Math 521 Final Project

Y. Dawit, J. DeSimone, R. Mohanty, M. Raabe-Lopez

September 12, 2023

Abstract

Our project concerns the implementation of a variety of mathematical and statistical methods to classify novel images of cats and dogs. Given a training set of 160 known images, we wish to classify the remaining 38 out of a total set of 198. After stating the problem and the available data set clearly, we will devise our methodology and formulate classification algorithms. The theory will then be implemented using numerical computation in Python and MATLAB. After this work is done, we will compare and contrast our methods and analyze the efficiency and accuracy of each of them.

1 Objective and Available Data Set

The MATLAB file *"PatternRecData.mat"* contains a 198×198 matrix *"KLDATA.mat"* and a row vector *"sub-labels"* of length 160. The data matrix *"KLDATA"* contains images of cats and dogs, courtesy of Dave Bolme and Dr. Ross J. Beveridge of the Department of Computer Science, Colorado Stare University (3). 99 images of each animal are placed at random throughout the columns of the data matrix, and the *"sub-labels"* vector identifies the first 160 of these images by the convention cat = 1 and dog = 0.

The file *"TIFFtraining.zip"* contains TIFF-format images of the first 160 patterns in *"KLDATA.mat"*. Each image is of size 64×64, except for *"Dog96.tif"*. Due to an error in the data set, this file consists of a $64 \times 64 \times 2$ matrix, where only the first layer is to be used.

With these data sets in place, our goal is to build different pattern recognition architectures in MATLAB and Python, in order to study their various advantages and disadvantages in practical application. For each method chosen, a set of novel images will be classified as either a cat or a dog, and the accuracy and efficiency of the performance will be discussed.

2 Methodology and Theoretical Background

Toward the goal of providing a detailed comparison of a variety of pattern recognition algorithms, we begin by listing our methods.

1. **Kohonen's Novelty Filter**
2. **Fischer's Linear Discriminant Analysis**
3. **Support Vector Machines**
4. **CNN trained using VGGNet**

Kohonen's Novelty Filter:

A relatively simple method for classifying images is *Kohonen's Novelty Filter*, which works by splitting image data into two components. One component lies in a subspace spanned by a set of training data, and the other component lies in a subspace orthogonal to the first, representing the

portion of the data which is novel. In the case of classifying cats from dogs, we have two training sets, and so we may use a novelty filter algorithm separately on each training set and compare the performance. If a probe image has more novelty with the cat training set than the dog training set, then its features are more consistent with that of a dog than of a cat, and we predict dog. Likewise, if a probe image has more novelty with the dog training set, then we predict cat.

Convolutional Neural Networks:

Artificial neural networks (ANN) or connectionist systems are computing systems vaguely inspired by the biological neural networks that constitute animal brains.(citation?) Such systems "learn" to perform tasks by considering examples, generally without being programmed with task-specific rules. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as "cat" or "no cat" and using the results to identify cats in other images. They do this without any prior knowledge of cats, for example, that they have fur, tails, whiskers and cat-like faces. Instead, they automatically generate identifying characteristics from the examples that they process. A *convolutional neural network* (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. They are also known as shift invariant or space invariant artificial neural networks, based on their shared-weights architecture and translation invariant characteristics. They have applications in image and video recognition, recommender systems, image classification, medical image analysis, natural language processing, and financial time series. (cite wiki)

VGG net is one such famous pretrained convolutional neural network developed by the *Visual Geometry Group* at Oxford. The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. It was one of the famous model submitted to *ILSVRC-2014*. It makes the improvement over *AlexNet* by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another. *VGG16* was trained for weeks and was using NVIDIA's *Titan Black* GPUs.(cite neurohive link)

The *VGG16* architecture is shown in Figures 1 and 2.

