ANNs' Capability to Capture Neuronal Dynamics in the Primate Visual System

Echo Liu, Gargi Kher, Joshua Leibold, Makoto Take, and Zidan Luo

December 2020

1 Introduction

One of the basic core functions of our brain is to capture external stimuli and do something with that information. Our cortical sensory systems accomplish this for a wide variety of tasks: in human audition, sound waves are encoded by our cochlea into electrical signals that our brain creates meaningful words and sentences from. This paper will focus on our visual system, where photons are encoded by our retina into electrical signals that our brain creates rich object-centric scenes from. It is well understood that for all of these stimulus-response systems, each have a series of anatomically distinguishable, but connected areas of the brain responsible for their task [6] [14] and that our brain responds to new stimuli in a cascade of simple neural operations such as weighted linear sums of inputs or nonlinearities such as activation thresholds and competitive normalization [4] down these connected areas.

One major problem that arises is that a series of these linear-nonlinear (LN) transformations can give rise to much more complex nonlinear transformations, meaning that after traversing several brain areas and thousands of neurons, the possible space of the resulting LN transforms is massive. For our problem of vision, our goal is to identify which exact transformations the true biological circuits are using along the ventral stream. A solution to this problem ultimately means creating an encoding model for each sensory system.

1.1 Core Problem: Towards a Complete Encoding Model for our Cortical Sensory Systems

An encoding model is an algorithm that takes the same stimulus as the sensory system (like a pixel map for vision) as input and outputs a correct prediction of the neural response to that stimulus.

In [27], Yammins and DiCarlo identified three criteria that any encoding model of a sensory cortical system should meet:

1. Stimulus-compatibility: the model should be able to accept arbitrary stimuli across the domain of possible inputs.

   A complete encoding model of our visual system should be able to accept a pixel map that has the same resolution and size as our retinal field of view.

2. Mappability: The model should have components that correspond to true biological components of the neural system.

   In our case, an encoding model should have components that match to each part of the ventral stream, from the retina, to the LGN, V1, V2, V4, and IT.

3. Predictivity: Arbitrarily chosen neurons in any region of the mapped neural system should have corresponding units in the model that have detailed predictions of stimulus-by-stimulus responses.

   Our encoding model of the visual system should be able to accurately describe the broad representation our ventral stream creates from arbitrary visual stimuli as well as the granular response of an individual neuron or group of neurons in any region like the IT or V4.

We are still quite far from a truly complete encoding model of our visual system, so our best efforts have been focusing on solving associated sub-problems of our visual system, such as facial recognition or developing object recognition for the task for driving. The goal of [3] is to create a better encoding model for our visual system's ability to detect objects.

1.2 Evolution of Artificial Neural Networks

Inspired by Hubel and Wiesel [9] initially, with many advances over the past years [19], Neural Networks (NNs) have become a powerful tool to perform many human like tasks that traditional algorithms have not been able to emulate.

In [3], everything revolves around the ability to perform an object recognition task which is best performed by a class of NNs called "Deep Convolutional Neural Networks" (DCNNs). (Fig. 1).

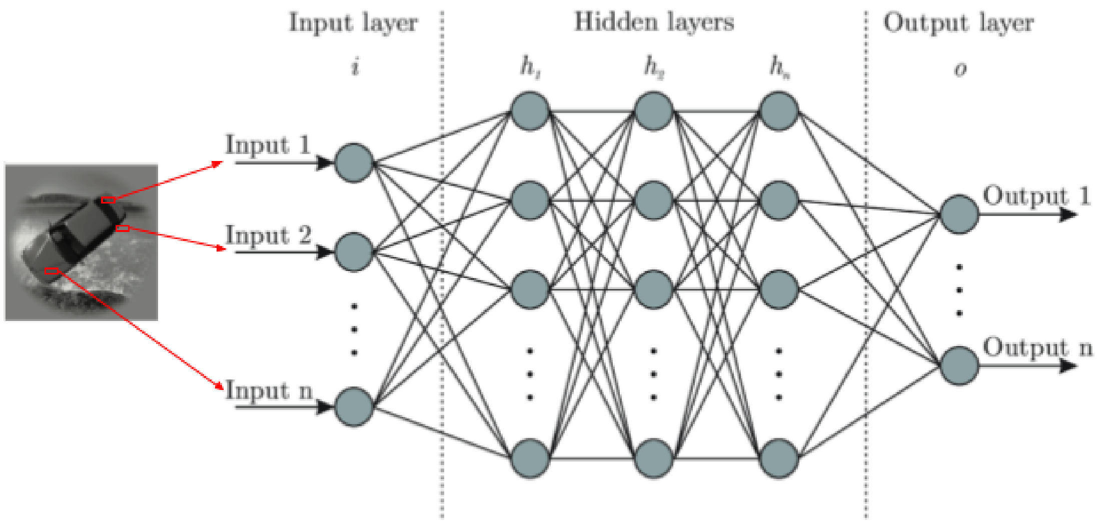


Figure 1: General Architecture of an Artificial Neural Network [2]

DCNNs are like a black box that you feed an image, then after a lot of thinking and processing of this input, outputs what it thinks that image was (Fig. 1). Let's look at this step by step. First, pixel values of images are put into the first (input) layer of the neural network as inputs x. As in (fig. 1) there are n input nodes, or input neurons, that each receive a different part of the image. These input values are processed at the input layer and fed into the first of many hidden layers, hence the "deep" part of DCNNs. These hidden layers are where information processing occurs, which comes down to how the activation from one layer brings on the activation of the next layer. The exact calculations that occur at each layer and at each neuron varies and are a distinguishing feature of different DCNNs, but the activation is generally described by Equation 1

$$y_j^k = f\left(\sum_{i=1}^{n} w_{ij}^k x_i^k + b_j^k\right) \tag{1}$$

If we look at the output of a generic neuron j in the hidden layer k, the output y (Eqn.1) is computed by performing an activation function f on weighted sums plus some bias [2]. There are several conventions for choice of the activation function, such as the "ReLu" and sigmoid, functions but this choice varies between DCNNs. Besides, the weights $w_i's$ tell us what pixel patterns each neuron in the specific layer is taking up on, and the bias $b_i's$ determines how high the weighted sum should be before the neuron starts to get meaningfully accurate. Then coming to the output layer, we have the activation values of each node at the penultimate layer to help us determine what object category the input image belongs to.

1.3 Bringing it Together

DCNNs have advanced significantly in their ability to do object recognition tasks and are approaching human like accuracy. The kicker is that DCNNs have emerged as some of the best encoding models for our own ability to perform object recognition. The results from [3] helped inspire what [27] describes as the "Goal Driven Approach" to neural modeling. The idea is that any neural network encoding model for a given sensory system will have to be effective at solving the behavior tasks that the sensory system supports. The approach is to first optimize network parameters for performance on an ethologically relevant task, and after the parameters are fixed, compare the network to neural data collected in solving the same task. The big question for this approach is whether or not such top-down goals strongly constrain biological structure. Will performance optimization imposed at the outputs of a network be sufficient to cause hidden layers in the network to behave like real neurons? [3] was one of the first results that showed that this might just be the case.

It still remains unclear whether such models are capturing the neuronal dynamics of the primate visual system. Some attempts have been made to bridge the gap between the model and the biological system, but there are limitations in their experiment setup.

2 2014 Paper

2.1 Experimental Set Up

2.1.1 Defining the Task: Core Visual Object Recognition

One amazing aspect of our visual system is the ability to detect objects in a very short period of time even when exposure of stimulus is brief. This is a generalization of the sub–problem [3] focuses on, formally referred to as "core visual object recognition". The natural basis for this sub–problem arises from trying to understand how our eyes fixate on a scene. Natural vision when observing a scene over a longer duration proceeds as a series of saccades (a quick, simultaneous movement of both eyes) at a rate of one saccade per 200-250 ms [4]. Therefore, the unit measurement of fixation is our ability to detect objects within one saccadic fixation without eye movement, contextual influence, or shifts in attention, which put together gives us the definition for the core visual object recognition task.

2.1.2 Core Object Recognition with DCNNs'

There are three DCNNs that have been examined by the authors because of the state-of-the-art performance on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) datasets: The Krizhevsky et al. 2012 (AlexNet) [10], Zeiler & Fergus 2013 (ZF Net) [29], and Yamins et al. 2014 (HMO) [28]. (We provide explainations in Supplementary Materials 1) Due to the rapid development of the DCNNs, the results have been improved continuously on this challenge since the paper has been published.

