# Gaussian Process Framework for Deep Neural Networks

**Xiang Fu** [* 1] **Shengyuan Hu** [* 1] **Shangdi Yu** [* 1]

## Abstract

Recent work (Garriga-Alonso et al., 2018) has shown deep convolutional neural networks (DNNs) can be approximated by a shallow Gaussian processes (GP) with much fewer parameters. A lot of features in modern convolutional neural network (CNN) have not been considered in this work. In this paper, to extend the flexibility of the transformation from modern neural network architecture to shallow Gaussian process, a framework for DNNs is introduced. An average pooling operation and a concatenation operation are derived to support densely-connection structures to fit in Garriga-Alonso et al.'s architecture by transforming them into simple matrix multiplication. The newly derived DenseNet-GP significantly reduce the time for calculating the kernel matrix for the GP while having comparable accuracy in classifying images in CIFAR10.

## 1. Introduction

Recent development in deep learning has brought about significant improvements in various machine learning tasks. In particular, neural network and deep learning works well in high dimensional data like images and has improved accuracy on tasks like image classification. However, in high dimensional data, the model is very easy to overfit to the training data and can have high curvature locally. Another model good at learning data distribution is Gaussian process (GP). Recent advance in the scalability of Gaussian process (Wilson & Nickisch, 2015) has enabled Gaussian process to work on large scale machine learning task. While Gaussian process is more likely to learn a smooth manifold representation of the data, it is, however, is known to be bad at learning high dimensional data like image.

In order to solve the problems in both models, the idea of using Bayesian learning is proposed to combined with artificial neural network. Neal 1996 provided an extensive study on such method and showed the Gaussian process behavior of single layer neural network. Later works by Matthews et al. 2018 extended this finding to the GP behavior in wide deep neural network. These works provide an alternative way to traditional neural network on image classification. Instead of training the neural network using back propagation, we inference using the kernel with the parameters of the network that fit the training data best. We will discuss in the paper why this is appropriate and the benefits of kernel inferences.

In a recent work by Garriga-Alonso et al. 2018, a shallow Gaussian process representation of deep convolutional neural networks is proposed and proven valid by deriving the kernel of the GP using the explicit mathematical formula of the neural network with convolution and ReLU activation, significantly reducing the parameters of the model compared with the original deep neural network. However, in practice, modern neural networks in computer vision have features such as pooling, concatenation of feature-maps, batch normalization, which have not had a counterpart in the GP kernel that represents the neural network.

This work will focus on developing the transformation from neural network techniques, specifically the average pooling and the concatenation, to their counterparts in a shallow GP kernel that represents the original neural network. We will show in section 3.2 that adding an average pooling is actually equivalent of adding one more modified convolutional layer. The transformation of concatenation is more involved and the details will be shown in section 3.3. We implemented the two transformations in TensorFlow, the framework that Garriga-Alonso et al. used. We have also implemented the mirror version of this transformation from Deep Neural Networks to Gaussian processes in PyTorch using the GPyTorch framework (Gardner et al., 2018). [1] Specifically, we have rewritten the two main kernels - the *DeepKernel* and the *ResNetKernel*. While rewriting the two kernels, we have also implemented the elementwise ReLU kernel and the elementwise Erf kernel. Our work produces a flexible transformation from modern convolutional neural networks with millions of parameters to a Gaussian process with much fewer parameters. In section 4 we conduct experiments on the MNIST and CIFAR10 to compare the accuracy

[*]Equal contribution [1]Cornell University. Correspondence to: Xiang Fu <xf74@cornell.edu>, Shengyuan Hu <sh797@cornell.edu>, Shangdi Yu <sy543@cornell.edu>.

[1]https://github.com/yushangdi/GPDNN

of our model with the average pooling operation and the baseline model of Garriga-Alonso et al. to illustrate how adding new features can influence the model performance on both the accuracy and running time.

## 2. Background & Related Work

Our work combines two ideas that are drawing significant attention in the machine learning community in the recent decades, namely Gaussian processes and neural networks. In this section, we will introduce the background of random neural networks, Gaussian processes, and the literature that study the relationship between the two. There are rich literature on both Gaussian processes (Rasmussen, 2006) and DNN structures (Schmidhuber, 2014) as the two approaches that both are very popular and shown to have competitive performance in data inference in the recent years. The study of the relationship between Gaussian processes and neural networks has also dated back to two decades ago.

