Gaussian Processes Meet Neural Nets: Interpretable Image Classifier

Yididiya Nadew[1] Kunle Oguntoye[3] Mario Mastrandrea[2]
Satyaki Mukherjee [1]

[1]Department of Computer Science, Iowa State Univeristy

[2]Department of Electrical and Computer Engineering, Iowa State Univeristy

[3]Department of Civil and Construction Engineering, Iowa State Univeristy

yididiya, mariomas, oguntoye, satyakim@iastate.edu

December 12, 2022

Abstract

We consider the problem of interpretability in image classification. Deep Neural Networks(DNNs) are the most popular choice in image classification due to their high expressive power. However, DNNs do not provide uncertainty quantification and thus are not easily interpretable. This hinders their adoption in high- stake applications. In contrast, probabilistic methods like Gaussian Processes(GPs) yield interpretable models with uncertainty estimation for their predictions. Recently, a new line of work called Deep Kernel Gaussian Processes(DKGP) has emerged. DKGPs couple the benefits of the two worlds. In this project, we demonstrate the benefits of DKGP in real-world datasets.

1 Introduction

With the proliferation of smart devices, the internet of things, and sensors, we create many images in our daily lives. Given these images, a class of problems known as *image classification* deals with assigning them to different categories based on their features. It has diverse applications like breast cancer detection in medicine, identifying pedestrians and street signs for autonomous vehicles, autonomous inspection in manufacturing, face recognition in surveillance, object detection in aerial maps, and many more.

Deep Neural Networks(DNNs) have been the most popular for classifying images into different categories [1]. Specifically, a variant called Convolutional Neural Networks(CNN) is widely used. These models employ hierarchical convolution operations on input images to predict their categories. It allows them to learn complex spatial relationships within images. Another important property of CNN models is translation invariance, such as image rotations and lateral shifts, which complements the model's accuracy. However, analogous to other black box models, with DNNs, it is difficult to explain model predictions. In addition, unlike Bayesian methods, they only learn point estimates with no uncertainty estimation [2].

On the other hand, Gaussian processes(GPs) are flexible non-parametric, probabilistic models that require fewer design decisions[3]. Unlike neural networks, they do not require to specify model complexity like the number of layers, neurons in each layer, etc. The complexity of the resulting model will be automatically calibrated based on the complexity of our training data. Furthermore, we can train GPs based on marginal likelihood objectives due to their probabilistic nature. And thus, their predictions also include uncertainty estimation. Consequently, unlike their non-probabilistic counterparts, GPs are said that "they know what they do not know."

GPs learn complex patterns by using kernel functions defined over training samples [4]. Kernel functions correlate samples based on their distances in the input space. And GPs assume stationary structure in the data, i.e., any two pairs of samples are equally correlated given their pairwise distances are equal. However, stationarity does not generally hold. A simple example could be a step function with abrupt jumps between close samples.

Over the last decade, there has been a surge of research interest in combining neural networks and Gaussian Processes. The motivation behind such efforts is to benefit from both the high expressive power of neural nets and the non-parametric flexibility of GPs. Generally, there are two common approaches to combine the two: 1) two-stage approach and 2)end-to-end approach. The two-stage approach uses DNNs as a pretraining[5]. Then, using the network, we can project the

samples to low dimensional feature space and use these projections as input to a GPs classifier. The end-to-end training approach uses a marginal loglikelihood objective to train both the neural network's weight and the GP's parameters.

In this project, we conduct experiments with both approaches. In particular, we apply them to two datasets, MNIST[6], a popular toy classification dataset, and Brain Tumor Dataset[7]. The report is organized into different sections. **Background** section highlights the building blocks for Deep Kernel Gaussian Process. The section **Method** presents details about the inference technique used for the model. Details of the datasets are presented in the **Dataset**. Following that, we present toy and real-world experiement and discuss the resuls under **Experiments** and **Results** section.

2 Background

In this section, we provide a brief review of Gaussian Processes and Convolutional Neural Networks and introduce Deep Kernel Gaussian Processes.

2.1 Gaussian Processes

Gaussian process(GP) is a collection of random variables, any finite subset of which has a joint Gaussian distribution[8]. A Gaussian Process is completely specified by its mean function and covariance function. Let $f(x)$ a Gaussian Process to app to $k(\mathbf{x}, \mathbf{x'})$ of a real process $f(x)$ where **x** and **x'** are input variables.

$$\begin{aligned} m(x) &= E[f(x)] \\ cov(f(x), f(x')) &= k(x, x') = E[(f(x) - m(x))(f(x') - m(x')] \end{aligned} \tag{1}$$

and the Gaussian Process $f(\mathbf{x})$ is given by

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x},\ \mathbf{x'}))) \tag{2}$$

The above model is formulated for real-valued regression responses. In classification, we need to map $f(\mathbf{x})$ to a probability value using squashing functions like logistic function.

