Constraints on Hyper-parameters in Deep Learning Convolutional Neural Networks

Ubaid M. Al-Saggaf[1]*, Abdelaziz Botalb[2], Muhammad Faisal[3]
Muhammad Moinuddin[4], Abdulrahman U. Alsaggaf[5], Sulhi Ali Alfakeh[6]
Electricaland Computer Engineering Department, King Abdulaziz University, Jeddah 21589, Saudi Arabia[1, 2, 4, 5]
Center of Excellence in Intelligent Engineering Systems (CEIES), King Abdulaziz University, Jeddah 21589, Saudi Arabia[1, 2, 4, 5]
Computer & Information Technology Dept., Dammam Community College
King Fahd University of Petroleum & Minerals, Dhahran 31261, Saudi Arabia[3]
Department of Internal Medicine, Child and Adolescent Psychiatrist
Faculty of Medicine, King Abdulaziz University, Jeddah 21589, Saudi Arabia[6]

***Abstract*—Convolutional Neural Network (CNN), a type of Deep Learning, has a very large number of hyper-meters in contrast to the Artificial Neural Network (ANN) which makes the task of CNN training more demanding. The reason why the task of tuning parameters optimization is difficult in the CNN is the existence of a huge optimization space comprising a large number of hyper-parameters such as the number of layers, number of neurons, number of kernels, stride, padding, rows or columns truncation, parameters of the backpropagation algorithm, etc. Moreover, most of the existing techniques in the literature for the selection of these parameters are based on random practice which is developed for some specific datasets. In this work, we empirically investigated and proved that CNN performance is linked not only to choosing the right hyper-parameters but also to its implementation. More specifically, it is found that the performance is also depending on how it deals when the CNN operations require setting of hyper-parameters that do not symmetrically fit the input volume. We demonstrated two different implementations, crop or pad the input volume to make it fit. Our analysis shows that padding performs better than cropping in terms of prediction accuracy (85.58% in contrast to 82.62%) while takes lesser training time (8 minutes lesser).**

Keywords*—Neural networks; convolution; pooling; hyper-parameters; CNN; deep learning; zero-padding; stride; back-propagation

I. Introduction

Convolutional Neural Networks (CNNs) have proved to be the perfect machine learning choice for a wide range of application fields, such as pattern classification and analysis of video, image, speech, and text (natural language processing). However, no doubt using CNNs requires much work compared to other machine learning solutions such as random forest, Support Vector Machines, etc. This added work is primarily due to the vast optimization space of parameters and hyper-parameters, which interact with each other in a very complex way. Furthermore, making this problem even more complex is that there is still no universal, robust theory that supports hyper-parameters optimization. That would enable us to choose the right hyper-parameters for the right problem at hand and give the best performance with less effort and time. Setting hyper-parameters without robust theory is like working blindly, as quoted by the German philosopher Immanuel Kant:

"Experience without theory is blind, but theory without experience is mere intellectual play".

Deep convolutional neural networks often have numerous layers piled on each other and are taught to do a specific task. At the end of each layer, the network learns a variety of low, medium, and high-level features. There are several papers in the literature with different approaches to setting CNN hyper-parameters, but none of them has presented a generalized and robust systematic approach to the problem. Thus, hyper-parameters optimization is not a problem that is ever entirely solved. This work aims to investigate empirically the impact of two different strategies to deal with input volume in CNN, that is, cropping and padding.

II. The Challenge of Hyper-Parameters Tuning in a CNN

Setting the best CNN hyper-parameters for a particular classification problem could be challenging. If the results are not making any sense or bad accuracy was achieved after the first trial, then there is no prior knowledge about what went wrong; anything of the following could be a reason:

- Activation function for every layer or set of layers.
- Learning rate.
- Momentum.
- Regularization type and factor value.
- Size of filters in a specific layer(s).
- The stride of convolution or pooling in a particular layer(s).
- Padding type (e.g., zeros, first and last values, values repeated cyclically).
- The number of convolution, pooling, and activation layers in the network.
- Order of layers e.g., conv-conv-pool; or conv-activation-pool-conv.

*Corresponding Author.

- Type of cost function.
- The approach to encoding the output is not appropriate.
- Backpropagation gradient implementation is incorrect (sanity check was not done or was not efficient).
- Dataset was not split correctly into proper ratios.
- Not enough datasets for training.
- Imbalanced dataset.
- Not pre-processed the dataset, e.g., not normalized at all or normalized incorrectly.
- Wrong label assignment during training could happen when a new split ratio is needed.
- The performance measure metric is not appropriate for the given problem.
- The mini-batch size is not appropriate for Stochastic Gradient Descent.
- Different approaches to weight initialization.
- Maybe CNN is not the right solution to the problem at hand.

Moreover, often the wrong choice of one hyper-parameter (e.g., sigmoid activation function) will never let the network converge no matter how all other hyper-parameters were chosen. So, it is straightforward to get frustrated and lose in the hyper-parameter space and this makes CNN debugging a challenging task. Some heuristics could be used to set the right hyper-parameters by developing a workflow that enables quick debugging. These heuristics are just rules of thumb, and they are not guaranteed to give the best possible results because the behaviour of the CNN entirely depends on the specific dataset of a particular problem. Also, this is what makes it hard to establish one universal solid theory of defining the appropriate hyper-parameters for any given problem.

A. Related Work

In contrast to the manual search for finding an optimum set of hyper-parameter values, automated search approaches were adopted by many research works, and here we highlight a small portion of them:

In [1], a grid search method and a manual search technique were investigated for neural networks and particularly for deep belief neural networks. According to their findings, the grid search method is not an optimal choice for setting the hyper-parameters of the neural networks. On the other hand, it was found that random search can be used to obtain a baseline result in order to assess the performance of the other hyper-parameter optimization methods. In [2], the issue of hyper-parameters tuning is addressed by formulating the problem as a constrained optimization task. Then, a derivative-free optimization technique is opted for tuning the parameters. As a result, a more accurate and automated technique is developed for tuning of the hyper-parameters. Moreover, it is found that this technique is not consistent to achieve the global optimum. In [3], the Taguchi method was utilized to obtain optimal hyper-parameters which can provide faster training and enhanced classification accuracy. In [4], the authors utilized a Bayesian learning technique by employing Gaussian process-based sampling to develop a learning model for designing the tuning parameters of the algorithm. It was reported that the proposed technique for evaluating the optimum values of hyper-parameters can provide significantly faster learning compared to the learning performance of the baseline methods. In [5], a probabilistic model is developed that can deal with early learning termination in the case a bad performance is detected.

