RACKNet : Robust Allocation of Convolutional Kernels in Neural Networks for Image Classification[*]

Yash Garg
ygarg@asu.edu
Arizona State University
Tempe, Arizona

K. Selçuk Candan
candan@asu.edu
Arizona State University
Tempe, Arizona

ABSTRACT

Despite their impressive success when these hyper-parameters are suitably fine-tuned, the design of good network architectures remains an art-form rather than a science: while various search techniques, such as grid-search, have been proposed to find effective hyper-parameter configurations, often these parameters are hand-crafted (or the bounds of the search space are provided by a user). In this paper, we argue, and experimentally show, that we can minimize the need for hand-crafting, by relying on the dataset itself. In particular, we show that the dimensions, distributions, and complexities of localized features extracted from the data can inform the structure of the neural networks and help better allocate limited resources (such as kernels) to the various layers of the network. To achieve this, we first present several hypotheses that link the properties of the localized image features to the CNN and RCNN architectures and then, relying on these hypotheses, present a *RACKNet* framework which aims to learn multiple hyper-parameters by extracting information encoded in the input datasets. Experimental evaluations of *RACKNet* against major benchmark datasets, such as MNIST, SVHN, CIFAR10, COIL20 and ImageNet, show that *RACKNet* provides significant improvements in the network design and robustness to change in the network.

CCS CONCEPTS

• **Information systems → Content analysis & feature selection**; **Image search**; • **Computing methods → Neural networks**;

KEYWORDS

Convolutional neural networks, recurrent neural networks, deep learning, meta-learning, hyper-parameter optimization

ACM Reference Format:
Yash Garg and K. Selçuk Candan. 2019. *RACKNet* : Robust Allocation of Convolutional Kernels in Neural Networks for Image Classification. In *International Conference on Multimedia Retrieval (ICMR '19), June 10–13, 2019, Ottawa, ON, Canada.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3323873.3325057

[*]Partially funded by NSF #1610282, #1633381, #1629888, #1827757, and "FourCModelling": EUH2020 Grant Agreement No. 690817.

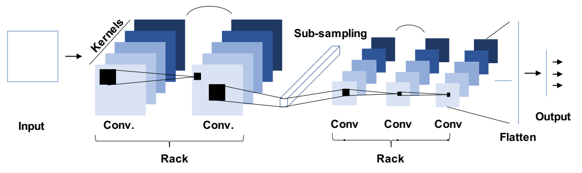


Figure 1: Outline of a CNN (in this paper we term the set of convolutional layers between pooling layers as "Racks")

1 INTRODUCTION

Deep neural networks, including convolutional neural networks (CNNs, Figure 1) have seen successful application in face recognition [26] as early as 1997, and more recently in various multimedia domains, such as time series analysis [45, 49], speech recognition [16], object recognition [29, 36, 38], and video classification [22, 41].

More recently, CNNs' successful application in a variety of multimedia domains has lead to a shift away from feature driven algorithms, such as SURF [3], HOG [11], and SIFT [31], into the design of well-crafted CNN architectures for specific datasets and application domains. Unfortunately, deep neural networks, including CNNs, tend to be complex with a large number of hyper-parameters. As [4] points out, the ultimate objective of finding a high performing architecture configuration to minimize the expected loss, $L(x; l_f)$, over i.i.d x samples of the learning function, l_f, representing the network, NN. Often the success of the learning function, l_f, depends on the choice of *hyper-parameters*, λ. Therefore, L is a function of hyper-parameters as well, where $L(x, \lambda; l_f)$. Despite their impressive success when these hyper-parameters are suitably fine-tuned, design of good network architectures still remains an art-form rather than a science, while various techniques, such as random search [4, 5], grid-search [25], and others [20, 21, 37, 43, 44, 50], have been proposed to help locate an effective (optimal or close-to-optimal) hyper-parameter configuration, λ_o. Due to high-dimensionality of the hyper-parameter space and the complexity and non-linearity of the CNN architectures, searching for an effective hyper-parameter configuration, λ_o, is a computationally-expensive process, which has led to an interest in specialized and targeted approaches [20, 21, 32, 44, 46] that introduce refinements *on-top* of existing network configurations. Yet, today these parameters need to be hand-crafted (or at least the bounds of the search space are provided by a user).

As further discussed in Section 2, a common shortcoming of the existing approaches to hyper-parameter search is that they "*work on the data, not with the data*", i.e. they aim to find a hyper-parameter configuration that minimizes L, but in the process they ignore the available data and the key insights that the data can provide in honing in on effective configurations of hyper-parameters. In

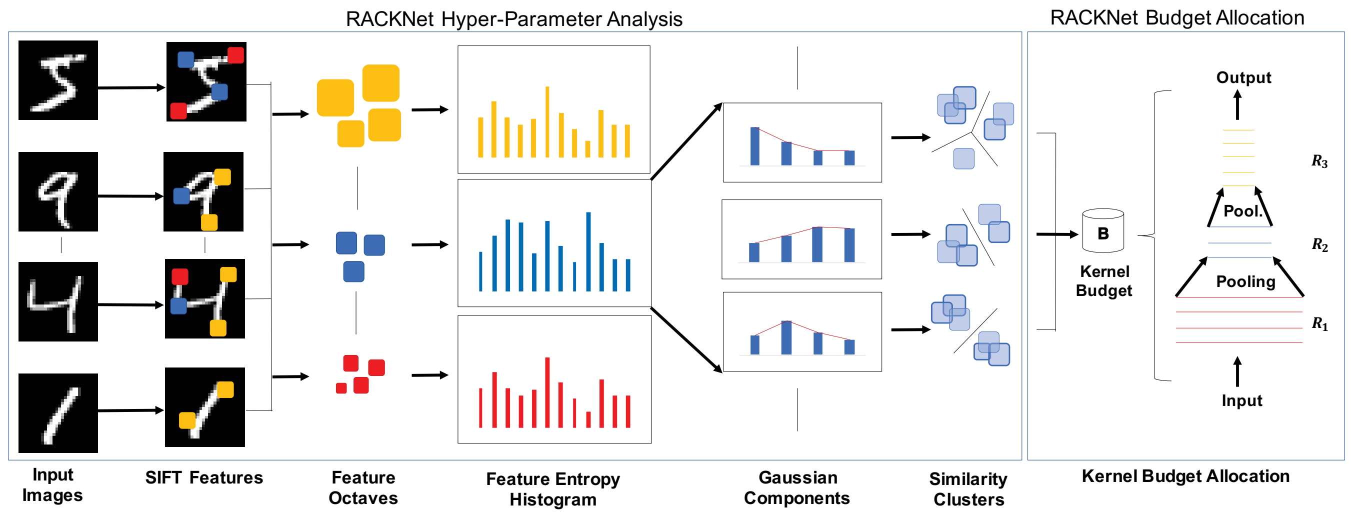


Figure 2: Overview of the *RACKNet* framework: sizes, complexities, and distributions of the local features extracted from the image dataset during pre-processing are used to inform the structure of the CNN and allocate convolution kernels within the neural network architecture

