FAWCA: A Flexible-greedy Approach to find Well-tuned CNN Architecture for Image Recognition Problem

Md Mosharaf Hossain[1], **Douglas A. Talbert**[1], **Sheikh K. Ghafoor**[1], **and Ramakrishnan Kannan**[2]
[1]Computer Science, Tennessee Technological University, Cookeville, TN, USA
[2]Computational Data Analytics, Oak Ridge National Laboratory, Oak Ridge, TN, USA

Abstract— *Convolutional Neural Networks are currently being used widely because they show competitive performance in several problem domains such as image detection, natural language understanding and signal processing. However, a complicated and cache inefficient CNN architecture might show good accuracy but typically requires long processing time and a larger volume of main memory. In this paper, we develop a greedy algorithm with flexible selection that efficiently tunes several hyper-parameters particularly the number of filters for a particular convolutional layer and the number of convolutional layers. Thus, our approach finds a CNN architecture by minimal exploitation of computing resources and time for any particular image classification problem. The important aspect of the algorithm is that it can find different well-tuned solution to different image datasets. The proposed CNN architectures generated by the algorithm show competitive performance in several areas of image recognition. We compare our Flexible-greedy approach with a Brute-force parameter tuning technique and find that our approach takes much less time finding well-tuned CNN architecture compared to Brute-force technique. In the case of finding minimal number of trainable parameters, our approach is also competitive with the general Brute-force methods.*

Keywords: CNN, Flexible-Greedy, Brute-force, CIFAR-10, Fashion-MNIST [1]

1. Introduction

At the time when CNN was introduced in the paper [1], the computing power was not sufficient to run a deep architecture of the network. When the LeNet-5 was developed, it had very few layers including only three convolutional layers [2]. A comparatively larger CNN architecture was AlexNet [3] that consists of eight layers including five convolutional layers. After AlexNet, Machine Learning researchers found that network depth plays crucial role in improving accuracy for image recognition problems [4], [5]. Moreover, they observed that building a deeper model with ReLU activation is comparatively less expensive compared to previous networks where $sigmoid$ and $tanh$ were commonly used as activation units. As a result, they started to build very deep models in several problem domains particularly in image classification and object detection problems. Such deep models [6], [7], [8] showed improved accuracies in image classification but suffer from having very large parameter sets to tune. GoogleNet [9] introduced "Inception module" in their deep network. This change helps reduce the number of parameters and can be used to build very deep CNNs. This technique was applied in the ILSVRC 2014 classification and detection challenges where their solution surpassed other state-of-the-art techniques. Another important obstacle that the deep networks suffer from is vanishing gradient [10]. This problem increases as the network grows and hinders the network from convergence. The probable solution to address this problem involves normalized initialization [8] and batch normalization (BN) techniques [8]. BN helps to reduce the impact of previous layers by keeping fixed mean and variance. This results in the intermediate layers being independent from each other and convergence becomes comparatively faster. As described in [11], when the depth of a network increases, accuracy gets saturated and after that the accuracy starts to degrade rapidly. The paper suggests that the problem is not caused by overfitting. This issue arises when more layers are added to a suitably deep model. He et al. solve the degradation problem by adding identity mapping technique [11]. Their solution network gives fewer training errors compared to it's shallower counterpart. A recent paper [12] published in 2017 talks about a different type of CNN architecture named DenseNets, which connects each layer to every other layers in a feed-forward fashion. The paper claims that DenseNets reduces the vanishing-gradient problem, strengthen feature propagation and reduces number of parameters significantly. All the popular CNN architectures including LeNet [2], AlexNet [3], VggNet [6], GoogLeNet [9], ResNet [11] and DenseNets [12] are developed using hand-crafted tuning. Tuning a CNN architecture such as finding optimal number of filters for convolution layers for a particular image recognition problem is cumbersome and

resource intensive. We aim to develop an algorithm that searches for the best possible filter size for any convolution phase. Our algorithm also finds a suitable depth for the CNN at which point it shows better accuracy.