Deep convolutional neural networks often have numerous layers piled on each other and are taught to do the specific task. At the end of each layer, the network learns a variety of low, medium, and high-level features.

Different variants of deep neural networks, including CNN, have been employed on various signal processing and machine learning tasks. However, their claimed performance is specific to a certain problem and dataset. In [6], an evolutionary algorithm was developed for designing of optimal hyper-parameter for different CNNs and their performance was investigated on the MNIST dataset. In [7], the hyper-parameters for the deep neural network were optimized using Nelder-Mead and coordinate search methods. In [8], a bandit-based strategy was proposed to solve the task of hyper-parameter optimization. Datasets CIFAR-10, MNIST, and Street View House Numbers were investigated and it is found that the proposed method is faster than the existing Bayesian optimization algorithms. Another work in [9] developed a faster algorithm for the tuning of hyperparameters by employing the strategy of Boolean functions analysis. Its performance was tested using deep neural networks on the CIFAR-10 dataset, and they proved that the proposed technique has much better performance than the existing baseline methods. In [10], a non-parametric regression model-based adaptive technique was designed to calculate the optimum value of the learning hyper-parameters in a short time. The proposed idea was applied on CNN and it was shown via various experiments that the proposed algorithm performs faster than the existing state-of-the-art techniques. In [11], Bayesian optimization-based generative model was developed to analyse the validation error w.r.t. the size of the training data. Again deep neural networks were utilized to test the performance of this algorithm and it was shown that the proposed method finds better hyper-parameters in lesser time. In [12], a novel technique for optimizing the hyper-parameters was developed which was based on the combination of the input features. The NMA was used with CNN for the CIFAR-10 dataset for testing the performance of the proposed model and it was shown to achieve better classification accuracy. In [13], the Covariance Matrix Adaptation Evolution Strategy was used to develop a novel optimization technique for designing the tuning parameters for CNN and it is found to be very effective.

There are various techniques in the literature focused on optimizing the tuning parameters of the CNN. For example, a Grid Search method was proposed in [14] to optimize the CNN parameters. A Random Search based optimization of hyper-parameters of the CNN was proposed in [15]. There

exist optimization techniques that employ Bayesian optimization strategy for designing CNN parameters such as those proposed in [16], [17], [18]. A Differential Evolution based optimization algorithm was proposed in [19]. A similar task was obtained using the Harmonic Search method in [20]. Reinforcement learning-based techniques were developed in [21] and [22]. Recently, in [23], the Micro canonical Optimization algorithm was employed for the automatic selection of the hyper-parameters in CNN.

In summary, all these existing works on the selection of hyper-parameters in CNN are concerned with the development of any automated algorithm that can provide optimum or near optimum values of these hyper-parameters. The primary relevant outcome of these related works is that they always link the accuracy and performance of the neural network to the hyper-parameters optimization of the micro and macro-structural levels of the model. However, we claim that in addition to the structural levels of the CNN model another factor affects the accuracy and performance of the model. The factor is the way of implementation of certain inter-layer operations in CNN. Our preliminary results on a single dataset with lesser details were published in [24]. In this work, we provide a more detailed analysis on various datasets in this context.

B. Main Contributions

The main contributions of our work can be summarized as follows:

1) In this work, we provide a framework to empirically investigate the effect of the way of implementation of certain inter-layer operations in CNN.

2) More specifically, we compare two different inter-layer operations namely cropping and padding the input volume to make it fit for the next layer operation.

3) Our investigation is based on analyzing the CNN performance in terms of classification accuracy, processing time, and generalization by implementing two models: one using crop and the second employing padding of the input volume on Digits, MNIST, Merch, Flowers, and CIFAR-10 datasets.

4) Our work provides a foundation for future investigation of the effects of other inter-layer operations in a CNN.

C. Problem Definition

A common practice in the research of deep learning is to use built-in CNN libraries, such as Tensorflow, Caffe, and Keras. However, users do not have control over the low-level implementation of different blocks of CNN which are implemented differently in different libraries. For example, in order to deal with input volumes when the stride hyper-parameter value results in a non-integer pre-calculated output, there are two standard approaches: first is cropping the volume and second is padding the volume. To understand these approaches, consider an example where an input volume of $90 \times 70 \times 100$ which corresponds to 70 channels with mini-batch size of 100. The pooling layer with a window size of 4×4 and a stride of 4 is used with no padding. The pre-calculated dimension for the output of both convolution and pooling is given as follows (see [25] for the formula used):

$$Feature\ Map\ Size = \frac{I-F+2P}{S}\ \frac{90-4+(2\times 0)}{4} + 1 = 22.5 \qquad (1)$$

where:

I: Row or column size of the Input volume

F: Row or column size of the filter

P: Zero-padding

S: Stride

It can be observed in the calculation provided in Eq. (1) that both the axes of the feature map gave the same measure (22.5). Otherwise, Eq. (1) can be used to calculate the dimension size separately for each case. It is to be noted that the calculation of Eq. (1) results in a float number (22.5) for this example. Thus, the pooling size used was not appropriate to fit the input volume. This problem can come across in any layer type and any layer number. Now, it is important to know how this issue is handled in different libraries. There are three possible solutions to this issue; the first is to go back to choose a different value of the hyper-parameter and check again its feasibility; the second is to crop the input volume in order to fit the volume for the next layer; and the third is to pad (usually zero-padding is employed) the volume to fit the volume. The question is: Are these approaches going to perform the same? Hence, there are Constraints on choice of hype-parameters in the design of CNNs.

D. Objectives

The issue to investigate is how both approaches will affect the network's performance. This will tell us whether non-optimal performance is purely due to the wrong choice of hyper-parameters or is it also partially due to how the library is handling that type of hyper-parameters (that gives a non-integer value for the pre-calculated output). The outcome of this work will be a reference for both CNN library users and those designing CNNs and other deep learning paradigms from scratch.

Two models are going to be implemented from scratch, the first model implementation will involve the first approach that crops the input volume, and the second model will present the implementation of the second approach that involves zero-padding the input.

III. Datasets

The used datasets in this work are Digits, MNIST, Merch, Flowers, and CIFAR-10. The MNIST dataset contains 70,000 images of handwritten digits having a size of 28×28. For machine learning applications, this dataset is split into 60%, 25%, and 15% for training, validation, and testing, respectively. The dataset is almost balanced among different classes, so the performance measure will not be misleading, at least from this side. Also, different image size datasets were chosen with a small to a reasonably large number of images. The number of classes ranges from 5 to 10. Table I. shows some of the characteristics of the datasets used. The experiments for the rest of the datasets were performed using

the original images and no transformation such as conversion to grayscale, rotation, etc. has been performed.

