Integrating a Gaussian Classifier into a CNN

Independent Work Final Report

Pulkit Singh
Advisor: Thomas L. Griffiths
Contributors: Joshua C. Peterson, Ruairidh M. Battleday

Abstract

The assignment of natural images to categories is an important problem both in machine learning and in the study of human cognition. Visual categorization models in psychology examine strategies of human categorization, which computer vision models seek to effectively discriminate between categories. Convolutional Neural Networks (CNNs) are the current state-of-the-art in computer vision — the training performance of these discriminative models is approaching near perfect levels, but their generalizability leaves room for improvement. On the other hand, cognitive models are density estimators: generative not discriminative. In particular, prototypical models categorize by measuring distance from the 'mean' or 'ideal' prototype of a category. Given the related nature of these fields, it is interesting to ask whether the research results from one can inform and enrich the goals of the other. Motivated by the notions of testing prototype theory on representations created by CNNs, and exploring ways for the CNN to make more human-like representations of categories, this project integrates a differentiable Gaussian classifier in a Convolutional Neural Network. In particular, it models classes in the CIFAR-10 dataset as multivariate Gaussian distributions, and creates three custom Keras classification layers corresponding to nested constraints on the covariance of the distributions. This project finds that these class representation do not impede accuracy of predictions, and significantly decreases the discrepancy between human and machine-learned distributions of judgements.

1 Motivation and Goal

The assignment of natural images to categories is an important problem both in machine learning and in the study of human cognition. The study of categorization in psychology seeks to understand the strategies humans use to categorize stimuli, while the field of image classification in computer vision seeks to build systems that can effectively discriminate between natural images of different categories. In other words, the field of categorization in psychology examines how humans generate categories and the field of classification in computer science is interested in teaching machines to discriminate between these categories. Given the related nature of these fields, it is interesting to ask whether the research results from one can inform and enrich the goals of the other.

In computer vision, the current state of the art models are Convolutional Neural Networks (CNNs) which employ convolutional filters to identify low-level features in images, and pool these features to build increasingly high-level representations. In essence, CNN's create spatial feature maps from images, and then discriminate between these created representations in order to classify images. These discriminative models are aimed at optimizing the accuracy, i.e. the proportion of correctly-classified inputs, and they deal directly with the probability of class membership given an image. This approach is very effective, and their performance on natural image classification tasks is asymptoting towards near-perfect levels on training data [Hua+18], but their generalization capability presents room for improvement [Zha+16].

On the other hand, cognitive models are concerned with capturing human classification behaviour — human categorization models are density estimators: generative not discriminative. The classic paper by Posner and Keele (1968) posits that people represent categorical concepts through prototypes. Prototypes are described to be mental constructs containing typical characteristics of various categories. Then, when a person encounters a new object, they compare it to the category prototypes in memory [PK68]. Thus, we see that in this prototype paradigm, categorization is wholly dependent on the representation of different classes of objects.

Then, taking a step back, we see that categorization theories prototype theory are very concerned with the representation of classes, whereas CNN classifiers have no fixed representation of classes. In general, we often have little control over the intermediate class representations in a neural network, but it would be interesting to examine the implications of imposing a particular density estimation. This is exactly the key motivation behind this project — to explore the consequences of exploring different class representations in discriminative models. This enables us to effectively test the classical prototype theory of categorization, and it may also provide insights into enabling CNNs to produce more robust, human-like errors.

While it is possible to test categorization theories at an image level, i.e., to compute image means for each class, it is very ineffective because the pixel space is extremely sparse and we are unable to gather enough critical information. We need to be able to isolate important features of the images and create representations that will enable classification - something that CNNs are very good at doing. Thus, the need for feature encoding and category modelling at the same time motivates the goal of **integrating a differentiable Gaussian classifier in a Convolutional Neural Network**. This approach enables the CNN to learn a representation of training images such that Gaussian densities over these representations accurately predicts category membership.

We will then be able to test these classifier-integrated model not only on traditional metrics like accuracy, but also on how human-like the resulting category representations are. This is possible because of a novel dataset, CIFAR10-H, which comprises full label distributions for the entire 10,000-image CIFAR10 test set, utilizing over 500k crowd-sourced human categorization judgments [BPG19]. This dataset enables us to compute the discrepancy between human distributions and the machine-learned distribution, enabling us to investigate the effect of enforcing class representations.

2 Problem Background and Related Work

Research related to both cognitive modelling and computer vision has been gaining traction in recent years. The success of Convolutional Neural Networks has led to exploration into the reasons for their efficacy, including their ability to produce useful representations of images. A paper by Peterson, Abbott & Griffiths examined the “relationship between the representations learned by these networks and human psychological representations recovered from similarity judgments", and found that “deep features learned in service of object classification account for a significant amount of the variance in human similarity judgments for a set of animal images" [PAG17]. Their research also indicated that the representations still lacked “some some qualitative distinctions that are a key part of human representations" [PAG17]. This work indicates not only the potential for neural networks to model human categories, but also prompts the exploration of integrating different class representations within a neural network to examine this discrepancy.

The notion of using neural networks to create embeddings of images that correspond to class-clusters has also been explored in a paper by Snell, Swersky & Zemel. They propose “prototypical network for few-shot learning", which “learns a metric space in which classification can be performed by computing distances to prototype representations of each class" [SSZ17]. In other words, they learn a "non-linear mapping of the input into an embedding space using a neural network", and then extract these representations from the network and perform classification using squared Euclidean distance. Using this approach, they were able to achieve state-of-the-art results on the zero-shot learning dataset CUB-200 [SSZ17]. This work indicates the potential of modelling classes as clusters/distributions. However, the classification process is separate from the production of embeddings, whereas this project proposes an integrated approach where both embeddings and densities over these embeddings are learned jointly.

Another related project by Wen, Zhang, Li & Qiao focuses on discriminative feature learning for deep face recognition using "center loss". In order to “enhance the discriminative power of deeply learned features" for face recognition, the center loss “simultaneously learns a center for deep features of each class and penalizes the distances between deep features and their corresponding class centers" [Wen+16]. The models are trained with a combination of conventional ’softmax’ loss and center loss, and they are able to achieve state-of-the-art results on “MegaFace (the largest public domain face benchmark)"[Wen+16]. This paper further demonstrates the potential of integrating distance computations in the embedding space of a neural network, and lays the foundation of modelling classes as probability distributions.