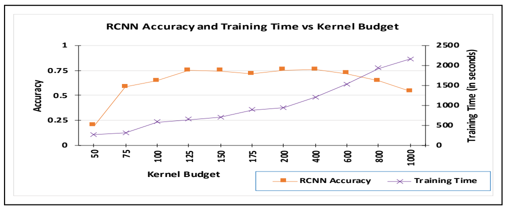


Figure 3: Impact of kernel budget allocation on accuracy and training time (more details in Section 4)

contrast, we argue, and experimentally show, that we can minimize the need for hand crafting of the search space, by relying on a pre-training analysis of the data itself. To this end, we present a <u>*RACKNet* framework</u> which aims to learn key hyper-parameters by extracting information encoded in the input datasets (Figure 2).

1.1 Key Contributions: Kernel Budget Allocation through Local Feature Analysis

RACKNet focuses on hyper-parameters that impact kernel budget allocation, which (as we see in Figure 3) can have significant impacts on model accuracies and training times. In particular, we show that the sizes, complexities, and distributions of localized features, such as SIFT features [31], extracted from the dataset can provide insights that can inform the structure of the neural networks and help better allocate limited resources (such as kernels) within the network.

> In other words, we argue that, even when the SIFT features may not be sufficiently informative to achieve high accuracies in media analysis tasks alone, they can provide reliable insights that can inform the design of effective CNN and RCNN architectures.

More specifically, unlike purely feature-based approaches, which leverage features extracted from the data to implement the analysis task, *RACKNet* uses these features only to inform the structure of the CNN (or RCNN) to be used for analysis. To achieve this, we first present four key observations that link the properties of the localized features to the CNN and RCNN architectures:

- *Observation 1:* Sizes of the localized features in the dataset can inform the kernel sizes and the numbers of sub-sampling racks of the CNN.
- *Observation 2*: Complexities of the features (measured through the entropies of their descriptors) at different scales can be used to discover the number of layers per rack.
- *Observation 3*: Overall complexities of convolutional layers, defined through corresponding image features, can be used to inform the arrangement of layers within a rack.
- *Observation 4*: Distribution of the features for a given complexity component can be used to discover the number of kernels per convolution layer.

It is important to note that, while *RACKNet* uses both localized and CNN features, they are used for entirely different purposes:

- cheaper-to-obtain, but rough, localized features are used to bootstrap the hyper-parameter of the CNN; whereas
- expensive-to-obtain, but finer-grain, CNN features are, then, used to obtain the classifier.

RACKNet is designed as an unsupervised general purpose single-shot architecture search framework that takes pre-computed localized image features for hyper-parameter extraction (as described in Section 3). As we argue, and experimentally show, these feature can provide insights on the dataset that can inform the architecture of the network. To present *RACKNet*, we use common benchmark datasets, such as MNIST, SVHN, CIFAR10, COIL20, and ImageNet, and a commonly used localized image feature, known as SIFT [31]. Experiments on these datasets show that *RACKNet* indeed provides significant improvements in the network performance.

Symbol	**Convolutional Neural Network (CNN)**	Symbol	**Scale Invariant Feature Transform (SIFT)**
r	# of racks of conv. layers	o	# of Octaves
c_i	# of conv. layers in the given rack, R_i	g_i	# of Gaussian comp. in entropy histogram an octave, o_i
$k_{i,j}$	# of kernels in a given conv. layer, $C_{i,j}$	$n_{i,j}$	# of feature-similarity clusters in a given Gauss. comp., $g_{i,j}$
$e_{i,j}$	Kernel size for a given conv. layer, $C_{i,j}$	$\sigma_{i,j}$	Feature scope in a given Gauss. comp., $g_{i,j}$

Table 1: Notations used in this paper

2 RELATED WORK

Successful application of convolutional neural networks (CNN) dates back to the 90s where it was applied to face recognition [26]. More recently CNNs have shown promising results in various multimedia applications such as image enhancement [8], speech recognition [16], video classification [22, 41], object recognition [29, 36, 38], and time series analysis [45, 49] have generated significant interest in CNN deisgn and configuration.

[4, 5, 25] proposed hyper-parameter configuration search techniques that evaluate performances of different configurations on training data. [4], for example, implemented a random search over a bounded 32-dimensional parameter space. [50] proposed a reinforcement learning based technique to incrementally improve hyper-parameter configurations. [37] proposed an evolutionary search algorithm that generates a large population of CNNs to converge on a good configuration through mutations. [19] designed a greedy search through multi-attribute learning.

The main difficulty with these search-based techniques is that the process can be very costly as the possible parameter space can be very large. The prohibitive cost of search-based approaches led to approaches that target specific network components; these include maxout [20], batch normalization [21], rectified-lu [14, 24, 32], and dropout [44], techniques: [20] proposed a *maxout* strategy that introduces a max-pooling layer that provides spatial invariance to the network. [21] proposed a batch normalization layer aiming to minimize the covariate shift in the network, allowing for higher learning rates. The *dropout* layer proposed in [44] aims to prevent overfitting of the network by randomly engaging and disengaging the convolutional kernels during training. [24] proposed Rectified Linear Units (ReLUs) aiming to minimize the loss of gradients during the training phase of the network. [32] proposed a non-linear rectifier that reduced the sparsity in the network. [14] proposed Probabilistic-ReLU, advancing the conventional ReLUs, to adaptively learn the ReLU parameters. Fusion networks have been proposed that combine shallow and deep features [7, 48].

In this paper, we note that a major weakness of the various techniques discussed above is that they ignore the input data itself. In particular, we argue that it may be possible to use the training dataset itself to help search for appropriate CNN hyper-parameter configurations. While this data-driven approach has not been used in CNN design, we see that it found successful use in several other application domains. For example, [47] demonstrated that spatio-temporal features extracted from time series data can reduce the computational cost of determining warping paths between multi-variate time series. [30, 40] leveraged spatio-temporal features extracted from multi-variate time series to improve on time series classification. [12, 13] leveraged metadata extracted from datasets to improve classification accuracies. In this paper, we note that, even when the local image features alone may not be able to provide high accuracies in media analysis, they can provide insights that can inform the design of effective CNNs.