TABLE I. CHARACTERISTICS OF DATASETS USED

Dataset	Image dimension	Classes	Total images
Merch	227 x 227	5	75
Flowers	224 x 224	5	3670
CIFAR-10	32 x 32	10	60,000
Digits	28 x 28	10	10,000
MNIST	28 x 28	10	70,000

IV. TERMINOLOGIES

There are terms used in the paper and what they entail for each one in the experiments. The terms are Channels, Filter, Stride, Padding, Dilation Factor, Output Size, Number of Neurons, Network Architecture, and Layer Size Calculation. Which are used from the MATLAB software [26].

We start by channels; if the number of channels is one, then the images are grayscale whereas it is three for colored images referring to the RGB spectrum. The filter convolves the input where a set of weights are applied to a specific image area. It can move either in horizontal or vertical directions or both at the same time. Stride determines the movement of the filter. The step size with which the filter moves determines the size of the output. Usually, the size of output reduces with increasing strides. The stride value is usually specified as an integer rather than a decimal. Padding refers to adding values on the border of images horizontally and/or vertically. It is used to control the output size of the layers. The dilation factor is the number of spaces used inside the filter. It expands the filter by inserting zeros between the filter elements. The adequate size of the filter can be computed as $(Filter\ Size - 1) * Dilation\ Factor + 1$. A 3×3 filter with a dilation factor of 2 in horizontal and vertical directions is equivalent to a 5×5 filter having 0s between the elements. The output height and width of a convolutional layer can be computed using the following relation:

$$Output\ Size = \frac{I-(F-1)*DF+1+2P}{S} + 1 \quad (2)$$

where DF represents the Dilation Factor. This calculation needs to be done separately for the x- and y-direction if they are not the same. The output size needs to be an integer value and MATLAB discards the remaining part if it is not an integer.

The total number of neurons in a convolutional layer is $(h * w * c + 1) * Number\ of\ Filters$, where h and w are the height and width of the filter, c is the number of channels and 1 is the bias. A convolutional layer with five filters and a filter size of 3×3 for colored images has a total number of neurons equal to $(3 \times 3 \times 3 + 1) \times 5 = 140$. A Convolutional Neural Network (CNN) can have one or more layers depending on the size and complexity of the data. A small network with one or two layers is reasonable for small grayscale datasets whereas a complex architecture of layers is required for a dataset having millions of images. Considering the datasets used, we choose a series of 15-layer Convolutional Neural Network (CNN) architecture. It is an architecture of a deep network with all layers connected sequentially. Since we have categorical responses, softmax and classification layers are used at the end. A snapshot of this architecture when run on the Digits dataset in MATLAB is shown in Fig. 1. Also, the parameters used for such architecture are shown in Table II.

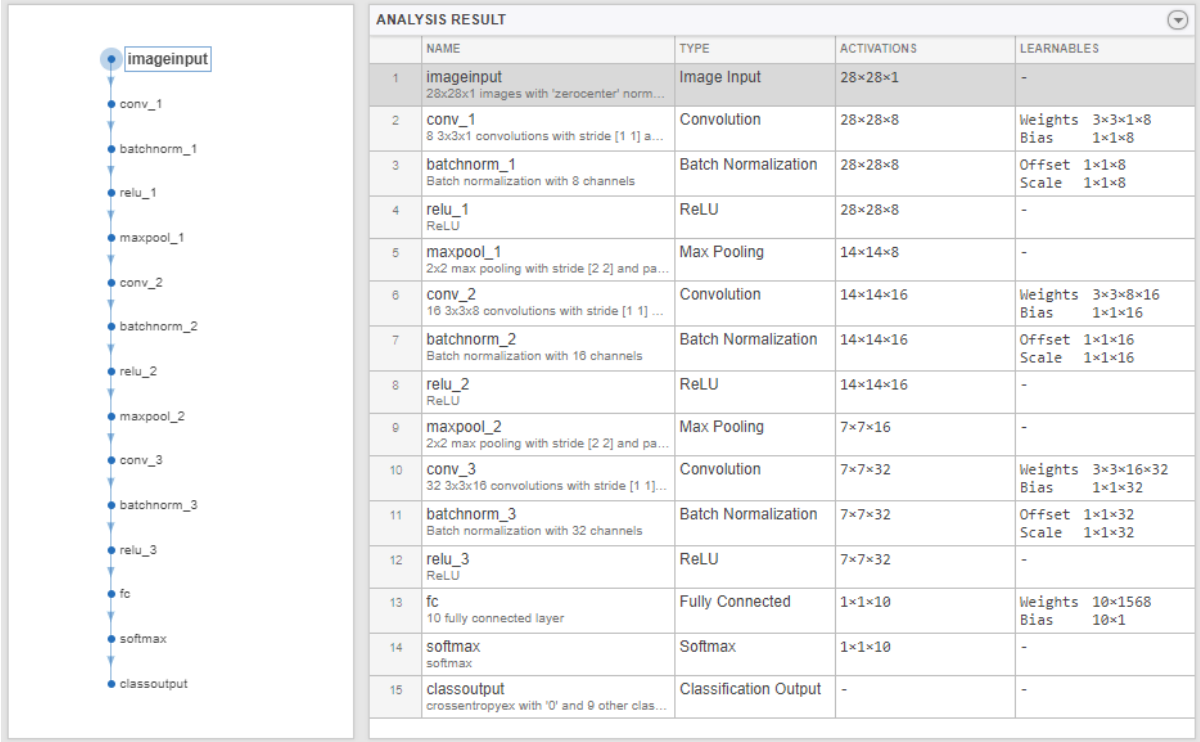

Fig. 1. 15-Layer CNN Architecture on Digits Dataset.

TABLE II. PARAMETERS OF THE CNN USED

Parameter	Value
InitialLearnRate	0.01
MaxEpochs	6
MiniBatchSize	100

The padding and stride are considered the same in both directions in all experiments. The output size needs an integer value to cover the whole image. If it does not cover the whole image, then the MATLAB software ignores the remaining part along the right and bottom edges in the convolution. The objective of the experiments is to see the effect of this coverage on accuracies while changing the padding size and stride values. The dilation factor is set as one in all experiments. For example, suppose the image size is 32×32 and the filter of 5×5 is used in the convolution. If the padding value is two and the stride is 2 in each direction, then the output size is calculated as.

$$\big(32-\big((5-1)*1+1\big)+2*2\big)/2+1=(32-5+4)/2+1=16.5$$

In this case, the output size will become 16×16 and some of the zero paddings are discarded from the right and bottom of the image.