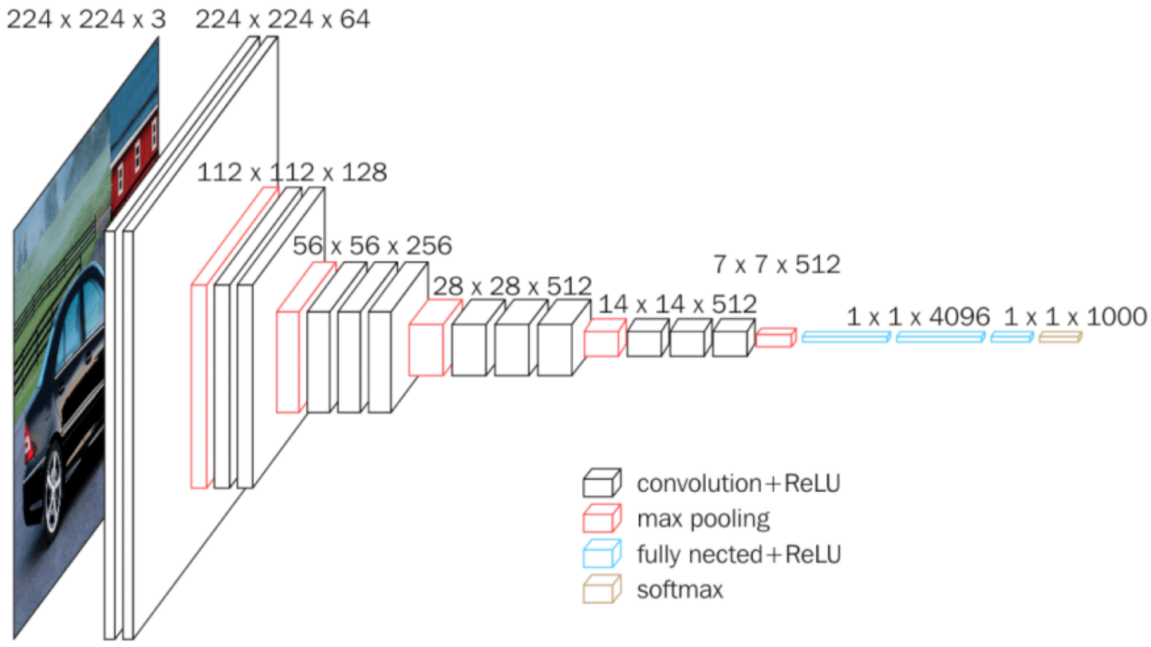

Figure 1: Visualization of the *VGG16* architecture in terms of tensors.

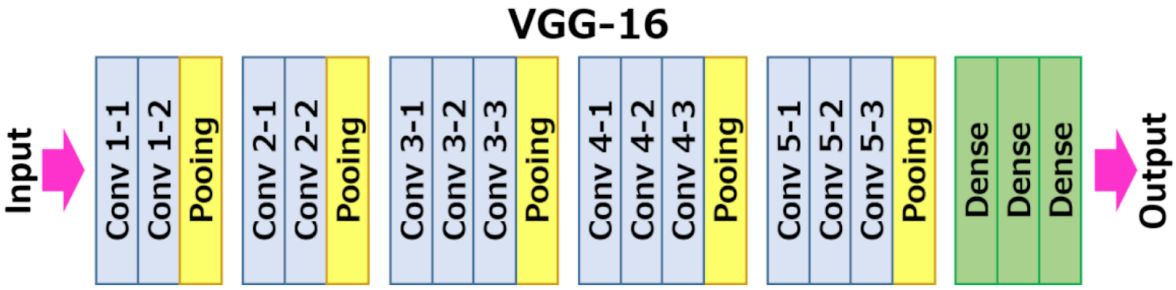

Figure 2: Visualization of the *VGG16* architecture in terms of filters.

FDA and LDA:
Fisher's Discriminant Analysis is a classifier that finds the weights which maximize the separation of points between classes and minimize the points within each class; the formula and details to attain w is explained in the section **Class Discrimination**. It is a form of dimensionality reduction because the weights calculated by FDA are projected onto the data points. In this project, the dimension D is one since the D is restricted by the number of classes(2) minus 1. Once the data points are projected, a threshold is defined for classification. There are several ways to decide on a threshold; for this experiment, we have chosen to take the mean of the projected points of each class and average them.
Fisher's Linear Discriminant Analysis(LDA) is an extension of FDA. The additional term 'linear' was added to address how the LDA method differs from FDA from the class lecture. Linear in LDA is intended to build a linear discriminant by using a multi-class gaussian distribution instead of defining a threshold. The distribution acquires the means, covariance matrix, and prior probabilities. LDA classifies an observation/flatten image by calculating the likelihood of each class from the multi-class distribution and picks the class with the highest likelihood.

Support Vector Machine:
Support Vector Machine is supervised learning algorithm that finds the hyperplane with the best separation between classes. The nature of how the hyperplane is found involves solving an optimization problem. The set up is to minimize $\frac{1}{2}w^Tw$ under the constraint $y_i(w^Tx_i+b) \geq 1$ where w is the vector of weights that form the hyperplane, x_i is the support vectors, b is the bias, and y_i is the class labels 1,-1. The distance from any point x_j to the hyperplane is $\frac{|w^Tx_j+b|}{||w||_2}$. Since the margin is the width of the boundary before hitting a data point (support vector), the margin is $\frac{2|w^Tx_i+b|}{||w||_2} = \frac{2}{||w||_2}$. Defining the margin this way explains why the minimizing $\frac{1}{2}w^Tw$ is an approximate objective function.

The idea is to use the support vectors to obtain weights that generate a large enough margin for new data to be within the boundary. In short, SVMs look for the margin of correctness in hopes the testing set can classified within the buffer and avoid the boundary of misclassified region. The loss function used to support this idea is the hinge loss defined as $l(w,b) = \frac{1}{2}||w||_2^2 + C\sum_{i=1}^{N} max\Big(0, 1 - (w^Tx^{(i)}+b)y^{(i)}\Big)$ where C is the penality/cost term and $y^{(i)}$ is the correct label of a given observation i. The influence of hinge loss is C. If the size of C is large, then the size of the weights becomes restricted. If the term $w^Tx^{(i)}+b$ has the same sign as $y^{(i)}$, then the summation term gets closer to 0. But if the signs are different, the loss increases. Thus, C restricts the size of w.
This explanation of SVM so far is the linear transformation or no changes in the data. If the data is not linearly separable, then we could change the kernel of the input space. In simple terms, we change