2. Related Work

Finding an optimal CNN architecture for a particular image classification problem is a NP-hard problem. No fruitful research has been conducted yet to find an optimal CNN architecture automatically for any image classification problem. Recent development of CNN architectures [9], [11], [12], [13], [14], [5], [15], [16] improves accuracy of computer vision problem by adding more convolutional layers, but all these works needed to put huge effort in finding out a suitable maximum bound of intermediate layers. Furthermore, all these popular networks required careful examination regarding the number of filters for intermediate convolutional layers to reduce overfitting. Thus, tuning the hyper-parameters of filter size and number of convolutional layer becomes an important job. Presently, CNN networks have also started to incorporate dropout [17] and batch normalization [8] techniques to reduce variance in the network. These techniques add some more hyper-parameters with the CNN's default hyper-parameters. Therefore, it becomes difficult to tune all the hyper-parameters in respect to time and resource if a commonly used greedy approach is applied. Our paper aims to describe a flexible greedy approach to find well-tuned number of filters for a convolutional layer and the total number of convolutional layers that best suit to a particular image classification problem.

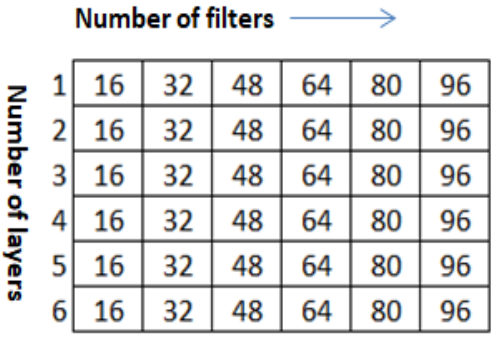

Fig. 1: Brute-force approach to find well-tuned CNN architecture.

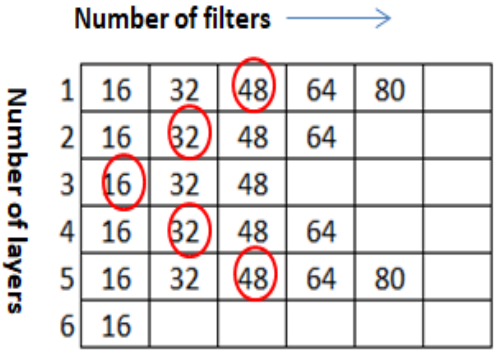

Fig. 2: FAWCA approach to find well-tuned CNN architecture.

3. FAWCA Overview

The FAWCA approach we present here can automatically select an approximately optimal number of filters for a particular convolutional (conv) layer. In a generic CNN algorithm, as we increase the number of filters in a conv layer, the complexity is supposed to improve. That means the network can capture more non-linearity in the data. However, increasing filter count likely increases variance in the network. As a result, the network starts over-fitting for a undiscovered or test dataset. To find well-tuned CNN architectures, FAWCA applies two loops in its algorithm that searches for the best choices. The outer loop finds the number of layers whereas the inner loop finds the number of filters for a conv layer. The inner loop starts with a given initial number of filters such as 16 or 32 and goes up-to a certain multiplier of the initial filter count. In the inner loop, the number of filters used in the current iteration is the number of filters in the previous iteration incremented by the size of initial filter.

Brute-force search explores all possible choices of filters in a grid space shown in Figure 1. It starts to find well-tuned filter for the first conv layer from a given minimum number of filters and checks all the way until it reaches a given max number of filters. After finding well-tuned filter for the first layer, it repeats the process until it finds a well-tuned filter count for the second conv layer. The outer loop breaks if the accuracy does not improve for any iteration in previous layer before starting to find filters for a new layer. On the other hand in FAWCA, in the inner loop, it stores the current accuracy as the best accuracy if it is greater than the previous iteration. Otherwise, it checks the current accuracy with the accuracy found in the iteration in two steps backward. Thus, unlike the hard greedy way to stop the current loop, FAWCA checks two consecutive iterations while finding well-tuned filter counts for a conv layer. If both the consecutive iterations show a performance drop compared to the previous best accuracy, the current inner loop does not run further and enters to next outer loop iteration. An example of FAWCA approach is shown in Figure 2. The filters shown in red circle are the well-tuned size of selected filters for each conv layer.

