mathematics

MDPI

Article

Evaluation of Deformable Convolution: An Investigation in Image and Video Classification

Andrea Burgos Madrigal, Victor Romero Bautista, Raquel Díaz Hernández and Leopoldo Altamirano Robles *

Instituto Nacional de Astrofísica, Óptica y Electrónica, Puebla 72840, Mexico; burgosmad@inaoep.mx (A.B.M.); victor.romero@inaoep.mx (V.R.B.); raqueld@inaoep.mx (R.D.H.)
* Correspondence: robles@inaoep.mx

Abstract: Convolutional Neural Networks (CNNs) present drawbacks for modeling geometric transformations, caused by the convolution operation's locality. Deformable convolution (DCON) is a mechanism that solves these drawbacks and improves the robustness. In this study, we clarify the optimal way to replace the standard convolution with its deformable counterpart in a CNN model. To this end, we conducted several experiments using DCONs applied in the layers that conform a small four-layer CNN model and on the four-layers of several ResNets with depths 18, 34, 50, and 101. The models were tested in binary balanced classes with 2D and 3D data. If DCON is used on the first layers of the proposal of model, the computational resources will tend to increase and produce bigger misclassification than the standard CNN. However, if the DCON is used at the end layers, the quantity of Flops will decrease, and the classification accuracy will improve by up to 20% about the base model. Moreover, it gains robustness because it can adapt to the object of interest. Also, the best kernel size of the DCON is three. With these results, we propose a guideline and contribute to understanding the impact of DCON on the robustness of CNNs.

Keywords: computer vision; image/video analysis; deformable neural networks

MSC: 68T05

Citation: Burgos Madrigal, A.; Romero Bautista, V.; Díaz Hernández, R.; Altamirano Robles, L. Evaluation of Deformable Convolution: An Investigation in Image and Video Classification. *Mathematics* **2024**, *12*, 2448. https://doi.org/10.3390/math12162448

Academic Editors: Gustavo Arroyo-Figueroa, Juan Humberto Sossa-Azuela and Osslan Osiris Vergara Villegas

Received: 29 May 2024
Revised: 25 July 2024
Accepted: 2 August 2024
Published: 7 August 2024

1. Introduction

Convolutional Neural Networks (CNNs) have dominated the field of computer vision, achieving superior results to traditional machine learning methods [1]. Their success is due to their capacity to capture spatial features and patterns in images using a layered hierarchical architecture, where mainly convolution and pooling operations are performed [2].

The core component of CNNs is the convolution operation, which extracts features and patterns from the input image. Each convolutional layer is composed of multiple kernels [3], which, during the training phase, the weights of which are learned, turning them into feature extractors, identifying specific patterns, edges, and textures present in the input data [4].

Convolutional layers use spatial filtering to learn meaningful features for high-level recognition. This is accomplished by sliding a kernel (matrix of weights) over the input to produce feature maps. The filter is limited to the sliding defined by the stride and the kernel size. Therefore, there is no flexibility during the learning process. As a consequence, the receptive field of the activation units remains the same in each convolutional layer.

Some authors propose overcoming geometric variations by focusing on the receptive field. One solution that emerged is to expand the receptive field; Ding et al. [5] used kernels of size 31 × 31 for downstream tasks taking advantage of re-parameterization by constructing a small kernel (e.g., 3 × 3 or 5 × 5) parallel to it, Liu et al. [6] applied large kernels inspired by sparsity. Augmented kernels were also investigated in 3DCNNs by Chen

et al. [7] with high performance in 3D semantic segmentation and object detection. However, incrementing the kernel size increases the computational cost (due to the quadratic computational complexity). Dilated convolutions merged, and are effective for expanding receptive fields without increasing computational cost. To cover a larger area of the input, the filter is "dilated" by inserting gaps between the filter values. Dilated convolutions are useful where a larger context is needed to classify or if the network needs to learn patterns with longer time dependencies. However, the dilated technique is imposed quite straight by its regular grid, and it appears to be only a rapid solution if you are interested in the context per se. Khalfaoui et al. [8] investigated a dilated convolution without fixed positions, but learnable via backpropagation. This approach is similar to deformable convolutions, but without the dependence between the offsets and the input feature map; in other words, the DCLs method is input-independent. Then, as kernels are commonly defined as squared filters, related works explored changing the kernel's shape; Sun et al. [9] analyzed the effects of hexagonal shapes of convolutional kernels on feature representations learned in CNNs. It was observed that hexagonal kernels tend to be less concentrated on local regions and rather distributed across a variety of sub-regions, compared to CNNs with standard square kernels improving the robustness of the base-line models to occlusion for classification of partially occluded images. He et al. [10] proposed circular convolutional kernels based on the biological visual system having approximately concentric receptive fields. The circular kernel gained robustness to rotated or sheared images. Gao et al. [11] proposed a deformable kernel to adapt to effective receptive field deformation of objects, experimentally proving deformable cores and deformable volumes.

A common technique to increase the capacity to model geometric transformations is to use data augmentation. However, this operation requires more training time, and if transformations are not appropriately modeled, the risk of overfitting increases.

Dai et al. [12] proposed deformable convolution (DCON) which replaces the standard convolution of a CNN layer with deformable convolution, increasing the receptive field to capture relevant features (usually called deformable convolutional networks, or DCNNs). Zhu et al. [13], extended deformable convolutions with DCNv2, introducing additional attention coefficients at the sampling points of deformable convolution kernels. The additional coefficients extends the scope of deformable modeling by introducing a modulation or adjustment mechanism. Wang et al. [14] proposed DCNv3, whose sampling offsets are flexible in long and short ranges and it separates the original weights into depth- wise and point-wise components. Xiong et al. [15] replaces DCNv3 with DCNv4 by removing softmax normalization in spatial aggregation and contributing significantly to faster converge and optimizing memory access. DCON is a promising approach to give a CNN the capacity to model transformations, such as scale, increasing its robustness. However, the optimal way to use DCN on low- and medium-scale models and datasets for image and action classification tasks in video is not clear.

In the state of the art, some works have been proposed that give clues on how to adequately incorporate DCON. Dai et al. [12] evaluated the effect of DCON using ResNet-101 for feature extraction, finding that accuracy improves with more DCONs included. Compared to their plain convNets counterparts, models with DCNs improved by 5.5 when applying three DCNs for semantic segmentation, and about 0.6 to 12.8 when using six DCNs in object detection. Based on the Dai et al. [12] study, Chen et al. [16] replaced deformable convolutions in the Res5 block of ResNet101 and ResNet50 applied on semantic segmentation. Three deformable convolutions were tested: DCNv1, DCNv2, and A-DCN, finding that using adaptive deformable convolutions could gain 1.5 improvement over the same original models with an addition of cost and complexity that could be ignored. Lai et al. [17] investigated the performance of the DCON in a C3D model in different layers. Unlike us, they found that lower layers yield better results. The 2D DCN is extended into 3D DCN investigating the deformable filters in various layers and the effect of different dilatation sizes. Surprisingly, the higher results were found with the plain C3D with 63.35, and then the better DCN was found when applied to Conv1a with 48.50 declining to a

value of 4.15% when applied to Conv2a and 1.26 in later layers. The result is different from that already reported in segmentation and object recognition. These studies have reported that replacing the standard convolution layer with a deformable one improves the performance of well-known Convolutional Neural Networks for feature extraction, such as the ResNet-50 and ResNet-101 models. However, they do not go further into the study of other architectures of the ResNet model, such as ResNet-18 and ResNet-34. Different from that research, in this work, we go further into the DCON evaluation for feature extraction, extending it to different architectures of the ResNet model, starting from ResNet-18 up to ResNet-101.

In this paper, we give a guideline for DCNNs by investigating the behavior of CNN models when DCON is introduced. For this purpose, several experiments to analyze DCNNs were conducted, different from previous works. We start our experiments from 2D models employing a small model, composed of four stages. After that, deeper models such as ResNet-18, ResNet-34, ResNet-50, and ResNet-101 are tested. To go further in our analysis, we extend the experiments to 3D models using the 3D ResNet variants.