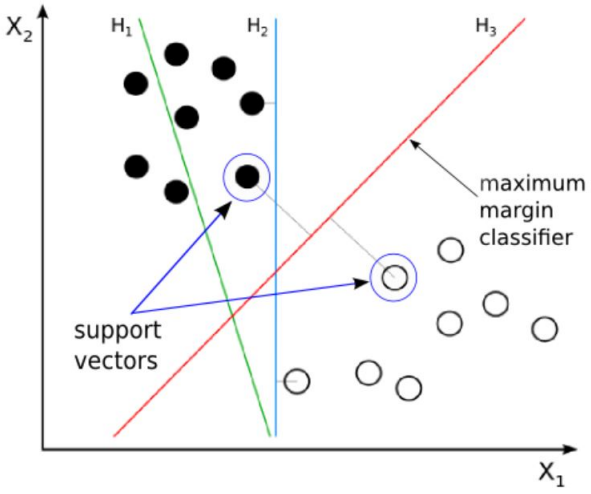

Figure 3: The figure is an example of hyperplanes of each iteration H_i on the features X_1 and X_2. The black and white points represent the classes; the two points with a blue circle are the support vectors. The goal of SVM is to maximize the minimum distance between the support vectors and the hyperplane. At the first iteration H_1, the separation is poor between classes. The second iteration H_2 did good to separate the classes but the margin is small since the black support support has a short distance from the hyperplane. The third iteration H_3 did well to maximize the minimum margin.

the x_j in the optimization problem into the transformed space $u_j = f(x_j)$. The only change in the setup is the constraint $y_j(w^T x_j + b) \geq 1$ being replaced with $y_j(w^T u_j + b) \geq 1$. However; each kernel have additional hyperparameters to consider other than the cost parameter C. There are 3 common kernels to choose from if the linear kernel does not work which are specified in the Python SkLearn library. One is the gaussian kernel or normal distribution; it requires a hyperparameter that specify the shape of the normal distribution. Second is the polynomial kernel which expresses transformed space as polynomial in terms of x where the power, the lead coefficient and the intercept need to be specified before training. Third is the sigmoid kernel which uses tanh function as the transformation for x; the hyperparameters available to adjust are the scale coefficient and the intercept where both terms are inside the tanh function.

Optimal Orthogonal Representations

As discussed in the course and [Geometric Data Analysis], [Patter Recog. & ML], a primary concern of researchers is how best to represent a collection of data such that predictions or classifications will have a high degree of accuracy and require minimum storage capacity on local machines. For our purposes we focus on the utilization of a Karhounen-Loueve Transform and its associated basis. We follow the general notation as used in the course and [low-D char. of human faces]. Given some set of pattern vectors or more concretely, a series of sampled data vectors. For instance a series of images could be loaded as concatenated column vectors, $X = \sum_{j=1}^{p} x_j = [x_1^T, x_2^T, \ldots, x_p^T]$. Where each x_j is an individual image. The mean subtracted data provides us a second moment matrix, which is amenable to the Karhounen-Loueve Transform. Which is then used to find a suitable eigenbasis.

$$\langle (X - \langle X \rangle)^T (X - \langle X \rangle) \rangle = \left\langle \tilde{X} \middle| \tilde{X} \right\rangle = C = \frac{1}{P} \sum_{n=1}^{P} \varphi^{(n)} \varphi^{(m)} \quad \text{i.e. Cov(X)}$$

From this ensemble averaged data, an eigenbasis is then extracted:

$$C = \Phi\Lambda\Phi^T \implies C^{-1} = \Phi\Lambda^{-1}\Phi^T$$

$$\text{And for each individual data vector:} \quad \tilde{\mathbf{x}}^\mu = \mathbf{x}^\mu - \langle\mathbf{x}\rangle = \mathbf{x}^\mu - \frac{1}{P}\sum_{\mu=1}^{P}\mathbf{x}^\mu$$

$$\text{And in the language of the SVD:} \quad C = U\Sigma V^T$$

This basis is used to find the maximal variance projection for novel data points into the sample space.

$$U^T X = \langle(\mathbf{a}, \mathbf{x})\rangle$$

Where we may state that given an N-dimensional basis derived from a corpus of data, $\mathcal{B}$, a D-term approximation may be used to represent novel data points:

$$\mathbf{x}_D^\mu = a_1\varphi_1 + \cdots + a_D\varphi_D$$

The corresponding continuous, eigenfunction problem is well-founded. There exists an optimal orthogonal representation of a set of data, which may be considered roughly as level set functions associated to points in a space. This representation is found by solving the following problem:

$$\int K\{\mathbf{x}, \mathbf{y}\}\psi_n(\mathbf{y})\,\mathrm{d}y = \lambda_n\psi(\mathbf{x}) \quad \text{where} \quad \int \psi_m(\mathbf{x})\psi_n(\mathbf{x})\,\mathrm{d}x = \delta_{nm}$$

Which we recognize as the standard inner product representation of a continuous vector valued function, $(f, g) \in L^2[\Omega]$. For which we have minimal error as a given constraint on the set of representation functions:

$$\varepsilon = \left\langle \left\| \varphi - \sum_{n=1}^{M} a_n\psi_n(\mathbf{x}) \right\|^2 \right\rangle$$

And the integral kernel is defined as:

$$K\{\mathbf{x}, \mathbf{y}\} = \langle\varphi(\mathbf{x}), \varphi(\mathbf{y})\rangle = \frac{1}{N}\sum_{n=1}^{N}\varphi(\mathbf{x})\varphi(\mathbf{y})$$

Which is recognized as the 2 point correlation function. Realizing this operator as the dyadic or outer vector product of an unnormalized or sample basis produces then a decomposition into projection operators of rank 1. We now see that using the covariance and vectorized forms of our sample data produces a mapping from a sample space to a mean-subtracted feature space by projection operators:

$$\varphi(\mathbf{x}) \cdot \varphi(\mathbf{y})^T = \mathbb{P}_\Omega \in \mathcal{M}^{n\times n} \quad \text{which is the space of n-by-n orthogonal matrices}$$