3 ROBUST ALLOCATION OF CONVOLUTION KERNELS FOR CNNs (*RACKNET*)

As discussed above, while well designed CNNs can be very effective in image classification, their effectiveness is often hampered by the need for hand-crafted hyper-parameter design, especially for new datasets that have not been seen in the past. In this section, we propose the *RACKNet* framework that bootstraps the hyper-parameter configuration for CNNs based on a pre-analysis of the image dataset, $\mathcal{D}$, (Figure 2). In particular, in order to better distribute the given budget, B, of convolutional kernels across CNN layers, we propose to leverage SIFT features that capture local image patterns. *RACKNet* translates the sizes, complexities, and distribution of the high-level feature patterns in the given dataset to hyper-parameters of CNN structures that best utilize the given kernel budget.

3.1 Hyper-Parameters of a CNN

A convolutional neural network (CNN [27]) is a type of neural network that works by leveraging the local spatial arrangements by establishing local connections among small spatial regions across the adjacent layers. A CNN consists of several complementary components organized into layers (Figure 1):

- Each *convolution* layer links local-spatial data (i.e., pixels at the lowest layer) through a set of filters or *kernel*s that represent the local spatial features identified in the data.
- Since each convolution layer operates on the output of the previous convolution layer, higher layers correspond to increasingly complex features obtained by combining lower-complexity features.
- Since relevant features of interest can be of different sizes, *pooling/subsampling* layers are introduced among convolution layers: these pooling layers carry out down-sampling of the output of a convolution layer, thereby (given a fixed kernel size) effectively doubling the size of the feature extracted by the corresponding filter.

Therefore, intuitively, a CNN searches for increasingly complex local features that can be used for understanding (and interpreting) the content of a dataset. Given a dataset, and labeled data, this is achieved by a process known as *back-propagation* that uses gradient-search to identify the filters that best fit the labeled data.

Before the *back-propagation* can be implemented, however, one needs to pick hyper-parameters, such as the number of convolution layers, type of pooling operations, and number of kernels. In this paper, we follow the following convention (see also Table 1):

- We divide the sequence of convolution layers into r *racks*, each corresponding to a different feature size: to enable this, the two consecutive racks of convolution layers are separated by a pooling/subsampling layer.
- For each rack, R_i, of convolutional layers, we associate c_i many convolution layers, each resulting in more and more complex features of the size corresponding to the rack, R_i.

- Each convolution layer, $C_{i,j} \in R_i$, has $k_{i,j}$ kernels, each of size $e_{i,j} \times e_{i,j}$, where $e_{i,j}$ is the edge length of the kernel.

Therefore, the search for the hyper-parameters of the CNN can be posed as searching for the appropriate r, c, $k_{i,j}$, and $e_{i,j}$ parameter values that best describes the data.

3.2 Background: Local Image Features

As we have seen in the introduction, a common shortcoming of the existing approaches to hyper-parameter search is that they "*work on the data, not with the data*", i.e. they ignore the available data and key insights that the data can provide in honing in on effective configurations of hyper-parameters. In contrast, we argue that we can minimize the need for hand crafting of the search space, by relying on a pre-training analysis of the data itself. More specifically, we observe that if we could cheaply extract localized features of a given dataset (independent of the labeled data) – even if these features may not be sufficiently effective in achieving high accuracies in media analysis – the sizes, complexities, and distributions of these localized features can provide reliable insights that can inform the design of effective CNN architectures.

There are several localized feature extraction algorithms for images: these include SURF [3], HOG [11], and SIFT [31]. Scale Invariant Feature Transform [31], in particular, has been the *de facto* image representation strategy for content-based image retrieval as these features have shown robustness against rotation, scaling, and various distortions. SIFT[31] extracts stable and scale invariant patterns contained in a given image through a multi-step approach. SIFT supports multi-scale feature extraction: intuitively, SIFT features correspond to regions in a given image that are *different* from their neighborhoods, also in different image scales. Consequently, starting from the smallest feature size (provided by the user), features are organized into *octaves* corresponding to doubling of the feature diameter. In the rest of this section, we show that sizes and complexities of the localized features in a given dataset can be used to inform the design of CNNs that will operate on that dataset.

3.3 Observation #1: Number of Racks

As described earlier, we organize the layers of a CNN in the form of racks of convolution layers, where each rack corresponds to features of a different size: in particular, the down-sampling (through average pooling) operation between two consecutive racks reduces the number of rows and columns by half. Therefore, it is easy to see that there is a correspondence between the number of racks, r, of a CNN and the number of octaves, o, of localized features extracted from images in a given dataset.

Note that, while in general the number of octaves is a user provided input parameter to the SIFT algorithm, in general the number of features the algorithm identifies drops with increasing octaves. This is because, while an image may contain many small size features, the number of large yet stable features declines with the feature size. Therefore, our first hypothesis is that

$$r = o_{\mathcal{D}},$$

where $o_{\mathcal{D}}$ is the number of octaves where one is able to detect sufficiently many features in a given dataset, $\mathcal{D}$.

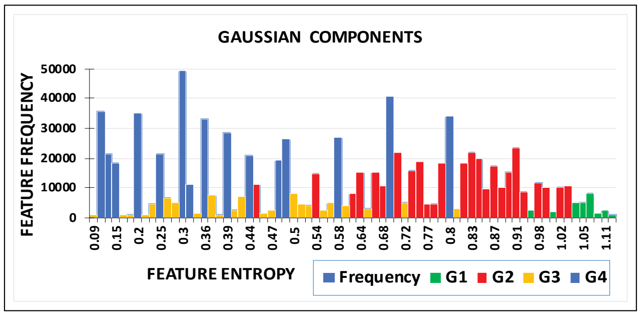


Figure 4: (Sample) Feature entropy distribution for the MNIST dataset [28]: here, different colors correspond to four different Gaussian components identified in this complexity histogram (figure best viewed in color).

3.4 Observation #2: Number of Convolution Layers in a Rack

As discussed in Section 1, each convolution layer corresponds to a set of features (represented by kernels) of a different complexity[39].

> In other words, the more diverse the complexities of image features of a given size, the more convolution layers the corresponding rack needs to have.

Therefore, our next observation is that we can determine the number of convolution layers in a given rack, by analyzing the distribution of the complexities of the SIFT features in the corresponding octave, extracted from the dataset.