### 2.1. Convolutional Neural Networks

CNN is a variant of feed-forward artificial neural networks. Let M be the number of basis functions and D be the number of variables. The general form of neural network can be written as

$$y(x, w) = f(\sum_{j=1}^{M} w_j \phi_j(x))$$

where $\phi_j(x)$ are the basis functions. Activation functions $h$ are chosen to transform activations at layer $l$ $a_j^{(l)} = \sum_{i=1}^{D} w_{ji}^{(l)} x_i + w_{j0}^{(l)}$ to hidden units $z_j = h(a_j)$. The output unit activations $a_k^{(l')} = \sum_{j=1}^{M} w_{kj}^{(l')} z_i + w_{k0}^{(l')}$. Output unit activation can then be transformed to $y_k = \sigma(a_k)$. Some commonly used building blocks of CNN include convolutional layers, pooling layers, and ReLU layers. (Bishop, 2006) Modern CNN architectures in image classification like DenseNet (Huang et al., 2016) and Inception (Szegedy et al., 2014) have achieved outstanding performance. However, these architectures suffer from 1) millions of trainable parameters and 2) long time to train the model, even when doing transfer learning with the pretrained version of these models. Hence, it is of our great interests to find few-parameter supplant to these neural networks while preserving their flexibility and complexity.

### 2.2. Gaussian Process

On the other hand, Gaussian processes approach requires only a few parameters and a relatively simple learning procedure with just matrix manipulations. In regression, we have the predictive mean on testing point $x_*$ is

$$\bar{f}_* = k_*^T (K + \sigma_n^2 I)^{-1} y$$

, and variance

$$V[f_*] = k(x_*, x_*) - k_*^T (K + \sigma_n^2 I)^{-1} k_*$$

where $K = K(X, X)$, $k(x_*) = k_*$. In multi-class classification, we can use the softmax function

$$p(y = C_e | x, W) = \frac{exp(x^T w_e)}{\sum_{e'} exp(x^T w_{e'})}$$

. This gives the probability of label $C_e$ of data point $x$ given weight matrix $W$. (Rasmussen, 2006)

### 2.3. Random Neural Networks as Gaussian processes

Researchers have have tried to uncover the relationship between the two approaches and develop algorithms that combine the Gaussian processes deep networks. For example, Wilson, Hu, Salakhutdinov, and Xing (2016) created deep kernels for Gaussian processes which combine deep networks, spectral mixture covariance functions (Wilson & Prescott Adams, 2013), and the above scalable GP methods. (Wilson et al., 2016)

#### 2.3.1. Single Hidden Layer Multilayer Perceptron

It has been known since long time that in the limit of an infinite number of hidden units, the prior over functions produced by a single layer neural network tends to a Gaussian process. (Williams, 1996; Neal, 1996)

To illustrate the idea of random neural networks, we use a *multilayer perceptron* network with a single hidden layer and infinite hidden units as an example. It has activation

$$h_j(x) = \tanh(a_j + \sum_{i=1}^{I}) u_{ij} x_i$$

and output

$$f_k(x) = b_k + \sum_{j=1}^{H} v_{jk} h_j(x).$$

Here, $u_{ij}$ are the input-to-hidden weights, $v_{ij}$ are the hidden-to-output weights. $a_j$ are the biases of the hidden units and $b_k$ are the biases of output units. $I$ is the number of inputs and $H$ is the number of layers. Giving $b_k, v_{ij}, a_j, u_{ij}$ zero mean Gaussian prior with variances $\sigma_b, \sigma_v, \sigma_a, \sigma_u$ respectively and drawing the values randomly from their priors, the resulting network is a random neural network. Neal (1996) showed that a Gaussian prior for hidden-to-output weights results in a Gaussian process (GP) prior for output functions. This fact also holds for any bounded activation function, i.i.d input-to-hidden weights and hidden unit biases, and zero mean finite variance hidden-to-output weights priors. (Neal, 1996)

The prior distributions represent our prior belief of the model. As we train the model with data, we update our prior belief to obtain the posterior distribution and then use the posterior for inference. When the number of hidden units is large, the networks may overfit on small training sets and perform poorly on testing sets. This problem can be alleviated by using Bayesian learning because the Bayesian approach allows us to spread our belief across a wide support. Using this Bayesian perspective, the predictive distribution is given by (Neal, 1996) as follows:

$$P(y^{(n+1)}|x^{(x+1)}, X, \vec{y}) =$$
$$\int P(y^{(n+1)}|x^{(x+1)}, \theta)P(\theta|X, \vec{y})d\theta$$

and the mean of the predictive distribution is

$$\hat{y}_k^{(n+1)} = \int f_k(x^{(n+1)}, \theta)P(\theta|X, \vec{y})d\theta.$$

#### 2.3.2. Deep Neural Networks

Though people have shown in theory a single hidden layer network with many hidden units can approximate any function defined on a compact domain arbitrarily closely, (Cybenko, 1989; Funahashi, 1989; Hornik et al., 1989) a more complex architecture can be advantageous in that more complex architectures can encode our inductive biases better into the networks and thus leads to better regression or classification results.

