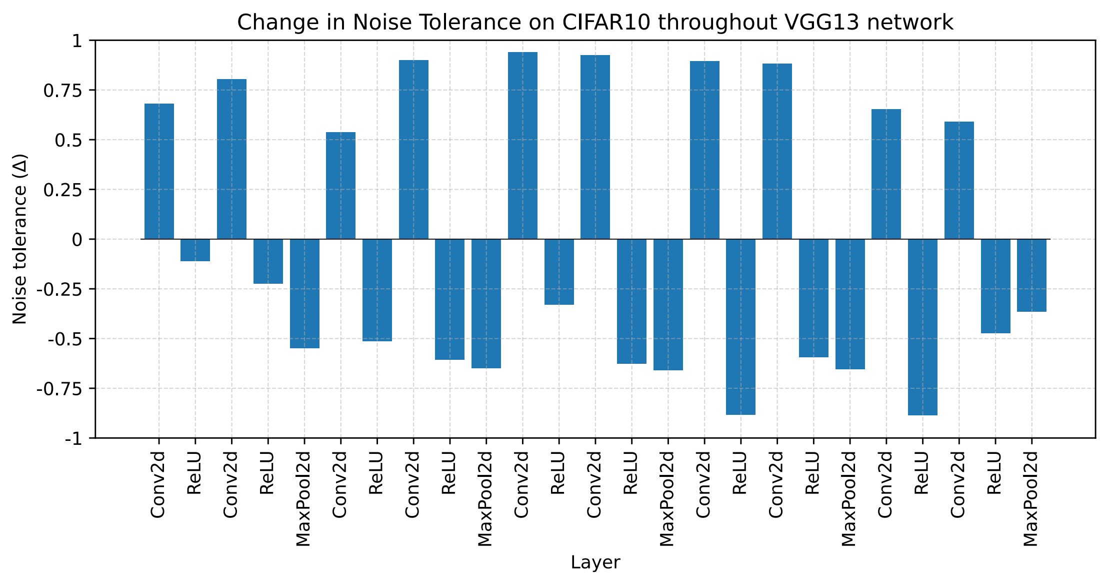


FIGURE A.18: Change in noise tolerance through network: Model=VGG13, Dataset=CIFAR10

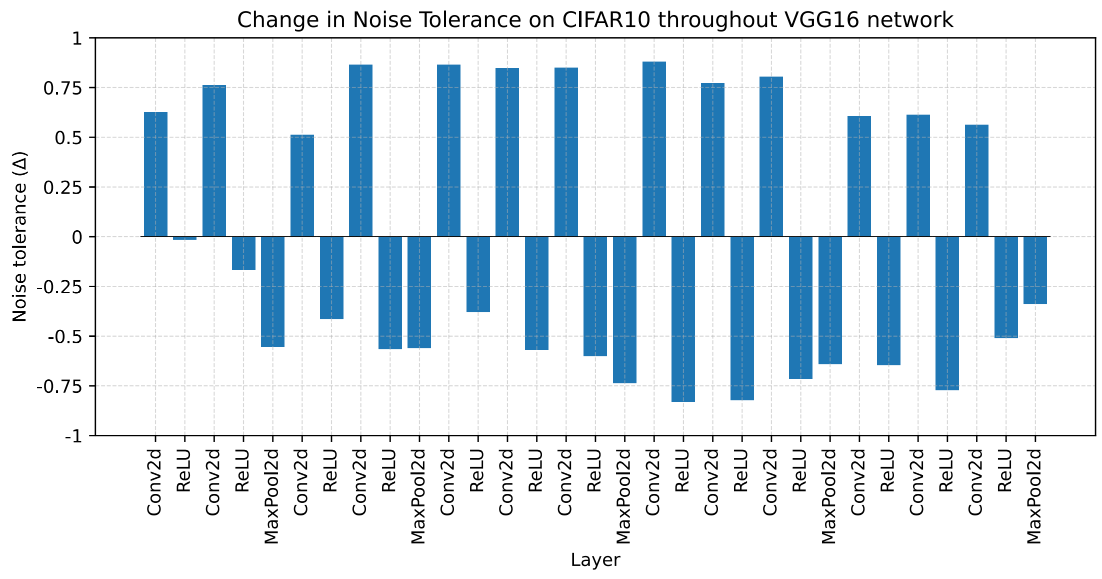


FIGURE A.19: Change in noise tolerance through network: Model=VGG16, Dataset=CIFAR10

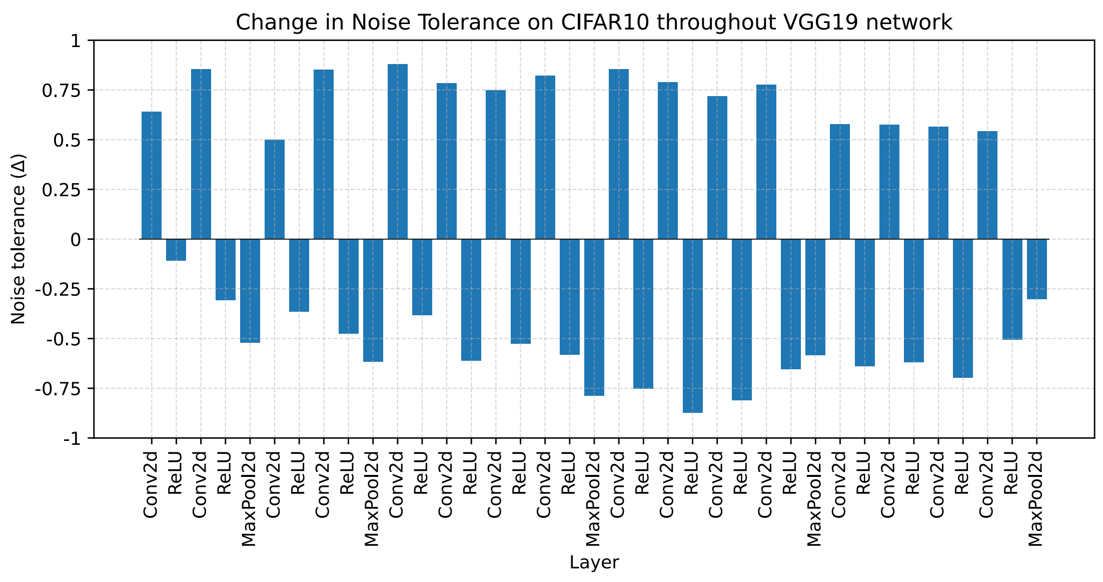


FIGURE A.20: Change in noise tolerance through network: Model=VGG19, Dataset=CIFAR10

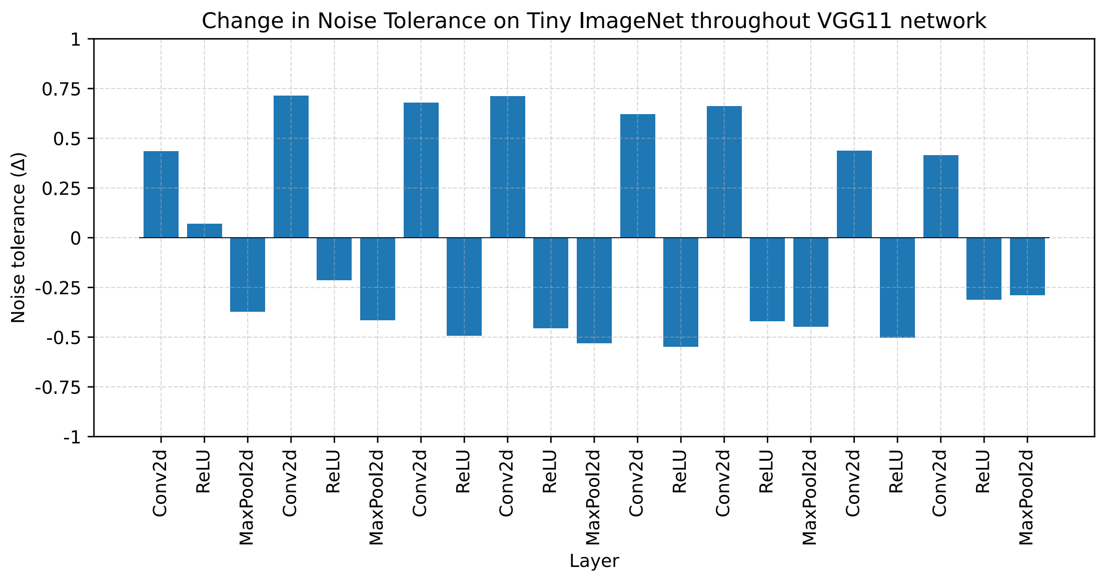


FIGURE A.21: Change in noise tolerance through network: Model=VGG11, Dataset=Tiny Imagenet

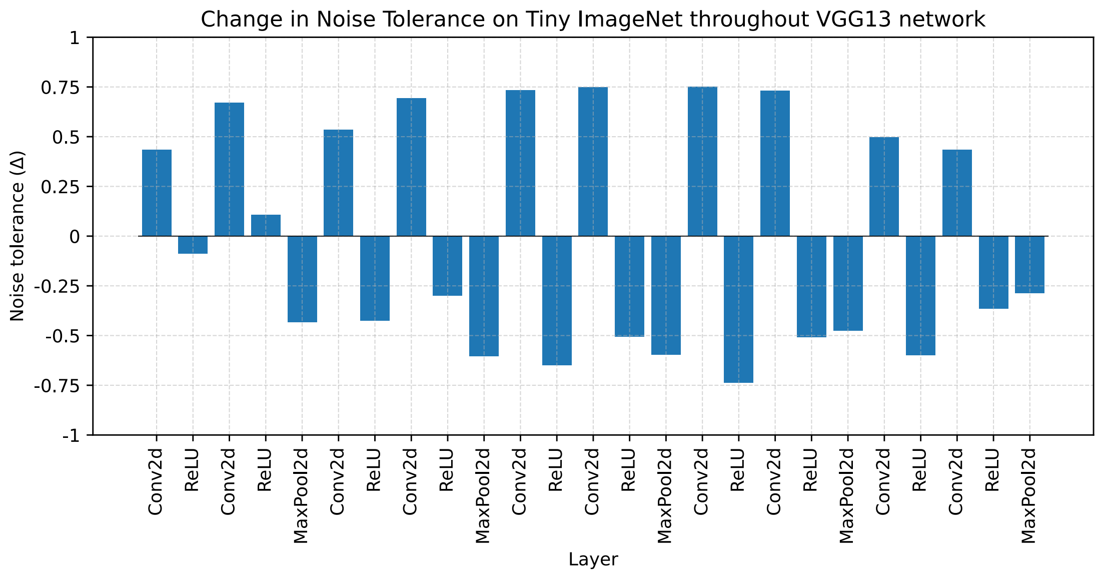


FIGURE A.22: Change in noise tolerance through network: Model=VGG13, Dataset=Tiny Imagenet

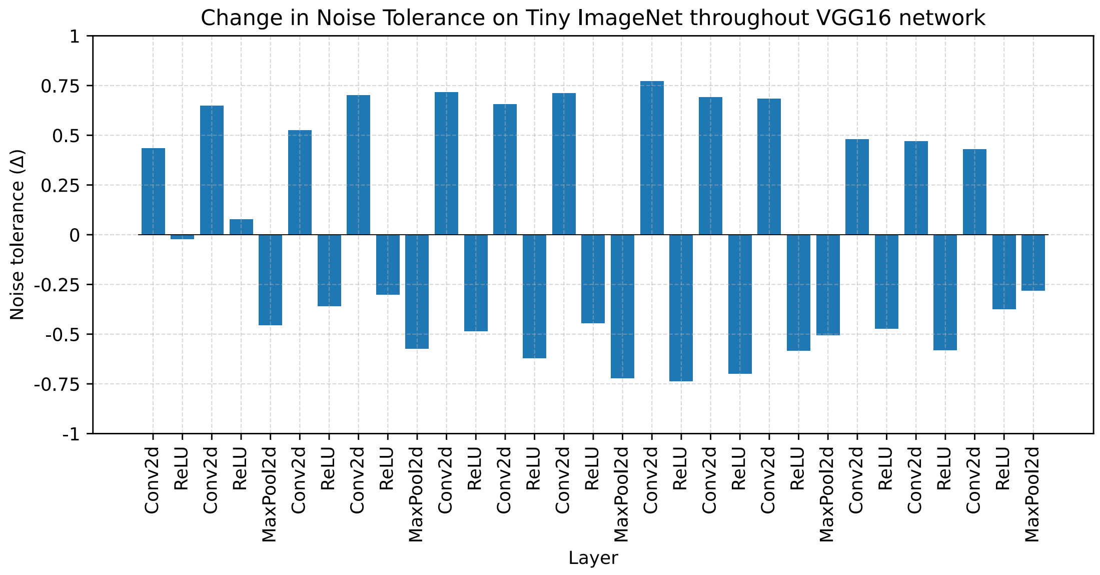


FIGURE A.23: Change in noise tolerance through network: Model=VGG16, Dataset=Tiny Imagenet

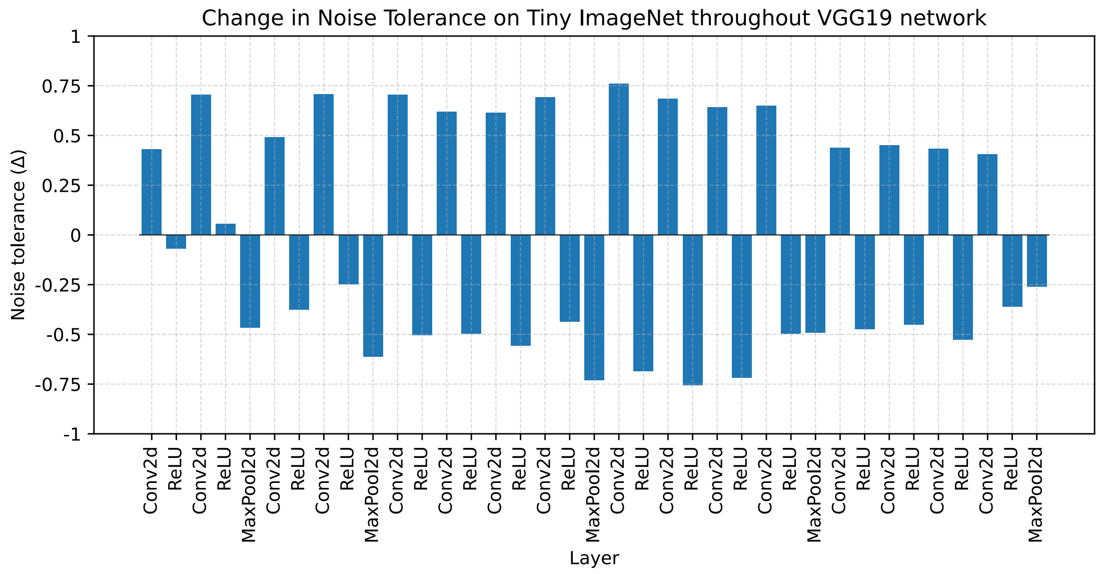


FIGURE A.24: Change in noise tolerance through network: Model=VGG19, Dataset=Tiny Imagenet

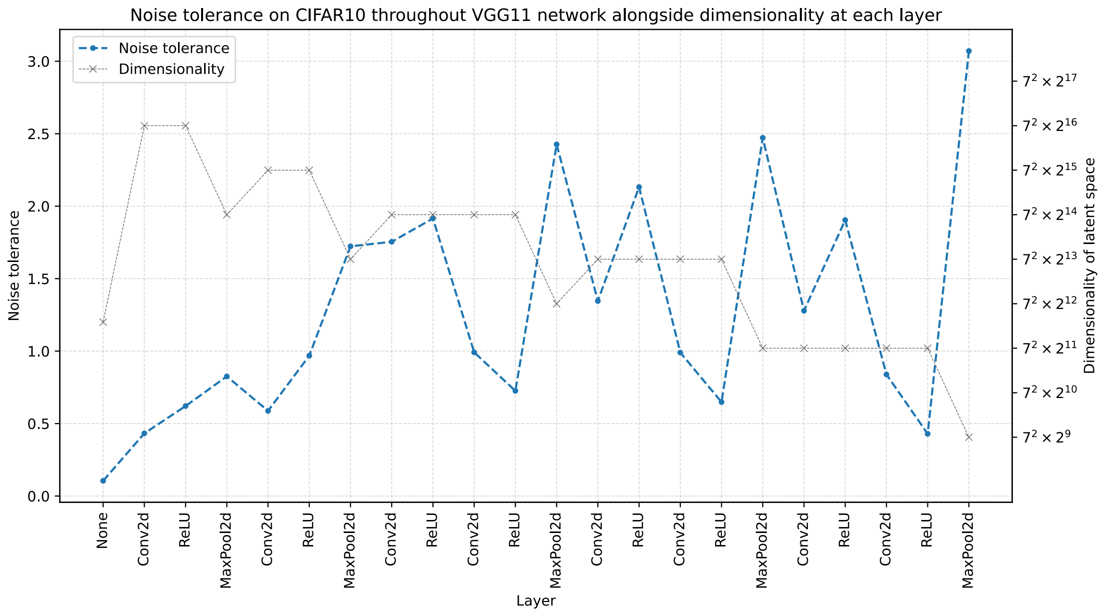


FIGURE A.25: Noise tolerance through network with Gaussian noise: Model=VGG11, Dataset=CIFAR10

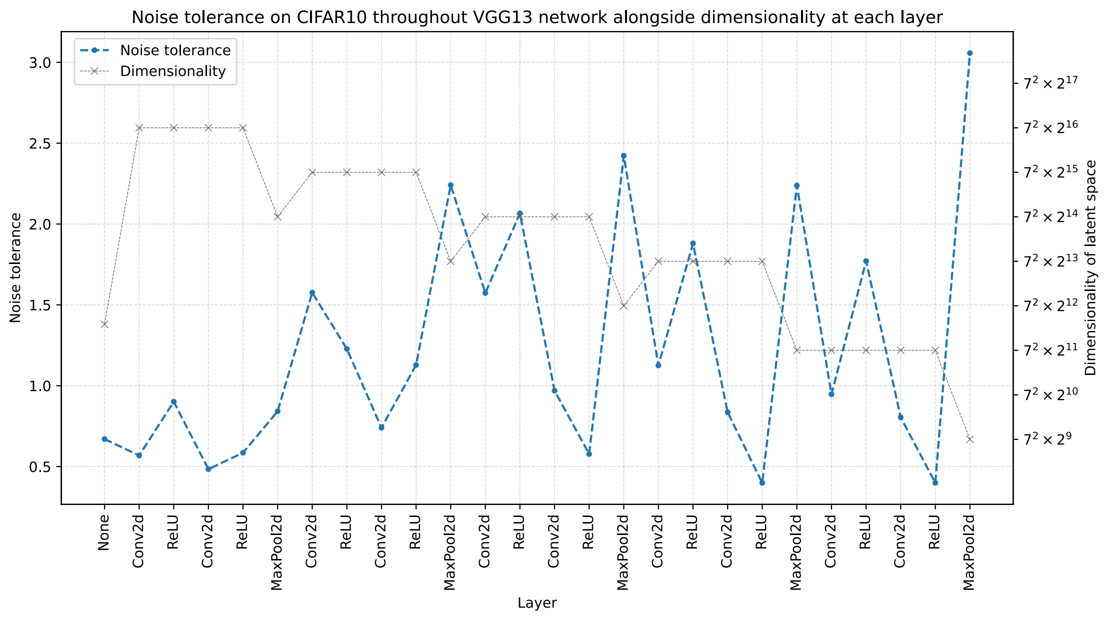


FIGURE A.26: Noise tolerance through network with Gaussian noise: Model=VGG13, Dataset=CIFAR10

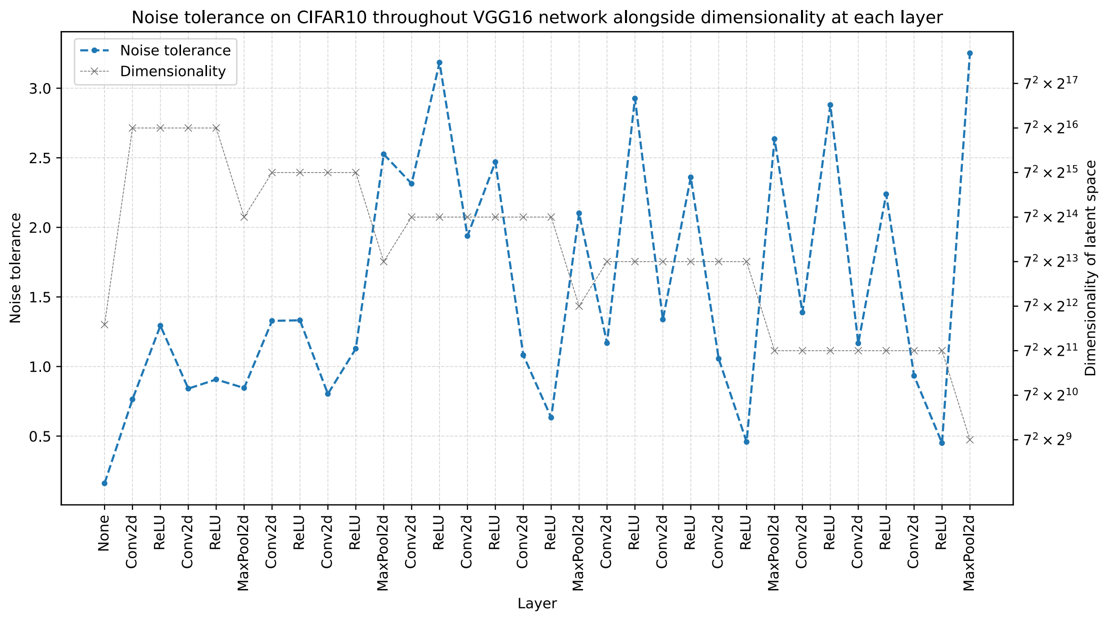


FIGURE A.27: Noise tolerance through network with Gaussian noise: Model=VGG16, Dataset=CIFAR10