3.4.1 *Feature Complexity and Dsitribution.*

In this paper, we propose to measure the complexity of a SIFT feature, F, in terms of the entropy of the pixels within the corresponding scope as

$$E(F) = - \sum_{a \in pixel\_amplitudes} P_{scope(F)}(a) log P_{scope(F)}(a),$$

where, $pixel\_amplitudes$ is the set of possible pixel values and $P_{scope(F)}(a)$ is the distribution of the amplitude value a within the scope of the feature F. Note that each SIFT feature, F, has a center $\langle x_F, y_F \rangle$ and a scale σ_F represent the amount of Gaussian smoothing corresponding to the feature. Since under Gaussian smoothing, 3 standard deviations would cover ~99.73% of the pixels that have contributed to the identified feature, we define the radius of a feature F as $3\sigma_F$. To compute $P_{scope(F)}(a)$, we consider all pixels that are $\pm 3\sigma_F$ from $\langle x_F, y_F \rangle$ in either of the two directions. We next compute a complexity histogram, $\mathcal{H}(\mathcal{D}, O_i)$, describing the entropy distribution of the features extracted from the dataset $\mathcal{D}$, corresponding to octave O_i.

3.4.2 *Number of Convolution Layers.*

As we see in Figure 4, the entropy distribution is rarely uniform and shows a certain degree of clustering. We therefore argue that we can leverage this distribution to determine the number of convolution layers in a given rack. In particular, we argue that we can treat the complexity histogram as a mixture of k (possibly overlapping) Gaussians, where each Gaussian component corresponds to a distinct feature's complexity. Thus, given the entropy histogram, $\mathcal{H}(\mathcal{D}, O_i)$, corresponding to octave, O_i, we use a non-parametric Gaussian mixture separation algorithm[1] to (a) identify the number, g_i, of Gaussian components and (b) to associate each distinct entropy instance to one of these components. We set the number of convolutional layers as

$$\forall_{i=1\ldots r}\ c_i = g_i.$$

where, c_i is the number of convolution layers corresponding to the i^{th} rack using the number of Gaussian components„ g_i of the corresponding octave, O_i.

3.5 Observation #3: Organization of Convolution Layers within a Rack

As discussed in Section 3.1, during the *feed forward* phase of CNN, the complexities of the patterns learned increases as the training process moves from one convolutional layer to another.

> Therefore, the local image features extracted from the image dataset should be mapped to the convolution layers in the order implied by the Gaussian components to which they belong.

More specifically,

$$\forall_{i=1\ldots r}\forall_{j,h=1\ldots c_i}\quad (j > h) \leftrightarrow \left(\mu_{i,j} > \mu_{i,h}\right),$$

where $\mu_{i,j}$ is the mean of the j^{th} Gaussian component of the i^{th} rack. As described next, this order of complexities is used in distributing the kernel budget to the convolution layers in a rack.

3.6 Observation #4: Number of Kernels in a Convolution Layer

Our next key observation is that:

> the number of kernels corresponding to j^{th} convolution layer of the i^{th} rack should reflect the number of *relevant* distinct patterns of the corresponding feature size and complexity.

Unfortunately, without access to the labeled data, we do not have any information about the number of relevant distinct patterns. However, as we will see, we can replace this constraint with a more relaxed constraint which we can readily compute.

3.6.1 Convolution Layers with Non-Uniform Kernel Counts. Let $G_{i,j}$ be one of the g_i Gaussian components obtained in the previous step through the analysis of the complexity histogram at octave O_i. Let $\mathcal{F}_{i,j}$ be the corresponding set of SIFT features, again identified as a by-product of the non-parametric Gaussian separation process. We argue that the number, $k_{i,j}$, of kernels corresponding to the j^{th} convolution layer of the i^{th} rack should be *inversely proportional* to the number of similarity clusters[2] for the features in $\mathcal{F}_{i,j}$:

$$\forall_{i=1\ldots r}\forall_{j=1\ldots g_i}\ k_{i,j} \overset{inv}{\sim} num\_clusters(\mathcal{F}_{i,j}).$$

While this initially sounds surprising, there is a simple explanation for this relationship between the number of feature similarity clusters and the number of kernels:

> the higher the number of clusters of $\mathcal{F}_{i,j}$ we can identify, the more distinguishable the underlying feature patterns are and thus, the fewer the number of kernels are needed to distinguish these patterns.

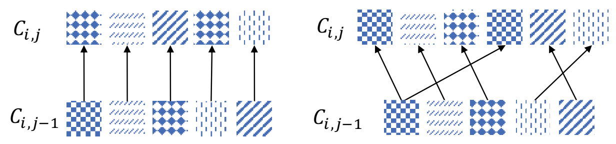
We experimentally validate this observation in Section 4 (Table 6). Note that given this observation and given a total budget of B kernels for the entire CNN, the number, $k_{i,j}$, of kernels corresponding to the j^{th} convolution layer of the i^{th} rack is computed as

$$\forall_{i=1\ldots r}\forall_{j=1\ldots c_i}\ k_{i,j} = \beta_{i,j} \times B,$$

and

$$\beta_{i,j} = 1 - \left(\frac{c_i}{\sum_{p=1\ldots r} c_p} \times \frac{n_{i,j}}{\sum_{l=1\ldots c_i} n_{i,l}}\right)$$

where, $n_{i,j} = num\_clusters(\mathcal{F}_{i,j})$ is the number of descriptor similarity clusters for octave O_i and Gaussian component, $G_{i,j}$.



(a) Conventional Weight Sharing **(b) Proposed Weight Sharing**

Figure 5: Weight sharing: in conventional strategy, weight sharing is on one-to-one basis; in the proposed approach, 1-M/N-1 sharing is possible based on kernel similarities.

3.6.2 Weight Sharing with non-Uniform Kernel Counts. An important decision in network design is the connectivity among kernels across consecutive layers. This has two aspects to consider: (a) how convolutions at a down-stream layer are related to the convolution results of the up-stream layer and (b) how, during training, weights can be shared across kernels in these two layers (or how weights learned for kernels in one layer are used to bootstrap the weight for the another consecutive layer). For instance,

- in terms convolution connections, [44] (a CNN implementation) and [29] (an RCNN implementation) both rely on full connectivity across consecutive convolution layers.
- in terms of weight sharing, however, [44] assumes no weight sharing across layers, whereas [29] assumes that weight sharing occurs on a one-to-one basis between corresponding kernels in two layers: i.e., during the feed-forward stage of each iteration, the weight for a downstream kernel is initialized with the weight of the corresponding kernel in the up-stream layer.