3 Approach

3.1 Modeling Categories as Probability Distributions

Convolutional Neural Networks (CNNs) rely on convolution-pooling cycles to identify features of the input image and pool the identified features together in order to create spatial feature maps. The feature map of an image can be converted into a generic vector using a fully-connected layer, or merely by flattening the representation. An image is a giant matrix (or set of matrices) of pixel values, and the CNN embeds in into a vector representation that is of lower dimension than the original but still relatively high. This means we can think of each image as being embedded as a point in this high-dimensional space.

In the context of a classifier, the classes of images that we would like the model to differentiate between are just groups of points that correspond to the representations of input images. This project models each class of images as a multivariate Gaussian probability distribution by integrating a differentiable Gaussian classifier within the CNN. The density of the points in the different class distributions rely on the representations created by the CNN. But, we want to learn the embeddings of the images and densities of the classes jointly, so that we learn a representation of images such that a Gaussian density over those representations best predicts category membership.

3.2 Gaussian Classifier

Now that the motivation for modelling the classes as Gaussian probability distributions in understood, we can move on to mathematically specifying the classifier. Let us consider a set of N images to be classified into k categories. We want to fit multivariate Gaussian distributions to each of these classes. For each class, let us define:

$$\mu_c : \text{mean feature vector (d-dimensional vector)}$$

$$\Sigma_c : \text{covariance matrix (d x d matrix)}$$

First, let us consider the functioning of the classifier with one image, and then we can generalize to the classification of N images.

3.2.1 Classifying One Image

Let us define a training image I (matrix of s x s pixels) and the true category label y. The image I is transformed into spatial feature maps, and then embedded into vector representation x. We must work on classifying x, the d-dimensional vector that the CNN has produced for image I. We need to find the specify a discriminant function that specifies the probability of each class, given the training example. Bayes rule tells us:

$$P[y = c \mid x] = \frac{P[x \mid y = c] \cdot P[y = c]}{\sum_{c'} P[x \mid y = c] \cdot P[y = c']}$$

But, if we have no information about the prior probabilities of the classes, then it would be safe to set all the values of $P[y = c]$ equal to each other. In this case, the posterior probability is equal to the normalized likelihood as shown below:

$$P[y = c \mid x] = \frac{P[x \mid y = c]}{\sum_{c'} P[x \mid y = c']}$$

We set our discriminant function g to exactly this normalized likelihood as follows:

$$g(x, c) = \frac{P[x \mid y = c]}{\sum_{c'} P[x \mid y = c']}$$

Since we are fitting multivariate Gaussian distributions to all the classes, the individual class-conditional likelihoods are proportional to the exponentiated negative Mahalanobis distance. This means the probability that x belongs to the c-th class (with mean μ_c and covariance matrix Σ_c) is proportional to:

$$P[x \mid y = c] \propto e^{-(x-\mu_c)^T \cdot \Sigma_c^{-1} \cdot (x-\mu_c)}$$

Then, we can reformulate the discriminant function in terms of Mahalanobis distances (for brevity $M(x, c)$).

$$g(x, c) = \frac{e^{-(x-\mu_c)^T \cdot \Sigma_c^{-1} \cdot (x-\mu_c)}}{\sum_{c'} e^{-(x-\mu_{c'})^T \cdot \Sigma_{c'}^{-1} \cdot (x-\mu_{c'})}} = \frac{e^{-M(x,c)}}{\sum_{c'} e^{-M(x,c')}}$$

We notice that the activation being applied to the Mahalanobis distances between the input and the classes is the normalized exponential function, or the softmax activation. This function can take a vector of real numbers and normalize it into a probability distribution vector. So, for every training image, the classifier needs to produce a k-dimensional vector of Mahalanobis distances from each of the k-classes, and then we can apply the softmax activation to this vector.

3.2.2 Classifying N images

Now that we have worked out the functioning of the classifier with a single input image x, we can generalize to a set of N images. For each image, the CNN produces a d-dimensional vector, so let us consider the (N x d) matrix X as the input to the Gaussian classification layer. In X, each row corresponds to the CNN representation of a unique image.

We can specify an analogous discriminant function $G(X, c)$ which produces the probability of each row of the matrix X belonging to class c. The result would be a (N x k) matrix of normalized likelihoods such that the i^{th} row corresponds to the class probabilities corresponding to the i^{th} image. Additionally, for each row, the j^{th} column corresponds to the likelihood that the corresponding image belongs to the j^{th} class. This matrix of normalized likelihoods (labelled NL) is illustrated below:

$$\text{NL} = \begin{bmatrix} \frac{P(x_1|y=c_1)}{\sum_{c'} P[x|y=c']} & \cdots & \frac{P(x_1|y=c_k)}{\sum_{c'} P[x|y=c']} \\ \vdots & \vdots & \vdots \\ \frac{P(x_N|y=c_1)}{\sum_{c'} P[x|y=c']} & \cdots & \frac{P(x_N|y=c_k)}{\sum_{c'} P[x|y=c']} \end{bmatrix}$$

Thus, we have successfully created a mechanism to compute a probability distribution for each image over all the classes. To calculate the discrepancy between the computed distribution and the true label or distribution in our dataset, we can use the cross-entropy function. Given two probability distribution vectors, p (true distribution) and q (fitted distribution), the cross-entropy between them can be calculated by:

$$\text{cross-entropy}(p, q) = -p^T \cdot log(q)$$

Then, to calculate the total discrepancy between the true and estimated distributions over all training examples in X, we can run the cross-entropy function on the prediction for every image and corresponding output label. Let Y be the (N x k) matrix corresponding to the true category distribution (or one-hot encoded label), such that the i^{th} row of C corresponds to the output distribution for the i^{th} image in X. Then, the total discrepancy, or training error can be calculated as:

$$\text{training error} = \text{crossentropy(C, NL)} = \text{- colsum(rowsum(C .* log(NL)))}$$

3.3 Examining the Nested Range of Models

It is now clear that the probability of class-membership is modelled as normalized likelihood, and that this is proportional to the exponentiated negative Mahalanobis distance between the input and the mean of the distribution. Since this distance computation is essential to the project, let us stop and examine it. Just as before, let x be d-dimensional input and μ_c and Σ_c be the mean and covariance matrix of the c^{th} class respectively. Then, the Mahalanobis distance between x and μ_c (M(x, μ_c))is given by:

$$M(x, \mu_c) = (x - \mu_c)^T \cdot \Sigma_c^{-1} \cdot (x - \mu_c)$$

We see that the only source of variability in this computation is the inverse covariance matrix Σ_c^{-1}, as x and μ_c remain fixed. The covariance matrix Σ_c can take many forms, and these forms dictate the flexibility of the multivariate Gaussian distributions that we are fitting to the various classes. In particular, the covariance matrix of a particular class can take the following forms:

- identity
- scaled identity
- diagonal / axis-aligned
- free

These constraints dictate the flexibility or the complexity of the gaussian distributions. The following figure illustrates the kinds of classes each matrix-form creates in 2-dimensions:

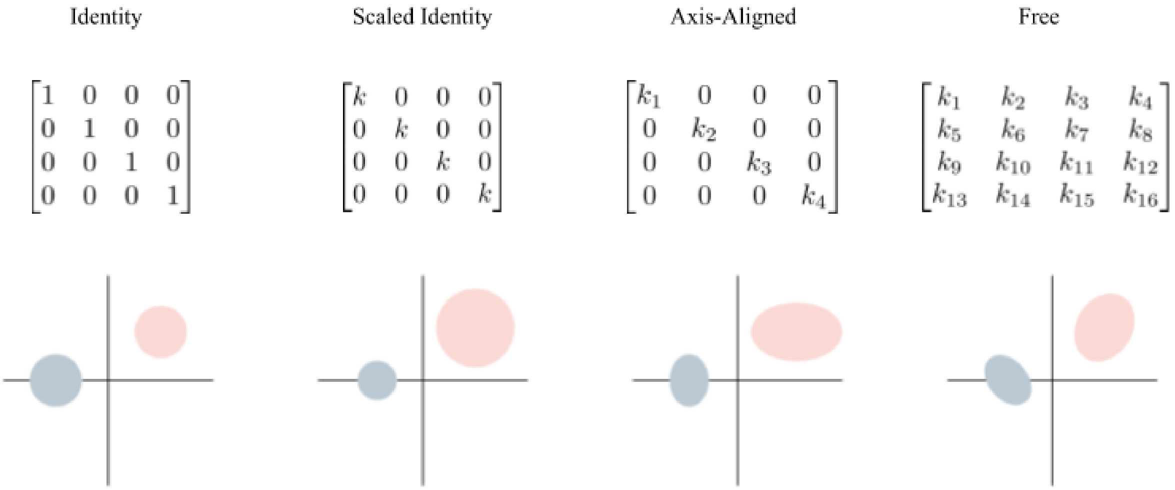
Identity
Scaled Identity
Axis-Aligned
Free
$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}$
$\begin{bmatrix}k&0&0&0\\0&k&0&0\\0&0&k&0\\0&0&0&k\end{bmatrix}$
$\begin{bmatrix}k_1&0&0&0\\0&k_2&0&0\\0&0&k_3&0\\0&0&0&k_4\end{bmatrix}$
$\begin{bmatrix}k_1&k_2&k_3&k_4\\k_5&k_6&k_7&k_8\\k_9&k_{10}&k_{11}&k_{12}\\k_{13}&k_{14}&k_{15}&k_{16}\end{bmatrix}$

Figure 1: Covariance matrix constraints and corresponding class representations

In two dimensions, we see that the identity covariance matrix produces circular classes - i.e. the two dimensions are independent and have unit variance. The scaled identity covariance matrix also produces circular classes, but the magnitude of the variance of the classes can be adjusted, resulting in the different size circles. Then, diagonal matrices enable the classes to be pulled along the axes, resulting in axis-aligned ellipses for each of the classes. Finally, a free covariance matrix allows the classes to be modelled as arbitrary ellipses, that can be oriented in any way. The following table generalizes these 2-dimensional intuitions to an arbitrary d-dimensional space.

Covariance Constraint	Matrix Form	Class Distribution
identity	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	classes are identical spheres in a d-dimensional space (d-spheres)
scaled identity	$\begin{bmatrix} k & 0 & 0 & 0 \\ 0 & k & 0 & 0 \\ 0 & 0 & k & 0 \\ 0 & 0 & 0 & k \end{bmatrix}$	classes are variable 'size' d-spheres
diagonal	$\begin{bmatrix} k_1 & 0 & 0 & 0 \\ 0 & k_2 & 0 & 0 \\ 0 & 0 & k_3 & 0 \\ 0 & 0 & 0 & k_4 \end{bmatrix}$	classes are axis-aligned d-dimensional ellipsoids
free	$\begin{bmatrix} k_1 & k_2 & k_3 & k_4 \\ k_5 & k_6 & k_7 & k_8 \\ k_9 & k_{10} & k_{11} & k_{12} \\ k_{13} & k_{14} & k_{15} & k_{16} \end{bmatrix}$	classes are arbitrary d-dimensional ellipsoids

Table 1: Constraints on covariance matrices and corresponding class distributions

Thus, the constraints on the covariance matrix produces a nested range of models that specify the flexibility of multivariate Gaussian distributions corresponding to the classes. In order to stay in the scope of a single-semester project, we create the first three models in the range, corresponding to the identity, scaled identity and diagonal covariance constraints. This section explained the motivation to model classes as probability distributions, worked through the Gaussian Classifier, and examined the range of models to be built. The next section details the implementation of these nested models.

4 Implementation

This section provides a high-level overview of the implementation of this project. The three models to be implemented correspond to the nested covariance-constraints discussed in the last section. In all three cases, the classifier is integrated into the neural network by creating a custom layer, that performs the required Mahalanobis distance computation. These custom layers receive the convolutional neural network's embeddings of the training images as input, and they output the normalized class-membership likelihoods of the images. The layers learn the means and paramaters of the covariance matrices as their 'weights'. The loss is computed by measuring the discrepancy between the true distribution and the estimated distribution (using the cross-entropy function), and this error signal is back propagated through the network so that the network jointly learns image representations and Gaussian densities over classes. The following figure illustrates the abstract architecture of this network:

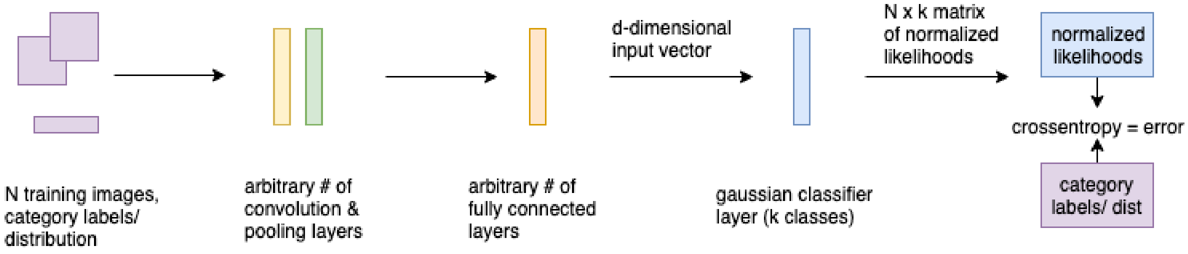
N training images, category labels/ distribution
arbitrary # of convolution & pooling layers
arbitrary # of fully connected layers
d-dimensional input vector
gaussian classifier layer (k classes)
N x k matrix of normalized likelihoods
normalized likelihoods
crossentropy = error
category labels/ dist

Figure 2: Abstract architecture of classifier-integrated network

4.1 Building a Custom Keras Layer

As mentioned before, the three custom layers to be built correspond to the covariance-constraints discussed in the approach section. This project uses Keras, a high-level neural networks API, with a TensorFlow backend. As noted before, the normalized likelihood vector for each image corresponds to computing the mahalanobis distance of each embedding from the mean of each class, and then applying to softmax activation to the layer. So, each classifier is a custom Keras ‘layer’ object that simply computes the required distance, with the native Keras ’softmax’ activation on top.

Turning our attention to the layers themselves, we specify the parameters that each layer learns. When the covariance is the identity matrix, the Mahalanobis distance computation reduced to Euclidean distance, so we label the first classifier the ’Euclidean distance layer’. The scaled identity matrix produces n-spheres scaled by some constant, so we label the corresponding classifier the ’Mahalanobis constant covariance layer’. Finally, the diagonal covariance matrix produce axis-aligned n-ellipsoids for classes, so we label the classifier ’Mahalanobis axis-aligned covariance layer’. The following table lists each of the custom layers and details the parameters it learns:

Classification Layer	Learnt Parameters
Euclidean distance layer	mean vector for each class, $\mu_c = [m_1 m_2 m_3 ... m_d]$
Mahalanobis constant covariance layer	mean vector for each class, $\mu_c = [m_1 m_2 m_3 ... m_d]$ scale constant for each class, s_c
Mahalanobis axis-aligned covariance layer	mean vector for each class, $\mu_c = [m_1 m_2 m_3 ... m_d]$ diagonal values of covariance matrix, $\mu_c = [a_1 a_2 a_3 ... a_d]$

Table 2: Parameters learned for each custom layer

4.2 Defining the Models

4.2.1 all-CNN Baseline

The All Convolutional Net (all-CNN) architecture was proposed by Springenberg, Dosovitskiy, Brox and Riedmiller in order to create a simple but powerful architecture using the principles that the vast majority of modern CNNs are built on. They came up with a “a homogeneous network solely consisting of convolutional layers, with occasional dimensionality reduction by using a stride of 2", and were able to achieve state-of-the-art performance without the need for “complicated activation functions" or “max-pooling".[Spr+15] The simple and parameter-efficient approach of this network makes it the ideal starting point for the neural network architectures for this project.

4.2.2 Modifying the Baseline

Two changes were made to the original all-CNN architecture in order to make a comparable and efficient baseline for the other custom models. First, batch normalization was added after every convolution to enable the network to train faster. Second, the original architecture performed classification in 192-dimensional space created by the convolutions. But, the custom classification layers perform classification in a 10-dimensional space (as the number of classes is 10). So, in order to make the models comparable, a simple fully-connected 10-dimensional layer was added to the original all-CNN architecture. In the case of the three custom models, this 10-dimensional fully-connected layer is simply replaced by the custom classification layers. The following figure lays out the differences in architecture between all models:

original all-CNN	custom models
3 x 3 conv. 96 ReLU 3 x 3 conv. 96 ReLU	3 x 3 conv. 96 ReLU batch normalization 3 x 3 conv. 96 ReLU batch normalization
3 x 3 conv. 96 ReLU (stride = 2) dropout	3 x 3 conv. 96 ReLU (stride = 2) dropout
3 x 3 conv. 96 ReLU 3 x 3 conv. 96 ReLU	3 x 3 conv. 96 ReLU batch normalization 3 x 3 conv. 96 ReLU batch normalization
3 x 3 conv. 96 ReLU (stride = 2) dropout	3 x 3 conv. 96 ReLU (stride = 2) dropout
3 x 3 conv. 96 ReLU 3 x 3 conv. 96 ReLU	3 x 3 conv. 96 ReLU batch normalization 3 x 3 conv. 96 ReLU batch normalization
global average pooling	global average pooling **dense 10** (baseline) **OR** **classification layer** (custom models)

Table 3: Modifying original all-CNN architecture

4.3 Datasets

The models are trained on the CIFAR-10 dataset, which consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. The dataset is divided into 50000 training images and 10000 test images.[Kri09] The following figure visualizes the 10 classes in the data, along with 10 random images from each:

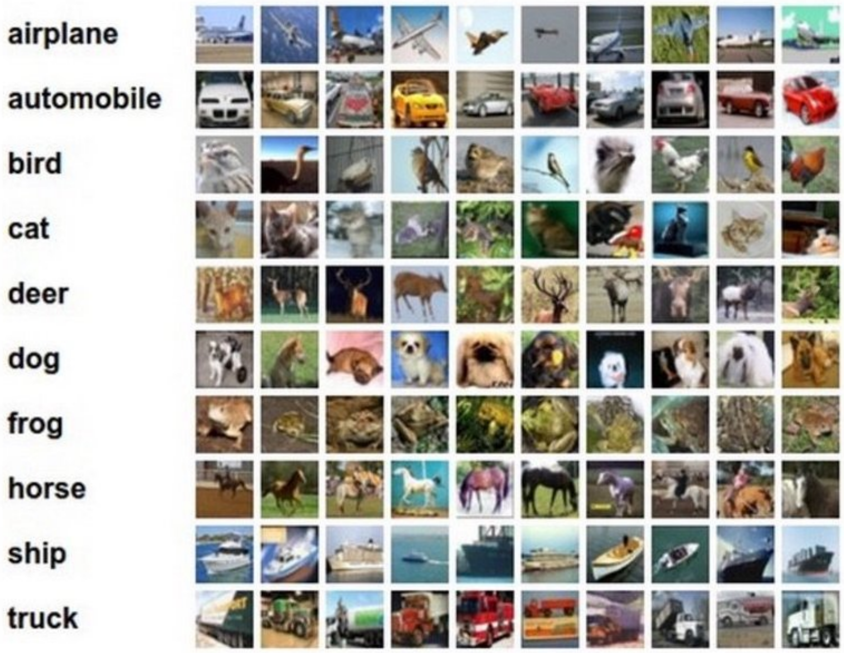

Figure 3: Visualization of the CIFAR10 dataset [Kri09]

