Speeded-up Convolution Neural Network for classification tasks using multiscale 2-dimensional decomposition.

Jhilik Bhattacharya

Thapar Institute of Engineering and Technology

Giovanni Ramponi

University of Trieste

Abstract

Convolutional neural networks (CNNs) are currently used for detection, recognition and verification tasks in most domains, and in particular in vision-related applications. However, their success depends on the availability of large datasets often consisting of millions of images. This makes the implementation of domain-specific training difficult, since such huge training sets are not available in many scenarios. Also, training CNNs calls for large computational resources in terms of energy, memory usage, and time. Much research therefore concentrates on the CNN architecture, e.g. to analyze whether smaller resolution inputs require less training data size as well as training time, while retaining uncompromised accuracy. Compressing network weights efficiently, and exploring alternatives for transfer learning from a large net to a smaller one are also important issues in this aspect. In this paper, we propose a strategy for network simplification and acceleration. First, we propose to generate a suitable resized image using multiscale patching in the first convolutional layer, which can then be used for the rest of the network. We also introduce a 2-dimensional decomposition for patch compression, as well as a layer weight decomposition technique followed by module-based finetuning for a new fast module-based CNN model. Extensive evaluations using public data sets like MNIST, Pascal VOC, and

Email addresses: jhilik@thapar.edu (Jhilik Bhattacharya), ramponi@units.it (Giovanni Ramponi)
WebCASIA, and with different state-of-the-art CNN architectures like OpenFace, VGG and DarkNet verify that our proposed model is able to accelerate the training process and even to provide higher classification accuracies for small-sized datasets. This article will help readers who strive to achieve good performances with limited datasets and/or small networks.

Keywords: image classification, modular finetuning, multiscale compression, weight decomposition

1. Introduction

Object detection and classification are fundamental tasks in a wide spectrum of applications, ranging from surveillance to crowd behavior analysis, to video semantics understanding, etc. Fortunately, we have been experiencing a sharp increase of the performances of unconstrained classification techniques since deep learning methods entered the field. For example, representations using Convolutional Neural Networks (CNN) trained on the ImageNet dataset [1] are now successfully applied in fine-grained object recognition, image understanding, autonomous driving.

Yet, deep learning methods generally come at a high cost. CNN training requires large annotated datasets (which for some local classification applications it may not even be possible to generate), powerful graphics processing units (GPUs), not to mention the required training time and energy consumption. Even the inference time, i.e. the time the trained network requires to do its classification job, can constitute a problem in real-time applications.

A strong research effort is being made to devise speeded-up networks characterized by a smaller number of parameters. Beyond shorter training and inference times, and lower training data requirements, smaller CNNs offer other advantages including less communication activity during distributed training, lower bandwidth for transfer, and lower memory requirements during processing.

Acceleration techniques can be roughly categorized into five basic classes: shared convolution feature maps, reduced number of layers, network architecture modifications, transfer learning, and weight compressions. An example of computation acceleration with shared feature maps is the modification of Region-CNNs [4] implementation for object detection to Fast-RCNN [5] and Faster-RCNN [6]. The same concept is exploited for multitask learning [7]. It is quite obvious that a reduced number of layers in a network architecture
is an option for faster inference time when limited computational resources and training data are available [8]. Hence Alex-Net [18] is preferred to VGG [19], and Tiny-YOLO and OpenFace are preferred to YOLO [3] and Facenet respectively in such scenarios. Architectural designs that reduce the number of filters per layer [9] or use $1 \times 1$ convolution kernels [20] also result in fewer computations. Another example of architectural design-based acceleration is [14], which adds an extra layer for compressing the input provided to the network. Transfer learning methods [10] train a powerful, yet cumbersome, model to extract structure from the data. The model’s knowledge is later transferred to a small deployable model, using the class probabilities produced by the large model as soft targets to train the small model. The fifth class of acceleration techniques use weight approximations for network compression. Compression is achieved in this class by network pruning and quantization [11]. This reduces the network size to a large extent making it appropriate for embedded systems. A direct dimension reduction using Singular Value Decomposition (SVD) can also be used [12].

Finally, it should be mentioned that the input resolution has a role, along with the network size, for the number of required training images. For example, a small version of FaceNet coined as OpenFace [2] uses images having size $96 \times 96$ instead of the $240 \times 240$ size used in Facenet, hence reducing the training time from one week to one day. This model, trained with only 500,000 images chosen from the 200 million training dataset of Facenet, obtains a 97.3% accuracy on the LFW (Labelled Faces in the Wild) dataset, compared to the Facenet record of 99.63%. The inference time, too, is directly related to the input resolution and to the number of layers in the network. Openface has an inference time around 59 ms (as reported by the Authors, and without a GPU), compared to 112 ms for Facenet.

The present work aims towards simplifying a pretrained CNN for faster inference time, by using a fast and efficient weight update with a small training dataset. We focus on using a reduced number of layers, implementing architectural modifications and weight approximations. Motivations for our study and advances we propose in each case are further elaborated in Section 2.