V. IMPLEMENTATION OF MODEL 1

The initial weights are saved so that the same weights should be used for the implementation of both strategies. Also, this is to ensure that the comparison should be fair and should be affected by only cropping or padding and not by any other factors. Moreover, neither regularization nor momentum of any form was used for the same reason of fair comparison. The details of the CNN architecture employed for Model 1 in our implementation are provided in Table III.

TABLE III. MODEL 1 ARCHITECTURE

Layer Type	Conv1	Pool1	Conv2	Pool2	FLC
Size of Filter	9×9	2×2	3×3	3×3	80×10
Depth of Filter	1	10	10	20	N/A
Number of Filters	10	N/A	20	N/A	N/A
Stride	1	2	1	3	N/A
Zero Padding	0	0	0	0	N/A
Input volume size: $28 \times 28 \times 1 \times 50$ Pooling type: mean					

To simplify the study, hyper-parameters of the last pooling layer are selected to investigate the impact of padding and cropping (this analysis can be applied to any other layer too). By using Eq. (1), the pre-calculated output sizes of every layer are integer values except for pool2 which is a float number 2.66. To see more details, let us run the CNN and see what the output size in the forward and backward pass will be and this is summarized in Table IV.

TABLE IV. OUTPUT AND GRADIENT SIZES FOR MODEL 1

Forward Pass					
Output size	$20 \times 20 \times 10 \times 50$	$10 \times 10 \times 10 \times 50$	$8 \times 8 \times 20 \times 50$	$2 \times 2 \times 20 \times 50$	10×50
Layer x	**conv1**	**pool1**	**conv2**	**pool2**	**FLC**
$\frac{\partial L}{\partial x}$	$20 \times 20 \times 10 \times 50$	$10 \times 10 \times 10 \times 50$	$8 \times 8 \times 20 \times 50$	$2 \times 2 \times 20 \times 50$	10×50
Backward Pass					

The results of Table IV show the size of outputs at different layers and it can be noted that all sizes are tensors of order four except the first layer. The term $\frac{\partial L}{\partial x}$ evaluates the partial derivative of the loss function with respect to the output x which gives the expression in terms of the local error (δ_x) in that neuron (note that x here is a feature map of neurons). The local error of every neuron is multiplied by its input in order to backpropagate the error influence. This can be summarized as:

$$\frac{\partial L}{\partial W} = \delta_x * In \quad (1)$$

For output layer $\delta_x = \delta_x \times f'(x)$ (2)

For any hidden layer $\delta_x = \delta_{x+1} * W_{x+1} \times f'(x)$ (3)

where:

$f'(.)$: the derivative of the activation function.

W_{x+1}: the weight matrix of the next layer.

δ_x: local error tensor of the current layer.

δ_{x+1}: local error tensor of next layer.

ln: is the input tensor to layer x.

$\frac{\partial L}{\partial W}$: the partial derivative of the loss with respect to the weights

Eventually, the size of local errors at layer x will be equal to the size of its output while the error is backpropagated through the layers. At this stage, we analyze what happens in the forward and backward pass only at the critical pool2 layer, as illustrated in Fig. 2.

The original sizes of tensors utilized is of order four. However, in Fig. 2, it can be noted that only the first and second dimensions of those tensors are demonstrated as these are the ones that are affected by different implementations.

1) Forward pass: During the forward pass, it can be noticed that the sliding 3×3 window with stride 3 on an input of 8×8 (which is the output size of the previous conv layer) can accommodate only the first six rows and columns. Thus, we need to truncate or crop the last two rows and columns to obtain a new input 6×6 for the pool2 layer and this matches the results of pool2 in Table IV.

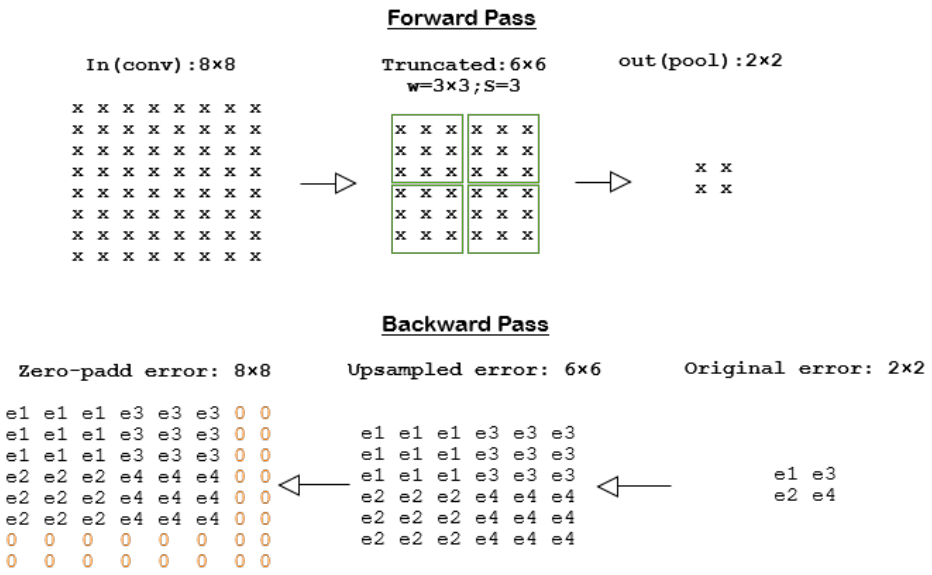

Fig. 2. Backward and Forward Pass at Layer 4 for Model 1.

2) Backward pass: For the backward pass, using Eq.(5), it is found that the local error or the local gradient of pool2 layer has a size 2×2 as shown in Table IV. At the same layer, the local error δ_x of conv2 layer (previous layer) will be computed also using Eq. (5). The reason of doing this calculation at pool2 and not at the conv2 layer is the fact that in the forward pass from conv2 to pool2 the only process involves was down-sampling through pooling. Thus, there will be no change in weights or bias. Hence, it is easy to see that the local errors of conv2 can be obtained by up-sampling the local errors of pool2. Using Eq. (5), the local errors of conv2 is achieved by employing up-sampling on local errors of the pool2 layer which is shown in Fig. 2. Now, the *Conv2* local errors are of size 6×6. However, it must be the same size as the size of its output, i.e., 8×8. By employing the padding with zeros it is possible to accomplish the size requirement and this was done via a zero-padded error 8×8 in Fig. 2. In our case, all the weights connected to the right two columns and bottom two rows of conv2 output are not going to be updated because their local errors $\delta_x = 0$ and from Eq. (3) $\frac{\partial L}{\partial W} = 0$. This is due to the fact that these links did not contribute to the final loss as they were cropped in the forward pass. This is a major side effect of cropping inputs in the scenarios when our choice of hyper-parameter stride does not result in an appropriate output size. The next question to be investigated is how this will affect the predictive performance. The answer of this will be found in the ensuing section.