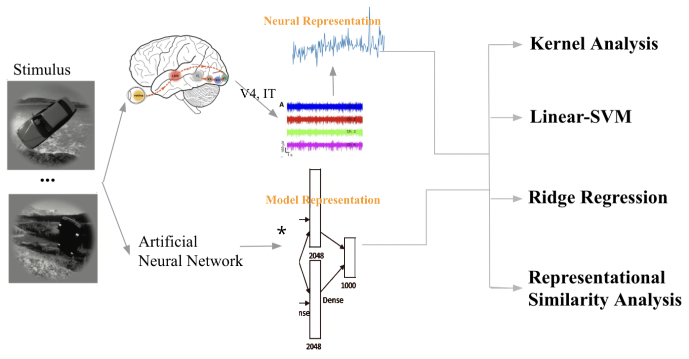


Figure 2: A Unifying Metric between Representations of Deep Neural Networks and Primate Visual IT Cortex

2.2 Unifying Metric

[3] attempts to address four main concerns that previous works have not taken into account, and propose a unifying metric to analyze how close the modern DCNNs are to representing objects in the same way that the visual cortex does. They claim that the DCNNs they looked at outperform previous models at this task. First, they account for the experimental limitations that arise when recording neural activity, making sure to introduce noise into the model representations, so that they can be fairly compared to the neuronal recordings. They also noted that other papers used models which used object classifiers whose boundaries were of a fixed complexity. What exactly this means will be addressed in the following section.

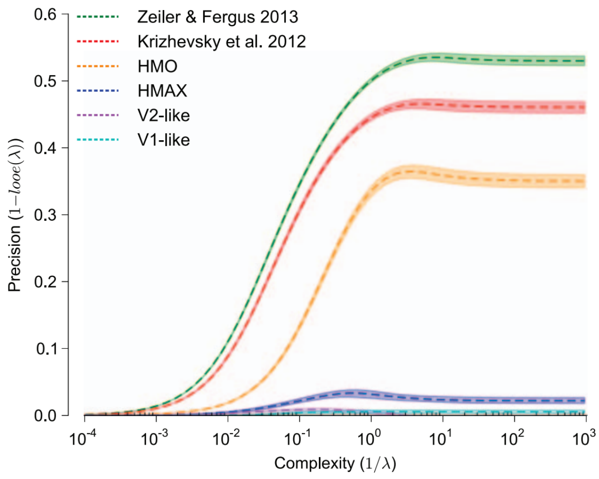


Figure 3: The paper demonstrates an interesting connection between the complexity and the precision of the neural networks' decision boundaries. Previous papers would have used a classifier which used only a single complexity (one of the values on the x-axis) instead of the much larger span of these values.

Knowing there to be a correlation between the accuracy of the classifier and its complexity, the team chose to utilize a modified version of kernel analysis which allowed variation to the complexity of each model and measure model and neuronal performance as the complexity increased. From this, they produced Fig. 3, which we attempted to recreate. Our recreation will be discussed in a later section. Third, the team analyzed the similarity of the models to the IT recordings and compared the IT variance represented by the models. Their result is shown in Fig. 4.

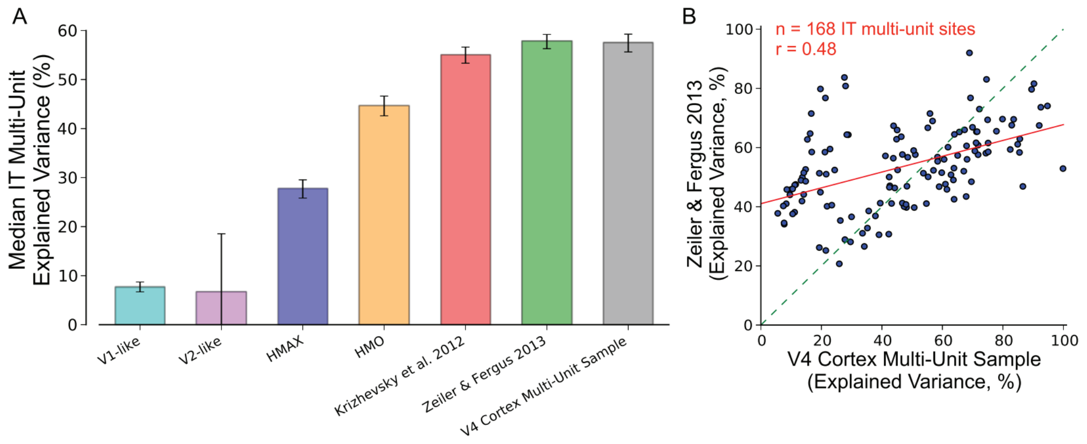


Figure 4: The paper seeks to find out how much of the neuronal recording variance is captured by each representational model (A) and also uses the V4 cortex multi-unit sample to predict the mean explained variance of each IT multi-unit site (B).

Lastly, the team mentions they made use of an image set with a high degree of variation, which allowed them to accurately identify models with different performance. We will explain these four main techniques this paper used to answer their main question, present our recreation of this data with modern neural networks, and discuss the questions we raised that extend the ideas of the DiCarlo paper.

2.3 Kernel Analysis

Kernel analysis is a procedure that plays a key role in comparing the performance and complexity of the representational neural network models with each other. As mentioned before, previous simulations used fixed complexity classifiers, which doesn't capture how precision can be affected by complexity. Generally, kernel analysis considers an object's features as points in an input space $\mathbb{R}^n$, and attempts to project this data into another space $\mathbb{R}^m$ (usually $m > n$), where the data can be easily partitioned by a plane or hyperplane [11] (this is shown by Fig. 5). In [3], the input consists of a set of computer-generated images which fall into one of seven categories (Cars, Fruits, Animals, Planes, Chairs, Tables, and Faces). The main goal of using kernel analysis in [3] was to find a model that separated these seven distinct classes as accurately as possible, reducing its generalization error during classification.

Each model that underwent kernel analysis used a function $\phi(x)$ to extract the important features of the image, which reduces the input size of the data into a more manageable set. The transformed data is plotted to $\mathbb{R}^m$, where it is labelled with the category that the image belonged to (whether it's an "Animal," "Plane," etc.). The model then needs to come up with a decision boundary, or a function representing how all seven categories should be split.

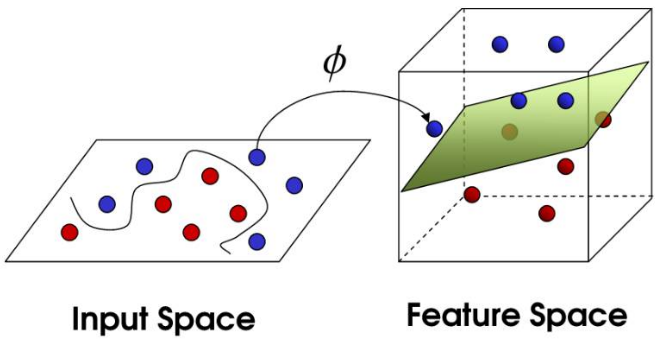


Figure 5: In this general representation of a decision boundary, data existing in $\mathbb{R}^2$ is transformed by ϕ into $\mathbb{R}^3$, where it is more easily separated by a plane. [15]

Fig. 5 shows input data consisting of two different classes in a two-dimensional plane. In this space, any line separating these categories (a decision boundary), will most likely be complicated because of how the points are arranged. The technique, then, is to transform the data into higher-dimensional space, and find a way of presenting the data in that space such that the decision boundary is simple. In Fig. 5's feature space, we can see that the decision boundary can now be represented by a plane. This shows that depending on the input data, the decision boundary can be simple or complicated. This is what determines the complexity of the model.

For our problem, the function ϕ is used to transform, or map, the input data X to their output classification values Y[16]. Using ϕ can only go so far when working with a large data set. For many applications, it can be computationally expensive to transform all of the input data. Because of this limitation, the team made use of the kernel, a matrix that allows us to make computations in the current dimension without moving to the higher space, in order to reduce computation time[8].

$$k_\sigma(x, x') = e^{(-||x-x'||^2/2\sigma^2)} \tag{2}$$

This paper specifically makes use of a standard Gaussian kernel which is centered over input features in a normal distribution. The resulting kernel matrix representing the kernels of all combinations of transformations is represented by Equation 3.

$$K_\sigma = \begin{bmatrix} k_\sigma(\phi(x_1), \phi(x_1)) & \dots & k_\sigma(\phi(x_1), \phi(x_n)) \\ \vdots & & \vdots \\ k_\sigma(\phi(x_n), \phi(x_1)) & \dots & k_\sigma(\phi(x_n), \phi(x_n)) \end{bmatrix} \tag{3}$$

Thus for a set of n inputs, the matrix performs a feature extraction, pulling the most important features from X and transforming them into a higher-dimensional space so the correct Y can be determined. Each kernel computes the similarity between each possible pair of vectors of extracted features. The closer they are in this space, the closer the value of $|||x - x'||$ will be to 0. This means that the value of the Gaussian kernel is close to 1 when $\phi((x_n))$ and $\phi((x_m))$ are nearby, and close to 0 when they're distant.
In this paper, kernel analysis was set up to optimize the performance of each model. This optimization, known as a regularized kernel ridge regression, minimizes the generalization error (thus maximizing the classification accuracy). It can be represented as shown in Equation 4.

$$\min_{\Theta \in R^n} \frac{1}{2} \Sigma_{i=1}^n ||Y - K_\sigma \Theta||_2^2 + \frac{\lambda}{2} \Theta^t K_\sigma \Theta \tag{4}$$