Since any projection operator can be used to decompose a space:

$$V = \mathcal{R}(\mathcal{P}) \bigoplus \mathcal{R}(\mathbb{I} - \mathcal{P})$$

Using these orthonormal vectors, we can also see that, for $W_i = span(\varphi_i)$:

$$V = \bigoplus_{i=1}^{n} W_i \quad \text{where we have that} \quad W_i \bigoplus W_j = 0$$

Thus, encoding a concrete data set into vectorized form and applying either the snap-shot or direct method of the KL Transform provides a suitable basis. Truncating by a suitable condition like the "max-energy" approach or otherwise reduces our basis from N to D vectors. We now have a

D-dimensional optimal basis for some inner-dimensional product space V equipped with an ordered orthonormal basis: $\mathcal{B} = \{\varphi_i\}_{i=1}^{N}$. Then, there is a representation for every point without error in terms of this basis.

$$\mathbf{q}^{(\mu)} = a_i^{(\mu)}\varphi_i + \dots a_N^{(\mu)}\varphi_N$$

And a data set, D-dimensional, which may be approximated by a truncation of the N-dimensional basis.

$$\mathbf{q}^{(\mu)} \approx \mathbf{q}_D^{(\mu)}$$

Furthermore, if $\mathbf{q}$ has an absolutely continuous probability density function $f_X(\mathbf{q})$, then we may also have recourse to the continuous formulation. For our purposes again, the expectation is simply the ensemble average:

$$\langle \mathbf{q} \rangle = \frac{\sum_{\mu=1}^{P} \mathbf{q}^{(\mu)}}{P}$$

The mean subtracted ensemble, or fluctuating field, or caricature is:

$$\tilde{\mathbf{q}}^{(\mu)} = \mathbf{q}^{(\mu)} - \langle \mathbf{q} \rangle$$

We now observe that for any approximation scheme which can be represented as the minimization of a D-dimensional approximating difference:

$$\langle \varepsilon \rangle = \left\langle \left\| \mathbf{q} - \mathbf{q}_D^{(\beta)} \right\| \right\rangle$$

Where β is a potential scheme parameter represented as P-dimensional, data-dependent variable. We see then:

$$\mathbf{q}^{(\mu)} = \mathbf{q}_D^{(\mu)} + \varepsilon_D^{(\mu)}$$

Which is the D-dimensional splitting of the larger sample space by an orthogonal projection. That is:

$$V = W_{D_i} \bigoplus_{i=1}^{r} W_{D_i}^{(\mu)} + \bigoplus_{i=r+1}^{n} W_{\varepsilon}$$

Since our representations may depend upon sample data or the ambient space in specific ways relating to the projections.

In particular, for the Kohonen Novelty Filter, these orthogonal projections are applied to split "novelty" from an already trained data set. That is, since any pattern in a corpus of data may be decomposed into two parts by an orthogonal projection:

$$\mathbf{x} = \mathbf{x}_{\perp} + \mathbf{x}_{\parallel}$$

$$\mathbf{x} = AA^{-1}\mathbf{x} + (\mathbb{I} - AA^{-1})\mathbf{x} \quad \text{where} \quad A \doteq \sum_{i=1}^{P} \mathbf{x}_i$$

This method shows how the concept of orthogonal projections connects LDA, Kohonen Novelty, and the KL Basis since how we construct the matrix AA^{-1} is implementation dependent.

Neural Networks: General Feed-Forward Model

We motivate this section by initially considering the possibility of a set of basis functions which may be used to represent data in an ambient space which is convenient for computing and prediction. Thus for some *non-linear* basis functions $\varphi_j(\mathbf{x})$:

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \varphi_j(\mathbf{x})\right)$$

Where, $f(\cdot)$ is a nonlinear activation function and $y(\cdot)$ is a weighted sum of linear combinations of the transformed variable. We will be primarily interested in developing the w_j and φ_j into a form that can be parametrized for corpus specific data. The basic neural network model is thus a series of functional transformations. Construct M linear combinations:

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

Where we will let $j = 1, \ldots, M$, so that we get the matrix $A^{(1)}$, where the superscript will represent the first "layer" in our network model. The parameters w_{ji} are weights and w_{j0} is a bias or intercept term. Each individual a_j is an activation term which will be transformed by a differentiable, nonlinear activation function $h(\cdot)$, $z_j = h(a_j)$. These new quantities will become the "hidden" layers or hidden units. Typically these are in practice sigmoidal functions like tanh or the logistic sigmoid but may be chosen differently. Now we have the following layer:

$$A^{(2)} \longmapsto a_k = \sum_{i=1}^{D} w_{kj}^{(1)} z_j + w_{k0}^{(1)}$$

We note here that for standard regression problems the activation function is simply the identity, and for logistic regression it is the sigmoid: $y_k = \sigma(a_k)$.