However, the relationship between Gaussian processes and neural networks with more than a single layer is not as well understood and is an active area of research. Recently researchers have proved that conditioned on an infinite number of channels at each layer, deep and fully connected neural networks do have a similar behavior as the single layer neural networks in that they also tend to Gaussian processes. (Matthews et al., 2018) Matthews et al. (2018) further studied the relationship between Gaussian processes with a recursive kernel definition and random wide fully connected feed-forward networks with *more than one* hidden layer. They showed that as the network becomes increasingly wide the distribution of the marginal distributions of the activations at each layer and of the output will become close to a Gaussian process. (Matthews et al., 2018) They proof this result by upper bounding how far each layer is from a multivariate normal distribution using Berry-Esseen inequality and then inductively propagating these inequalities through the network. (Matthews et al., 2018)

While the works of (Matthews et al., 2018) on deep neural networks significantly extended the work of Neal (1996), these fully-connected networks are rarely used in image classification tasks. In the next section, we will discuss the equivalent GP representations state-of-the-art architectures such as CNNs and ResNets (Garriga-Alonso et al., 2018) and our extensions.

## 3. The GP Framework

### 3.1. Model Setup

To convert from a convolutional neural network to a Gaussian process, we use the formulation and methodology introduced in (Garriga-Alonso et al., 2018). Input $X$ is an image of size $C^{(0)} \times (H^{(0)}D^{(0)})$. We denote each channel as $\mathbf{x_1}, ..., \mathbf{x_{C^{(0)}}}$, flatten to form a vector. We use a linear transformation as the first activation. So for $j \in \{1, ..., C^{(1)}\}$, we have the first activiation:

$$a_j^{(1)}(X) = b_j^{(0)} + \sum_{i=1}^{C^{(0)}} W_{j,i}^{(0)} x_i$$

Define the activation function to be $\phi$, activation vector after $\ell$-th layer on $j$-th channel is $\phi(a_j^{(\ell)}(X))$. The iterative updates for $a_j^{(\ell)}(X)$ is the following:

$$a_j^{(\ell+1)}(X) = b_j^{(\ell)} + \sum_{i=1}^{C^{(\ell)}} W_{j,i}^{(\ell)} \phi(a_i^{(\ell)}(X))$$

We denote the $\ell^{th}$ activation as $A^{\ell}$. Consider the neural network has $L$ hidden layers, the output of the neural network are the last activations $A^{(L+1)}(X)$. And for a regression or classification task, we must have $H^{(L+1)} = D^{(L+1)} = 1$.

In the formulas above, we express the convolution operation as matrix multiplication. We call the $W$ matrix pseudo-weights. It is shown in (Garriga-Alonso et al., 2018) how to get $W$ by transforming the original convolutional filter $U$. This transformation does not involve non-linear operations.

The key assumptions needed to give a GP representation of the above CNN are:

1. There are infinite number of filters (channels) in each convolutional layers.
2. Each entry in the filters $U_{j,i}^{(\ell)}$, and bias at each layer $b_j^{(\ell)}$ are independent Gaussian random variables. For each layer $\ell$, channels $j, i$ and locations within the filter $x, y$:

$$U_{j,i,x,y}^{(\ell)} \sim \mathcal{N}(0, \sigma_w^2/C^{(\ell)})$$
$$b_j^{(\ell)} \sim \mathcal{N}(0, \sigma_b^2)$$

When both assumption holds, it is shown in (Garriga-Alonso et al., 2018) that the behavior of such a CNN converges to a GP. Now in terms of calculating the kernel matrix, The

assumptions of infinite channels and all convolutional layers sharing the same prior makes the computation for the kernel matrix very complicated. However, for a classification or regression task with a final dense layer, we only need the variance of the output. It has been proven in (Garriga-Alonso et al., 2018) that this property propagate backwards and thus we only need the diagonal covariance. Garriga-Alonso et al. also proved that the covariances are independent of the output channel $j$. Therefore, we can calculate the covariance matrix in an iterative fashion, which is shown below:

Activations:

$$A_{j,g}^{(\ell+1)}(X) = b_j^{(\ell)} + \sum_{i=1}^{C^{(\ell)}} \sum_{h=1}^{H^{(\ell)}D^{(\ell)}} W_{j,i,g,h}^{(\ell)} \phi(A_{i,h}^{(\ell)}(X))$$

Compute Covariance:

$$v_g^{(\ell+1)}(X,X') = \mathbb{C}[A_{j,g}^{(\ell+1)}(X), A_{j,g}^{(\ell+1)}(X')]$$
$$= \sigma_b^2 + \sigma_w^2 \sum_{h \in g^{th} patch} s_h^{(\ell)}(X,X')$$

where:

$$s_h^{(\ell)}(X,X') = \mathbb{E}[\phi(A_{i,h}^{(\ell+1)}(X))\phi(A_{i,h}^{(\ell+1)}(X'))]$$