These models were evaluated in image classification tasks using varied datasets with controlled conditions. The models should carry on binary decisions (the presence or absence of glaucoma) and animal identification. Moreover, we applied space-time tests in the field of action recognition. DCONs perform better when placed in layers containing more complex features rather than using them in the first layers. Also, it was found that kernels with a 1 × 1 size learn only local features. In the case of kernels with size 5 × 5, the computational cost increases notably without remarkable performance benefits. Then, kernel 3 × 3 was left as the best option. These results are also confirmed in [5,14].

The contribution of this study is: (1) We explore the optimal use of deformable convolution for low- and medium-scale models when the training data set is not massive by applying in all layers and a combination of them. (2) We analyzed the robustness by applying a scale deformation during testing. (3) The model analysis goes from a small model and the ResNets depths of 101, 50, 34, and 18 for better understanding. (4) We investigated the kernel size impact. (5) The study begins with 2D data and extends to 3D data with controlled and uncontrolled conditions to meet the conditions, under which it takes advantage of the deformation caused by the receptive field. The rest of this paper is organized as follows. Section 2 gives the materials and methods involved in the study. Section 3 presents the methodology. Section 4 offers experiments and results. Sections 5 and 6 present the discussion and conclusions of this work.

2. Materials and Methods

2.1. Standard Convolution (Used in the Traditional CNN)

The convolution operation is the core of CNNs. For a two-dimensional image, the convolution is calculated as:

$$y(i,j) = \sum_{k}\sum_{l} x(i+l, j+k) \cdot w(k,l), \tag{1}$$

where $y(i,j)$ is the output or the neural activation in i,j, x is the input, w is the kernel with dimensions k,l. Commonly, y is defined as feature map. Strictly, Equation (1) is a cross-correlation; however, due to its similarity to the convolution operation, in machine learning this equation is used to refer to the convolution operation [18].

The receptive field refers to the inputs that influence in $y(i,j)$. The kernel or filter w is composed of weights learned during training that are responsible for linking the receptive field units with the output [19]. The kernel size or receptive field is defined by a grid R. For example, for a kernel with size 3 × 3 defines R as follows:

$$R = (-1,-1),(-1,0),\ldots,(0,1),(1,1), \tag{2}$$

where the location of the elements that will influence the output of the convolution operation for one neuron are defined by Equation (2). Then, Equation (1) can be defined as:

$$y(p_0) = \sum_{p_n \in R} x(p_0 + p_n) \cdot w(p_n), \tag{3}$$

where p_0 enumerates the locations i, j of the output, p_n enumerates the locations in R, x is the input feature map, and $w(p_n)$ is the weight at a certain position [12]. Figure 1 shows the representation of the 2D convolution operation for an input (input feature map) using a 3 × 3 kernel, where it can be seen that the grid R of kernel w remains rigid for computing the output (output feature map).

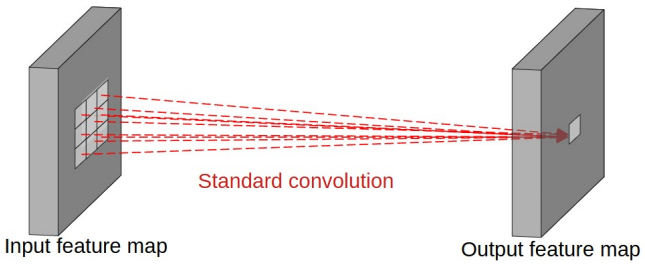


Figure 1. Standard convolution with kernel 3 × 3.

2.2. Deformable Convolutional Network (DCNN)

Proposed by Dai et al. [12] trying to diminish the difficulties of modeling transformations with standard convolutions. Then, the flexibility is acquired with a receptive field that is not limited to a rigid grid. The extension of Equation (3) will augment R with displacements defined as Δp_n. Then, deformable convolution is defined as:

$$y(p_0) = \sum_{p_n \in R} x(p_0 + p_n + \Delta p_n) \cdot w(p_n), \tag{4}$$

where Δp_n is an offset that deforms R, being $n = 1, \ldots, N; N = \mid R \mid$. Then, the receptive field is irregularly deformed by $p_n + \Delta p_n$. Figure 2 shows the deformable convolution representation, where the offsets are obtained by applying a convolutional layer on the same input feature map. The convolution kernel has the same spatial resolution as the current 3 × 3 convolution layer. The output displacement fields have the same spatial resolution as the input feature map. During training, the kernel synaptic weights are simultaneously learned to generate the output features. Also, during training are learned the offsets to deform the kernel grid and increase the receptive field.

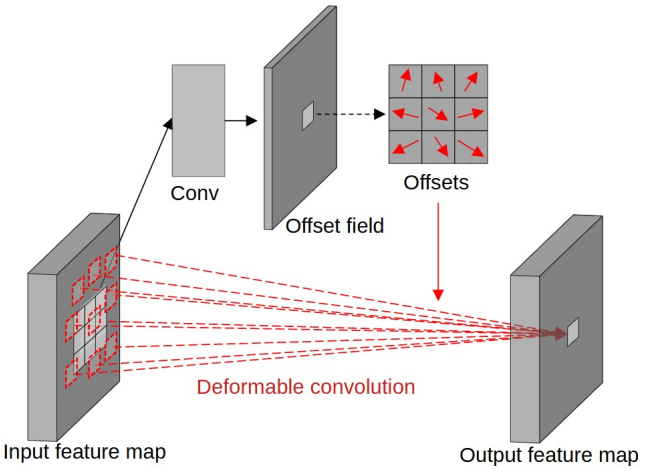


Figure 2. Deformable convolution using kernel size of 3 × 3, based on [12].

Zhu et al. in [13], found that although deformable convolution conforms more closely to the structure of objects than standard convolution, the grid shift mechanism can extend far beyond the region of interest (mainly by stacking multiple layers of deformable convolution), causing features to be influenced by irrelevant image content. Because of this drawback, they propose a modulation mechanism in the deformable convolution modules, where each sample that has undergone the learned displacement is modulated by a feature amplitude, which is also learned during the training process. The extension is called deformable convolution version 2 (DCNv2), and can be expressed as:

$$y(p_0) = \sum_{p_n \in R} \cdot x(p_0 + p_n + \Delta p_n) \cdot w(p_n) \cdot \Delta m_{p_n}, \tag{5}$$

where Δm_n is another trainable parameter between 0 and 1 for the localization p_n. Therefore, it is possible to vary the spatial distribution (offset Δp_n) and the influence of them (modulation Δm_{p_n}). Then, it is possible to stack more than one deformable convolution layer without Δp_n falling too apart from the object of interest. The ability of deformable convolution to reduce the inductive bias inherent in traditional convolution by expanding the receptive field grid, and modulating it has inspired the use of deformable CNNs in large-scale foundation models, an approach where Vision Transformer-based models [20,21] are the first choice, as they have outperformed traditional CNNs [14].

Wang et al. [14] propose a large-scale foundation model based on CNNs, where the main operator is deformable convolution. They made adjustments to DCNv2 to better adapt to the requirements of large-scale foundation models. These adjustments are focused on increasing the computational efficiency needed to build models with billions of parameters to be learned. This extension is called DCNv3, which introduces three elements: (1) Weight sharing between the convolutional neurons based on separable convolutions [22]. The weights of $w(p_n)$ are separated into depth-wise and point-wise parts, where the depth-wise part is responsible for the original location-aware modulation scalar m_k ,and the point-wise part is the shared projection weights w among sampling points. (2) Multigroup mechanism, where the spatial displacement Δp_n is divided into groups G. Each group has an individual Δp_n with modulation Δm_{p_n}. (3) Modulation normalization by using softmax instead of sigmoid function, since for training with large-scale parameters and data (typical of large-scale foundation models) the sigmoid function generates unstable gradients [14]. Thus, DCNv3 is defined as follows:

$$y(p_0) = \sum_{g \in G} \sum_{p_n \in R} x_g(p_0 + p_n + \Delta p_{gn}) \cdot w_g \cdot \Delta m_{gp_n}, \tag{6}$$

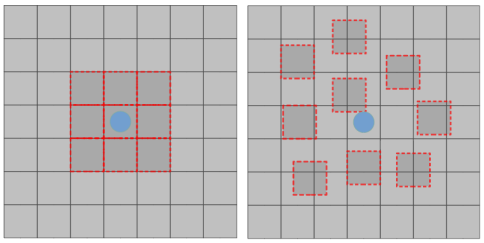
where G denotes the total of aggregation groups, m_{gp_n} indicates the modulation scalar of the sample point belonging to the group normalized by the softmax activation function, x_g it represents the entry of the fragmented feature map belonging to the group g, Δp_{gn} it is the offset corresponding to the location p_n in the group g.