Note that weight sharing described above is possible because the number of kernels across two consecutive layers are the same. As we have seen in the Section 3.6.1, in *RACKNet*, we may have different number of kernels across consecutive layers; this means that, when we implement weight sharing, a one-to-one strategy will not work. This difficulty can be overcome by mapping each $K_{i,j,l}$ (in the j^{th} convolution layer of the i^{th} rack) to the most similar kernel, $K_{i,j-1,h}$, s.t. $\forall_{i=1\ldots r}, \forall_{j=2\ldots g_i}$, and $\forall_{l=1\ldots k_{i,j}}$, we have

$$\mu(K_{i,j,l}) = K_{i,j-1,h}\ \text{ s.t. }\ h = \underset{h'=1\ldots K_{i,j-1}}{argmax}\left\{cos(\vec{K}_{i,j-1,h'}, \vec{K}_{i,j,l})\right\}.$$

[1] In our implementation, we use the Gaussian mixture model available through the Python library, scikit-learn[35].

[2] In our implementation, we use the non-parametric density-based spatial clustering algorithm DBSCAN [10] with cosine distance to identify the number of similarity clusters, $n_{i,j}$, of the descriptors of the features in $\mathcal{F}_{i,j}$.

Here $\mu()$ corresponds to the mapping function and $\vec{K}_*$ corresponds to the vector representations of the convolution patterns learned during the training of the network. Note that, since the discovered patterns change at each iteration, the mapping $\mu()$ needs to be revised accordingly. Note also that the kernel alignments learned during the feed-forward stage are preserved and leveraged during the back-propagation of the gradients as well. In particular, during back-propagation, gradients are redirected to the corresponding kernel mappings, $\mu()$, for synchronous error correction.

4 EXPERIMENTS

In this section, we evaluate the proposed *RACKNet* framework for data-driven CNN design and compare its classification accuracy performance against alternative schemes.

4.1 Experimental Setup

We implemented *RACKNet* in Python environment using TensorFlow framework [1] and Keras [9]. During the training of CNN, 10% of the training data was reserved for validation to evaluate model quality. Root Mean Square Error (RMSE) is used as the model optimizer. We use *rectified-linear unit* (ReLU) [32] and *softmax* [6] as the hidden and output activation function, and for pooling we use *average* or *maxout* [20]. We use 400 as the default kernel budget and 0.25 as the default *dropout* rate . We used MatLab to extract SIFT features [31], and scikit-learn [35] both to search for Gaussian components of entropy histograms, and for feature similarity clustering using DBSCAN. All experiments were executed on an Intel Xeon E5-2670 2.3 GHz Quad-Core Processor with 32GB RAM[3].

4.2 Competitors

We have compared the proposed approach against several competitors: *wide net* [2], *maxout* [20], *dropout* [44], and RCNN [29]. Furthermore, to assess the usefulness of the four key observations that form the core of *RACKNet*, we also considered several alternative kernel allocation strategies:

- **Random budget allocation** distributes the kernel budget, B, to the conv. layers at random, $s.t.\ \sum \forall_{i=1\ldots r} \sum \forall_{j=1\ldots c_i} \beta_{ij} \times B = B$.
- **Uniform rack budget allocation** allocates the same number of kernels for each rack in the CNN and then uniformly allocates kernels for each layer in the rack; $\forall_{i=1\ldots r} \forall_{j=1\ldots c_i}\ \beta_{ij} = \frac{1}{c_i} \times \frac{1}{r}$.
- **Uniform layer budget allocation** allocates the same number of kernels for each convolutional layer irrespective of the convolution layer to which it belongs; i.e., $\forall_{i=1\ldots r} \forall_{j=1\ldots c_i}\ \beta_{ij} = \frac{1}{\sum_{p=1}^{r} c_p}$.
- In Section 3.6, we have seen that *RACKNet* allocates kernels to convolution layers in a rack inversely proportional to the feature complexities. **C-Proportional allocation** strategy uses the opposite strategy and allocates kernels to convolution layers in a rack *directly proportional* to the feature complexities: $\forall_{i=1\ldots r} \forall_{j=1\ldots c_i}\ \beta_{ij} = \frac{c_i}{\sum_{p=1}^{r} c_p} \times \frac{n_{i,j}}{\sum_{p=1}^{c_i} n_{i,p}}$.
- In noted in Section 3.5, *RACKNet* orders convolution layers ibased on the feature complexities. **Inverse-order allocation** uses the opposite allocation strategy, and places convolution layers with higher feature complexities later in the *feed-forward* sequence.

[3]Results presented in this paper were obtained using "*Chameleon: A Large-Scale Reconfigurable Experimental Environment for Cloud Research*" (NSF Award No. 1743354)

(a) MNIST[28] (b) SVHN[34] (c) CIFAR[23] (d) COIL[33]

Figure 6: Samples from the benchmark datasets

Datasets	Convolution Layers ⟨Kernels⟩
MNIST	4-4-4 ⟨35,36,32,32-30,36,30,40-35,31,35,35⟩
SVHN	3-4-3 ⟨39,30,53-35,39,37,40-40,40,41⟩
CIFAR	5-3-3 ⟨39,29,38,39,39-33,39,39-37,37,37⟩
COIL	2-4-4 ⟨8,77-44,29,44,44-46,25,46,46⟩
ImageNet	10-12-6 ⟨10,16,12,13,16,21,19,15,10,11-14,11,16,15, 17,12,15, 19,12,15,14,12-14,18,10,13,16,14⟩

Table 2: Convolutional layers and numbers of kernels ('-' denotes downsampling layer by 2) with kernel budget 400.

Bin Size	MNIST	SVHN	CIFAR	COIL
0.001	98.93	90.12	85.36	99.54
0.01	**99.48**	**97.85**	**95.74**	**99.86**
0.02	99.04	90.05	84.79	99.37
0.05	95.20	87.02	81.88	98.68

Table 3: Entropy histogram bin size vs model accuracy *RACKNet* (RCNN Implementation)

In all scenarios, the number of racks and the numbers of convolution layers per rack are selected per Observations #1 and #2.