In addition to testing the accuracy of these models on CIFAR-10, this project also tests the discrepancy between the distribution of human judgments and machine-learned judgments. As mentioned in the motivation, this is possible because of a novel dataset, CIFAR10-H, which comprises full label distributions for the entire 10,000-image CIFAR10 test set, utilizing over 500k crowd-sourced human categorization judgments.[BPG19] The following image visualizes the difference between the CIFAR-10 and CIFAR10-H labels:

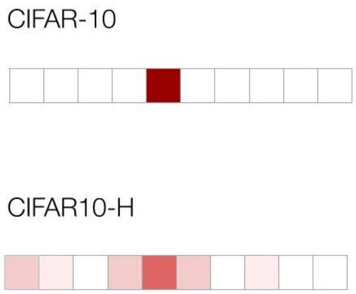

Figure 4: Visualizing CIFAR-10 vs CIFAR10-H labels

Thus, as CIFAR10-H contains a full distribution of judgments, we can compare the relative activation of humans and the machine systems for each image. In other words, cross-entropying the vector of human judgments and machine-learned judgments for every images gives us the discrepancy between them, and summing over these discrepancies produces a measure of the difference between humans and machines. Thus, each models cross-entropic loss on the CIFAR10-H dataset enable us to compare them on how human-like their judgments are.

5 Results

5.1 Training and Testing on CIFAR-10

All models were trained for 100 epochs on the CIFAR-10 training set, and subsequently tested on the validation set. The variation of training and test loss for all models, across epochs of training, is demonstrated below. The epoch with lowest test loss is marked with a dotted black line in each case.

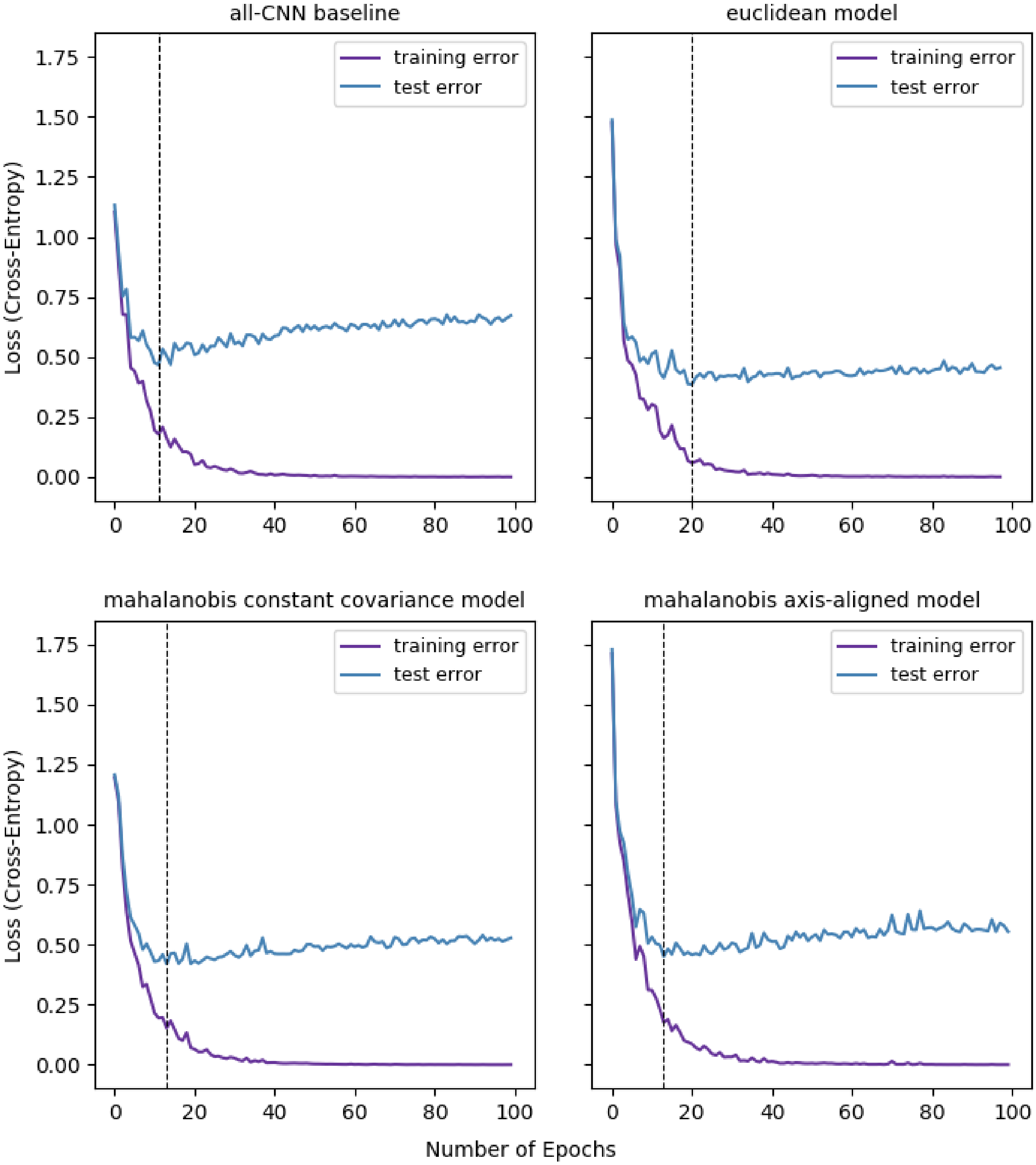

Figure 5: Training and test error on CIFAR-10 for all models

Similarly, the next figure illustrates the variation in accuracy (proportion of correct judgments) for all models, across epochs of training. The epoch with highest test accuracy is marked with a dotted black line in each case.