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

For more advanced models we have, in addition to an input activation, also an output activation. This may differ from the input in type or purpose. Thus, a full model has many parameters which may be fitted and adjusted in designing a particular solution for a specific corpus of data. For a two-layer network:

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma \left(\sum_{j=1}^{M} w_{kj}^{(2)} h \left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right)$$

Some simplifications, such as absorbing the bias terms into the summation, and natural abstraction process allows us to consider networks which are rough correspondences to:

$$\text{input corpus data} \quad = (\mathbf{x}, \mathbf{w}) \longmapsto A^{(1)} \longmapsto \ldots \longmapsto A^{(n)} \longmapsto y(\mathbf{x}, \mathbf{w}) = \quad \text{transformed output}$$

For training such a model some necessary conditions will be required. Our primary concern is first defining a "cost" function, or in more abstract terms we need to posit some Lagrangian formulation for the optimization problem. This corresponds to making certain assumptions about the natural ambient space which our data resides in and possible feature spaces which allow us to make predictions and classifications. Under the standard maximum likelihood model from statistics we will note that the mean squared norm and its related entropy maximization formulations provide us a natural framework for an error or cost function:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \| \mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n \|^2$$

Where the *target* values, $\mathbf{t}_n$, are chosen from the corpus of initial data to represent idealized output from our neural network. The ordinary least squares criterion can be used to solve a wide class of problems but it will fail for many classification problems. This is due to the fact that it is the maximum likelihood estimator under the assumption of a Gaussian conditional distribution. However given even a *binary* target vector, we will be interested in resolving a distribution which is far from Gaussian. Solving the multi-class problem with potentially non-linear boundaries in feature space can be logically considered an extension of Fisher's Linear Discriminant. The umbrella term for solvers, non-linear or otherwise, is maximum margin classfiers.

Class Discrimination

For some input vector, perhaps suitably approximated in D-dimensions, we project down onto one vector by the following:

$$\mathbf{y} = \mathbf{w}^T\mathbf{x} \quad \text{with} \quad \mathbf{y} \geq -w_0 \implies \quad \text{class} \quad \mathcal{C}_1 \quad \text{Or else} \quad \text{class} \quad \mathcal{C}_2$$

The defining criterion for Fisher's Discriminant being a functional of the form:

$$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2} \quad \text{with} \quad s_i^2 = \quad \text{sample variance for class i} \quad = \sum_{n \in \mathcal{C}_k} (y_n - m_k)^2$$

This criterion also has the following representation:

$$J(\mathbf{w}) = \frac{\mathbf{w}^T S_B \mathbf{w}}{\mathbf{w}^T S_W \mathbf{w}}$$

Since we are mostly interested in only the direction of a separation vector and not the magnitude we will state that:

$$\mathbf{w} \propto S_W^{-1}(\mathbf{m}_2 - \mathbf{m}_1)$$

To tie all of the theory together we close with the framework covered in class which relates the learning methodology for a non-linear boundary in neural networks to the classical linear approaches like FDA and LDA. This is otherwise known as a Kernel Basis or the "Kernel Trick". For our purposes, we begin with a covariance matrix, where we assume the data has already been mean-subtracted.

$$C\mathbf{x}_i = \lambda_i \mathbf{x}_i \quad \text{where} \quad C \doteq \left\langle XX^T \right\rangle$$

We also have:

$$C\{\mathbf{x}, \mathbf{y}\} = \langle \varphi(\mathbf{x}), \varphi(\mathbf{y}) \rangle = \frac{1}{N} \sum_{n=1}^{N} \varphi(\mathbf{x})\varphi(\mathbf{y})$$

$$\mathbf{x}C\mathbf{v}_j = \lambda_j \mathbf{x}_j \mathbf{v} \quad \text{where } \mathbf{v} \text{ is any eigenvector}$$

$$\lambda_i \mathbf{v}_i = \frac{1}{N} \sum_{n=1}^{N} \varphi(\mathbf{x}_n)\{\varphi(\mathbf{x}_n)\mathbf{v}_i\} \quad \text{Where} \quad \mathbf{v}_i = \sum_{a_i n} \varphi(\mathbf{x_n})$$

$$\text{The key step is then:} \quad \underbrace{\sum_{n=1}^{N} \varphi_n \varphi_m}_{\text{K}} \sum_{m=1}^{N} a_{im} \underbrace{\varphi_n \varphi_m}_{\text{K}} = \lambda_i \sum_{n=1}^{N} a_{in} \varphi_n$$

$$K^2 a_i = \lambda_i K N a_i \implies K a_i = \lambda_i N a_i$$

Thus, for any transformation which can be represented in a product form of eigenfunctions we may use the inverse of this transform as a mapping. This method applies widely to Kernel Basis functions, or the many classification methods. In particular, Neural Networks rely on this representation method and Mercer's Theorem.

3 Implementation of Algorithms

Before a given pattern recognition algorithm can be implemented in a given programming language, the matter of breaking down the theory into a numerical recipe must be considered. Here, we will discuss implementation of each of our methods. The exact codes used for our chosen languages will be given in section 5.

Kohonen's Novelty Filter:

The reader is assumed to be familiar with the theoretical underpinning of *Kohonen's Novelty Filter*, following the discussions in (cite textbook) and (cite Kohonen's paper.) We will focus on an adaptation of the original algorithm to the case of classifying among multiple training sets. The following algorithm was used for our analysis.

Algorithm 1 (Classification by Novelty Filter)

Input:

Cat training data set T_C.
Dog training data set T_D.
A set of probe data X.

Output:

A prediction vector **Pred**.

1. Using PCA, construct an orthonormal basis U_C for T_C, and U_D for T_D.

2. Obtain the complementary orthogonal matrices P_C and P_D given by $P_C = I - U_C U_C^T$ and $P_D = I - U_D U_D^T$.

3. For each $\mathbf{x}^{(\mathbf{k})} \in \mathrm{Col}(X)$, compute the projections $\mathbf{x1} = P_C \mathbf{x}^{(\mathbf{k})}$ and $\mathbf{x0} = P_D \mathbf{x}^{(\mathbf{k})}$.

