5IA23 - Deep learning based computer vision

Practical Work on OOD Detection and Neural Collapse

Rian SANTOS COSTA
Vitor ODORISSIO PEREIRA

2025

Table des matières

Introduction **2**

1 ResNet Implementation and Training **3**
1.1 ResNet Architecture . . . 3
1.2 Base Block . . . 4
1.3 ResNet for ImageNet . . . 5
1.4 ResNet for CIFAR . . . 6
1.5 Training Results . . . 7

2 OOD Scores **9**
2.0.1 Softmax . . . 11
2.0.2 Maximum Logit Score . . . 12
2.0.3 Mahalanobis . . . 12
2.0.4 Energy Score . . . 12
2.0.5 ViM . . . 12
2.1 Results . . . 13

3 Neural Collapse **15**
3.1 NC1 : Variability Collapse (Within-Class Variation) . . . 15
3.2 NC2 : Convergence to Simplex ETF . . . 15
3.3 NC3 : Self-Duality . . . 16
3.4 NC4 : Simplification to Nearest Class Center (NCC) . . . 16

4 NC5 : ID/OOD Orthogonality **16**
4.1 Definition and Intuition . . . 16
4.2 Mathematical Formulation . . . 17
4.3 Implications for Detection . . . 17

5 NECO Implementation **18**
5.1 Neural Collapse Analysis . . . 19

6 Neural Collapse on Earlier Layers (BONUS) **24**
6.1 Motivation and Architectural Context . . . 24
6.2 Theoretical Expectations for Intermediate Representations . . . 24
6.3 Implications for NECO and OOD Detection . . . 24

7 Mechanistic Interpretability (BONUS) **26**

8 Conclusion **27**

Introduction

In this project we will explore the Out-of-Distribution concepts. More specifically, its detection methods, scoring systems and how it behaves under a Neural Collapse state.

The original NECO paper done by Mouïn Ben Ammar, Nacim Belkhir, Sebastian Popescu, Antoine Manzanera, and Gianni Franchi is available at [1], and will be the base of our report.

We will try and reproduce their experiments and dive deeply in the concepts first introduced by them.

1 ResNet Implementation and Training

For the implementation of the ResNet deep neural network, we got guidance from the original paper [4], as well as these implementations in PyTorch : the original implementation of ResNet models [5], the NECO paper implementation [2], an open source implementation by Alan Martyn [7], and the Facebook Research implementation of various ResNet versions [3] (in lua Torch).

1.1 ResNet Architecture

The ResNet architecture was originally introduced for large-scale image classification on ImageNet, where input images have a resolution of 224×224 [4] (section 3.4). In this setting, the network progressively reduces the spatial resolution through an initial convolution with stride 2, followed by a max-pooling layer and four residual stacks, with the final three performing downsampling at their first block. This design halves the image 5 times and results in a final feature map of size 7×7, on which a global average pooling is applied before the final classification layer.

When examining the original architecture in the context of smaller datasets such as CIFAR, several limitations become apparent. CIFAR images have a much lower resolution (32×32), and applying the ImageNet configuration without modification leads to an overly aggressive reduction of spatial information in the early layers. In particular, the combination of the initial strided convolution and max-pooling layer causes a significant loss of information before meaningful feature extraction can take place. As a consequence, the spatial resolution after the four residual stacks collapses to 1×1, rendering the final average pooling operation useless.

The original ResNet paper addresses this issue by proposing a simplified architecture for CIFAR-10, where the initial convolution is replaced by a 3×3 convolution with stride 1 and padding 1, and the max-pooling layer is removed entirely. Furthermore, the network is reduced to three residual stacks with fewer feature maps (32, 16 and 8; equivalent to 16, 32, and 64 channels), which allows the spatial resolution to decrease more gradually. This design preserves more information throughout the network and results in a final feature map of size 8×8, comparable to the 7×7 output in the ImageNet configuration.

Our objective, however, is to train a ResNet-18 architecture originally designed for ImageNet on the CIFAR-100 dataset. This introduces a trade-off between staying faithful to the original ResNet-18 design and adapting the architecture to the constraints imposed by smaller input images. Using the ImageNet architecture unchanged would lead to excessive downsampling, while fully adopting the CIFAR-10 configuration would significantly alter the depth and capacity of the network.

To address this, we explore two different architectures. The first remains closer to the ImageNet configuration. The second variant follows the design principles of the CIFAR-10 variant.

1.2 Base Block

The residual block architecture can be visualized in the following image shown by the Facebook Research report on their torch implementation.

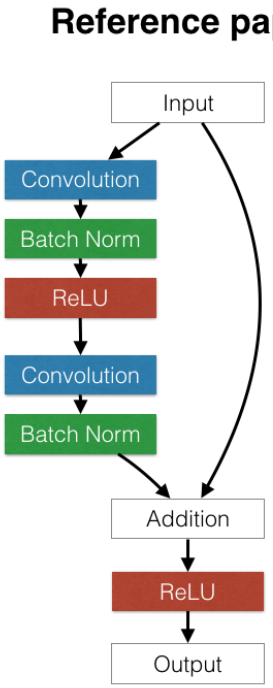


FIGURE 1 – Residual Block Architecture [3]

Thus our implementation follows

```python
class ResidualBlock(nn.Module):
    """
    Residual block described for ResNet-18 and ResNet-34 in the paper

    Also used for ResNet Cifar-10

    Basically 2 3x3 convolutions with a skip connection, and an optional downsampling in the first convolution if stride > 1
    """

    def __init__(self, in_channels, out_channels, stride=1):
        super(ResidualBlock, self).__init__()
        self.first_conv = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False)
```

```
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.second_conv = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1,
        ↪  padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channels)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride,
                ↪  bias=False),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        out = F.relu(self.bn1(self.first_conv(x)))
        out = self.bn2(self.second_conv(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out
```

1.3 ResNet for ImageNet

The described architecture for the ResNet-18 can be seen in the following diagram

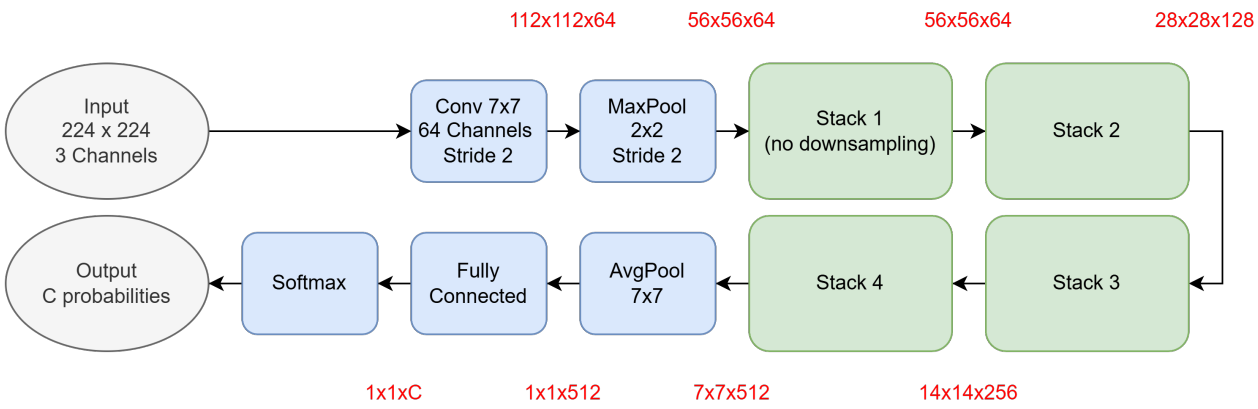


FIGURE 2 – ResNet-18 architecture

Here the batch norms are omitted for legibility, but they exist after each convolution. Each stack is composed of two residual blocks. If a stack has downsampling, the first convolution of the first block will apply a convolution of stride 2, halving the image size and doubling its channels. The residual part (shortcut) will then pass through a 1×1 convolution of stride 2, to then be added to the new sample size. Check table 1 of [4] for more details in the ResNet for ImageNet architectures.

