# Neural Kernel Without Tangents

**ICML'20 Citation: 37**

Vaishaal Shankar, Alex Fang, Wenshuo Guo, Sara Fridovich-Keil, Ludwig Schmidt, Jonathan Ragan-Kelley, Benjamin Recht

UC Berkeley, MIT

# Motivation

- NTK, CNTK... do not match the performance of neural networks on most tasks of interest.
- The NTK constructions themselves are not only hard to compute, but their mathematical formulae are difficult to even write down.

# Problem Formulation

- Are there computationally tractable/easier kernels that approach the expressive power of neural networks?
- Is there a correlation between neural architecture performance and the performance of the associated kernel?

# Outline

- Main Idea
- Experiments
- Conclusion

# Main Idea

- Construct CNN architecture using only $3 \times 3$ **convolutions,** $2 \times 2$ **average pooling, ReLU**.
- **Compositional Kernel**: Kernelize $1..., L$ layers as kernel functions $k_1..., k_L$ and **compute the kernel hierarchily** $k_L(k_{L-1}(...k_1(x, y)))$ as the kernel of the corresponding CNN architecture.
- **5-layers compositional kernel**(in Myrtle5 architecture) can **significantly outperform(about 10% classification accuracy)** than **14-layers CNTK on CIFAR-10(Arora et al. 2020)** while the **training samples are less than 1000**.


*Figure 2.* A 5 layer network from the "Myrtle" family (Myrtle5).

# Methodology

- **Bag of features** is simply a generalization of a matrix or tensor: whereas a matrix is an indexed list of vectors, a bag of features is a collection of elements in a Hilbert space $\mathcal{H}$ with a finite, structured index set $\mathcal{B}$.
- EX: we can consider an **image** to be a bag of features where the **index set $\mathcal{B}$ is the pixel's row and column location** and $\mathcal{H}$ **is $\mathbb{R}^3$: at every pixel location, there is a corresponding vector encoding RGB in $\mathbb{R}^3$**.
- Given two bags of features with the same $(\mathcal{B}, \mathcal{H})$, we define the kernel function

$$k(\mathbf{X}, a, \mathbf{Z}, b) = \langle \mathbf{X}_a, \mathbf{Z}_b \rangle$$

It defines a **kernel matrix between two bags of features**: we compute the kernel function for each pair of indices in $\mathcal{B} \times \mathcal{B}$ to form a $|\mathcal{B}| \times |\mathcal{B}|$ **matrix**

# Input Kernel

**Input kernel.** The input kernel function $k_0$ relates all pixel vectors between all pairs of images in our dataset. Computationally, given $N$ images, we can use an image tensor $\boldsymbol{T}$ of shape $N \times D_1 \times D_2 \times 3$ to represent the whole dataset of images, and map this into a kernel tensor $\boldsymbol{K}_{out}$ of shape $N \times D_1 \times D_2 \times N \times D_1 \times D_2$. The elements of $\boldsymbol{K}_{out} = k_0(\boldsymbol{T})$ can be written as:

$$K_{out}[i, j, k, \ell, m, n] = \langle T[i, j, k], T[\ell, m, n] \rangle \, .$$

All subsequent operations operate on 6-dimensional tensors with the same indexing scheme.

# Convolution Kernel

**Convolution.** The convolution operation $c_w$ maps an input tensor $\boldsymbol{K}_{in}$ to an output tensor $\boldsymbol{K}_{out}$ of the same shape: $N \times D_1 \times D_2 \times N \times D_1 \times D_2$. $w$ is an integer denoting the size of the convolution (e.g. $w = 1$ denotes a $3 \times 3$ convolution).

The elements of $\boldsymbol{K}_{out} = c_w(\boldsymbol{K}_{in})$ can be written as:

$$K_{out}[i, j, k, \ell, m, n] = \sum_{dx=-w}^{w} \sum_{dy=-w}^{w} K_{in}[i, j + dx, k + dy, \ell, m + dx, n + dy]$$

For out-of-bound location indexes, we simply zero pad the $\boldsymbol{K}_{in}$ so all out-of-bound accesses return zero.