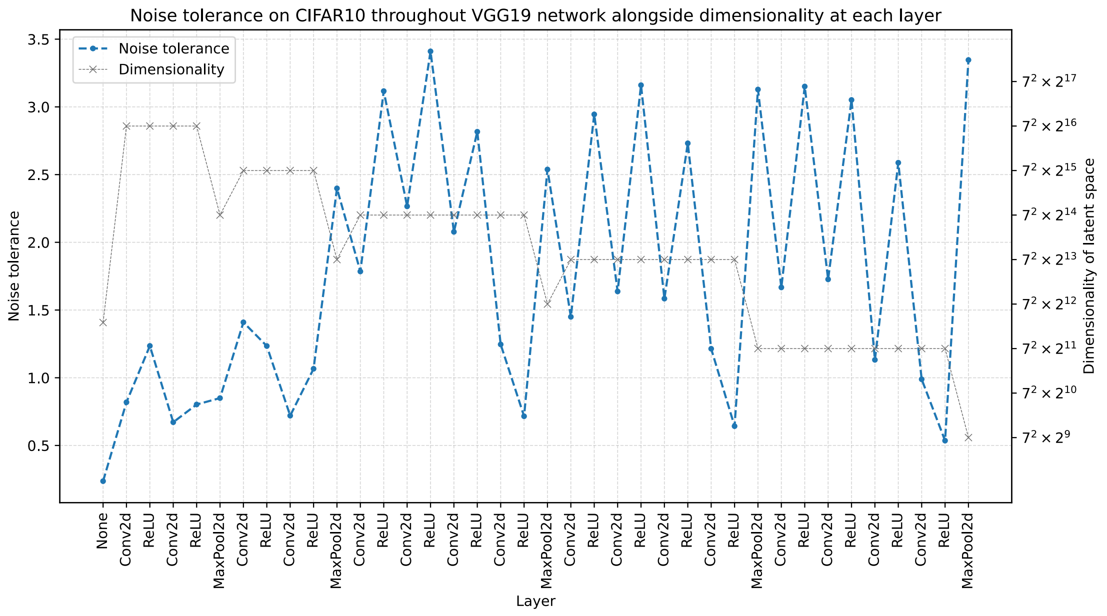


FIGURE A.28: Noise tolerance through network with Gaussian noise: Model=VGG19, Dataset=CIFAR10

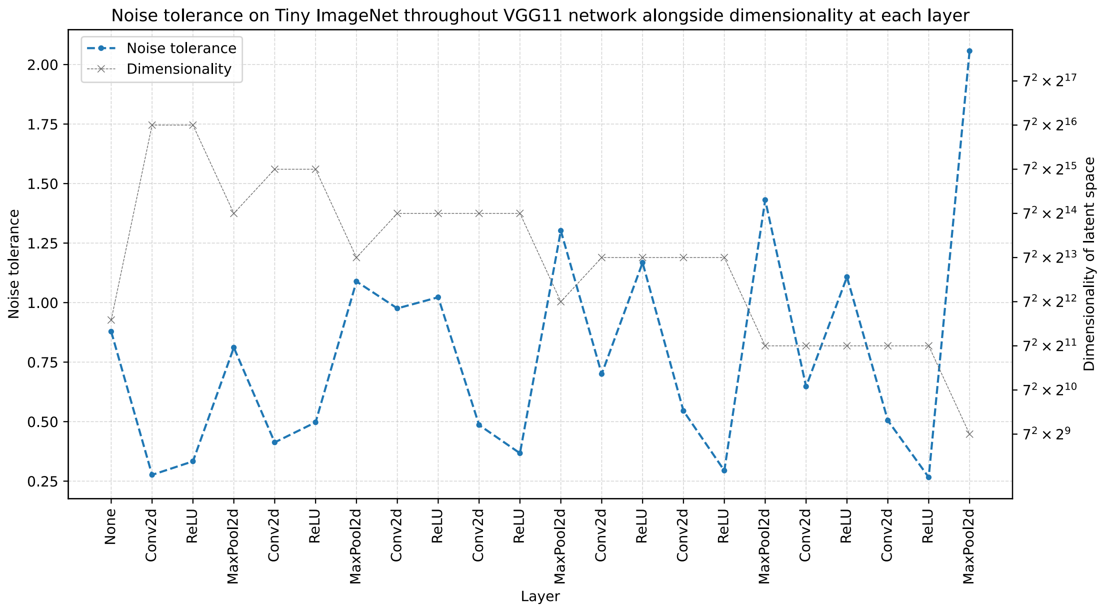


FIGURE A.29: Noise tolerance through network with Gaussian noise: Model=VGG11, Dataset=Tiny Imagenet

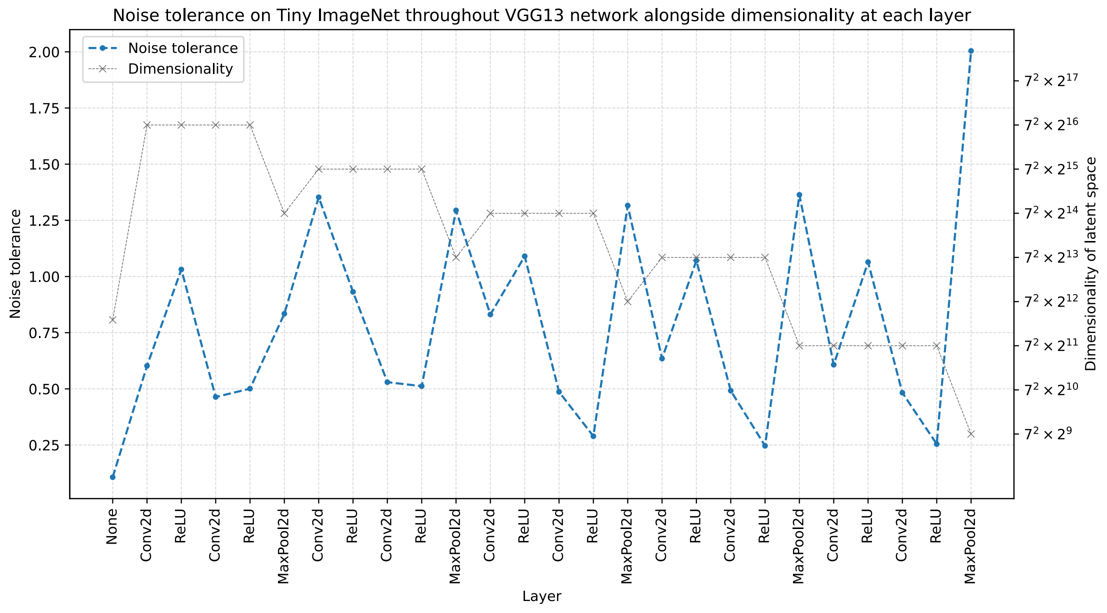


FIGURE A.30: Noise tolerance through network with Gaussian noise: Model=VGG13, Dataset=Tiny Imagenet

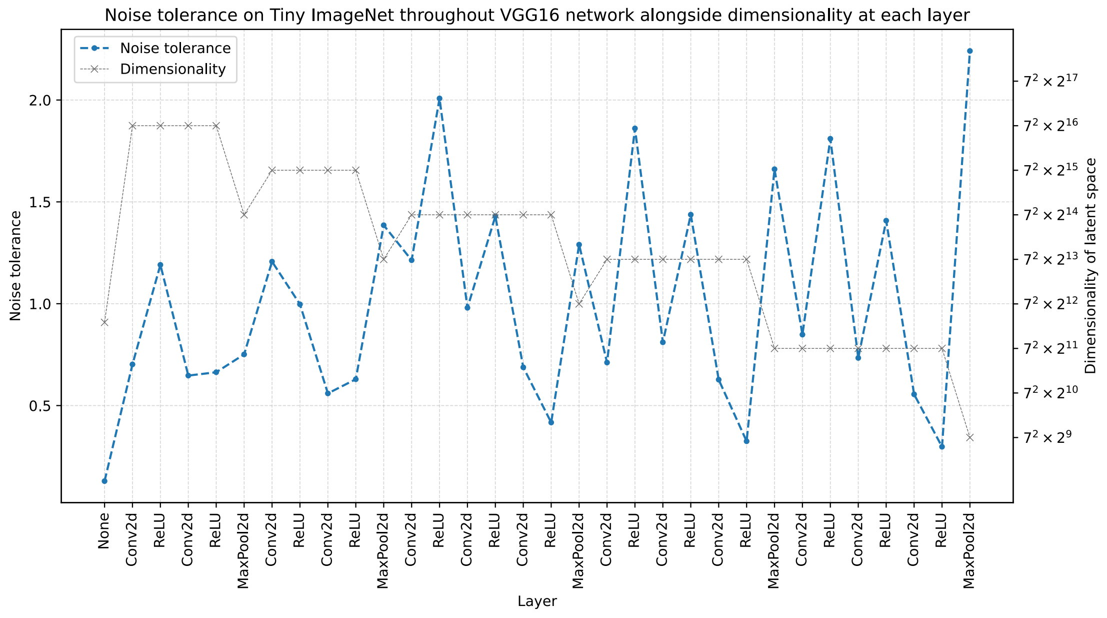


FIGURE A.31: Noise tolerance through network with Gaussian noise: Model=VGG16, Dataset=Tiny Imagenet

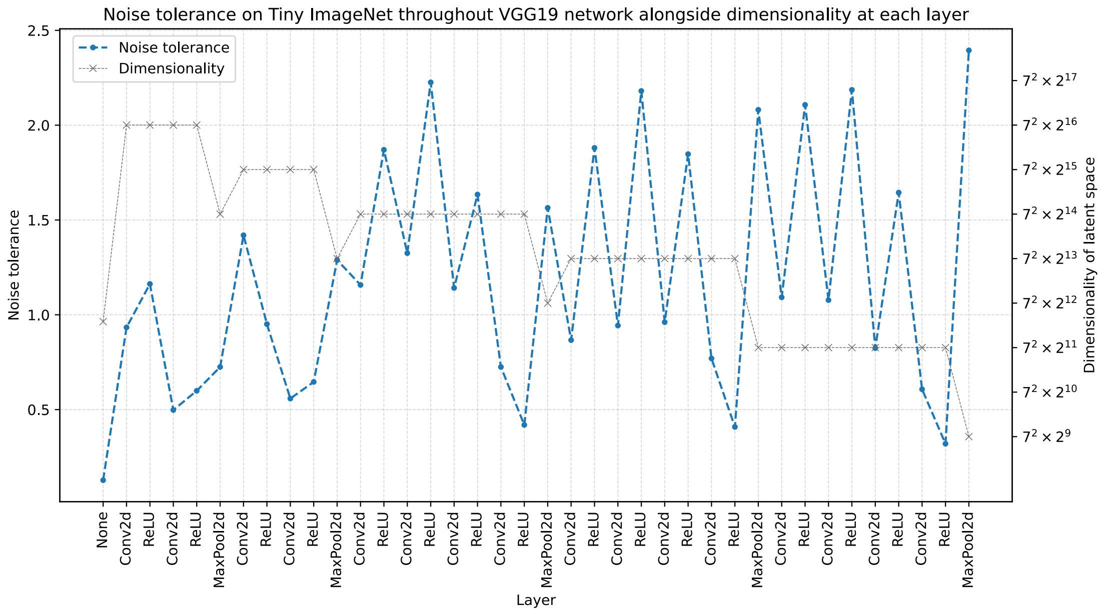


FIGURE A.32: Noise tolerance through network with Gaussian noise: Model=VGG19, Dataset=Tiny Imagenet

Appendix B

Complete Results from Chapter 4

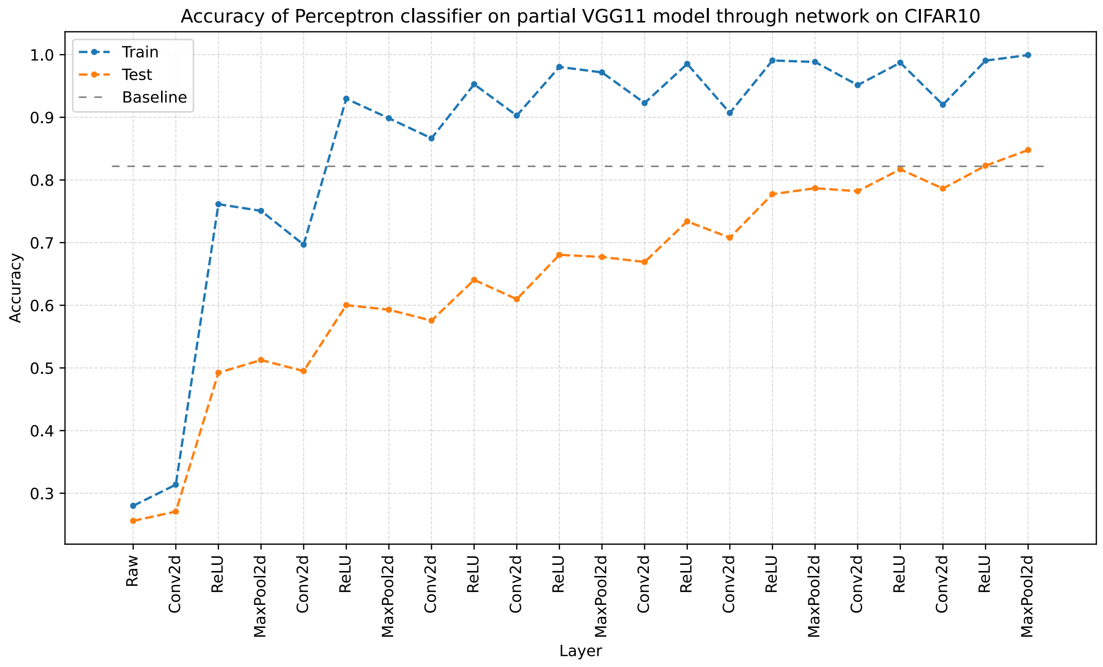


FIGURE B.1: Test & Train accuracy of perceptron probes throughout network: Model=VGG11, Dataset=CIFAR10

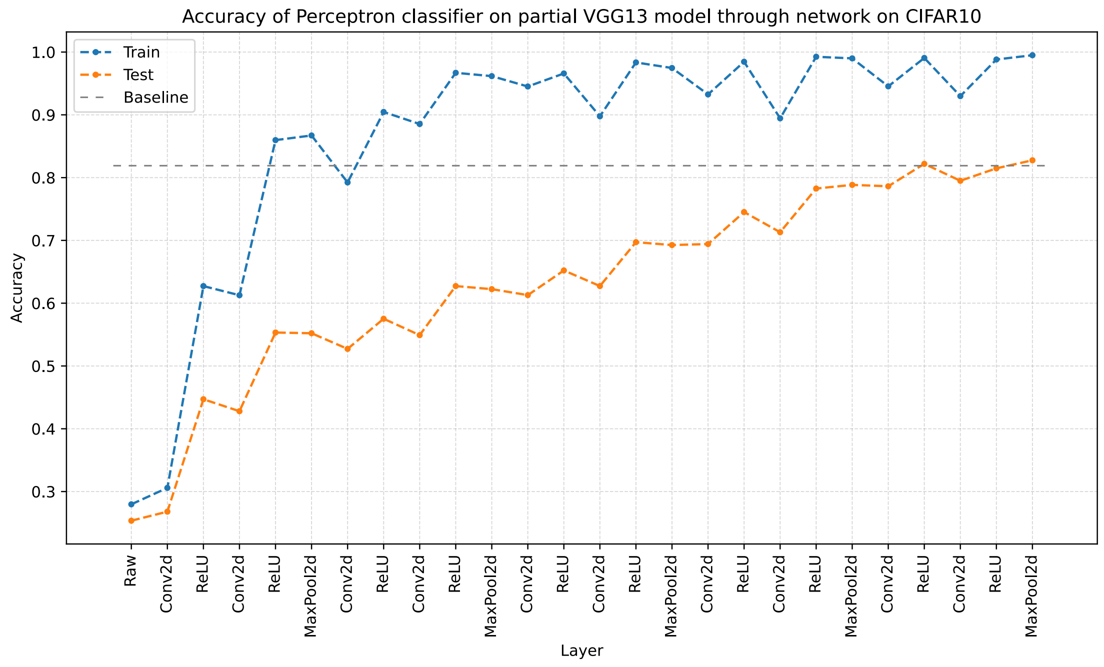


FIGURE B.2: Test & Train accuracy of perceptron probes throughout network: Model=VGG13, Dataset=CIFAR10

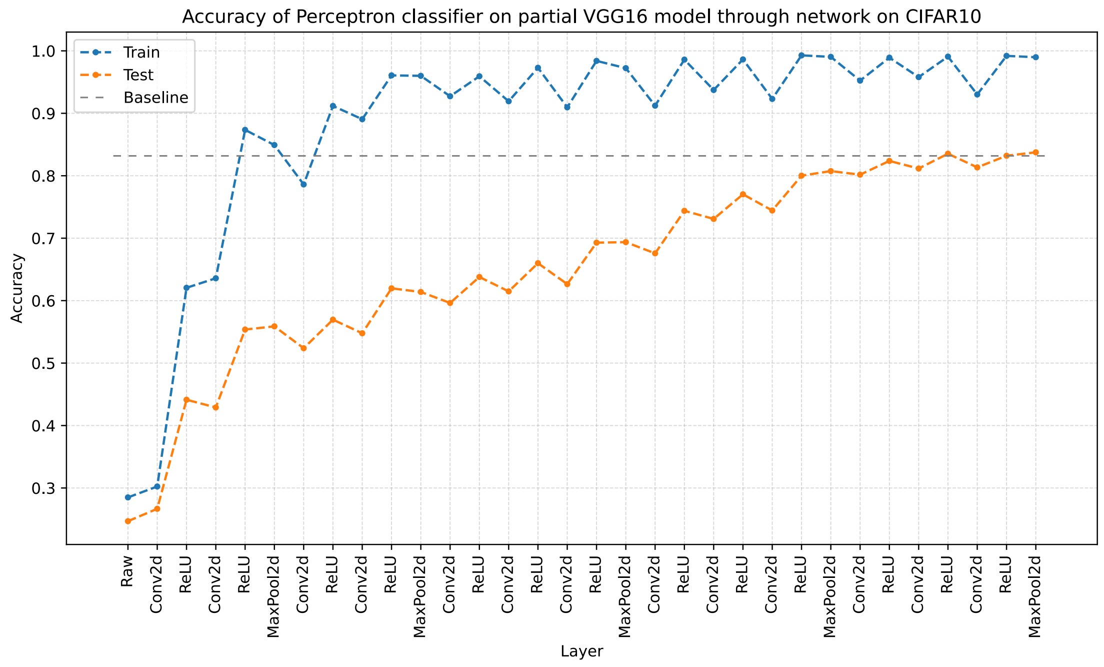


FIGURE B.3: Test & Train accuracy of perceptron probes throughout network: Model=VGG16, Dataset=CIFAR10

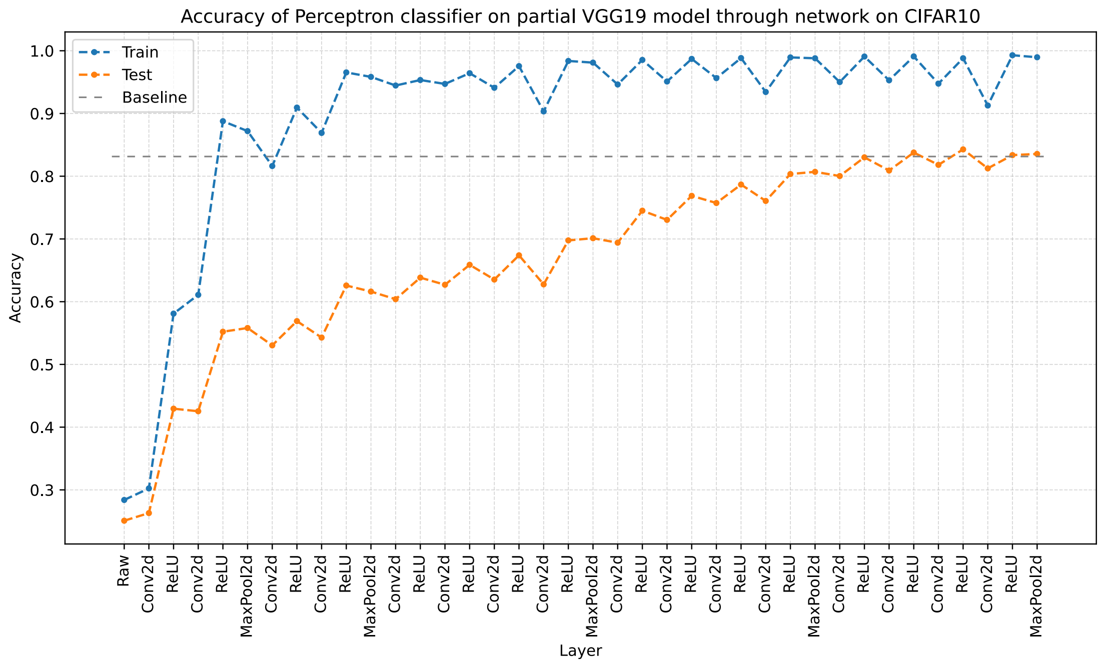


FIGURE B.4: Test & Train accuracy of perceptron probes throughout network: Model=VGG19, Dataset=CIFAR10

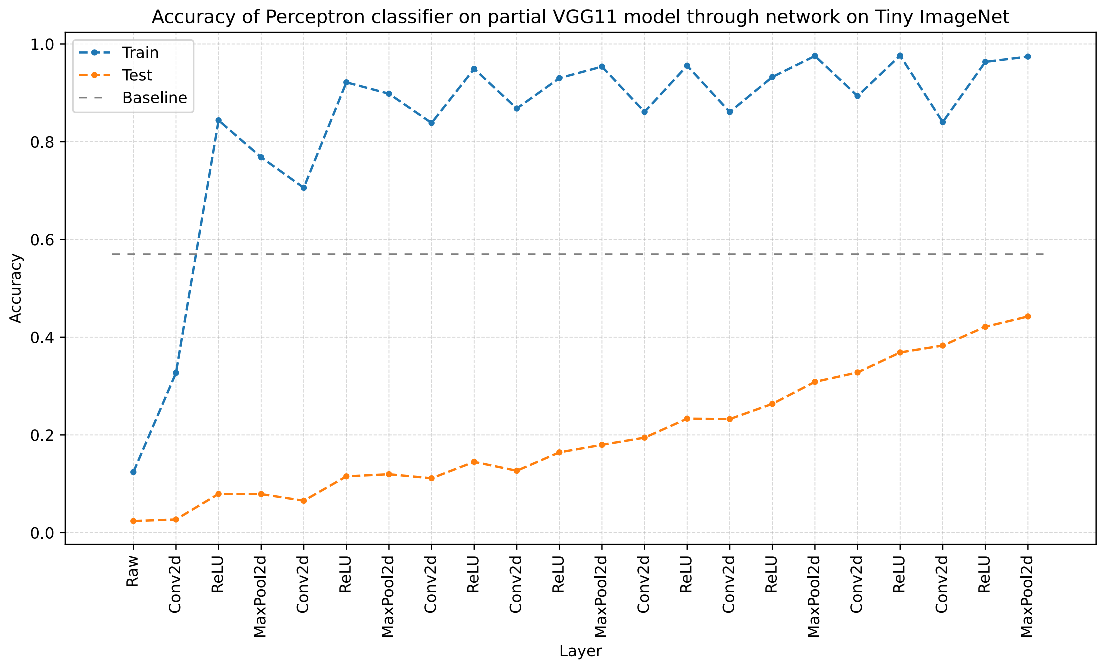


FIGURE B.5: Test & Train accuracy of perceptron probes throughout network: Model=VGG11, Dataset=Tiny Imagenet

