Advanced Software Praktikum

Final Project Report

Nida Murad

Analysis of the paper "End-to-End Kernel Learning with Supervised Convolutional Kernel Networks"

`https://github.com/nida612/SoftwarePrakitkum23_`

Heidelberg University

October 2, 2023

Contents

1 Introduction & Background **1**
1.1 Reproducing Kernel Hilbert Space (RKHS) 2
1.2 Convolution Nueral Network: Overview 3

2 Convolution Kernel Network **4**
2.1 Building a Convolution Kernel Network 5
2.2 Approximating the Multilayer Convolutional Kernel using the Gaussian Kernel 6
2.3 Learning Hierarchies of Subspaces with CKNs 7
2.3.1 Utilizing the Kernel Trick to Create Local Image Representations: . . 7
2.3.2 Project onto a finite-dimensional subspace of the RKHS with convolution layers 8
2.3.3 Linear Pooling for Invariance: 9
2.3.4 Multilayer Representation 9
2.4 Backpropagation Rules for Convolutional Kernel Networks 10
2.5 Optimization 11

3 Experiment: Image classification **11**

A RKHS: in depth **13**
A.1 Functional Analysis Background 13
A.2 Proof of Theorem 2 15
A.3 Connection between RKHSs and the ReLU function 15

B Example/Run through of CNN **16**

1 Introduction & Background

Machine learning was dominated before deep learning by non-parametric models with positive definite kernels. These kernel methods are versatile but computationally intensive. However, traditional kernel methods separate data representation from learning, making them incompatible with end-to-end learning, which is a key feature of deep neural networks. The paper's [1] main focus is on addressing this issue in the context of image modelling using convolutional kernel networks. These networks map image neighbourhoods to points in a kernel space and build hierarchical representations, providing an approximation of convolutional neural networks without supervision.[1]
The Section 1 recaps the elementary concepts used in this paper:

— Reproducing Kernel Hilbert space (RKHS)

— Convolution Nueral Network

1.1 Reproducing Kernel Hilbert Space (RKHS)

In order to define RKHS, we will make use of several terms from functional analysis, which can be found in Appendix A.1

Definition 1.1. An evaluation functional over the Hilbert space of functions $\mathcal{H}$ is a linear functional $\mathcal{F}_t : \mathcal{H} \to \mathbb{R}$ that evaluates each function in the space at the point t, or

$$\mathcal{F}_t[f] = f(t) \quad \text{for all } f \in \mathcal{H}.$$

Definition 1.2. A Hilbert space $\mathcal{H}$ is a reproducing kernel Hilbert space (RKHS) if the evaluation functionals are bounded, i.e. if for all t there exists some $M > 0$ such that

$$|\mathcal{F}_t[f]| = |f(t)| \leq M\|f\|_H \quad \text{for all } f \in \mathcal{H}$$

This condition is not trivial. For $L_2[a, b]$, we showed above that there exist functions that are squareintegrable, but which have arbitrarily large values on finite point sets. In this case, no choice of M will give us the appropriate bound on these functions on these point sets.

While this condition might seem obscure or specific, it is actually quite general and is the weakest possible condition that ensures us both the existence of an inner product and the ability to evaluate each function in the space at every point in the domain. In practice, it is difficult to work with this definition directly. We would like to establish an equivalent notion that is more useful in practice. To do this, we will need the "reproducing kernel" from which the reproducing kernel Hilbert space takes its name.

First, from the definition of the reproducing kernel Hilbert space, we can use the Riesz representation theorem to prove the following property.

Theorem 1. If $\mathcal{H}$ is a RKHS, then for each $t \in X$ there exists a function $K_t \in \mathcal{H}$ (called the representer of t) with the reproducing property

$$\mathcal{F}_t[f] = \langle K_t, f \rangle_{\mathcal{H}} = f(t) \quad \text{for all } f \in \mathcal{H}.$$

This allows us to represent our linear evaluation functional by taking the inner product with an element of $\mathcal{H}$. Since K_t is a function in $\mathcal{H}$, by the reproducing property, for each $x \in X$ we can write

$$K_t(x) = \langle K_t, K_x \rangle_H .$$

We take this to be the definition of reproducing kernel in $\mathcal{H}$.

Definition 1.3. The reproducing kernel (rk) of $\mathcal{H}$ is a function $K : X \times X \to \mathbb{R}$, defined by

$$K(t, x) := K_t(x)$$

In general, we have the following definition of a reproducing kernel.

Definition 1.4. Let X be some set, for example a subset of $\mathbb{R}^d$ or $\mathbb{R}^d$ itself. A function $K : X \times X \to \mathbb{R}$ is a reproducing kernel if it is symmetric, i.e. $K(x, y) = K(y, x)$, and

positive definite:

$$\sum_{i,j=1}^{n} c_i c_j K(t_i, t_j) \geq 0$$

for any $n \in \mathbb{N}$ and choice of $t_1, \ldots, t_n \in X$ and $c_1, \ldots, c_n \in \mathbb{R}$.

Having this general notion of a reproducing kernel is important because it allows us to define an RKHS in terms of its reproducing kernel, rather than attempting to derive the kernel from the definition of the function space directly. The following theorem formally establishes the relationship between the RKHS and a reproducing kernel.

Theorem 2. A RKHS defines a corresponding reproducing kernel. Conversely, a reproducing kernel defines a unique RKHS.

Please check appendix AA.1 for the proof
Examples of reproducing kernels

— Linear kernel

$$K(x, x') = x \cdot x'$$

— Gaussian kernel

$$K(x, x') = \mathrm{e}^{-\frac{\|x - x'\|^2}{\sigma^2}}, \quad \sigma > 0 \tag{1}$$

— Polynomial kernel

$$K(x, x') = (x \cdot x' + 1)^d, \quad d \in \mathbb{N}$$

Note : It might also be interesting to check out the connection between ReLU functions and RKHS in Appendix A.3

1.2 Convolution Nueral Network: Overview

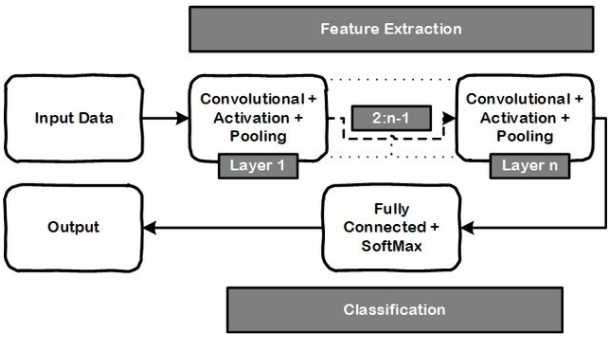

Figure 1: Block-diagram-of-CNN-architecture[23]

CNNs are highly effective in computer vision tasks due to their ability to automatically learn and extract hierarchical features from images. They have played a crucial role in advancing fields like image recognition, object detection, facial recognition, and more.