Note that if the input image size were to be 32×32, then the AvgPool should be a 1×1 (do nothing). Thus here is the implementation adapted to these kind of inputs.

```python
class ResNet_ImageNet(nn.Module):
    def __init__(self, block, num_blocks, num_classes=100):
        super(ResNet_ImageNet, self).__init__()
        self.in_channels = 64

        self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.stack1 = self._make_stack(block, 64, num_blocks[0], first_block_stride=1)
        self.stack2 = self._make_stack(block, 128, num_blocks[1], first_block_stride=2)
        self.stack3 = self._make_stack(block, 256, num_blocks[2], first_block_stride=2)
        self.stack4 = self._make_stack(block, 512, num_blocks[3], first_block_stride=2)
        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512, num_classes)
        self.softmax = nn.Softmax(dim=1)

    def _make_stack(self, block, out_channels, num_blocks, first_block_stride):
        strides = [first_block_stride] + [1] * (num_blocks - 1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_channels, out_channels, stride))
            self.in_channels = out_channels
        return nn.Sequential(*layers)

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.stack1(out)
        out = self.stack2(out)
        out = self.stack3(out)
        out = self.stack4(out)
        out = self.avg_pool(out)
        out = torch.flatten(out, 1)
        out = self.fc(out)
        out = self.softmax(out)
        return out

resnet_18 = ResNet_ImageNet(ResidualBlock, [2, 2, 2, 2], num_classes=100)
```

1.4 ResNet for CIFAR

Considering the trade-offs discussed in section 1.1 for the CIFAR dataset, here follows our implementation of the suggested architecture in [4] section 4.2.

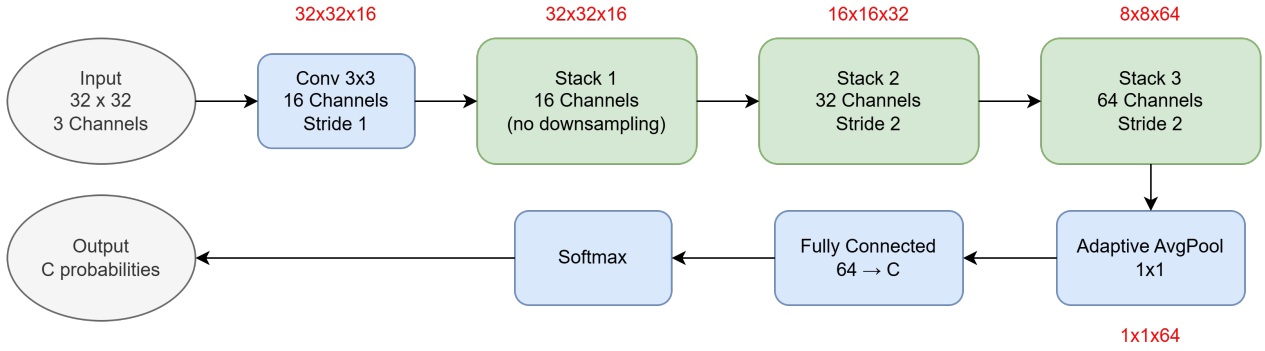


FIGURE 3 – ResNet-18 for CIFAR architecture

```python
class ResNet_CIFAR(nn.Module):
    def __init__(self, block, num_blocks, num_classes=100):
        super(ResNet_CIFAR, self).__init__()
        self.in_channels = 16

        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(16)
        self.stack1 = self._make_stack(block, 16, num_blocks[0], first_block_stride=1)
        self.stack2 = self._make_stack(block, 32, num_blocks[1], first_block_stride=2)
        self.stack3 = self._make_stack(block, 64, num_blocks[2], first_block_stride=2)
        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(64, num_classes)
        self.softmax = nn.Softmax(dim=1)

    def _make_stack(self, block, out_channels, num_blocks, first_block_stride):
        strides = [first_block_stride] + [1] * (num_blocks - 1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_channels, out_channels, stride))
            self.in_channels = out_channels
        return nn.Sequential(*layers)

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.stack1(out)
        out = self.stack2(out)
        out = self.stack3(out)
        out = self.avg_pool(out)
        out = torch.flatten(out, 1)
        out = self.fc(out)
        out = self.softmax(out)
        return out

resnet_cifar = ResNet_CIFAR(ResidualBlock, [2, 2, 2], num_classes=100)
```

1.5 Training Results

All training results can be seen here. This spreadsheet contains some information about how we trained the architecture.

We found out very quickly that for CIFAR, the proposed architecture for ImageNet couldn't learn, so we trained the network `resnet_cifar = ResNet_CIFAR(ResidualBlock, [2, 2, 2], num_classes=100)` in different ways to try and achieve Neural Collapse.

The training method that gave us the smallest training loss was the "collapse" method in the code, where we removed data augmentation and trained with Adam with zero weight decay.

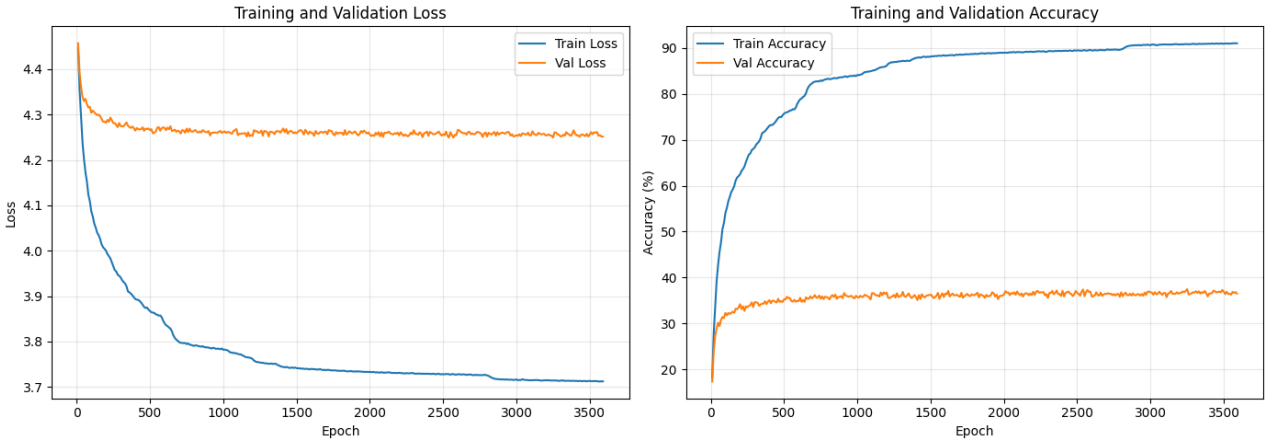


FIGURE 4 – Training curve for Collapsed Network

The best network, achieved 65.85% of accuracy after training in 908 epochs with SDG and cosine scheduler for learning rate. It is clear to me that with better training arguments and hyper parameters we could increase these metrics. But we were reaching the 4 hour time limit in our cluster. We believe that training with smaller batches and Adam regressor would lead to better results in loss and accuracy, in exchange it would take longer to train.