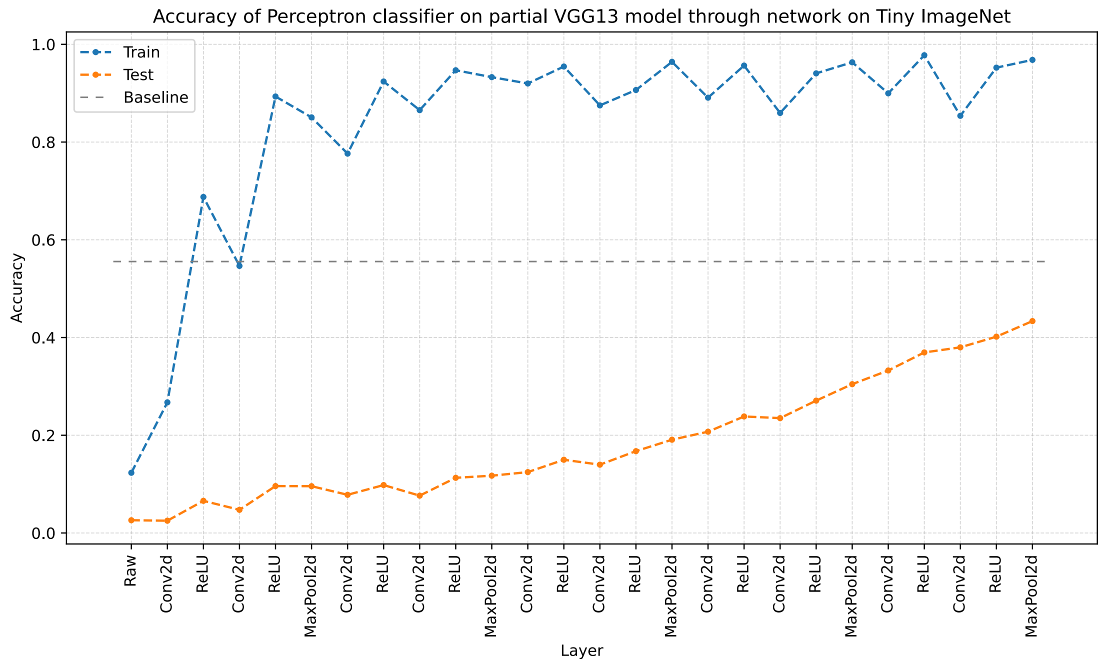


FIGURE B.6: Test & Train accuracy of perceptron probes throughout network: Model=VGG13, Dataset=Tiny Imagenet

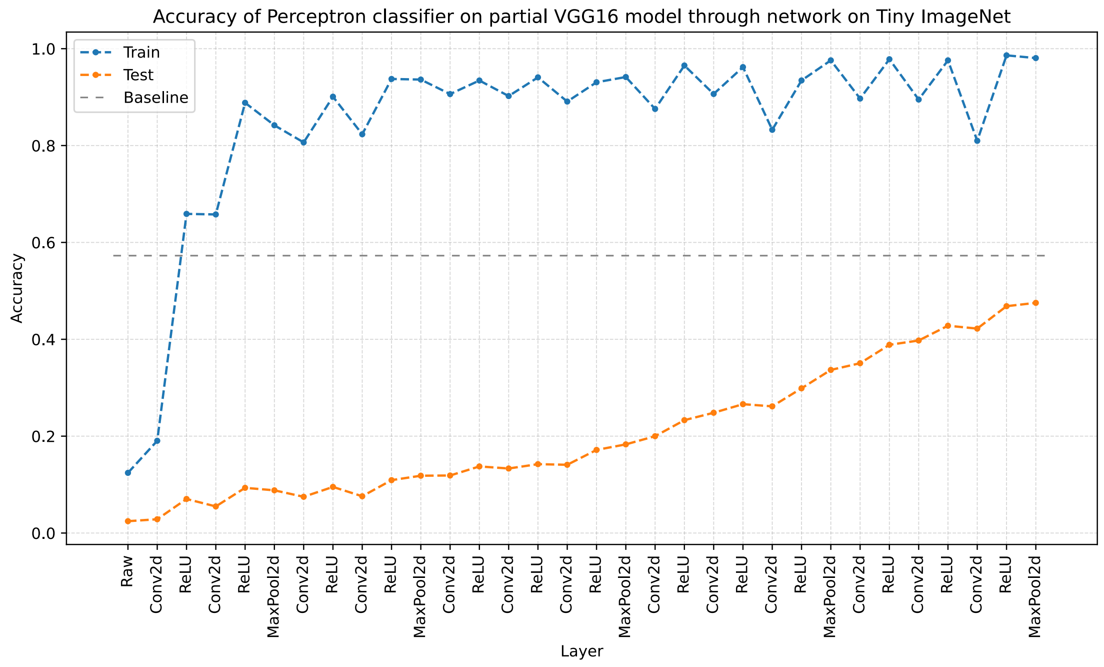


FIGURE B.7: Test & Train accuracy of perceptron probes throughout network: Model=VGG16, Dataset=Tiny Imagenet

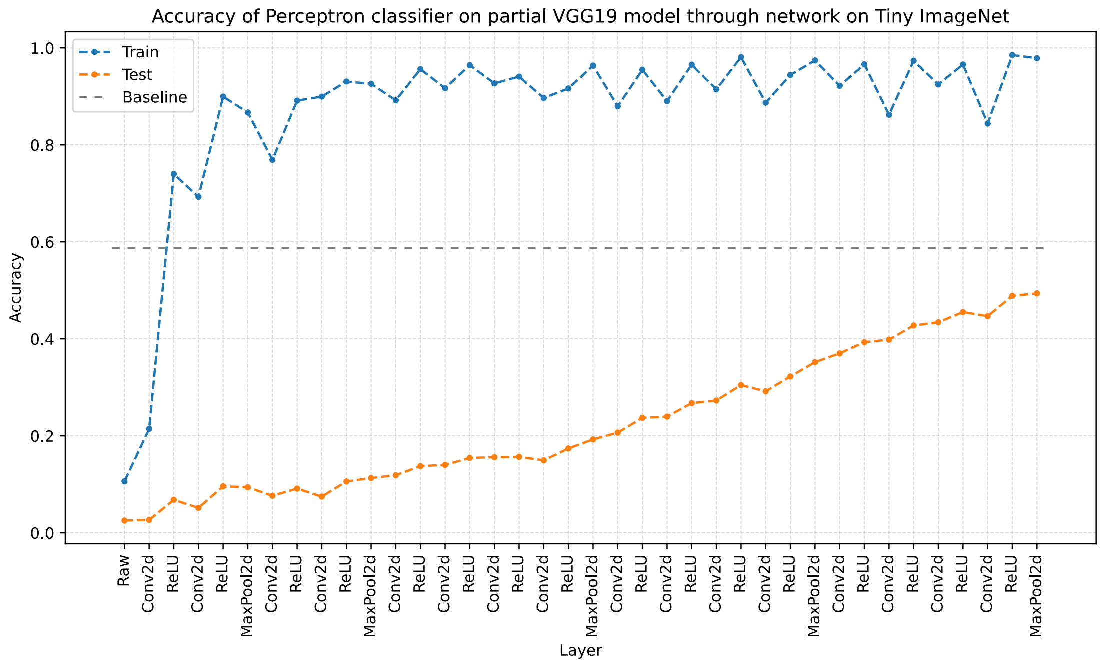


FIGURE B.8: Test & Train accuracy of perceptron probes throughout network: Model=VGG19, Dataset=Tiny Imagenet

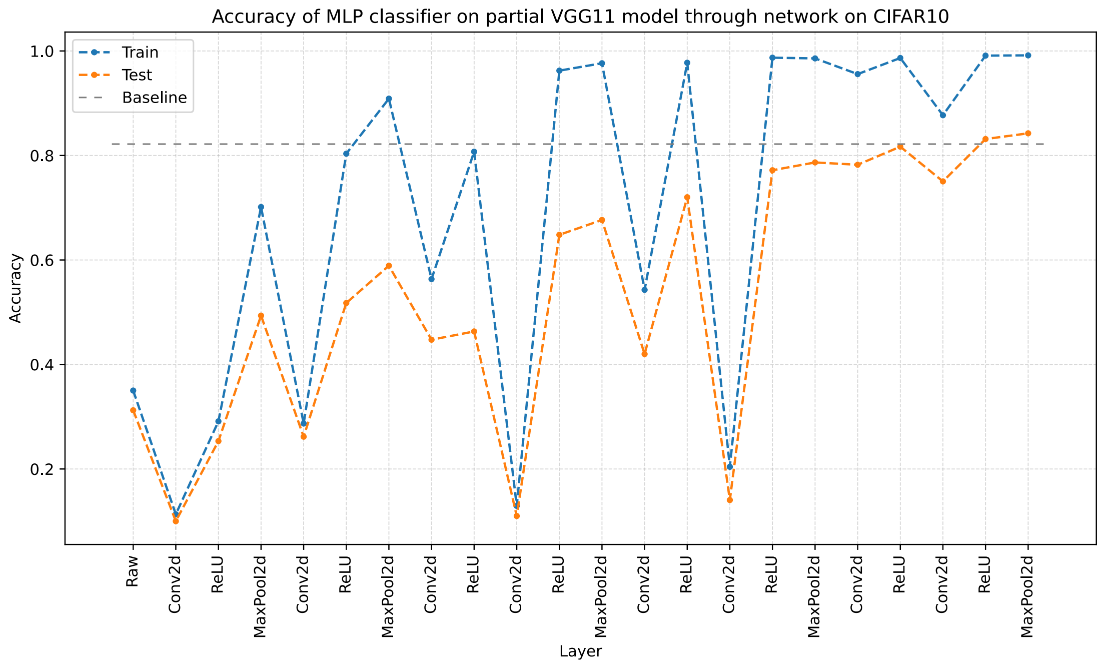


FIGURE B.9: Test & Train accuracy of MLP probes throughout network: Model=VGG11, Dataset=CIFAR10

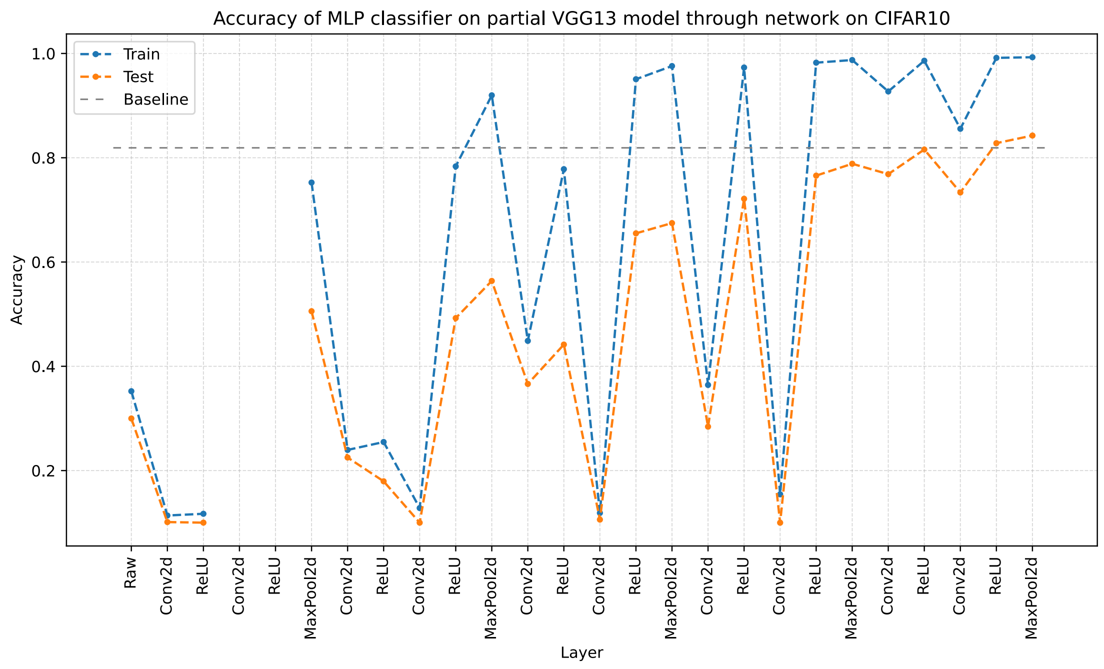


FIGURE B.10: Test & Train accuracy of MLP probes throughout network: Model=VGG13, Dataset=CIFAR10

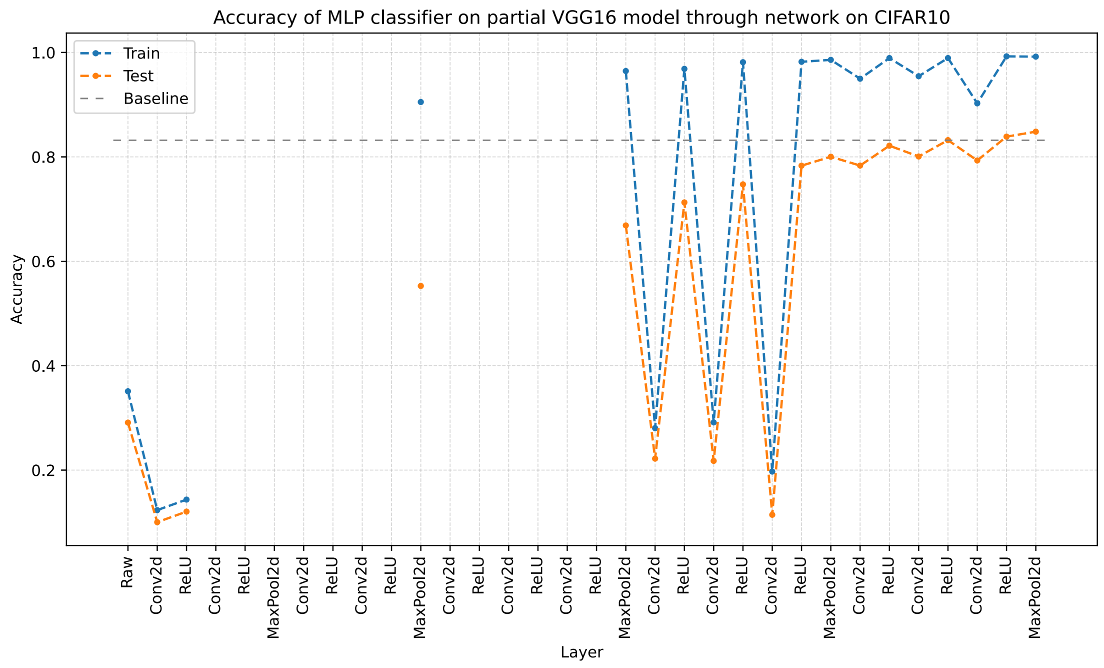


FIGURE B.11: Test & Train accuracy of MLP probes throughout network: Model=VGG16, Dataset=CIFAR10

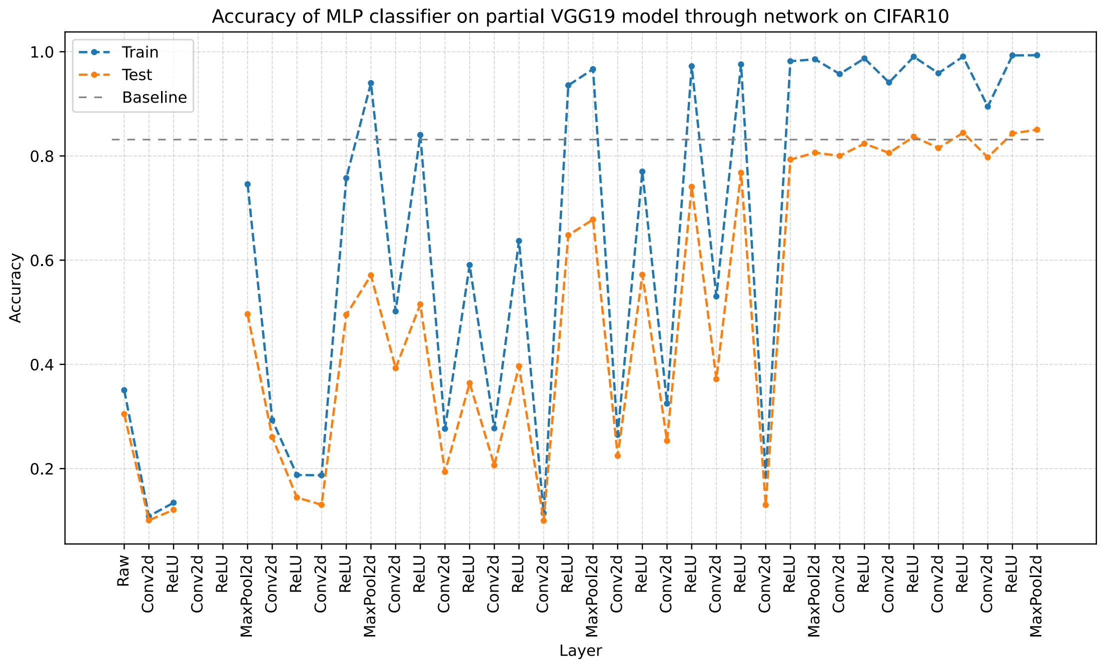


FIGURE B.12: Test & Train accuracy of MLP probes throughout network: Model=VGG19, Dataset=CIFAR10

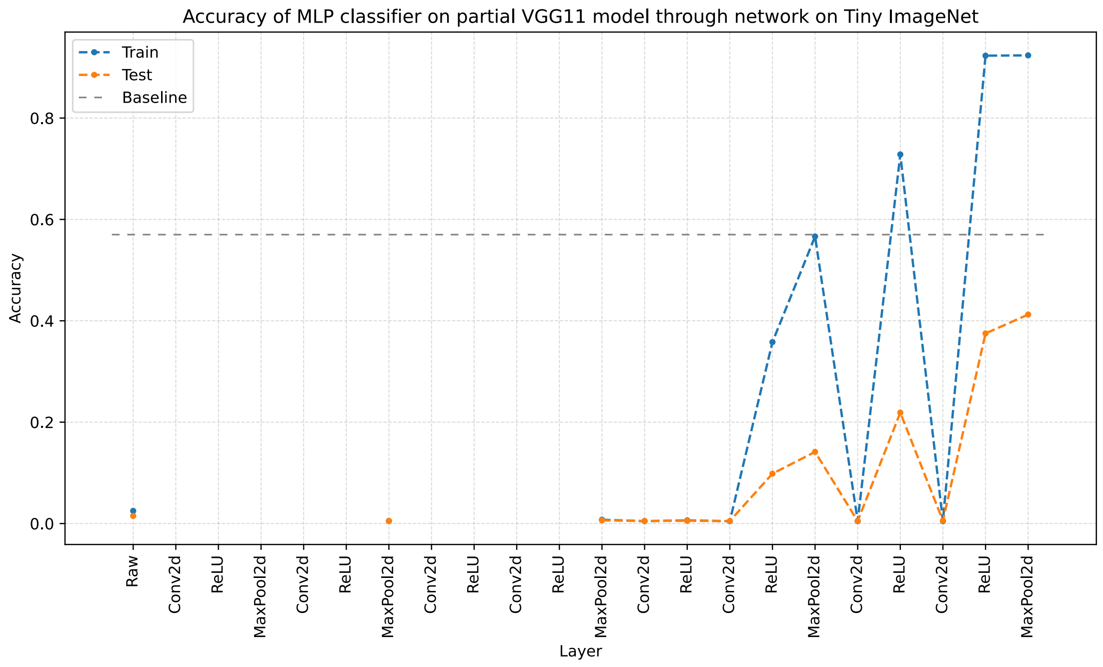


FIGURE B.13: Test & Train accuracy of MLP probes throughout network: Model=VGG11, Dataset=Tiny Imagenet

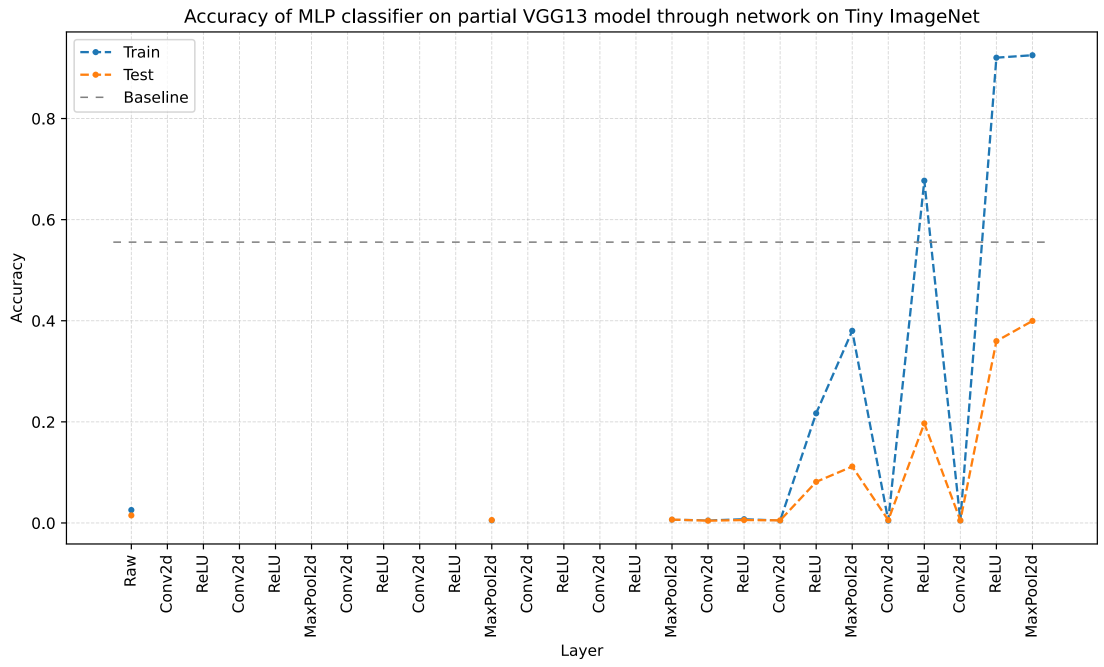


FIGURE B.14: Test & Train accuracy of MLP probes throughout network: Model=VGG13, Dataset=Tiny Imagenet