In pseudo-code 1, we present the Find_Sub_Opt_CNN procedure of the FAWCA approach. The procedure consists of two *for* loops. The outer loop iterates from layer 1

Algorithm 1 FAWCA Algorithm

procedure FIND_SUB_OPT_CNN($net\_image, cur\_img\_dim, base\_f, max\_multip, conv\_dropout\_rate, dense\_dropout\_rate$)
  $opt\_acc \leftarrow 0.0$
  $l \leftarrow 1$
  $i \leftarrow 0$
  $flag \leftarrow 0$
  $opt\_i \leftarrow 0$
  $net \leftarrow$ `NULL`
  $net\_temp \leftarrow$ `NULL`
  $opt\_net \leftarrow$ `NULL`
  $saved\_opt\_conv\_net \leftarrow net\_image$
  $saved\_sub\_conv\_net \leftarrow$ `NULL`
  $acc\_list \leftarrow$ `set 0.0 to all the elements`
  for $l \leq L$ **do**
    $cur\_img\_dim \leftarrow cur\_img\_dim/2$
    for $f \in \{base\_f, 2 * base\_f \ldots, max\_multip * base\_f\}$ **do**
      $net\_temp \leftarrow$ `Add a conv layer with filter size` f `to the` $saved\_opt\_conv\_net$ `network`
      if $cur\_img\_dim \geq min\_img\_dim$ **then**
        $net\_temp \leftarrow$ `Add a MaxPool layer to the` net_temp `network`
      if $conv\_dropout\_rate \neq 0.0$ **then**
        $net\_temp \leftarrow$ `Add Dropout with` $conv\_dropout\_rate$ `to the` net_temp `network`
      $net \leftarrow$ `Flatten the` net_temp `Network`
      $net \leftarrow$ `Add Dense layer with` $dense\_layer\_units$ `to the current network`
      if $dense\_dropout\_rate \neq 0.0$ **then**
        $net \leftarrow$ `Add Dropout with` $dense\_dropout\_rate$ `to the current network`
      $acc \leftarrow$ `Measure accuracy with cross validation for the current network`
      $acc\_list[i] \leftarrow acc$
      if $acc\_list[i] > opt\_acc$ **then**
        $opt\_acc \leftarrow acc\_list[i]$
        $opt\_net \leftarrow net$
        $saved\_sub\_conv\_net \leftarrow net\_temp$
        $flag \leftarrow 1$
        $opt\_i \leftarrow i$
      else
        if $i \geq 2$ **then**
          if $i \geq opt\_i + 2$ **then**
            `Break the inner for loop`
      $i \leftarrow i + 1$
    if $flag == 1$ **then**
      $saved\_opt\_conv\_net \leftarrow saved\_sub\_conv\_net$
    else
      `Break the outer for loop`
  return opt_net

through layer L. The inner loop selects the best count size for a particular outer iteration. We use max pooling after the convolutional layer if the current image dimension is greater than a minimum dimension such as 6×6 or 8×8. The max pooling layer usually reduces the image dimension to half resulted from a conv layer. Thus, we restrict it by setting a variable of minimum image dimension that limits the current image from downsizing further after reaching that minimum dimension. The algorithm also has the flexibility to use dropout technique both in the conv layer and dense layer. When a particular image classification problem starts to show over-fitting on test data, this dropout technique can reduce the network complexity by dropping out some units from the conv layer or the dense layer based on the parameters $conv\_dropout\_rate$ and $dense\_dropout\_rate$ respectively. At this point, the procedure measures accuracy for the current network using a k-fold cross-validation technique. The pseudo-code 2 presents a procedure $Find\_CV\_Accuracy$ that calculates the mean accuracy over the k-folds. As the original image dataset might be very large, so in a particular iteration over the k iterations, the procedure takes a sample dataset from the original data by stratified sampling. The sample dataset again gets split into training and test datasets by the stratified sampling. Stratified sampling keeps same ratio of the target classes in the sampled classes. Thus, it helps keep homogeneity in the sample dataset if the original dataset is unbalanced over classes. Finally, the procedure measures mean value over the k accuracies.