— **Convolution**: CNNs use convolutional layers to detect features in the input data. Convolution involves sliding a small filter (also known as a kernel) over the input

data. The filter applies a mathematical operation to a small region of the input at a time, producing a feature map that highlights specific patterns or features in the data. These patterns can range from simple edges to more complex shapes and textures.

— **Pooling**: After convolution, CNNs often use pooling layers to reduce the spatial dimensions of the feature maps. Pooling helps reduce the computational load and makes the network more robust to variations in scale and position of features. Max pooling, for example, selects the maximum value from a small region of the feature map, effectively downscaling it.

— **Activation Functions**: Non-linear activation functions like ReLU (Rectified Linear Unit) are applied to the feature maps after each convolution and pooling operation. This introduces non-linearity into the model, enabling it to capture complex relationships within the data.

— **Fully Connected Layers**: After several convolutional and pooling layers, CNNs typically have one or more fully connected layers, which are traditional neural network layers. These layers combine the high-level features extracted by the previous layers to make final predictions or classifications.

— **Output Layer**: The final fully connected layer of the CNN produces the output, which depends on the specific task. For image classification, this might be a softmax layer that assigns probabilities to different classes.

— **Training**: CNNs are trained using a labeled dataset through a process called backpropagation and optimization algorithms (e.g., gradient descent). During training, the network adjusts its internal parameters (weights and biases) to minimize the difference between its predictions and the true labels in the training data.

In simple terms, a CNN learns to see patterns in an image by sliding small windows over it, combining those patterns to understand the content, and then making predictions based on what it has learned. This process is repeated through multiple layers, each building upon the features learned in the previous layer, ultimately allowing the network to recognize complex objects or scenes in images.

Note: For a better understanding, check Appendix B for a run through/example.

2 Convolution Kernel Network

An important goal in visual recognition is to devise image representations that are invariant to particular transformations. Unlike traditional approaches where neural networks are learned either to represent data or for solving a classification task, CKN network learns to approximate the kernel feature map on training data. [8]

2.1 Building a Convolution Kernel Network

By teaching CNNs to be invariant, we obtain simple network architectures that achieve a similar accuracy to more complex ones, while being easy to train and robust to overfitting.[8]

Definition 2.1. An **image feature map** φ is a function $\varphi : \Omega \to \mathcal{H}$, where Ω is a (usually discrete) subset of $[0,1]^d$ representing normalized "coordinates" in the image and $\mathcal{H}$ is a Hilbert space.

Definition 2.2. (**Convolutional Kernel with Single Layer**).
Let us consider two images represented by two image feature maps, respectively φ and $\varphi' : \Omega \to \mathcal{H}$, where Ω is a set of pixel locations, and $\mathcal{H}$ is a Hilbert space. The one-layer convolutional kernel between φ and φ' is defined as

$$K\left(\varphi, \varphi'\right) := \sum_{\mathbf{z}\in\Omega} \sum_{\mathbf{z}'\in\Omega} \|\varphi(\mathbf{z})\|_{\mathcal{H}} \left\|\varphi'\left(\mathbf{z}'\right)\right\|_{\mathcal{H}} e^{-\frac{1}{2\tilde{\sigma}^2}\left\|\mathbf{z}-\mathbf{z}'\right\|_2^2} e^{-\frac{1}{2\sigma^2}\left\|\bar{\varphi}(\mathbf{z})-\bar{\varphi}'\left(\mathbf{z}'\right)\right\|_{\mathcal{H}}^2}, \tag{2}$$

Given $\mathbf{z}$ in Ω, the point $\varphi(\mathbf{z})$ represents some characteristics of the image at location $\mathbf{z}$, or in a neighborhood of $\mathbf{z}$. It consists of a sum of pairwise comparisons between the image features $\varphi(\mathbf{z})$ and $\tilde{\varphi}'\left(\mathbf{z}'\right)$ computed at all spatial locations $\mathbf{z}$ and $\mathbf{z}'$ in Ω.The parameters β and σ respectively control these two definitions of "closeness". Therefore, the role of β is to control how much the kernel is locally shift-invariant.Hence, β and σ are smoothing parameters of Gaussian kernels (1), and $\tilde{\varphi}(\mathbf{z}) := (1/\|\varphi(\mathbf{z})\|_{\mathcal{H}})\,\varphi(\mathbf{z})$ if $\varphi(\mathbf{z}) \neq 0$ and $\tilde{\varphi}(\mathbf{z}) := 0$ otherwise. Similarly, $\tilde{\varphi}'\left(\mathbf{z}'\right)$ is a normalized version of $\varphi'\left(\mathbf{z}'\right)$ [8]

Concretely, the kernel K_k between two patches of φ_{k-1} and φ'_{k-1} at respective locations $\mathbf{z}_k$ and $\mathbf{z}'_k$ is

$$\sum_{\mathbf{z}\in\mathcal{P}_k} \sum_{\mathbf{z}'\in\mathcal{P}_k} \left\|\varphi_{k-1}\left(\mathbf{z}_k+\mathbf{z}\right)\right\| \left\|\varphi'_{k-1}\left(\mathbf{z}'_k+\mathbf{z}'\right)\right\| e^{-\frac{1}{2\beta_k^2}\left\|\mathbf{z}-\mathbf{z}'\right\|_2^2} e^{-\frac{1}{2\sigma_k^2}\left\|\tilde{\varphi}_{k-1}\left(\mathbf{z}_k+\mathbf{z}\right)-\tilde{\varphi}'_{k-1}\left(\mathbf{z}'_k+\mathbf{z}'\right)\right\|^2}, \tag{3}$$

where $\|.\|$ is the Hilbertian norm of $\mathcal{H}_{k-1}$. Information encoded in the k-th layer differs from the $(k-1)$-th one in two aspects: first, each point $\varphi_k\left(\mathbf{z}_k\right)$ in layer k contains information about several points from the $(k-1)$-th layer and can possibly represent larger patterns; second, the new feature map is more locally shift-invariant than the previous one due to the term involving the parameter β_k in (3).