In GP models, the mean functions $m(x)$ are often set as a constant or zero. Thus the covariance, also called the kernel function, determines the property of the Gaussian Processes. There are different choices of kernels available. However, the most common one is the Radial Basis Function Kernel(RBF), defined as follows.

$$cov(f(\mathbf{x}), f(\mathbf{x'})) = k(\mathbf{x}, \mathbf{x'}) = \exp\left(-\frac{||\ \mathbf{x} - \mathbf{x'}||^2}{2l^2}\right) \tag{3}$$

Here, the kernel correlates two datapoints **x** and **x'** based on their squared Euclidean distance. The characteristics scale l is a free parameter that regularizes the distance. As the distance increases, for a constant l, $k(\mathbf{x}, \mathbf{x'})$ shrinks to zero. In other words, a pair of data points closer to each other in the input space will have highly similar function values. From an interpretability perspective, this is an important property as we can explain the predictions on test datapoint by looking at their similarity with training samples.

Despite the model's simplicity, the RBF kernel fails to capture non-Euclidean patterns in the input space. For high-dimensional data exhibiting non-Euclidean patterns like images, RBF is too simple to capture their complexity.

Furthermore, learning the hyperparameters of a GP model with either a Gaussian likelihood (regression problems) or non-Gaussian likelihood(for classification problem) requires applying a series of matrix operations on an n x n covariance matrix which incurs $\mathbf{O(n^3)}$ computations and $\mathbf{O(n^2)}$. For large datasets, with over thousands of data points [9], this presents a computational challenge. [9] propose to work with a smaller number of inducing points that approximate the original data points. Under **Method** section, we discuss the formulation of the method.

2.2 Convolutions Neural Networks

Convolutional neural network (CNN) is a neural network variant, specifically, the deep learning network, that typically identifies features and patterns that are intrinsically present in pixel data [10]. CNN, in its architecture and computational graph, shares close similarities with the neuron connections in the human's brain frontal lobe. Intuitively, CNN mirrors the part of the brain responsible for processing visual stimuli. Thus, the emergence of CNN has helped reimagine how computer vision and speech recognition related-problems are tackled. The architecture's parameter-sharing ability qualifies its outstanding performance in speed and training time metrics compared to older neural networks for similar tasks. To explain CNN's success with vision problems, it is pertinent to dig into the components of the architecture.

Typically, the CNN model input data is characterized by n x n x c dimensional size, with n being the width and height and c – the number of channels. The discretization of image data into a matrix of numbers ranging from 0 to 255 makes it easy to extract features using filters (kernels) that typically have a dimensional size of m x m x d. These filters convolve around the set of input layer matrices to output a feature map. The filters are analogous to the coefficients and biases used in classical machine learning regression problems. After completing the convolution matrix multiplications, a non-linear and differentiable activation layer such as sigmoid, Tanh, Relu, and others help map the input convolution layer into a deeper layer, enabling the model to learn even more complex features.

Furthermore, it is pertinent to reduce the input size to achieve the desired output size. The pooling layer with a specified window size of k x k convolves around convolution layers and selects either the maximum or average values appearing in each window stride. CNN models typically comprise a repetitive group of convolution, activation, and pooling layers. The complexities to be learned in any given input data dictate the choice of CNN textures from input data. The later convolution layers extract and learn more complex features, such as objects or parts of the objects. As the above figure shows, the fully connected layer receives vectorized data from the last feature map. It follows the principle of the conventional multi-layer perceptron neural network to feed forward the last feature map to the final output neuron(s). The residual error between the predicted outcome and actual label sets is then optimized using the backpropagation technique to achieve learning weights and biases with improved performance on test sets.

A known fact is that deep networks have better performance, but an underlying problem with this statement is quite apparent. Deeper networks are susceptible to gradients vanishing [11], and not to forget, they are computationally expensive to train. As a result, different models have evolved other the past decade to tackle problems such as scalability, gradient zeroing, overfitting,

and a host of others. While there are networks that precede AlexNet[12], every story of CNN networks often starts with AlexNet because of its incredible performance at that time on ImageNet. With a depth of eight feature extraction stages (five convolution layers and three fully connected layers) and filter sizes of 5, 11, and 3, the model addressed the problem of overfitting with local response normalization and overlapping subsampling. Since the emergence of AlexNet, several other models have evolved to tackle different problems, primarily by reconfiguring CNN deep architecture. Models such as visual geometry group (VGG), GoogLeNet, residual net, mobile net, and a host of others are widely used today in solving computer vision-related problems. Also, thanks to the concept of transfer learning, already trained models on sizeable training data can serve as a backbone in extracting features from a custom dataset.