2. Motivations and novel contributions

In this contribution, the acceleration approaches are investigated as follows:
• **The role of subsampling to feed a smaller resolution network.** The Openface network has a $96 \times 96$ input size, which is much smaller than the one of most images. There is a substantial difference between using interpolation to resize the available input images and compressing them with techniques like CS-CNN [14]. The technique used in CS-CNN demonstrates that adding on top of a CNN architecture a first convolutional layer that compresses the input dimension reduces the training and inference time. With respect to the original architectures, it also provides better performances with smaller datasets. We enhance this technique and propose a dynamic compression technique where any input size greater than the desired output size (96 in this particular case) can be accepted for compression. Instead of resizing it by interpolation prior to applying the projection matrix (as is the case for [14]), we select a few weighted overlapping patches that meet the block size requirements Without adding computation time. This is elaborated in Section 4.1. We have experimented a few different overlapping patch selection methods taking the multiscale factor into consideration.

• **The impact of two-dimensional multiscale decomposition for input subsampling.** In order to increase the effectiveness and efficiency of the features, we use a two-dimensional (2D) decomposition instead of a one-dimensional decomposition. The multiscale patches generated in the first step incorporate spatial correlation and are further compressed using the 2D decomposition technique as described in Section 4.2. This permits to restrict the multiscale patches generation to a fixed number per image, unlike the 1D technique. We infer that using 2D multiscale patches for image resizing or compression leads to prominent improvements in recognition accuracy. The computation increase caused by moving from 1D to 2D patch decomposition is only slightly higher.

• **Techniques used for weight approximation for network compression.** The above discussed weight approximations (pruning [11], SVD [12, 13]) are weight representations of a large network. They result in a lower number of multiplications (SVD) or less space (pruning). However, a network with a different smaller architecture needs to be trained from scratch. The possibility to reuse the weights of the bigger network with minimal or no finetuning is yet to be considered. To explore this
issue we propose a new weight initialization technique where weights
from a bigger network can be passed on to a smaller network. To our
knowledge this is the first attempt at compressing weights such that a
layer with weight size e.g. 512 × 1024 can be assigned to a layer with
weight size 256 × 512. This is further elaborated in Section 5. Using
this, we demonstrate the differences among the performances of VGG-
Face, tiny-network using original VGG weights, Darknet, tiny-network
using Darknet. The approximated weights can further be finetuned
using a module-based transfer learning mechanism (Section 6.5).

In summary, we propose the following novel contributions:

- A dynamic compression technique using multiscale patch selection which
  allows input compression and leads to faster inference and higher ac-
  curacy without the need of resizing the input to suit the architecture.

- The use of a 2-dimensional reduction instead of a single dimension
  projection, which we demonstrate to improve recognition accuracy.

- A weight initialization technique that can promote transfer learning
  from a bigger network to a smaller network with modular finetuning.

Section 3 provides a brief background of two existing compression mech-
anisms, CS-CNN [14] and Runtime Compression [13], to pave the way to our
work.

3. Background of CNN compression

3.1. CS-CNN

In CS-CNN [14], the original image is first divided into \( Q = M \times N \)
patches, each of size \( m \times n \). \( k \) such images are taken and a matrix \( D \) of size
\( mn \times kMN \) is generated. A SVD of matrix \( D \) gives

\[
D = U_{mn \times mn} \Lambda_{mn \times kMN} V_{MN \times kMN}^T
\]

(1)

Here \( U, V \) are the orthogonal matrices and \( \Lambda \) is the diagonal matrix of the
SVD decomposition. These are used along with the matrix dimensions
\( (mn, kMN) \) in subscripts throughout the paper for representing the stan-
dard \( U, V, \Sigma \) notation for SVD. The first \( p \) columns are then selected and \( p \)
convolutional filters of size \( mn \times mn \) each are generated. Each image is then
processed using these filters, to generate an output of size $p \times MN$. Since $p \ll mn$, a significant compression is achieved. This technique however requires the same number of blocks per image. In order to permit compression of any input size without resizing, we use a multiscale block selection as described in Section 4.1.

3.2. Runtime Compression

Assuming a weight matrix $W_{m \times n}$ between two layers $L$ and $L + 1$, the number of parameters and pairwise computations is reduced from $m \times n$ to $(m + n)c$ if $c$ principal components are selected from $m,n$. Here $c$ is selected based on the available resources, such that $c \ll m,n$. Next, the weight matrix $W_{m \times n}$ is replaced by the new matrices, $U_{m \times c}$ and $V_{c \times n}$, to obtain an approximation of the original $W_{m \times n}$ using the singular decomposition components $U$, $\Lambda$ and $V$ of $W$ as seen in equation 4. This is achieved by introducing a new layer $L'$ with $c$ units between $L$ and $L+1$. Since layers are fully connected, the number of pairwise calculations and weight parameters reduces from $mn$ to $mc + cn$.

$$W_{m \times n} = U_{m \times m} \Lambda_{m \times n} V_{n \times n}^T$$  \hspace{1cm} (2)

approximated as:

$$W_{m \times n} = U_{m \times c} \Lambda_{c \times c} V_{c \times n}^T$$  \hspace{1cm} (3)

$$W_{m \times n} = U_{m \times c} V_{c \times n}$$  \hspace{1cm} (4)

Other techniques exist [12] that perform parameter reduction, e.g. via biclustering and SVD, but in a slightly different manner. In spite of the differences, they are designed to approximate weight matrices with fewer parameters. In this work, we use the weight matrix of a larger network to approximate the weights of a smaller network, i.e. a $W_{m \times n}$ is reduced to $W_{a \times b}$ where $m > a$ and $n > b$. This is discussed in Section 5. We adapt the technique described in Equation 4 above for some experimental comparisons, with a fixed $c$; i.e., our $c$ is predetermined and is not initialized based on the available resources.
4. Dynamic Multiscale Patch Generation

Inspired by the promising results in CS-CNN, we further investigate the scope of using a multiscale patch generation technique prior to projection. This has two advantages: first, the multiscale patches will provide pertinent information which will directly improve accuracy. Also, it will permit dynamic patch generation without resizing and without increasing the computation time. Suitable ways to generate multiscale patches include overlapping patches of different resolutions on the original image, overlapping lowpass-filtered and downsampld patches of a larger window, overlapping wavelet-transformed patches of a larger window, etc. In this work we used patches of different sizes as well as downsampld images.