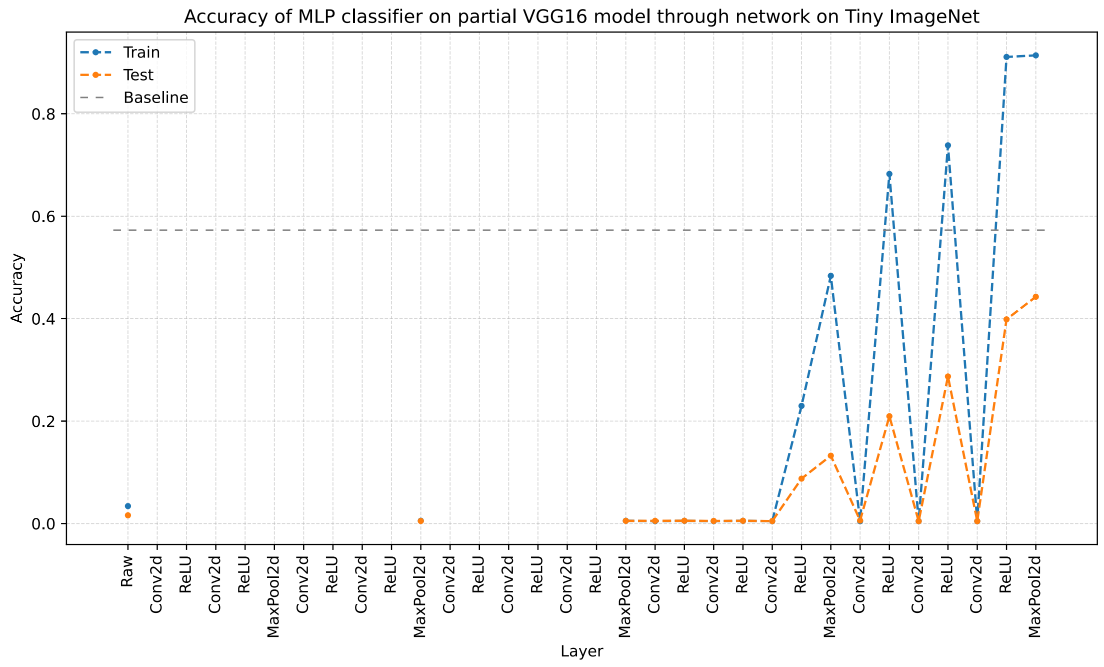


FIGURE B.15: Test & Train accuracy of MLP probes throughout network: Model=VGG16, Dataset=Tiny Imagenet

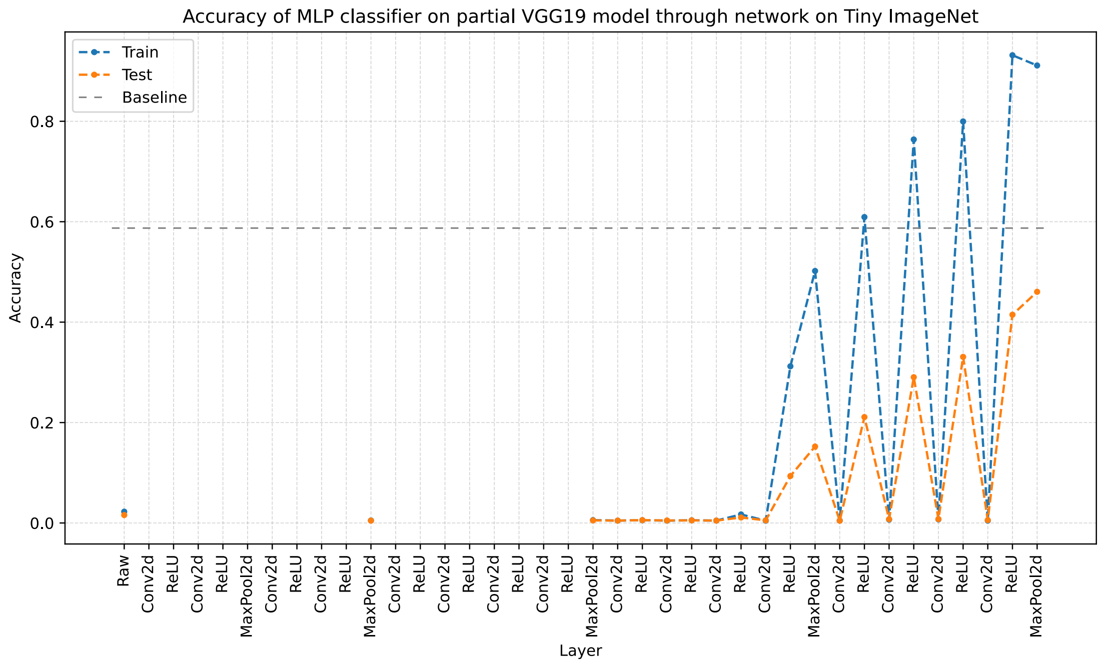


FIGURE B.16: Test & Train accuracy of MLP probes throughout network: Model=VGG19, Dataset=Tiny Imagenet

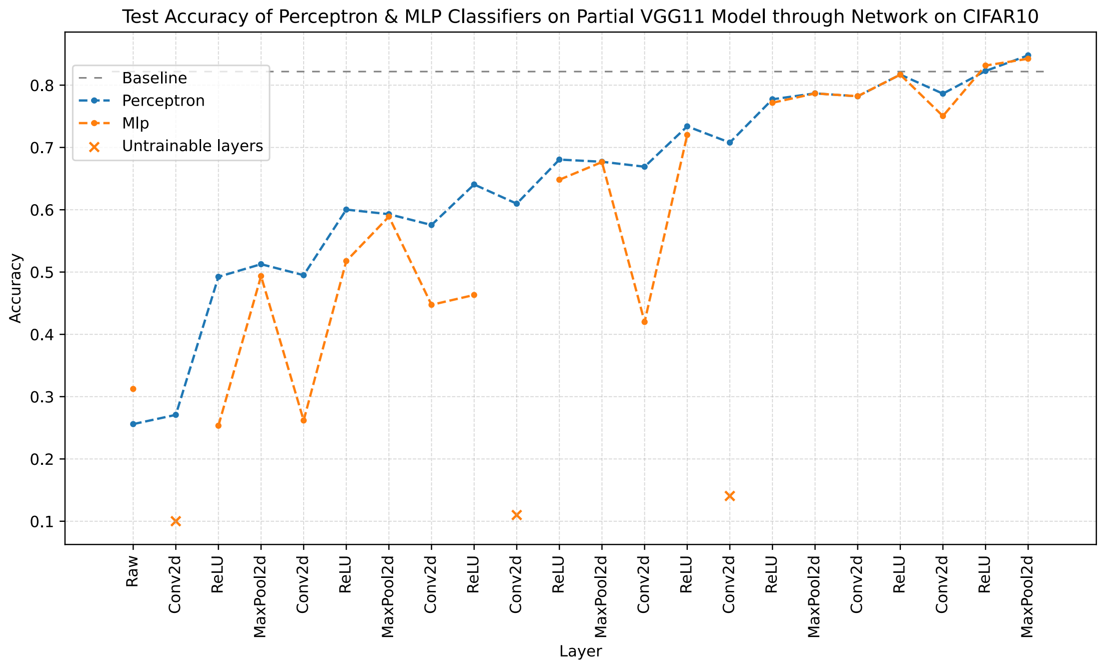


FIGURE B.17: Test accuracy of Perceptron & MLP probes throughout network: Model=VGG11, Dataset=CIFAR10

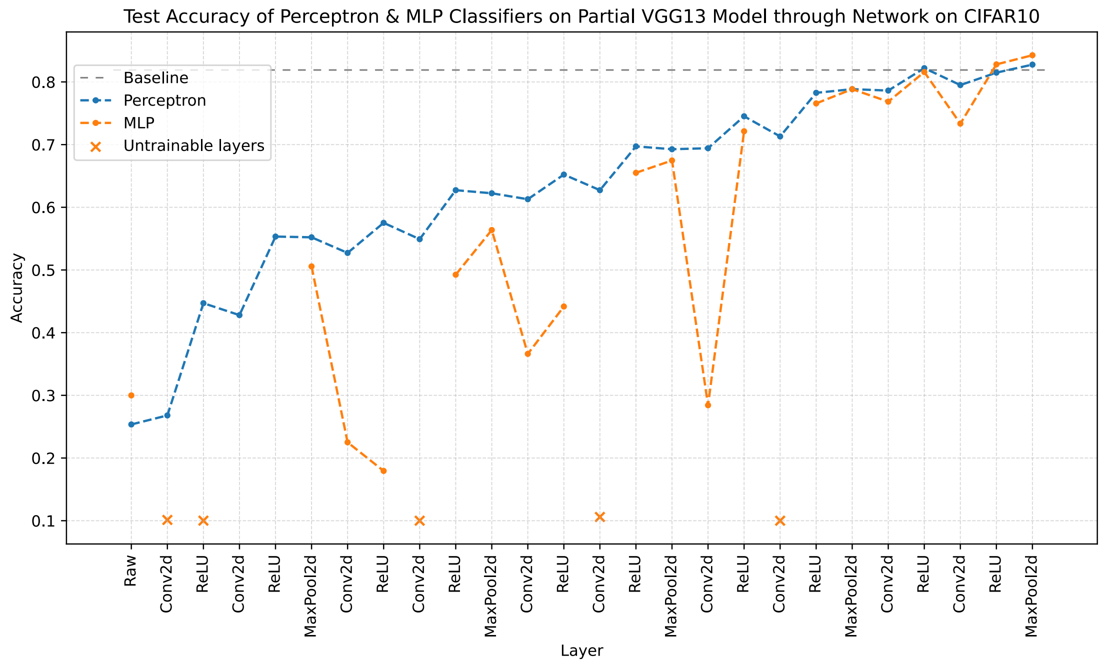


FIGURE B.18: Test accuracy of Perceptron & MLP probes throughout network: Model=VGG13, Dataset=CIFAR10

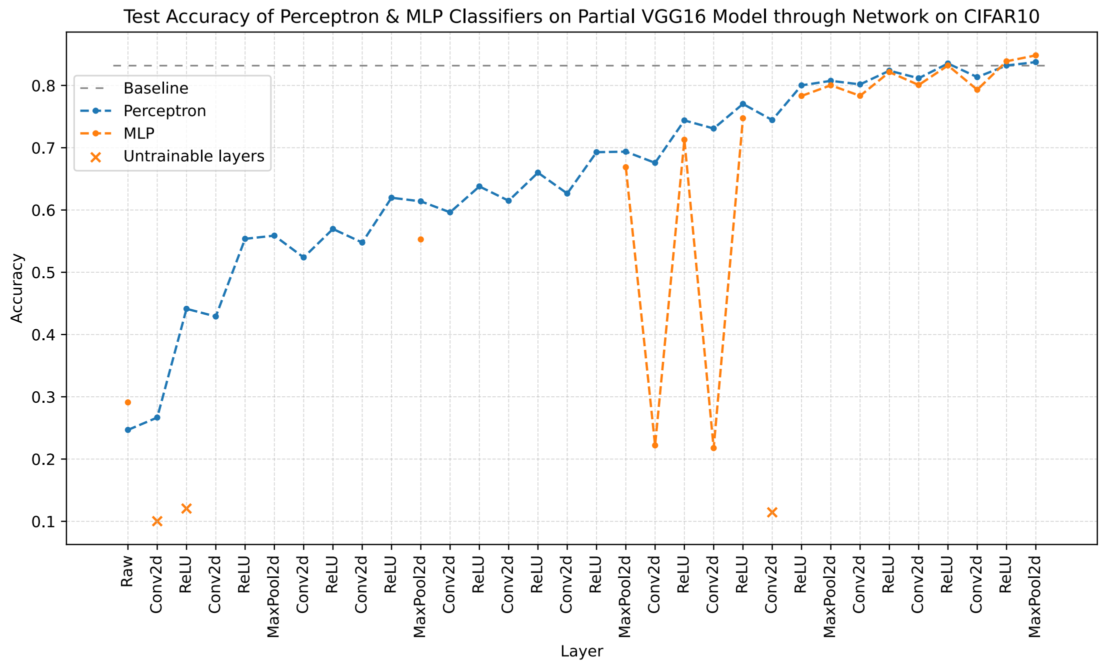


FIGURE B.19: Test accuracy of Perceptron & MLP probes throughout network: Model=VGG16, Dataset=CIFAR10

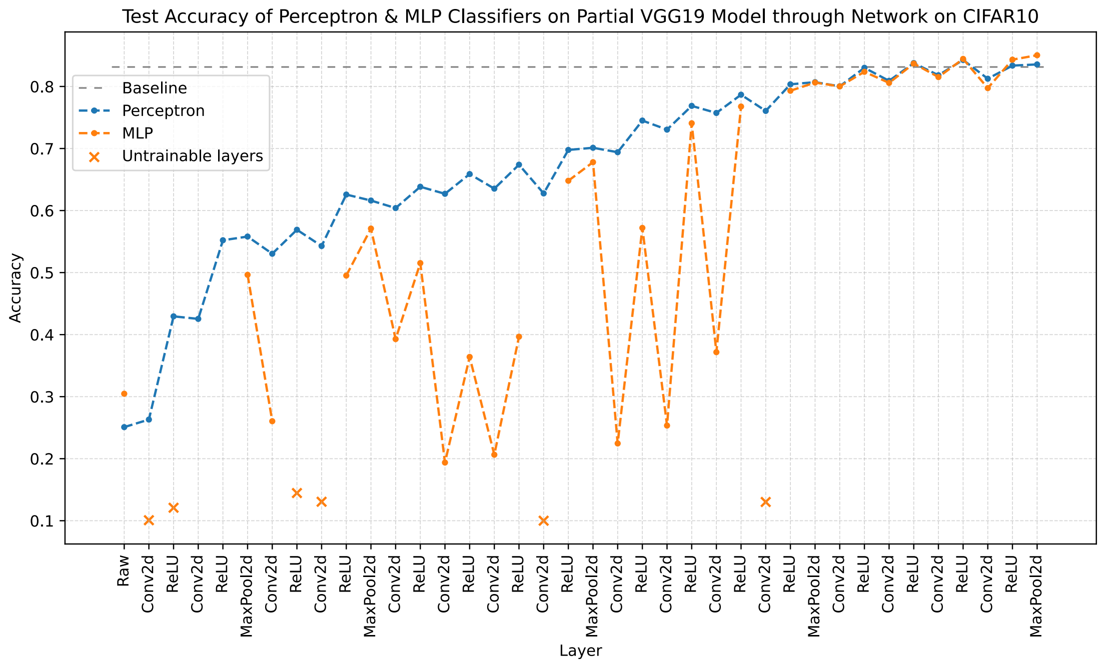


FIGURE B.20: Test accuracy of Perceptron & MLP probes throughout network: Model=VGG19, Dataset=CIFAR10

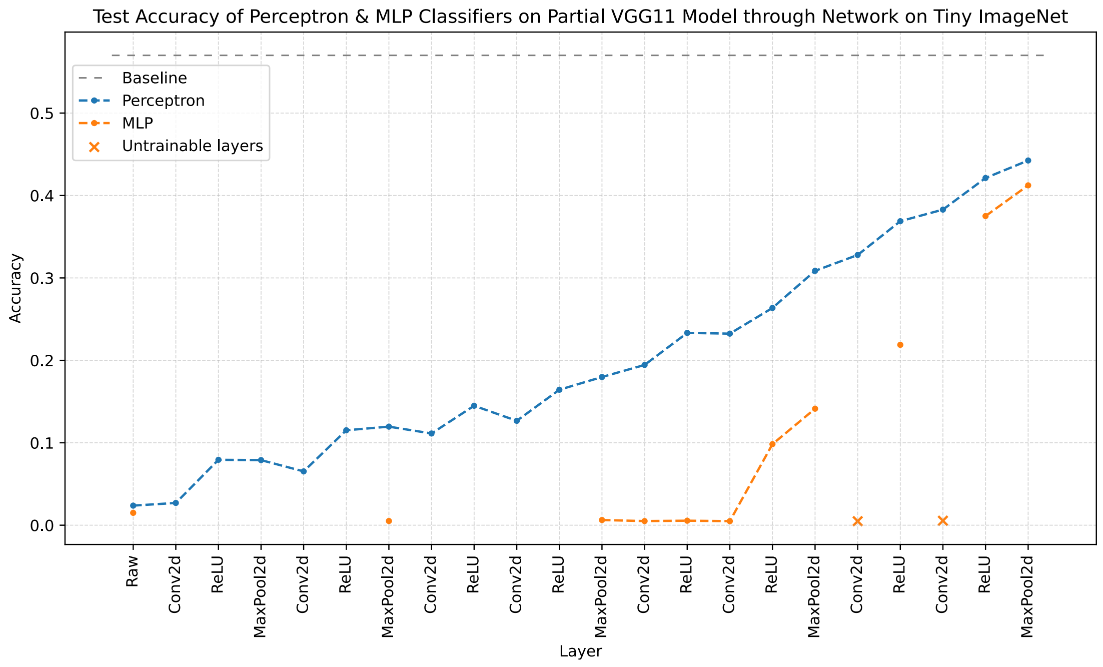


FIGURE B.21: Test accuracy of Perceptron & MLP probes throughout network: Model=VGG11, Dataset=Tiny Imagenet

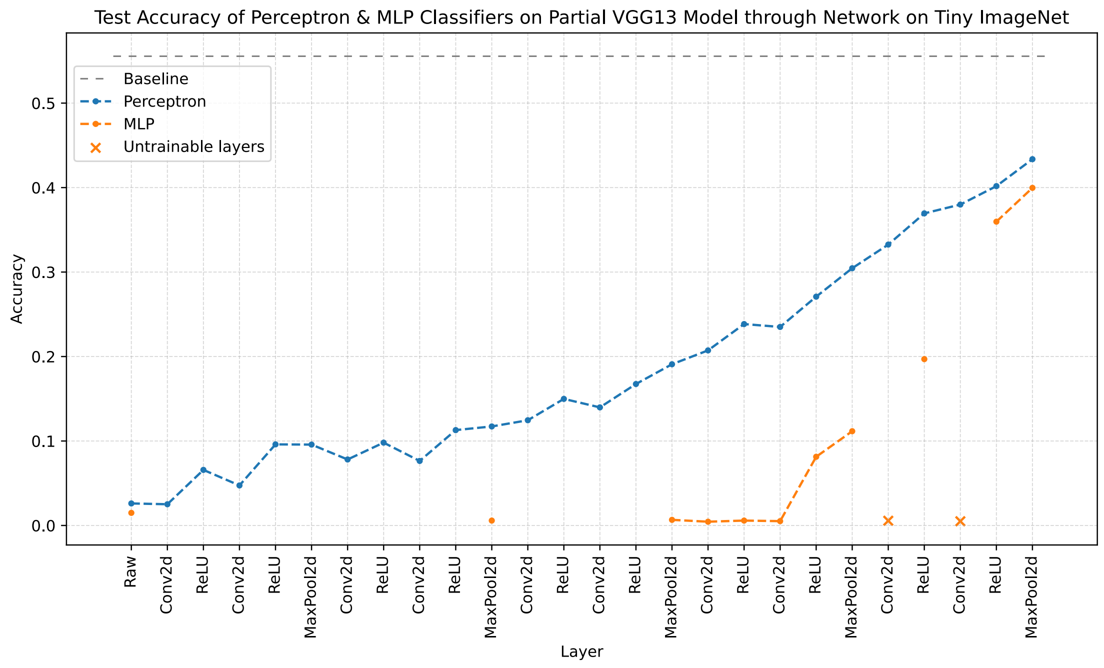


FIGURE B.22: Test accuracy of Perceptron & MLP probes throughout network: Model=VGG13, Dataset=Tiny Imagenet

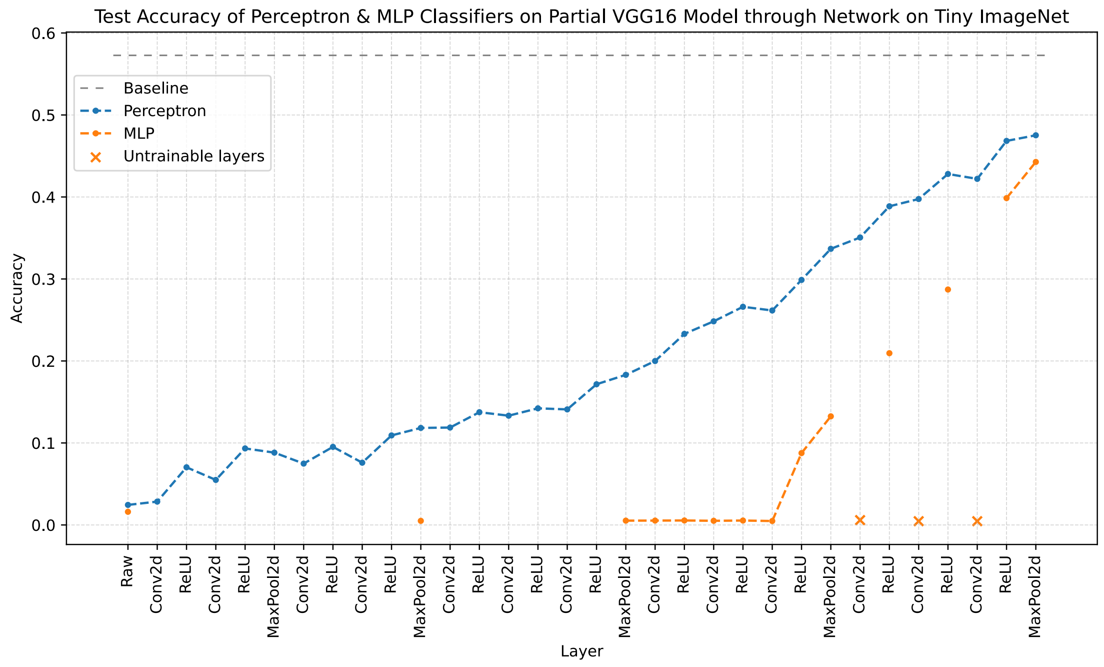


FIGURE B.23: Test accuracy of Perceptron & MLP probes throughout network: Model=VGG16, Dataset=Tiny Imagenet