# Average Pooling Kernel

**Average pooling.** The average pooling operation $p_w$ downsamples the spatial dimension, mapping an input tensor $\boldsymbol{K}_{in}$ of shape $N \times D_1 \times D_2 \times N \times D_1 \times D_2$ to an output tensor $\boldsymbol{K}_{out}$ of shape $N \times (D_1/w) \times (D_2/w) \times N \times (D_1/w) \times (D_2/w)$. We assume $D_1$ and $D_2$ are divisible by $w$.

The elements of $\boldsymbol{K}_{out} = p_w(\boldsymbol{K}_{in})$ can be written as:

$$K_{out}[i, j, k, \ell, m, n] = \frac{1}{w^4} \sum_{a=1}^{w} \sum_{b=1}^{w} \sum_{c=1}^{w} \sum_{d=1}^{w}$$

$$\left( K_{in}[i, wj + a, wk + b, \ell, wm + c, wn + d] \right)$$

# ReLU Kernel

The ReLU embedding, $k_{relu}$, is shape preserving, mapping an input tensor $\boldsymbol{K}_{in}$ of shape $N \times D_1 \times D_2 \times N \times D_1 \times D_2$ to an output tensor $\boldsymbol{K}_{out}$ of shape $N \times D_1 \times D_2 \times N \times D_1 \times D_2$. To ease the notation, we define two auxiliary tensors: $\boldsymbol{A}$ with shape $N \times D_1 \times D_2$ and $\boldsymbol{B}$ with shape $N \times D_1 \times D_2 \times N \times D_1 \times D_2$, where the elements of each are:

$$A[i,j,k] = \sqrt{K_{in}[i,j,k,i,j,k]}$$

$$B[i,j,k,\ell,m,n] = \arccos\left(\frac{K_{in}[i,j,k,\ell,m,n]}{A[i,j,k]A[\ell,m,n]}\right)$$

# ReLU Kernel

The elements of $\boldsymbol{K}_{out} = k_{relu}(\boldsymbol{K}_{in})$ can be written as:

$$
\begin{aligned}
& K_{out}[i,j,k,\ell,m,n] \\
&= \frac{1}{\pi}\Bigg( A[i,j,k]A[\ell,m,n]\sin(B[i,j,k,\ell,m,n]) + \\
& (\pi - B[i,j,k,\ell,m,n])\cos(B[i,j,k,\ell,m,n]) \Bigg)
\end{aligned}
$$

It's the same as the **arccosine kernel** used in NTK. Refers to NIPS'09 Kernel Methods for Deep Learning

# Gaussian Kernel

In addition to the ReLU kernel, we also work with a normalized Gaussian kernel. The elements of $\boldsymbol{K}_{out} = k_{gauss}(\boldsymbol{K}_{in})$ can be written as:

$$\begin{aligned} & K_{out}[i,j,k,\ell,m,n] \\ = & A[i,j,k]A[\ell,m,n]\exp(B[i,j,k,\ell,m,n]-1) \end{aligned}$$

The normalized Gaussian kernel has a similar output response to the ReLU kernel (shown in Figure 1). Experimentally, we find the Gaussian kernel to be marginally faster and more numerically stable.

# Related to Neural Network

Let a simple convolution layer as

$$\Psi(U) = relu(W * U)$$

Where $W \in \mathbb{R}^{(2w+1)\times(2w+1)\times D_3\times D_4}$ is a weight tensor and $U \in \mathbb{R}^{N\times D_1\times D_2\times D_3}$ is the input, which can be $N$ images. Suppose the entries of $W$ are appropriately **scaled random Gaussian variables**. We can evaluate the following **expectation according to the calculation**

$$\mathbb{E}[\sum_{c=1}^{D_4}\Psi(U)[i,j,k,c]\Psi(U)[l,m,n,c]] = k_{relu}(c_w(k_0(U)))[i,j,k,l,m,n]$$