4.1. Multiscale patch selection

To process the image data, we need a fixed patch size $m \times n$ to generate the projection filters of size $p \times mn$, and a fixed number of patches $MN$ such that the output to the next layer is $p \times MN$. Hence, we select all the possible non-superposed $m \times n$ patches from the available images. If the number of such patches is less than $MN$, the remaining patches are collected using scales $a \times b$ or $c \times d$ such that $ab = cd = mn$ or $a = 2m$ or $b = 2n$. Patches generated from the former condition are directly extracted from the images, while the downsampled results are used in the latter case. For example, we select a window of size $2m \times 2n$ from the image and perform a decomposition on it. The lowpass coefficient matrix of size $m \times n$ is now our desired patch. This is further demonstrated in Figure 1. Some details are mentioned in Section 6.3. The positions from which the downsampled patches are generated, are selected randomly. The compression technique following patch selection is the same as the one given in Equation 1.

4.2. 2D decomposition

According to Equation 1, we stack all the available image patches ($MN$ per image) along the columns to get $kMN$ total patches for $k$ images. When $k$ is large it is sensible and more efficient to use a 2D decomposition. It should be noted that we can generate more than $kMN$ patches to calculate the projection filters; however, only $MN$ patches are required to generate the output image.

To determine the optimal projection matrices $X$ and $Y$ for both directions, we refer to the 2DPCA algorithm in [25]. The optimal solution is
Figure 1: Multiscale patch selection technique. (a) and (b) both depict first $MN - k$ out of $MN$ patches of size $mn$ generated from the image. In (a) the remaining $k$ patches are lowpass components from random $2m \times 2n$ sized windows using wavelet decomposition, resulting in $m \times n$ sized patches while in (b) the remaining patches are random patches of size $a \times b$ given $ab = mn$.

provided by maximizing the total scatter of the projected data; in turn, the scatter is defined as the trace of the covariance matrix of such data.

We apply the 2D decomposition in three ways:

- A 3D matrix $A$ of size $mn \times MN \times k$ is generated from $k$ images. $A$ can be written as $\{A^1, A^2, ... A^k\}$. Each $A^i$ has $MN$ patches of size $m \times n$. We can generate $G$ patches for each $A^i$, where $G$ is chosen such that it is greater than or equal to the required minimum number of patches $Q = MN$ per image. We then calculate the vertical and horizontal covariance matrices, $C_v$ and $C_h$ of size $mn \times mn$ and $G \times G$ respectively, for $A$ using Equation 6. It should be noted that if $G = Q$ we need to select all the components from the horizontal covariance matrix. Further, we select $p$ and $MN$ principal components from the eigenvectors $Y$ and $X$ of the covariance matrices $C_v$ and $C_h$. The final matrix $I^i$ is generated as shown in equation 6. We refer this as 2DDMS.
(dynamic multiscale 2D decomposition).

\[ C_h = E[(A - E(A))^T(A - E(A))] \]
\[ = \frac{1}{k} \sum_{i=1}^{k} (A^i - \hat{A})^T (A^i - \hat{A}) \]
\[ \hat{A} = \frac{1}{k} \sum_{i=1}^{k} A^i \]  
\[ C_v = E[(A - E(A))(A - E(A))^T] \]
\[ I_{p \times MN}^i = Y_{p \times mn}^T A_{mn \times G}^i X_{G \times MN} \]

where \( G > MN \), and \( \hat{A} \) denotes the mean.

- We directly use the input matrices to compute the compressed version having size \( p \times MN \). In this case we cannot avoid resizing the input to a fixed size so that the covariance matrices can be computed. This is referred as 2DD.

- Finally, we apply the 2D decomposition to the patch selection proposed in [14]. This is referred as 2DCSCNN.

Applying a random orthogonal transformation to the 2D decomposed data balances the variance along different directions. This orthogonal transformation can further be refined iteratively. The random orthogonal transformation is applied as shown in Equation 6 to refine the \( X \) and \( Y \) used in Equation 6 into \( \hat{X} \) and \( \hat{Y} \). In each iteration we compute \( R_1 \) and \( R_2 \) as the SVD of \( Y^T A (Y^T A)^T \) and \( (AX)^T AX \) respectively.

\[ \hat{Y} = Y R_1 \]
\[ \hat{X} = X R_2 \]  

The impact of each action is later reviewed through experiments.