FIGURE 5 – Training curve for Best Network

We then tested this model in some OOD scores, which we will discuss in the next section. The checkpoint for this model can be found here.

2 OOD Scores

To understand how OOD scores are calculated we referred to the paper that introduced energy-based OOD scores [6], which had a very didactic explanation on how the math is done. To implement diverse OOD scores we took inspiration from the NECO repository [2].

We will start by introducing how we extracted the last layer features of our network. Full code can be found here

```
# Most important parts of the code

# Create model and load trained weights
model = ResNet_CIFAR(ResidualBlock, [2, 2, 2], num_classes=100)
checkpoint = torch.load(checkpoint_path, map_location=device)

# Set to evaluation mode and capture the only fully connected layer of the model
model.eval()
fc = model.fc

# Save wwights and bias of last layer
weight = fc.weight.detach().cpu()
bias = fc.bias.detach().cpu()

# Create a hook to add features as we run inference
def hook_fn(module, inputs, output):
    features.append(inputs[0].detach().cpu())
hook = fc.register_forward_hook(hook_fn)
```

After this we will have inferred both our ID and OOD data through our model and saved them into some CSV's so that we can calculate the scores more quickly.

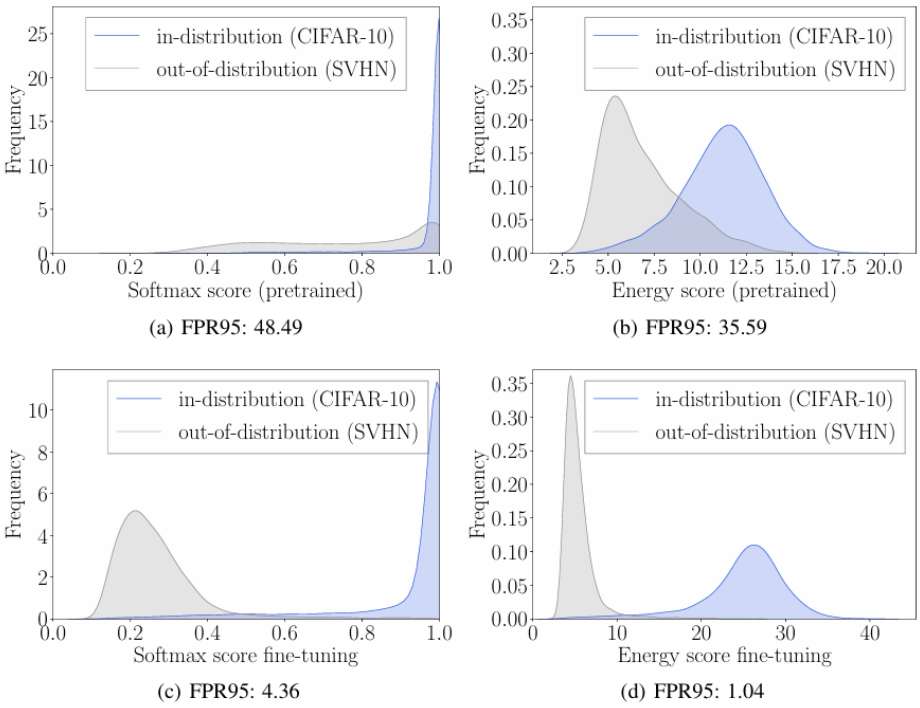
Then we have to prepare the input data of our score calculating methods

```
logit_id_train = feature_id_train @ weight.T + bias
logit_id_val = feature_id_val @ weight.T + bias
logit_ood = feature_ood @ weight.T + bias

softmax_id_train = softmax(logit_id_train, axis=-1)
softmax_id_val = softmax(logit_id_val, axis=-1)
softmax_ood = softmax(logit_ood, axis=-1)
```

So all we are missing is the calculation. For this we will have to understand what the scores actually mean and how to calculate them.

This image from [6] helped us to visualize a lot what the metrics functions in the NECO repository were replicating.



(a) FPR95: 48.49 (b) FPR95: 35.59

(c) FPR95: 4.36 (d) FPR95: 1.04

FIGURE 6 – Distribution of softmax scores vs. energy scores

So then we can use the sklearn methods along with some numpy functions to create these distributions and measure the areas on those curves.

This portion of the NECO code creates the two curves

```
conf = np.concatenate((ind_conf, ood_conf))
ind_indicator = np.concatenate((np.ones_like(ind_conf), np.zeros_like(ood_conf)))

fpr, tpr, _ = metrics.roc_curve(ind_indicator, conf)
precision_in, recall_in, _ = metrics.precision_recall_curve(ind_indicator, conf)
precision_out, recall_out, _ = metrics.precision_recall_curve(1 - ind_indicator, 1 - conf)
```

and this portion

```
auroc = metrics.auc(fpr, tpr)
aupr_in = metrics.auc(recall_in, precision_in)
aupr_out = metrics.auc(recall_out, precision_out)
```

calculates three very useful metrics :

Area Under ROC Curve

— Measures : "Can this confidence score distinguish ID from OOD ?"

— Range : 0-1 (1 = perfect separation, 0.5 = random)

Area Under Precision-Recall curve (ID class)
— Measures : "When we predict ID, how often are we correct ?"
— Useful when ID samples are the minority
Area Under Precision-Recall curve (OOD class)
— Measures : "When we predict OOD, how often are we correct ?"
Additionally, we can measure the False Positive Rate at a specific True Positive Rate (recall), which is particularly useful for understanding OOD detection performance at high recall thresholds.

This portion of the NECO code establishes the threshold at a target recall level :

```
recall_num = int(np.floor(tpr * num_ind))
thresh = np.sort(ind_conf)[-recall_num]
num_fp = np.sum(ood_conf >= thresh)
fpr = num_fp / max(1, num_ood)
```

This computes a single, interpretable metric :

False Positive Rate at Target Recall
— Measures : "At a fixed recall level (e.g., 95%), what fraction of OOD samples are incorrectly accepted ?"
— Useful for understanding trade-offs : higher recall means accepting more ID samples, but also more OOD false positives
— Lower FPR at high recall indicates better OOD detection capability
This metric directly answers : given that we correctly identify 95% of ID samples, how many OOD samples slip through ? So it is looked when missing OOD samples is costly.

Ok, now that we understand what metrics are going to be important for us, we simply need to learn how to actually calculate scores before passing to these functions.