4.3 Benchmark Datasets

For evaluation, we considered various commonly used benchmark datasets; including *digit* datasets, MNIST and SVHN, and *real-world image* datasets, CIFAR10, COIL20, and ImageNet (Figure 6). **MNIST** is a dataset containing $60k$ and $10k$ training and testing images, respectively of 28×28 handwritten digit captures (Figure 6(a)). **SVHN** dataset consists of 32×32 house numbers extracted from Google Street View images. The dataset consists of $73k$ and $26k$ images for training and testing (Figure 6(b)). **CIFAR10** contains $50k$ training and $10k$ testing images, respectively, with 32×32 resolution and the dataset contains 10 labels (Figure 6(c)). **COIL20** contains images of 20 real-world objects. For each object, the dataset includes 72 images, captured at 5-degree intervals by rotating a turntable 360 degrees (Figure 6(d)). **ImageNet** contains ~1.23 million images for 1000 real-world entities, with ~1000 images per entity [24].

Table 2 shows the hyper-parameters extracted using the *RACKNet*. In particular, each feature octave corresponds to a "rack", and the number of layers per rack is determined using the data - not by user input. Entropy histogram bin size of 0.01 is used as default: as we see in Table 3, *RACKNet* performs well with this bin size, independent of the dataset.

4.4 Results

4.4.1 RACKNet vs. Competitors. In Table 4 and 5, we compare the RCNN-based implementation of *RACKNet* against various state

	MNIST	SVHN	CIFAR	COIL
Maxout[20]	99.55	97.53	88.32	–
Dropout[44]	99.21	97.45	87.39	–
Deep Net[2]	99.07	–	–	99.23
Wide Net[2]	97.66	–	–	99.36
RCNN[29]	98.12	93.67	75.28	90.02
RACKNet	99.48	97.85	95.74	99.86
***RACKNet*-maxout**	**99.72**	**98.54**	**97.62**	**100**

Table 4: Reported accuracies for the competitors vs *RACKNet* (RCNN implementation) accuracy.

Approaches	Parameters	Accuracy
Dropout[44]	–	61.90
VGG-16 [42]	134M	72.7
PReLU [14]	–	78.41
Batch Normalization [21]	–	78.01
DenseNet-126 [18]	7M	74.98
RESNet-34 B [15]	0.46M	78.16
RESNet-34 C [15]	0.46M	78.47
SENet [17]	26.9M	**81.32**
Random	0.39M	69.12
Uniform-Layer (CNN-Plain)		71.74
***RACKNet*-CNN**	0.39M	76.82
***RACKNet*-RCNN**		79.14
***RACKNet*-RCNN-Maxout**		81.02

Table 5: Top-1 classification accuracy for ImageNet

Strategies	Observation	MNIST	SVHN	CIFAR	COIL
Random	$\langle o_1, o_2\rangle$	98.98	90.34	69.69	99.53
Uni. Rack	$\langle o_1, o_2\rangle$	98.97	90.90	70.18	99.55
Uni. Layer	$\langle o_1, o_2\rangle$	98.97	91.04	69.50	94.57
C-Prop.	$\langle o_1, o_2, o_3\rangle$	98.9	90.77	68.50	99.16
Inv.-Order	$\langle o_1, o_2, o_4\rangle$	98.99	91.11	69.88	99.38
RACKNet	$\langle o_1, o_2, o_3, o_4\rangle$	**99.25**	**94.57**	**71.65**	**99.94**

Table 6: Accuracy for various kernel allocation strategies.

	MNIST	SVHN	CIFAR	COIL
Rack−2	99.56	98.51	97.31	**100**
Rack−3	**99.72**	**98.54**	**97.62**	**100**
Rack−4	98.78	97.21	96.83	99.63

Table 7: Accuracy vs Number of Racks, *RACKNet*-RCNN

of the art CNN and RCNN techniques. As the tables show, the network design based on the *RACKNet* framework provide the best overall accuracies, indicating that *RACKNet* allocates kernel resources more effectively than the handtuned competing architectures, with the exception of SENet[17] which has a similar accuracy to *RACKNet.* It is important to note that SENet which is significantly deeper than *RACKNet*, with 154 layers against only 28 layers wheras *RACKNet* achieves similar accuracies with as low as 390k hyperparameters as opposed to 26.9 million in SENet.

4.4.2 RACKNet vs. Alternative Allocation Strategies. Table 6 compares *RACKNet*-based kernel allocation to alternative kernel allocation strategies. Results in this table confirm that the four key observations that form the core of the *RACKNet* framework are highly

	MNIST (60K)	SVHN (73K)	CIFAR (50K)	COIL (1.3K)	ImageNet (1.23M)
Feature	340.34	449.35	302.10	110.59	8439.32
Entropy Hist.	81.04	119.47	60.41	26.56	4110.04
Mixture Sep.	1.79	1.71	2.17	1.93	3.76
Clustering	75.76	97.45	62.13	30.21	1529.67
Training	929.33	1396.79	1189.61	187.01	31952.12

Table 8: Execution time (*in seconds*)

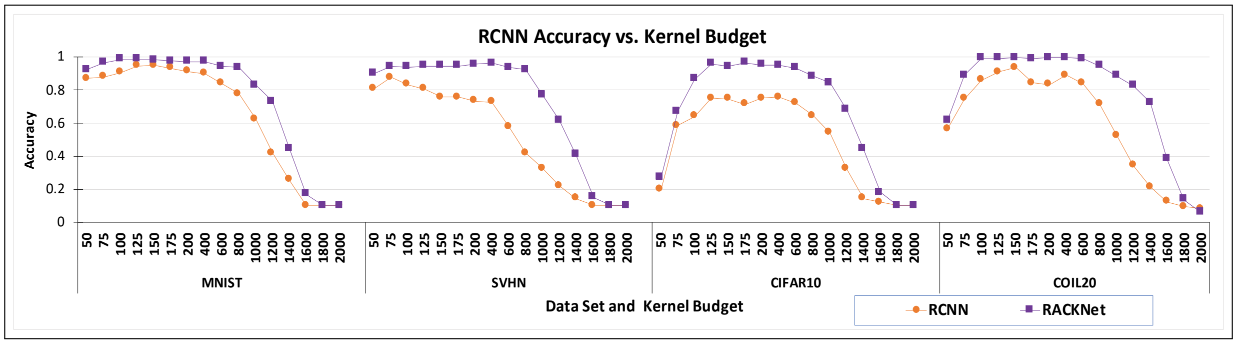
effective and that *RACKNet* is the only strategy that consistently outperforms the random allocation strategy. It is especially important to note that while naiïve allocation strategies, such as random, show reasonable performance on simple datasets, like MNIST and CIFAR10, they show very poor performance for complex ones, such as ImageNet, In contrast, *RACKNet* provides consistently superior performance for both simple and complex data sets.