Now, if this accuracy is greater than the previous best accuracy, FAWCA replaces that accuracy with the current accuracy. The previously stored best network configuration is also replaced with the current network. On the other hand, if the current accuracy does not improve more than the previous best accuracy and the current iteration is two more than the iteration for that best accuracy, then the algorithm breaks the inner loop. This is how FAWCA allows two more iterations after the iteration of previous best accuracy for seeking improvement in the inner loop. On the other hand, the Brute-force approach checks all possible filters to find best number of conv filters in the inner loop. The Brute-force pseudo-code is not shown in the paper, but can be obtained by omitting line 33 through 36 from the $Find\_Sub\_Opt\_CNN$ procedure.

4. Experimental Design

4.1 Data

We use two publicly available image datasets in our experiment. The First dataset is CIFAR-10 that consists of $60,000$ color images in 10 different classes. Each class contains 6000 images and the dimension of each image is 32×32. In this large dataset, number of training image is $50,000$ and the remaining images are test dataset. The second dataset we use is Fashion-MNIST that contains gray-scale images. Number of training and test images in this dataset are $60,000$ and $10,000$ respectively. This dataset is very similar to the MNIST dataset where each image has dimension 28×28 and consists images of 10 distinct fashion type classes. Before fitting data into the network model we divided the pixel value by 255 following [18] to get the pixel value in [0, 1] range.

4.2 Machine Details

All the experiments are performed on a compute node in a four-node cluster setup. The compute node consists of two Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80GHz processors using NUMA architecture. The compute node also has one NVidia K40 GPU card. The Keras [19] Deep Learning python library with Tensorflow backend is used in our experiment. The experiment uses the K40 GPU card to leverage the streaming processors. For a particular image dataset, the computation time is measured by summing the time required to find a well-tuned CNN architecture and the time required to fit the whole training data with the selected well-tuned network.

5. Experimental Results and Analysis

In this section, we present an analysis of the time and memory complexity of FAWCA approach with the Brute-force technique. The comparison of well-tuned CNN architecture and classification results between FAWCA and Brute-force are also described for both image datasets.

5.1 CIFAR-10

Before feeding CIFAR-10 data into FAWCA, we set several hyper-parameters. To reduce over-fitting, we used L2-regularization in conv and dense layers and set the weight decay parameter to 0.001. Along with L2-regularization, we use $Dropout$ both in conv and dense layer and the percentage of $Dropout$ rates are 20% and 40% respectively. We set the maximum number of layers to 6 and started with 16 as initial filter count. The filter dimension is set to 5×5 across all the Conv layers and maximum multiplier for a single Conv layer is set to 6. CNN is computationally expensive, hence we used 4-fold cross validation with 40% stratified sampling each time. Number of epoch and batch size are set to 5 and 128 respectively. The algorithm has options to take a completely different hyper-parameter setting.

Figure 3 shows the well-tuned CNN architectures generated by FAWCA and the Brute-force approaches. We observe that to generate a well-tuned solution, FAWCA does not necessarily explore more conv layers compared to Brute-force method.