If we use the Rectified Linear Units (ReLU) activation as $\phi$, we would have:

$$s_g^{(\ell)}(X,X')$$
$$= \frac{\sqrt{v_g^{(\ell)}(X,X)v_g^{(\ell)}(X',X')}}{\pi}(\sin\theta_g^{(\ell)} + (\pi - \theta_g^{(\ell)}))$$

where:

$$\theta_g^{(\ell)} = \cos^{-1}(v_g^{(\ell)}(X,X') / \sqrt{v_g^{(\ell)}(X,X)v_g^{(\ell)}(X',X')})$$

Now we have all the required components to give a GP representation to the Vanilla CNNs. Given a GP classification or regression task, we can compute the prediction and the covariance matrix using the procedures described above.

### 3.2. Average Pooling

With the average pooling operation, we have the new rule of updating $a_j^{(\ell)}$. The average pool could be treated as the original feature map doing convolution with a fixed filter of size $a \times b$ and stride $a \times b$. Hence just like the convolution operation can be rewritten into matrix multiplication, average pool could be represented by a matrix $A$ (Figure 1) and

$$a_j^{(\ell+1)}(X) = b_j^{(\ell)} + \sum_{i=1}^{C^{(\ell)}} AW_{j,i}^{(\ell)} \phi(a_i^{(\ell)}(X))$$

The feature-maps is thus given by

$$a_j^{(\ell+1)}(X,X') = b_j^{(\ell)}\mathbf{1} + \sum_{i=1}^{C^{(\ell)}} \begin{bmatrix} AW_{j,i}^{(\ell)} & \mathbf{0} \\ \mathbf{0} & AW_{j,i}^{(\ell)} \end{bmatrix} \phi(a_i^{(\ell)}(X))$$

The kernel recursion can be calculated correspondingly with the same expression given in (Garriga-Alonso et al., 2018). Note that $W_{j,i}^{(\ell)}$ is iid, $A$ is a matrix that is invariant to each layer. Hence, within one layer one can view $A$ as a constant matrix. Thus $AW_{j,i}^{(\ell)}$ is iid. Define $W_{j,i}^{*(\ell)} = AW_{j,i}^{(\ell)}$ and this would be the new weight of the matrix. Following the proof of (Garriga-Alonso et al., 2018), adding average pool preserves the fact that $a_j^{(\ell+1)}(X,X')$ and $a_{j'}^{(\ell+1)}(X,X')$ are iid for $j \neq j'$.

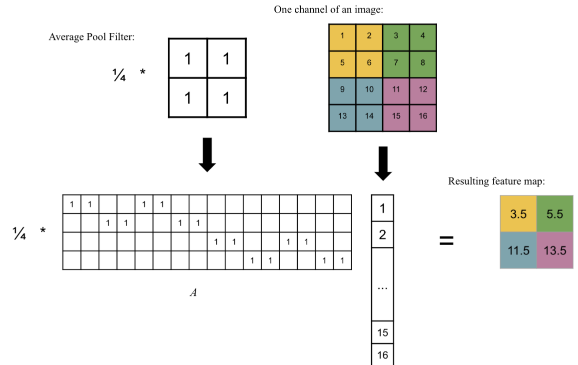


*Figure 1.* The 2x2 filter of average pooling on a 4x4 feature is converted to be the matrix $A$ and the feature is flattened. In this way, the average pooling operation is represented by matrix multiplication between a sparse constant matrix and the feature

### 3.3. Concatenation Operations and Dense Blocks

Consider the ResNets architecture, which add a skip-connection that bypasses the non-linear transformations with an identity function. We denote the output of the $\ell^{th}$ layer as $x_\ell$ and the non-linear transformation at layer $\ell$ as $H_\ell(\cdot)$. Then in Resnets, let $s$ be the number of layers that the skip connection spans. We will have the input for the $(\ell+1)^{th}$ layer being:

$$x_\ell = H_\ell(x_{\ell-1}) + x_{\ell-s}$$

According to (Garriga-Alonso et al., 2018), in the GP framework, to represent the bypassing operation in the GP framework, we will have the NN recursion becomes:

$$a_j^{(\ell+1)}(X) = a_j^{(\ell-s)}(X) + b_j^{(\ell)} + \sum_{i=1}^{C^{(\ell)}} W_{j,i}^{(\ell)} \phi(a_i^{(\ell)}(X))$$

And the kernel recursion becomes:

$$v_g^{(\ell+1)}(X, X') = v_g^{(\ell-s)}(X, X') + \sigma_b^2 + \sigma_w^2 \sum_{h \in g^{th} patch} s_g^{(\ell)}(X, X')$$

Recall the expressions for activations and covariance matrix in section 3.1, the bypassing operation is just using the sum of the output of some previous layer $A_j^{(\ell-s)}(X)$ and the output of the $\ell^{th}$ layer as the input to the $(\ell+1)^{th}$ later. Correspondingly we can get the formula for calculating the kernel matrix.