With reference to original model of CKN [8] the scheme below approximates the kernel map of K defined in (2) at layer k by finite-dimensional spatial maps $\xi_k : \Omega'_k \to \mathbb{R}^{p_k}$, where Ω'_k is a set of coordinates related to Ω_k, and p_k is a positive integer controlling the quality of the approximation. Consider indeed two images represented at layer k by image feature maps φ_k and φ'_k, respectively. Then,

a) the corresponding maps ξ_k and ξ'_k are learned such that $K\left(\varphi_{k-1}, \varphi'_{k-1}\right) \approx \langle \xi_k, \xi'_k \rangle$, where $\langle ., . \rangle$ is the Euclidean inner-product acting as if ξ_k and ξ'_k were vectors in $\mathbb{R}^{\left|\Omega'_k\right| p_k}$;

b) the set Ω'_k is linked to Ω_k by the relation $\Omega'_k = \Omega_k + \mathcal{P}'_k$ where $\mathcal{P}'_k$ is a patch shape, and the quantities $\varphi_k\left(\mathbf{z}_k\right)$ in $\mathcal{H}_k$ admit finite-dimensional approximations $\psi_k\left(\mathbf{z}_k\right)$ in

$\mathbb{R}^{|\mathcal{P}'_k|p_k}$; as illustrated in Figure 1(b), $\psi_k(\mathbf{z}_k)$ is a patch from ξ_k centered at location $\mathbf{z}_k$ with shape $\mathcal{P}'_k$;

c) an activation map $\zeta_k : \Omega_{k-1} \mapsto \mathbb{R}^{p_k}$ is computed from ξ_{k-1} by convolution with p_k filters followed by a non-linearity. The subsequent map ξ_k is obtained from ζ_k by a pooling operation.

We call this approximation scheme a convolutional kernel network (CKN). Very comparable to CNNs, this approach enjoys similar benefits such as efficient prediction at test time, and involves the same set of hyper-parameters: number of layers, numbers of filters p_k at layer k, shape $\mathcal{P}'_k$ of the filters, sizes of the feature maps. Training a CKN can be argued to be as simple as training a CNN in an unsupervised manner [22] since the main difference is in the cost function that is optimized[8].

2.2 Approximating the Multilayer Convolutional Kernel using the Gaussian Kernel

A key component of our formulation is the **fast approximation of the Gaussian Kernel**. We start by approximating it by a linear operation with learned filters followed by a pointwise non-linearity which leads to the following [17] [18]. To prevent the curse of dimensionality, we learn to approximate the kernel on training data, which is intrinsically low-dimensional. We optimize importance weights $\boldsymbol{\eta} = [\eta_l]_{l=1}^p$ in $\mathbb{R}_+^p$ and sampling points $\mathbf{W} = [\mathbf{w}_l]_{l=1}^p$ in $\mathbb{R}^{m\times p}$ on n training pairs $(\mathbf{x}_i, \mathbf{y}_i)_{i=1,\dots,n}$ in $\mathbb{R}^m \times \mathbb{R}^m$:

$$\min_{\boldsymbol{\eta}\in\mathbb{R}_+^p, \mathbf{W}\in\mathbb{R}^{m\times p}} \left[\frac{1}{n} \sum_{i=1}^{n} \left(e^{-\frac{1}{2\sigma^2}\|\mathbf{x}_i - \mathbf{y}_i\|_2^2} - \sum_{l=1}^{p} \eta_l e^{-\frac{1}{\sigma^2}\|\mathbf{x}_i - \mathbf{w}_l\|_2^2} e^{-\frac{1}{\sigma^2}\|\mathbf{y}_i - \mathbf{w}_l\|_2^2} \right)^2 \right]. \quad (4)$$

We then build our convolutional kernel network. We start by making assumptions on the input data, and then present the learning scheme and its approximation principles.

The zeroth layer. We assume that the input data is a finite-dimensional map $\xi_0 : \Omega'_0 \to \mathbb{R}^{p_0}$, and that $\varphi_0 : \Omega_0 \to \mathcal{H}_0$ "extracts" patches from ξ_0. Formally, there exists a patch shape $\mathcal{P}'_0$ such that $\Omega'_0 = \Omega_0 + \mathcal{P}'_0, \mathcal{H}_0 = \mathbb{R}^{p_0|\mathcal{P}'_0|}$, and for all $\mathbf{z}_0$ in $\Omega_0, \varphi_0(\mathbf{z}_0)$ is a patch of ξ_0 centered at $\mathbf{z}_0$. Then, property (b) described at end of section 2.1 is satisfied for $k = 0$ by choosing $\psi_0 = \varphi_0$.

The convolutional kernel network. The zeroth layer being characterized, we present in Algorithms 1 and 2 the subsequent layers and how to learn their parameters in a feedforward manner. It is interesting to note that the input parameters of the algorithm are exactly the same as a CNN - that is, number of layers and filters, sizes of the patches and feature maps (obtained here via the subsampling factor). Ultimately, CNNs and CKNs only differ in the cost function that is optimized for learning the filters and in the choice of non-linearities.

For approximating, in the paper [8] one proceeds recursively to show that the kernel approximation property (a) is satisfied; we assume that (b) holds at layer k–1, and then,

Algorithm 1 Convolutional kernel network - learning the parameters of the k-th layer.

input $\xi_{k-1}^1, \xi_{k-1}^2, \ldots : \Omega'_{k-1} \to \mathbb{R}^{p_{k-1}}$ (sequence of $(k-1)$-th maps obtained from training images); $\mathcal{P}'_{k-1}$ (patch shape); p_k (number of filters); n (number of training pairs);

1: extract at random n pairs $(\mathbf{x}_i, \mathbf{y}_i)$ of patches with shape $\mathcal{P}'_{k-1}$ from the maps $\xi_{k-1}^1, \xi_{k-1}^2, \ldots$;

2: if not provided by the user, set σ_k to the 0.1 quantile of the data $(\|\mathbf{x}_i - \mathbf{y}_i\|_2)_{i=1}^n$;

3: **unsupervised learning:** optimize (4) to obtain the filters $\mathbf{W}_k$ in $\mathbb{R}^{|\mathcal{P}'_{k-1}| p_{k-1} \times p_k}$ and $\boldsymbol{\eta}_k$ in $\mathbb{R}^{p_k}$;

output $\mathbf{W}_k$, $\boldsymbol{\eta}_k$, and σ_k (smoothing parameter);

Algorithm 2 Convolutional kernel network - computing the k-th map form the $(k-1)$-th one.