The performance result in Table 1 shows that FAWCA requires 57.3% less computation time to find a well-tuned solution for CIFAR-10 dataset. The total number of trainable parameters require for FAWCA is 1.6 millions compared to

Algorithm 2 FAWCA Algorithm

procedure FIND_CV_ACCURACY($folds, X, y, batch\_size, epoch$)
  for $k \leq folds$ **do**
    $tr\_X \leftarrow$ `Sample from` X `by stratified sampling`
    $tr\_y \leftarrow$ `Sample from` y `by stratified sampling`
    `split` tr_X `and` tr_y `into training and test datasets by stratified sampling`
    `Fit a model using Adam optimizer, batch_size and epoch on the new training dataset`
    `Measure accuracy on the new test dataset`
  $cv\_acc \leftarrow$ `Take mean over all the` k `accuracies`
  return cv_acc

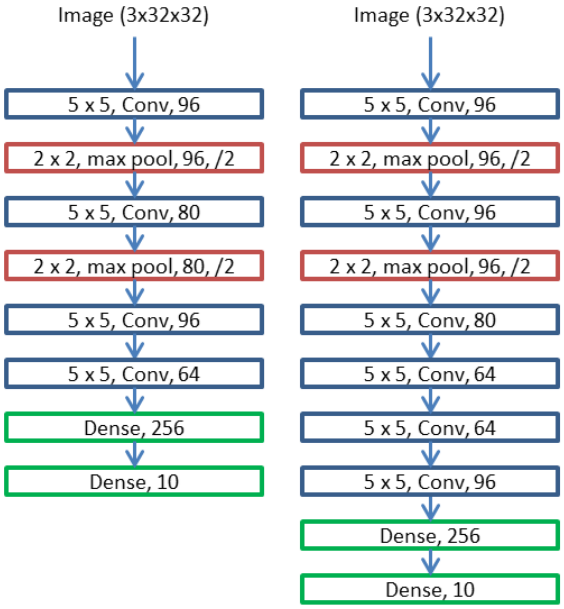

Fig. 3: *Well-tuned CNN architecture for CIFAR-10 dataset. Left: This network is generated by FAWCA. Right: This CNN is generated by Brute-force Approach.*

2.4 millions for Brute-force approach. To store the parameters thereby, FAWCA saves 49.6% memory than the Brute-force. After generating the well-tuned CNN architecture by both approaches, we trained those networks with full training data. We measured accuracy on the $10,000$ test data using the well-tuned trained models. The accuracies found for FAWCA and Brute-force are 81.86% and 81.49% respectively. This result shows that our FAWCA approach does not compromise the accuracy but gives better performance with respect to running time and number of parameters.

5.2 Fashion-MNIST

In case of Fashion-MNIST, the over-fitting problem is not as high as with CIFAR-10, hence the weight decay parameter is set to 0.0001. The number of folds, size of the base conv filters, filter multiplier of a conv layer, $Dropout$ rate and size of batch are kept same as the case of CIFAR-10. As the size

Table 1: *Performance of FAWCA compared to Brute-force on CIFAR-10 dataset.*

Performance measures	**FAWCA**	**Brute-force**
Running Time (min)	112.87	177.52
Trainable parameters (in mil)	1.6	2.4
parameters memory size (MB)	6.09	9.11
Accuracy on test data (%)	81.86	81.49

of the image in this data is smaller than CIFAR-10(28×28), filter dimension is changed to 3×3. The depth of the dense layers and total number of conv layers hyper-parameters are changed to 512 and 8 respectively. However, all the hyper-parameters have the same value when applied to Brute-force algorithm to find well-tuned solution. Figure 4 shows the well-tuned solutions generated by FAWCA and the Brute-force approaches. In Figure 4, the network in left image by FAWCA only requires four conv layer to show better accuracy compared to the right image by Brute-force that explores all the eight conv layers to find optimal solutions.