Now we consider the concatenation operation for dense connectivity. To achieve dense connection, the $\ell^{th}$ layer receives the feature-maps of all preceding layers $x_0, ..., x_{\ell-1}$ as its input:

$$x_\ell = H_\ell([x_0, x_1, ..., x_{\ell-1}])$$

Then correspondingly, in the GP representation, the NN recursion becomes:

$$a_j^{(\ell+1)}(X) = b_j^{(\ell)} + \sum_{y=0}^{\ell} \sum_{i=1}^{C^{(y)}} W_{j,i}^{(y)} \phi(a_i^{(y)}(X))$$

The concatenation operation is analogous to the skipping operation in (Garriga-Alonso et al., 2018): instead of adding the output of one previous layer, we are summing up the outputs of all previous layers in a densely connected block and passing them into the activation function. We will have the kernel recursion becomes:

$$v_g^{(\ell+1)}(X, X') = \sigma_b^2 + \sigma_w^2 \sum_{y=0}^{\ell} \sum_{h \in g^{th}\text{patch}} s_h^{(y)}(X, X')$$

Again, compare to a normal CNN model, when calculate activations and covariance matrix, we need to consider the output of all previous layers in the dense block. In terms of activations, we need to sum over the feature-maps from all previous layers and feed in the transformation function of the current layer. For the covariance matrix, we alsp need to sum over all image patches from all previous layers in the densely connected block.

For CNN models, the advantage of dense connection is, it can utilize the conditional information of previous layers to make the current layer more expressive. Here we present the new kernel recursion representing dense connection in a neural networks. With the concatenation operation, our framework can construct GP representations of more complicated CNN architectures, and hopefully also exploit the structural properties of the corresponding CNN model.

## 4. Experiments

All the experiments on MNIST and CIFAR-10 are run on 2 GeForce GTX 1080 GPUs.

*Table 1.* GP accuracy on different architecture

| SETTING | BASELINE | WITH AVG.POOL |
|---|---|---|
| 3 CONV LAYERS | **96.2** | **95.87** |
| 6 CONV LAYERS | 95.87 | 95.58 |
| 9 CONV LAYERS | 95.82 | 95.48 |
| 12 CONV LAYERS | 95.72 | 95.36 |
| 15 CONV LAYERS | 95.48 | 95.18 |
| 18 CONV LAYERS | 95.11 | 94.85 |

*Table 2.* GP run time on MNIST

| SETTING | VANILLA CONVNET |
|---|---|
| 3 CONV LAYERS | 157.17 |
| 6 CONV LAYERS | 271.24 |
| 9 CONV LAYERS | 392.96 |
| 12 CONV LAYERS | 516.57 |
| 15 CONV LAYERS | 632.11 |

### 4.1. MNIST

We run the experiments on MNIST classification task to test our CNN with Average Pool performance against the baseline implementation of Garriga-Alonso et al.. The result is shown in Table 1, with different hyperparameters and model architectures.

For the same number of layers, we compare the baseline performance and the performance of the model after randomly applying average pooling to 3 convolution layers. We test it through different deep convolutional neural network architectures. From the table we see the model with average pooling performs slightly worse than the baseline. Surprisingly, it could be discovered that as the number of layers increases, the accuracy of the model drops for both models. In (Garriga-Alonso et al., 2018), the author just random search for the best solution but ignore systematic experiment on how the number of layers as a hyperparameter of the Gaussian Process effect the performance of the classification task. This is an interesting finding because it directly contradicts with the nature of deep convolutional neural network where deeper architecture tends to produce higher accuracy.

In terms of running time, the amount of time to compute the covariance matrix and fit the data is much faster than the amount of time used to train a convolutional neural network counterpart.

### 4.2. CIFAR-10

We run the experiments on CIFAR10 classification task on 3 models:

*Table 3.* GP accuracy on different architecture on CIFAR 10 in percentage. From the first block to the third block, the number of convolution layers is 26,38,50 respectively. Experiments on the RBF kernel only runs for 10 epoch. We stopped running more experiments using the RBF kernel due to the long running time.

| SAMPLES | CONVNET | RESNET | DENSENET | RBF |
|---|---|---|---|---|
| 1000 | 38.6 | 39.2 | 39.0 | 14.3 |
| 5000 | 50.9 | 49.8 | 49.7 | - |
| 25000 | 61.6 | 56.5 | 60.7 | - |
| 1000 | 39 | 39.9 | 40.1 | - |
| 5000 | 51.9 | 50.6 | 50.9 | - |
| 25000 | 63.4 | 61.4 | 62 | - |
| 1000 | 39.2 | 40.6 | 40.9 | - |
| 5000 | 51.5 | 50.5 | 50.2 | - |
| 25000 | 63.0 | 61.9 | 60.9 | - |