input $\xi_{k-1} : \Omega'_{k-1} \to \mathbb{R}^{p_{k-1}}$ (input map); $\mathcal{P}'_{k-1}$ (patch shape); $\gamma_k \geq 1$ (subsampling factor); p_k (number of filters); σ_k (smoothing parameter); $\mathbf{W}_k = [\mathbf{w}_{kl}]_{l=1}^{p_k}$ and $\boldsymbol{\eta}_k = [\eta_{kl}]_{l=1}^{p_k}$ (layer parameters);

1: **convolution and non-linearity:** define the activation map $\zeta_k : \Omega_{k-1} \to \mathbb{R}^{p_k}$ as

$$\zeta_k : \mathbf{z} \mapsto \|\psi_{k-1}(\mathbf{z})\|_2 \left[\sqrt{\eta_{kl}} e^{-\frac{1}{\sigma_k^2} \left\| \tilde{\psi}_{k-1}(\mathbf{z}) - \mathbf{w}_{kl} \right\|_2^2} \right]_{l=1}^{p_k}, \tag{5}$$

where $\psi_{k-1}(\mathbf{z})$ is a vector representing a patch from ξ_{k-1} centered at $\mathbf{z}$ with shape $\mathcal{P}'_{k-1}$, and the vector $\tilde{\psi}_{k-1}(\mathbf{z})$ is an ℓ_2-normalized version of $\psi_{k-1}(\mathbf{z})$. This operation can be interpreted as a spatial convolution of the map ξ_{k-1} with the filters $\mathbf{w}_{kl}$ followed by pointwise non-linearities;

2: set β_k to be γ_k times the spacing between two pixels in Ω_{k-1};

3: **feature pooling:** Ω'_k is obtained by subsampling Ω_{k-1} by a factor γ_k and we define a new map $\xi_k : \Omega'_k \to \mathbb{R}^{p_k}$ obtained from ζ_k by linear pooling with Gaussian weights:

$$\xi_k : \mathbf{z} \mapsto \sqrt{2/\pi} \sum_{\mathbf{u} \in \Omega_{k-1}} e^{-\frac{1}{\beta_k^2} \|\mathbf{u} - \mathbf{z}\|_2^2} \zeta_k(\mathbf{u}). \tag{6}$$

output $\xi_k : \Omega'_k \to \mathbb{R}^{p_k}$ (new map);

Figure 2: Algorithm 1: CKN-learning the parameters of the kth layer; Algorithm 2: CKN-computing the kth map from the (k-1)th one.

we show that (a) and (b) also hold at layer k. This is sufficient for our purpose since we have previously assumed (b) for the zeroth layer. One can then may be use stochastic gradient descent (SGD) to optimise (4), since a potentially infinite amount of training data is available.

2.3 Learning Hierarchies of Subspaces with CKNs

Here we present the principles of convolutional kernel networks and a few generalizations and improvements of the original approach [8] described above in Section 2.2 and Section 2.1 . The model builds upon four ideas that are detailed below and that are illustrated in Figure 3 for a model with a single layer. [1]

2.3.1 Utilizing the Kernel Trick to Create Local Image Representations:

Given a set $\mathcal{X}$, a positive definite kernel $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ implicitly defines a Hilbert space $\mathcal{H}$, called reproducing kernel Hilbert space (RKHS), along with a mapping $\varphi : \mathcal{X} \to \mathcal{H}$. This embedding is such that the kernel value $K(\mathbf{x}, \mathbf{x}')$ corresponds to the inner product $\langle \varphi(\mathbf{x}), \varphi(\mathbf{x}') \rangle_{\mathcal{H}}$. Called "kernel trick", this approach can be used to obtain nonlinear representations of local image patches [13] [8]. More precisely, consider an image $I_0 : \Omega_0 \to \mathbb{R}^{p_0}$,

where p_0 is the number of channels, e.g., $p_0 = 3$ for RGB, and $\Omega_0 \subset [0,1]^2$ is a set of pixel coordinates, typically a two-dimensional grid. Given two image patches $\mathbf{x}, \mathbf{x}'$ of size $e_0 \times e_0$, represented as vectors in $\mathbb{R}^{p_0 e_0^2}$, we define a kernel K_1 as

$$K_1(\mathbf{x}, \mathbf{x}') = \|\mathbf{x}\| \|\mathbf{x}'\| \kappa_1\left(\left\langle \frac{\mathbf{x}}{|\mathbf{x}|}, \frac{\mathbf{x}'}{\|\mathbf{x}'\|} \right\rangle\right) \text{ if } \mathbf{x}, \mathbf{x}' \neq 0 \text{ and 0 otherwise,} \tag{5}$$

Specifically, κ_1 should be smooth and its Taylor expansion have non-negative coefficients to ensure positive definiteness [2]. Here we have used the Gaussian (RBF) kernels (1): given two vectors $\mathbf{y}, \mathbf{y}'$ with unit ℓ_2-norm, choose for instance

$$\kappa_1(\langle \mathbf{y}, \mathbf{y}' \rangle) = e^{\alpha_1(\langle \mathbf{y}, \mathbf{y}' \rangle - 1)} = e^{-\frac{\alpha_1}{2}\|\mathbf{y} - \mathbf{y}'\|_2^2}. \tag{6}$$

Then, we have implicitly defined the RKHS $\mathcal{H}_1$ associated to K_1 and a mapping $\varphi_1 : \mathbb{R}^{p0 e_0^2} \to \mathcal{H}_1$.

2.3.2 Project onto a finite-dimensional subspace of the RKHS with convolution layers

After applying the above transformation ($I_0 : \Omega_0 \to \mathbb{R}^{p_0}$) to all overlapping patches of the input image I_0, a spatial map $M_1 : \Omega_0 \to \mathbb{R}^{p_1}$ may be obtained such that for all z in $\Omega_0, M_1(z) = \psi_1(\mathbf{x}_z)$, where $\mathbf{x}_z$ is the $e_0 \times e_0$ patch from I_0 centered at pixel location z. With the approximation scheme[8] of Section 2.2, M_1 can be interpreted as the output feature map of a one-layer convolutional neural network (2). A conceptual drawback of [8] is that data points $\varphi_1(\mathbf{x}_1), \varphi_1(\mathbf{x}_2), \ldots$ are approximated by vectors that do not live in the RKHS $\mathcal{H}_1$. This issue can be solved by using variants of the Nyström method [11], which consists of projecting data onto a subspace of $\mathcal{H}_1$ with finite dimension p_1. For this task, we have adapted the approach of [9]:

— build a database of n patches $\mathbf{x}_1, \ldots, \mathbf{x}_n$ randomly extracted from various images and normalized to have unit ℓ_2-norm

— perform a spherical K-means algorithm to obtain p_1 centroids $\mathbf{z}_1, \ldots, \mathbf{z}_{p_1}$ with unit ℓ_2-norm

— Then, a new patch $\mathbf{x}$ is approximated by its projection onto the p_1-dimensional subspace $\mathcal{F}_1 = \mathrm{Span}(\varphi(\mathbf{z}_1), \ldots, \varphi(\mathbf{z}_{p_1}))$.