Despite its proven performance in diverse image datasets, CNNs, like other parametric and non-probabilistic models, still suffers from two major issues. First, the model is not easily interpretable. Due to its black-box nature, we can not explain the model predictions necessary for high-stake applications. Furthermore, due to its non-probabilistic aspect, it does not provide us with confidence estimates on its predictions. In the next subsection, we highlight the recent efforts in combining Gaussian Processes and CNNs to create the so-called Deep Kernel Gaussian Processes.

2.3 Deep Kernel Gaussian Processes

Unlike base kernels presented in (2.1), deep kernels do not use input x directly. Instead, they work with its non-linear transformation based on neural nets. This transformation is parametrized using weights w. Hence, the GP model with such kernels is called Deep Kernel Gaussian Processes(DKGP). The covariance function of a deep kernel is defined as follows.

$$k(\mathbf{x}, \mathbf{x'}) = k(g(\mathbf{x}, w), g(\mathbf{x'}, w)|w) \tag{4}$$

where $g(., w)$ is a deep architecture with weights w

In a DKGP model, the non-linear transformation captures the non-stationary and hierarchical structure of our data, whereas the kernel, like RBF kernel, learns the stationary aspect of the data and hence results in an easily interpretable model [13].

After training the DKGP, we can examine the covariance matrix induced by our data points. The covariance function summarizes similarity among data points and thus is instrumental in explaining predictions of the model by providing a simple explanation of why a given test sample is classified under a certain category. This can be done by filtering highly correlated data points in our training set for the sample. In the **Results** section, we present examples of such explanations on both correctly classified and misclassified samples.

In addition, unlike plain neural nets, DKGP provides us with uncertainty estimates of its predictions. This provides domain experts with additional information resulting in a more reliable and explainable model.

3 Method

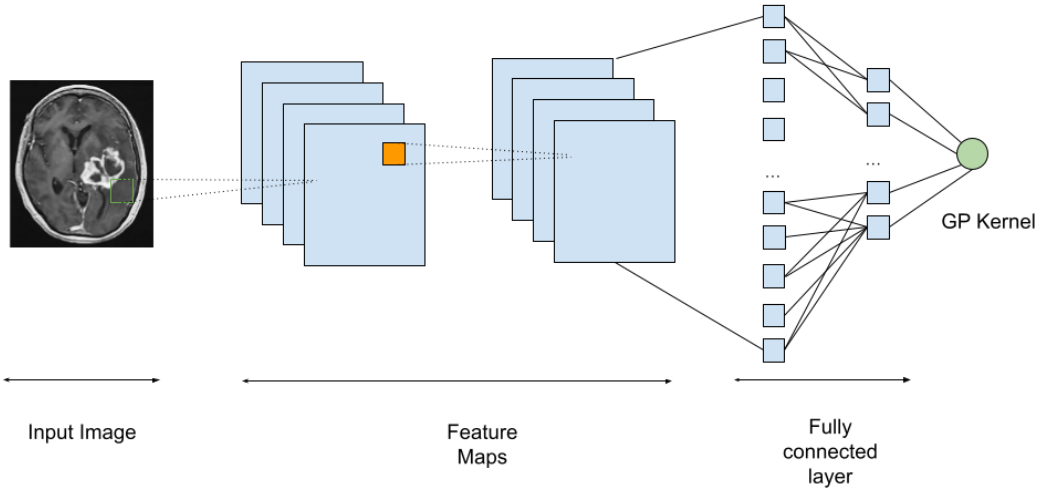


Figure 1: DKGP Architecture

In this section, we define and formalize the DKGP model for the classification model.

Let C be the number of classes in a classification problem with d-dimensional input. And let $\mathcal{D} = \{x_i, y_i\}_{i=1}^n$ be our dataset where $x_i \in \mathbb{R}^d$ and $y_i \in \{0, 1\}^C$ are input vector and class label respectively for the ith sample.

Our likelihood is given by,

$$p(y_i|f_i) = \frac{\exp(f_i y_i)}{\sum_c \exp(f_i I_c)} \tag{5}$$

where $f_i = f(x_i)$ is Gaussian Processes as defined in (2.1) and I_c is a one-hot vector with 1 on the cth index. Here, we choose the common RBF kernel over the outputs of a deep neural network(DNN) as described in equation 4. Thus, learning a DKGP model requires inference of two sets of parameters: θ is the hyperparameter of the GP, such as its length scale, and w weights of the DNN w.