Here, the regression and regulation parameters, Θ and λ, are introduced. In Equation 4 the minimization computes the Euclidean distance between Y and the regularized kernel $K_\sigma\Theta$, with smaller distances indicating more accurate representations. A high λ value acts as a "penalty" term, punishing the model for fitting exclusively to the training data and thus preventing overfitting. However, the model might not be as tightly fitted as it could be with an low value for λ. [1][20]. This penalty term will be explained in further detail when discussing ridge regression. Θ, the solution to this minimization problem, is represented by Equation 5, where I represents the identity matrix.

$$\Theta_\sigma^*(\lambda) = (K_\sigma + \lambda * I)^{-1} Y \tag{5}$$

This minimization problem is part of a process known as ridge regression, which serves to decrease the overfitting of the model. Ridge regression will be explained in detail in a later section. λ represents the inverse of the complexity of the model, because the smaller lambda gets, the more unconstrained the fitting becomes, making the model more complex. Ultimately, this minimization allows the model to attempt to solve the problem in Equation 6.

$$K_\sigma \Theta = Y \tag{6}$$

We can see by Equation 6 that Θ also affects the fitting of the model, and can be thought of as the weights of the neural network. The regularized kernel represented by $K_\sigma\Theta$ attempts to solve this system of equations so that every input feature has an output category [7]. The team performed this kernel analysis technique in ten trials, searching over a range of σ values that gave the lowest Θ. σ is known as the Gaussian kernel scale parameter, and can be thought of as the width of the

Gaussian distribution. Eventually, through this analysis, values for the model's complexity ($\frac{1}{\lambda}$) and precision ($1 - looe(\lambda)$) were found, with the generalization error being shown in Equation 9.

$$LOOE_{\sigma}(\lambda) = \frac{\Theta^*_{\sigma}(\lambda)}{diag((K_{\sigma} + \lambda I)^{-}1)} \tag{7}$$

In Equation 9, "$LOOE$" stands for the leave-one-out error. This is just one way to calculate the generalization error - it takes one of the points out of the dataset, trains the model, and computes the error on that point [5]. In the setup, the team calculated the $LOOE$ as an average over the seven categories. They computed the error for one category at a time and then averaged over all seven of these values. This mean squared error for $LOOE$ can then be written as Equation 8

$$looe_{\sigma}(\lambda) = \frac{1}{n} \sum_{i} (LOOE_{\sigma}(\lambda))^2 \tag{8}$$

It is important to be familiar with the $looe$ because of how it affects model precision, because being a measure of error, $1 - looe$ can be viewed as the precision. As mentioned before, the regularization parameter λ represents the inverse of the complexity of the model. Thus the higher the complexity, the more precise the model should become. In general, the authors determined that an ideal model would have a high precision for a simple decision function. This means that even if the regularization is high (meaning a high value for $K_{\sigma}\Theta$), the model still should have high accuracy for a small variation in the input data. A poor model, however, will generally need high variation in the input to create complex decision functions. Generally, the best performing models should have a high classification accuracy even when trained with a small amount of data. With the kernel analysis method implemented in this paper, it became possible to measure a model's performance in relation to its complexity.

Parameter	Definition
λ	regularization parameter
Θ	regression parameter
$\frac{1}{\lambda}$	complexity
$1 - looe(\lambda)$	precision
σ	Gaussian distribution scale factor
X	image features
Y	normalized labels
ϕ	feature extraction process
$K_{sigma}\Theta$	regularized kernel

Table 1: Parameters used in Kernel Analysis

2.4 Linear Support Vector Machine

The Support Vector Machine (SVM) is a classic machine learning technique used to solve big data classification problems [21]. In [3], Linear-SVM is used to test the generalization performance of object classification on neural and model representations. In this algorithm, each data point is plotted in $\mathbb{R}^n$, with n being the number of features. Then the classification is performed by finding the hyperplane that separates the two classes of the data [21]. Given a set of samples x_i for $i = 1, 2, \ldots, m$, where m is the number of samples, there are two classes associated with the sample (positive class and negative class). Let's denote $y_i = 1$ for the positive class and $y_i = -1$ for the negative class, respectively. Then the hyperplane is found by solving $f(x) = 0$ [12]:

$$f(x) = w^T x + b = \sum_{j=1}^{m} w_j x_j + b = 0 \tag{9}$$

where $w \in \mathbb{R}^m$ and $b \in \mathbb{R}$ are used to define the hyperplane.

Linear-Supported Vector Machine (LSVM) generalization performance of neural and model representations in Fig. 6 is measured in the paper [3] to verify the results of the kernel analysis procedure. The LSVM is trained on 80% of the images and tested on 20%. This procedure is repeated for 10 random sets of the training-testing split. The LSVM produces similar results as using kernel analysis, which reveals that the ZF net (Zeiler & Fergus 2013) representation achieves generalization comparable to the IT multi-unit neural sample for a simple linear decision boundary [3].

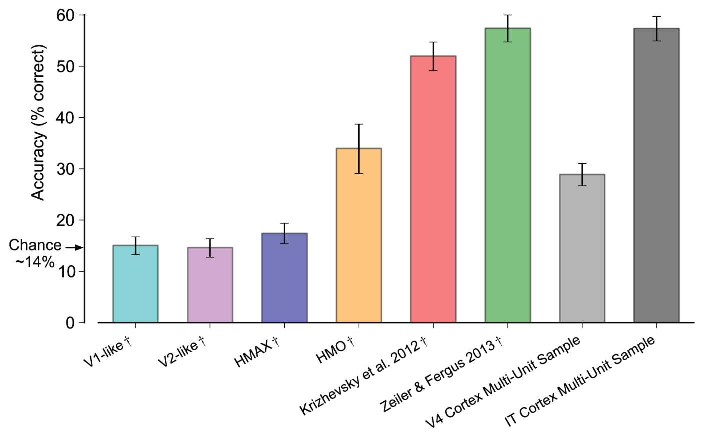


Figure 6: Linear-SVM generalization performance of neural and model representations [3].

2.5 Ridge Regression

In order to predict the IT multi-unit recordings using the model representations, ridge regression was performed. Ridge regression is one form of linear regression, a technique used to find the best-fitting line between related points of data[25][20]. Linear regression can be said to fit data (a set of points) to the slope intercept Equation $Y = mX + b$, where X is the independent variable and Y is the dependent variable [24]. Once again, X represents the input images while Y represents the category labels. An important measure in this technique is the explained variance, which captures how much of each datapoint's deviation from the mean is depicted by the relationship between X and Y [13].

$$\min_{w} ||X_w - Y||_2^2 \tag{10}$$

[3] specifically makes use of the explained variance to measure each model's performance. Over ten tries, linear regression models from each representation were estimated for 80% of the images. The remaining 20% were used for testing. IT multi-unit responses were also predicted by performing a linear regression on the V4 multi-unit responses. The regularized kernel ridge regression technique was visited in Section 2.3 in the context of Kernel Analysis, but [3] specifically used ridge regression to predict neuronal recordings from the model representations. Ridge regression is an extension of

linear regression in that it adds an extra penalty term. This term was mentioned in reference to kernel analysis.

$$\min_w ||X_w - Y||_2^2 + \alpha||w||_2^2 \tag{11}$$

Equation 10 represents a generic linear regression problem. Equation 11 represents a generic ridge regression problem, with X and Y still representing the independent and dependent variables, respectively. w represents the coefficients that comprise the best fit line. The parameter α acts as the penalty term in this case, decreasing the value of w and preventing the model from overfitting. When a model overfits, its coefficients tend to increase and match the training data. This prevents it from performing well on other data [1]. By adding α, the team ensured that the coefficients wouldn't get too large. By now, it is probably clear that the ridge regression minimization equation given above is nearly identical to the regularized kernel ridge regression function in Section 2.3. Equation 4 is equation 11, just more specific to the kernel analysis problem. With ridge regression, [3] was able to account for the variance in the IT multi-unit recordings, allowing them to determine the models that best represented neuronal responses.

Parameter	Definition
X	image features
Y	normalized labels
α	penalty term
w	weights

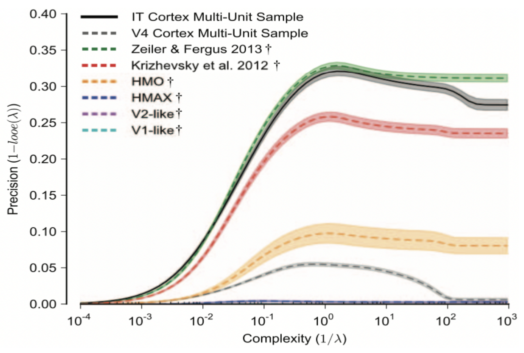
Table 2: Parameters used in Ridge Regression

3 Results

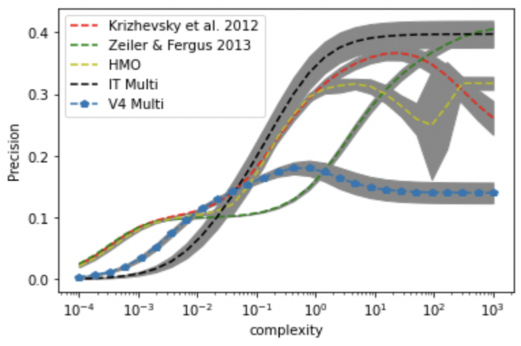
One way to understand a super complex system, such as the primate visual system, would be to study its equivalent bases [27], which in our case are the model representations that can be linearly fit to the neural representation. Thus, in our work, we aim to focus on two perspectives regarding the representations of images from DCNNs and the primate visual system. First, we are interested in whether the model representations can achieve near-human performance. Second, we want to see if a linear regression from the model representations to neural representations is possible. To this end, we use two methods presented in [3] to conduct our analysis, Kernel Analysis for our first objective, and Ridge Regression for the second.