VI. Implementation of Model 2

The architecture of this model will stay the same as the previous one with one change in the FLC weights size which will become 320×10. Table V contains the output sizes in the forward and backward pass (as before) after running CNN model 2. It can be noticed that the vectorized output of pool2 has the dimension $4 \times 4 \times 20 = 320$, which explains why the FLC layer weights were changed in this architecture. Fig. 3 illustrates how the forward and backward passes are implemented.

1) Forward pass: In this second strategy of hyper-parameters implementation, the input padding with four zeros is employed in contrast to the cropping process used in model 1 which results in an input of size 12×12. Hence, this makes the size compatible with the next layer. Eventually, pooling resulted in the output size of 4×4 as shown in Table V.

2) Backward pass: Again as was done in Model 1 implementation, the pool2 local errors are calculated using Eq. (5) which gave a size of 4×4 as shown in Table V. Next, up-sampling was employed to obtain a zero-padded error 12×12 as shown in Fig. 3. One important fact to be noted is that unlike the Model 1 implementation, in this case all weights of the neurons in conv2 are updated using Eq. (3). Hence, this can drastically impact on the predictive performance of model 2.

TABLE V. Output and Gradient Sizes for Model 2

Forward Pass					
Output size	$20 \times 20 \times 10 \times 50$	$10 \times 10 \times 10 \times 50$	$8 \times 8 \times 20 \times 50$	$4 \times 4 \times 20 \times 50$	10×50
Layer x	**conv1**	**pool1**	**conv2**	**pool2**	**FLC**
$\frac{\partial L}{\partial x}$	$20 \times 20 \times 10 \times 50$	$10 \times 10 \times 10 \times 50$	$8 \times 8 \times 20 \times 50$	$4 \times 4 \times 20 \times 50$	10×50
Backward Pass					

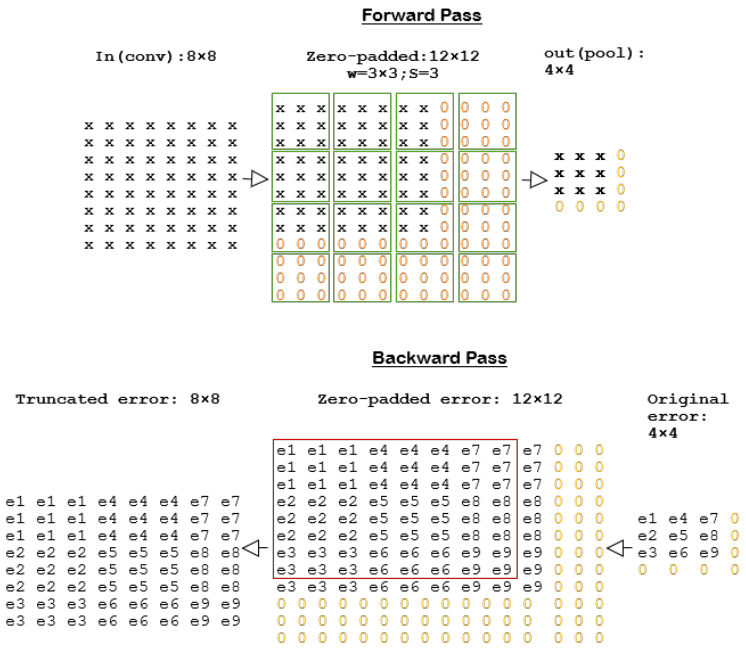

Fig. 3. Backward and Forward Pass at Layer 4 for Model 2.

VII. Experiments and Results

In this section, the performance of the implemented Models 1 and 2 is compared in terms of loss, accuracy, and processing time. The objective of the experiments is to see the effect of padding size and stride at layer 10 which is a convolutional layer with the name 'conv3'. Filter size of three is used at the convolution layer. Table VI shows the size of the three layers used for Digits dataset. Here, the size of the convolution layer is calculated using the formula given in Eq. (2).

TABLE VI. Size of Layers in Digits Dataset

Image Dimension	Padding dimension	Stride Size	Conv. Layer Filter Size	Actual Output Size
28 x 28	0	1	5 x 5	5
		2	3 x 3	3
		3	2 x 2	2.33
		4	2 x 2	2
	1	1	7 x 7	7
		2	4 x 4	4
		3	3 x 3	3
		4	2 x 2	2.5
	2	1	9 x 9	9
		2	5 x 5	5
		3	3 x 3	3.66
		4	3 x 3	3

The experiments were performed on several datasets to observe the effect of changing padding and stride on all three types/sets of accuracies (training, validation, testing). The padding values from zero to four and stride values from one to four are used for all experiments. It is to be noted that the stride value of one is a trivial case and does not cause a problem but we keep it for comparison purposes. To achieve unbiased results, we first train the architecture with specific parameters such as the same padding size in all layers. After that, we get the weights of all layers out of which we freeze the first nine layers' weights and then see the effects of padding and stride starting from layer 10.

Results of the first experiment reported in Fig. 4 and Fig. 5 for Model 1 and Model 2, respectively, show that the algorithm is approximately converging in 20 epochs for both models. Next, Fig. 6 shows results at the final epoch for the two models. Here, it can be observed that model 1 has achieved a testing accuracy of 82.62% in contrast to 85.580% achieved by the Model 2. The same behavior can also be observed while comparing the training and validation accuracies of the two models. In Fig. 7 it is found that model 2 took almost 2 hours 30 minutes to train in contrast to the 2 hours 38 minutes of training by model 1. Thus, it is concluded that model 2 has a faster speed of convergence than model 1.

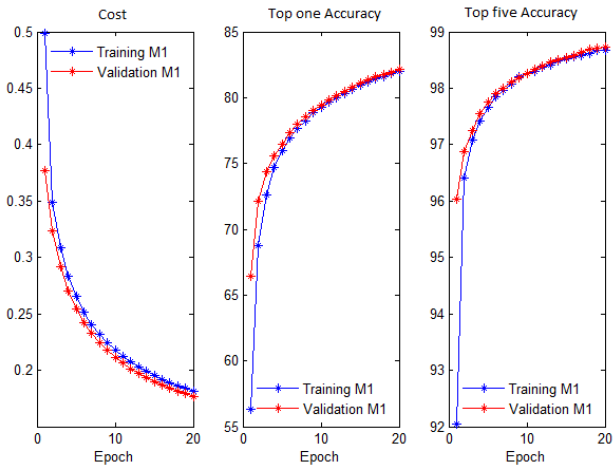

Fig. 4. Implementation of Model 1 Performance Measures.