Xiong et al. [15] presented DCNv4, which addresses the limitations of its predecessor DCNv3, mainly incorporating two improvements, which are: (1) removing the softmax normalization in DCNv3, since they identified that bounding the modulation scalar of the synaptic weights shift to values between 0 and 1 to add spatial features, degrades the model performance and increases the convergence time. (2) Optimizing memory access to minimize redundant operations to increase processing speed, which is mainly performed by eliminating redundant workload and memory instructions. These adjustments make DCNv4 a faster operator than its predecessor DCNv3.

DCNv3 and DCNv4 present a valuable approach for deformable convolution with promising results in large-scale models. In this study, we decided to use DCNv2 as the main operator instead of DCNv3 or DCNv4, since most of the adjustments made to DCNv2 to extend to DCNv3 and DCNv4 are for massive data and large-scale foundation models. However, our study is focused on exploring the performance of deformable convolution on

low- and medium-scale models and datasets for image and action classification in video, and DCNv2 presents the main idea of deformable convolution.

Each sampling point of the kernel learns an offset to achieve adaptation. In the process of DCON, additional convolution layers are required to extract the offset fields of the same size. Figure 3 illustrates the difference between the receptive field of standard convolution operation (Figure 3a) and the deformable convolution operation (Figure 3b), employing 3×3 kernel size. In standard convolution, the receptive field is limited to the kernel window, whereas in its deformable counterpart, the receptive field is irregular as a result of the displacement $p_n + \Delta p_n$.



(a) (b)

Figure 3. Receptive field (represented in red). (**a**) Limited in standard convolution; (**b**) Irregular and expanded by $p_k + \Delta p_k$ displacement [23].

The implementations of DCONs applied in this study are listed below. These implementations can be used to replace a standard convolution layer with the deformable counterpart:

- Tvdcn v0.5.0 [24]: Torchvision-like package of the DCNN with both 1D, 2D, and 3D operators.
- PyTorch-Deformable-Convolution-v2 [25]: Developed and published by Y. Kwon, this library implements a DCNN model based on the Pytorch operator *deform_conv2d*.

2.3. Models Tested

2.3.1. Small Model

The small model is composed of four stages of feature extraction, based on 2D convolution operations and a classifier. Figure 4 shows the schematic representation of the small model, where the dimension of the input and output tensor is defined as H × W × C, where H, W, C mean height, width, channels, respectively. The input to the model is a rgb image with size of 100 × 100, i.e., the input has dimension 100 × 100 × 3. Each stage is composed of 4 layers, as shown on the left side of Figure 4. The first layer is the 2D convolution operation with kernel size 3 × 3, padding of 1 (to maintain the same size in height and width of the input), and C filters. The BatchNorm layer, applies the normalization of the output values of the previous convolution layer; this helps to improve learning. The third layer applies the ReLU activation function. In the fourth layer MaxPool, applies the maximum value pooling with 2 × 2 window size; this layer reduces the input size by half.

In stage 1, the 2D convolution operation is applied with 16 filters, in the following stages the number of filters is increased twofold, 32 filters are generated in stage 2, then 64 filters are obtained in stage 3, and in stage 4, an output with 128 filters is produced. Note that the width and height dimension of the feature maps are reduced by half at each stage. The output of the last stage is a tensor with size of 6 × 6 × 128, which is flattened to generate a vector of 1 × 4608; this vector is the input to the classifier. The classifier is a fully connected architecture with two hidden layers and one unit in the output layer. It incorporates 40608 units in the input layer, 64 and 32 units in the first and second hidden layers, respectively, and one neuron in the output layer with sigmoid activation function. For the training process, we employ the Adam optimizer with 1×10^{-3} learning rate, and 60 epochs.

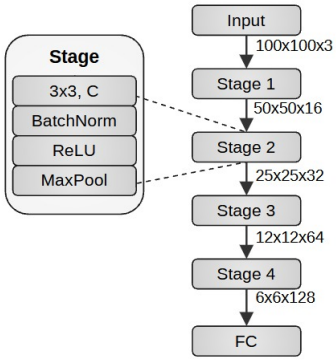


Figure 4. Graphical description of the CNN small model implemented during the experiments of Section 4.1.

2.3.2. ResNet

The ResNet model proposed by He et al. [26] constitutes five stages, as shown in Figure 5 (left), where the first stage, named conv1, is composed of one convolution layer with kernel 7 × 7, stride two, followed by a MaxPooling layer with window size three and stride two. The following stages, conv2_x to conv5_x, are composed of stacked residual blocks organized in a sequential manner, where x indicates the number of repetitions that the residual block will take in each stage.

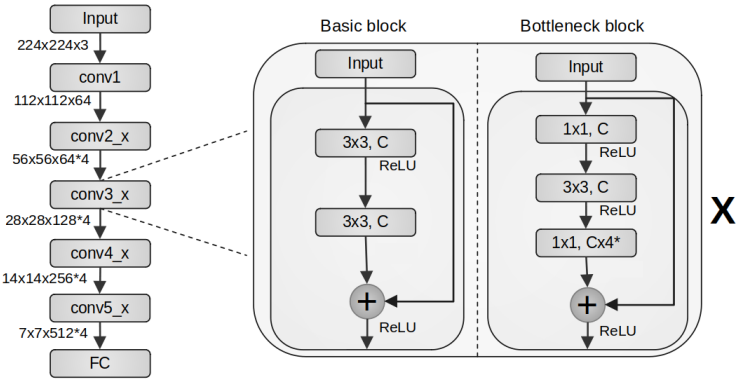


Figure 5. The ResNet model is shown on the left side. ResNet's residual block types are the basic and bottleneck blocks, shown on the right side of the figure. C means the respective channel (derived from [26]).

The idea behind the residual block is to modify the flow of the gradient (which updates the weights of the network during training) by adding a connection for an optimal gradient flow, reducing the gradient vanishing problem, thus establishing a shortcut to avoid losing the input information. This process is performed by adding the input signal to the output of the last convolutional layer of each residual block. In a residual block, the input is called residual information.

Each residual block can be replicated x times in a stage. Therefore, deeper networks can be built to preserve the input information. Common ResNet architectures are ResNet-18, ResNet-34, ResNet-50, ResNet-101, and ResNet-152. The residual block for ResNet-18 and ResNet-34 architectures is called basic block, and consist of two convolution layers with a 3 × 3 kernel without a MaxPooling layer, whereas the residual block for ResNet-50 and ResNet-101 is named bottleneck block, which comprises three convolution layers, where the first and last layers use 1 × 1 kernel size, and the intermediate layer employs 3 × 3 kernel size. In addition, in case the input size does not match (spatially or in number of filters), the residual block incorporates a mechanism to project the input to the

required dimensions. ResNet-18, ResNet-34, ResNet-50, and ResNet-101 architectures were considered in this research.

2.4. Metrics

The metrics used in this study are explained below.

- Accuracy: In this study, we have balanced classes for all the data sets. Accuracy is defined as the sum of the number of true positives and true negatives (correct observations) divided by the total number of examples [27]; it is computed as follows.

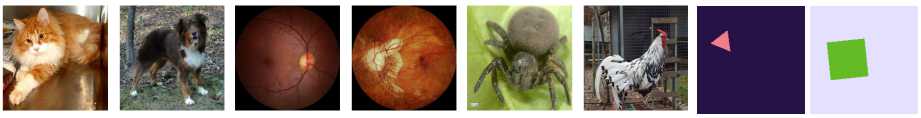
$$\text{Accuracy} = \frac{TP + FP}{TP + FP + TN + FN}, \tag{7}$$

  where TP is a correct observation put in class 1; FP is an incorrect observation put in class 1; TN is a correct observation put in class 2; FN is an incorrect observation put in class 2. Then, the accuracy describes how the model performs across all classes.
- Validation accuracy with transformation: A scale transformation is applied to evaluate the robustness of the model.
- Flops: Denotes the number of floating point operations executed per second. Essentially, the Flops measure the processing speed of models, thereby serving as an indicative benchmark for assessing their computational cost. We used it as a measure of training and testing time in our research. It is measured through one image or per batch conformed of a group of images.
- Parameters: The internal variables learned by the model from the training data.
- Time execution: Amount of time that the process takes to execute.