In our experiments, we consider a multi-class classification problem. To implement DKGP, we follow the *two-stage approach* [9]. First, we construct a CNN model with the architecture shown in Figure 1. In the figure, all blue componenet are the layers of the CNN model. The final layer has a softmax function to choose the most probable class. To train the model, We use a categorical cross-entropy loss objective. Following the training, we save the learned parameters of the best model selected based on the accuracy of the validation set. In the second stage, we replace the classification layer with a GP kernel. Then, we partially freeze network weights in the CNN layers of the network. However, parameters in the fully connected layers(FC) and hyperparameters of our GP kernel are allowed to be updated in training. The benefits of this are two folds. One, it allows for faster GP training since we are only retraining the final layers of the network. It also

gives the model flexibility to fine-tune the parameters for another objective function: the marginal likelihood of the GP.

3.1 Inference

In this project, we consider GP model specifically for a classification problem whose likelihood is non-Gaussian, thus analytically intractable. In addition, as noted in (2.1), classical GP inference methods are notoriously slow for large-scale datasets. Therefore, we use common approximation techniques called *stochastic variational inference*(SVI) to train DKGP.

SVI benefits from a common optimization technique called stochastic gradient descent where noisy gradients of our objective function are calculated using only a fraction of our data. In expectation, these noisy gradients are proven to approximate natural gradients, gradients calculated using all the training samples.

4 Dataset

4.1 MNIST

MNIST [6] is a large dataset of handwritten digit images and their corresponding labels. It comprises 60,000 training samples and 10,000 test samples, each categorized as one of $C = 10$ classes. Each sample is a 28x28 pixel image with only one channel. We follow a common standardization technique to center the pixel values around the mean. Each pixel value is divided by 255. Then, we subtract the mean and divide it by the standard deviation calculated across all images in the training set.

4.2 Brain Tumor Dataset

Brain Tumor Dataset[7] is a relatively small-sized dataset that contains MRI images of human subjects with labels on the presence or absence of brain tumors. The labels are $glioma\ tumor$, $meningioma\ tumor$, $pituitary\ tumor$, $notumor$, the last one being the negative class. The dataset contains $N_{train} = 2870$ training samples and $N_{test} = 394$ test samples. We first resize each image into a 150x150 pixel image. We then apply a similar standardization technique as done for the MNIST dataset. Figure 2 shows 4 samples from each of the four classes.

5 Experiments

For the MNIST experiment, we use Pytorch to implement a simple CNN architecture with 2 convolutional layers and 2 fully connected layers. To train the model, we used a training batch of size 64 images. We ran the training for 15 epochs with a decaying learning rate of 1.0 scheduled using the Adadelta method [14]. On the other hand, to train the corresponding DKGP model, we use Pyro, a deep probabilistic learning library [15]. For this experiment, we started training our DKGP model with random initialization. In both cases, we saved the best model based on its performance on the test set.