1. Vanilla ConvNet
2. ResNet
3. DenseNet

For the ResNet architecture, we built residual connection between every 4 layers. For the DenseNet architecture, we built dense block with 6 layers and all the layers inside one block is densely connected. Between different blocks, we apply convolution with strides [2,2]. For each of the three models, we run the experiment with number of convolution layers to be 26, 38, 50. For each model configuration, we run the model on 1000, 5000, 25000 training samples. To evaluate the performance of the DenseNet Kernel, we record the test accuracy on the test data and the time used to compute the kernels. Since the model here untrainable, the time required to fit the training data is equivalent to the time to calculate the covariance matrix.

We report the result of our experiments on different architectures and different sample sizes in Table 2 and Table 3.

Figure 2 illustrates the relationship between number of training points and the testing accuracy of the model. From the graph we see that given the number of layers of the neural network, accuracy is a linear function of $\log(samples)$:

$$acc \approx C_1^{(n)} \log(samples)$$

for some constant $C_1^{(n)}$ dependent on $n$. Figure 3 illustrates the relationship between the number of training points and the time needed to calculate the covariance matrix. We can see the calculation time is approximately a quadratic function of number of samples:

$$time \approx C_2^{(n)} \times samples$$

*Table 4.* GP running time on different architecture on CIFAR 10 in seconds. From the first block to the third block, the number of convolution layers is 26,38,50 respectively. Experiments on the RBF kernel only runs for 10 epoch. We stopped running more experiments using the RBF kernel due to the long running time.

| SAMPLES | CONVNET | RESNET | DENSENET | RBF |
|---|---|---|---|---|
| 1000 | 98.13 | 108.47 | 120.57 | 5852.66 |
| 5000 | 519.49 | 390.21 | 407.99 | - |
| 25000 | 9316.94 | 2323.73 | 2327.48 | - |
| 1000 | 116.5 | 126.77 | 177.95 | - |
| 5000 | 718.21 | 529.16 | 588.85 | - |
| 25000 | 13253.51 | 8700 | 8580.55 | - |
| 1000 | 135.24 | 150.85 | 250.51 | - |
| 5000 | 920.61 | 675.59 | 794.19 | - |
| 25000 | 17324.06 | 11554.01 | 12381.11 | - |

for some constant $C_2^{(n)}$ dependent on $n$.

Compare the accuracy of any specific model given the number of training samples, we see that the effect of accuracy inverse proportional to number of layers disappear in this case. However, unlike traditional deep neural network, it stays true that in our Gaussian process representation of deep convolutional neural network, the depth of the network is not directly related to the accuracy of the model.

From Figure 2, we see that when the sample number is large, ConvNet architecture results in a slightly higher accuracy than ResNet and DenseNet in all three variants. However, when the sample number is small, the contrary happens - ConvNet architecture results in a slightly lower accuracy. When layer is 26, DenseNet gives a higher accuracy than ResNet, but in other two variants, these two have very similar accuracy. From Figure 3, we see that ConvNet has a significantly longer running time than the other two architectures. ResNet and DenseNet have similar running time with DenseNet slightly slower. So we see that there is a trade-off between accuracy and speed. The significantly longer running time of ConvNet does not buy us much extra accuracy.

We also compared deep convolutional neural network kernel with the RBF kernel and trained the RBF kernel on 1000 samples. We stopped at 10 epochs and evaluated the accuracy and run time and it turns out RBF kernel gives much lower accuracy than deep convolutional neural network kernel (14.3%) and much higher time for fitting the data (5852.66 s). This is due to the fact that RBF kernel actually needs to be trained while deep convolutional neural network kernel doesn't. Due to the long time of training, incompetitive testing accuracy, and limited computational power, we stopped training on RBF kernels with more itera-