The projection of $\varphi_1(\mathbf{x})$ onto $\mathcal{F}_1$ admits a natural parametrization $\psi_1(\mathbf{x})$ in $\mathbb{R}^{p_1}$. The explicit formula is classical [11] [9], leading to

$$\psi_1(\mathbf{x}) := \|\mathbf{x}\| \kappa_1\left(\mathbf{Z}^\top \mathbf{Z}\right)^{-1/2} \kappa_1\left(\mathbf{Z}^\top \frac{\mathbf{x}}{\|\mathbf{x}\|}\right) \text{ if } \mathbf{x} \neq 0 \text{ and 0 otherwise,} \tag{7}$$

where we have introduced the matrix $\mathbf{Z} = [\mathbf{z}_1, \ldots, \mathbf{z}_{p_1}]$, and, the function κ_1 is applied pointwise to its arguments. In other words, we obtain a particular convolutional neural network, with non-standard parametrization. Hence, we observe that learning requires only performing a K-means algorithm and computing the inverse square-root matrix.

2.3.3 Linear Pooling for Invariance:

Linear pooling in $\mathcal{F}_1$ is equivalent to linear pooling on the finite-dimensional map M_1. The previous steps transform an image $I_0 : \Omega_0 \to \mathbb{R}^{p_0}$ into a map $M_1 : \Omega_0 \to \mathbb{R}^{p_1}$, where each vector $M_1(z)$ in $\mathbb{R}^{p_1}$ encodes a point in $\mathcal{F}_1$ representing information of a local image neighborhood centered at location z. Then, convolutional kernel networks involve a pooling step to gain invariance to small shifts, leading to another finite-dimensional map $I_1 : \Omega_1 \to \mathbb{R}^{p_1}$ with smaller resolution:

$$I_1(z) = \sum_{x' \in \Omega_0} M_1(z')\, e^{-\beta_1 \left\| z' - z \right\|_2^2}. \tag{8}$$

The Gaussian weights act as an anti-aliasing filter for downsampling the map M_1 and β_1 is set according to the desired subsampling factor [8], which does not need to be integer. Then, every point $I_1(z)$ in $\mathbb{R}^{p_1}$ may be interpreted as a linear combination of points in $\mathcal{F}_1$, which is itself in $\mathcal{F}_1$ since $\mathcal{F}_1$ is a linear subspace.

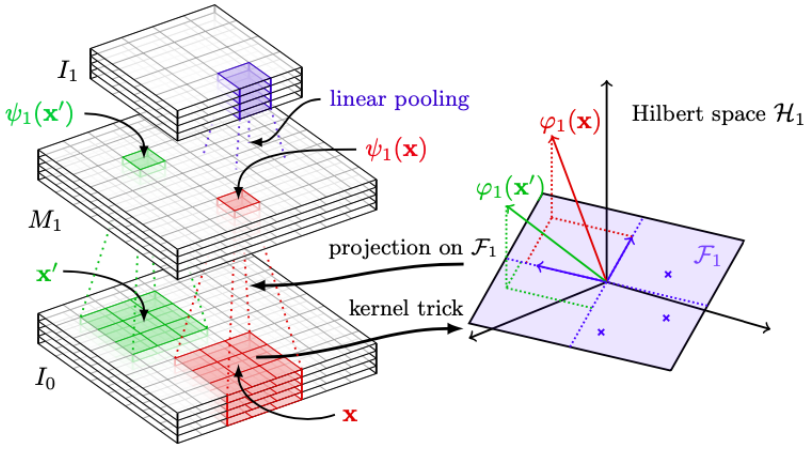

Figure 3: Our variant of convolutional kernel networks, illustrated between layers 0 and 1. Local patches (receptive fields) are mapped to the RKHS H_1 via the kernel trick and then projected to the finite-dimensional subspace. With no supervision, optimizing F_1 consists of minimizing projection residuals. With supervision, the subspace is optimized via back-propagation. Going from layer k to layer $k+1$ is achieved by stacking the model described here and shifting indices.[1]

2.3.4 Multilayer Representation

We build a multilayer image representation by stacking and composing kernels. By following the first three principles described above and, by going up in the hierarchy, the vectors $I_k(z)$ in R^{p_k} represent larger and larger image neighborhoods (aka. receptive fields) with more invariance gained by the pooling layers, akin to classical convolutional neural networks.

The multilayer scheme produces a sequence of maps $(I_k)_{k \geq 0}$, where each vector $I_k(z)$ encodes a point-say $f_k(z)$-in the linear subspace $\mathcal{F}_k$ of $\mathcal{H}_k$. Thus, we implicitly represent an

image at layer k as a spatial map $f_k : \Omega_k \to \mathcal{H}_k$ such that $\langle I_k(z), I_k'(z') \rangle = \langle f_k(z), f_k'(z') \rangle \mathcal{H}_k$ for all z, z'. As mentioned previously, the mapping to the RKHS is a key to the multilayer construction. Given I_k, larger image neighborhoods are represented by patches of size $e_k \times e_k$ that can be mapped to a point in the Cartesian product space $\mathcal{H}_k^{e_k \times e_k}$ endowed with its natural inner-product; finally, the kernel K_{k+1} defined on these patches can be seen as a kernel on larger image neighborhoods than K_k.

2.4 Backpropagation Rules for Convolutional Kernel Networks

Now, after training a convolutional kernel network with k layers, the resultant objective function that we try to minimize by combining the classical empirical risk minimization formulation and (7) is,

$$\min_{\mathbf{W} \in \mathbb{R}^{\eta^\times \times \square_k|}} \frac{1}{n} \sum_{i=1}^{n} L\left(y_i, \left\langle \mathbf{W}, I_k^i \right\rangle\right) + \frac{\lambda}{2} \|\mathbf{W}\|_{\mathrm{p}}^2, \tag{9}$$

- Where we are given a training set of images $I_0^1, I_0^2, \ldots, I_0^6$ with respective scalar labels $y_1, \ldots, y_n$ living either in $\{-1; +1\}$ for binary classification and $\mathbb{R}$ for regression (extensions to multiclass classification and multivariate regression can be done).
- We also assume that we are given a smooth convex loss function $L : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ that measures the fit of a prediction to the true label y.
- The parameter λ controls the smoothness of the prediction function f with respect to the geometry induced by the kernel, hence regularizing and reducing overfitting [2]
- $\|\|_F$ is the Frobenius norm that extends the Euclidean norm to matrices, and, the maps I_k^i are seen as matrices in $\mathbb{R}^{p_k \times |0_k|}$, where I_k^is are the $\mathbf{k-th}$ finite-dimensional feature maps of I_0^i, which have been defined in the previous section.