2.0.1 Softmax

The baseline OOD detector uses the maximum softmax probability :

$$S_{\text{softmax}}(\mathbf{x}) = \max_i p_i = \max_i \frac{e^{f_i(\mathbf{x})}}{\sum_j e^{f_j(\mathbf{x})}} \tag{1}$$

where f_i is the logit for class i. High probability indicates confidence in the prediction.

2.0.2 Maximum Logit Score

Uses raw logits directly, bypassing softmax normalization :

$$S_{\text{logit}}(\mathbf{x}) = \max_i f_i(\mathbf{x}) \tag{2}$$

Often more effective than softmax as it avoids calibration issues and captures unnormalized confidence.

2.0.3 Mahalanobis

Measures distance from sample features to class-conditional Gaussian distributions. For a given class k :

$$S_{\text{mahal}}(\mathbf{x}) = -\min_k \sqrt{(\mathbf{z} - \boldsymbol{\mu}_k)^T \Sigma_k^{-1} (\mathbf{z} - \boldsymbol{\mu}_k)} \tag{3}$$

where $\mathbf{z}$ is the penultimate feature vector, $\boldsymbol{\mu}_k$ and Σ_k are class-specific mean and covariance estimated from training features. Lower distance (higher score) indicates ID.

2.0.4 Energy Score

Based on energy-based models, defines energy as :

$$S_{\text{energy}}(\mathbf{x}) = -\log \sum_i e^{f_i(\mathbf{x})} = -\text{LogSumExp}(\mathbf{f}) \tag{4}$$

Equivalent to the denominator of softmax. More sensitive to low-probability outliers than max-softmax.

2.0.5 ViM

Vision Mixture exploits the norm of gradients w.r.t. features at the final layer :

$$S_{\text{vim}}(\mathbf{x}) = \left\| \nabla_{\mathbf{z}} f(\mathbf{z}) \right\|_2 \tag{5}$$

where the gradient is computed from the logit output. ID samples produce lower gradient norms ; OOD samples exhibit higher norms due to lack of class structure in feature space.

2.1 Results

We compare here the scores for both the "best" network and the "collapsed" network.

svhn	AUROC	FPR@95%	textures	AUROC	FPR@95%
softmax	0.5689	0.9342	softmax	0.5095	0.9516
max_logit	0.3898	0.9957	max_logit	0.3728	0.9862
energy	0.3895	0.9958	energy	0.3726	0.9867
mahalanobis	0.7172	0.911	mahalanobis	0.6123	0.8287
vim	0.406	0.9952	vim	0.3794	0.984

Table 1 – Collapsed Network scores

The network trained to collapse (91% train / 37% val) produces near-random or sub-random scores on nearly every method. Max logit, energy, and ViM all fall around 0.38–0.41 on both OOD sets, consistently below chance. An AUROC < 0.5 means OOD samples score higher in confidence than ID test samples : the model, having memorized training patterns rather than learning generalizable structure, assigns large logit magnitudes to arbitrary inputs while the held-out CIFAR-100 test set already sits partially out-of-distribution relative to its training manifold. Softmax marginally escapes this inversion (0.57 on SVHN) because the softmax normalization suppresses raw logit scale differences.

Mahalanobis is the sole reliable method (0.72 SVHN / 0.61 textures). By centering on per-class training means and using a shared precision matrix, it exploits the fact that training class centroids are well-defined even in an overfit regime, OOD samples simply fall far from all 100 centroids in feature space, regardless of what confidence the classifier assigns to them.

svhn	AUROC	FPR@95%	textures	AUROC	FPR@95%
softmax	0.7909	0.7848	softmax	0.6249	0.9271
max_logit	0.8558	0.7291	max_logit	0.6257	0.9314
energy	0.8585	0.7329	energy	0.6221	0.9245
mahalanobis	0.5625	0.98	mahalanobis	0.8048	0.592
vim	0.8007	0.8496	vim	0.8368	0.5782

Table 2 – Best Network scores

A well-generalizing model reverses the picture almost completely. For SVHN, max logit (0.86) and energy (0.86) are the top performers, they are numerically near-identical throughout, as expected : when one logit dominates the others, $\log \sum_k e^{z_k} \approx \max_k z_k$ up to an additive constant, so both rank samples essentially the same way. SVHN digit images lie far enough from the CIFAR-100 natural-scene manifold that the entire logit distribution is suppressed, and both methods capture this cleanly. ViM (0.80) follows, with its energy term largely driving the score. Softmax (0.79) lags slightly because normalisation discards the absolute scale of the logits, which is the primary discriminating signal here.

For textures the situation is reversed : softmax, max logit, and energy all plateau around 0.62, indicating that DTD patches can elicit confident class predictions. The model assigns high probability mass to some CIFAR-100 class when presented with a texture patch, defeating confidence-based methods. Mahalanobis (0.80) and ViM (0.84) succeed because texture samples deviate from the class-conditional Gaussian structure in feature space even when the classifier is confident.

The starkest contrast is Mahalanobis on SVHN : 0.72 under collapse, 0.56 under the best model. A well-trained model organizes feature space such that SVHN samples are attracted toward the nearest class centroid by the same low-level features (edges, color) that define CIFAR-100 classes at those scales, reducing Mahalanobis distance. The collapsed model has no such structure, so SVHN samples land in inter-class voids.

All FPR values are high, most above 0.8. Even the best-case results (ViM textures 0.578, Mahalanobis textures 0.592 on the best network) indicate that the detector must apply a very permissive threshold to catch 95% of ID samples, at which point a substantial fraction of OOD samples pass too. This reflects both the intrinsic difficulty of low-resolution 32×32 OOD detection and the limited capacity of the 64-dimensional feature space — a known limitation of CIFAR-scale ResNets relative to larger architectures evaluated in the NECO paper.

3 Neural Collapse

Neural Collapse (NC) is a pervasive phenomenon observed in deep neural networks during the terminal phase of training (specifically, the period of training extended beyond zero training error). As the model approaches perfect fitting on the training data, the geometric structure of the penultimate layer features and the last layer classifier weights converges to a rigid, symmetric configuration.

This structure is characterized by four distinct properties (NC1-NC4), originally formalized by Papyan et al. [8].

3.1 NC1 : Variability Collapse (Within-Class Variation)

The first property describes the compression of feature representations. As training progresses, the feature representations of all samples belonging to the same class k collapse onto a single unique point : their class mean $\boldsymbol{\mu}_k$.

Mathematically, this implies that the within-class covariance $\boldsymbol{\Sigma}_W$ converges to zero relative to the between-class covariance $\boldsymbol{\Sigma}_B$:

$$\boldsymbol{\Sigma}_W \to \mathbf{0} \tag{6}$$

Intuition : The model effectively discards all "intra-class" nuisance information (e.g., background, orientation) and retains only the core class identity. For OOD detection, this suggests that In-Distribution (ID) data should be tightly clustered, whereas OOD data is expected to be more dispersed and violate this collapse.

3.2 NC2 : Convergence to Simplex ETF

Once the class features collapse to their means $\boldsymbol{\mu}_k$, these means arrange themselves in the most separate configuration possible within the feature space. This structure is known as a *Simplex Equiangular Tight Frame* (ETF).