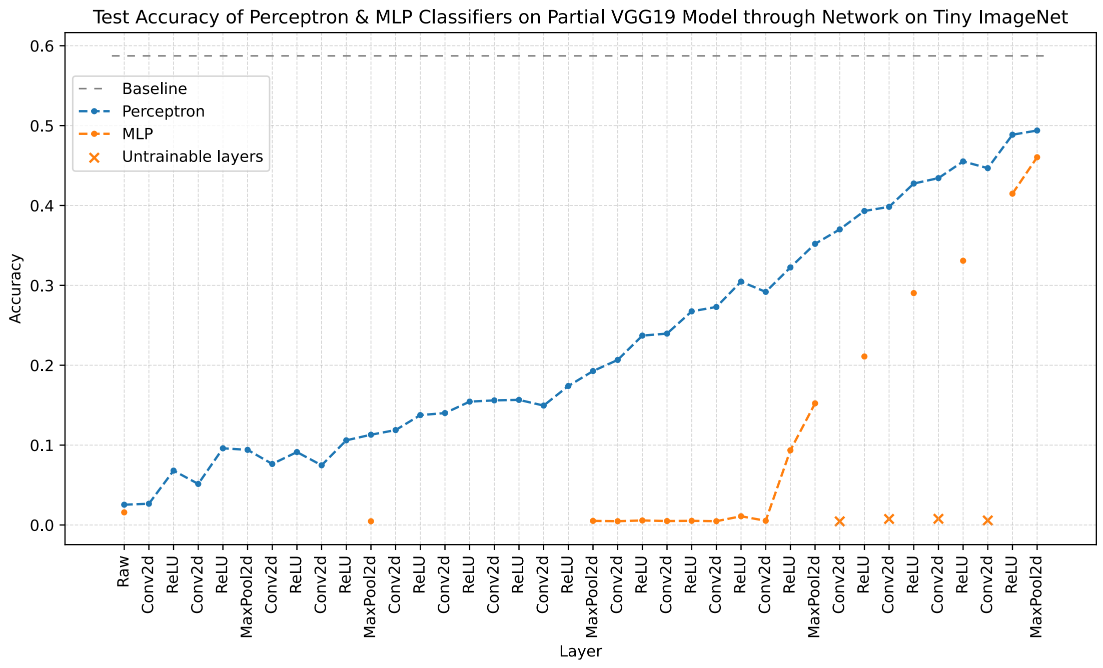


FIGURE B.24: Test accuracy of Perceptron & MLP probes throughout network: Model=VGG19, Dataset=Tiny Imagenet

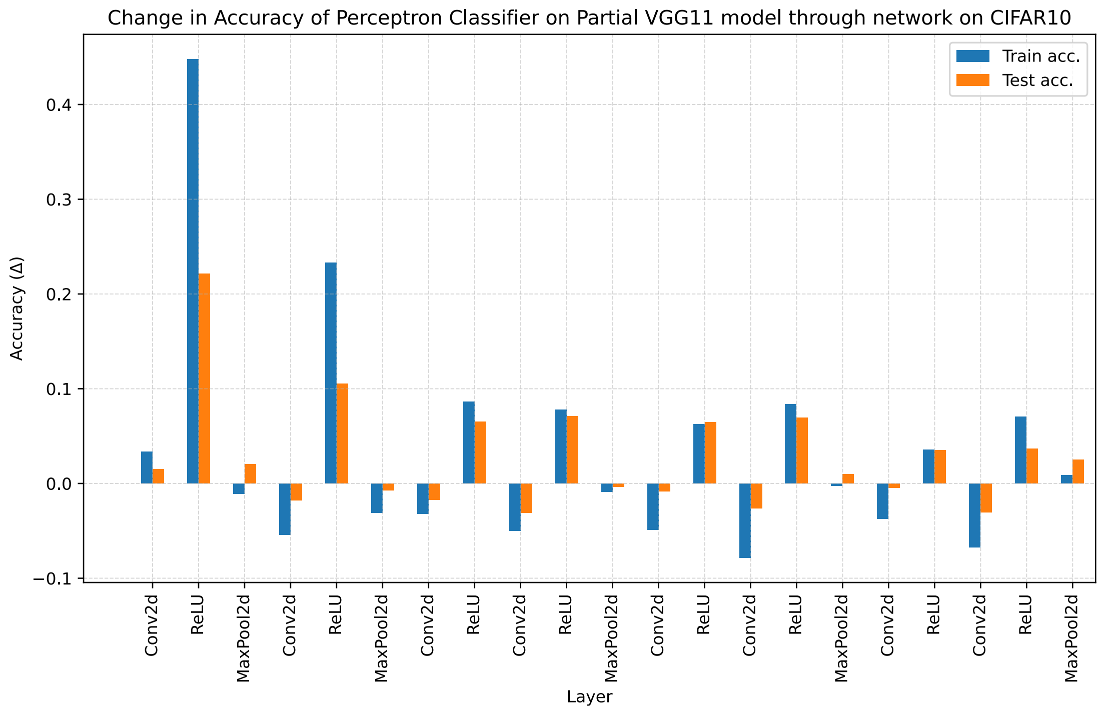


FIGURE B.25: Change in train and test accuracy of Perceptron probes throughout network: Model=VGG11, Dataset=CIFAR10

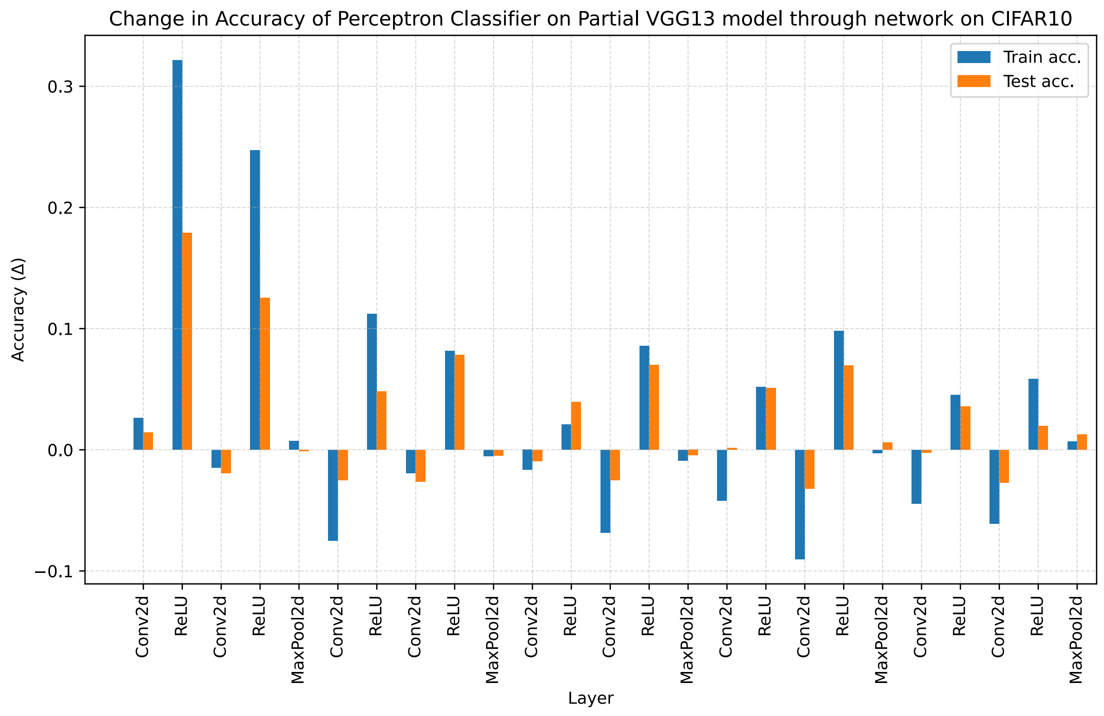


FIGURE B.26: Change in train and test accuracy of Perceptron probes throughout network: Model=VGG13, Dataset=CIFAR10

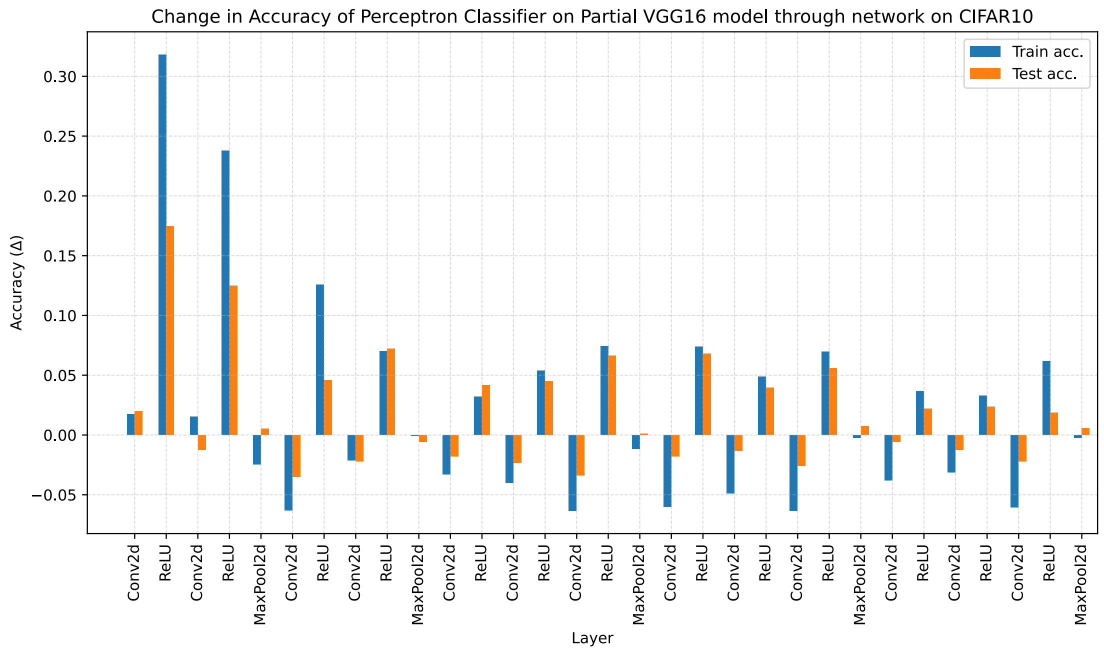


FIGURE B.27: Change in train and test accuracy of Perceptron probes throughout network: Model=VGG16, Dataset=CIFAR10

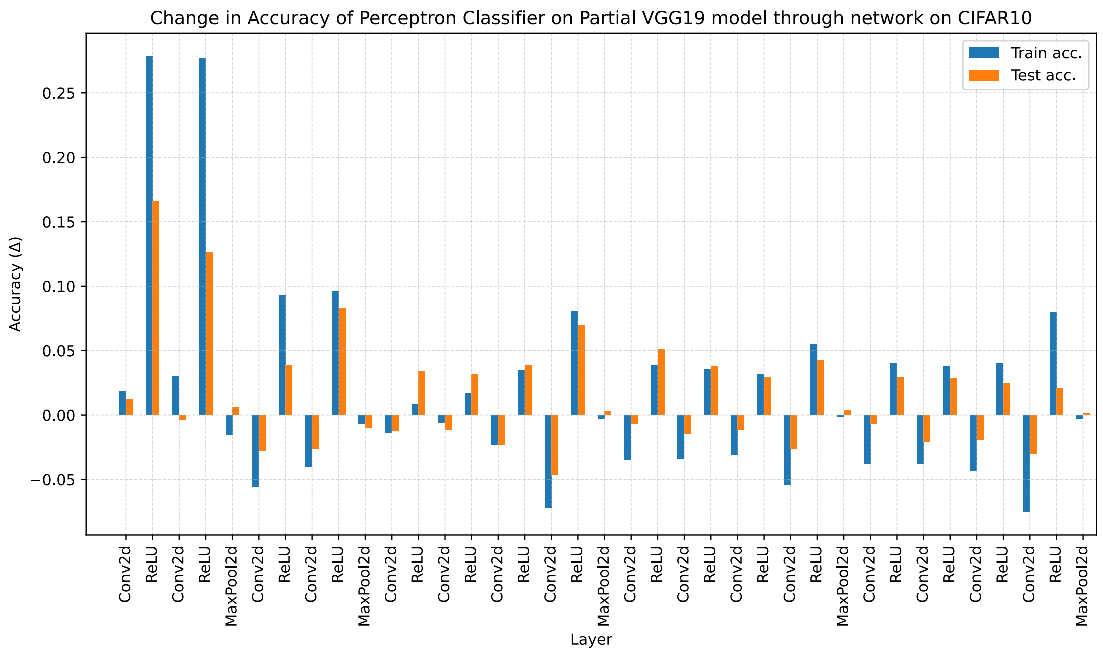


FIGURE B.28: Change in train and test accuracy of Perceptron probes throughout network: Model=VGG19, Dataset=CIFAR10

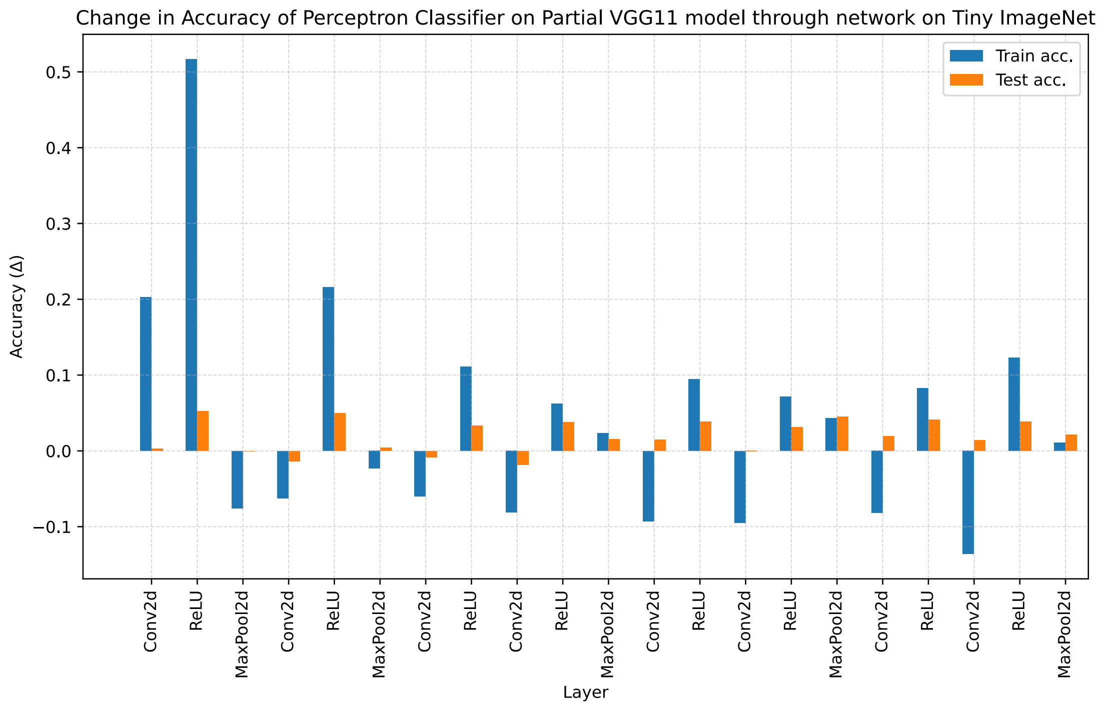


FIGURE B.29: Change in train and test accuracy of Perceptron probes throughout network: Model=VGG11, Dataset=Tiny Imagenet

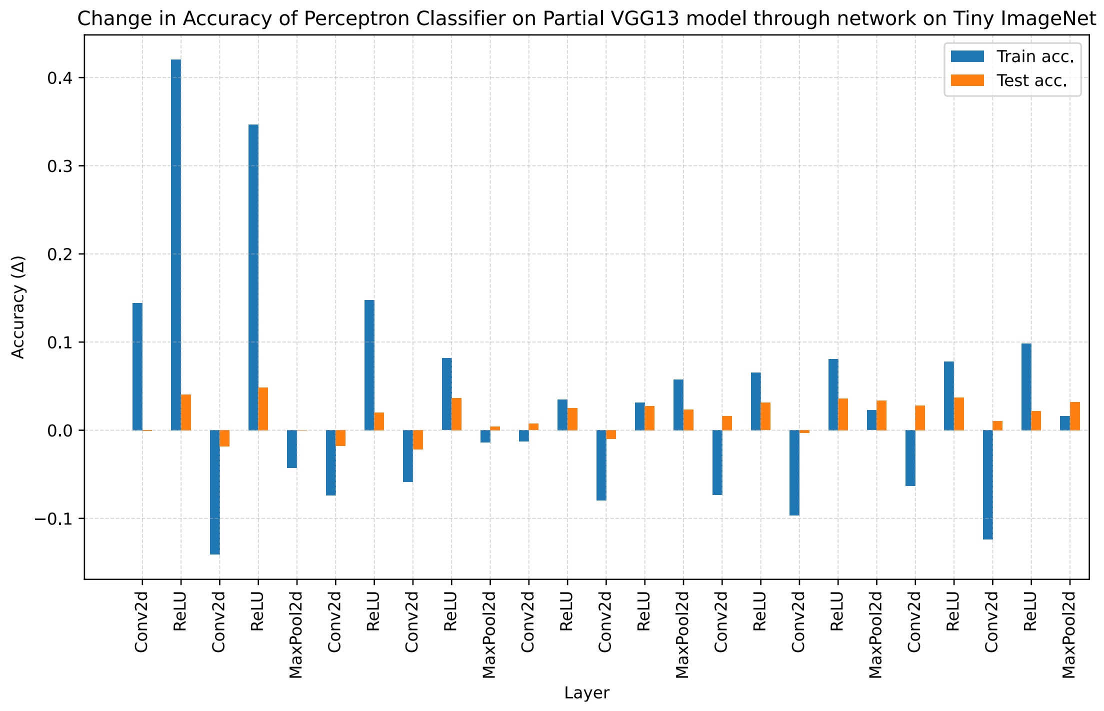


FIGURE B.30: Change in train and test accuracy of Perceptron probes throughout network: Model=VGG13, Dataset=Tiny Imagenet

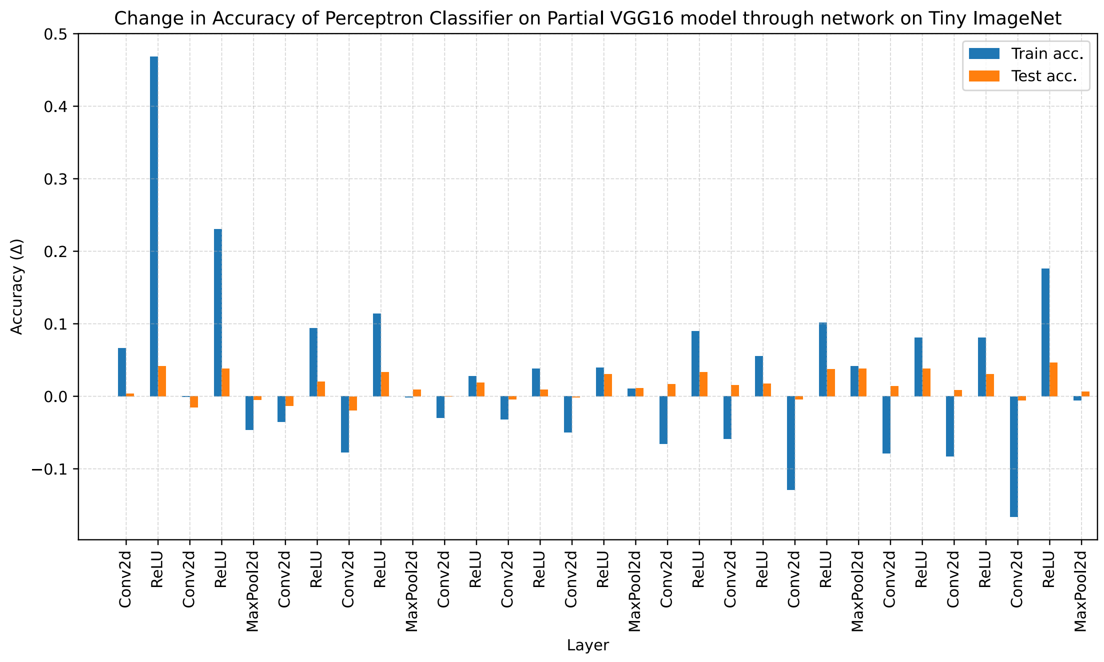


FIGURE B.31: Change in train and test accuracy of Perceptron probes throughout network: Model=VGG16, Dataset=Tiny Imagenet

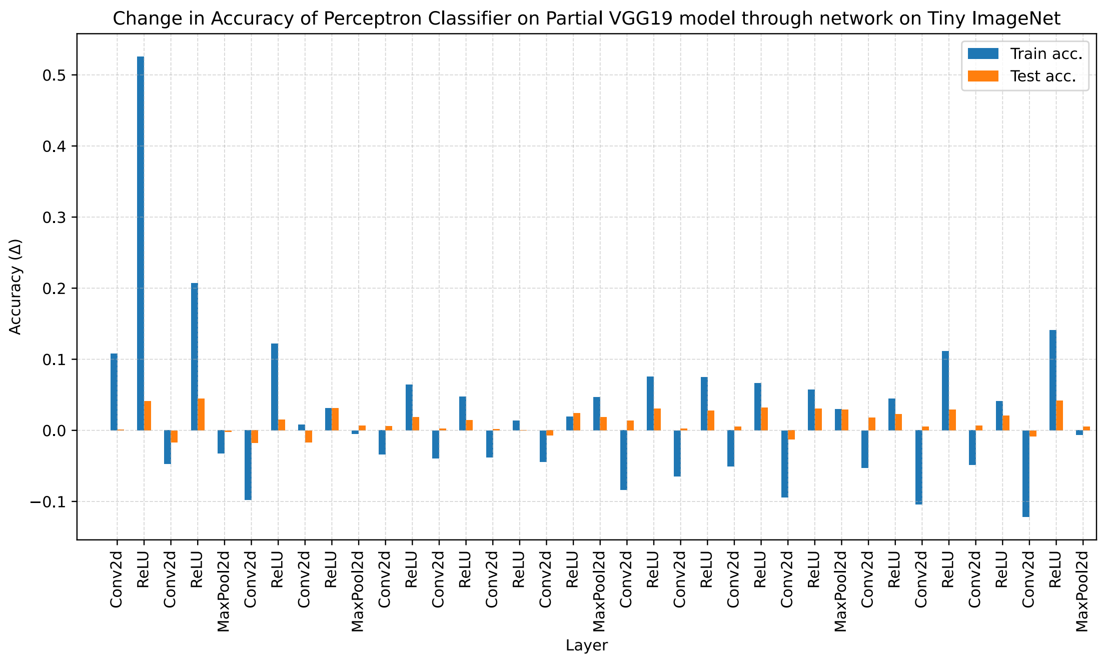


FIGURE B.32: Change in train and test accuracy of Perceptron probes throughout network: Model=VGG19, Dataset=Tiny Imagenet