3.1 Reproduction

To validate our novel results presented in later sections, we first aim to reproduce the same results presented in [3] as shown in Fig. 7a and Fig. 8a.



(a) Original



(b) Reproduced

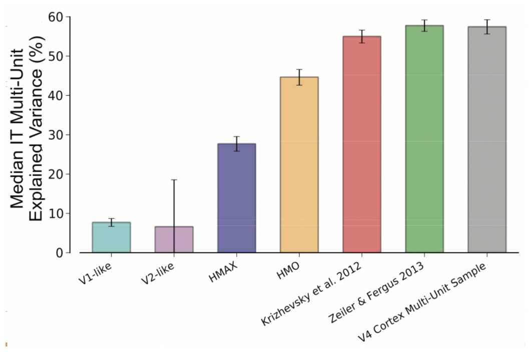
Figure 7: After performing kernel analysis, the precision and complexity of the model and neural representations was plotted. The "Original" plot has been taken directly from [3] and the "Reproduced" plot is our recreation.

3.1.1 Kernel Analysis

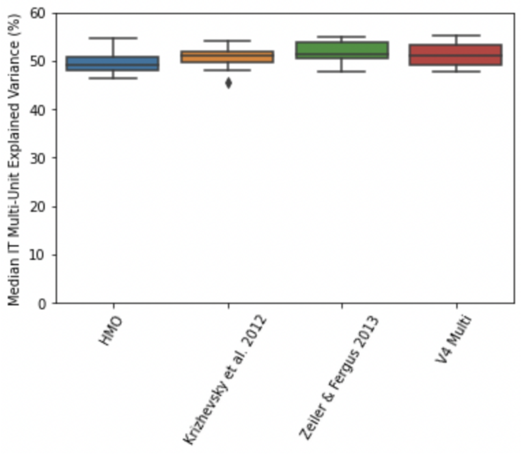
To test the generalization ability of the model representations, we sub–sample 80% of the 1960 total images for 10 times. During each time, we feed all those sampled images into the three models and obtain the penultimate layer's results as the model representations, which would be the $\phi(X)$ in Equation 3. Similarly, we choose the neural representations recorded when showing such images. For each category, we create corresponding labels for those sampled images, where 0 indicates the image does not belong to the category and 1 indicates the opposite, which is the Y in Equation 4. We then normalize the labels in such a way that the $looe_\sigma(\lambda)$ in Equation 8 would be 0 if every predicted label is wrong, and it would be 1 if every predicted label is correct.

Since we are interested in the precision with different level of regularization, i.e. the larger λ is, the smaller the model's complexity is, we loop through different values of λ when optimizing Equation 4. To eliminate the influence of σ, we also loop through an array of σ, when conducting the optimization for a fixed σ value, and the smallest $looe_\sigma(\lambda)$ is our defined optimized result. After optimizing for each category's labels, we then average the values of $looe(\lambda)$ among different categories.

Fig. 7b shows our reproduction results regarding the performance versus the complexity of the model that uses the model and neural representations as inputs. As discussed earlier, we expect the improvement of performance as the complexity, $\frac{1}{\lambda}$, increases, or rather regularization, λ decreases. But, after a certain point, over-fitting would limit the performance, resulting in the flat curve in the end. Both dynamics in Fig. 7b match up with Fig. 7a. Although Fig. 7b and Fig. 7a do have some small differences, we suspect it is because of the sub–sampling procedure, which gives us different image samples from the ones sub–sampled by the DiCarlo Group.



(a) Original



(b) Reproduced

Figure 8: The neural and model representation predictions of IT multi-unit responses were plotted. The "Original" plot has been taken directly from [3] and the "Reproduced" plot is our recreation.

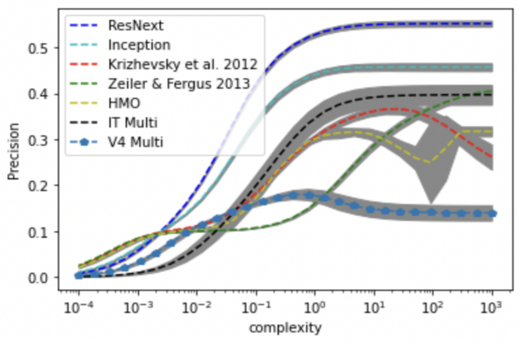
3.1.2 Ridge Regression

Using the Ridge Regression method, we are interested in how much variance of the neural representations is explained by the model representations. To do so, we want to construct a model that takes in the model representations for a certain image and returns the corresponding neural representations. We use 80% of the images to obtain the model and neural representations that are used to train a linear model. The model and neural representations for the rest of the images are then used to test how good the model is by calculating how much variance is captured. We conduct this cross-validation 10 times and during each time, we choose the median explained variance among 168 sites of IT as our result.

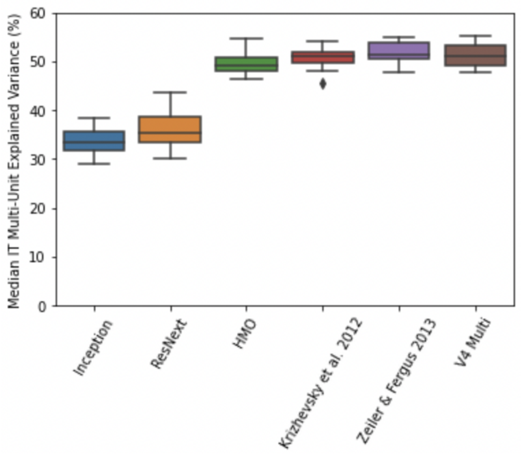
In Fig. 8b, the four box plots represent the reproduction for the last four bar plots using HMO, AlexNet, ZFNet, and neural representations of V4 in Fig. 8a (we used box plots since we think it would show the distribution of median explained variance better among 10 cross-validation iterations). We can see the reproduction is close to the original results.

3.2 Our Novel Results

We noted that DCNNs have experienced significant advancements since [3] was published in 2014, which show much improved performance for the object classification task. However, it is unclear if the improved DCNNs will more accurately explain the dynamics of certain areas of the primate visual system. We were interested in exploring how [3]'s conclusions about the comparisons between the performances of various DCNNs and neuronal responses might change or be validated in the face of more modern DCNNs. There were other candidates, with one in particular, that fit the question we wanted to explore. We will later go more into detail about why we chose this particular question. For now, we will focus on how we reproduced the conclusions of this study as well as how we updated the study with modern methods.



(a) Kernel Analysis



(b) Prediction for IT multi-unit

Figure 9: Kernel Analysis Curve of Model and Neural Representations & Neural and Model Representation Predictions of IT Multi-Unit Responses

Based on the results shown by Fig. 9a and 9b, we came to the conclusion that the more advanced neural networks appear to complete the core visual object recognition better, with higher precision values for all values of $1/\lambda$. [3] considered a neural network to be better if it could perform highly accurate decisions for varying levels of complexity. These networks are more effective at object-recognition tasks, and this increased performance is reflected in their ability to perform well while changing complexity.

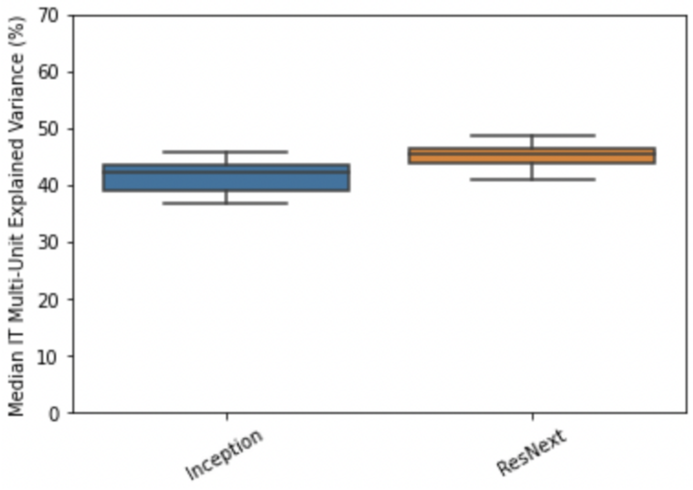
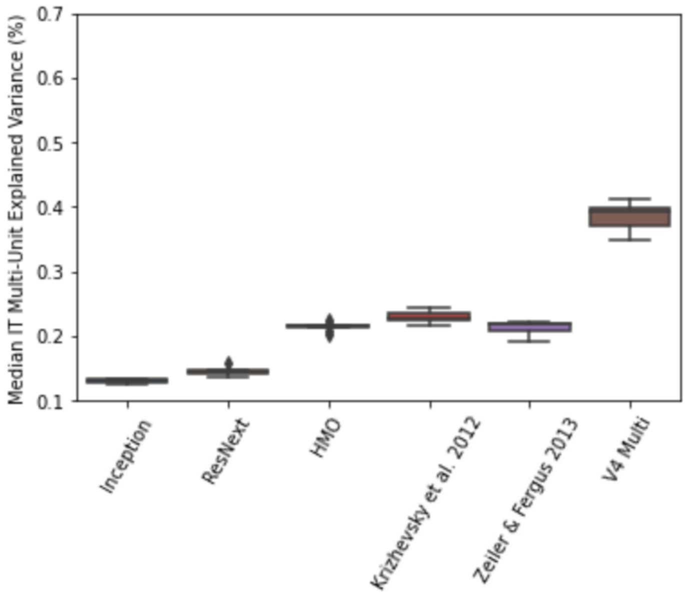