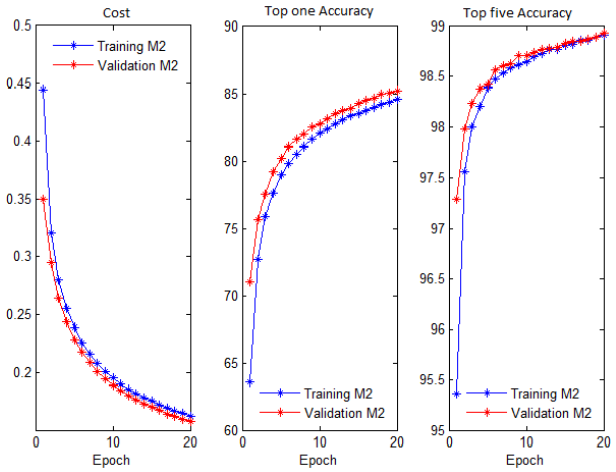

Fig. 5. Implementation of Model 2 Performance Measures.

Another important observation from the results in Fig. 4 and Fig. 5 is that validation accuracy is a bit higher than training accuracy, especially in Fig. 5 showing that the trained model are more generalized in training.

In summary, it is observed that model 2 implementation has better classification accuracy, processing time, and generalization than the ones achieved by model 1. This is mainly due to the removal of the 25% conv2 neurons in the backward pass as cropping was employed. In contrast, padding with zeros used in model 2 keeps all the conv2 neurons and hence all conv2 parameters were utilized. Hence, contributions from all the neurons are included which enhances the predictive accuracy of the model.

Next, bar graphs are presented to see the effects of padding and strides in mainly the training and testing accuracies for the datasets used. The training accuracy is higher when the stride value is equal to 1, as shown in Fig. 8. In general, for all stride values, the training accuracy becomes better as the padding increases. The same is observed for testing accuracy in Fig. 9, and although not shown here, the validation accuracy also follows a similar pattern.

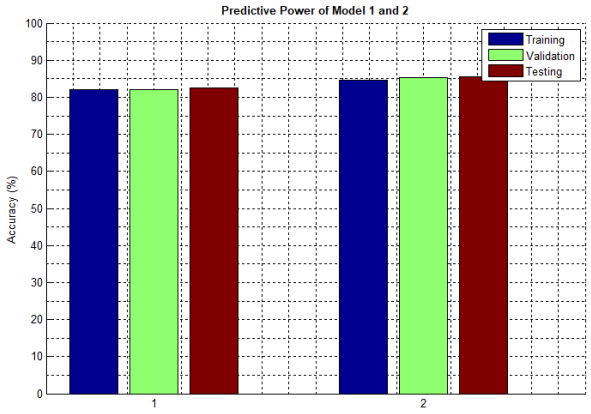

Fig. 6. Classification Accuracy of Models 1 and 2.

By observing Fig. 10, the difference between the training and testing accuracy is more when the stride value is four for all padding values. The difference reduces when the stride value is three, and it is minimum when the stride is 1. Hence, we can say that the minimum the stride, the lesser the difference between training and testing accuracy. This result is intuitive as well because more strides mean more skipping of the bits/values and less learning from the training set, which in turn reduces the testing accuracy. Similar results were observed for other datasets of Merch and CIFAR10 as can be seen in Fig. 11 to 13.

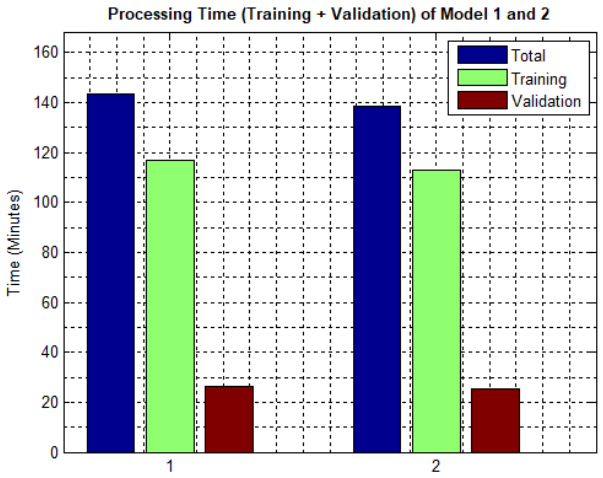

Fig. 7. Processing Time of Models 1 and 2.

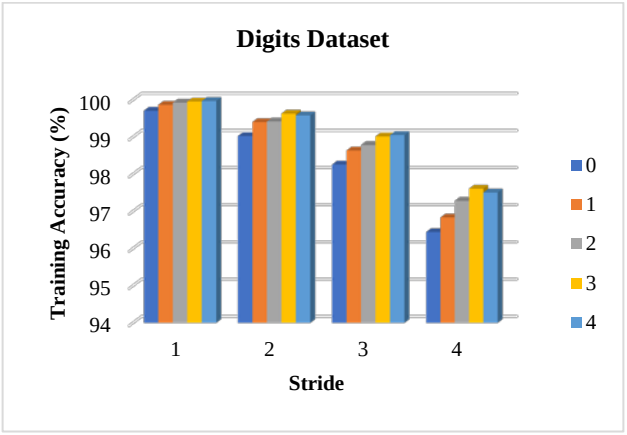

Fig. 8. Training Accuracy Digits Dataset.

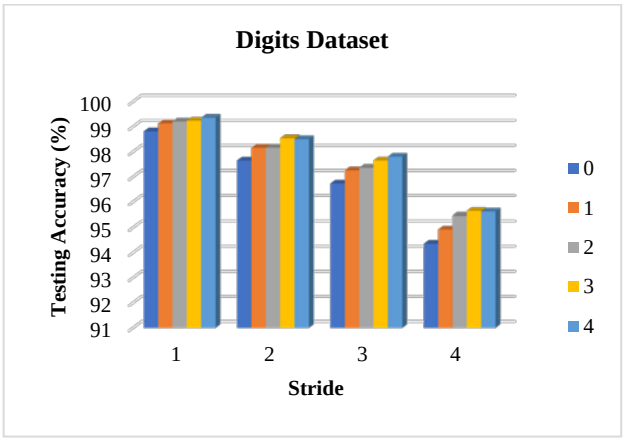

Fig. 9. Testing Accuracy Digits Dataset.