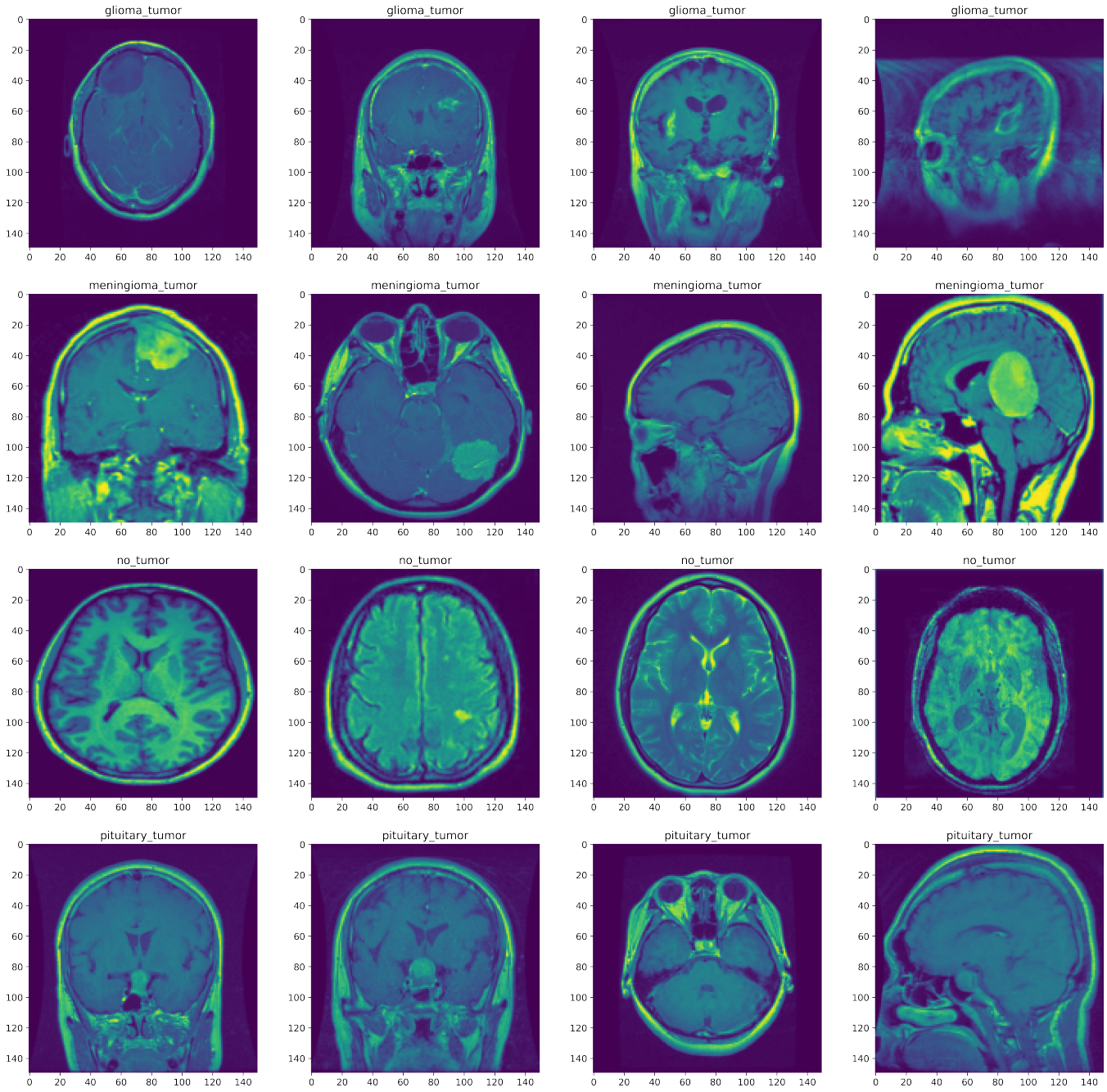


Figure 2: Brain Tumor Dataset, 4 training images for each category

Method	Accuracy
MNIST(CNN)	99.15
MNIST(DKGP)	**98.66**

Table 1: Test Accuracy on MNIST dataset

For the DKGP model, we used 70 randomly selected samples from the training set as inducing points. And we also varied the number of inducing points from 50 to 120 on equally spaced intervals and calculated test set accuracy for each.

For the Brain Tumor Dataset, we constructed a deep architecture with 3 convolutional layers coupled with pooling layers and 3 fully connected layers(FC). Before the FC layers, we applied drop-out as a regularizer. And we used the softmax loss function to optimize the model using a stochastic gradient descent algorithm. We used Adam optimizer[16] with a learning range of 0.01. We ran the training for 10 epochs, with a batch size of 64 samples, and selected the best models based on accuracy on the validation set.

Following the DNN training, we trained the DKGP model with an RBF kernel of input size equal to the C=4 classes. We initialized all lengthscales of the kernel to be 1. We experimented with three approaches of combining the CNN model with DKGP. First, we used the *random initialization* where we allow all model parameters, GP hyperparameters, and neural network weights to be learned using marginal likelihood objective. Furthermore, we used trained *CNN model as static feature map* where all learned DNN parameters are frozen. As a third variant, we used *partially frozen* DNN layers. Here we froze the all layers except for the last fully connected layers. We randomly initialized 64 inducing points to approximate our dataset. Then we used variational stochastic gradient algorithm [15] to learn its parameters.

All experiments were done on a desktop machine with Nvidia GTX 1080 GPU and 11GB dedicated memory.

6 Results

6.1 Accuracy

In the MNIST experiment, our trained CNN model has an accuracy of 99.15% on the test set. Whereas DKGP achieves 98.66% accuracy. This shows DKGP does not result in an appreciable loss of accuracy on the MNIST dataset. In addition, test results for models with a varying number of inducing points show that increasing inducing points does not significantly increase the accuracy.

On the Brain Tumor Data, we fit both CNN and DKGP models. Our CNN model achieves 73.09% accuracy, whereas the DKGP model achieves around 71.83%. For DKGP, we fit three variants as discussed under **Experiments**. We found that using random initialization, the model was not able to converge. This could be due to improper hyperparameters. However, we compared the results for (1) using freezing all DNN layers and (2) allowing the final layers of the DNN to be retrained. Retraining only the last fully connected layer improves the accuracy by 0.2% from (1). However, adding more trainable layers does not seem to improve the accuracy. Figure 3 summarizes the accuracy results.