For K classes, the centered class means form a geometric structure where :

1. All class means have equal length (equal energy).
2. The angle between any pair of distinct class means is identical and maximal.

The cosine of the angle between any two normalized class means converges to :

$$\langle \tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\mu}}_{k'} \rangle = -\frac{1}{K-1} \quad \text{for } k \neq k' \tag{7}$$

This creates a geometric "simplex" where all classes are maximally distinguishable from one another.

3.3 NC3 : Self-Duality (Alignment of Weights and Means)

This property links the feature extractor to the final linear classifier. It states that the classifier weights $\mathbf{w}_k$ for class k align perfectly with the class means $\boldsymbol{\mu}_k$ of the features.

Up to a scalar rescaling, the normalized weights and normalized means become identical :

$$\tilde{\mathbf{w}}_k \approx \tilde{\boldsymbol{\mu}}_k \tag{8}$$

Intuition : The classifier does not learn arbitrary decision boundaries; instead, the weight vectors point directly at the centers of the data clusters formed by the feature extractor.

3.4 NC4 : Simplification to Nearest Class Center (NCC)

As a consequence of the previous properties (specifically NC1 and NC3), the decision rule of the complex deep network simplifies drastically. The network effectively behaves as a *Nearest Class Center* (NCC) classifier.

Given a test sample $\mathbf{x}$ with feature representation $\mathbf{h}$, the network classifies it based on the Euclidean distance to the nearest class mean, rather than complex non-linear boundaries :

$$\arg\max_k \langle \mathbf{w}_k, \mathbf{h} \rangle \rightarrow \arg\min_k \|\mathbf{h} - \boldsymbol{\mu}_k\| \tag{9}$$

This simplification is crucial for methods like NECO, as it implies that distance to class centers (or projection onto the ETF structure) is the theoretically optimal metric for assessing reliability in well-trained networks.

4 NC5 : ID/OOD Orthogonality

While the standard Neural Collapse properties (NC1–NC4) describe the geometric simplification of In-Distribution (ID) data, they do not explicitly model the behavior of Out-of-Distribution (OOD) data. The NECO, from the paper Ammar et al, (2024) [1] method introduces and relies on a fifth property, **NC5 (ID/OOD Orthogonality)**, which emerges during the terminal phase of training.

4.1 Definition and Intuition

NC5 postulates that as the model converges to the Simplex ETF structure for ID data, the feature representations of OOD data tend to become **orthogonal** to the ID class means.

Geometrically, this means that while ID samples cluster tightly around their respective vertices of the simplex (NC1 + NC2), OOD samples drift toward the "null space" of the principal subspace spanned by the ID features.

4.2 Mathematical Formulation

Let μ_c be the class mean for ID class c, and μ_{OOD}^G be the global mean of the OOD data representations. NC5 states that the cosine similarity between any ID class mean and the OOD global mean converges to zero :

$$\forall c \in \{1, \dots, K\}, \quad \frac{\langle \mu_c, \mu_{OOD}^G \rangle}{\|\mu_c\|_2 \|\mu_{OOD}^G\|_2} \to 0 \tag{10}$$

To quantify this phenomenon empirically, we measure the *OrthoDev* (Deviation from Orthogonality), which is the average absolute cosine similarity :

$$\text{OrthoDev} = \frac{1}{K} \sum_{c=1}^{K} \left| \frac{\langle \mu_c, \mu_{OOD}^G \rangle}{\|\mu_c\|_2 \|\mu_{OOD}^G\|_2} \right| \tag{11}$$

4.3 Implications for Detection

The existence of NC5 is the foundational hypothesis for the NECO score.

— If ID data lies on a specific low-dimensional simplex (the ETF),

— And OOD data is orthogonal to this simplex (NC5),

Then, projecting a test sample onto the subspace spanned by the ID class means effectively separates ID from OOD. ID samples will retain significantly more energy (norm) after projection, whereas OOD samples (being orthogonal) will have a projected norm close to zero.

5 NECO Implementation

Our implementation follows the reference code released alongside the original NECO paper [2]. The score is computed as the fraction of a sample's feature norm captured by the leading d_{NECO} principal components of the (standardized) training feature distribution, a quantity that is high for in-distribution samples, whose representations are well-explained by the dominant ID subspace, and low for OOD samples that project significantly onto residual dimensions. The mathematical formulation was presented in the previous section, here we focus on the implementation decisions.

Three adaptations were made for the ResNet CIFAR setting. First, `StandardScaler` is applied unconditionally. The original code bypasses scaling for ViT and Swin architectures whose `[CLS]` tokens are already approximately unit-norm ; ResNet penultimate features carry no such guarantee. Second, the max-logit multiplicative term

```
if model_architecture_type != 'resnet':
    score_id *= score_id_maxlogit
    score_ood *= score_ood_maxlogit
```

is dropped entirely, as the paper explicitly excludes it for ResNet architectures. Third, PCA is fitted over all 64 feature dimensions (the full rank of the 64-dimensional penultimate space) with $d_{\text{NECO}} = 10$ as the default cutoff, selected as a conservative estimate of the number of directions that carry structured ID variance at CIFAR scale.

svhn	AUROC	FPR@95%	textures	AUROC	FPR@95%
softmax	0.5689	0.9342	softmax	0.5095	0.9516
max_logit	0.3898	0.9957	max_logit	0.3728	0.9862
energy	0.3895	0.9958	energy	0.3726	0.9867
mahalanobis	0.7172	0.911	mahalanobis	0.6123	0.8287
vim	0.406	0.9952	vim	0.3794	0.984
neco	0.1872	0.9977	neco	0.3893	0.9718

Table 3 – Extended Collapsed Network scores

svhn	AUROC	FPR@95%	textures	AUROC	FPR@95%
softmax	0.7909	0.7848	softmax	0.6249	0.9271
max_logit	0.8558	0.7291	max_logit	0.6257	0.9314
energy	0.8585	0.7329	energy	0.6221	0.9245
mahalanobis	0.5625	0.98	mahalanobis	0.8048	0.592
vim	0.8007	0.8496	vim	0.8368	0.5782
neco	0.4812	0.949	neco	0.6009	0.8782

Table 4 – Extended Best Network scores

NECO's performance is sensitive to the quality of the learned feature manifold in a way the other methods are not, which makes this experimental setting a particularly demanding one for it.

In the collapsed network, NECO achieves 0.19 AUROC on SVHN, the worst result in the table, worse than random. The method's geometric premise requires that training features form a structured, approximately low-rank subspace ; a model trained to overfit without augmentation does not produce this structure. The principal components of the training distribution do not approximate a stable ETF, so the ratio $\|\mathbf{z}_{:d}\|/\|\mathbf{z}\|$ carries no discriminative signal. The failure here is a direct consequence of the training regime, not of the scoring criterion itself.

In the best network, NECO reaches 0.48 on SVHN and 0.60 on textures : above random and competitive with softmax on textures, but below the geometric methods. The limiting factor is architectural rather than methodological. NECO was designed and validated on large models where the penultimate space is 768- or 2048-dimensional, and neural collapse concentrates class structure into a small fraction of dimensions while leaving a meaningful residual subspace. With a 64-dimensional feature space and 100 classes, the ETF under perfect collapse requires 99 dimensions, nearly saturating the available space. The leading 10 PCA components consequently capture a smaller fraction of the variance differential between ID and OOD samples. A model with a wider penultimate layer, or the same architecture trained on a simpler dataset with fewer classes, would likely close this gap considerably.