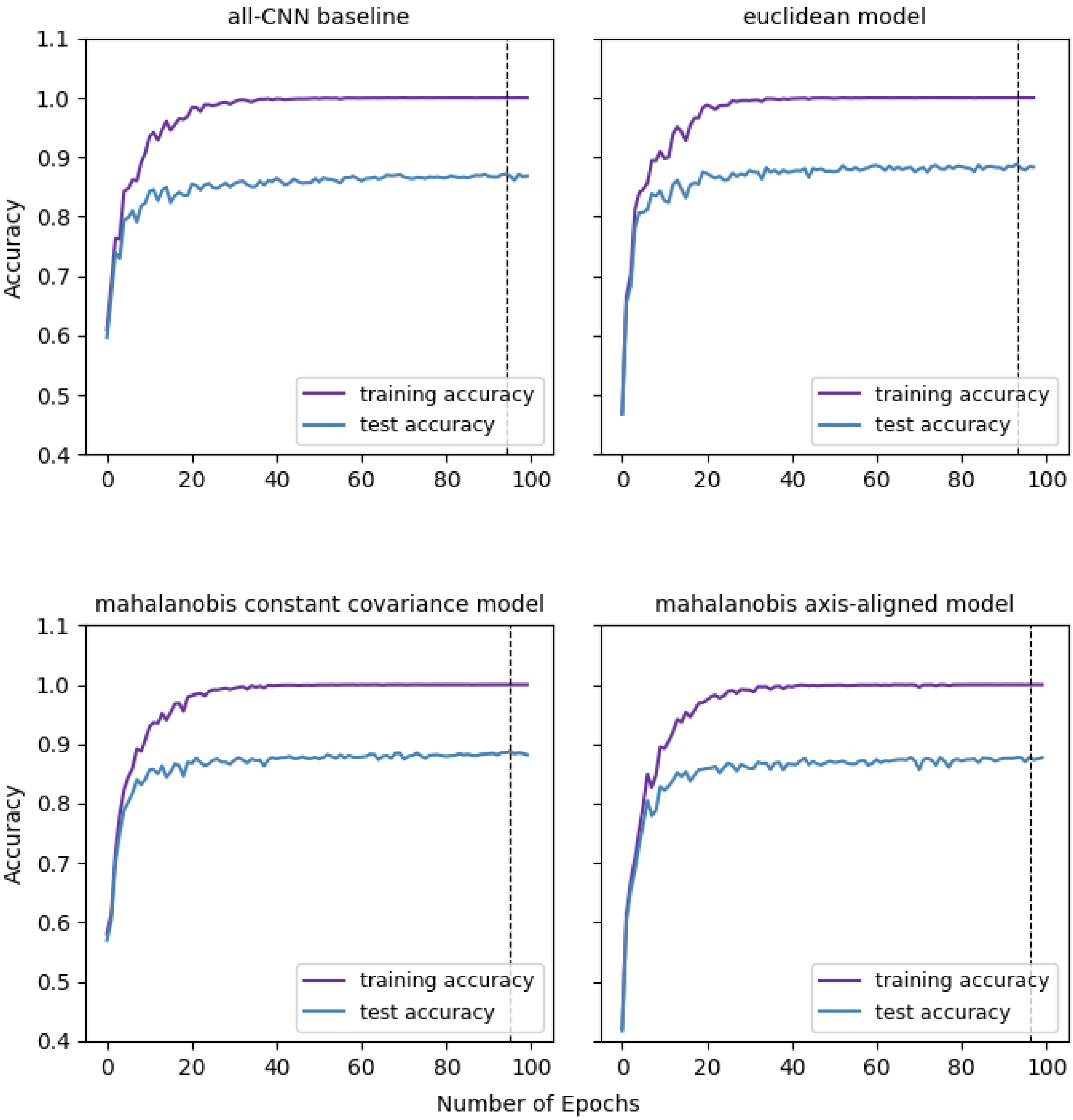

Figure 6: Training and test accuracy on CIFAR-10 for all models

In each case, we select the weights that maximize the test accuracy for each model, in order to perform further analysis. The following figure illustrates the selected weights and corresponding training and test accuracy for each model:

Model	Epoch	Training Accuracy	Test Accuracy
all-CNN baseline	94	0.99992	0.87190
euclidean distance model	93	1.00000	0.88860
mahalanobis constant-covariance model	95	0.99996	**0.88871**
mahalanobis axis-aligned model	96	0.99996	0.87830

Table 4: Selected weight and corresponding accuracy for each model

We see that all the models have near-perfect performance on the training data, and comparable performance of about 88% on the CIFAR-10 test set. The Mahalanobis constant-covariance model does marginally better than the Euclidean model, followed by the Mahalanobis axis-aligned model and the all-CNN baseline. Thus, we see that imposing constraints on the class-representations of the CNN does not impede test accuracy.

5.2 Testing on CIFAR-10H

Now, to compare how human-like the class representations are, we test the loss of the models (with the weights selected above) on the CIFAR10-H dataset. The following figure illustrates the CIFAR10-H loss and accuracy of all the models (with the weights selected above):

Model	CIFAR10-H Accuracy	CIFAR10-H Loss
all-CNN baseline	0.88250	1.01579
euclidean distance model	0.88310	**0.68827**
mahalanobis constant-covariance model	0.88570	0.85824
mahalanobis axis-aligned model	0.87780	0.89453

Again, we see that all models have comparable accuracy on human labels, of around 88%. But, we see that all models perform better than the baseline in terms of loss on human labels. The Euclidean distance model in particular, does drastically better than the baseline, followed by the two Mahalanobis distance models. In order to get a better sense on the changes in loss and accuracy on the human labels over the course of training, we evaluate all the models on the human data after every epoch. The results from this experiment are illustrated below. The lowest loss and highest accuracy are marked with a dotted black line in each case.