4.4.3 Kernel Budget, Dropout Rates, and Racks. In Figures 7 and 8, we investigate the robustness of *RACKNet* (CNN and RCNN) against varying kernel budgets and dropout rates. As we see in the figures, *RACKNet* is robust against varying kernel budgets and dropout rates and can help provide significant protection against introduction of noisy kernels that degrade the network accuracy. In Table 7, we see that *RACKNet* works well with 3 racks, as predicted by the number of feature octaves of localized SIFT features. Providing a higher number of racks does not contribute to performance as the number of SIFT features drop significantly for larger octaves, limiting any discernible insights from the corresponding features.

4.4.4 Execution Time. *RACKNet* requires extraction and analysis of local image features for their sizes and complexities before the CNN is trained. Table 8 presents the time cost of this process along with the CNN training time. As this table shows, the pre-processing overhead of *RACKNet* is not high. Since the pre-processing cost does not grow as fast as the CNN training cost, *RACKNet* becomes cost efficient especially for large training sets, such as SVHN. A key advantage of advantage of *RACKNet* over other hyper-parameter search approaches is that, while they have to train **multiple** network configurations during the hyper-parameter search, *RACKNet* trains only a **single** network configuration; instead, it pre-processes the data to determine the best performing configuration Since the pre-processing cost grows much slower that the network training cost, this provides large cost savings.

5 CONCLUSION

In this paper, we proposed *RACKNet*, a framework for allocating convolution kernels for different layers of a CNN. In particular, noting that local image features pre-extracted from the training data can provide reliable insights that can inform the design of the CNN architectures, we presented four key observations that link sizes, complexities, and distributions of these local features to CNN hyper-parameters. Experiments using several benchmark datasets have shown that the proposed approach leads to highly accurate classifiers without requiring hand-crafting of the hyper-parameters. Experiments have further shown that the proposed framework leads to more-accurate CNNs that are robust against kernel budget availabilities, dropout rates, and Racks.



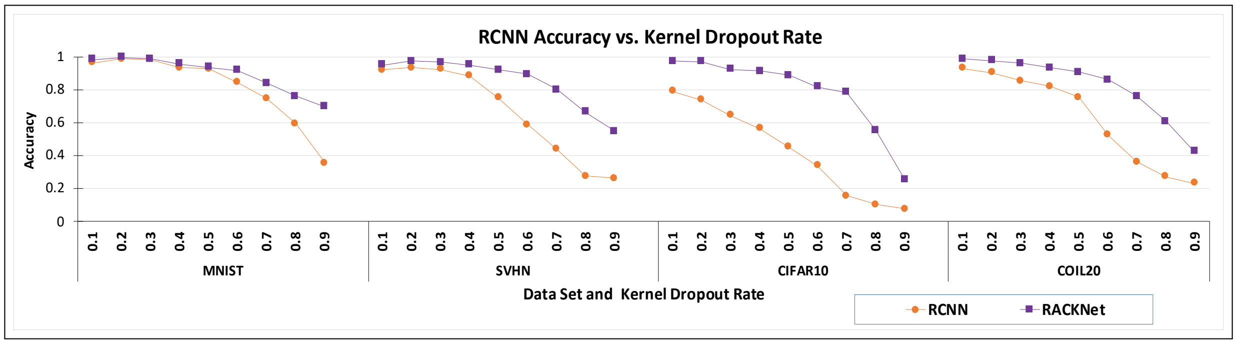
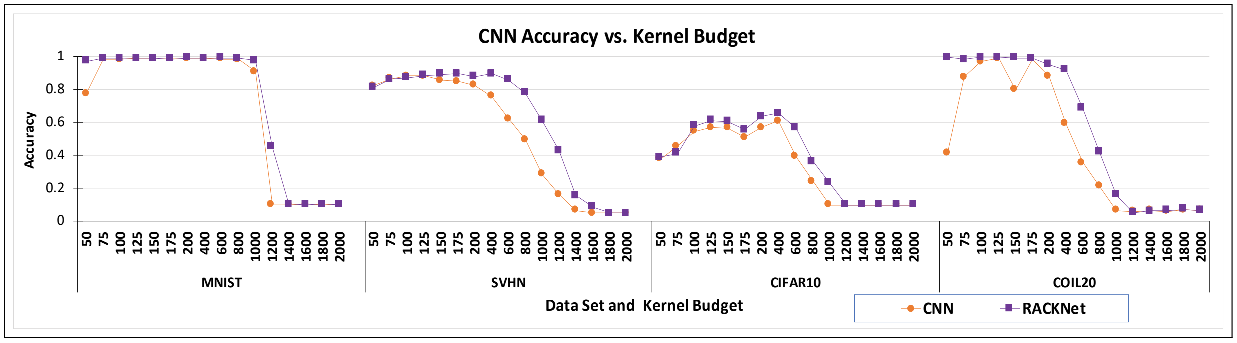


Figure 7: Accuracy vs. kernel budget and dropout rates (RCNN implementation), for clarity, we only show RCNN with uniform-layer budget allocation strategy.



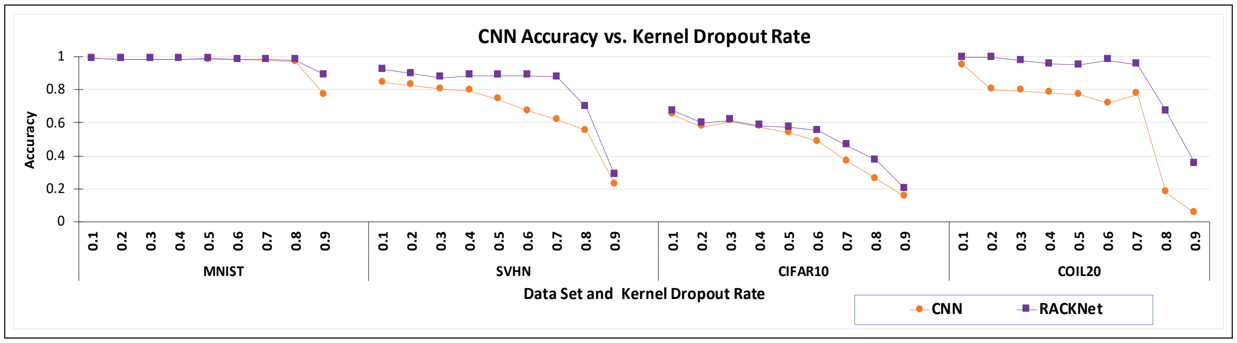


Figure 8: Accuracy vs. kernel budget and dropout rates (CNN implementation), for clarity, we only show CNN with uniform-layer budget allocation strategy.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRRs* (2016).