The contrast with Mahalanobis and ViM is instructive : both methods exploit feature-space geometry class-conditionally (Mahalanobis) or via full-covariance eigendecomposition (ViM), and therefore adapt to whatever structure the 64-dimensional space actually contains, however limited. NECO's stronger structural assumption is its weakness here, but that assumption is well-justified in the large-model regime for which the method was intended.

5.1 Neural Collapse Analysis

Figures 7–14 report the four collapse metrics for both networks on the CIFAR-100 training set.

NC1 - Within-class variability collapse. The best network achieves an NC1 ratio of 1.05 (mean within-class variance 44.1), meaning within-class and between-class variances are nearly equal. This is far from the ideal NC1 $\to$ 0, but represents a meaningful compression relative to the collapsed network, which reaches a ratio of 11.43 (mean 827.6), within-class spread is more than an order of magnitude larger than the separation between class means. The inverted training regime produced the opposite of collapse : features within each class are widely dispersed and largely undifferentiated.

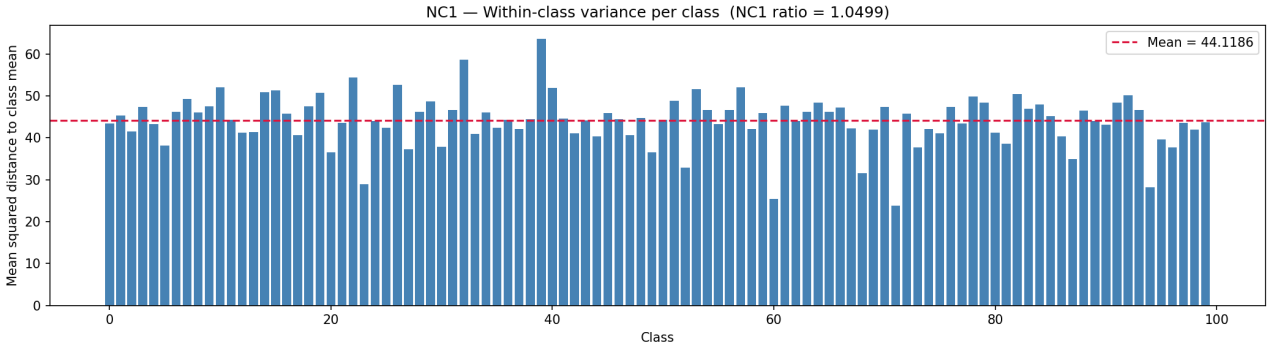


FIGURE 7 – Best Network NC1 - Within Class Variance

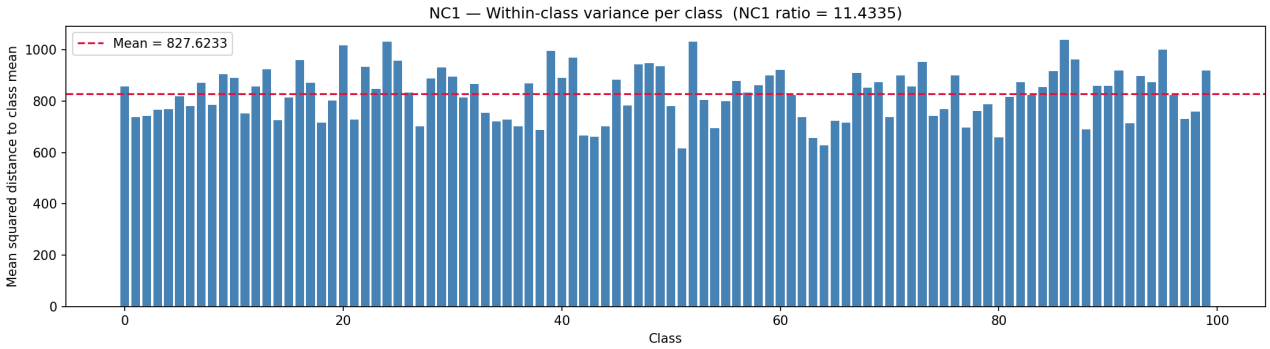


FIGURE 8 – Collapsed Network NC1 - Within Class Variance

NC2 - Convergence to simplex ETF. Under a perfect ETF with $C = 100$ classes, all pairwise cosine similarities between centered class means should equal $-1/(C-1) \approx -0.0101$. The best network's observed mean is -0.0072, close to the ideal, but the distribution is broad and left-skewed, spanning $[-0.5, 1.0]$, the class means point in roughly the right average direction but have not collapsed to equiangular geometry. The collapsed network's distribution is symmetric and nearly uniform around -0.003, consistent with class means placed nearly at random in feature space with no ETF structure.