Appendix C

Complete Results from Chapter 5

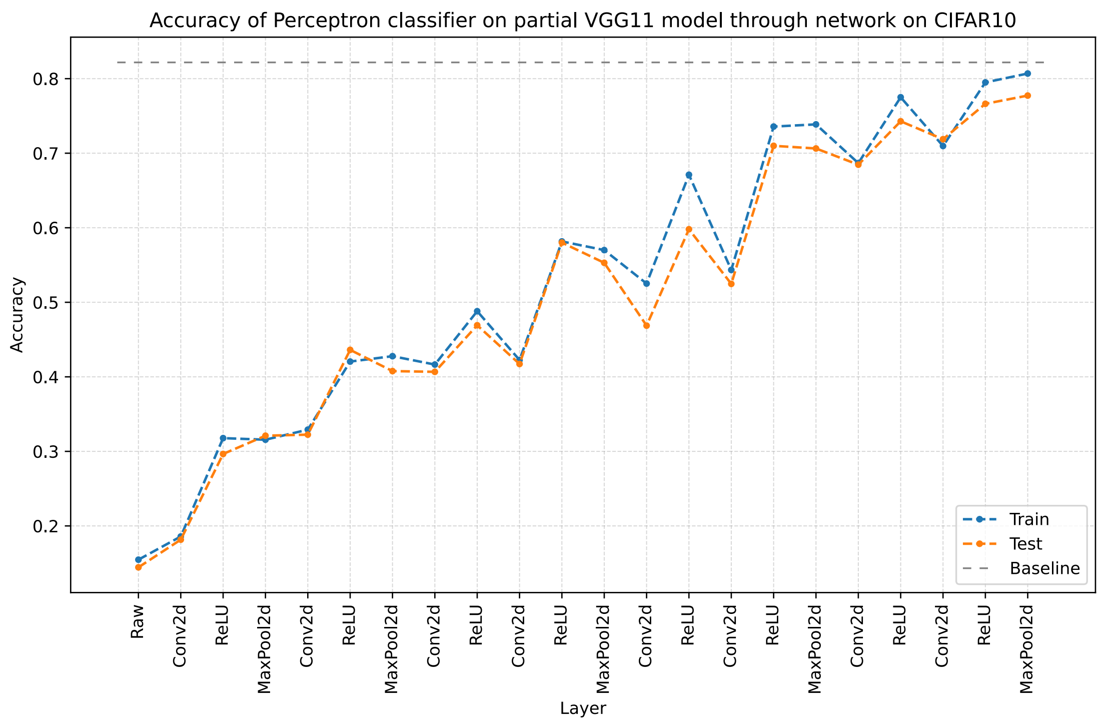


FIGURE C.1: Test & Train accuracy of perceptron probes throughout network with global pooling: Model=VGG11, Dataset=CIFAR10

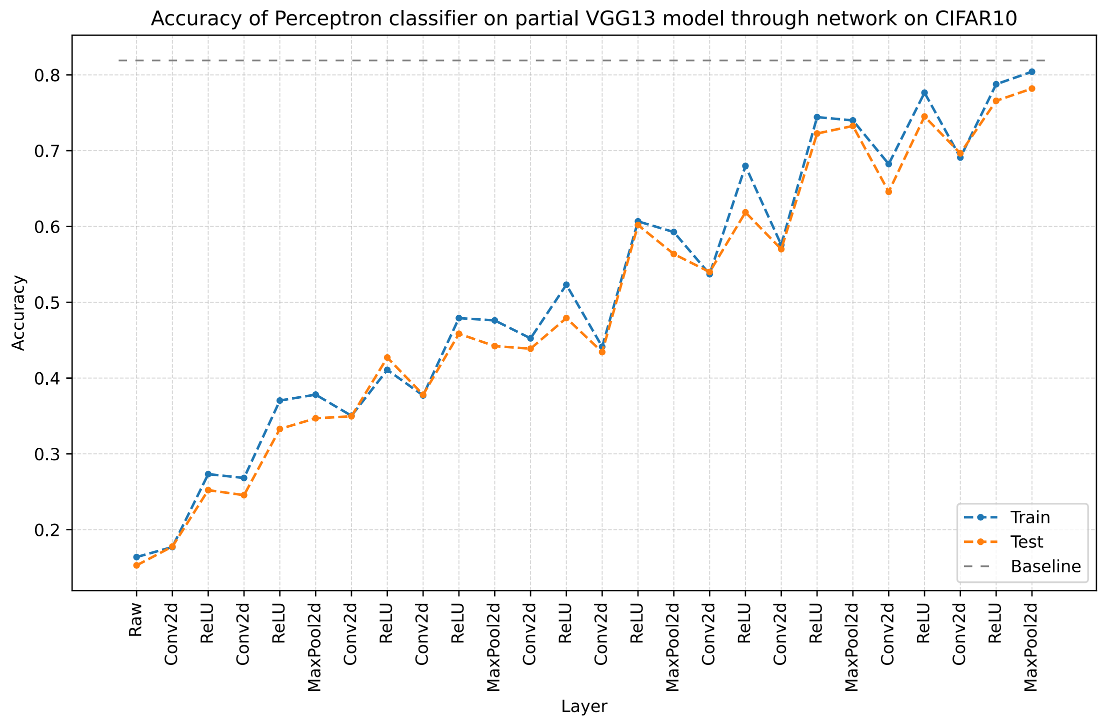


FIGURE C.2: Test & Train accuracy of perceptron probes throughout network with global pooling: Model=VGG13, Dataset=CIFAR10

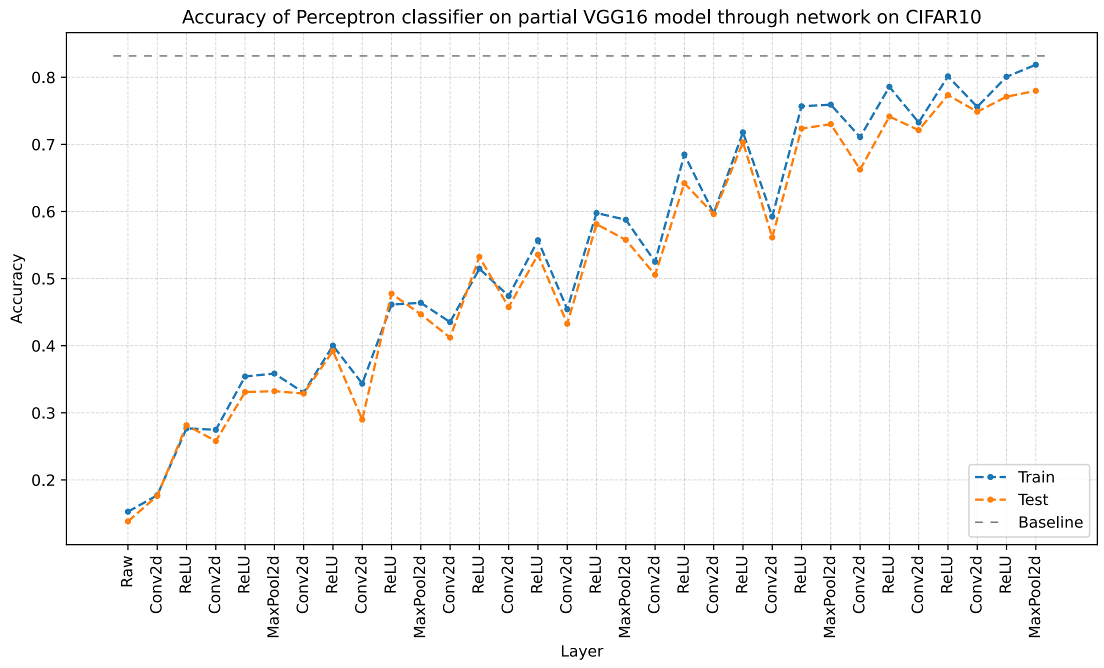


FIGURE C.3: Test & Train accuracy of perceptron probes throughout network with global pooling: Model=VGG16, Dataset=CIFAR10

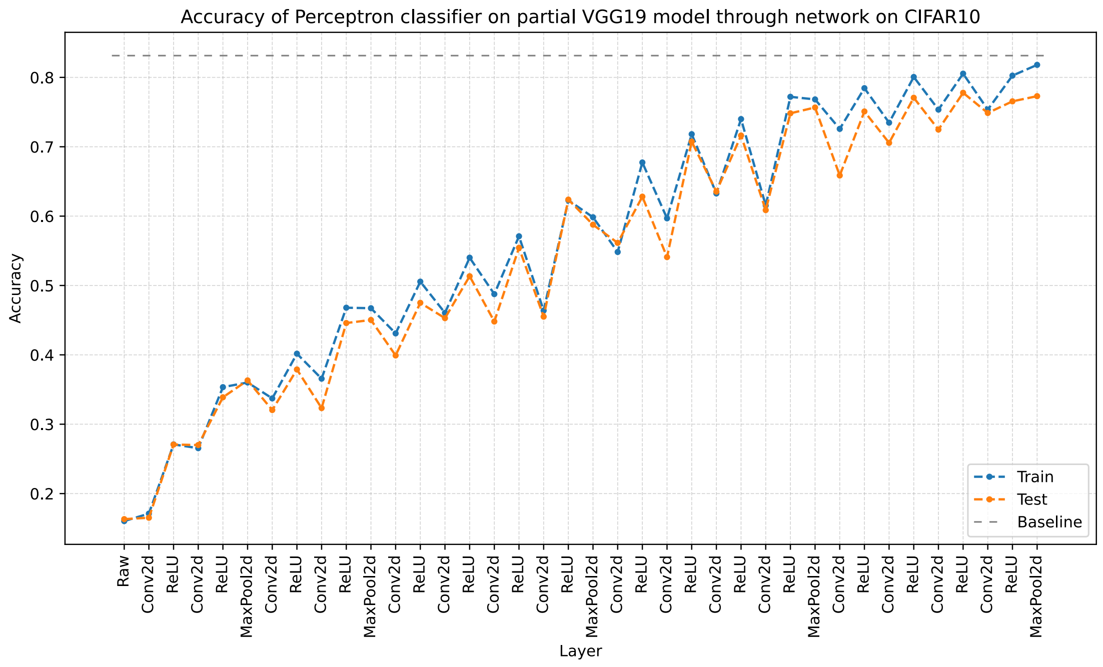


FIGURE C.4: Test & Train accuracy of perceptron probes throughout network with global pooling: Model=VGG19, Dataset=CIFAR10

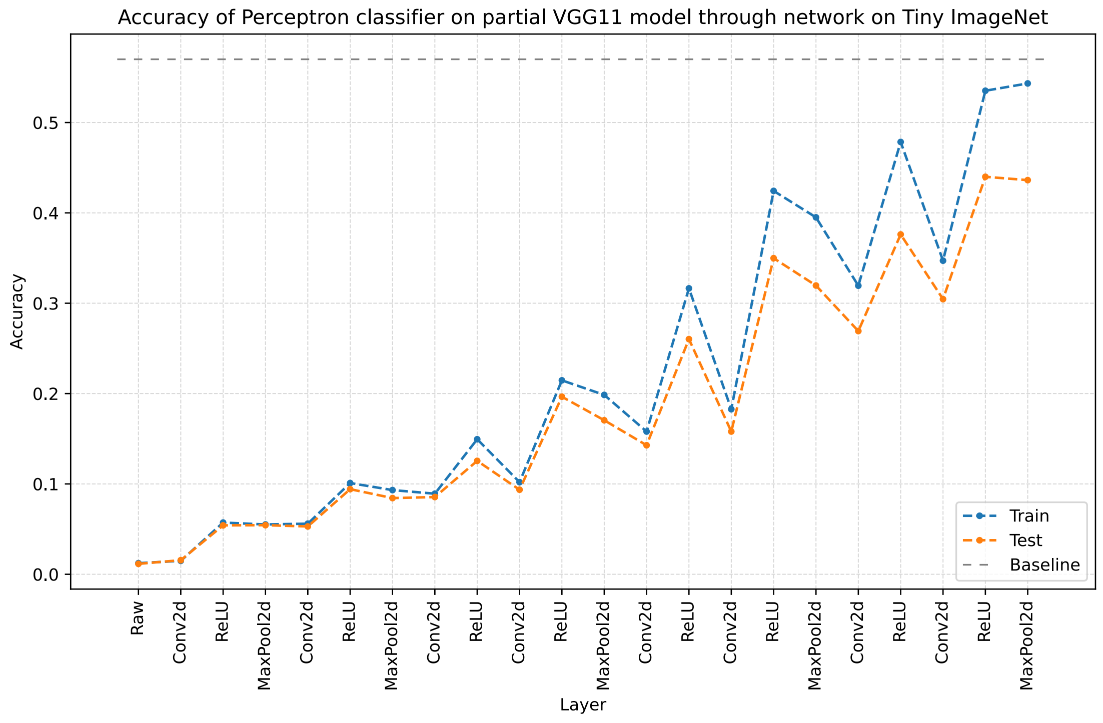


FIGURE C.5: Test & Train accuracy of perceptron probes throughout network with global pooling: Model=VGG11, Dataset=Tiny Imagenet

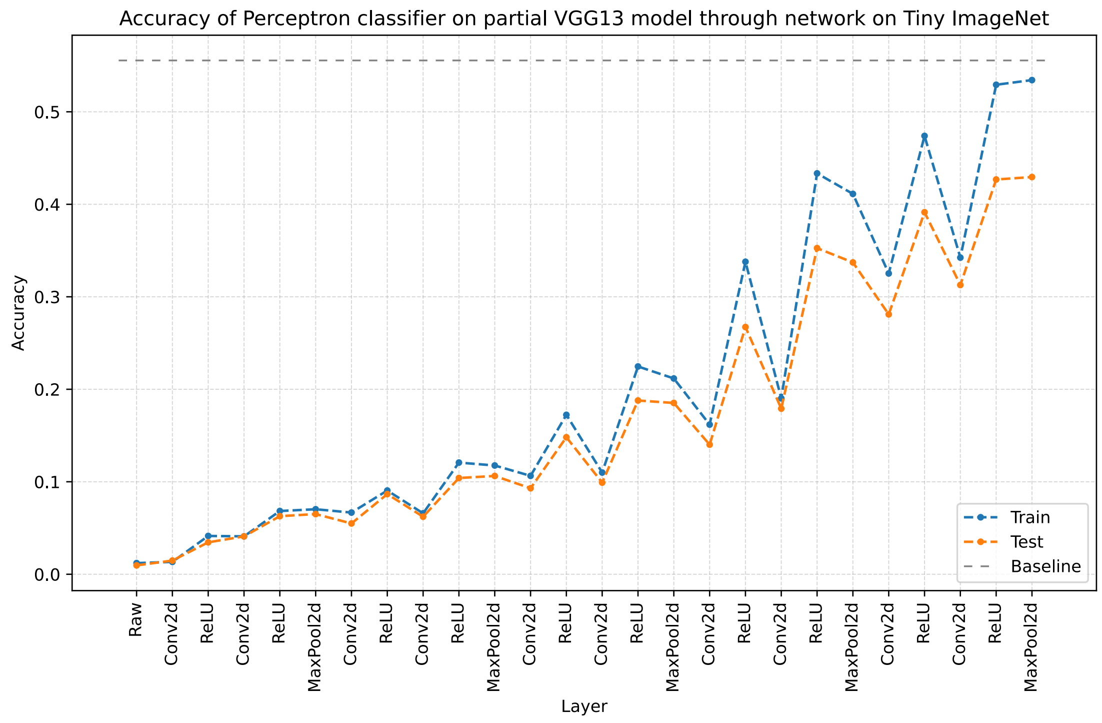


FIGURE C.6: Test & Train accuracy of perceptron probes throughout network with global pooling: Model=VGG13, Dataset=Tiny Imagenet

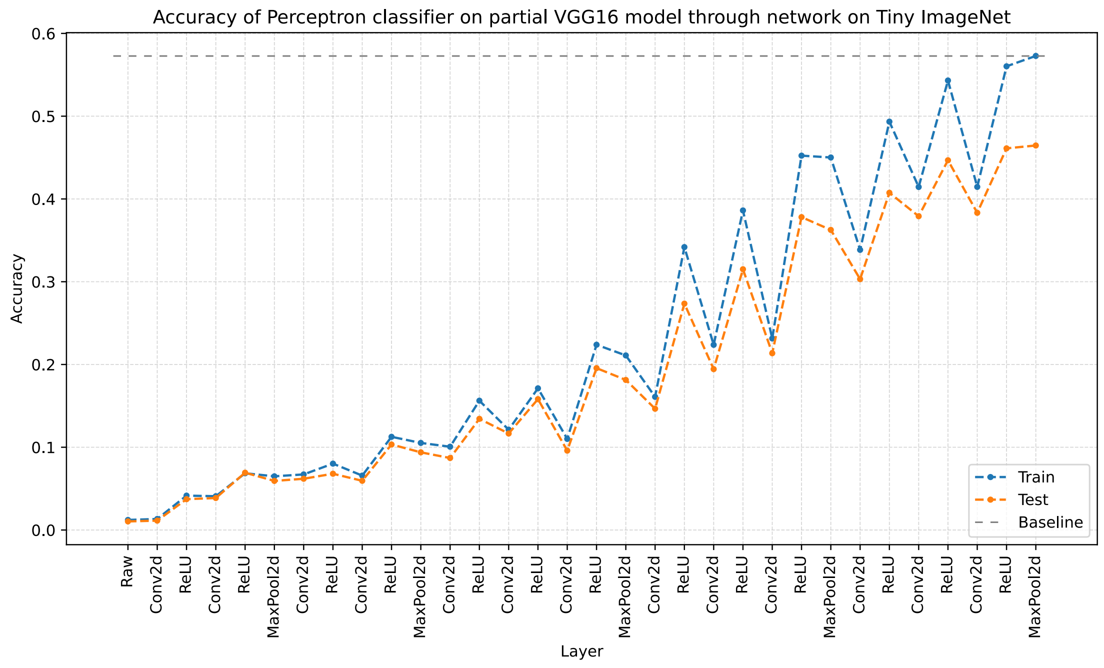


FIGURE C.7: Test & Train accuracy of perceptron probes throughout network with global pooling: Model=VGG16, Dataset=Tiny Imagenet

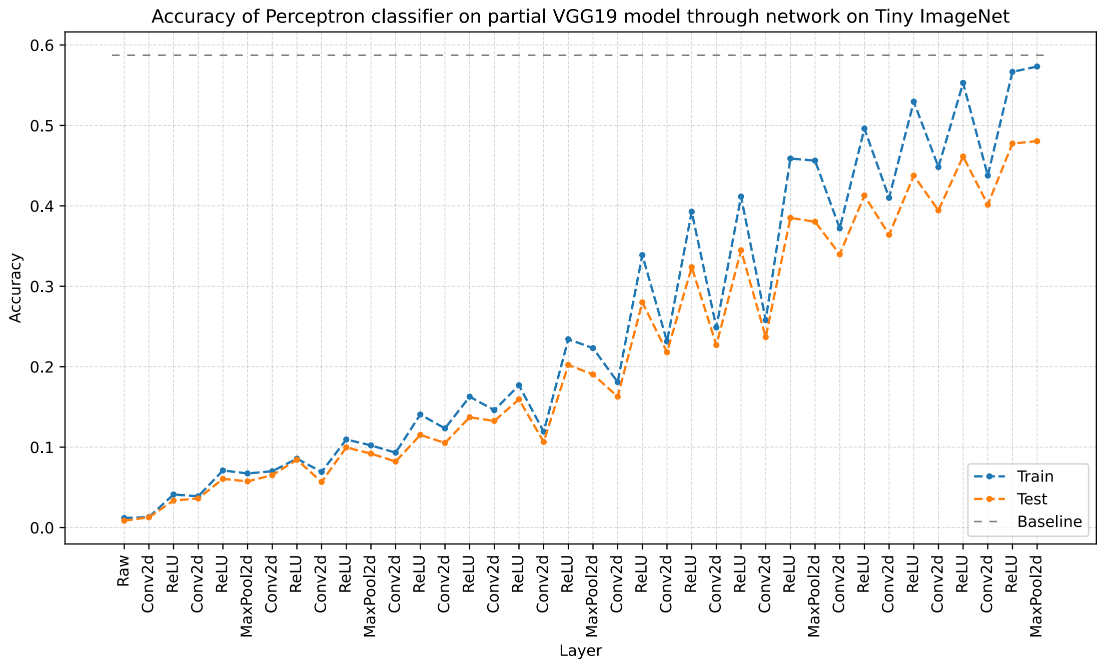


FIGURE C.8: Test & Train accuracy of perceptron probes throughout network with global pooling: Model=VGG19, Dataset=Tiny Imagenet