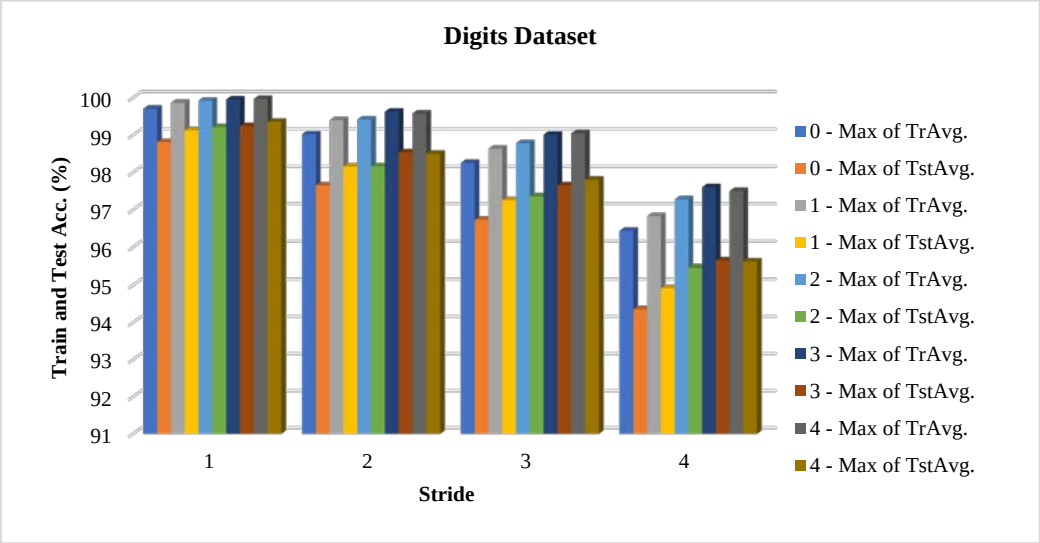

Fig. 10. Training and Testing Accuracy.

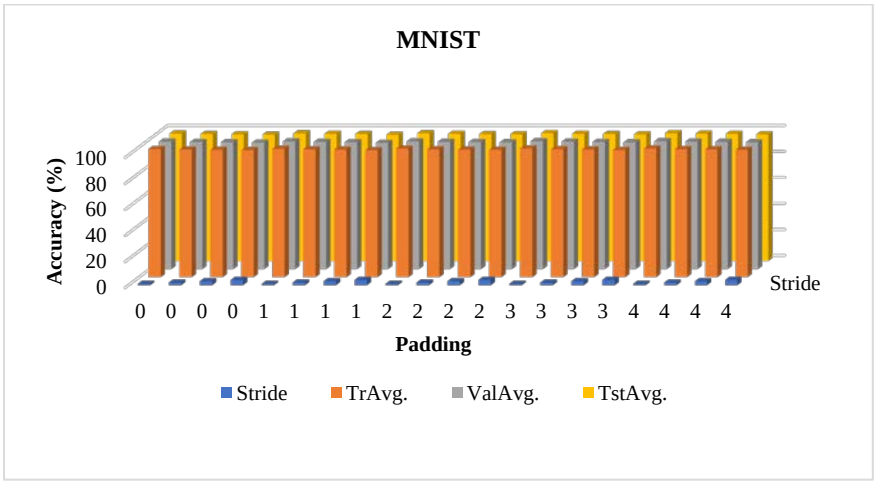

Fig. 11. FMNIST Accuracies.

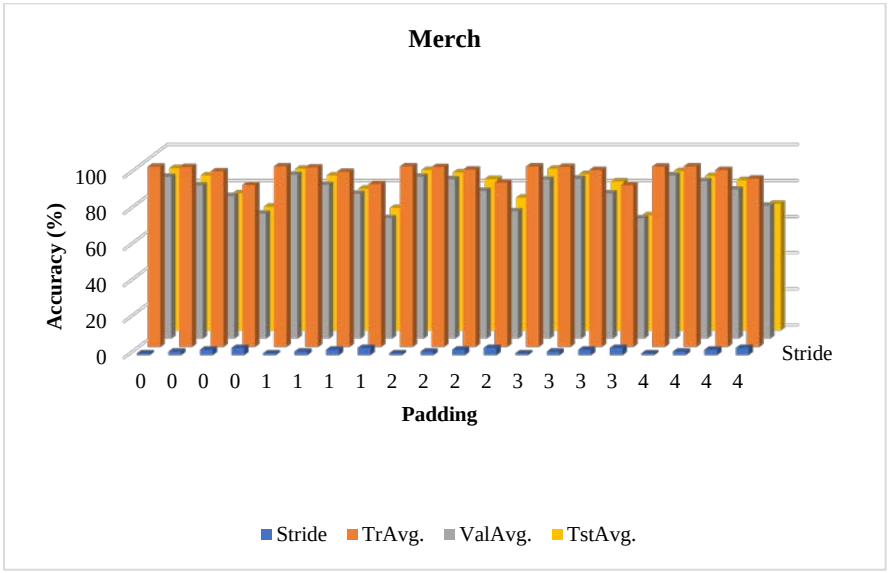

Fig. 12. Merch Accuracies.

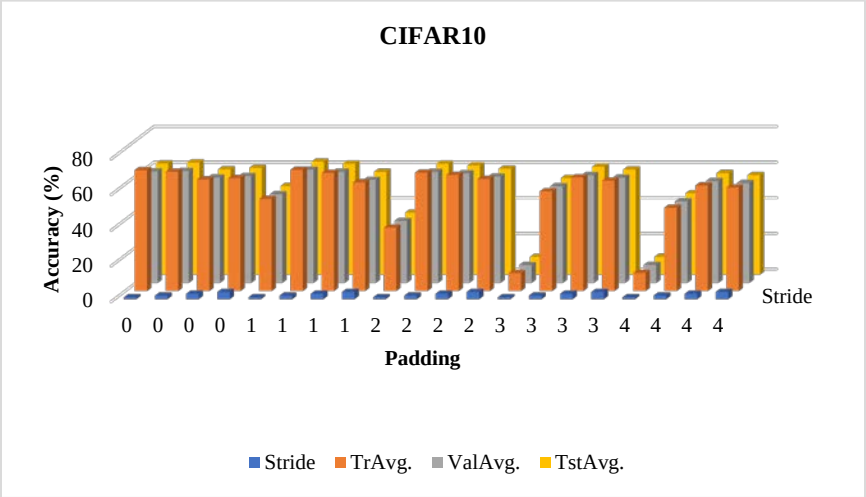

Fig. 13. CIFAR10 Accuracies.