Then, the supervised convolutional kernel network formulation consists of jointly minimizing (9) with respect to $\mathbf{W}$ in $\mathbb{R}^{p_k \times |\Omega_k|}$ and with respect to the set of filters $\mathbf{Z}_1, \ldots, \mathbf{Z}_k$ from (7), whose columns are constrained to be on the Euclidean sphere. Now, we compute the derivative with respect to the filters $\mathbf{Z}_1, \ldots, \mathbf{Z}_\mathrm{k}$. Since we consider a smooth loss function L (squared hinge loss in this case), optimizing (9) with respect to $\mathbf{W}$ can be achieved with any gradient-based method.

Optimizing the filters $\mathbf{Z}_j, j = 1, \ldots, k$ is more challenging due to the absence of convexity. However, despite this non-convexity, the objective function is still differentiable, and there's potential to discover a "favorable" stationary point by employing traditional stochastic optimization methods that have proven effective in training deep networks. To achieve this, we must calculate the gradient using the chain rule, commonly referred to as "backpropagation" [12]. Having established these backpropagation rules, we then apply a stochastic gradient descent method (SGD) for optimization[1]

2.5 Optimization

In the following sections, we will introduce several strategies aimed at enhancing the optimization process in our specific context.

— **Hybrid Convex/Non-Convex Optimization:** To speed up training, they use techniques originally designed for convex optimization problems. These methods work well for problems with large but finite datasets, providing faster convergence compared to standard stochastic gradient descent. They alternate between two steps: first optimizing certain network parameters with a convex method and then updating another set using stochastic gradient descent. This hybrid approach helps when dealing with non-convex problems.

— **Preconditioning on the Sphere:** Since the network's filters operate on spherical data, they constrain these filters to have a unit L2-norm. To accelerate convergence, they use a technique called preconditioning. This involves changing the optimization variables to reduce correlations in gradient updates. They apply this technique while taking into account the constraint of unit L2-norm, which involves some complex math but helps speed up learning.

— **Automatic Learning Rate Tuning:** Choosing the right learning rate is crucial in training neural networks. They use a simple heuristic: starting with a reasonably large learning rate and periodically checking the training loss. If the loss increases, they decrease the learning rate by half, helping the network converge more effectively.

— **Active-Set Heuristic** In classification tasks, some data samples have little impact on the training process. They introduce an "active-set" heuristic, initially considering all samples and removing those that contribute negligibly to the gradient. In subsequent optimization steps, only active samples are used, and a portion of inactive ones is randomly reintroduced after each training epoch.

3 Experiment: Image classification

The MNIST dataset [2] consists of 60 000 images of handwritten digits for training and 10 000 for testing. We evaluate the performance of a 5-layer network, designed with few hyper-parameters: for each layer, we learn 128 filters and choose the RBF kernels κ_j defined in (6) with initial parameters $\alpha_j = 0.001$. Layers 1,3,5 use 3×3 patches and layers 2, 4 use simply 1×1 patches. Also, they all use a subsampling pooling factor of $\sqrt{2}$ (Section 2.3.3). The parameters α_j are updated in the same way as the filters during training. We use the squared hinge loss in a one vs all setting to perform multi-class classification (with shared filters Z between classes). We use the optimization heuristics from the previous section, notably the automatic learning rate scheme, and a gradient momentum with parameter 0.9, following [19]. The λ is chosen near the canonical parameter $\lambda = 1/\mathrm{n}$ and the number of epochs is at most 105. The initial learning rate is 10 with a minibatch size of 128.
Due to lack of space locally, we ran it on kaggle. The following was the result,

accuracy	training time
0.9926	137.57194194393335

We attain a test error rate of approximately 0.007%, placing our approach in a fairly competitive position and on par with the deeply supervised networks [20] or the network-in-network architecture [21].

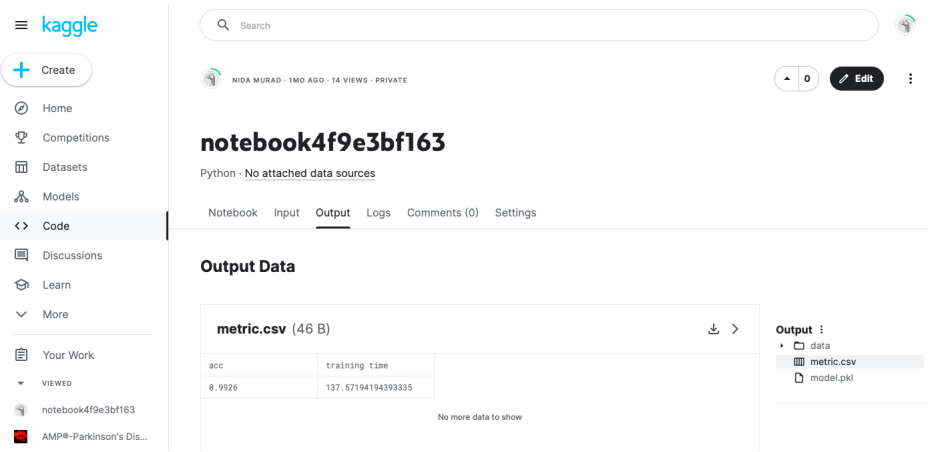

Figure 4: screenshot of the kaggle result

References

[1] arXiv:1605.06265 End-to-End Kernel Learning with Supervised Convolutional Kernel Networks

[2] Y.LeCun,L.Bottou,Y.Bengio,andP.Haffner.Gradient-basedlearningappliedtodocumentrecognition. P. IEEE, 86(11):2278–2324, 1998.

[3] M. Paulin, M. Douze, Z. Harchaoui, J. Mairal, F. Perronin, and C. Schmid. Local convolutional features with unsupervised training for image retrieval. In Proc. ICCV, 2015.¡br¿

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In Adv. NIPS, 2012.

[5] C.-Y. Lee, P. W. Gallagher, and Z. Tu. Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In Proc. AISTATS, 2016.