# Algorithm

**Algorithm 1** Compositional Kernel

**Input**

$\mathcal{N}$ Input architecture of $m$ layers from $\mathcal{A}$

$\mathcal{K}$ Map from $\mathcal{A}$ to layerwise operators

$\boldsymbol{X}$ Tensor of input images, shape $(N \times D \times D \times 3)$

**Output**

$\boldsymbol{K}_m$ Compositional kernel matrix, shape $(N \times N)$

$\boldsymbol{K}_0 = k_0(\boldsymbol{X})$

**for** $i = 1$ **to** $m$ **do**

$k_i \leftarrow \mathcal{K}(\mathcal{N}_i)$

$\boldsymbol{K}_i \leftarrow k_i(\boldsymbol{K}_{i-1})$

**end for**

# Experiment Setup

## MNIST, CIFAR-10, CIFAR-10.1, CIFAR-100 Dataset

- Myrtle5, 7, 10 with ReLU kernel
- ZCA whitening preprocessing
- Flip data augmentation to our kernel method by flipping every example in the training set across the vertical axis
- Kernel ridge regression with respect to one-hot labels

## 90 UCI Dataset

- Myrtle5, 7, 10 with Gaussian kernel
- Hinge loss with libSVM

## Architecture

All architectures that can be represented as a list of operations from the set **{conv3, pool2, relu}** as the **"Myrtle" family**. The right one is **Myrtle7** and the left on is **Myrtle10**