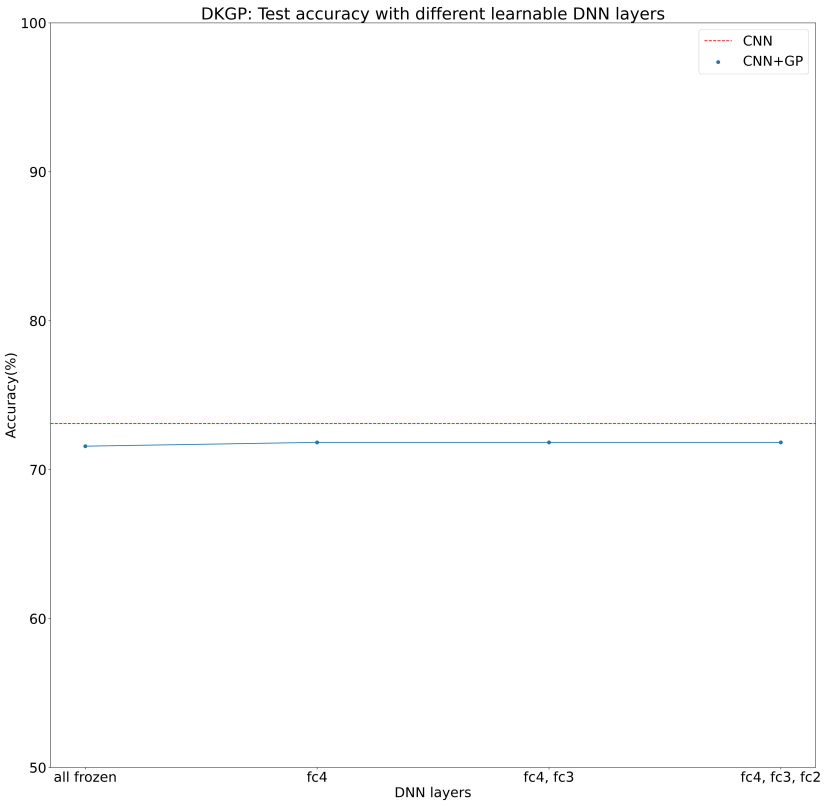
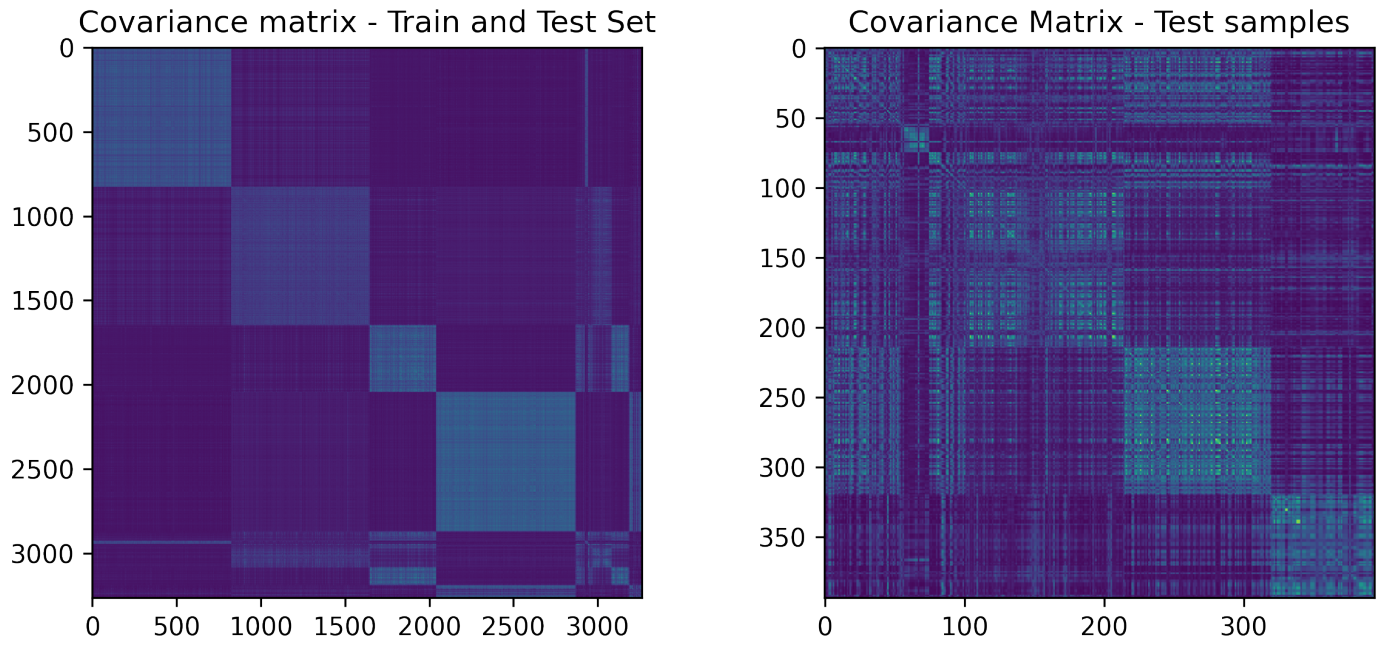