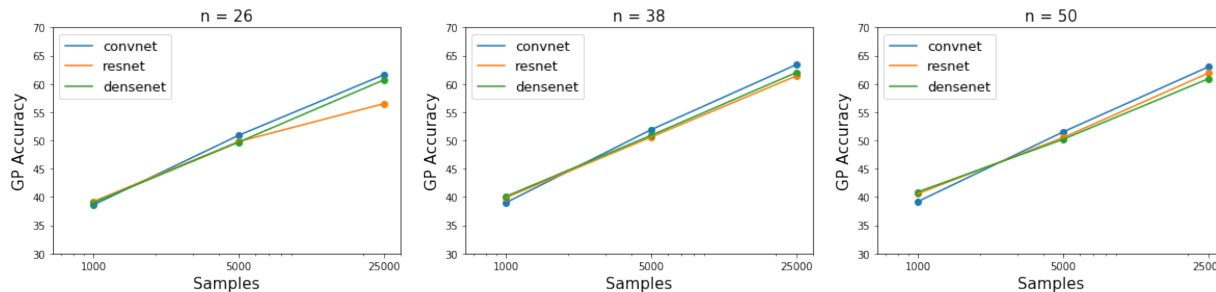


*Figure 2.* number of training samples versus classification accuracy on test set on vanilla ConvNet, ResNet, and DenseNet. *Left*: models with 26 conv layers; *Middle*: models with 38 conv layers; *Right*: models with 50 conv layers. From the graph we see that all three models have similar accuracy rate on classification. The vanilla CNN-GP works slightly better than DenseNet-GP slightly better than ResNet-GP

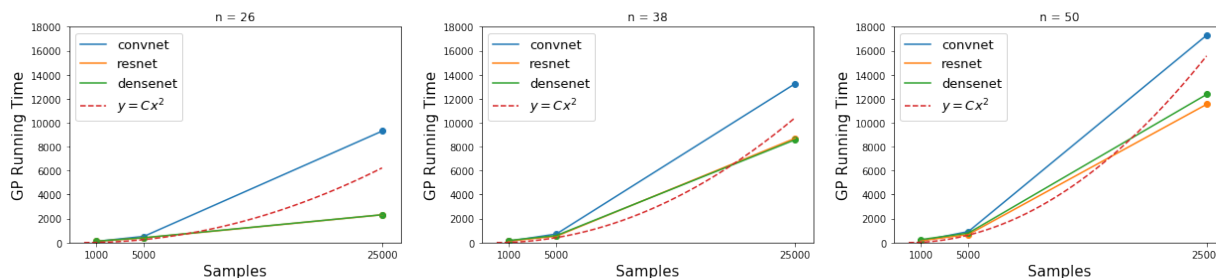


*Figure 3.* number of training samples versus time to calculate the covariance matrix on vanilla ConvNet, ResNet, and DenseNet. *Left*: models with 26 conv layers; *Middle*: models with 38 conv layers; *Right*: models with 50 conv layers. It can be seen from the graph that it takes much longer for the vanilla convolutional neural network GP to compute its covariance matrix than ResNet and DenseNet GP do as number of samples increase drastically. The dotted line here in every graph represents an quadratic formula that approximates the changing trend of running time in terms of number of training samples.

tions and more data.

## 5. Conclusion

We derived the expression of the Average Pooling operation, and we finished the implementation of the Average Pooling operation with the original Tensorflow implementation. We incorporated the Average Pooling operation into the baseline model and compared the performance of our model with the baseline. The accuracy of the models with the pooling operation (95.87%) is very similar to the model without the pooling operation (96.2%) on MNIST. Although that is lower than the state-of-the-art model performance, the time needed to fit the data to our model is much faster than training a convolutional neural network counterpart.

We also derived the expression of the concatenation operation, which enabled us to derive the GP representation of dense block (Huang et al., 2016). We implemented a GP representation for the densely connected CNN architecture with the TensorFlow implementation. We run experiments on the Vanilla CNN kernel GP, ResNets kernel GP, and the DenseNets kernel GP with different number of layers and different amount of input data. Experiments are run on the CIFAR-10 dataset. We evaluated the performance and run time of each model. We observe that the testing accuracy grow approximately linearly relative to $\log(samples)$. Models with different number of layers and different structures have similar performance: around $62\%$ accuracy with $25,000$ images being the training input. However, compared with vanilla CNN kernel GP, ResNet kernel GP and DenseNet kernel GP takes much shorter amount of time in fitting the data to the model especially on large number of training points: approximately on average 4000 5000 seconds faster while achieving similar testing accuracy. Hence, the DenseNet kernel is more competitive than the vanilla CNN kernel in terms of overall performance.

On CIFAR-10 dataset, the deep convolutional neural network kernel runs much faster than RBF kernel, about 48 times faster, with 1000 training points. Meanwhile, the accuracy of the RBF kernel is much worse than that of deep convolutional neural network kernel.

# 6. Discussion

## 6.1. Inducing point method failed

We finished the implementation of the framework with PyTorch and GPyTorch. To be specific, we implemented the deep kernel and ResNets kernel used in Garriga-Alonso et al. and tested its behaviour. The kernels gave the same kernel matrices as the original TensorFlow implementation with the random inputs. We are able to do GP classification with the GPyTorch GP model. The previous objective of this project was to incorporate scalable GP methods, such as the inducing point method with KISS-GP into our model. However, it is impossible to use popular inducing point based scalable GP methods, because the kernel of the GP representation of a CNN is not stationary, i.e. $K(X, X')$ for any two data points $X$ and $X'$ cannot be expressed as a function of $X - X'$. Note that $v_g^{(1)}(X, X')$ can be written as $m_g^{(1)}(X^T X')$ for some function $m_g^{(1)}$ and $v_g^{(l+1)}(X, X')$ can be written as a function with elements in $v^{(l)}(X, X')$. Hence, every $v_g^{(l)}(X, X')$ can be written as $m_g^{(l)}(X^T X')$ for some $m_g^{(l)}$. Works done by Scholkopf & Smola 2001 has shown that dot-product kernel is a better choice than stationary kernel on image classification tasks with inputs in $[-1, 1]$. Therefore we still use the TensorFlow implementation on this dot-product kernel for running our experiments. However, the GP model built with GPyTorch is more flexible and easier to configure.