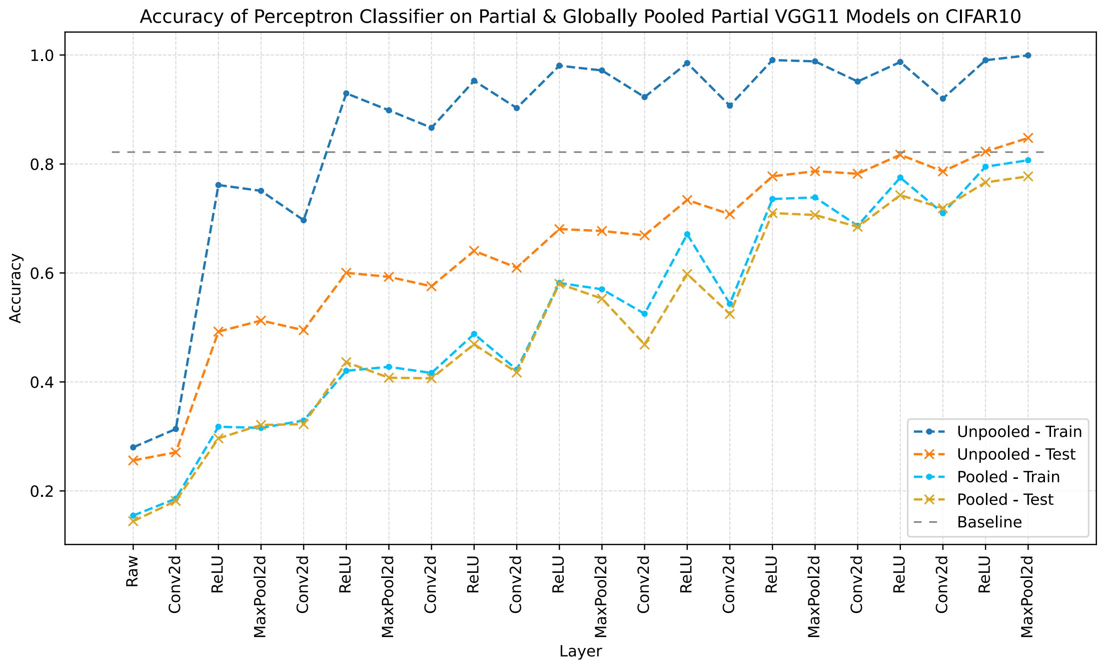


FIGURE C.9: Test & Train accuracy of perceptron probes throughout network with and without global pooling: Model=VGG11, Dataset=CIFAR10

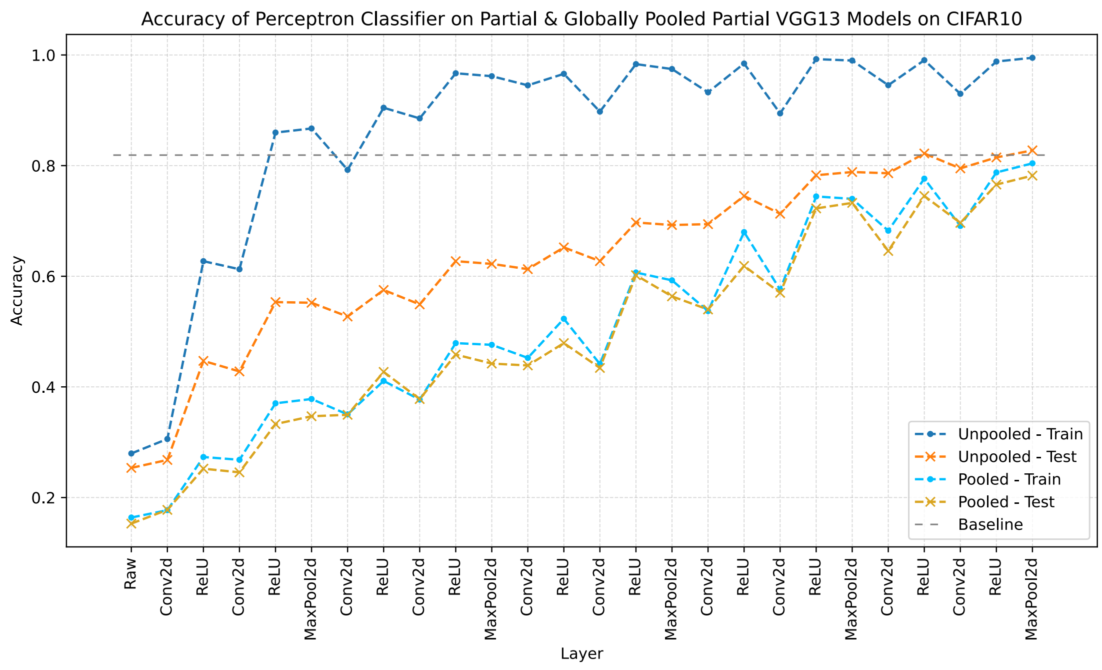


FIGURE C.10: Test & Train accuracy of perceptron probes throughout network with and without global pooling: Model=VGG13, Dataset=CIFAR10

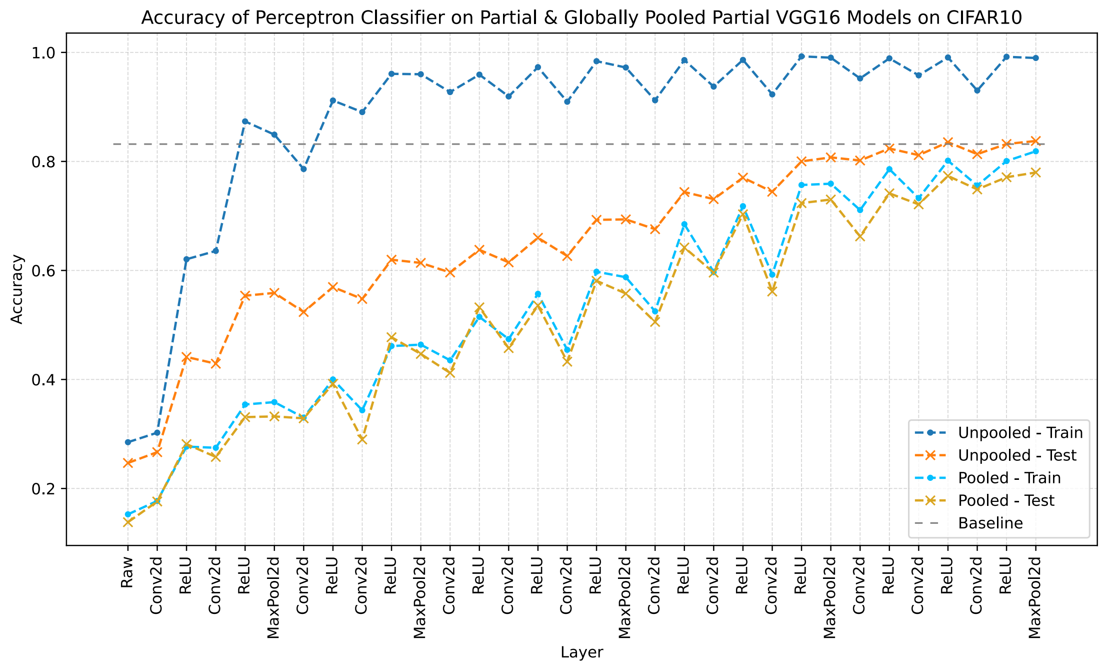


FIGURE C.11: Test & Train accuracy of perceptron probes throughout network with and without global pooling: Model=VGG16, Dataset=CIFAR10

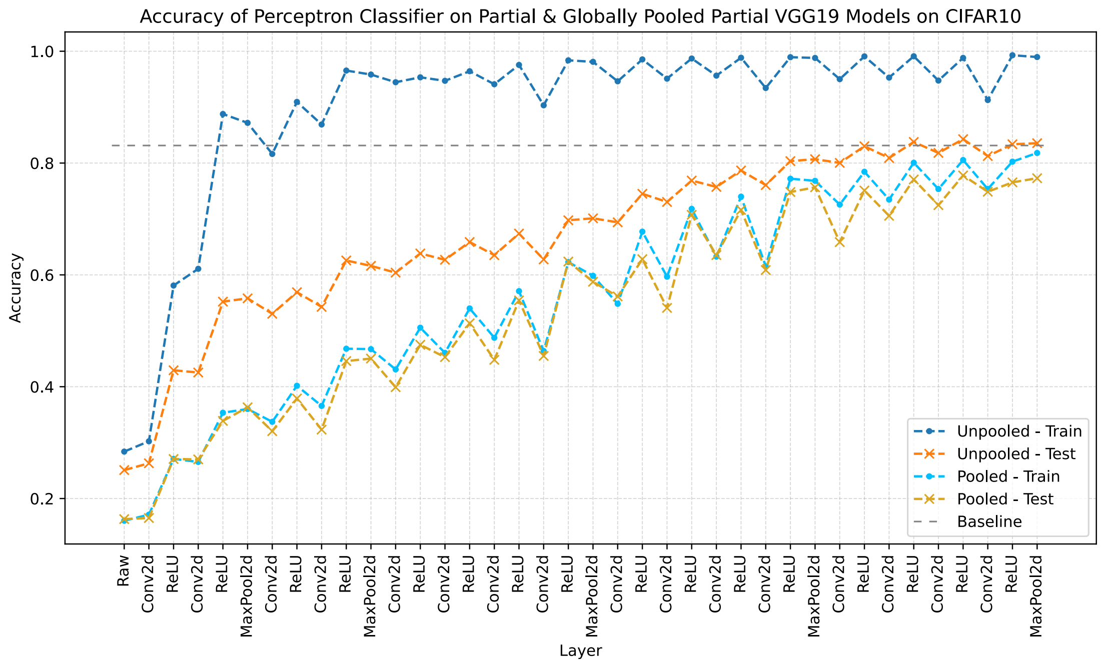


FIGURE C.12: Test & Train accuracy of perceptron probes throughout network with and without global pooling: Model=VGG19, Dataset=CIFAR10

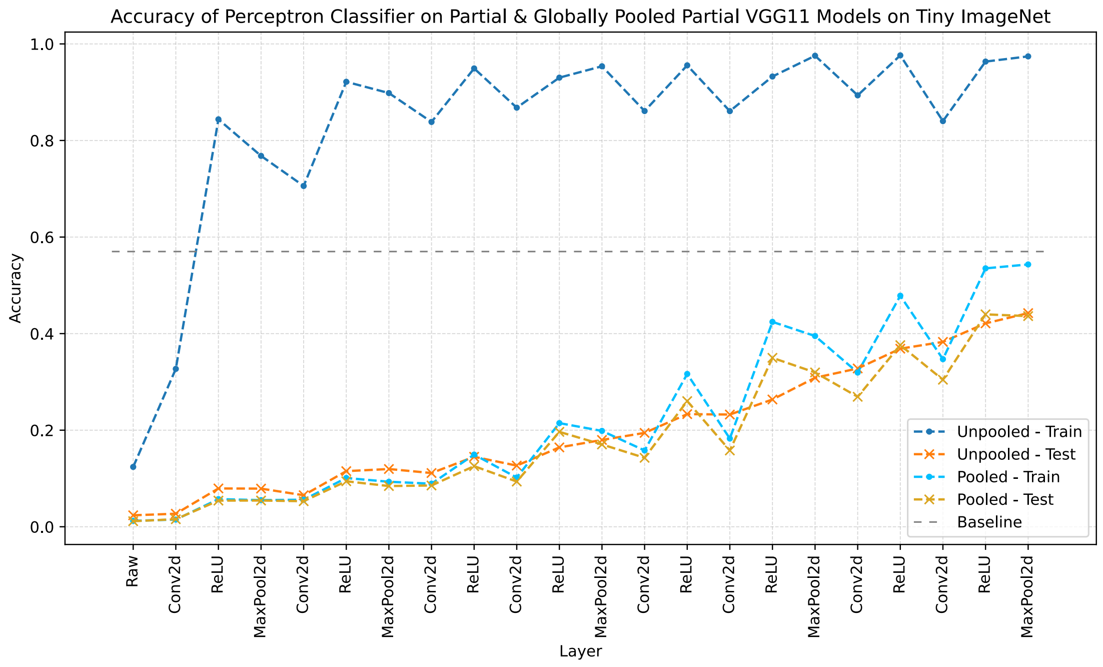


FIGURE C.13: Test & Train accuracy of perceptron probes throughout network with and without global pooling: Model=VGG11, Dataset=Tiny Imagenet

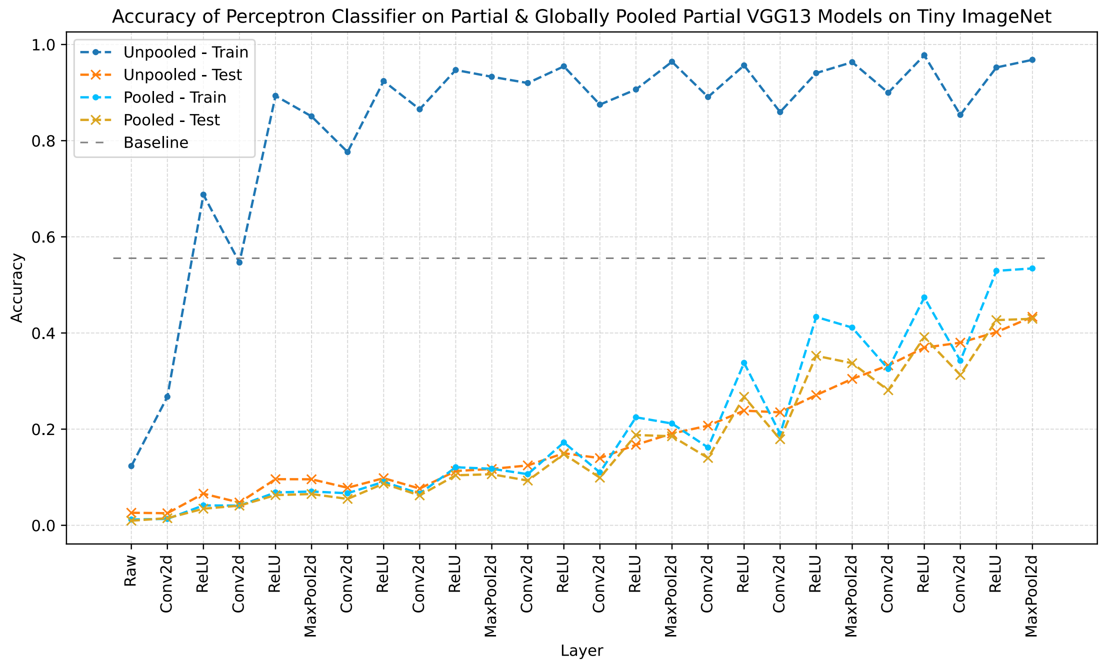


FIGURE C.14: Test & Train accuracy of perceptron probes throughout network with and without global pooling: Model=VGG13, Dataset=Tiny Imagenet

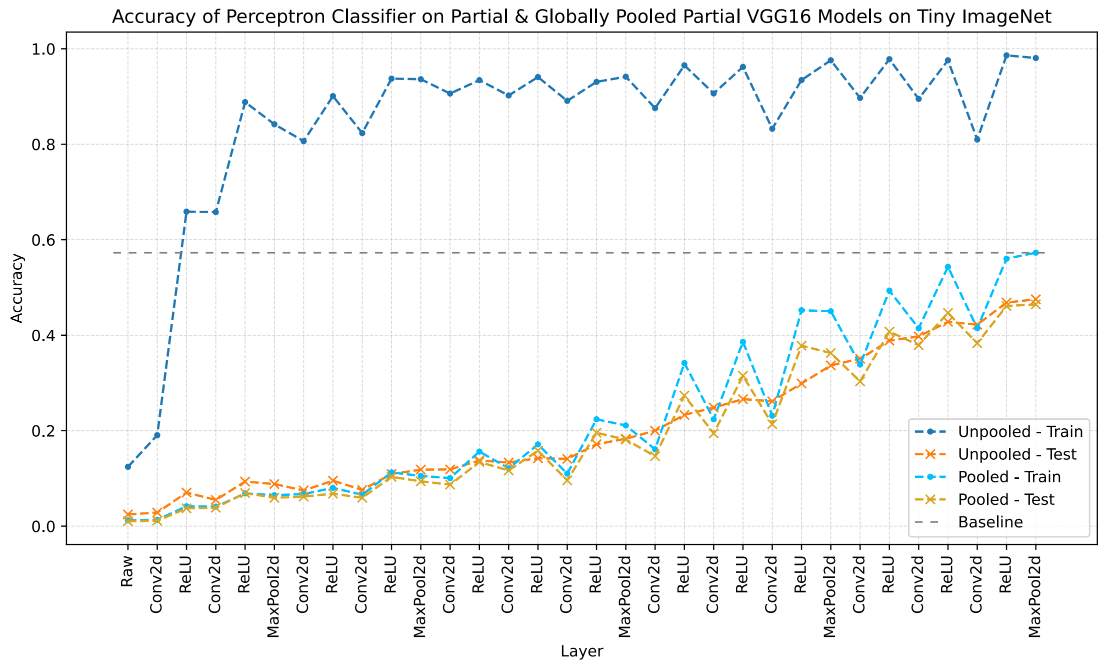


FIGURE C.15: Test & Train accuracy of perceptron probes throughout network with and without global pooling: Model=VGG16, Dataset=Tiny Imagenet

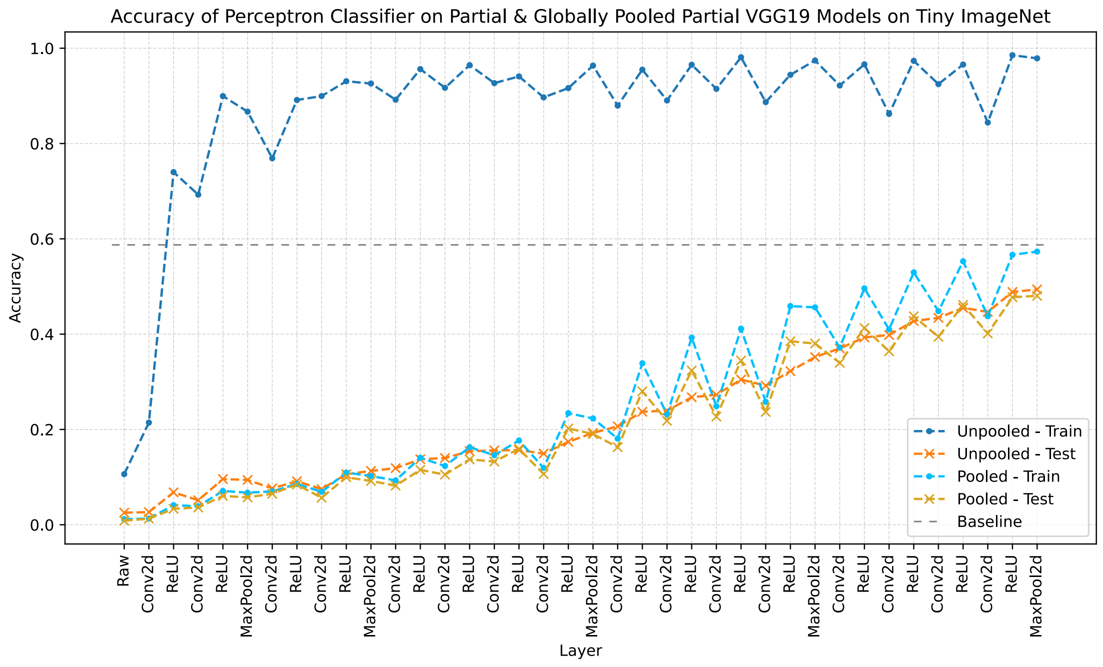


FIGURE C.16: Test & Train accuracy of perceptron probes throughout network with and without global pooling: Model=VGG19, Dataset=Tiny Imagenet

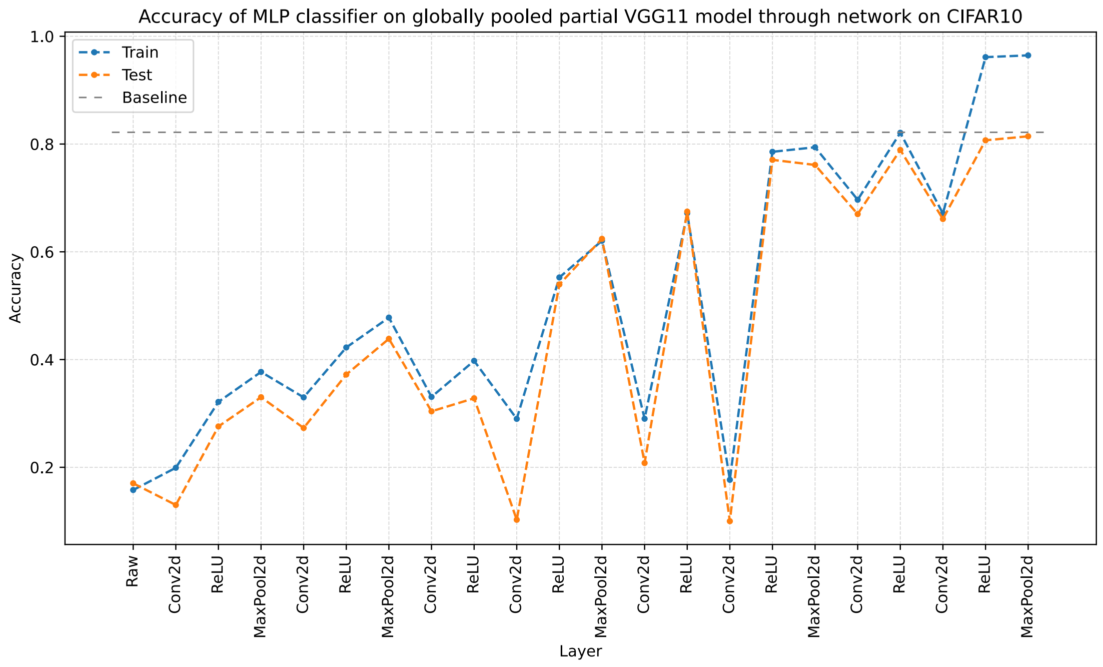


FIGURE C.17: Test & Train accuracy of MLP probes throughout network with global pooling: Model=VGG11, Dataset=CIFAR10