4. If $\mathrm{norm}(\mathbf{x0}) > \mathrm{norm}(\mathbf{x1})$
   set $\mathbf{Pred}^{(k)} = 1$.
   Else
   set $\mathbf{Pred}^{(k)} = 0$.

Convolutional Neural Networks:

A pretrained VGG 16 model is publicly available. So, we utilize it as the starting layers of the model. This is called *transfer learning*. Our implementation can be summarized by the following Python code snippet, which fits the model and returns the accuracy:

```
def vgg_cat_dog_grey_m_h(l1_units,l1_d,epochs,batch_size,lr):
    model = models.Sequential()
    model.add(conv_base)
    model.add(layers.Flatten())
    model.add(layers.Dense(int(l1_units), activation='relu',kernel_regularizer=l2(0.001)))
    model.add(layers.Dropout(l1_d))
    model.add(layers.Dense(1, activation='sigmoid'))

    conv_base.trainable = False

    model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=lr),
              metrics=['acc'])

    history=model.fit(X, y, epochs=int(epochs), batch_size=int(batch_size), validation_data=(X_test, y_test), verbose=0)
    results = model.evaluate(X_test,y_test)
    return results,model,history
```

VGG16 layers (Non trainable)

Trainable layer

Dropout regularization

For binary classification

Figure 4: Code Snippet of CNN Implementation

Algorithm 2 (Classification by CNN using VGGNet)

Input:
Cat and Dog Training set
Test Set
Tunable Parameters, including convolution base
Output:
A CNN Model, Results, History of Training Parameters

1. Using Pre-Trained VGGNet, implement trainable layers
2. Implement a ReLu and train a Classifier Layer
3. Apply Dropout Regularization until Convergence
4. Apply Bayesian Hyper parameter Tuning

The tunable hyperparameters for model training are:

1. *l1_units*: Number of units in our dense/fully connected layer.
2. *l1_d*: The dropout probability our fully connected layer.
3. *epochs*: The number of training passes through the training dataset.
4. *batch_size*: The batch size for minibatch gradient decent.
5. *lr*: The learning rate for gradient decent.

How do we know which setting of these hyperparameters will work best for our dataset? For that we employ *Bayesian Optimizations. Bayesian optimization* is a sequential design strategy for global optimization of black-box functions that do not require derivatives. Since the objective function is unknown, the Bayesian strategy is to treat it as a random function and place a prior over it. The prior captures beliefs about the behavior of the function. After gathering the function evaluations, which are treated as data, the prior is updated to form the posterior distribution over the objective function. The posterior distribution, in turn, is used to construct an acquisition function (often also referred to as infill sampling criteria) that determines the next query point.

In our case the Blackbox function is the neural network model that takes in the test set images of cats and dogs as input and returns the accuracy as the output. The code snippet XXXX and results below show how Bayesian optimization helped us tune our model hyper parameters to maximize test set accuracy.

LDA and Support Vector Machines:

Note: For LDA and SVM, the matrix composed of flatten images were augmented before model implementation. A convolution or a series of convolutions was applied to the matrix. Then, the PCA scores were calculated to reduce dimensionality of the dataset before training. In particularly, the vertical directional filter was applied first along with the sobel filter. The selection for this sequence came from an empirical experiment of different convolutions using k-fold cross validation for each model.