Figure 10: Neural and Model Representation Predictions of V4 Multi-Unit Responses

We also see that the median IT multi-unit explained variance underwent a clear decrease for the more modern neural networks. This isn't the result we expected to find. We had expected that as neural networks improved, the explained variance would show improvement. However, we are curious to see if the newer neural networks would fit other sites in V4 or even earlier sites in the ventral stream better. Fig. 10 show an increase in explained variance for both ResNext and Inception Neural Networks. With further research on the structure of ResNext, we found out that the residue structure actually mimics the ventral stream structure where V5 and V6 receives direct connections from V1 [23]. We have some thoughts on how we can improve this in a later section.

Another subtle part of [3] that we noticed was that they used ridge regression to predict the IT multi-unit responses from each of the model representations but we couldn't find a justification as to why this was the particular linear model analysis technique they chose to use. Recent work done by the same group [17] has continued exploration into the goal-driven approach to find better DCNN models that can best predict our neural dynamics. We noticed that this recent work from 2018 used comparable neural data on a similar object recognition task and calculated the trial-averaged firing rate for each electrode for each image to be the neural representation that they fit their new models to. The linear model analysis technique they used this time was Stochastic Gradient Descent (SGD) with standard L2 loss. Unfortunately, they were also not clear as to why they chose to transition to SGD to fit the neural data in their more recent work. However, we decided to test how our previous results of predicting IT responses with Inception and ResNext and their reproduction of IT response prediction would change if we instead used SGD. These results are presented in 11a. Interestingly, the explained variance across models is significantly lower with a much smaller standard deviations. We were not able to properly understand the differences between ridge regression and SGD, but it is very interesting that even though absolute performance was worse with SGD, the relative performance in prediction IT response of Inception and ResNext was still worse than the other three feed-forward models.



(a) Reproduced

Figure 11: Neural and Model Representation Predictions of IT Multi-Unit Responses fit with Stochastic Gradient Decent (note: y axis labels are in decimal form, not percentage)

4 Conclusion

4.1 Open Questions and Future Discussions

One main question we still have is if comparing the neural and model representations directly is a good way to understand the brain or ANNs. We thought that the metric results were counter-intuitive: the primate visual system should perform much better than the model - not the other way around - even for state of the art models. Yet this is the result that's presented in the paper.

Two directions exist that we consider major pathways of novel exploration. One is relatively easy to explore and accepts what the paper proposes, while the other requires a much more in depth approach and considers the legitimacy of what the paper assumes. For this reason, we decided to label these as the "Blue Pill" and "Red Pill" questions, since the choice between them bears resemblance to a choice made by Neo in the 1999 movie "The Matrix."

4.2 Blue Pill

By taking the Blue Pill, we assume that [3]'s method of creating a unification method is a proper and repeatable technique to compare the performance between DCNNs and our brains' ability to complete the same task. Then, the natural question to ask is: if we use the unifying metric to compare the state of the art DCNNs' to neural recordings, would we see similar results? This question was what we explored as our novel question. Another approach that we wanted to pursue was to find a comparable but more comprehensive data set for neural recordings, with more recording sites and more images presented, and with presumably less noise with more modern electrodes. Since this data set only contained information from V4 and the IT, we would have liked to see if data collected from areas even further upstream like the LGN and V1 could be predicted with even earlier DCNNs layers. Unfortunately, finding such a data set proved too difficult. This isn't exactly an adventurous question, since this is essentially reproducing the results with modern methods. But answering it would give us some insight into whether the resemblance between neural networks and actual, biological representations has grown closer as networks have advanced, or if we've reached a point at which the basic architectures of computational networks and biological networks demand different ways of representing data, and must diverge.

Due to an absence of significant computing power, we only performed this comparison on the three highest-performing networks available to us. This comparison could certainly be extended to a larger sample of high-performing networks, as well as networks unavailable publicly. Our conclusions about our data might be valid for this small set of neural networks, but too much of a generalization for others. This is a path of exploration whose only obstacle is obtaining the resources to complete it, as opposed to the upcoming "Red Pill" question, which is a more unorthodox and difficult approach.

4.3 Red Pill

By taking the Red Pill, we reject [3]'s "reality" of the proposed unifying metric as a proper method of comparison. Our blue pill results indicate that their unification method is incomplete, since we did not observe consistent results with modern DCNNs. Moving forward, we hope to dig deeper into what an ideal unification metric should be and what aspects are missing from the one proposed here. When proposed, [3] claims that parts of their unification metric had been accounted for in past endeavors, but claimed that they were the first to account for all four. Naturally, it's our job to discover what additional areas exist that a new comparison would need to account for.

4.4 Final Thoughts

Our recreations didn't reflect the results shown by [3], and there are a few reasons we think this may be. First, we think one reason our recreation was different was due to how we modified the models we tested. In [3], the authors pre-processed the test images by resizing the images and cropping or flipping them in a way that makes them suitable to the model. However, in our reproduction using the pre-trained models, we just followed the standard pre-processing method (described in S.5). To address this, one thing we could do the is train these models with the synthetic (test) images provided in [3], since it has been showed that the synthetic images trained models have a different performance than the ImageNet trained models [18].

Due to the limited computational power we had, we used pre-trained models unlike the models used in [3], which would have definitely contributed to the differences seen between our results and theirs. We also were not able to explore as many layers as models used in [3], which leads us to believe that if we made modifications to our models we would be able to see results more similar to theirs. For instance, we could increase the depth of our models and use different feature maps and layer connections.

Another aspect of this analysis that we can change is increasing the size of the image set. We used 1960 images just like [3] used in their methods. This also relates to the amount of computational power we had because it takes time to process a large set of images, but if we were to have this capability then perhaps we could get more similar results.

The most natural comparison to be made between neural network object recognition and visual cortex recognition is to simply the performance that each achieves. Neural networks still do not achieve the levels of primate object recognition but [3]'s metric suggest that they do, when classified with different levels of complexity. Based on this, we wonder if there is a way we can improve this formulation and capture the brain's responses with DCNNs. We were able to explore some red pill results by using a more modern approach on the neural representation fitting procedures using SGD instead of ridge regression. Because we were not able to fully understand the differences between the two techniques, we must be careful not to take these results as conclusive evidence that ridge regression over estimated the IT multi-unit response. However, it is very interesting that the relative performance of all models was preserved across fitting techniques and may support our proposed reasoning as to why these newer models were relatively worse at predicting explained variance than the older models.

We highly respect the work the DiCarlo group has done and are hope that our skepticism lead analysis presents a meaningful commentary to new potential areas of research for the Goal Driven Approach.

References

[1] Pedregosa et al. *Scikit-learn: Machine Learning in Python.* 2011.

[2] Facundo Bre, Juan M Gimenez, and Víctor D Fachinotti. "Prediction of wind pressure coefficients on building surfaces using artificial neural networks". In: *Energy and Buildings* 158 (2018), pp. 1429–1441.

[3] Charles F Cadieu et al. “Deep neural networks rival the representation of primate IT cortex for core visual object recognition”. In: *PLoS Comput Biol* 10.12 (2014), e1003963.

[4] Matteo Carandini et al. “Do we know what the early visual system does?” In: *Journal of Neuroscience* 25.46 (2005), pp. 10577–10597.

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

[6] Daniel J Felleman and David C Van Essen. “Distributed hierarchical processing in the primate cerebral cortex”. In: *Cereb cortex*. Citeseer. 1991.

[7] Jens Hainmueller and Chad Hazlett. “Kernel regularized least squares: Reducing misspecification bias with a flexible and interpretable machine learning approach”. In: *Political Analysis* (2014), pp. 143–168.

[8] Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. “Kernel methods in machine learning”. In: *The annals of statistics* (2008), pp. 1171–1220.

[9] David H Hubel and Torsten N Wiesel. “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex”. In: *The Journal of physiology* 160.1 (1962), p. 106.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[11] Bor-Chen Kuo, Cheng-Hsuan Li, and Jinn-Min Yang. “Kernel nonparametric weighted feature extraction for hyperspectral image classification”. In: *IEEE Transactions on Geoscience and Remote Sensing* 47.4 (2009), pp. 1139–1155.

[12] Yaguo Lei. “Individual intelligent method-based fault diagnosis”. In: Dec. 2017, pp. 67–174. ISBN: 9780128115343. DOI: 10.1016/B978-0-12-811534-3.00003-2.