2.5. Image Datasets

The image classification data sets are shown in Figure 6, described in Table 1, and listed bellow.

- Cats & Dogs [28]: Two classes of mammals with some characteristics in common, but at the same time, differences that are very clear to the human eye. Cats & Dogs images were collected by the Visual Geometry Group (VGG) at the University of Oxford.
- EyePACS-AIROGS-light-V2 [29,30]: This is an improved machine-learning-ready glaucoma dataset using a balanced subset of standardized fundus images from the Rotterdam EyePACS AIROGS divided into referable glaucoma (RG) and non-referable glaucoma (NRG). A binary classification where the change between the classes is focused on the presence or absence of specific details.
- Spiders & Chickens [31]: Taken from Animals10 conformed of quality animal images where two radically different animals were selected to analyze a complex spatial scenario.
- Shapes (Triangle & Square) [32]: Geometric shapes drawn randomly on a 200 × 200 RGB image. The perimeter, the position of each shape, the rotation angle (between 0° and 360°), the background color, and the filling color are selected randomly and independently for each image. Shapes is a dataset with controllable differences between each class. We expect to better reflect the deformable impact in a controllable scenario like this.



(a) (b) (c) (d) (e) (f) (g) (h)

Figure 6. Classes on Cats & Dogs, EyePACS, Spiders & Chickens, and Triangle & Square datasets, respectively. (**a**) Cat, (**b**) Dog; (**c**) NRG, (**d**) RG; (**e**) Spider, (**f**) Chicken; (**g**) Triangle (**h**) Square.

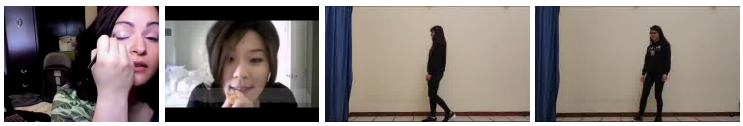
Table 1. Number of instances of each dataset involved in the study.

Dataset	Average Frames	Training Samples	Test Samples	Total Samples
Cats & Dogs	-	4000	1015	5015
EyePACS	-	4000	385	4385
Spiders & Chickens	-	2500	500	3000
Shapes	-	8000	2000	10,000
Apply Makeup	172	183	76	259
Human2	187	24	6	30

2.6. Video Datasets

The video data sets used mainly for action recognition are shown in Figure 7, described in Table 1, and explained below.

- Apply Makeup [33]: Two classes with little movement were taken from UCF101; Eye Makeup & Apply Lipstick. These classes were selected because there is no background movement, and although both activities are very close to each other, the characteristics that determine their label are based on spatial and temporal information, the spatial objects and locations in the image, and the hand movement.
- Human2: We collected 15 videos per class of a human walking forward and turning to the left. The background, clothes, and light conditions are fully controlled. The speed at which the activity is performed, and the body position varies between the instances.



(a) (b) (c) (d)

Figure 7. Video class frames from UCF101 [33] and Human2, respectively: (**a**) Apply Eye Makeup, (**b**) Apply Lipstick; (**c**) Forward, (**d**) Turn left.

3. Methodology

This work is inspired by the studies presented by Dai et al. in [12] and Zhu et al. in [13], where they propose a mechanism to increase the receptive field of a convolution layer by deforming its kernel, referred to as DCON. Although they reported better results applying DCONs in the last layers, they did not go further by extending it to different architectures of the ResNet model, starting from ResNet-18 up to ResNet-101. We also incorporate a shallow convolutional network architecture called a small model. With the use of this small model, it was possible to realize several experiments.

To identify the potential of a DCON to improve a model for feature extraction, different configurations were generated to replace the standard convolution layer with its deformable counterpart according to each model type. We compared the Vanilla model conformed of standard convolutions, and we replaced the standard convolutions stage by stage (Section 2.6). This substitution was performed to identify when a deformable layer can improve the model's performance, and when it can be counterproductive.

Each configuration was trained from scratch for each dataset, and validated. A scaling transformation was applied to the test images to assess the generalization of geometric transformations when using DCON layers. Figure 8 illustrates the multiple-image scales applied during testing. The validation accuracy, time, and the quantity of Flops achieved for each configuration were evaluated.

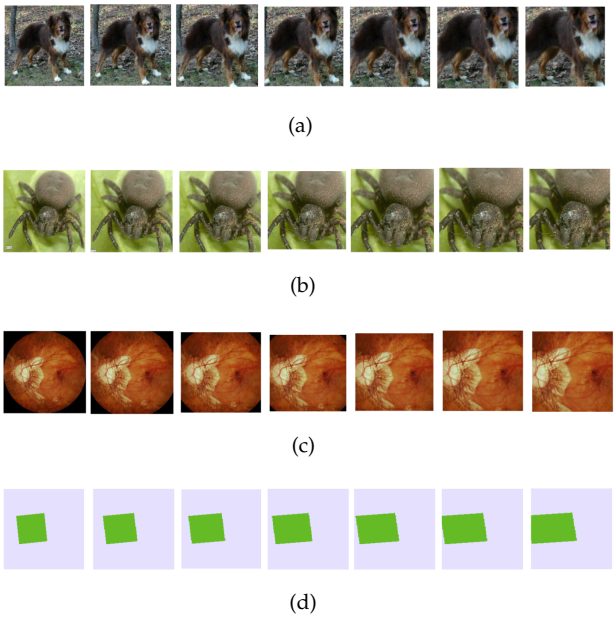


Figure 8. Image scale applied to validate models (from scale 1 to 1.6 in steps of 0.1). Cats & Dogs (**a**); Spiders & Chickens (**b**); EyePACS (**c**); Triangle & Square (**d**).

DCON in ResNet Models

To answer the question of where it is more suitable to use a deformable convolution layer in a deeper model, the ResNet model with architectures from ResNet-18 to ResNet-101 were tested (Figure 5). The switching of a standard convolution by its deformable counterpart (DCON) was conducted in the following configurations:

- Stage-by-stage: Refers to exchanging the standard convolution layer for its counterpart only in one stage at a time, i.e., swapping the standard convolution in the blocks (basic, bottleneck) of *conv2* stage for its counterpart and the other stages (*conv3*, *conv4*, *conv5*) remain intact.
- Combining two successive stages: The same as above, but in two successive stages, i.e., swapping the standard convolution by its counterpart in *conv4* and *conv5*, the stages *conv2* and *conv3* remain unchanged.
- Combining all stages: Here, all stages that are composed of basic or bottleneck blocks (*conv2*, *conv3*, *conv4* and *conv5*), include the deformable convolution instead of the standard one.

For architectures involving the basic block, the standard convolution layer was replaced by its deformable counterpart within each block. On the other hand, the standard 3×3 kernel convolution located in the middle was replaced by its deformable counterpart for the architectures involving bottleneck blocks.

In 3D ResNet, deformable convolution is applied only in convolutions with kernel size three (because, with size one, we already saw that there was no benefit). This experiment includes some executions with strides one or two. Stride two is present in all stages, but only in the first block due to the bottleneck. On the other hand, stride one is in all the blocks that contain the stage. The stride size affects the output spatial dimension and the computation load. More importantly, a more significant stride will capture more global features, but can also overlook helpful information. In neural networks, since the stride specifies the number of pixels by which the filter matrix moves across the input matrix, the choice of stride is a trade-off that needs to be carefully considered based on the specific task and dataset. With stride 2, the dimensionality will be reduced, we analyzed if the deformable convolution takes advantage of this.

4. Results

4.1. Small Model Analysis

In Table 2, we conducted a comparative study between DCNv2, DCNv3, and tvdcn for the image classification task using the Cats & Dogs, EyePacs, Spider & Chicken, and Shapes datasets, with training data between 2.5 K and 4 K; and using low-scale CNNs model composed of four convolutional layers for feature extraction (small model) and DCNv2 presented better results. The model performance using DCON was assessed by developing several configurations to incorporate DCON layers in the model. But first, in Table 2 we compared the implementations of DCNv2, DCNv3 and tvdcn getting better results with DCNv2.