VIII. Conclusion

In this work, we investigated an important issue as to how the performance of a CNN is affected by choice of hyper-parameters in the case when this choice does not fit the input volumes. It was investigated experimentally how the internal implementation of hyper-parameters particularly cropping and padding the input volumes are going to affect the performance measures of CNN. For this purpose, Digits, MNIST, Merch, Flowers, and CIFAR-10 datasets were analyzed and the two models were compared in terms of classification accuracy, processing time, and generalization. It was proved via various experiments that the model employing padding of input volume has higher accuracy with lesser training time in contrast to the model using cropping of input volume. Thus, it is concluded that the fair comparison of the performance for various CNN methods will be obtained when the hyper-parameters are set fairly.

Acknowledgment

The authors extend their appreciation to the Deputyship for Research and Innovation, Ministry of Education in Saudi Arabia, for funding this research work through the project number (IFPRC-118-135-2020) and King Abdulaziz University, DSR, Jeddah, Saudi Arabia.

Funding Statement

This research work is funded by Institutional Fund Projects by the Ministry of Education, Saudi Arabia, under grant no. (IFPRC-118-135-2020).

Conflict of Interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

References

[1] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," Journal of Machine Learning Research, vol. 13, no. Feb, pp. 281-305,2012.

[2] G. I. Diaz, A. Fokoue-Nkoutche, G. Nannicini, and H. Samulowitz, "An effective algorithm for hyperparameter optimization of neural networks," IBM Journal of Research and Development, vol. 61, no. 4, pp. 9-1, 2017.

[3] J. F. Khaw, B. Lim, and L. E. Lim, "Optimal design of neural networks using the taguchi method," Neurocomputing, vol. 7, no. 3, pp. 225-245,1995.

[4] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in Advances in neural information processing systems, pp. 2951-2959, 2012.

[5] T. Domhan, J. T. Springenberg, and F. Hutter, "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves.," in IJCAI, vol. 15, pp. 3460-8, 2015.

[6] E. Bochinski, T. Senst, and T. Sikora, "Hyper-parameter optimization for convolutional neural network committees based on evolutionary algorithms," in Image Processing (ICIP), 2017 IEEE International Conference on, pp. 3924-3928, IEEE, 2017.

[7] Y. Ozaki, M. Yano, and M. Onishi, "Effective hyperparameter optimization using nelder-mead method in deep learning," IPSJ Transactions on Computer Vision and Applications, vol. 9, no. 1, p. 20, 2017.

[8] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," arXiv preprint arXiv:1603.06560, 2016.

[9] E. Hazan, A. Klivans, and Y. Yuan, "Hyperparameter optimization: A spectral approach," arXiv preprint arXiv:1706.00764, 2017.

[10] A. F. Cardona-Escobar, A. F. Giraldo-Forero, A. E. Castro-Ospina, and J. A. Jaramillo-Garzon, "Effcient hyperparameter optimization in convolutional neural networks by learning curves prediction" in Iberoamerican Congress on Pattern Recognition, pp. 143-151, Springer, 2017.

[11] A. Klein, S. Falkner, S. Bartels, P. Hennig, F. Hutter, et al., "Fast bayesian hyperparameter optimization on large datasets," Electronic Journal of Statistics, vol. 11, no. 2, pp. 4945-4968, 2017.

[12] S. Albelwi and A. Mahmood, "Automated optimal architecture of deep convolutional neural networks for image recognition," in Machine Learning and Applications (ICMLA), 2016 15th IEEE International Conference on, pp. 53-60, IEEE, 2016.

[13] I. Loshchilov and F. Hutter, "Cma-es for hyperparameter optimization of deep neural networks," arXiv preprint arXiv:1604.07269, 2016.

[14] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio,``An empirical evaluation of deep architectures on problems with many factors of variation," in Proc. 24th Int. Conf. Mach. Learn. (ICML), 2007, pp. 473_480.

[15] J. Bergstra and Y. Bengio, ``Random search for hyper-parameter optimization," J. Mach. Learn. Res., vol. 13, pp. 281_305, Feb. 2012.

[16] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, ``Algorithms for hyperparameter optimization," in Proc. Adv. Neural Inf. Process. Syst., 2011, pp. 2546_2554.

[17] F. Hutter, H. H. Hoos, and K. Leyton-Brown, ``Sequential model-based optimization for general algorithm con_guration," in Proc. Int. Conf. Learn. Intell. Optim. Berlin, Germany: Springer, 2011, pp. 507-523.

[18] M. W. Hoffman and B. Shahriari, ``Modular mechanisms for Bayesian optimization," in Proc. NIPS Workshop Bayesian Optim., 2014, pp. 1-5.

[19] B. Wang, Y. Sun, B. Xue, and M. Zhang, ``A hybrid differential evolution approach to designing deep convolutional neural networks for image classification," in Proc. Australas. Joint Conf. Artif. Intell. Cham, Switzerland: Springer, 2018, pp. 237-250.

[20] W.-Y. Lee, S.-M. Park, and K.-B. Sim, ``Optimal hyperparameter tuning of convolutional neural networks based on the parameter-setting-free harmony search algorithm," Optik, vol. 172, pp. 359_367, Nov. 2018.

[21] B. Baker, O. Gupta, N. Naik, and R. Raskar, ``Designing neural network architectures using reinforcement learning," 2016, arXiv:1611.02167. [Online]. Available: http://arxiv.org/abs/1611.02167.

[22] P. Neary, ``Automatic hyperparameter tuning in deep convolutional neural networks using asynchronous reinforcement learning," in Proc. IEEE Int. Conf. Cognit. Comput. (ICCC), Jul. 2018, pp. 73-77.

[23] A. Gülcü and Z. KUş, "Hyper-Parameter Selection in Convolutional Neural Networks Using Microcanonical Optimization Algorithm," in IEEE Access, vol. 8, pp. 52528-52540, 2020, doi: 10.1109/ACCESS.2020.2981141.

[24] U. M. Al-Saggaf, A. Botalb, M. Moinuddin, S. A. Alfakeh, S. S. A. Ali and T. T. Boon, "Either crop or pad the input volume: What is beneficial for Convolutional Neural Network?," 2020 8th International Conference on Intelligent and Advanced Systems (ICIAS), 2021, pp. 1-6, doi: 10.1109/ICIAS49414.2021.9642661.

[25] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," arXiv preprint arXiv:1603.07285, 2016.

[26] “Specify Layers of Convolutional Neural Network,” Specify Layers of Convolutional Neural Network - MATLAB & Simulink, 2022. [Online]. Available: https://www.mathworks.com/help/deeplearning/ug/layers-of-a-convolutional-neural-network.html.