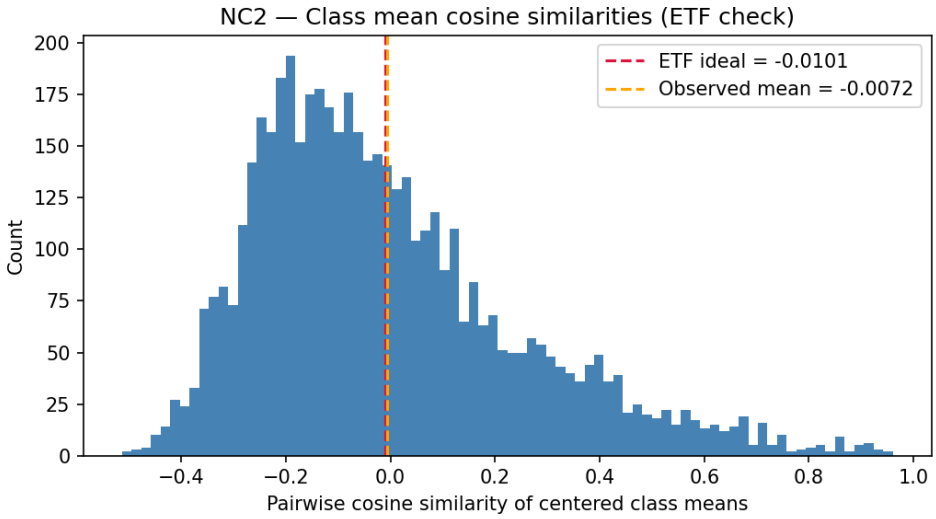


FIGURE 9 – Best Network NC2 - Class mean cosine similarities

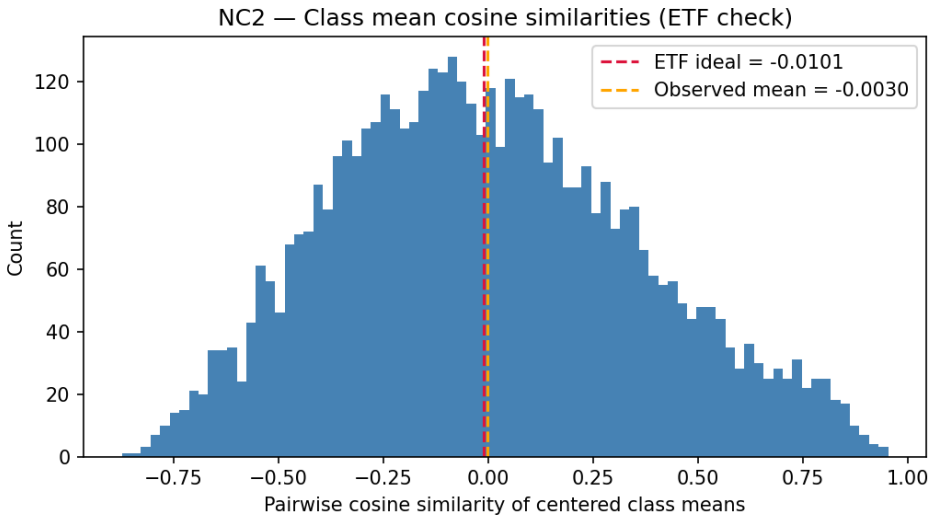


FIGURE 10 – Collapsed Network NC2 - Class mean cosine similarities

NC3 - Self-duality of weights and means. The most diagnostic result is NC3. In the best network, $W[c]$ and μ_c are positively aligned across all classes (mean cosine similarity 0.46), indicating partial but consistent self-duality : the classifier has learned to use the same directions for decision boundaries as the feature extractor has used for class structure. In the collapsed network every bar is negative (mean -0.16), meaning classifier weights point *away* from the corresponding class means. This is a direct consequence of overfitting without augmentation : the network memorises training samples by pushing features toward large-logit directions while the classifier weights rotate to fit the memorised configuration, decoupled from any geometric coherence with the mean.

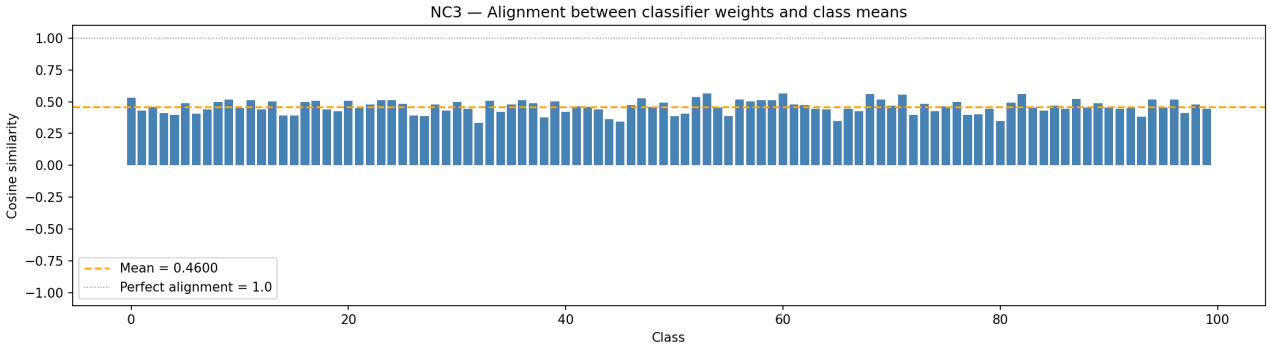


FIGURE 11 – Best Network NC3 - Alignment between classifier weights and class means

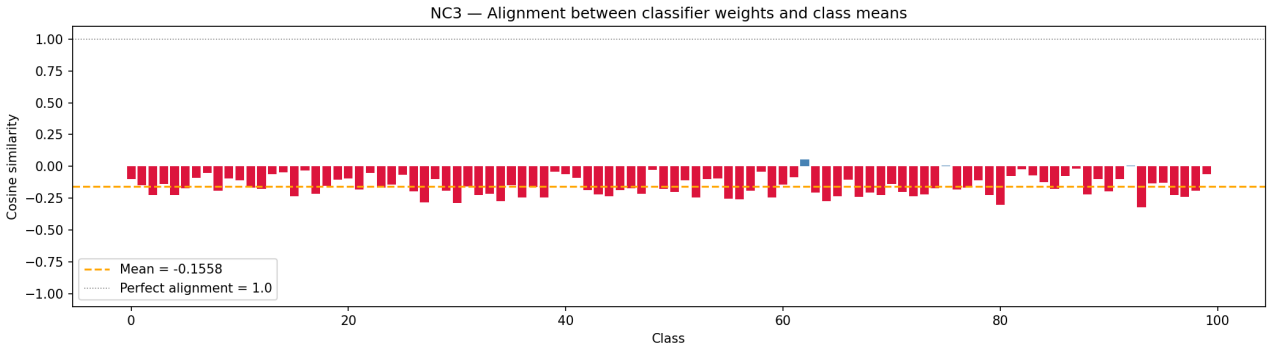


FIGURE 12 – Collapsed Network NC3 - Alignment between classifier weights and class means

Pairwise class mean distances. The best network's distance heatmap is visually more uniform (inter-class mean $\approx$ 7–9, max $\approx$ 14), with a few outlier classes (notably near index 50) that are unusually distant from all others. The collapsed network shows higher absolute distances (max $\approx$ 25) with stronger stripe artefacts, reflecting that some classes have been pushed to extreme corners of feature space by memorisation while others remain tightly clustered near the global origin. Neither heatmap approaches the uniform off-diagonal structure expected of a perfect equidistant ETF.