5. Weight Initialization

The decomposition method used in Equation 4 provides an approximation of the original network using a lower number of weights. However, these
cannot directly be transferred to a smaller network. An option to solve this problem is soft target based learning [10], discussed before. To the best of our knowledge, a possibility to reuse the weights even if the size does not match is not reported yet. We introduce a decomposition of the original weights via dimension reduction to match the weight matrix of a smaller network. A weight matrix $\overline{W}_b \in \mathbb{R}^{m \times n}$ can be reduced to $\overline{W}_l \in \mathbb{R}^{x \times y}$ where $x \leq m$ and $y \leq n$ as shown below:

$$\overline{W}_l = \overline{W}_b R$$

$$R = \text{SVD}(\overline{W}_l^T \overline{W}_b)$$

$R$ is basically projecting the data as used for minimizing quantization loss in [15]. Here, $R$ is computed to meet the dimension requirements for the smaller weight matrix. For example, we can initialize $R \in \mathbb{R}^{n \times y}$ yielding an initial $\overline{W}_l \in \mathbb{R}^{m \times y}$. We need to minimize $||\overline{W}_l - \overline{W}_b R||_F^2$. The first term $\overline{W}_l$ denotes the ideal reduced weights whereas the second term denotes the reduced weights $\overline{W}_b R$ to be computed iteratively until convergence. This is equivalent to

$$||\overline{W}_l - \overline{W}_b R||_F^2 = ||\overline{W}_l||_F^2 + ||\overline{W}_b||_F^2 - 2tr(\overline{W}_l R^T \overline{W}_b^T)$$

As $\overline{W}_b$ is fixed, the minimization is equivalent to solving for $tr(\overline{W}_l R^T \overline{W}_b^T)$. We solve it in 2 steps iteratively using $\overline{W}_l = \overline{W}_b R$ and equation 8. $R$ is initialized randomly. Convergence was achieved within very few iterations. This problem is posed as an iterative approximation problem inspired from IQA [16]. Further details can be found in [17].

6. Experimental results and discussion

6.1. Datasets and models

We initially repeat the experiments performed using CS-CNN, to show the impact of using dynamic multiscale patches. For this purpose we use 10,000 samples of the MNIST dataset for training and testing an architecture similar to LeNet [22] (called LeNet-like). Further, we demonstrate the effects of our technique for image resizing and weights projection on more complicated datasets and architectures. For this second group of experiments, we make use of the WebCASIA [21] face dataset and the Pascal VOC
WebCASIA is a publicly available dataset consisting of 10,575 subjects and 494,414 images. The size of this dataset ranks second in the literature, only smaller than the private dataset of Facebook (SCF). Pascal VOC has 20,000 images and 20 classes. The architectures we use are OpenFace (Table 1), Darknet, and VGG-Face. A tiny-VGG and a tiny-Darknet are generated from the original VGG-Face and Darknet as shown in Tables 2 and 3. In Table 1, the number of filters and the filter size are given for the Spatial Convolution layers; window size and stride are given for the Maxpool layers, and the feature size output for the View and Linear layers are provided. The Inception Layer is a concatenation (depicted as ":") of two or four operations. \((f_1, n_1|f_2, n_2)\) denotes a depth-two layer with \(f_1\) and \(f_2\) filters of size \(n_1\) and \(n_2\).

It should be noted that the recognition results with face/object datasets and pretrained networks are used for demonstration purposes. The algorithms proposed in this work can be generalized to any dataset and application in the wild.

6.2. Classification using LeNet-like architecture

In this subsection we perform the experiments demonstrated in CS-CNN [14]. We also demonstrate the impact of the number of samples and of the number of projections. The inference times we obtain as a function of the number of projections are the same obtained with CS-CNN (that is lower than the inference times of the original architecture), as the network structure remains the same in both cases; only the technique to compute the patches differ. For example, the inference time per image using 2, 4, 6, and 8 projections is 0.21, 0.26, 0.28, and 0.29 ms respectively in our system. Figure 2.3 depicts the classification accuracy on MNIST obtained using CSCNN and our multiscale patches. The results are shown with respect to the number of training samples as well as the numbers of projections. The patches have size \(4 \times 4\), and are generated from 20 images from each class in both cases. The results in Figure 3 refers to the case of four projections.

6.3. Dynamic Multiscale Patch Generation

We use 361 classes from Web-CASIA for our experiments (a total of 67,000 images) to train the classifier, and 3610 images to test it. In case of Pascal VOC we use region proposals for testing. We generated 650,000 region proposals to test the object classification results for 80 classes. We perform two different kinds of experiments with each of the three networks. First
<table>
<thead>
<tr>
<th>Layers</th>
<th>filters, filter size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spatial Convolution</td>
<td>64,7</td>
</tr>
<tr>
<td>BatchNorm</td>
<td></td>
</tr>
<tr>
<td>ReLU</td>
<td></td>
</tr>
<tr>
<td>MaxPool</td>
<td>3,2</td>
</tr>
<tr>
<td>CrossMapLRN</td>
<td></td>
</tr>
<tr>
<td>Spatial Convolution</td>
<td>64,1</td>
</tr>
<tr>
<td>BatchNorm</td>
<td></td>
</tr>
<tr>
<td>ReLU</td>
<td></td>
</tr>
<tr>
<td>Spatial Convolution</td>
<td>192,3</td>
</tr>
<tr>
<td>BatchNorm</td>
<td></td>
</tr>
<tr>
<td>ReLU</td>
<td></td>
</tr>
<tr>
<td>CrossMapLRN</td>
<td></td>
</tr>
<tr>
<td>MaxPool</td>
<td>3,2</td>
</tr>
<tr>
<td>Inception</td>
<td>64,1:(96,1</td>
</tr>
<tr>
<td>Inception</td>
<td>64,1:(96,1</td>
</tr>
<tr>
<td>Inception</td>
<td>(128,1</td>
</tr>
<tr>
<td>Inception</td>
<td>256,1:(96,1</td>
</tr>
<tr>
<td>Inception</td>
<td>224,1:(112,1</td>
</tr>
<tr>
<td>Inception</td>
<td>(160,1</td>
</tr>
<tr>
<td>Inception</td>
<td>384,1:(192,1</td>
</tr>
<tr>
<td>Inception</td>
<td>384,1:(192,1</td>
</tr>
<tr>
<td>AvgPool</td>
<td>1024</td>
</tr>
<tr>
<td>View</td>
<td>896</td>
</tr>
<tr>
<td>Linear</td>
<td></td>
</tr>
<tr>
<td>Normalize</td>
<td>128</td>
</tr>
</tbody>
</table>