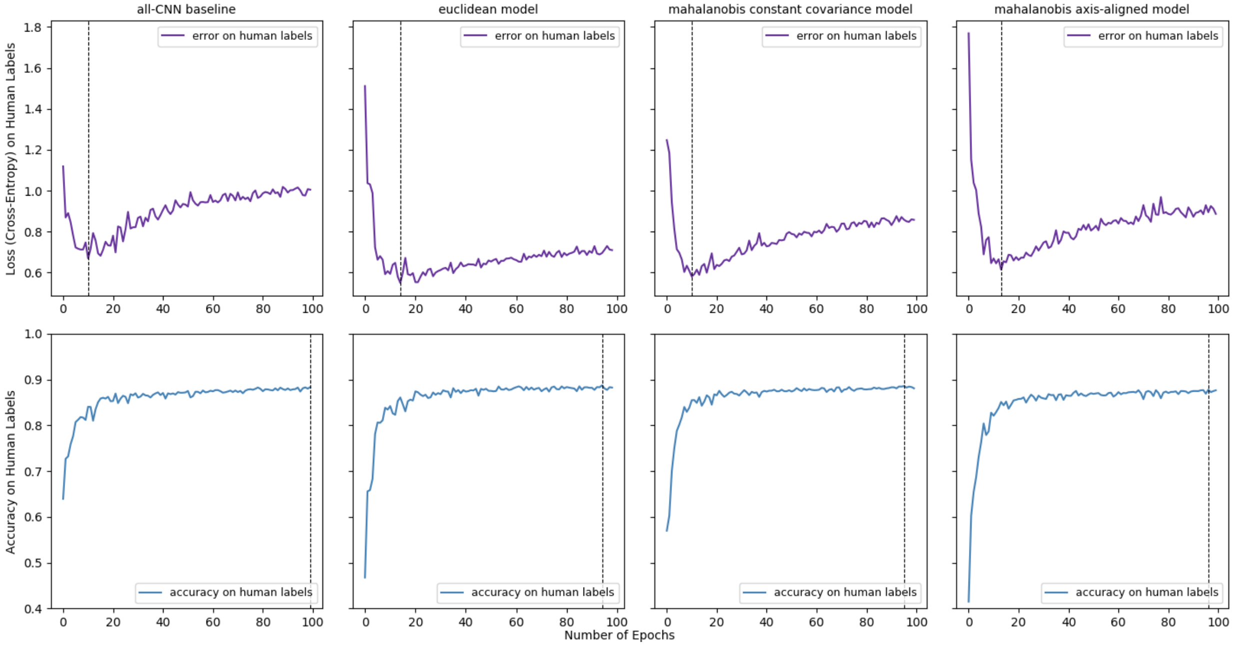

Figure 7: Loss and Accuracy on CIFAR10-H over the course of training

It is interesting to note that for all models the loss on human-labels dips very early-on in the training and then continues to increase as the model fits for accuracy. This increase happens most sharply for the baseline and most gradually for the euclidean distance model. Meanwhile, even though loss on human-labels has increased by the end of training, accuracy on human labels is highest towards the end. It is unclear why this pattern occurs across models, and it is hence pertinent to examine the minimum loss in each case. This is illustrated in the figure below:

Model	Minimum Loss on Human Labels	Epoch Num
all-CNN baseline	0.66922	10
euclidean distance model	0.54913	14
mahalanobis constant-covariance model	0.58046	10
mahalanobis axis-aligned model	0.61439	13

Table 5: Minimum loss on human labels across models

It seems that the baseline's minimum CIFAR10-H loss is a lot closer to the minimum loss of the other models, but the Euclidean model still does the best, followed by the two Mahalanobis distance models. These patterns warrant further investigation. One possibility could be that it is easier for the CNN to simply learn representations that take feature-scaling into account than to learn a covariance matrix, and this is why the Mahalanobis distance models perform worse than the Euclidean distance ones.

5.3 Visualizing Class Representations

In order to get a sense of what the range of models learn, we visualize the class representations. In other words, we take the 192-dimensional inputs to the classification layers in each of the models, and project them onto 2 dimensions in order to get a sense of the class representations in each case. This dimensionality reduction is performed using 3 different techniques - Principal Component Analysis, Linear Discriminant Analysis and t-distributed Stochastic Neighbor Embedding. The resulting visualizations are plotted below:

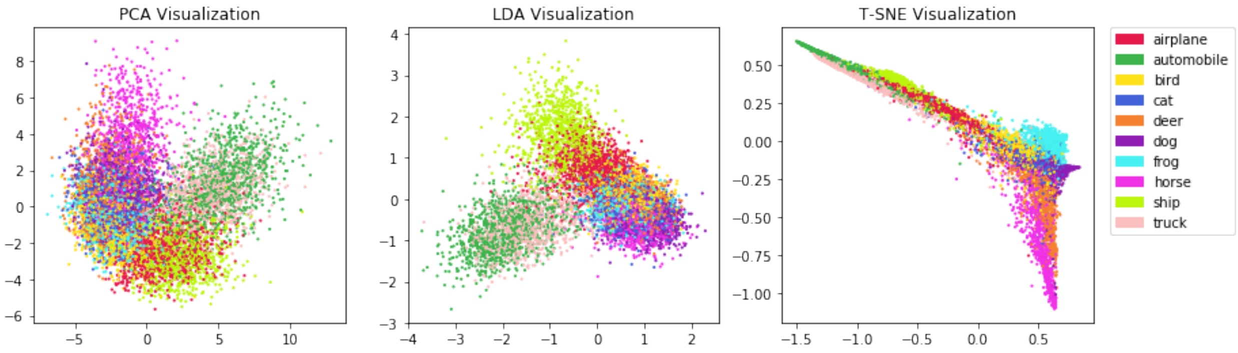

Figure 8: Visualizing class representations of Baseline Model

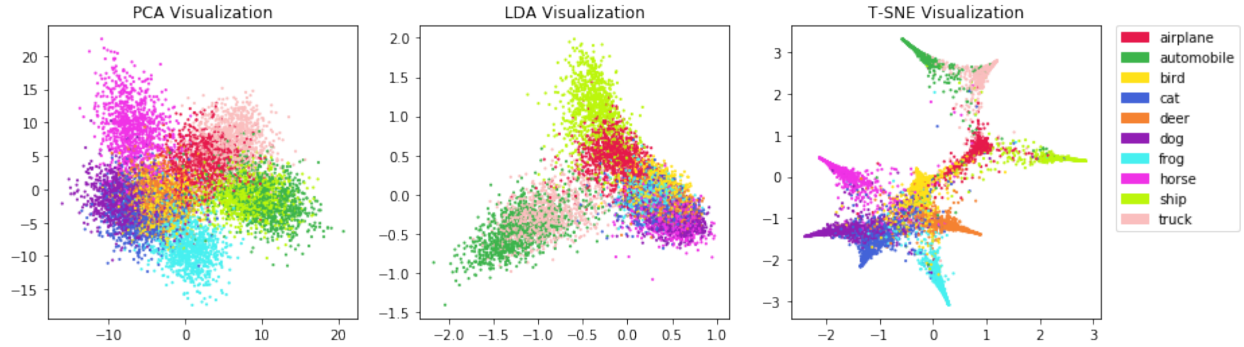

Figure 9: Visualizing class representations of Euclidean Distance Model

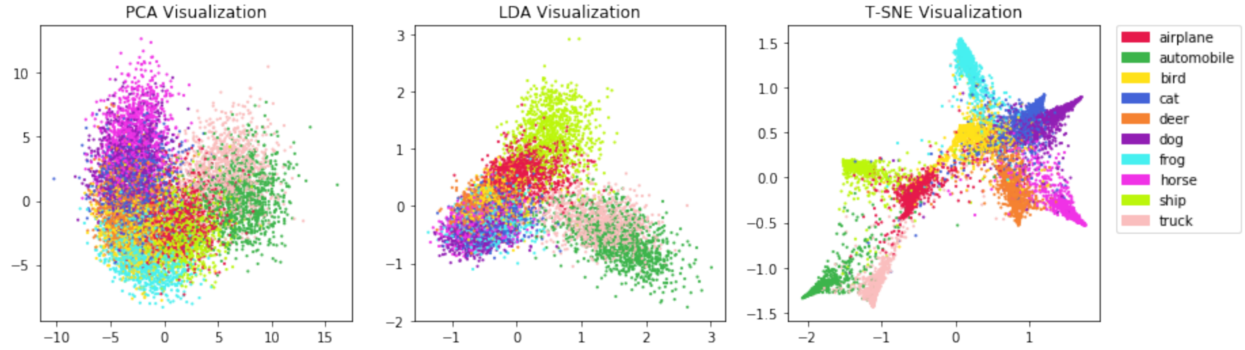

Figure 10: Visualizing class representations of Mahalanobis Constant-Covariance Model

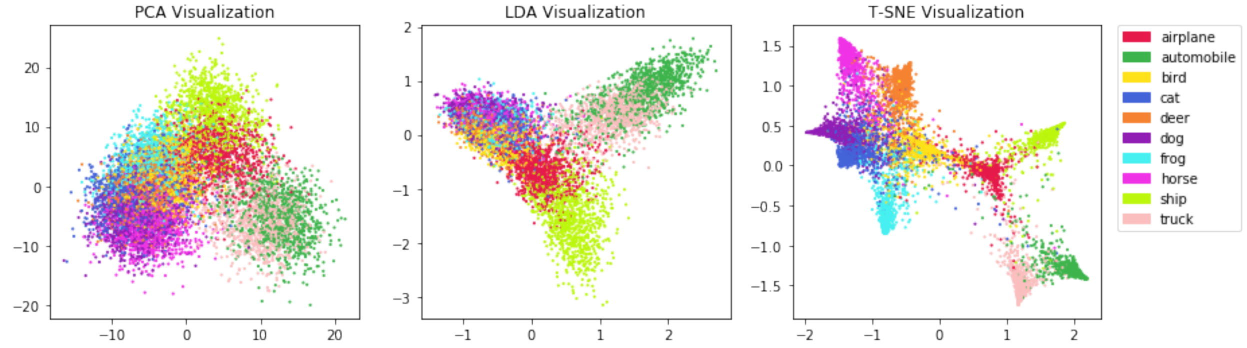

Figure 11: Visualizing class representations of Mahalanobis Axis-Aligned Covariance Model

It seems that the baseline is a bit more scattered than the rest of the models. One interesting point to note is the Principal Component Analysis produces the most separated results in the Euclidean model. This indicates that the Euclidean model is more separable either on all dimensions or on the dimensions of most overall variation (across all images in all categories) than the other models.

6 Evaluation and Further Work

This project has shown that imposing class representations on discriminative models can produce more human-like representations of classes. There are many questions to be answered, including why the loss across models dips early on in the training and the reason the Euclidean model performs best on human labels. One idea for future work would be to compare these models to those that aim for low intra-class variability - like center less described above - in order to better understand the optimal representations of classes. In addition, these classifier layers should be embedded into models of different architectures, and trained on larger datasets to test the replication of behaviour.

7 Acknowledgments

I would like to acknowledge my advisor Tom Griffiths for his continued support and encouragement. I am also incredibly grateful to Josh Peterson and Ruairidh Battleday for their mentorship and contributions to this project. I would also like to thank my family and friends for supplying me with endless amounts of moral support and caffeine.

References

[PK68] Posner and Keele. *Prototypes*. 1968. URL: https://coglab.cengage.com/labs/prototypes.shtml.

[Kri09] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. 2009. URL: https://www.cs.toronto.edu/~kriz/cifar.html.

[Spr+15] Jost Tobias Springenberg et al. "Striving for Simplicity: The All Convolutional Net". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*. 2015. URL: http://arxiv.org/abs/1412.6806.

[Wen+16] Yandong Wen et al. “A Discriminative Feature Learning Approach for Deep Face Recognition”. In: *ECCV*. 2016.

[Zha+16] Chiyuan Zhang et al. “Understanding deep learning requires rethinking generalization”. In: *CoRR* abs/1611.03530 (2016). arXiv: `1611.03530`. URL: `http://arxiv.org/abs/1611.03530`.

[PAG17] Joshua C. Peterson, Joshua T. Abbott, and Thomas L. Griffiths. “Adapting Deep Network Features to Capture Psychological Representations: An Abridged Report”. In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence* (Aug. 2017). DOI: `10.24963/ijcai.2017/697`. URL: `http://dx.doi.org/10.24963/ijcai.2017/697`.

[SSZ17] Jake Snell, Kevin Swersky, and Richard S. Zemel. “Prototypical Networks for Few-shot Learning”. In: *CoRR* abs/1703.05175 (2017). arXiv: `1703.05175`. URL: `http://arxiv.org/abs/1703.05175`.

[Hua+18] Yanping Huang et al. “GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism”. In: *CoRR* abs/1811.06965 (2018).

[BPG19] Ruairidh M Battleday, Joshua C Peterson, and Thomas L Griffiths. “Capturing human categorization of natural images at scale by combining deep networks and cognitive models”. In: *arXiv preprint arXiv:1904.12690* (2019).