Figure 3: Accuracy: with frozen DNN layers



(a) Covariance matrix on train and test sets

(b) Covariance matrix on test set

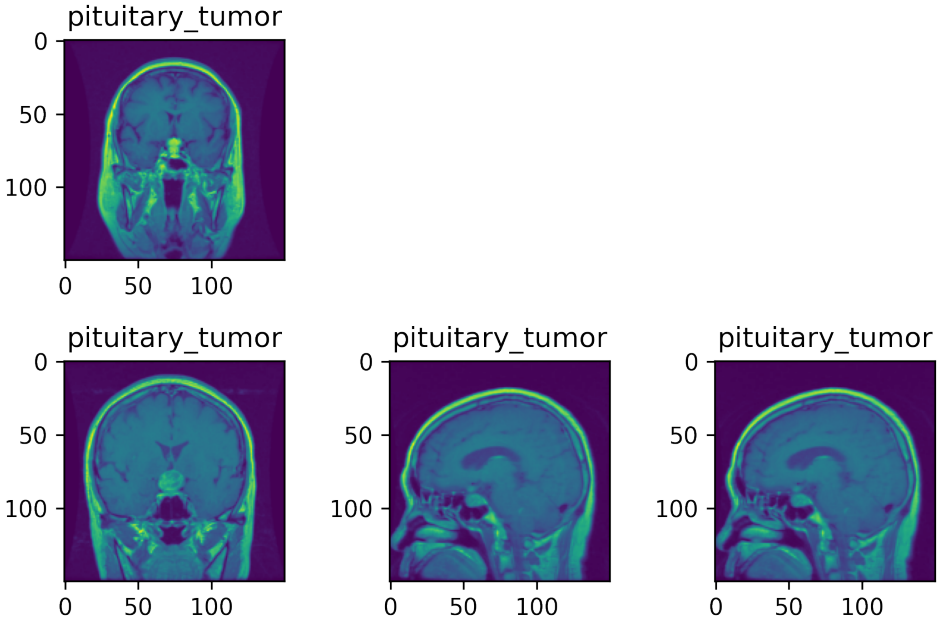
Figure 4: Brain Tumor, DKGP model covariance matrices

6.2 Datapoint similarity

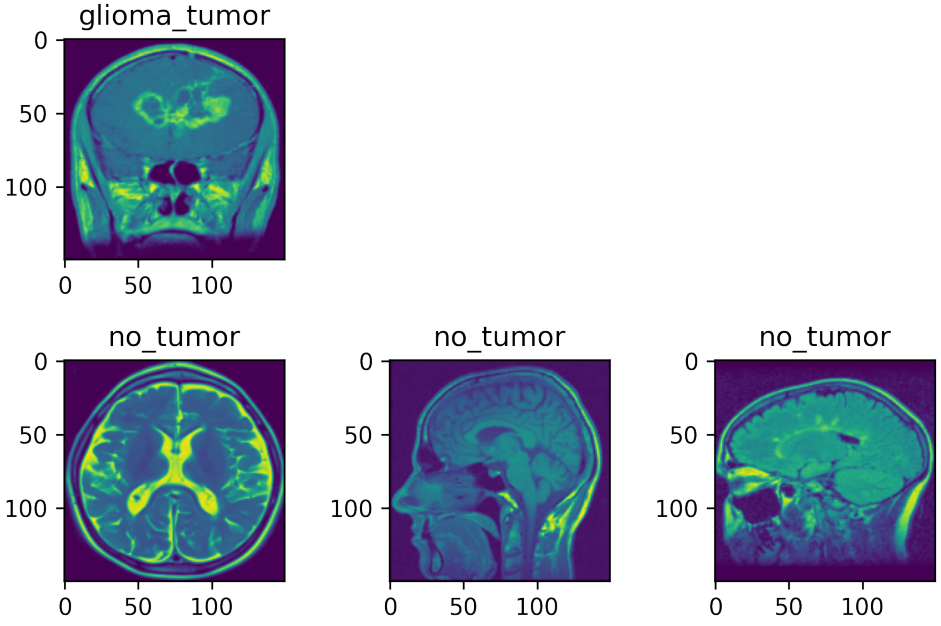
In Gaussian Processes, covariance functions define pairwise correlations of data points. If x_i and $x_j \in X$ are highly correlated, then their functional values y_i and y_j will be similar. In a classification problem like ours, it implies they likely belong to the same class. Therefore, for a given model predictions $(x_i, \hat{y_i})$, it is a common practice to point out the most correlated data points in the training dataset. This can be done by estimating the covariance matrix of both train and test samples together, and looking at individual test samples and their most correlated training samples.