## 6.2. Untrainable nature of the model

The authors of the original paper chose their hyperparamers using a random search approach - randomly drawing hyperparameter samples and picking set of hyperparameters with the highest accuracy on validation set. With the GPyTorch implementation, it is possible to train the model. But the GP representation of a CNN is a model that does not need training, since training the model will break the assumptions needed for GP behaviours. The key assumption of the model is the entries of all the convolutional filters are identically independently distributed Gaussian random variables. If we train the model, the backpropagation step will break the independence of the filter weights between layers.

## 6.3. Incongruent to neural networks depth behaviour

In the experiments with the MNIST dataset and Vanilla CNN Gaussian processes, the behavior that the model with less number of layers having better performance might be related to the fact that the model itself is confident in the classification task. However, this behavior disappear on experiments with the CIFAR-10 dataset, where the model has greater uncertainty. However, more interestingly, while it's well known that deeper neural networks gives better accuracy, this deep convolutional neural network kernel GP doesn't behave in the same way. The number of layers doesn't seem to have a direct correlation with the accuracy of classification on more complex and uncertain datasets. We can expect that we will also not be able to observe better performance with shallower network on CIFAR-100 and ImageNet. Because in these datasets, the model would not capture the distribution of the dataset as well as it did on MNIST.

# 7. Future Work

In this section, we describe some of the future works that can be extended from this paper. First, one can run experiments on larger and more complex datasets, such as ImageNet, to better understand the behavior of the Gaussian processes framework for Deep Neural Networks.

To make the framework more flexible, one can also develop more kernels for different activation functions. Besides the ArcCosine kernel, which is a natural counterpart to the ReLU activation function, one can derive kernels which are counterparts for other activation functions such as Exponential Linear Units (ELUs) and sigmoid.

# 8. Acknowledgement

We would like to thank Jacob R. Gardner and Geoff Pleiss for helping on GPytorch issues.

We would like to thank to Pavel Izmailov for his advice on the project.

# References

Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006.

Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, Dec 1989. ISSN 1435-568X. doi: 10.1007/BF02551274. URL https://doi.org/10.1007/BF02551274.

Funahashi, K.-I. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3):183 – 192, 1989. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(89)90003-8. URL http://www.sciencedirect.com/science/article/pii/0893608089900038.

Gardner, J. R., Pleiss, G., Bindel, D., Weinberger, K. Q., and Wilson, A. G. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *NeurIPS*, 2018.

Garriga-Alonso, A., Aitchison, L., and Rasmussen, C. E.

Deep Convolutional Networks as shallow Gaussian Processes. *ArXiv e-prints*, August 2018.

Hornik, K., Stinchcombe, M. B., and White, H. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. Densely Connected Convolutional Networks. *ArXiv e-prints*, August 2016.

Matthews, A. G. d. G., Rowland, M., Hron, J., Turner, R. E., and Ghahramani, Z. Gaussian Process Behaviour in Wide Deep Neural Networks. *ArXiv e-prints*, April 2018.

Neal, R. M. *Bayesian learning for neural networks*. Springer-Verlag, 1996.

Rasmussen, C. E. Gaussian processes for machine learning. MIT Press, 2006.

Schmidhuber, J. Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828, 2014. URL http://arxiv.org/abs/1404.7828.

Scholkopf, B. and Smola, A. J. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001. ISBN 0262194759.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going Deeper with Convolutions. *ArXiv e-prints*, September 2014.

Williams, C. Computing with infinite networks. In *Advances in Neural Information Processing Systems 9*, pp. 295–301. MIT Press, 1996.

Wilson, A. G. and Nickisch, H. Kernel Interpolation for Scalable Structured Gaussian Processes (KISS-GP). *ArXiv e-prints*, March 2015.

Wilson, A. G. and Prescott Adams, R. Gaussian Process Kernels for Pattern Discovery and Extrapolation. *ArXiv e-prints*, February 2013.

Wilson, A. G., Hu, Z., Salakhutdinov, R. R., and Xing, E. P. Stochastic variational deep kernel learning. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 2586–2594. Curran Associates, Inc., 2016.