Table 1: OpenFace Network Architecture with number of filters and filter sizes in each layer. An entry \textit{Inception (160,1|256,3):(64,1|128,5)} denotes that an Inception block with two concatenation operations is used. The first operation is a depth-2 layer with 160 and 256 filters with kernel sizes $1 \times 1$ and $3 \times 3$ respectively. The second operation is another depth-2 layer with 64 and 128 filters with kernel sizes $1 \times 1$ and $5 \times 5$ respectively.
<table>
<thead>
<tr>
<th>Layers</th>
<th>no. of filters, filter size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spatial Convolution</td>
<td>48,3</td>
</tr>
<tr>
<td>Maxpool</td>
<td>2,2</td>
</tr>
<tr>
<td>ReLU</td>
<td></td>
</tr>
<tr>
<td>Spatial Convolution</td>
<td>64,3</td>
</tr>
<tr>
<td>Maxpool</td>
<td>2,2</td>
</tr>
<tr>
<td>ReLU</td>
<td></td>
</tr>
<tr>
<td>Spatial Convolution</td>
<td>128,3</td>
</tr>
<tr>
<td>ReLU</td>
<td></td>
</tr>
<tr>
<td>Spatial Convolution</td>
<td>256,3</td>
</tr>
<tr>
<td>ReLU</td>
<td></td>
</tr>
<tr>
<td>Spatial Convolution</td>
<td>256,3</td>
</tr>
<tr>
<td>Maxpool</td>
<td>2,2</td>
</tr>
<tr>
<td>ReLU</td>
<td></td>
</tr>
<tr>
<td>Spatial Convolution</td>
<td>512,3</td>
</tr>
<tr>
<td>Spatial Convolution</td>
<td>512,3</td>
</tr>
</tbody>
</table>

Table 2: Tiny-VGG architecture with number of filters and filter size in each layer. An entry *Spatial Convolution 512,3* denotes that a convolution layer with 512 filters of kernel size 3 × 3 is used.
Table 3: Tiny Darknet architecture with number of filters and filter sizes in each layer. An entry *Spatial Convolution 512,3* denotes that a convolution layer with 512 filters of kernel size \(3 \times 3\) is used.

<table>
<thead>
<tr>
<th>Layers</th>
<th>no.of filters,filter size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spatial Convolution</td>
<td>16,3</td>
</tr>
<tr>
<td>Maxpool</td>
<td>2,2</td>
</tr>
<tr>
<td>ReLU</td>
<td></td>
</tr>
<tr>
<td>Spatial Convolution</td>
<td>32,3</td>
</tr>
<tr>
<td>Maxpool</td>
<td>2,2</td>
</tr>
<tr>
<td>ReLU</td>
<td></td>
</tr>
<tr>
<td>Spatial Convolution</td>
<td>64,3</td>
</tr>
<tr>
<td>Maxpool</td>
<td>2,2</td>
</tr>
<tr>
<td>ReLU</td>
<td></td>
</tr>
<tr>
<td>Spatial Convolution</td>
<td>128,3</td>
</tr>
<tr>
<td>ReLU</td>
<td></td>
</tr>
<tr>
<td>Spatial Convolution</td>
<td>256,3</td>
</tr>
<tr>
<td>ReLU</td>
<td></td>
</tr>
<tr>
<td>Spatial Convolution</td>
<td>512,3</td>
</tr>
<tr>
<td>Maxpool</td>
<td>2,2</td>
</tr>
<tr>
<td>ReLU</td>
<td></td>
</tr>
<tr>
<td>Spatial Convolution</td>
<td>512,3</td>
</tr>
<tr>
<td>Spatial Convolution</td>
<td>512,3</td>
</tr>
<tr>
<td>Spatial Convolution</td>
<td>1024,3</td>
</tr>
</tbody>
</table>


Figure 2: Classification accuracy (top-1) on MNIST with CSCNN and multiscale patching vs. number of projections. The accuracy improves using a higher number of projections; the performances of the two 2D techniques are best in all cases.
Figure 3: Classification accuracy (top-1) on MNIST with CSCNN and multiscale patching vs. number of training samples. The classification performance increases by 20% using 1000 samples instead of 500 for training. A further 5% increase is noticed increasing the number of training samples above 1000.
Table 4: Verification accuracies (Top 1, Top 5 recognition rates referred as RR) and F1 score using different patch processing techniques. The obtained accuracy scores are averaged over all the classes. These results are obtained using all training samples from each class and 96 patches per image.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Top-1 RR</th>
<th>F1 Score</th>
<th>Top-5 RR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interpolation</td>
<td>69</td>
<td>70</td>
<td>74</td>
</tr>
<tr>
<td>CS-CNN [14]</td>
<td>73</td>
<td>73</td>
<td>79</td>
</tr>
<tr>
<td>Dynamic multiscale patching (DMS)</td>
<td>74</td>
<td>74</td>
<td>80</td>
</tr>
<tr>
<td>2DD</td>
<td>74</td>
<td>74</td>
<td>81</td>
</tr>
<tr>
<td>2DSCNN</td>
<td>76</td>
<td>75</td>
<td>82</td>
</tr>
<tr>
<td>2DDMS</td>
<td>77</td>
<td>75</td>
<td>82</td>
</tr>
</tbody>
</table>