Table 2. Results of applying different implementations of deformable convolution in the last layer of the small layer.

Dataset	DCNv2	DCNv3	tvdcn
Cats & Dogs	**0.73**	0.70	0.70
EyePACS	**0.73**	0.69	0.73
Spiders & Chickens	**0.86**	0.85	0.84
Shapes	**0.91**	0.89	0.87

The results on the considered configurations (Section DCON in ResNet Models) are shown in Table 3. The accuracy below is after applying the scaling operation illustrated in Figure 8 and as a result, it reflects how robust is the configuration in the Triangle & Square classes.

Table 3. The table shows the configurations that incorporate DCON into the model. Also, the number of parameters and the quantity of Flops (per batch) for each variation of DCON are listed (tvdcn implementation). The best result is shown in bold.

Config.	Total Parameters	Training Flops	Validation Flops	Accuracy
Vanilla	**394,769**	1.244 G	1.273 G	0.87
Stage1	395,498	1.339 G	1.368 G	0.91
Stage2	398,657	1.186 G	1.215 G	0.92
Stage3	402,545	1.031 G	1.059 G	0.93
Stage4	410,321	0.976 G	1.004 G	**0.96**
Stage1,2	399,386	1.281 G	1.31 G	0.89
Stage1,3	403,274	1.126 G	1.154 G	0.90
Stage2,3	406,433	0.973 G	1.002 G	0.95
Stage2,4	414,209	0.918 G	0.947 G	**0.96**
Stage3–4	418,097	**0.763 G**	**0.791 G**	**0.96**

4.1.1. Performance

When DCON is in the first layer, the quantity of Flops and the number of parameters increment by 7.63% (Tables 3 and 4, and Figure 9). Despite using DCON, accuracy often remains the same or even decreases. On the other hand, it achieves lower quantity of Flops and better performance when working with more complex features (from higher layers). For example, applying the quantity of Flops increments by 21.55% in stage 4, and combining two deformable stages in 3 and 4 the quantity of Flops reaches an improvement of 38.69% with an accuracy improvement between 0.01 or 0.03. In Tables 3 and 4, it is observed that the number of parameters can increment (caused by the DCONs); for example,

the implementation of tvdcn increments about 4.17% when applying the deformable convolution in the last layer. But, it can be beneficial if it works with more complex features. Applying DCON will increase or remain the same number of parameters without necessarily reflecting an improvement; for example in the first layer, no matter the dataset analyzed. This implies that the first features obtained through convolution are still so uninformative that they confuse the deformable ones. In other words, DCONs require pre-processing before learning their parameters. The model with DCONs that achieves the best relation between accuracy and the quantity of Flops is conv3–4 (Figure 10). Also, observe that all the datasets show the same behavior as the controlled one (Shapes): the latter the DCON is applied the better relation between performance and accuracy. Nonetheless, it is also illustrated that applying DCON in the first layer is not as convenient as applying it in the latter stages where the features are more complex. It is better to work with standard convolution than a DCON in the first stage. Also, the validation accuracy (Figure 11) that uses scale transformation is better when the model is a DCNN, denoting gained robustness in the model. This performance improvement is due to the capability of modeling geometric transformation once the best location of DCON is found.

Table 4. The quantity of Flops (per image and batch) and parameters in Vanilla model and with DCNNs implemented with DCNv2 and tvdcn. The best result is shown in bold.

	Flops per Image	Flops per Batch	Parameters DCNv2	Parameters tvdcn
Vanilla	38.87 M	1.244 G	0.395 M	**0.395 M**
Stage1	41.84 M	1.339 G	0.395 M	**0.395 M**
Stage2	37.07 M	1.186 G	0.394 M	0.397 M
Stage3	32.21 M	1.031 G	0.384 M	0.403 M
Stage4	30.49 M	0.976 G	0.337 M	0.41 M
Stage3,4	**23.83 M**	**0.763 G**	**0.326 M**	0.418 M
Stage1,2,3,4	25.00 M	0.829 G	**0.326 M**	0.418 M

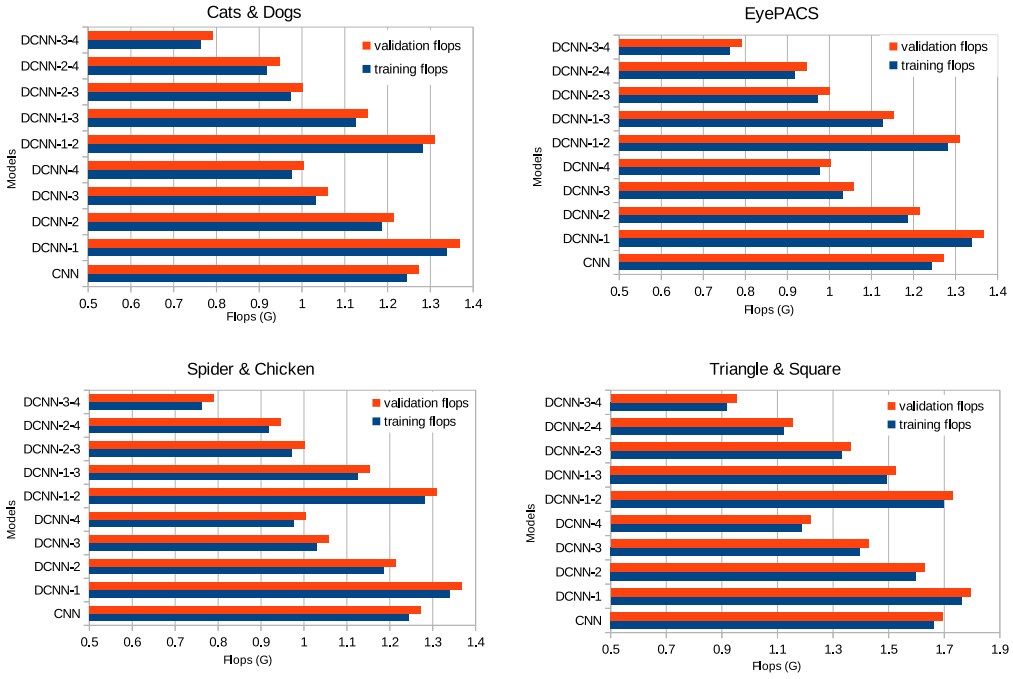


Figure 9. Comparison of the quantity of Flops (per batch) with Vanilla (base model) and applying the DCONs in different layers.

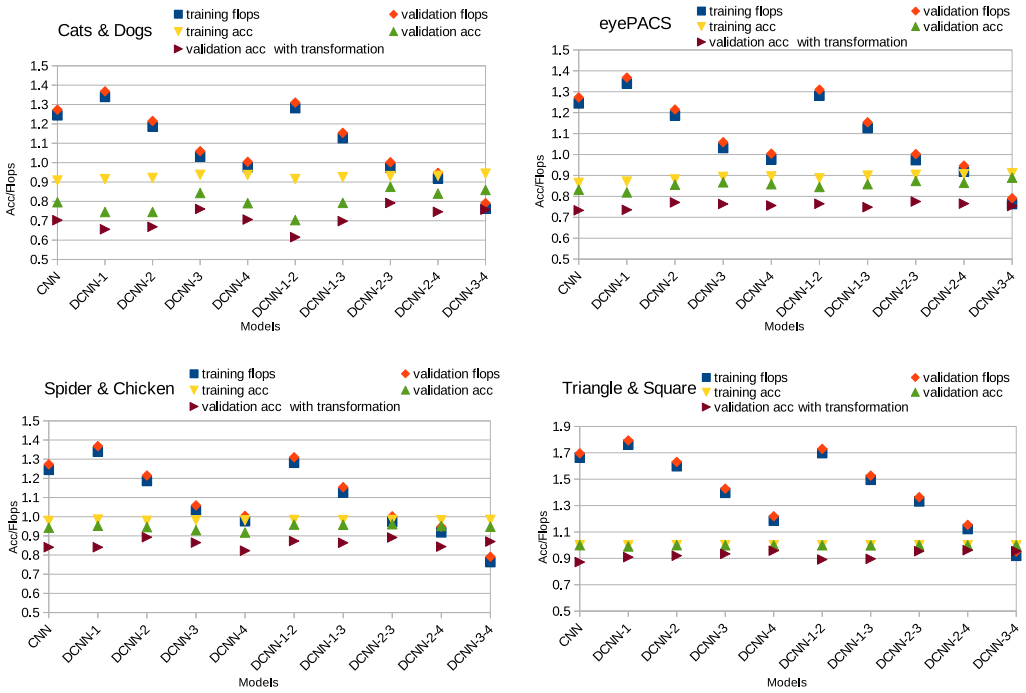


Figure 10. The figure overlaps the accuracy and the quantity of Flops to observe the best cases. We expect the accuracies to reach high values and the quantity of Flops to reach low ones.