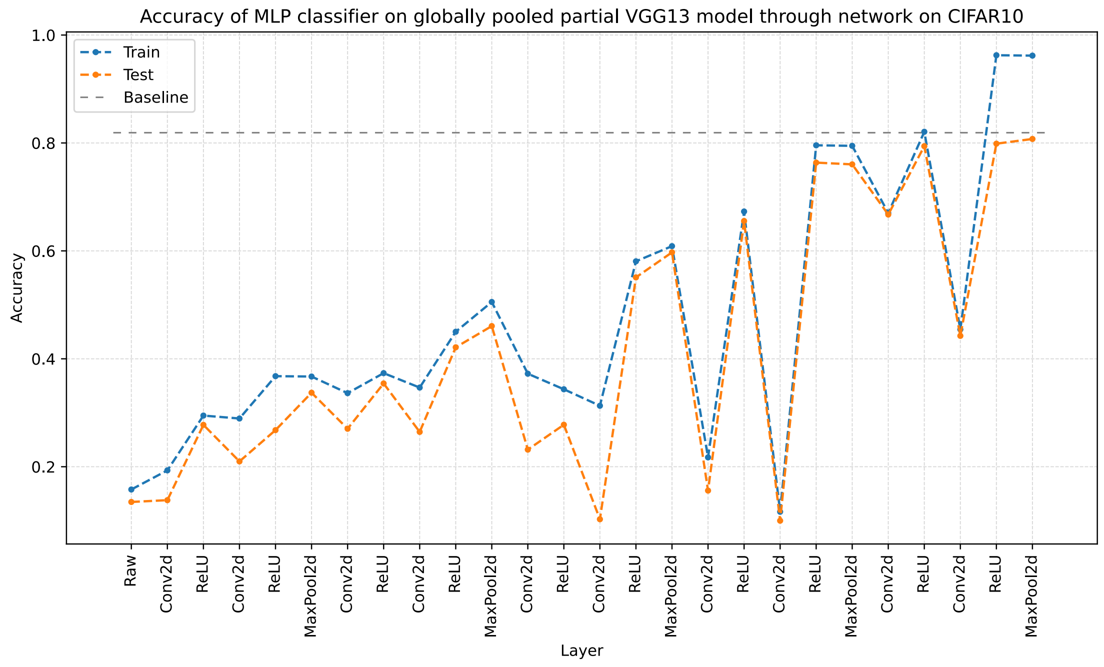


FIGURE C.18: Test & Train accuracy of MLP probes throughout network with global pooling: Model=VGG13, Dataset=CIFAR10

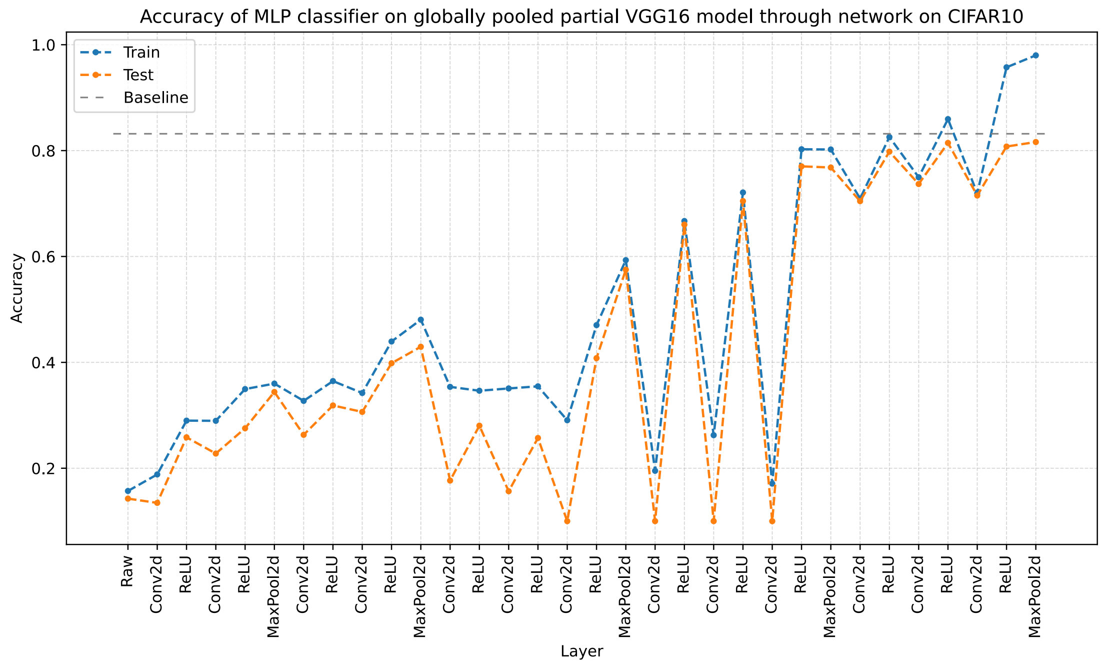


FIGURE C.19: Test & Train accuracy of MLP probes throughout network with global pooling: Model=VGG16, Dataset=CIFAR10

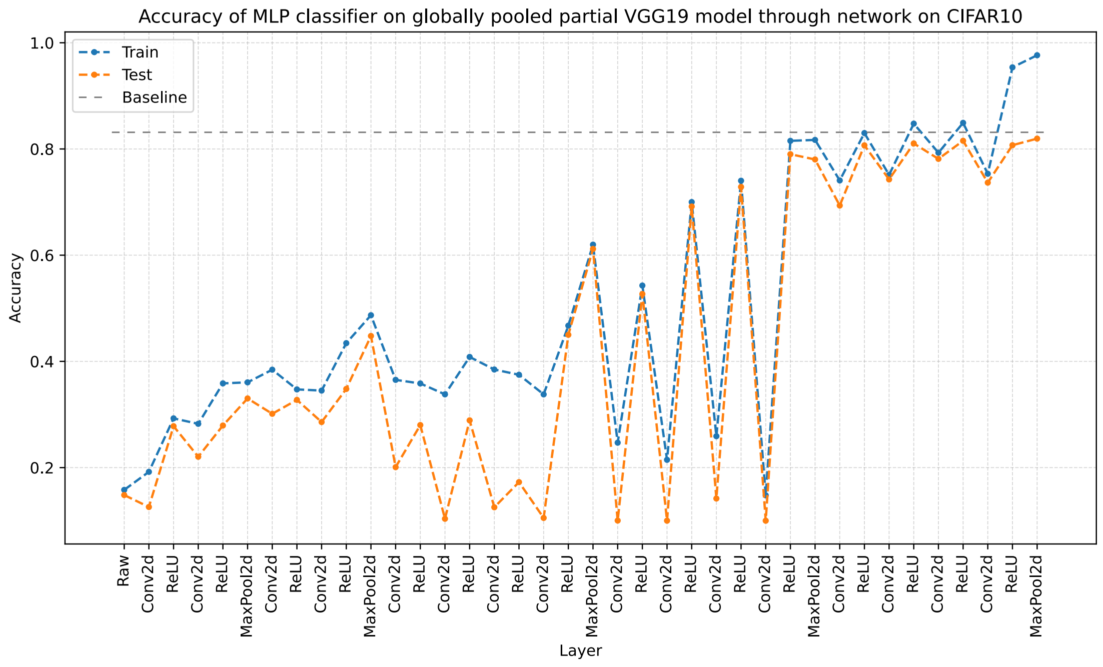


FIGURE C.20: Test & Train accuracy of MLP probes throughout network with global pooling: Model=VGG19, Dataset=CIFAR10

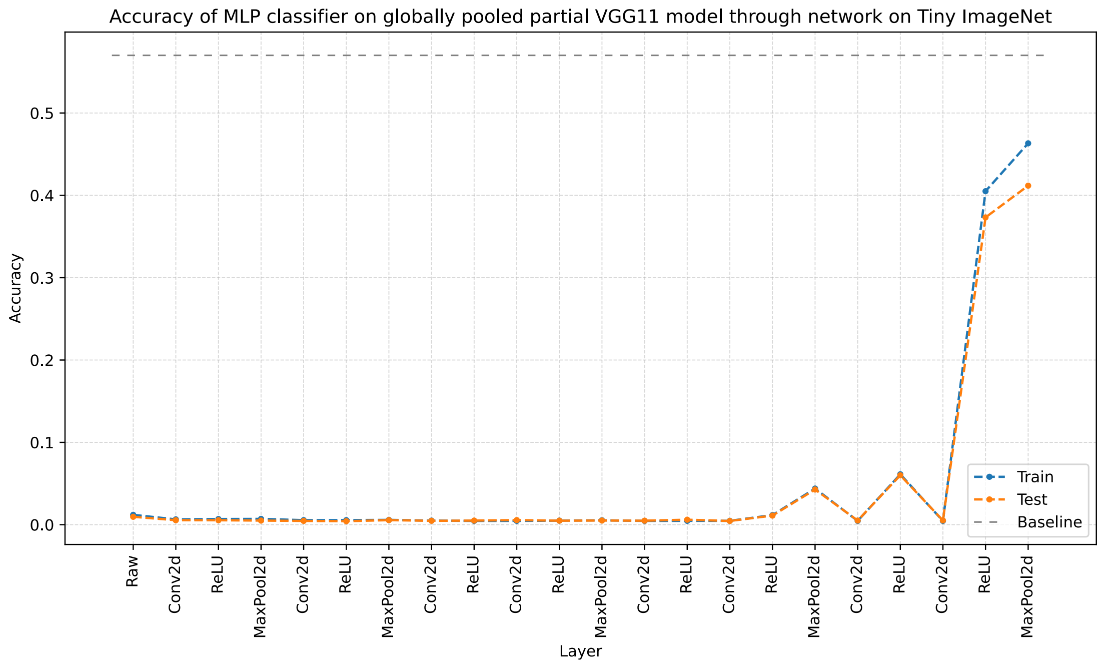


FIGURE C.21: Test & Train accuracy of MLP probes throughout network with global pooling: Model=VGG11, Dataset=Tiny Imagenet

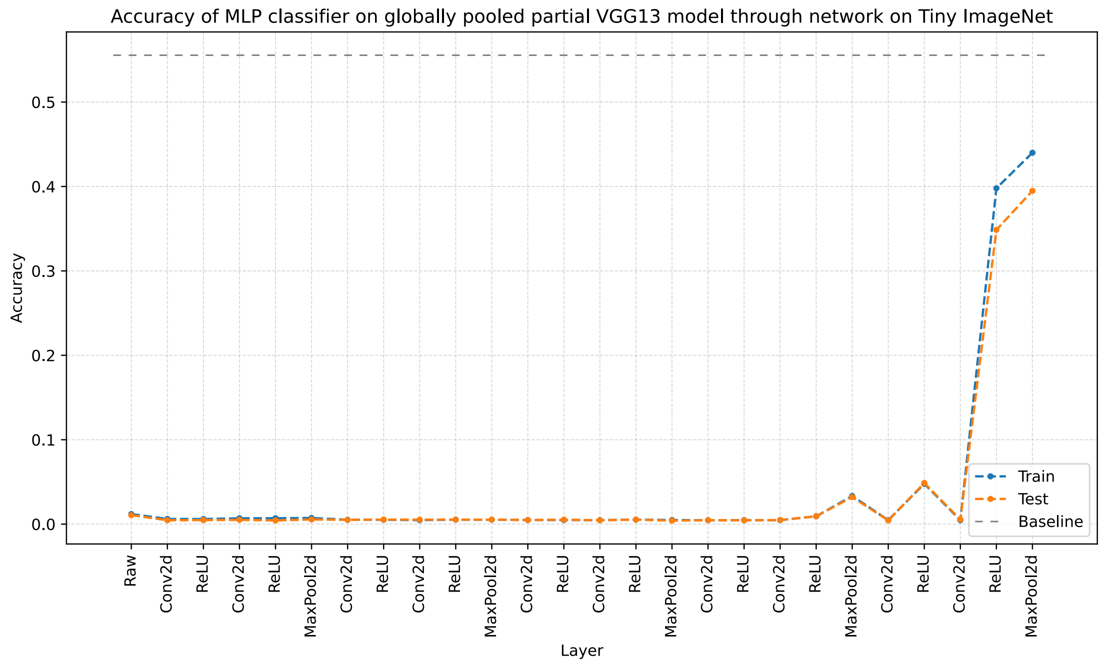


FIGURE C.22: Test & Train accuracy of MLP probes throughout network with global pooling: Model=VGG13, Dataset=Tiny Imagenet

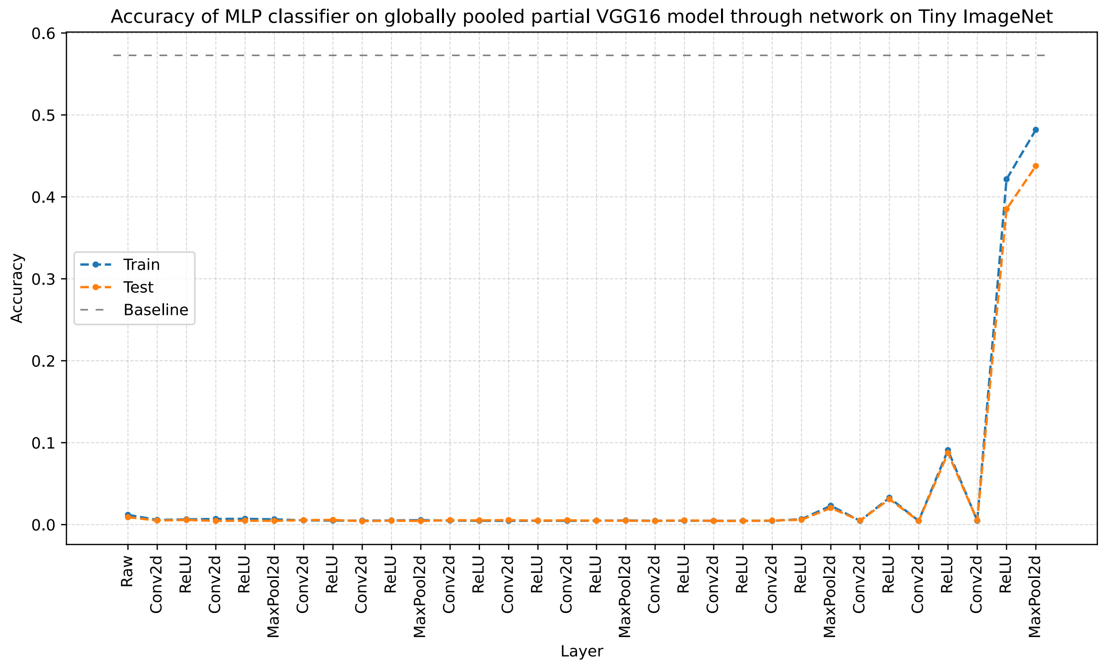


FIGURE C.23: Test & Train accuracy of MLP probes throughout network with global pooling: Model=VGG16, Dataset=Tiny Imagenet

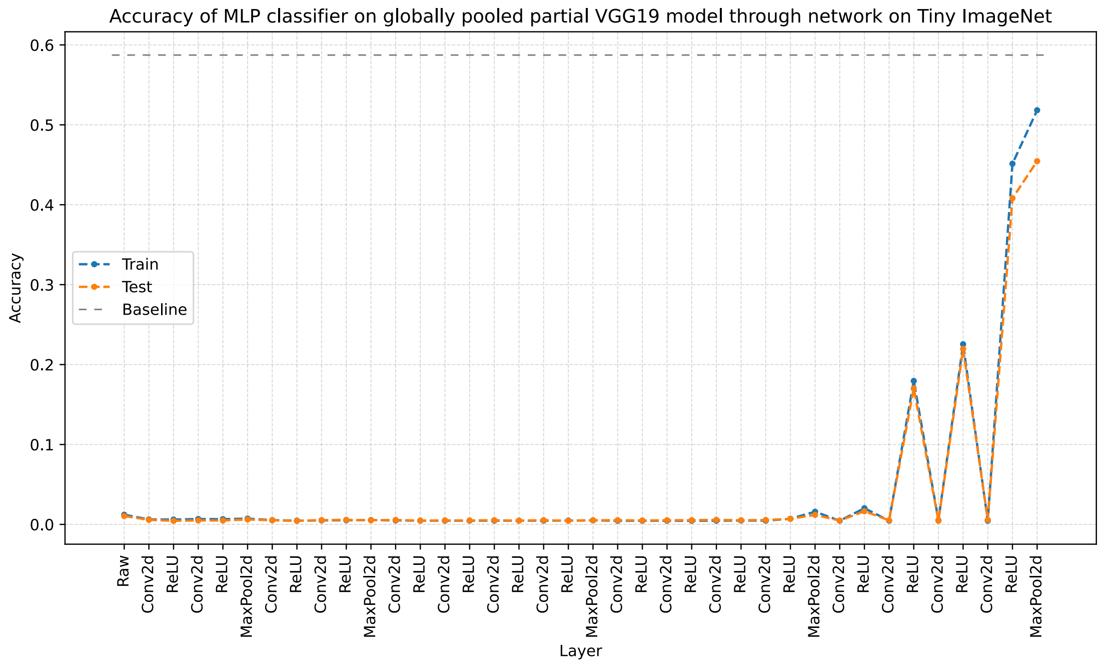


FIGURE C.24: Test & Train accuracy of MLP probes throughout network with global pooling: Model=VGG19, Dataset=Tiny Imagenet

Appendix D

Software Used in this Work

This work is based on experiments performed using code of my own authorship, utilising various existing libraries and tools.

All code was written using the Python programming language.

In all cases, neural networks were implemented using the PyTorch framework (Ansel et al. 2024). The VGG models and pre-trained weights used were provided by the Torchvision library (maintainers et al. 2016). Functionality for training and evaluating models was provided by the Torchbearer library (Harris, Painter, et al. 2018). Various additional functionality was provided by the NumPy (Harris, Millman, et al. 2020) and SciPy (Virtanen et al. 2020) libraries.

Figures were produced using Matplotlib (Hunter 2007)

Bibliography

Alain, Guillaume and Yoshua Bengio (2017). *Understanding intermediate layers using linear classifier probes*. URL: https://openreview.net/forum?id=ryF7rTqgl.

Alom, Md. Zahangir et al. (Mar. 2018). “The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches”. In: DOI: 10.48550/arXiv.1803.01164.

Ansel, Jason et al. (Apr. 2024). “PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation”. In: *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS ’24)*. ACM. DOI: 10.1145/3620665.3640366. URL: https://pytorch.org/assets/pytorch2-2.pdf.

Baldock, Robert J. N., Hartmut Maennel, and Behnam Neyshabur (2021). *Deep Learning Through the Lens of Example Difficulty*. arXiv: 2106.09647 [cs.LG]. URL: https://arxiv.org/abs/2106.09647.

Belcher, Dominic, Adam Prugel-Bennett, and Srinandan Dasmahapatra (2020). “Generalisation and the Geometry of Class Separability”. In: *NeurIPS 2020 Workshop: Deep Learning through Information Geometry*. URL: https://openreview.net/forum?id=4NtqESjOIAz.

Belkin, Mikhail et al. (July 2019). “Reconciling modern machine-learning practice and the classical bias–variance trade-off”. In: *Proceedings of the National Academy of Sciences* 116.32, pp. 15849–15854. ISSN: 1091-6490. DOI: 10.1073/pnas.1903070116. URL: http://dx.doi.org/10.1073/pnas.1903070116.

Choudhary, Kamal et al. (Apr. 2022). “Recent advances and applications of deep learning methods in materials science”. In: *npj Computational Materials* 8.1, p. 59. ISSN: 2057-3960. DOI: 10.1038/s41524-022-00734-6. URL: https://doi.org/10.1038/s41524-022-00734-6.

Deng, Jia et al. (2009). “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee, pp. 248–255.

Harris, Charles R., K. Jarrod Millman, et al. (Sept. 2020). “Array programming with NumPy”. In: *Nature* 585.7825, pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

Harris, Ethan, Matthew Painter, and Jonathon Hare (2018). “Torchbearer: A Model Fitting Library for PyTorch”. In: *arXiv preprint arXiv:1809.03363*.

He, Kaiming et al. (2016). "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

Hunter, J. D. (2007). "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3, pp. 90–95. DOI: 10.1109/MCSE.2007.55.

ImageNet Benchmark (2024). URL: https://paperswithcode.com/sota/image-classification-on-imagenet.

Keskar, Nitish Shirish et al. (2017). *On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima*. arXiv: 1609.04836 [cs.LG]. URL: https://arxiv.org/abs/1609.04836.

Kolesnikov, Alexander et al. (2021). "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In.

Krizhevsky, Alex and Geoffrey Hinton (2009). *Learning multiple layers of features from tiny images*. Tech. rep. Citeseer.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

maintainers, TorchVision and contributors (Nov. 2016). *TorchVision: PyTorch's Computer Vision library*. URL: https://github.com/pytorch/vision.

mnmoustafa, Mohammed Ali (2017). *Tiny ImageNet*. URL: https://kaggle.com/competitions/tiny-imagenet.

Nakkiran, Preetum et al. (2019). *Deep Double Descent: Where Bigger Models and More Data Hurt*. arXiv: 1912.02292 [cs.LG]. URL: https://arxiv.org/abs/1912.02292.

Neyshabur, Behnam, Srinadh Bhojanapalli, et al. (2017). *Exploring Generalization in Deep Learning*. arXiv: 1706.08947 [cs.LG]. URL: https://arxiv.org/abs/1706.08947.

Neyshabur, Behnam, Zhiyuan Li, et al. (2018). *Towards Understanding the Role of Over-Parametrization in Generalization of Neural Networks*. arXiv: 1805.12076 [cs.LG]. URL: https://arxiv.org/abs/1805.12076.

Neyshabur, Behnam, Ryota Tomioka, and Nathan Srebro (2015). *Norm-Based Capacity Control in Neural Networks*. arXiv: 1503.00036 [cs.LG]. URL: https://arxiv.org/abs/1503.00036.

Shalev-Shwartz, Shai and Shai Ben-David (2014). *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press.

Simonyan, Karen and Andrew Zisserman (2015). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv: 1409.1556 [cs.CV]. URL: https://arxiv.org/abs/1409.1556.

Srivastava, Siddharth and Gaurav Sharma (2023). *OmniVec: Learning robust representations with cross modal sharing*. arXiv: 2311.05709 [cs.CV]. URL: https://arxiv.org/abs/2311.05709.

Vapnik, V. (1991). “Principles of Risk Minimization for Learning Theory”. In: *Advances in Neural Information Processing Systems*. Ed. by J. Moody, S. Hanson, and R.P. Lippmann. Vol. 4. Morgan-Kaufmann. URL: https://proceedings.neurips.cc/paper_files/paper/1991/file/ff4d5fbbafdf976cfdc032e3bde78de5-Paper.pdf.

Vaswani, Ashish et al. (2017). “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Virtanen, Pauli et al. (2020). “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17, pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

Yu, Jiahui et al. (2022). *CoCa: Contrastive Captioners are Image-Text Foundation Models*. arXiv: 2205.01917 [cs.CV]. URL: https://arxiv.org/abs/2205.01917.

Zhang, Chiyuan et al. (2017). *Understanding deep learning requires rethinking generalization*. arXiv: 1611.03530 [cs.LG]. URL: https://arxiv.org/abs/1611.03530.

– (Feb. 2021). “Understanding deep learning (still) requires rethinking generalization”. In: *Commun. ACM* 64.3, pp. 107–115. ISSN: 0001-0782. DOI: 10.1145/3446776. URL: https://doi.org/10.1145/3446776.