we demonstrate the effects of different kinds of image resizing for Openface, then we perform some weight dimension reductions on VGG and Darknet. We need an input image of size $96 \times 96$. Hence, we take 96 (i.e. $8 \times 12$) image patches of size $18 \times 18$. To implement CS-CNN [14] we resize our image to $216 \times 144$ to get exactly 96 patches. For our dynamic multiscale patch generation we can use a number of scales, although the number of patches is still fixed to 96. In this case we use 3 different scales, $18 \times 18$, $36 \times 9$, and $12 \times 27$. Depending on the input resolution we first get as many patches as possible with aspect ratio 1 : 1, 1 : 4, or 1 : 2.25. We then use the remaining ones till the number of required patches is reached. According to this rule, the smaller the image size, the larger number of scales are employed. It is also possible to use at least a few patches per scale irrespective of the image size. For example, suppose we have 90 patches with size $18 \times 18$. We can now gather the remaining 6 from the $12 \times 27$ set. Or, we can take 6 $36 \times 36$ patches, lowpass filtered and downsampling by a factor 2, which will also generate the required component size of 324 in our case.

Some results are provided in Table 4. We observe that all the compression techniques show some performance improvement compared to standard interpolation, without any additional computation time during inference. The projection weight computation during training using the entire dataset (67k samples) takes approximately 3 seconds. 2DDMS achieves 8% accuracy improvement over normal interpolation-based resizing. Also, 2D decomposition is better than 1D and multiscale patches are better than normal patches. For example, both CS-CNN and our proposed Dynamic multiscale patching show 3% improvement using 2D decomposition. Many parameters variations can
Table 5: Effect of patch generation type for recognition accuracy (top 1 recognition rates) using Dynamic multiscale patching and 2DDMS patch processing. Minimum required patches are marked as $p_b$. These are initially generated from the whole image without any stride. $p_{bd}$ refers to a combination of $p_b$ and downsampled patches; $p_b$ and overlapping patches combination are referred as $p_{bo}$; minimum, downsampled and overlapping patches together are referred as $p_{bdo}$. Increasing patch variety and number improves feature representation and hence recognition accuracy.

<table>
<thead>
<tr>
<th></th>
<th>$p_b$</th>
<th>$p_{bd}$</th>
<th>$p_{bo}$</th>
<th>$p_{bdo}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMS</td>
<td>74</td>
<td>75</td>
<td>75</td>
<td>79</td>
</tr>
<tr>
<td>2DDMS</td>
<td>77</td>
<td>77</td>
<td>78</td>
<td>85</td>
</tr>
</tbody>
</table>

influence the results to some extent. A few factors are discussed below.

The number of images we use to compute the projection matrices and the number of patches we select for each image of course affect the performances of the system. The method we propose of dynamic multiscale patching creates no practical restrictions for the number of patches we can use. More patches promote data augmentation and improve network performance. In case of 2D projection we need to fix the number of patches $G$ we use for each image; for 1D projection instead we can use different numbers of patches per image, although the total number (96) of compressed image patches should be fixed. Table 5 demonstrates the impact of different types of patch selection. We also tried clustering the patches, in order to include only the least correlated data subsets, but this proved to be scarcely effective. To accommodate different numbers of patches we take patches to match the minimum requirement $p_b$; $p_b$ and downsampled patches are referred to as $p_{bd}$; $p_b$ and overlapping patches are $p_{bo}$; all are $p_{bdo}$.

6.4. Weight Initialization

To illustrate the effect of the weights decomposition we perform a classification using VGG-Face and its tiny version shown in Table 2. We use a decomposed version of the original VGG weights for the tiny network using Equation 8. Features are extracted from the second last layer of VGG/Darknet and the last layer of the tiny networks, and are fed to an SVM classifier. As expected, the original VGG and Darknet perform better than the speeded up original networks and than the tiny-networks (Tables 2,3). Results for the tiny networks are generated by using weights initialized from the original networks with and without modular finetuning. These
<table>
<thead>
<tr>
<th>Architecture</th>
<th>Dataset</th>
<th>Top 1 RR</th>
<th>Top 5 RR</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG</td>
<td>WebCasia</td>
<td>78</td>
<td>86</td>
</tr>
<tr>
<td>VGG 4x (section 3.2)</td>
<td>WebCasia</td>
<td>73</td>
<td>80</td>
</tr>
<tr>
<td>Darknet</td>
<td>Pascal VOC</td>
<td>79</td>
<td>84</td>
</tr>
<tr>
<td>Darknet 4x (section 3.2)</td>
<td>Pascal VOC</td>
<td>74</td>
<td>80</td>
</tr>
<tr>
<td>tiny-VGG D</td>
<td>WebCasia</td>
<td>72</td>
<td>77</td>
</tr>
<tr>
<td>tiny-VGG D+MF</td>
<td>WebCasia</td>
<td>77</td>
<td>81</td>
</tr>
<tr>
<td>tiny-Darknet D</td>
<td>Pascal VOC</td>
<td>74</td>
<td>79</td>
</tr>
<tr>
<td>tiny-Darknet D+MF</td>
<td>Pascal VOC</td>
<td>77</td>
<td>82</td>
</tr>
</tbody>
</table>