[13] Urbano Lorenzo-Seva. “How to report the percentage of explained common variance in exploratory factor analysis”. In: *Unpublished manuscript* (2013).

[14] Rafael Malach, Ifat Levy, and Uri Hasson. “The topography of high-order human object areas”. In: *Trends in cognitive sciences* 6.4 (2002), pp. 176–184.

[15] Pranov Mishra. “Comprehensive Support Vector Machines Guide - Using Illusion to Solve Reality!” In: (2017).

[16] Grégoire Montavon, Mikio L Braun, and Klaus-Robert Müller. “Kernel Analysis of Deep Networks.” In: *Journal of Machine Learning Research* 12.9 (2011).

[17] Aran Nayebi et al. *Task-Driven Convolutional Recurrent Models of the Visual System*. 2018. arXiv: 1807.00053 [q-bio.NC].

[18] Rishi Rajalingham et al. “Large-scale, high-resolution comparison of the core visual object recognition behavior of humans, monkeys, and state-of-the-art deep artificial neural networks”. In: *Journal of Neuroscience* 38.33 (2018), pp. 7255–7269.

[19] Waseem Rawat and Zenghui Wang. “Deep convolutional neural networks for image classification: A comprehensive review”. In: *Neural computation* 29.9 (2017), pp. 2352–2449.

[20] Ryan M Rifkin and Ross A Lippert. “Notes on regularized least squares”. In: (2007).

[21] Shan Suthaharan. “Machine learning models and algorithms for big data classification”. In: *Integr. Ser. Inf. Syst* 36 (2016), pp. 1–12.

[22] Christian Szegedy et al. “Rethinking the inception architecture for computer vision”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[23] Alex Thomson. “Neocortical layer 6, a review”. In: *Frontiers in Neuroanatomy* 4 (2010), p. 13. ISSN: 1662-5129. DOI: 10.3389/fnana.2010.00013. URL: https://www.frontiersin.org/article/10.3389/fnana.2010.00013.

[24] Mark Tranmer and Mark Elliot. “Multiple linear regression”. In: *The Cathie Marsh Centre for Census and Survey Research (CCSR)* 5 (2008), pp. 30–35.

[25] Wessel N van Wieringen. “Lecture notes on ridge regression”. In: *arXiv preprint arXiv:1509.09169* (2015).

[26] Saining Xie et al. “Aggregated residual transformations for deep neural networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1492–1500.

[27] Daniel LK Yamins and James J DiCarlo. “Using goal-driven deep learning models to understand sensory cortex”. In: *Nature neuroscience* 19.3 (2016), pp. 356–365.

[28] Daniel LK Yamins et al. “Performance-optimized hierarchical models predict neural responses in higher visual cortex”. In: *Proceedings of the National Academy of Sciences* 111.23 (2014), pp. 8619–8624.

[29] Matthew D Zeiler and Rob Fergus. “Visualizing and understanding convolutional networks”. In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

Supplementary Materials

S.5 Description of Each Deep Neural Network

AlexNet, Krizhevsky et al. 2012 [10]. Cadieu et al. 2014 [3] examined the DCNNs model "SuperVision" described in [10]. This model has seven layers and is trained by supervised learning on the ImageNet 2011 Fall release which has around 15M images and 22K categories with additional training on the LSVRC-2012 dataset that has 1000 categories. Cadieu et al. 2014 [3] extracted the features from the penultimate layer of this model. There are 4096 features that are computed based on the testing images that are cropped out the center 224 by 224 pixels which corresponds to the input size of the model. Then logistic regression are performed on these features to predict category labels.

ZF Net, Zeiler & Fergus 2013 [29]. The ZFnet model is an eight layer convolutional neural network that is trained using supervised learning on the LSVRC-2012 dataset. To train the model while preventing overfitting, variations of the input data are taken by either changing its orientation or cropping it. This model contains a softmax function which allows for the features to be viewed as probabilities. To prevent overfitting, the training data consists of cropped and flipped versions of the 256x256 pixel images.

HMO, Yamins et al. 2014 [28]. This model described in [28] uses a hierarchical modular optimization (HMO) to develop a large, deep network which is a combination of convolutional neural networks [3]. The HMO algorithm is an adaptive boosting procedure that interleaves hyperparameter optimization (see [28] Methods for more details). The original model is used to examine a screening task on images of objects with randomly selected background. The specific model that Cadieu et al. 2014 [3] uses in this task is composed of four layers and produces 1250 top-level outputs. Although the model used in [3] is identical to the original model presented in Yamins et al. 2014 [28], there are several differences to the analyses that are important to note: first, [3] only took a subset of the images in the high variation task compared to [28].

Inception V3, Szegedy et al. 2015 [22]. This model is a large, deep convolutional network with 48 layers and trained on ImageNet LSVRC-2012 dataset which consists of 1.2 million training images, with 1000 categories, which outperforms than some of the other DCNNs with 21.2% $top - 1$ and 5.6% $top - 5$ error for single crop evaluation [22]. This model is composed of symmetric and asymmetric building blocks: convolutions, average pooling, max pooling, concats, dropouts, and fully connected layers [22]. While many models obtain their outputs from the penultimate layer, we used the average pooling layer to obtain the outputs, as this is more comparable to the models used by [3]. The test images set provided by Cadieu et al. 2014 [3] has dimensions of 256 by 256.

ResNeXt, Xie et al. 2017 [26]. This network contains repeated building blocks that aggregates a set of transformations. A *next* dimension called "cardinality" is essential to added on the top of the dimensions of depth and width in order to achieve higher accuracy instead of going deeper or wider in the network [26]. This "cardinality" refers to the size of the set of transformations. This model is trained on ILSVRC-2016 classification task and wins the 2nd place with 20.69% $top - 1$ and 5.47% $top - 5$ error (ResNeXt-101-32x8d, 101 layers network). The 2048 dimensional features are extracted from the penultimate layer of this model. The input images need to be loaded into a range $[0, 1]$. In this specific pre-trained model, we cropped the test images from the center with dimension of 224 by 224 and normalized the test images using $mean = [0.485, 0.456, 0.406]$ and

$std = [0.229, 0.224, 0.225]$ (The means and standard deviations are computed based on the ILSVRC dataset).

S.6 Equations and Parameters

S.6.1 Equations

Basics of Artificial Neural Networks

General output equation for an artificial neural network: $y_j^k = f\Bigg(\sum_{i=1}^{n} w_{ij}^k x_i^k + b_j^k \Bigg)$

Kernel Analysis

The standard Gaussian Kernel: $k_\sigma(x, x') = e^{(-||x-x'||^2/2\sigma^2)}$

The Gaussian Kernel matrix:

$$K_\sigma = \begin{bmatrix} k_\sigma(\phi((x_1)), \phi((x_1)) & ... & k_\sigma(\phi((x_1)), \phi((x_n))) \\ : & & : \\ k_\sigma(\phi((x_n)), \phi((x_1)) & ... & k_\sigma(\phi((x_n)), \phi((x_n)) \end{bmatrix}$$

Regularized kernel ridge regression minimiation equation: $\min_{\Theta \in R^n} \frac{1}{2} \Sigma_{i=1}^{n} ||Y - K_\sigma \Theta||_2^2 + \frac{\lambda}{2} \Theta^t K_\sigma \Theta$
Solution to the regularized kernel ridge regression equation: $\Theta_\sigma^*(\lambda) = (K_\sigma + \lambda * I)^{-1} Y$
The fitting equation: $K_\sigma \Theta = Y$
Generic leave-one-out error: $LOOE_\sigma(\lambda) = \frac{\Theta_\sigma^*(\lambda)}{diag((K_\sigma + \lambda I)^{-1})}$
Mean squared leave-one-out error: $looe_\sigma(\lambda) = \frac{1}{n} \sum_i (LOOE_\sigma(\lambda))^2$
Linear Support Vector Machine
Solution for hyperplane of a Linear SVM: f(x) $= w^T x + b = \sum_{j=1}^{m} w_j x_j + b = 0$
Ridge Regression
General linear regression equation: $\min_w ||X_w - Y||_2^2$
General ridge regression equation: $\min_w ||X_w - Y||_2^2 + \alpha ||w||_2^2$

S.6.2 Parameters

Parameter	Definition
λ	regularization parameter
Θ	regression parameter
$\frac{1}{\lambda}$	complexity
$1 - looe(\lambda)$	precision
σ	Gaussian distribution scale factor
X	images
Y	normalized labels
ϕ	feature extraction process
$K_{sigma}\Theta$	regularized kernel

Table 1: Parameters used in Kernel Analysis

S.7 Python Code

```
import torch
import torchvision.models as models
```

Parameter	Definition
X	image features
Y	normalized labels
α	penalty term
w	weights