Figure 4 (a) and (b) are heatmaps of the covariance matrices on the Brain Tumor dataset. The rows and columns denote the sample indices, in total, we have $N = N_{train} + N_{test} = 3264$ samples. Out of these, the first 2870 samples are training samples, and the last 394 samples are test samples. In these figures, the light color indicates high correlation or similarity, whereas dark indicates low correlation or dissimilarity. It can be seen in Figure 4 (a), the first submatrix with lighter color indicates the similarity between samples from the first class, $glimona\ tumor$ class. Figure 4 (b) zooms in the bottom right corner of (a) to show the covariance between only test samples.

Using such matrices, we can explain the results of our model. Given a test sample, we can rank similar samples based on pairwise correlations. This provides an important tool to explain model results which are previously unexplainable. Unlike the standard confusion matrix, such a technique enables us to diagnose our model. In our experiments, we looked at both correctly classified and misclassified test samples and samples in highly correlated samples. Figure 5 shows such examples from the Brain Tumor dataset.



(a) Correctly classified sample(sample on the first row) and its most similar training samples



(b) Misclassified sample (sample on the first row) and its most similar training samples

Figure 5: Brain Tumor; datapoint similarities

6.3 Uncertainity Estimate

In addition to the above explanations, DKGP model provides uncertainity estimates for its prediction. The results of the model are a multivariate Gaussian distribution with a mean value, one for each of its classes, and a variance associated with these classes. Thus, one can simply sample this distribution and recover a confidence region out of it. Due to the high-dimensional nature of the problem, we have not included the results in the paper. However, the experiments can be found in the included code directory.

7 Conclusions

In this project, we have explored the idea of mixing a highly expressive parameteric model with a non-parameteric and probabilistic model to reap the benefits of both. In particular, we coupled Gaussian Processes with a Deep Neural Network architecture to solve an image classification problem. This enabled us to use out-of-the-box tools found in GP like uncertainty estimation, and model explanation in a classification problem. We evaluated the model on real-world examples where model explanation is as equally necessary as model performance. In the future, we plan to explore the use of such techniques for more complex models.

References

[1] Arkadiusz Kwasigroch, Agnieszka Mikołajczyk, and Michał Grochowski. Deep neural networks approach to skin lesions classification — a comparative analysis. In *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*, pages 1069–1074, 2017.

[2] Jeremiah Liu, Zi Lin, Shreyas Padhy, Dustin Tran, Tania Bedrax Weiss, and Balaji Lakshminarayanan. Simple and principled uncertainty estimation with deterministic deep learning via distance awareness. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 7498–7512. Curran Associates, Inc., 2020.

[3] Kalvik Jakkala. Deep gaussian processes: A survey. *arXiv preprint arXiv:2106.12135*, 2021.

[4] Andrew Wilson and Ryan Adams. Gaussian process kernels for pattern discovery and extrapolation. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1067–1075, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[5] Andrew Wilson and Hannes Nickisch. Kernel interpolation for scalable structured gaussian processes (kiss-gp). In *International conference on machine learning*, pages 1775–1784. PMLR, 2015.

[6] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[7] Jun Cheng. brain tumor dataset. 4 2017.

[8] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

[9] Andrew G Wilson, Zhiting Hu, Russ R Salakhutdinov, and Eric P Xing. Stochastic variational deep kernel learning. *Advances in Neural Information Processing Systems*, 29, 2016.

[10] Matthew Browne and Saeed Shiry Ghidary. Convolutional neural networks for image processing: An application in robot vision. In Tamás (Tom) Domonkos Gedeon and Lance Chun Che Fung, editors, *AI 2003: Advances in Artificial Intelligence*, pages 641–652, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[11] Xiaojiao Mao, Chunhua Shen, and Yu-Bin Yang. Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[13] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial intelligence and statistics*, pages 370–378. PMLR, 2016.

[14] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[15] Eli Bingham, Jonathan P Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D Goodman. Pyro: Deep universal probabilistic programming. *The Journal of Machine Learning Research*, 20(1):973–978, 2019.

[16] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.