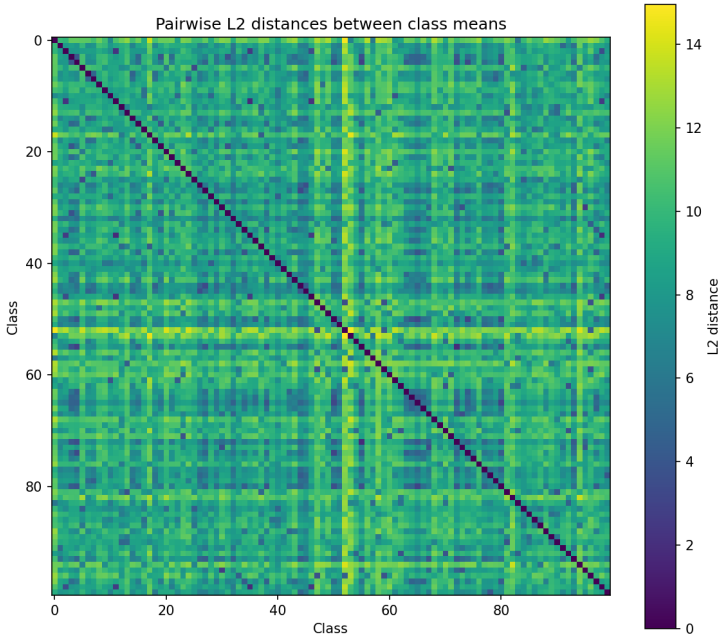


FIGURE 13 – Best Network L2 distances between class means

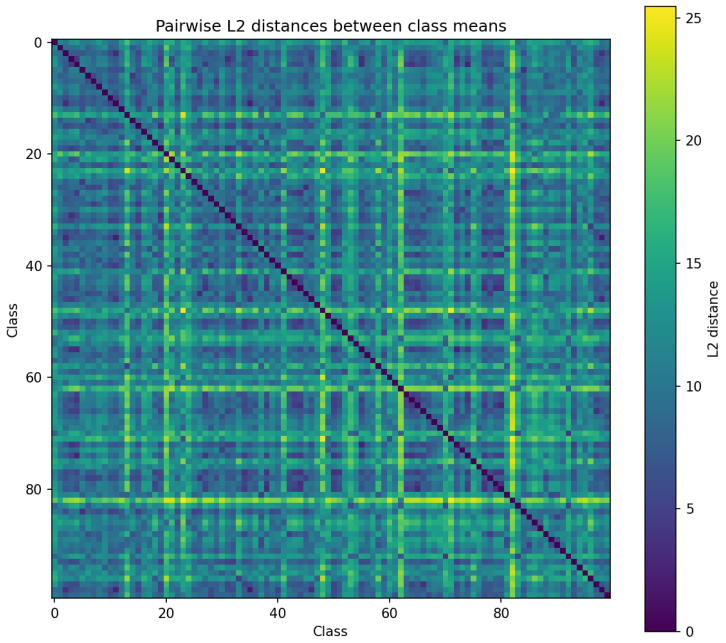


FIGURE 14 – Collapsed Network L2 distances between class means

Summary. Taken together, the metrics confirm that the best network exhibits only partial neural collapse, NC2 mean and NC3 alignment are in the right direction but far from the ideal values, and NC1 has not approached zero. The collapsed network, despite being trained specifically to drive training error to zero, fails on all four criteria, demonstrating that zero training loss achieved through memorisation without generalisation does not produce neural collapse. Collapse is a property of the learned representation, not merely of training accuracy.

6 Neural Collapse on Earlier Layers (BONUS)

6.1 Motivation and Architectural Context

In our study of Neural Collapse (NC), it is important to clearly separate two things : the main properties of the phenomenon (NC1 through NC4) and the different layers of the neural network. NC1–NC4 describe the geometric and statistical patterns that appear at the end of training. In contrast, "earlier layers" refer to the middle parts of our ResNet model (for example, `layer1`, `layer2`, and `layer3`), before the final average pooling and fully connected classification layer.

6.2 Theoretical Expectations for Intermediate Representations

Based on how deep neural networks learn, we expect that earlier layers slowly improve the meaning of features instead of showing an immediate geometric collapse :

— **No Variability Collapse (NC1) :** In our experiments, we measured the ratio between within-class variance (Σ_W) and between-class variance (Σ_B) at the final layer. If we extract features from earlier residual blocks, the within-class variance Σ_W would still be much larger. The features of samples from the same class are still spread out, so NC1 $\to 0$ does not happen in early layers.

— **No Simplex ETF Structure (NC2 & NC3) :** The Simplex Equiangular Tight Frame (ETF) appears because the network minimizes cross-entropy loss at the final classification layer. Earlier layers are not directly connected to the classifier weights, so their class means are not forced to form an ETF. They also do not show self-duality (NC3).

6.3 Implications for NECO and OOD Detection

Our NECO (NEural Collapse-based Out-of-distribution detection) score measures how much of a sample's feature norm is explained by the first d_{NECO} principal components of the training data. This method uses the ID/OOD Orthogonality (NC5) property, where OOD samples move toward the null space of the main subspace formed by the collapsed ID features.

Using NECO on earlier layers would not work well for several reasons :

— Without strong NC2 in early layers, ID data does not clearly lie in a well-defined $C - 1$ dimensional Simplex ETF subspace. As a result, the principal components used in NECO would mostly capture noise and within-class variation instead of clean class-mean structure.

— Because the ID features are not well organized, the NC5 property (ID/OOD orthogonality) does not hold. OOD samples (such as the SVHN or Textures datasets we tested) would not consistently project into a clear orthogonal null space in intermediate layer features.

7 Mechanistic Interpretability (BONUS)

8 Conclusion

In this project, we studied Out-of-Distribution (OOD) detection and Neural Collapse using a ResNet-18 model trained on CIFAR-100. We implemented two versions of ResNet and observed that the architecture adapted for CIFAR performed much better than the original ImageNet version. Although our best model reached moderate accuracy, it was limited by training time and computational constraints.

We tested several OOD detection methods, including softmax, max logit, energy, Mahalanobis, ViM, and NECO. The results showed that model quality strongly affects OOD detection performance. The network trained to generalize performed much better than the collapsed (overfitted) network. In particular, Mahalanobis and ViM were more robust in difficult cases, while softmax-based methods were less reliable when the model was overconfident.

We also analyzed Neural Collapse properties (NC1–NC4). The best model showed partial signs of Neural Collapse, especially in the alignment between classifier weights and class means (NC3), but it did not fully reach the ideal theoretical structure. The collapsed network, despite achieving low training loss, did not exhibit Neural Collapse. This shows that Neural Collapse is linked to meaningful generalization, not just zero training error.

Finally, we evaluated the NECO method, which is based on the NC5 (ID/OOD orthogonality) hypothesis. NECO did not perform as well as expected in our setting, mainly because our feature space was only 64-dimensional and the dataset had 100 classes. This limited the ability of the model to form a strong simplex structure. Our results suggest that NECO is better suited for larger models with higher-dimensional feature spaces.

Overall, this work helped us better understand the relationship between representation geometry, Neural Collapse, and OOD detection. We learned that good feature structure and generalization are essential for reliable OOD detection, and that architectural capacity plays an important role in achieving the theoretical properties described in the literature.

Références

[1] Mouïn Ben Ammar et al. *NECO: NEural Collapse Based Out-of-distribution detection*. 2024. arXiv : 2310.06823 [stat.ML]. URL : https://arxiv.org/abs/2310.06823.

[2] Mouïn Ben Ammar et al. *NECO: Neural Collapse Based Out-of-Distribution Detection*. https://gitlab.com/drti/neco. Original implementation of NECO.

[3] facebookarchive. *fb.resnet.torch GitHub Repository*. https://github.com/facebookarchive/fb.resnet.torch. Archived Torch implementation of ResNet architectures from "Deep Residual Learning for Image Recognition". 2019.

[4] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv : 1512.03385 [cs.CV]. URL : https://arxiv.org/abs/1512.03385.

[5] Kaiming He et al. *deep-residual-networks GitHub Repository*. https://github.com/KaimingHe/deep-residual-networks. Official repository for Deep Residual Networks code; associated with "Deep Residual Learning for Image Recognition" ResNet models. 2015.

[6] Weitang Liu et al. *Energy-based Out-of-distribution Detection*. 2021. arXiv : 2010.03759 [cs.LG]. URL : https://arxiv.org/abs/2010.03759.

[7] Alan Martyn. *PyTorch ResNet Implementation*. https://www.alanmartyn.com/. Open-source ResNet implementation in PyTorch.

[8] Vardan Papyan, X. Y. Han et David L. Donoho. "Prevalence of Neural Collapse during the terminal phase of deep learning training". In : *CoRR* abs/2008.08186 (2020). arXiv : 2008.08186. URL : https://arxiv.org/abs/2008.08186.