Table 2: Parameters used in Ridge Regression

```
from torchvision import transforms
from torch.autograd import Variable
import torch.nn as nn
import cv2 as cv

import PIL
import PIL.Image

import scipy
from scipy import io
from scipy.spatial.distance import pdist, squareform
from scipy import exp
from scipy.linalg import eigh
from scipy.optimize import minimize
from numba import jit, float64, int64
import numpy as np
import matplotlib.pyplot as plt

IT_multi=io.loadmat('Downloads/DiCarlo2014Project/PLoSCB2014_data_20141216/NeuralData_
IT_single=io.loadmat('Downloads/DiCarlo2014Project/PLoSCB2014_data_20141216/NeuralDat
V4_multi=io.loadmat('Downloads/DiCarlo2014Project/PLoSCB2014_data_20141216/NeuralData_
V4_single=io.loadmat('Downloads/DiCarlo2014Project/PLoSCB2014_data_20141216/NeuralDat

AlexNet = io.loadmat('Downloads/DiCarlo2014Project/20150218/Models_Krizhevsky2012.mat'
ZFNet = io.loadmat('Downloads/DiCarlo2014Project/20150218/Models_ZeilerFergus2013.mat'
HMO = io.loadmat('Downloads/DiCarlo2014Project/20150218/Models_HMO.mat')

resnext101 = models.resnext101_32x8d(pretrained=True)
inception = models.inception_v3(pretrained=True)

avgpool_layer_resnext101 = resnext101._modules.get('avgpool')
avgpool_layer_inception = inception._modules.get('avgpool')

resnext101.eval()
inception.eval()

preprocess = transforms.Compose([transforms.Resize(256),
                                 transforms.CenterCrop(224),
                                 transforms.ToTensor(),
                                 transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[
def copy_data_res(m, i, o):
    my_embedding_res.copy_((o.data.reshape(o.data.size(1))))
def copy_data_inc(m, i, o):
    my_embedding_inc.copy_((o.data.reshape(o.data.size(1))))

my_embedding_res = torch.zeros([2048])
my_embedding_inc = torch.zeros([2048])
ResNext_X = torch.zeros([1960, 2048])
Inception_X = torch.zeros([1960, 2048])

for i in range(1960):

    img_str = 'Downloads/DiCarlo2014Project/PLoSCB2014_data_20141216/' + str(IT_multi[
    img = PIL.Image.open(img_str)
    input_tensor = preprocess(img)
    input_batch = Variable(input_tensor.unsqueeze(0))

    h_res = avgpool_layer_resnext101.register_forward_hook(copy_data_res)
    h_inc = avgpool_layer_inception.register_forward_hook(copy_data_inc)

    resnext101(input_batch)
    inception(input_batch)

    h_res.remove()
    h_inc.remove()

    ResNext_X[i, ] = my_embedding_res
    Inception_X[i, ] = my_embedding_inc
    print(i)

np.savetxt('Downloads/DiCarlo2014Project/Results/ResNext_X.txt', ResNext_X)
np.savetxt('Downloads/DiCarlo2014Project/Results/Inception_X.txt', Inception_X)

ResNext = np.loadtxt('Downloads/DiCarlo2014Project/Results/ResNext_X.txt')
Inception = np.loadtxt('Downloads/DiCarlo2014Project/Results/Inception_X.txt')

def gaussian_kernel(X, sigma, n_components, PCA = False): # potentially can be used wi

    """
    Gaussian kernel implementation.
    Parameters
    ----------
    X: {NumPy ndarray}, shape = [n_examples, n_features]
    lambda, sigma: float
        Tuning parameter of the kernel
    n_components: int
        Number of principal components to return
    Returns
    ------------
    X_pc: {NumPy ndarray}, shape = [n_examples, k_features]
        Projected dataset
    """

    # Calculate pairwise squared Euclidean distances
    # in the MxN dimensional dataset.
    sq_dists = pdist(X, 'sqeuclidean')
    # Convert pairwise distances into a square matrix.
    mat_sq_dists = squareform(sq_dists)
    # Compute the symmetric kernel matrix.
    K = np.exp(-mat_sq_dists/(2*sigma**2))

    if PCA:
        # Center the kernel matrix.
        N = K.shape[0]
        one_n = np.ones((N,N)) / N
        K = K - one_n.dot(K) - K.dot(one_n) + one_n.dot(K).dot(one_n)
        # Obtaining eigenpairs from the centered kernel matrix
        # scipy.linalg.eigh returns them in ascending order
        eigvals, eigvecs = eigh(K)
        eigvals, eigvecs = eigvals[::-1], eigvecs[:, ::-1]
        # Collect the top k eigenvectors (projected examples)
        X_pc = np.column_stack([eigvecs[:, i]
                               for i in range(n_components)])
        return X_pc
    else:
        return K


def Theta(K, lmbda, Y):

    return np.matmul(np.linalg.inv(K + lmbda * np.identity(K.shape[0])), Y)

def looe(lmbda, K, Y):

    theta = Theta(K, lmbda, Y)
    loo_error = theta / np.diagonal(np.linalg.inv(K+lmbda * np.identity(K.shape[0])))

    return loo_error

def max_precision(X, Y, lmbda):

    # This is too slow! Even though this is used in the paper.
    sq_dists = pdist(X, 'sqeuclidean')
    mat_sq_dists = squareform(sq_dists)
    sigma_m = mat_sq_dists.std()
    bnds = ((.01*sigma_m, sigma_m),) #[.1*sigma_m], [10*sigma_m])

    def min_looe(sigma):

        K = gaussian_kernel(X, sigma, 1)
        LOOE = looe(lmbda, K, Y)

        return (LOOE**2).mean()

    def grad_ml(sigma):
        h = 1e-9
        grad_sig = np.zeros_like(sigma)

        f_0 = min_looe(sigma)
        x_d = np.copy(sigma)
        x_d += h
        f_d = min_looe(x_d)
        grad_sig = (f_d - f_0) / h

        return grad_sig

    min_res = minimize(min_looe, sigma_m, method='SLSQP', options={'maxiter': 1e4}, ja
    max_prec = 1-min_res.fun

    return max_prec


# here we sub-sampling the images 10 times.
categories = ['Animals','Cars','Chairs','Faces','Fruits','Planes','Tables']
lmbda = np.logspace(np.log10(.001),np.log10(10000), 28)
prec_ResNext = np.zeros((10, 28))
prec_Inception = np.zeros((10, 28))
prec_AlexNet = np.zeros((10, 28))
prec_ZFNet = np.zeros((10, 28))
prec_HMO = np.zeros((10, 28))

for s in range(10):

    features = 0
    f_ResNext = np.zeros((1568, 2048)) # 80% of the whole imageset.
    f_Inception = np.zeros((1568, 2048))
    f_AlexNet = np.zeros((1568, 4096))
    f_ZFNet = np.zeros((1568, 4096))
    f_HMO = np.zeros((1568, 1250))

    Y_c = np.zeros((7, 1568))

    for i in range(1960):
        img_cat_sample = str(IT_multi['meta'][i])[52:]

        for c in range(7):
            if categories[c] in img_cat_sample:
                if img_cat_sample[len(categories[c])+1+2*s] == '1':

                    f_ResNext[features ,:] = ResNext[i ,]
                    f_Inception[features ,:] = Inception[i ,]
                    f_AlexNet[features ,:] = AlexNet['features'][i ,]
                    f_ZFNet[features ,:] = ZFNet['features'][i ,]
                    f_HMO[features ,:] = HMO['features'][i ,]

                    Y_c[c,features] = 1

                    features += 1

    # Normalize the labels.
    # This is super important!
    for c in range(7):
        Y_c[c,] = Y_c[c,] * np.sqrt(1568./Y_c[c,].sum())

    print('features gathered')

    for l in range(28):
        for c in range(7):

            prec_ResNext[s, l] += max_precision(f_ResNext, Y_c[c,], lmbda[l])
            prec_Inception[s, l] += max_precision(f_Inception, Y_c[c,], lmbda[l])
            prec_AlexNet[s, l] += max_precision(f_AlexNet, Y_c[c,], lmbda[l])
            prec_ZFNet[s, l] += max_precision(f_ZFNet, Y_c[c,], lmbda[l])
            prec_HMO[s, l] += max_precision(f_HMO, Y_c[c,], lmbda[l])

            print('category', c)

        # average over 7 categories.
        prec_ResNext[s, l] /= 7.
        prec_Inception[s, l] /= 7.
        prec_AlexNet[s, l] /= 7.
        prec_ZFNet[s, l] /= 7.
        prec_HMO[s, l] /= 7.

        print('lambda', l)

np.savetxt('Downloads/DiCarlo2014Project/Results/sampling_ResNext_0-3.txt', prec_ResNe
np.savetxt('Downloads/DiCarlo2014Project/Results/sampling_Inception_0-3.txt', prec_Inc