[6] C.-Y. Lee, S. Xie, P. W. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. In Proc. AISTATS, 2015.

[7] M. Lin, Q. Chen, and S. Yan. Network in network. In Proc. ICLR, 2013.

[8] J. Mairal, P. Koniusz, Z. Harchaoui, and C. Schmid. Convolutional kernel networks. In Adv. NIPS, 2014.¡br¿

[9] K. Zhang, I. W. Tsang, and J. T. Kwok. Improved Nyström low-rank approximation and error analysis. In Proc. ICML, 2008.

[10] https://people.eecs.berkeley.edu/ bartlett/courses/281b-sp08/7.pdf

[11] C. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In Adv. NIPS, 2001.

[12] Y. Le Cun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In Neural Networks, Tricks of the Trade, Lecture Notes in Computer Science LNCS 1524. 1998.

[13] L. Bo, K. Lai, X. Ren, and D. Fox. Object recognition with hierarchical kernel descriptors. In CVPR, 2011.

[14] https://www.mit.edu/ 9.520/scribe-notes/class03_gdurett.pdf

[15] https://medium.com/deep-math-machine-learning-ai

[16] Zhang, Haizhang; Xu, Yuesheng; Zhang, Qinghui (2012). "Refinement of Operator-valued Reproducing Kernels" (PDF). Journal of Machine Learning Research

[17] J. Shawe-Taylor and N. Cristianini. Kernel methods for pattern analysis. 2004.

[18] L. Bottou, O. Chapelle, D. DeCoste, and J. Weston. Large-Scale Kernel Machines (Neural Information Processing). The MIT Press, 2007.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural

[20] C.-Y. Lee, S. Xie, P. W. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. In Proc. AISTATS, 2015. networks. In Adv. NIPS, 2012.

[21] M. Lin, Q. Chen, and S. Yan. Network in network. In Proc. ICLR, 2013.

[22] M. Ranzato, F.-J. Huang, Y-L. Boureau, and Y. LeCun. Unsupervised learning of invariant feature hierar- chies with applications to object recognition. In Proc. CVPR, 2007.

[23] Maraaba, Luqman and Milhem, Abdulaziz and Nemer, Ibrahim and Al-Duwaish, Hussain and Abido, Mohammed, title = Convolutional Neural Network Based Inter-Turn Fault Diagnosis in LSPMSMs, doi = 10.1109/ACCESS.2020.2991137

Appendix A RKHS: in depth

A.1 Functional Analysis Background

Definition A.1. A function space $\mathcal{F}$ is a space whose elements are functions,
e.g. $f : \mathbb{R}^d \to \mathbb{R}$.

Definition A.2. $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel if

— k is symmetric: $k(x, y) = k(y, x)$.

— k is positive semi-definite, i.e., $\forall x_1, x_2, \ldots, x_n \in \mathcal{X}$, the "Gram Matrix" K defined by $K_{ij} = k(x_i, x_j)$ is positive semi-definite. (A matrix $M \in \mathbb{R}^{n \times n}$ is positive semi-definite if $\forall a \in \mathbb{R}^n, a'Ma \geq 0$.)

Definition A.3. An inner product is a function $\langle \cdot, \cdot \rangle : \mathcal{F} \times \mathcal{F} \to \mathbb{R}$ that satisfies the following properties for every $f, g \in \mathcal{F}$ and $\alpha \in \mathbb{R}$:

— Symmetric: $\langle f, g \rangle = \langle g, f \rangle$

— Linear: $\langle r_1 f_1 + r_2 f_2, g \rangle = r_1 \langle f_1, g \rangle + r_2 \langle f_2, g \rangle$

— Positive-definite: $\langle f, f \rangle \geq 0$ for all $f \in \mathcal{F}$ and $\langle f, f \rangle = 0$ iff $f = 0$.

Definition A.4. A norm is a non negative function $\| \cdot \| : \mathcal{F} \to \mathbb{R}$ such that for all $f, g \in \mathcal{F}$ and $\alpha \in \mathbb{R}$

— $\|f\| \geq 0$ and $\|f\| = 0$ iff $f = 0$;

— $\|f + g\| \leq \|f\| + \|g\|$;

— $\|\alpha f\| = |\alpha| \|f\|$.

A norm can be defined via an inner product, as $\|f\| = \sqrt{\langle f, f \rangle}$

Now we can define the notion of a Hilbert space.

Definition A.5. A Hilbert Space is an inner product space that is complete and separable with respect to the norm defined by the inner product

Some examples of Hilbert spaces include:
1. The vector space $\mathbb{R}^n$ with $\langle a, b \rangle = a'b$, the vector dot product of a and b.
2. The space l_2 of square summable sequences, with inner product $\langle x, y \rangle = \sum_{i=1}^{\infty} x_i y_i$
3. The space L_2 of square integrable functions (i.e., $\int_s f(x)^2 dx < \infty$), with inner product $\langle f, g \rangle = \int_s f(x) g(x) dx$

A norm in $\mathcal{H}$ can be naturally defined from the given inner product, as $\| \cdot \| = \sqrt{\langle \cdot, \cdot \rangle}$. Furthermore, we always assume that $\mathcal{H}$ is separable (contains a countable dense subset) so that $\mathcal{H}$ has a countable orthonormal basis.

Essentially, a Hilbert space lets us apply concepts from finite-dimensional linear algebra to infinite-dimensional spaces of functions. In particular, the fact that a Hilbert space is complete will guarantee the convergence of certain algorithms. More importantly, the presence of an inner product allows us to make use of orthogonality and projections, which will later become important.

A.2 Proof of Theorem 2

RKHS defines a corresponding reproducing kernel. Conversely, a reproducing kernel defines a unique RKHS.