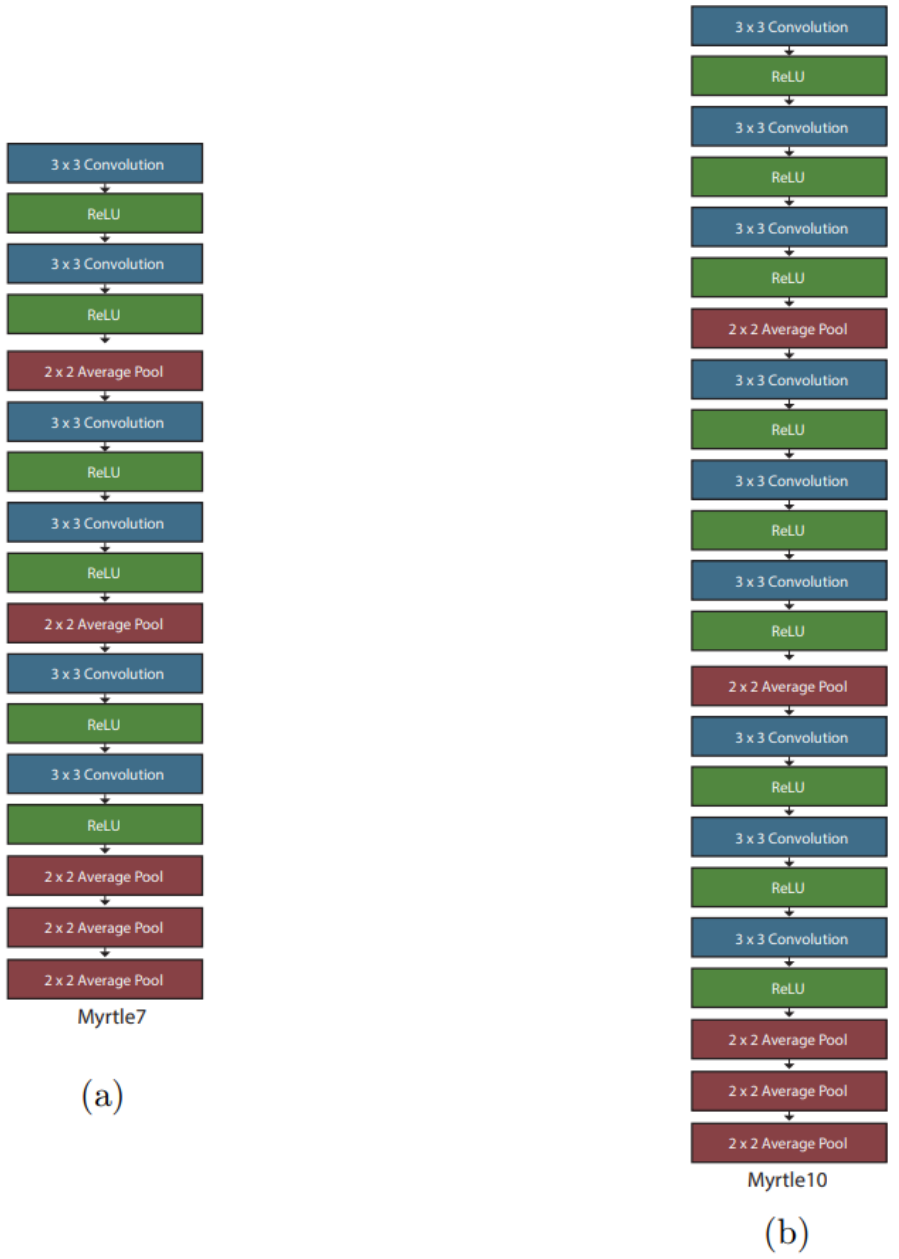


Figure 1: a) 7 layer b) 10 layer variants of the Myrtle architectures

# MNIST

*Table 1.* Classification performance on MNIST. All methods with convolutional structure have essentially the same performance.

| Method | MNIST Accuracy |
|---|---|
| NTK | 98.6 |
| ArcCosine Kernel | 98.8 |
| Gaussian Kernel | 98.8 |
| Gabor Filters + Gaussian Kernel | 99.4 |
| LeNet-5 (LeCun et al., 1998a) | 99.0 |
| CKN (Mairal et al., 2014) | 99.6 |
| Myrtle5 Kernel | 99.5 |
| Myrtle5 CNN | 99.5 |

# CIFAR-100

*Table 2.* Accuracy on CIFAR-100. All CNNs were trained with cross entropy loss.

| Method | CIFAR-100 Accuracy |
|---|---|
| Myrtle10-Gaussian Kernel | 65.3 |
| Myrtle10-Gaussian Kernel + Flips | 68.2 |
| Myrtle10 CNN | 64.7 |
| Myrtle10 CNN + Flips | 71.4 |
| Myrtle10 CNN + BatchNorm | 70.3 |
| Myrtle10 CNN + Flips + BatchNorm | 74.7 |

# 90 UCI

*Table 4.* Results on 90 UCI datasets for the NTK and Gaussian kernel (both tuned over 4 eval folds).

| Classifier | Friedman Rank | Average Accuracy (%) | P90 (%) | P95 (%) | PMA (%) |
|---|---|---|---|---|---|
| SVM NTK | 14.3 | $83.2 \pm 13.5$ | 96.7 | 83.3 | $97.3 \pm 3.8$ |
| SVM Gaussian kernel | 11.6 | $83.4 \pm 13.4$ | 95.6 | 83.3 | $97.5 \pm 3.7$ |

- **Friedman rank**: The ranking metric reports the average ranking of a given classifier compared to all other classifiers on datasets. The lower, the better.
- **P90/P95**: The percentage of datasets on which the classifier achieves more than 90%/95% of the maximum achievable accuracy. The higher, the better.
- **PMA**: The average percentage of the maximum accuracy of the classifier for datasets. The higher, the better.

# CIFAR-10

- Evaluate on 10,000 test images from CIFAR-10 and the additional 2,000 "harder" test images from CIFAR-10.1
- For all kernel results on CIFAR-10, we gained an improvement of roughly 0.5% with **Leave-One-Out tilting** and **ZCA augmentation** techniques.
- A substantial drop in accuracy for the compositional kernel without ZCA preprocessing.

*Table 3.* Classification performance on CIFAR-10.

| Method | CIFAR-10 Accuracy | CIFAR-10.1 Accuracy |
|---|---|---|
| Gaussian Kernel | 57.4 | - |
| CNTK + Flips (Li et al., 2019) | 81.4 | - |
| CNN-GP + Flips (Li et al., 2019) | 82.2 | - |
| CKN (Mairal, 2016) | 85.8 | - |
| Coates-NG + Flips (Recht et al., 2019) | 85.6 | 73.1 |
| Coates-NG + CNN-GP + Flips (Li et al., 2019) | 88.9 | - |
| ResNet32 | 92.5 | 84.4 |
| Myrtle5 Kernel + No ZCA | 77.7 | 62.2 |
| Myrtle5 Kernel | 85.8 | 71.6 |
| Myrtle7 Kernel | 86.6 | 73.1 |
| Myrtle10 Kernel | 87.5 | 74.5 |
| Myrtle10-Gaussian Kernel | 88.2 | 75.1 |
| Myrtle10-Gaussian Kernel + Flips | 89.8 | 78.3 |
| Myrtle5 CNN + No ZCA | 87.8 | 75.8 |
| Myrtle5 CNN | 89.8 | 79.0 |
| Myrtle7 CNN | 90.2 | 79.7 |
| Myrtle10 CNN | 91.2 | 79.9 |
| Myrtle10 CNN + Flips | 93.4 | 84.8 |
| Myrtle10 CNN + Flips + CutOut + Crops | 96.0 | 89.8 |