Table 6: Verification accuracies (Top 1 and Top 5 recognition rates (RR)) using different network architectures. VGG 4x refers to a speeded up VGG architecture. tiny-VGG and tiny-Darknet (Tables 2, 3) are generated using weight projections from the original architecture. Tiny-VGG D denotes the architecture with weights projection without any finetuning while tiny-VGG D+MF denote module-finetuning after weights projection. Results are without end-to-end fine-tuning (wft) and look promising. It is expected that they will slightly improve with end-to-end fine-tuning (ft). However, the tiny networks with module finetuning give slightly better results than the speeded up original network. It should be noted that with our weight decomposition initialization and module finetuning, for 10 epochs and with just 67,000 images (in case of VGG), we are able to achieve acceptable accuracy (almost equivalent to the original VGG trained with 2.6 million images). Hence it is possible to inherit weights of a large network, trained with millions of images, and finetune the weights module-wise, using a small dataset (67,000 in our case). The module finetuning takes approximately 15 minutes on CPU. This result strengthens the fact that modular finetuning after weight initialization from a larger network is beneficial compared to retraining a smaller network. Training any similar small network from scratch would take a much longer time (a couple of days on equivalent hardware) and many more images (approximately 1 million) to achieve an acceptable performance.

It should be noted that the 4x speedup (obtained with a pruned network which has inference time 4 times less than the original) is determined considering only the layers which we used for features. It is observed that as we use a lower number of components for decomposition or approximation the time reduces at the cost of accuracy. For example, the performance of the original VGG (i.e. 78%) reduces by 5% and 10% when we reduce the
weights such that the inference time is speeded up by 0.4 sec and 1.4 sec respectively. However, we get an inference time of 0.9 sec (2.2 seconds faster than the original VGG time) when we use our reduced network with weight projections (tiny-VGG D) with almost similar accuracy (72%) as VGG 4x performing in 2.7 sec. Our objective here is to obtain similar performance (exact or within 2 − 2.5%) with a much smaller network derived from the original bigger network. That is, we focus more on applications where inference time plays a significant role. The best results shown in Table 6 are those using the architectures shown in Tables 2 and 3. If we further reduce the number of elements, the accuracy starts deteriorating in an unpredictable way. It should be noted that the time figures are obtained using CPU (GPU is not used), 12GB RAM, Torch environment.

6.5. Module based Finetuning

In this subsection we demonstrate the role of the weight initialization we obtained from the projection technique. We infer that there is a huge difference when using this, compared to random initialization. To support this inference we utilize a t-SNE representation of extracted features from the networks using the original net, a randomly initialized small net, a decomposed weight initialization, and a module-wise finetuned network. t-SNE clustering is used to evaluate the quality of the extracted features. It is seen that the features from the original network generate distinct clusters. Same is the case for our decomposed weight initialization and the module-wise finetuned network. However, the randomly initialized network does not generate distinct clusters. This emphasizes the fact that we are actually able to transfer weight information from a large network using the decomposition procedure. This is shown in Figures 5 to 8, and verifies that the initialized weights do play a role. For clarity of clustering perception the t-SNE results are shown using five randomly selected classes (3, 5, 19, 28, 35) from the CASIA dataset. Other random selections have also been tested and it was noted that they perform similarly. The results in Figures 7 and 8 depict distinct clusters, which justifies the usefulness of the reduced weights. It may however be noted that some outliers and subclusters show up: they are dealt with using suitable threshold selections or network certainty measures. As mentioned before, we use a module based finetuning to further improve the results of the initial decompositions. This can be ignored if the number of layers in the original network is the same as the small network but the filters in each layer are less. However, in our case we use a smaller number of
layers compared to the original networks. Hence, we prefer using finetuning. For this purpose we treat the network as a series of modules. For example, the first five convolutional layers are segregated as two modules as shown in Figure 4. Once the network is initialized with decomposed weights, we finetune each module with the transferred targets from the original network using the loss function as per Equation 10. For example, the left module in Figure 4 is finetuned using the output of the second convolutional layer of the original network. To enable this action, the modules are generated such that their output size matches even if intermediate sizes are different. In this case, both the tiny and the original network generate $224 \times 224 \times 64$ feature vectors. Figure 9 highlights a significant loss with respect to the second last layer VGG vector output using different numbers (10k, 20k, 30k) of training samples. We use 31 test batches consisting of 3100 images for the test loss computation. All the modules were trained for 10 epochs only.

The finetuning iterations exploit a smooth absolute distance criterion to prevent exploding gradients. That is, Equation 10 is substituted with Equation 11 when the absolute difference is less than 1:

\[
W_t = W_{t-1} - \alpha \sum_i 0.5(|x_i - y^o_i| + |x_i - y^c_i|) - 0.5 \tag{10}
\]

\[
W_t = W_{t-1} - \alpha \sum_i 0.5(0.5(|x_i - y^o_i| + |x_i - y^c_i|))^2 \tag{11}
\]

In these equations $x_i$ is the estimated output, whereas $y^o_i$ and $y^c_i$ respectively are the outputs from the original and compressed network versions. The smooth absolute difference [24] is also less sensitive to outliers [5]. We also prefer to use a soft learning, realized by estimating the error as the average of the errors between estimated and original matrices, and estimated and compressed matrices. $\alpha$ gives the learning rate.