np.savetxt('Downloads/DiCarlo2014Project/Results/sampling_AlexNet_0-3.txt', prec_AlexN
np.savetxt('Downloads/DiCarlo2014Project/Results/sampling_ZFNet_0-3.txt', prec_ZFNet)
np.savetxt('Downloads/DiCarlo2014Project/Results/sampling_HMO_0-3.txt', prec_HMO)
```

```
# For Neuraonal Data.

# here we sub-sampling the images 10 times.
categories = ['Animals','Cars','Chairs','Faces','Fruits','Planes','Tables']
lmbda = np.logspace(np.log10(.001),np.log10(10000), 28)
prec_IT = np.zeros((10, 28))
prec_V4 = np.zeros((10, 28))

for s in range(10):

    features = 0
    f_ResNext = np.zeros((1568, 2048)) # 80% of the whole imageset.
    f_Inception = np.zeros((1568, 2048))

    Y_c = np.zeros((7, 1568))

    for i in range(1960):
        img_cat_sample = str(IT_multi['meta'][i])[52:]

        for c in range(7):
            if categories[c] in img_cat_sample:
                if img_cat_sample[len(categories[c])+1+2*s] == '1':

                    f_ResNext[features,:] = ResNext[i,]
                    f_Inception[features,:] = Inception[i,]

                    Y_c[c,features] = 1

                    features += 1

    print('features gathered')


    for l in range(28):
        for c in range(7):

            prec_ResNext[s, l] += max_precision(f_ResNext, Y_c[c,], lmbda[l])
            prec_Inception[s, l] += max_precision(f_Inception, Y_c[c,], lmbda[l])

            print('category', c)

        # average over 7 categories.
        prec_ResNext[s, l] /= 7.
        prec_Inception[s, l] /= 7.

        print('lambda', l)
```

```
Alex_Prec = np.zeros((10, 28))
ZF_Prec = np.zeros((10, 28))
HMO_Prec = np.zeros((10, 28))

ResNext_Prec = np.zeros((10, 28))
Inception_Prec = np.zeros((10, 28))

Alex_Prec = np.loadtxt('Downloads/DiCarlo2014Project/Results/sampling_AlexNet_0-3.txt'
ZF_Prec = np.loadtxt('Downloads/DiCarlo2014Project/Results/sampling_ZFNet_0-3.txt')
HMO_Prec = np.loadtxt('Downloads/DiCarlo2014Project/Results/sampling_HMO_0-3.txt')
ResNext_Prec = np.loadtxt('Downloads/DiCarlo2014Project/Results/sampling_ResNext_0-3.t
Inception_Prec = np.loadtxt('Downloads/DiCarlo2014Project/Results/sampling_Inception_0

IT_Prec = np.zeros((10, 28))
V4_Prec = np.zeros((10, 28))

IT_Prec = np.loadtxt('Downloads/DiCarlo2014Project/Results/sampling_IT_multi_4-6.txt')
V4_Prec = np.loadtxt('Downloads/DiCarlo2014Project/Results/sampling_V4_multi_4-6.txt')

lmbda = np.logspace(np.log10(.001),np.log10(10000), 28)
complexity = 1./lmbda

plt.plot(complexity, ResNext_Prec.mean(axis=0), 'b--', label='ResNext')
plt.plot(complexity, Inception_Prec.mean(axis=0), 'c--', label='Inception')
plt.fill_between(complexity, ResNext_Prec.mean(axis=0) - 3*ResNext_Prec.std(axis=0), R
plt.fill_between(complexity, Inception_Prec.mean(axis=0) - 3*Inception_Prec.std(axis=0

plt.plot(complexity, Alex_Prec.mean(axis=0), 'r--', label='Krizhevsky␣et␣al.␣2012')
plt.plot(complexity, ZF_Prec.mean(axis=0), 'g--', label='Zeiler␣&␣Fergus␣2013')
plt.plot(complexity, HMO_Prec.mean(axis=0), 'y--', label='HMO')
plt.fill_between(complexity, Alex_Prec.mean(axis=0) - 3*Alex_Prec.std(axis=0), Alex_Pr
plt.fill_between(complexity, ZF_Prec.mean(axis=0) - 3*ZF_Prec.std(axis=0), ZF_Prec.mea
plt.fill_between(complexity, HMO_Prec.mean(axis=0) - 3*HMO_Prec.std(axis=0), HMO_Prec.

plt.plot(complexity, IT_Prec.mean(axis=0), 'k--', label='IT␣Multi')
plt.plot(complexity, V4_Prec.mean(axis=0), 'p--', label='V4␣Multi')
plt.fill_between(complexity, IT_Prec.mean(axis=0) - 3*IT_Prec.std(axis=0), IT_Prec.mea
plt.fill_between(complexity, V4_Prec.mean(axis=0) - 3*V4_Prec.std(axis=0), V4_Prec.mea
plt.legend(loc='upper␣left')

plt.xscale('log')
plt.xlabel('complexity')
plt.ylabel('Precision')

normalize = 1
sklearn.preprocessing.normalize(ResNext, axis=normalize, copy=False)
sklearn.preprocessing.normalize(Inception, axis=normalize, copy=False)
```

```
AlexNet = sklearn.preprocessing.normalize(AlexNet['features'], axis=normalize)
ZFNet = sklearn.preprocessing.normalize(ZFNet['features'], axis=normalize)
HMO = sklearn.preprocessing.normalize(HMO['features'], axis=normalize)
V4 = sklearn.preprocessing.normalize(V4_multi['features'], axis=normalize)

IT_multi = sklearn.preprocessing.normalize(IT_multi['features'], axis=normalize)

ZF_explained_v = np.zeros((10, 168))

for i in range(10):
    # one split.

    # fit to IT Multi.
    X_train, X_test, y_train, y_test = train_test_split(ZFNet, IT_multi,test_size=0.2,

    for site in range(168):
        ridge_ZF = linear_model.Ridge(1.) # linear_model.SGDRegressor()
        ridge_ZF.fit(X_train, y_train[:,site])

        ZF_explained_v[i,site] = explained_variance_score(y_test[:,site], ridge_ZF.pre

    print('done!')

Alex_explained_v = np.zeros((10, 168))

for i in range(10):
    # one split.

    # fit to IT Multi.
    X_train, X_test, y_train, y_test = train_test_split(AlexNet, IT_multi,test_size=0.

    for site in range(168):
        ridge_Alex = linear_model.Ridge(1.) # linear_model.SGDRegressor()
        ridge_Alex.fit(X_train, y_train[:,site])

        Alex_explained_v[i,site] = explained_variance_score(y_test[:,site], ridge_Alex

    print('done!')

HMO_explained_v = np.zeros((10, 168))

for i in range(10):
    # one split.

    # fit to IT Multi.
    X_train, X_test, y_train, y_test = train_test_split(HMO, IT_multi,test_size=0.2,ra

    for site in range(168):
        ridge_HMO = linear_model.Ridge(1.) # linear_model.SGDRegressor()
        ridge_HMO.fit(X_train, y_train[:,site])

        HMO_explained_v[i,site] = explained_variance_score(y_test[:,site], ridge_HMO.p

    print('done!')

ResNext_explained_v = np.zeros((10, 168))

for i in range(10):
    # one split.

    # fit to IT Multi.
    X_train, X_test, y_train, y_test = train_test_split(ResNext, IT_multi,test_size=0.

    for site in range(168):
        ridge_ResNext = linear_model.Ridge(1.) # linear_model.SGDRegressor()
        ridge_ResNext.fit(X_train, y_train[:,site])

        ResNext_explained_v[i,site] = explained_variance_score(y_test[:,site], ridge_R

    print('done!')

Inception_explained_v = np.zeros((10, 168))

for i in range(10):
    # one split.

    # fit to IT Multi.
    X_train, X_test, y_train, y_test = train_test_split(Inception, IT_multi,test_size=

    for site in range(168):
        ridge_Inception = linear_model.Ridge(1.) # linear_model.SGDRegressor()
        ridge_Inception.fit(X_train, y_train[:,site])

        Inception_explained_v[i,site] = explained_variance_score(y_test[:,site], ridge

    print('done!')

V4_explained_v = np.zeros((10, 168))

for i in range(10):
    # one split.

    # fit to IT Multi.
    X_train, X_test, y_train, y_test = train_test_split(V4, IT_multi,test_size=0.2,ran

    for site in range(168):
        ridge_V4 = linear_model.Ridge(1.) # linear_model.SGDRegressor()
        ridge_V4.fit(X_train, y_train[:,site])

        V4_explained_v[i,site] = explained_variance_score(y_test[:,site], ridge_V4.pre

    print('done!')

zf_ = np.sqrt(np.abs(ZF_explained_v))
zf_.sort(axis=0)
alex_ = np.sqrt(np.abs(Alex_explained_v))
alex_.sort(axis=0)
hmo_ = np.sqrt(np.abs(HMO_explained_v))
hmo_.sort(axis=0)
resnext_ = np.sqrt(np.abs(ResNext_explained_v))
resnext_.sort(axis=0)
incep_ = np.sqrt(np.abs(Inception_explained_v))
incep_.sort(axis=0)
v4_ = np.sqrt(np.abs(V4_explained_v))
v4_.sort(axis=0)

df = pd.DataFrame(data = np.array([incep_[:,84], resnext_[:,84], hmo_[:,84], alex_[:,8

ax=sns.boxplot(x="variable", y="value", data=pd.melt(df))
ax.set(xlabel='␣',ylabel='Median␣IT␣Multi-Unit␣Explained␣Variance␣(%)')
ax.set_xticklabels(ax.get_xticklabels(),rotation=60)
plt.ylim(0.1,.7)
plt.show()
```