# Subsampled CIFAR-10

- Subsampled datasets are class balanced
- **Compositional kernel and NTK in the low data regime**
- **Network** with the same architecture as compositional kernel severely **underperforms both the compositional kernel and NTK in the low data regime**
- After adding batch normalization, the network outperforms both compositional kernel and the NTK

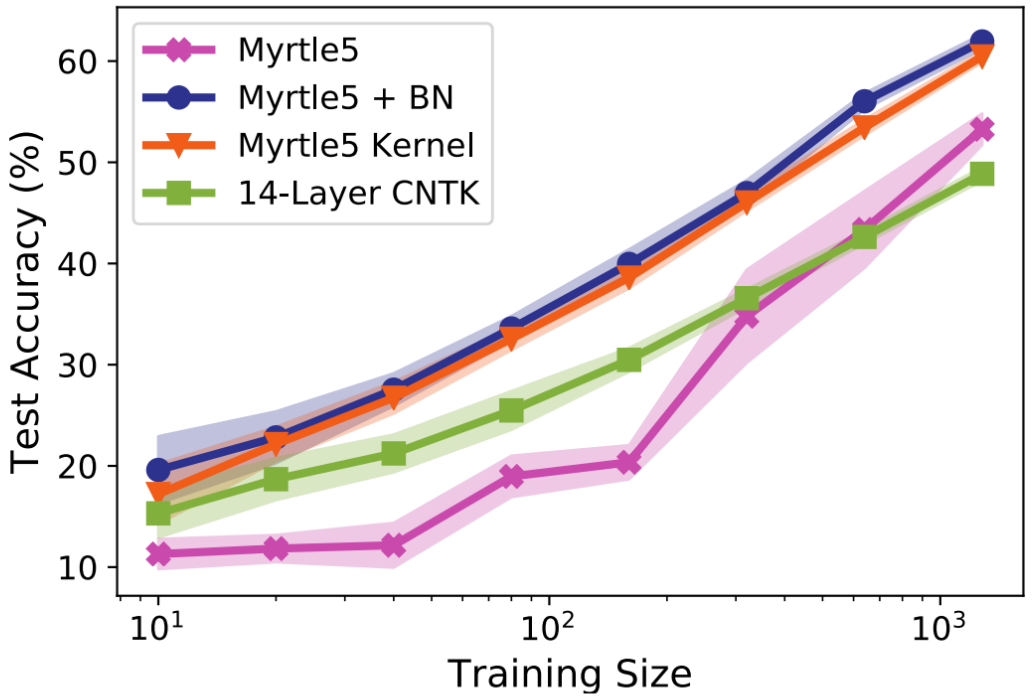


*Figure 3.* Accuracy results on random subsets of CIFAR-10, with standard deviations over 20 trials. The 14-layer CNTK results are from Arora et al. (2020).

# Conclusion

- Provide a promising starting point for designing practical, high performance, domain specific kernel functions
- Some notion of **compositionality and hierarchy** may be necessary to build kernel predictors that match the performance of neural networks
- **NTKs** themselves may **not actually provide particularly useful guides** to the practice of kernel methods.
- We may underscores the importance of proper preprocessing for kernel methods
- There **still performance gaps between kernel methods and neural networks** and the reasons remain unknown.

# Reference


- Preprocessing for deep learning: from covariance matrix to image whitening
- 知乎 - CNN数值——ZCA
- NIPS'09 Kernel Methods for Deep Learning
- ICLR'20 Harnessing the Power of Infinitely Wide Deep Nets on Small-data Tasks