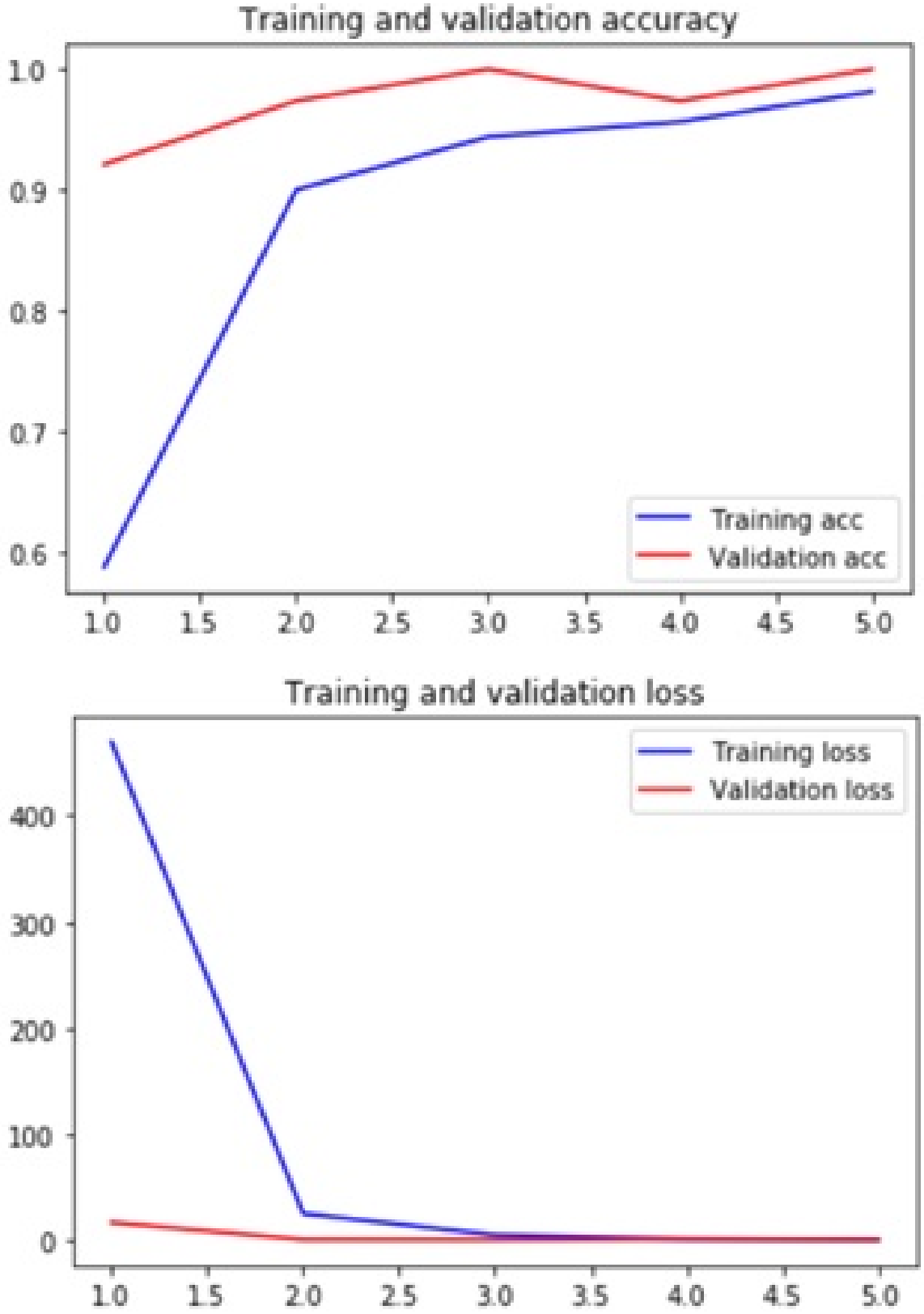

Figure 5: Training and validation accuracy/loss by training epoch for CNN

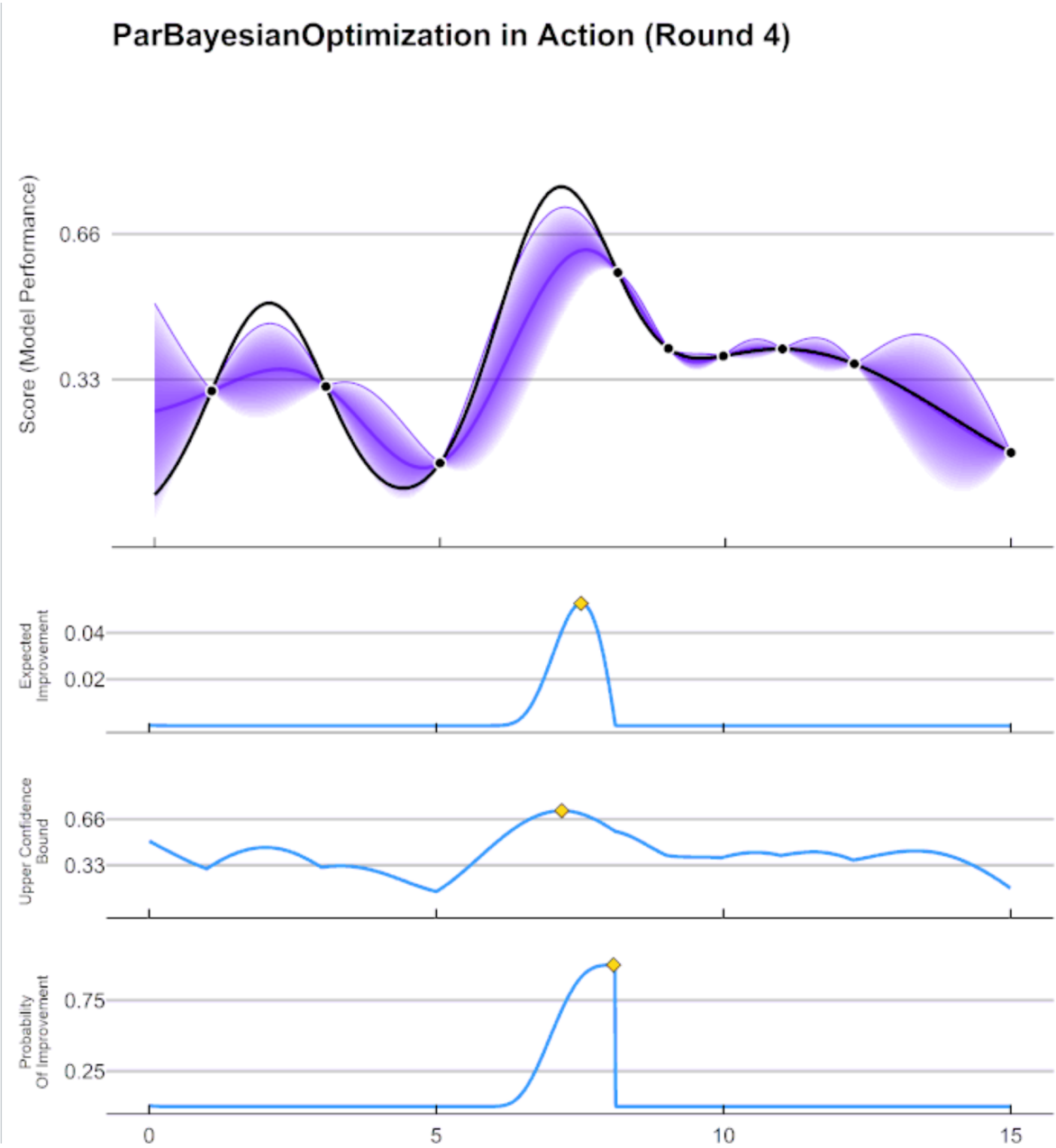

Figure 6: An example of *Bayesian Optimization* in action.

```
%time catdogBO.maximize(init_points=4,n_iter=16)

|   iter    |  target   | batch_... |  epochs   |   l1_d    | l1_units  |    lr     |
-------------------------------------------------------------------------------------
WARNING:tensorflow:Large dropout rate: 0.875682 (>0.5). In TensorFlow 2.x, dropout() c
38/38 [==============================] - 0s 1ms/step
|  1        |  0.9211   |  82.06    |  306.7    |  0.8757   |  85.57    |  0.02186  |
WARNING:tensorflow:Large dropout rate: 0.615486 (>0.5). In TensorFlow 2.x, dropout() c
38/38 [==============================] - 0s 1ms/step
|  2        |  0.9474   |  83.45    |  243.1    |  0.6155   |  1.496e+0 |  0.05995  |
38/38 [==============================] - 0s 1ms/step
|  3        |  0.9211   |  74.92    |  37.86    |  0.07051  |  1.724e+0 |  0.08843  |
WARNING:tensorflow:Large dropout rate: 0.97484 (>0.5). In TensorFlow 2.x, dropout() us
38/38 [==============================] - 0s 1ms/step
|  4        |  0.9737   |  83.27    |  195.5    |  0.9748   |  1.013e+0 |  0.07518  |
38/38 [==============================] - 0s 1ms/step
|  5        |  0.9737   |  156.2    |  498.1    |  0.4057   |  801.8    |  0.09243  |
38/38 [==============================] - 0s 1ms/step
|  6        |  0.9737   |  68.24    |  191.6    |  0.01192  |  1.015e+0 |  0.02512  |
WARNING:tensorflow:Large dropout rate: 0.869618 (>0.5). In TensorFlow 2.x, dropout() c
38/38 [==============================] - 0s 1ms/step
|  7        |  0.9474   |  16.1     |  489.7    |  0.8696   |  811.8    |  0.09125  |
```

Figure 7: Hyperparameter Tuning in Action

Algorithm 3 (Classification by FDA)

Input:

Cat training data set T_C.
Dog training data set T_D.
A set of probe data X.

Output:

A prediction vector **Pred**.

1. Calculate S_W and S_B.
2. Perform eigen decomposition on $S_W^{-1}S_B$ and extract w.
3. Return the w along with the matrices of cat and dog extracted from step 2.
4. Calculate the threshold: $[\text{mean}(T_C) + \text{mean}(T_D)]/2$
5. If $\text{mean}(T_C) > \text{mean}(T_D)$, follow statement step 7; if not reverse the statement.
6. Calculate score of X
7. If the score > threshold
   set $\mathbf{Pred}^{(k)} = 1$.
   Else
   set $\mathbf{Pred}^{(k)} = 0$.

Algorithm 4 (Classification by LDA)

Input:
train data X_{train}.
train label y_{test}.
test data X.
Output:
A prediction vector **Pred**.

1. Instantiate the model LinearDiscriminantAnalysis from SkLearn using the least square methods('lsqr') solver and a shrinkage coefficient of 0.65.
2. Fit the model from the training data and label using the fit method.
3. Predict the test data X with the fitted model using the predict method.

Algorithm 5 (Classification by SVM)

Input:
train data X_{train}.
train label y_{train}.
test data X.
Output:
A prediction vector **Pred**.

1. Instantiate the model SVC from SkLearn using the gaussian kernel ('rbf'), default gamma (1/number of features), and cost(C) of 3.7.
2. Fit the model from the training data and label using the fit method.
3. Predict the test data with the fitted model using the predict method.

The hyperparameters from LDA and SVM were decided based on the 5-fold cross validation along with 5 shuffles on the training set before incorporating the entire training set. Due to time constraints, there were a limit amount of experimentation in the hyperparameters for SVM.

4 Analysis of Classification Error and Efficiency

By now it should be clear that the problem of distinguishing the cats from the dogs has many possible approaches. Simply knowing that we can classify novel data from a known training set is of little value, however. At the heart of our study lies two more important questions. Firstly, how accurately can a given algorithm classify the data? Secondly, how efficiently can an algorithm perform? We dedicate this section to answering these two questions.

THIS SECTION NEEDS MORE CONTENT!!!

5 Printout of Computer Codes

```
%MAT 521 Project folder

load PatternRecData;

Cat = KLDATA(:, find(sublabels==1));
Dog = KLDATA(:, find(sublabels==0));

%plot an example all KL coefficients of a cat and dog
figure(1)
idC = randi(size(Cat,2));
idD = randi(size(Dog,2));
```

6 References

References

[1] Bishop, Christopher M.. *Pattern Recognition and Machine Learning*. Springer New York, New York, NY, 2016.

[2] *Muneeb ul Hassan: VGG16 – Convolutional Network for Classification and Detection* `https://neurohive.io/en/popular-networks/vgg16/`

[3] Jen-Mei Chang. *Matrix Methods For Geometric Data Analysis and Pattern Recognition*(Accessed 05 March 2020.) `http://web.csulb.edu/ jchang9/files/MatrixMethodsMain.pdf`

[4] L. Sirovich and M. Kirby *Low-dimensional procedure for the characterization of human faces* J. Opt. Soc. Am. A, 1987.