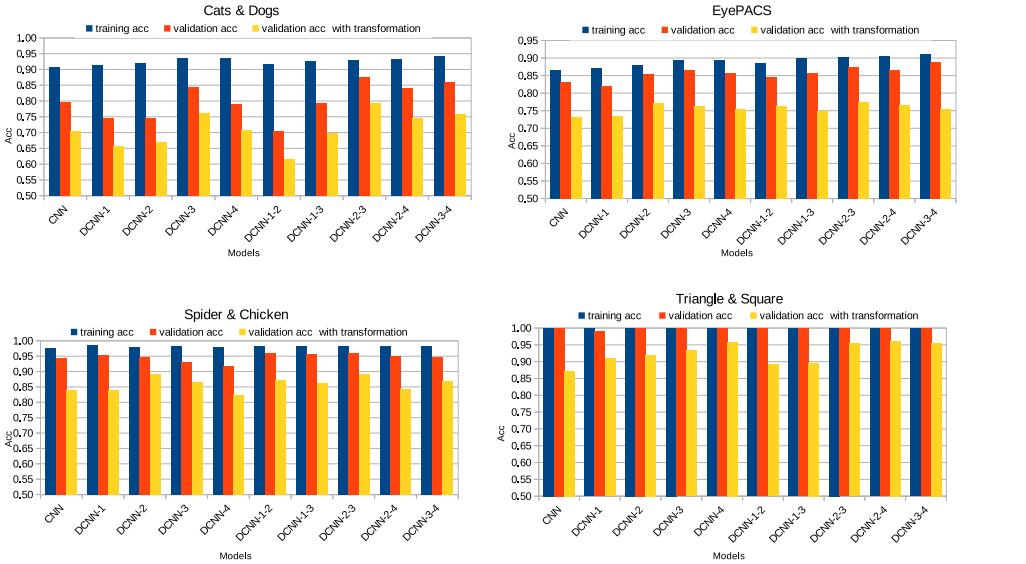


Figure 11. Comparison of accuracy with Vanilla (model base) and models applying the DCONs in different layers. DCONs are unsuitable for the first layer when features are still too simple. Especially if the Dataset contains complex features as Cats & Dogs, unlike Triangle & Square.

DCNNs show advantages because of their inherent adaptability. Nonetheless, DCONs can also become harmful or indifferent if we use them without previous knowledge acquired with standard convolution. Figure 12 compares the standard CNN (base model) and the best model with DCNNs (conv3–4). The DCNN with transformation is always over the one without deformable convolution, adding evidence that the DCNN generalizes geometric transformations. We confirm that using the DCON to calculate the most superficial features will have lower results than using complex features. The model tends to be more robust when placing the DCON in the last layers (Table 5). Although the base model reaches the best time execution, the time with DCONs increments only by 0.02 or 0.04 min.

The increments are not too far. These numbers are considering the experiments with best relation between accuracy and execution time from Vanilla model (Table 6).

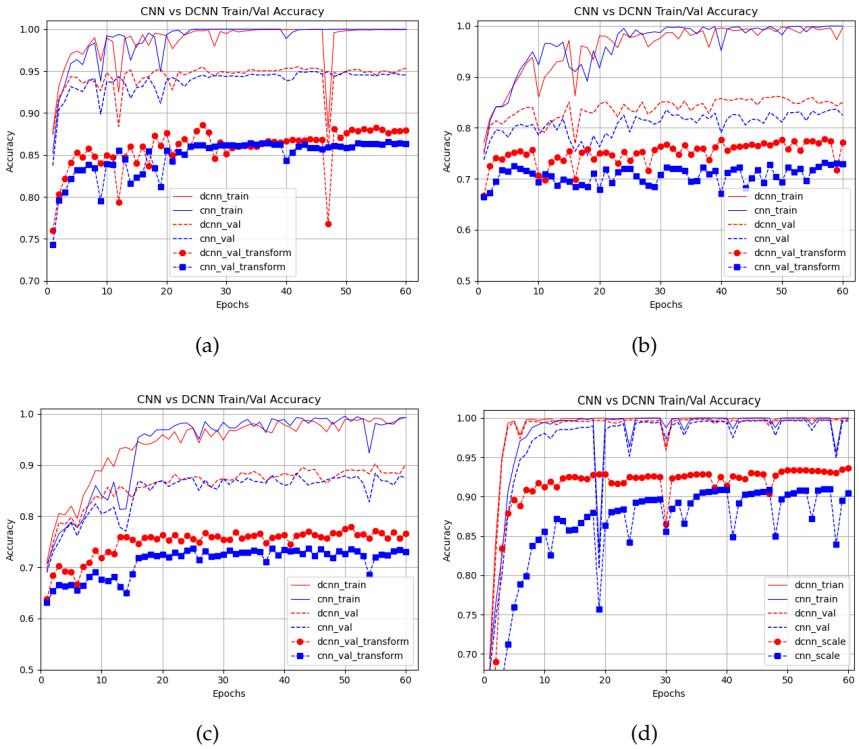


Figure 12. DCNN vs. Vanilla during training (thin lines) and validation, without transformation (thin dotted lines), and with transformation(thick lines). (**a**) Spiders & Chickens (from Animal-10 dataset); (**b**) Cats & Dogs; (**c**) EyePACS; (**d**) Shapes.

Table 5. Validation accuracies with scale transformation in the implementations DCNv2 and tvdcn. The best result is shown in bold.

Config.	**Datasets**							
	Cats & Dogs		**EyePACS**		**Spiders & Chickens**		**Shapes**	
	Acc DCNv2	**Acc tdvcn**	**Acc DCNv2**	**Acc tdvcn**	**Acc DCNv2**	**Acc tdvcn**	**Acc DCNv2**	**Acc tdvcn**
Vanilla	0.73	0.70	0.73	0.73	0.86	0.84	0.91	0.87
Stage1	0.72	0.66	0.75	0.74	0.85	0.84	0.90	0.91
Stage2	0.74	0.67	0.76	**0.77**	0.86	**0.89**	0.91	0.92
Stage3	0.75	0.76	**0.77**	0.76	0.86	0.86	0.92	0.93
Stage4	0.75	0.71	0.76	0.76	0.86	0.82	0.90	**0.96**
Stage1,3	**0.77**	0.70	0.75	0.74	**0.92**	0.86	0.89	0.89
Stage1,4	0.69	0.70	0.76	0.75	**0.92**	0.83	0.92	0.92
Stage2,4	**0.77**	0.76	**0.77**	0.75	0.91	0.84	0.92	**0.96**
Stage3,4	**0.77**	0.76	**0.77**	0.75	0.88	0.87	**0.93**	**0.96**
Stage1,2,3,4	**0.77**	**0.80**	**0.77**	0.67	0.87	0.87	**0.93**	0.78

Table 6. Time execution in image classification datasets.