Proof. : To prove the first statement, we must prove that the reproducing kernel $K(t,x) = \langle K_t, K_x \rangle_{\mathcal{H}}$ is symmetric and positive-definite. Symmetry follows from the symmetry property of inner products:

$$\langle K_{\mathrm{t}}, K_x \rangle_{\mathcal{H}} = \langle K_x, K_t \rangle_{\mathcal{H}} .$$

K is positive-definite because

$$\sum_{i,j=1}^{n} c_i c_j K\left(t_i, t_j\right) = \sum_{i,j=1}^{n} c_i c_j \left\langle K_{t_i}, K_{t_j} \right\rangle_{\mathcal{H}} = \left\| \sum_{j=1}^{n} c_j K_{t_j} \right\|_{\mathcal{H}}^2 \geq 0.$$

To prove the second statement, given K one can construct the RKHS $\mathcal{H}$ as the completion of the space of functions spanned by the set $\{K_z \mid x \in X\}$ with an inner product defined as follows: given two functions f and g in $\operatorname{span}\{K_x \mid x \in X\}$

$$f(x) = \sum_{i=1}^{\star} \alpha_i K_{x_i}(x)$$
$$g(x) = \sum_{i=1}^{s} \beta_i K_{x_i'}(x)$$

we define their inner product to be

$$\langle f, g \rangle_{\mathcal{H}} = \sum_{i=1}^{s} \sum_{j=1}^{s'} \alpha_i \beta_j K\left(x_i, x_j'\right).$$

(This is only a sketch of the proof.) Now we have a more concrete concept of what an RKHS is and how we might create such spaces for ourselves. If we can succeed at writing down a reproducing kernel, we know that there exists an associated RKHS, and we need not concern ourselves with the particulars of the boundedness criterion.[14] □

A.3 Connection between RKHSs and the ReLU function

The ReLU function is commonly defined as $f(x) = \max\{0, x\}$ and is a mainstay in the architecture of neural networks where it is used as an activation function. One can construct a ReLU-like nonlinear function using the theory of reproducing kernel Hilbert spaces. Below, we derive this construction and show how it implies the representation power of neural networks with ReLU activations.

We will work with the Hilbert space $\mathcal{H} = L_2^1(0)[0, \infty)$ of absolutely continuous functions with $f(0) = 0$ and square integrable (i.e. L_2) derivative. It has the inner product

$$\langle f, g \rangle_{\mathcal{H}} = \int_0^{\infty} f'(x) g'(x) dx.$$

To construct the reproducing kernel it suffices to consider a dense subspace, so let $f \in C^1[0,\infty)$ and $f(0) = 0$. The Fundamental Theorem of Calculus then gives

$$f(y) = \int_0^y f'(x)dx = \int_0^\infty G(x,y)f'(x)dx = \langle K_y, f\rangle$$

where

$$G(x,y) = \begin{cases} 1, & x < y \\ 0, & \text{otherwise} \end{cases}$$

and $K_y'(x) = G(x,y), K_y(0) = 0$ i.e.

$$K(x,y) = K_y(x) = \int_0^x G(z,y)dz = \begin{cases} x, & 0 \le x < y \\ y, & \text{otherwise.} \end{cases} = \min(x,y)$$

This implies $K_y = K(\cdot, y)$ reproduces f. Moreover the minimum function on $X \times X = [0,\infty) \times [0,\infty)$ has the following representations with the ReLu function:

$$\min(x,y) = x - \text{ReLU}(x-y) = y - \text{ReLU}(y-x).$$

Using this formulation, we can apply the representer theorem to the RKHS, letting one prove the optimality of using ReLU activations in neural network settings [16]

Appendix B Example/Run through of a CNN

Consider any image, as you know every image represent some pixels in simple terms. Filters are tensor which keeps track of spatial information and learns to extract features like edge detection, smooth curve, etc of objects in something called a **convolutional layer**. The major part is to detect edges in the images and these are detected by the filters. It helps to filter out unwanted information to amplify images. There are high-pass filters where the changes occur in intensity very quickly like from black to white pixel and vice-versa.

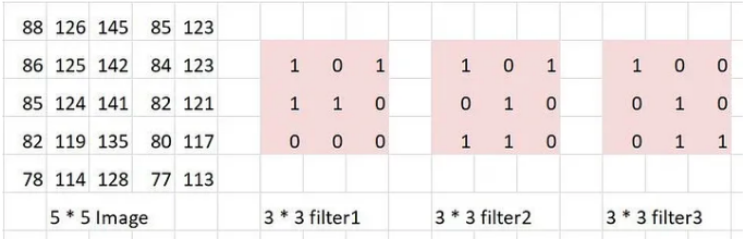

Figure 5: Filters[15]

Let's consider an image of size 5X5 size and 3 filters (since each filter will be used for each color channel: RGB) of 3X3 size. For simplicity, we took 0 and 1 for filters, usually, these are continuous values. The filter convolute with the image to detect patterns and features. Then the convolution of 5 x 5 image matrix multiplies with 3 x 3 filter matrix which is called **"Feature Map"**. We apply the dot product to the scaler value and then move the filter

by the stride over the entire image. Sometimes filter does not fit perfectly fit the input image. Then there is a need to pad the image with zeros as shown below. This is called padding.

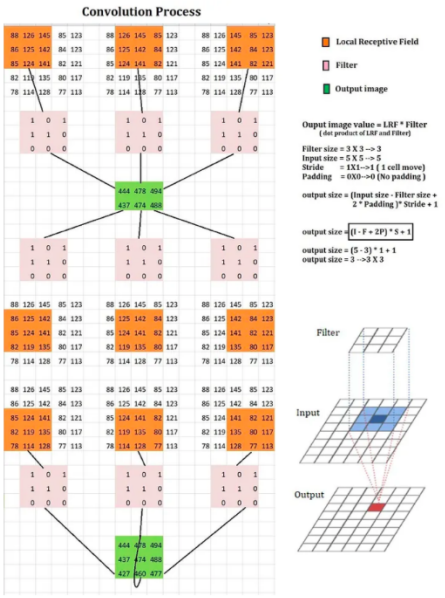

Figure 6: Convolution Process [15]

Next, we need to reduce the size of images, if they are too large. **Pooling layers** section would reduce the number of parameters when the images are too large. As shown in the above image, the padding is applied so that the filter perfectly fits the given image. Adding pooling layer then decrease the size of the image and hence decrease the complexity and computations.

Next Step, is **Normalization**. Usually, an activation function ReLu is used. ReLU stands for Rectified Linear Unit for a non-linear operation. The output is f(x) = max(0,x). The purpose of ReLu is to add non-linearity to the convolutional network. In usual cases, the real-world data want our network to learn non-linear values. A rectified linear unit has output 0 if the input is less than 0, and raw output otherwise. That is, if the input is greater than 0, the output is equal to the input. Here we are assuming that we have negative values since dealing with the real-world data. In case, if there is no negative value, you can skip this part.

$$RELU(x) = \begin{cases} 0 \text{ if } x < 0 & -478 & 494 & = & 0 & 494 \\ x \text{ if } x >= 0 & 460 & -477 & & 460 & 0 \end{cases}$$

The final step is to flatten our matrix and feed the values to **Fully connected layer**.

Next, we need to train the model in the same way, we train other neural networks. Using the certain number of epochs and then backpropagate to update weights and calculate the loss.