When the first module is finetuned there are no $y^c_i$. From the second module on, $y^c_i$ is the output from the corresponding layer of the original net using $x_i$ of the previous finetuned module as input.

7. Discussion

All experiments have been conducted using Python 3.6 with Pytorch 1.2 deep learning framework. In terms of accuracy, it is observed that the proposed 2D multiscale patch generation approach for classification provides an
Figure 4: The figure shows 2 sample modules of the tiny VGG network. These modules are finetuned independently post weight projections from a bigger network. The first module (left) consists of 2 filter sets while the second module (right) has 3 filter sets.

Figure 5: Clusters for 5 different classes (Class 3, 5, 19, 28, 35 randomly selected) from CASIA are plotted with features extracted from second last layer of VGG network using original VGG weights. t-SNE is used to plot the feature dimension in 2D space. Three different distance measures, i.e. Cosine, Chebychev and Euclidean, are used. A prominent cluster distinction indicates quality feature representation.
Figure 6: Clusters for 5 different classes (Class 3, 5, 19, 28, 35 randomly selected) from CASIA are plotted with features extracted from last layer of tiny-VGG using random initialization. The scattered 2D feature points plotted using t-SNE depict no learning as expected in this case.
Figure 7: Clusters for 5 different classes (Class 3, 5, 19, 28, 35 randomly selected) from CA-SIA are plotted with features extracted from tiny-VGG using Decomposed Weights from VGG. The 2D feature points depict that the feature space derived from these decomposed weights yield good representation and distinct clusters.
Figure 8: Clusters for 5 different classes (Class 3, 5, 19, 28, 35 randomly selected) from CASIA are plotted with features extracted from the module-wise finetuned network weights for tiny-VGG. The 2D feature points depict that the feature space derived from decomposed weights further improves feature representation after finetuning.
Figure 9: Test Loss measured using features difference from original network and features from weight projected network. It is seen that feature representations have lower deviations when module-finetuned. Results of 31 different test batches are shown here. The module is finetuned with different numbers of training images, marked as DS10K, DS20K, DS30K denoting 10K, 20K, and 30K images respectively.
8% increase in Top-1 and Top-5 recognition rates, and 5% increase in F1 score over general interpolation-based resizing. It obtains 4%, 2%, 3% improvement in Top-1 and Top-5 rates and in F1 scores respectively over the CSCNN technique. The dynamic multiscale patching can be used without resizing images. Also, there is no considerable increase in inference time due to the patch generation phase prior to network feeding. In case of finetuning a small network with projected weights of a larger network, the performance of the original VGG architecture (i.e. 78%) reduces by 5% if we reduce the weights by an amount such that the inference is speeded up by 0.4 sec. However, we get an inference time of 0.9 sec (2.2 seconds faster than the original VGG time) when we use our reduced network with weight projections (tiny-VGG D) with almost similar accuracy (72%) as VGG 4x. When tiny-VGG D is finetuned (tiny-VGG D+MF ) we get a 77% accuracy in the same inference time, i.e. 0.9 sec. That is, with respect to the original VGG architecture, the speed is increased by a factor 3.44 ( (2.2+0.9)/0.9 ) while the accuracy remains practically the same. The finetuning is done with a dataset of 67k images for 10 epochs (15 min on a CPU), which is a significant improvement in terms of training time and dataset size when compared to training from scratch.

8. Conclusions and Future Work

A novel multiscale-based patch selection technique is proposed in this work, and is used as an alternative approach to resize the input according to the network requirements. This solution can be exploited for input layer compression or for feature enhancement, according to the specific application. Further, it is shown that using a 2D decomposition instead of 1D decomposition improves the performances of the proposed technique. The same reduction method is also used for weight initialization from a layer with a higher number of parameters to a layer with a lower number of parameters. The weight initialization methods presented in this work are currently being used along with soft targets as a modular update approach to transfer the knowledge from a big network to a smaller one.

Developments left as future work entail selecting a fixed number of patches from every spatial and wavelet scale in case of multiscale patch generation. Also, selecting patches from effective positions which constitute the dominant features of the image, rather than from random positions, should prove
beneficial even though it will imply a large preprocessing load to determine the prominent regions. Finally, the results shown for weights initialization are obtained by extracting features from the network and training an SVM classifier. In order to use the network end-to-end (i.e., to use the neural network classification layer instead of an SVM) we require a set of weights for each layer instead of one, so that a full-fledged 2D decomposition can be carried out. As only a single weight set was currently available, the current results do not exploit this. We need to train a large network with different sets of data, and use the weight matrices. Work is in progress to further expand these results.

References


[8] Shaoqing Ren, Kaiming He, Ross Girshick, Xiangyu Zhang, Jian Sun, Object Detection Networks on Convolutional Feature Maps, Transactions on Pattern Analysis and Machine Intelligence, IEEE, July 2017


[14] Yiran Shen, Tao Han, Qing Yang, Xu Yang, Yong Wang, Feng Li and Hongkai Wen, CS-CNN: Enabling Robust and Efficient Convolutional Neural Networks Inference for Internet-of-Things Applications, Special Section On Multimedia Analysis For Internet-Of-Things, IEEE 2018.


[16] Yunchao Gong and Svetlana Lazebnik, Iterative Quantization: A Pro-custean Approach to Learning Binary Codes, CVPR 2011