Config.	Datasets			
	Cats & Dogs Time (min)	EyePACS Time (min)	Spiders & Chickens Time (min)	Shapes Time (min)
Vanilla	**0.33**	**0.52**	**0.12**	**0.27**
Stage1	0.41	0.6	0.17	0.33
Stage2	0.41	0.59	0.16	0.33
Stage3	0.37	0.55	0.15	0.31
Stage4	0.35	0.54	0.14	0.29
Stage1,3	0.47	0.70	0.20	0.46
Stage1,4	0.46	0.70	0.20	0.44
Stage2,4	0.46	0.70	0.20	0.44
Stage3,4	0.39	0.58	0.15	0.32
Stage1,2,3,4	0.52	0.71	0.24	0.43

4.1.2. Kernels

Common choices of kernel sizes are three or five. The kernel size is related to the number of parameters and the capability of generalizing. Table 7 shows the parameters and the quantity of Flops of the best DCNN models already discussed. The increase in parameters means not necessarily decreasing the processing speed involved, probably because the added parameters are the ones that help the DCNN learn better representations. The standard convolutions require fewer parameters, but a higher quantity of Flops.

Table 7. DCNN parameters and the quantity of Flops (per batch) for the small model with different kernel sizes. The best result is shown in bold.

	Parameters			Training Quantity of Flops			Validation Quantity of Flops		
	1 × 1	3 × 3	5 × 5	1 × 1	3 × 3	5 × 5	1 × 1	3 × 3	5 × 5
Vanilla	0.833 M	**0.395 M**	**0.568 M**	0.291 G	1.244 G	3.404 G	0.329 G	1.273 G	3.433 G
Stage2,3	0.833 M	0.406 M	0.658 M	0.181 G	0.973 G	4.956 G	0.218 G	1.002 G	4.985 G
Stage2,4	0.833 M	0.414 M	0.718 M	0.142 G	0.918 G	4.39 G	0.18 G	0.947 G	4.418 G
Stage3,4	0.833 M	0.418 M	0.748 M	**0.125 G**	**0.763 G**	3.19 G	**0.162 G**	**0.791 G**	3.218 G
Stage4	0.833 M	0.41 M	0.688 M	0.189 G	0.976 G	**3.014 G**	0.226 G	1.004 G	**3.042 G**

Figure 13 illustrates that the ones with kernel 5 × 5 have similar accuracies to the ones with kernel 3 × 3. But, as shown in Table 7, the quantity of Flops drastically increases with kernel 5 × 5. On the other hand, a kernel 1 × 1 always obtains less accuracy because, no matter the flexibility DCONs should bring, this kernel´s size lacks context and only generates local features.

4.2. DCNN in ResNet Models for Image Classification

To further investigate the DCON performance in deeper models, we use the ResNet architectures to test the DCON, starting from ResNet-18 to ResNet-101. As in the small model, to evaluate the quality of the features, binary classification was conducted using the datasets: Cats & Dogs, EyePACS, Spider & Chickens, and Shapes. We evaluated the accuracy achieved, the training time, and the quantity of Flops for each configuration. The results are presented in the following subsections.

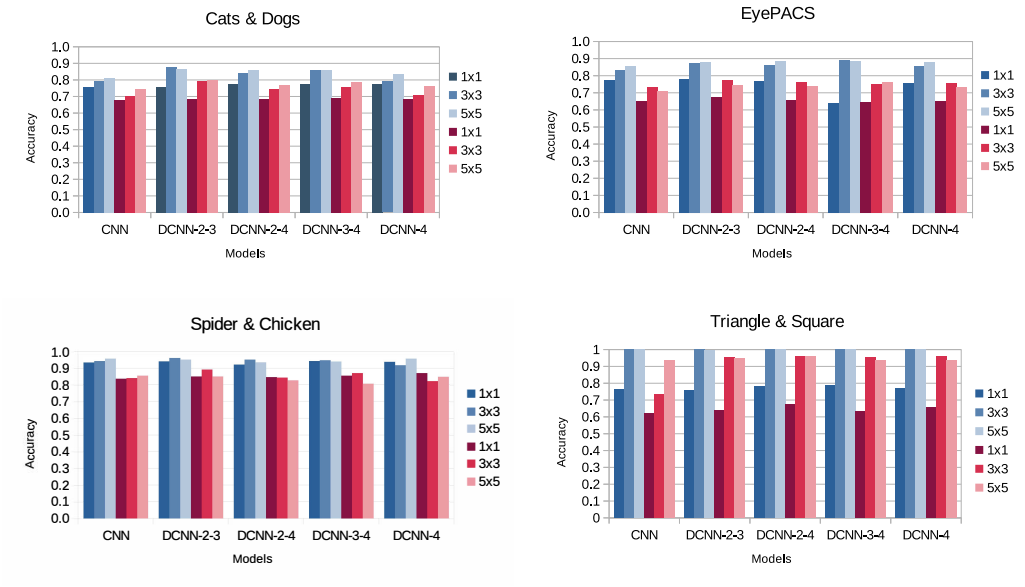


Figure 13. Validation accuracies without transformation (blue) and with scale transformation (red) using different kernel sizes.

4.2.1. Accuracy Test

The ResNet-18 architecture performance is shown in Table 8. In the stage-by-stage configuration, we observed that, when using the deformable convolution in conv2, the poorest accuracy was achieved. The accuracy is lower even than the vanilla configuration by 0.001 to 0.006; for all datasets except Shapes, where this configuration obtained the highest accuracy, with 0.951. It was found, conversely, that when applying the deformable convolution in conv4, the accuracy results are better than with the stage-by-stage configurations by approximately 0.04, except in the Shapes dataset with 0.949.

Table 8. Accuracy using ResNet-18 vanilla model and DCNN configurations. The best result is shown in bold.

Config.	**Datasets**			
	Cats & Dogs	**Spiders & Chickens**	**EyePACS**	**Shapes**
Vanilla	0.756	0.907	0.772	0.936
Conv2	0.753	0.901	0.771	**0.951**
Conv3	0.794	0.904	0.772	0.929
Conv4	0.841	0.935	0.804	0.935
Conv5	0.785	0.931	0.773	0.949
Conv2,4	**0.873**	0.928	**0.842**	0.927
Conv2,5	0.853	0.923	0.777	0.940
Conv3,5	**0.873**	0.929	0.814	0.918
Conv4,5	0.857	**0.940**	0.804	0.944
Conv2,3,4,5	0.869	0.934	0.830	0.942

Combining two stages, conv4, and conv5, the DCNN achieves higher accuracy in all datasets, except in Shapes dataset (which is under controlled conditions). It is observed that if classes are too different between them, it is convenient to combine the two last stages. But if not, the first and the second last achieve better results by 0.04 to 0.01. The results suggest that when classes have very different characteristics from each other (at least visually for people), the flexibility acquired through deformable convolution is beneficial in the last layers. On the contrary, if classes are visually closer between each other, it is helpful to apply

deformation in the first layer combined with the last one when simple features are extracted (such as edges). This same behavioral trend is maintained at depths 34, 50, and 101.

The performance of the ResNet-34 architecture is shown in Table 9. A similar behavior as the ResNet-18 architecture was observed here. The best result using the stage-by-stage configuration is obtained using conv4, reaching the highest accuracy for the Cats & Dogs dataset. Whereas combining two consecutive stages, conv4, and conv5, achieved the best accuracy on the Spiders & Chickens data set. Combining all stages achieved the best accuracy for the remaining two data sets.

Table 9. Accuracy using ResNet-34 vanilla model and DCNN configurations. The best result is shown in bold.

Config.	Datasets			
	Cats & Dogs	**Spiders & Chickens**	**EyePACS**	**Shapes**
Vanilla	0.755	0.907	0.781	0.964
Conv2	0.770	0.912	0.785	0.968
Conv3	0.808	0.916	0.794	0.936
Conv4	0.852	0.932	0.836	0.934
Conv5	0.818	0.920	0.792	0.958
Conv2,4	**0.870**	0.933	0.837	0.961
Conv2,5	0.818	0.926	0.795	0.933
Conv3,5	0.854	**0.937**	0.814	0.918
Conv4,5	0.842	0.935	0.811	0.932
Conv2,3,4,5	0.847	0.934	**0.849**	**0.979**

Contrary to the ResNet-18 and ResNet-34 architectures, the performance of the ResNet-50 architecture exhibited better performance by replacing standard convolution for its deformable counterpart in the stage-by-stage configurations in Cats & Dogs and Shapes datasets and combining the consecutive stages conv4 and conv5 in the EyePACS dataset. Combining all stages delivers the poorest performance of all configurations, even the vanilla configuration, except for the Spiders & Chickens dataset, where this configuration presents the highest accuracy. These results are shown in Table 10.