[2] Md Zahangir Alom, Theodore Josue, Md Nayim Rahman, Will Mitchell, Chris Yakopcic, and Tarek M Taha. 2018. Deep Versus Wide Convolutional Neural Networks for Object Recognition on Neuromorphic System. *arXiv preprint arXiv:1802.02608* (2018).

[3] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. 2008. Speeded-up robust features (SURF). *Computer vision and image understanding* 110, 3 (2008), 346–359.

[4] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13, Feb (2012), 281–305.

[5] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems.* 2546–2554.

[6] John S Bridle. 1990. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In *Advances in neural information processing systems.* 211–217.

[7] Andrea Ceroni, Chenyang Ma, and Ralph Ewerth. 2018. Mining Exoticism from Visual Content with Fusion-based Deep Neural Networks. In *Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval.* ACM, 37–45.

[8] Parag Shridhar Chandakkar and Baoxin Li. 2016. A structured approach to predicting image enhancement parameters. In *Applications of Computer Vision (WACV), 2016 IEEE Winter Conference on.* IEEE, 1–9.

[9] François Chollet et al. 2015. Keras. https://github.com/fchollet/keras. (2015).

[10] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96).* AAAI Press, 226–231. http://dl.acm.org/citation.cfm?id=3001460.3001507

[11] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. 2010. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence* 32, 9 (2010), 1627–1645.

[12] Yash Garg. 2015. *Multi-Variate Time Series Similarity Measures and Their Robustness Against Temporal Asynchrony.* Arizona State University.

[13] Yash Garg and Silvestro Roberto Poccia. 2017. On the Effectiveness of Distance Measures for Similarity Search in Multi-Variate Sensory Data: Effectiveness of Distance Measures for Similarity Search. In *ICMR.* ACM, 489–493.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision.* 1026–1034.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR.* 770–778.

[16] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* 29, 6 (2012), 82–97.

[17] Jie Hu, Li Shen, and Gang Sun. [n. d.]. Squeeze-and-excitation networks. ([n. d.]).

[18] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. 2017. Densely Connected Convolutional Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* IEEE, 2261–2269.

[19] Siyu Huang, Xi Li, Zhiqi Cheng, Alexander Hauptmann, et al. 2018. GNAS: A Greedy Neural Architecture Search Method for Multi-Attribute Learning. *Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval* (2018).

[20] Mehdi Mirza Aaron C. Courville Ian J. Goodfellow, David Warde-Farley and Yoshua Bengio. 2013. Maxout Networks. In *Proceedings of the 30th International Conference on Machine Learning.*

[21] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37 (ICML'15).* JMLR.org, 448–456. http://dl.acm.org/citation.cfm?id=3045118.3045167

[22] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. 2014. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition.* 1725–1732.

[23] Alex Krizhevsky and Geoffrey Hinton. 2009. Learning multiple layers of features from tiny images. (2009).

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems.* 1097–1105.

[25] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. 2007. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th international conference on Machine learning.* ACM, 473–480.

[26] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. 1997. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks* 8, 1 (1997), 98–113.

[27] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. 1990. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems.* 396–404.

[28] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[29] Ming Liang and Xiaolin Hu. 2015. Recurrent convolutional neural network for object recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 3367–3375.

[30] Sicong Liu, Yash Garg, K Selçuk Candan, Maria Luisa Sapino, and Gerardo Chowell-Puente. 2015. Notes2: Networks-of-traces for epidemic spread simulations. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence.*

[31] David G Lowe. 2004. Distinctive image features from scale-invariant keypoints. *International journal of computer vision* 60, 2 (2004), 91–110.

[32] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. 2013. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, Vol. 30. 3.

[33] Sameer A Nene, Shree K Nayar, Hiroshi Murase, et al. [n. d.]. Columbia object image library (coil-20). ([n. d.]).

[34] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. [n. d.]. Reading digits in natural images with unsupervised feature learning.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[36] Xiang Zhang Michaël Mathieu Rob Fergus Pierre Sermanet, David Eigen and Yann LeCun. 2014. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. In *2nd International Conference on Learning Representations.*

[37] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. 2017. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70.* JMLR. org, 2902–2911.

[38] Pierre Sermanet, Soumith Chintala, and Yann LeCun. 2012. Convolutional neural networks applied to house numbers digit classification. In *Pattern Recognition (ICPR), 2012 21st International Conference on.* IEEE, 3288–3291.

[39] Claude Elwood Shannon. 1948. A mathematical theory of communication. *Bell system technical journal* 27, 3 (1948), 379–423.

[40] Poccia Silvestro Roberto, Luisa Sapino Maria, Liu Sicong, Chen Xilun, Garg Yash, Huang Shengyu, Hyun Kim Jung, Li Xinsheng, Nagarkar Parth, and K Selcuk Candan. 2017. SIMDMS: Data Management and Analysis to Support Decision Making through Large Simulation Ensembles. In *20th International Conference on Extending Database Technology (EDBT'17).* OpenProceedings. org, 582–585.

[41] Karen Simonyan and Andrew Zisserman. 2014. Two-stream convolutional networks for action recognition in videos. In *Advances in neural information processing systems.* 568–576.

[42] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. (2015).

[43] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems.* 2951–2959.

[44] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

[45] Jianbo Yang, Minh Nhut Nguyen, Phyo Phyo San, Xiaoli Li, and Shonali Krishnaswamy. 2015. Deep Convolutional Neural Networks on Multichannel Time Series for Human Activity Recognition.. In *IJCAI.* 3995–4001.

[46] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. 2010. Deconvolutional networks. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on.* IEEE, 2528–2535.

[47] Zheng Zhang, Ping Tang, and Rubing Duan. 2015. Dynamic time warping under pointwise shape context. *Information sciences* 315 (2015), 88–101.

[48] Liang Zheng, Yi Yang, and Qi Tian. 2018. SIFT meets CNN: A decade survey of instance retrieval. *IEEE transactions on pattern analysis and machine intelligence* 40, 5 (2018), 1224–1244.

[49] Yi Zheng, Qi Liu, Enhong Chen, Yong Ge, and J Leon Zhao. 2014. Time series classification using multi-channels deep convolutional neural networks. In *International Conference on Web-Age Information Management.* Springer, 298–310.

[50] Barret Zoph and Quoc V. Le. 2017. Neural Architecture Search with Reinforcement Learning. In *International Conference on Learning Representations, ICLR 2017.*