We present a performance table 2 comparing several measures between the FAWCA and Brute-force approaches. FAWCA requires only 96.89 minutes to find a well-tuned solution compared to 208.64 minutes for Brute-force. It shows that FAWCA requires about 116% less time when both the algorithm runs on K40 GPU card. Moreover, the accuracies on test data for FAWCA and Brute-force are 93.41% and 92.76% respectively. Thus, FAWCA again results in higher accuracy for Fashion-MNIST dataset. Finding well-tuned number of filters for two extra conv layers in Brute-force approach adds more time than FAWCA. The long architecture solution in Brute-force increases variance and more likely responsible for the slight accuracy drop. However, one bottleneck to the FAWCA approach is that the solution network needs more parameters to train compared to Brute-force for this particular dataset. As the solution network of FAWCA approach is small, the algorithm does not leverage more max-pooling layers to reduce the final image dimension. Thus, having bigger image dimension by the final conv layer results in bigger parameter size to the well-tuned solution network.

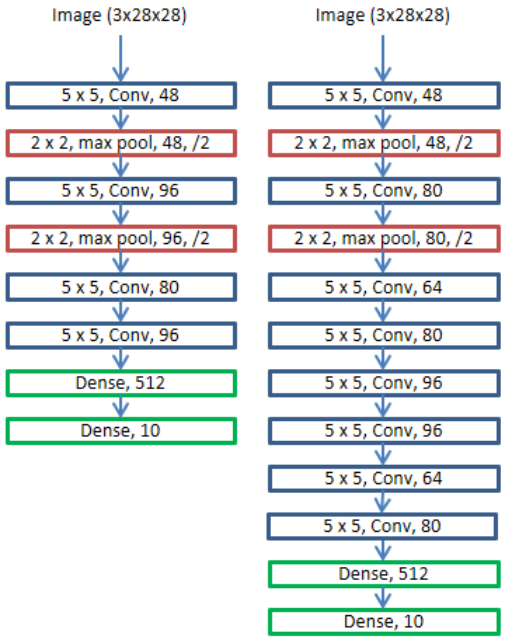

Fig. 4: *Well-tuned CNN architectures for Fashion-MNIST dataset. Left: This network is generated by FAWCA. Right: Larger CNN architecture is generated by Brute-force Approach.*

Table 2: *Performance of FAWCA compared to Brute-force on Fashion-MNIST dataset.*

Performance measures	**FAWCA**	**Brute-force**
Running Time (min)	96.89	208.64
Trainable parameters (in mil)	2.6	2.4
parameters memory size (MB)	9.90	9.13
Accuracy on test data (%)	93.41	92.76

6. Summary and Conclusion

In this paper, we present a Flexible-Greedy approach to find well-tuned CNN architecture for any image classification datasets. We also compare our FAWCA approach with Brute-force technique to find well-tuned CNN. We find competitive performance measures with respect to training time, trainable parameters, and accuracy for the Flexible-Greedy approach both in CIFAR-10 and Fashion-MNIST datasets. In the case of CIFAR-10, FAWCA requires 57% less time to find a well-tuned CNN compared to Brute-force without compromising accuracy. For the Fashion-MNIST dataset, FAWCA also shows competitive performance compared to Brute-force. Thus, the approach we present in this paper has significant contribution in the area of image recognition as it finds well-tuned solution much more efficiently than brute force approach.

References

[1] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[4] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," *arXiv preprint arXiv:1505.00387*, 2015.

[5] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training very deep networks," in *Advances in neural information processing systems*, 2015, pp. 2377–2385.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[8] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.

[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[10] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[12] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, "Densely connected convolutional networks," *arXiv preprint arXiv:1608.06993*, 2016.

[13] K. Greff, R. K. Srivastava, and J. Schmidhuber, "Highway and residual networks learn unrolled iterative estimation," *arXiv preprint arXiv:1612.07771*, 2016.

[14] J. G. Zilly, R. K. Srivastava, J. Koutník, and J. Schmidhuber, "Recurrent highway networks," *arXiv preprint arXiv:1607.03474*, 2016.

[15] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2014, pp. 1725–1732.

[16] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[17] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[18] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.

[19] F. Chollet *et al.*, "Keras: Deep learning library for theano and tensorflow," *URL: https://keras. io/k*, vol. 7, p. 8, 2015.