Table 10. Accuracy using ResNet-50 vanilla model and DCNN configurations. The best result is shown in bold.

Config.	Datasets			
	Cats & Dogs	**Spiders & Chickens**	**EyePACS**	**Shapes**
Vanilla	0.725	0.897	0.787	0.727
Conv2	0.745	0.904	0.795	0.947
Conv3	0.769	0.916	0.808	0.949
Conv4	0.789	0.917	0.798	0.621
Conv5	0.764	0.908	0.804	**0.970**
Conv2,4	**0.829**	0.913	**0.820**	0.833
Conv2,5	0.621	0.907	0.786	0.931
Conv3,5	0.623	0.911	0.790	0.936
Conv4,5	0.645	0.914	0.806	0.917
Conv2,3,4,5	0.688	**0.932**	0.666	0.513

In ResNet-101, the conv4 configuration in layer-by-layer basis exhibits the highest accuracy than the other layer-by-layer configurations (Table 11). Combining the two stages conv4 and conv5, good results are achieved in the Spiders & Chickens and Shapes datasets.

However, the accuracy achieved is very low in the Cats & Dogs and EyePACS datasets. In contrast to the above, combining all stages, high-accuracy results were obtained in the Spiders & Chickens and EyePACS datasets, where this configuration presented the best accuracy.

Table 11. Accuracy using ResNet-101 vanilla model and DCNN configurations. The best result is shown in bold.

Config.	Datasets			
	Cats & Dogs	Spiders & Chickens	EyePACS	Shapes
Vanilla	0.768	0.911	0.790	0.869
Conv2	0.768	0.919	0.795	0.871
Conv3	0.762	0.912	0.773	0.925
Conv4	0.788	0.917	**0.816**	0.585
Conv5	0.723	0.913	0.721	0.766
Conv2,4	**0.824**	0.905	0.810	0.930
Conv2,5	0.572	0.913	0.571	0.729
Conv3,5	0.786	0.916	0.636	0.599
Conv4,5	0.682	0.920	0.652	**0.934**
Conv2,3,4,5	0.705	**0.928**	**0.816**	0.815

4.2.2. Training Time and the Quantity of Flops

In addition to accuracy, we also took into account the training time and computational resources needed for model execution. The aim is to determine the optimal stage and configuration for replacing the standard convolution with its deformable counterpart. For this purpose, we measure the training time and the quantity of Flops. The results obtained along these experiments are shown in Table 12 for ResNet-18, Table 13 for ResNet-34, Table 14 for ResNet-50 and Table 15 for ResNet-101.

Table 12. Training time, Flops (per image), and parameters in ResNet-18 configurations. The best result is shown in bold.

Config.	Training Time in Datasets (min)					
	Cats & Dogs	Spiders & Chickens	EyePACS	Shapes	Flops	Parameters
Vanilla	**1.003**	**0.533**	**1.322**	**1.532**	1.818 G	**11.17 M**
Conv2	1.570	0.883	1.930	2.550	1.551 G	11.23 M
Conv3	1.248	0.716	1.605	2.002	1.499 G	11.28 M
Conv4	1.122	0.644	1.467	1.754	1.456 G	11.39 M
Conv5	1.053	0.606	1.436	1.639	1.435 G	11.61 M
Conv2,4	1.701	0.949	1.982	2.805	1.189 G	11.89 M
Conv2,5	1.615	0.892	1.868	2.677	1.168 G	11.67 M
Conv3,5	1.311	0.702	1.564	2.063	1.116 G	11.72 G
Conv4,5	1.197	0.701	1.532	1.931	**1.073 G**	11.83 M
Conv2,3,4,5	2.005	1.204	2.309	3.515	0.486 G	12.00 M

Training time is the lowest in the vanilla model for all architectures and all datasets. In the stage-by-stage configuration, starting from conv2, the training time is higher than the vanilla model, as the deformable convolution is introduced in deeper stages, the training time decreases but is not able to be lower than the vanilla model. Conv5 configuration generally has the lowest training time of all the configurations, where the deformable convolution is applied.

Table 13. Training time, Flops (per image), and parameters in ResNet-34 configurations. The best result is shown in bold.

Config.	Training Time in Datasets (min)					
	Cats & Dogs	Spiders & Chickens	EyePACS	Shapes	Flops	Parameters
Vanilla	**1.346**	**0.639**	**1.407**	**1.713**	3.670 G	**21.28 M**
Conv2	2.200	1.150	2.230	3.440	3.269 G	21.37 M
Conv3	1.873	0.982	1.974	2.854	2.986 G	21.51 M
Conv4	1.802	0.925	1.858	2.675	2.481 G	22.00 M
Conv5	1.488	0.715	1.527	2.039	3.068 G	21.96 M
Conv2,4	2.434	1.415	2.697	4.336	2.080 G	22.09 M
Conv2,5	2.106	1.205	2.362	3.674	2.67 G	22.06 M
Conv3,5	1.824	1.028	2.078	3.108	2.38 G	22.20 M
Conv4,5	1.885	1.035	1.986	2.954	1.879 G	22.68 M
Conv2,3,4,5	3.287	1.897	3.435	5.771	**0.794 G**	23.01 M

Table 14. Training time, Flops (per image), and in ResNet-50 configurations. The best result is shown in bold.

Config.	Training Time in Datasets (min)					
	Cats & Dogs	Spiders & Chickens	EyePACS	Shapes	Flops	Parameters
Vanilla	**1.727**	**0.987**	**2.037**	**2.937**	4.109 G	**23.51 M**
Conv2	2.120	1.26	2.500	3.780	3.908 G	23.55 M
Conv3	1.987	1.186	2.383	3.538	**2.363 G**	23.74 M
Conv4	1.951	1.168	2.307	3.435	3.488 G	23.88 M
Conv5	1.799	1.059	2.119	3.051	3.780 G	23.88 M
Conv2,4	2.400	1.405	2.692	4.281	3.288 G	23.93 M
Conv2,5	2.180	1.272	2.451	3.901	3.580 G	23.93 M
Conv3,5	2.116	1.206	2.352	3.654	3.416 G	24.00 M
Conv4,5	2.013	1.238	2.350	3.551	3.160 G	24.25 M
Conv2,3,4,5	2.699	1.665	3.054	4.954	2.595 G	24.42 M

Table 15. Training time, Flops (per image), and parameters in ResNet-101 configurations. The best result is shown in bold.

Config.	Training Time in Datasets (min)					
	Cats & Dogs	Spiders & Chickens	EyePACS	Shapes	Flops	Parameters
Vanilla	**2.462**	**1.399**	**2.586**	**4.188**	7.831 G	**42.50 M**
Conv2	2.85	1.680	3.030	5.100	7.631 G	42.54 M
Conv3	2.707	1.604	2.900	4.820	7.467 G	42.62 M
Conv4	3.228	2.036	3.537	6.104	5.453 G	43.93 M
Conv5	2.496	1.426	2.629	4.302	7.503 G	42.87 M
Conv2,4	3.663	2.185	3.900	6.935	5.252 G	43.98 M
Conv2,5	2.785	1.641	3.027	5.107	7.303 G	42.92 M
Conv3,5	2.662	1.563	2.907	4.860	7.139 G	43.00 M
Conv4,5	3.311	2.023	3.546	6.258	5.124 G	44.30 M
Conv2,3,4,5	4.055	2.489	4.281	7.628	**4.559 G**	44.47 M

When combining two consecutive stages, conv4 and conv5, the training time increases slightly, greater than the conv5 configuration of the stage-by-stage mode. When combining all stages, the training time is significantly higher than all other configurations.

Contrary to the training time, the processing speed increases as the deformable convolution is introduced in more layers. In the case of ResNet-18, using the vanilla configuration, 3.670 GFlops are reached, while in the conv5 configuration, 3.068 GFlops are achieved, and when all stages are combined, 0.794 GFlops are attained.

4.3. DCNN's through Time Dimension

In this section, 3D ResNets with depths 18, 34, 50, and 101 were analyzed using controlled videos where the person is walking forward and turning to the left. The controlled conditions will permit the model to focus on the movement, thus allowing us to analyze the behavior of deformable convolutions over time. Also, the models were analyzed in two similar activities of applying makeup. The activities were taken from UCF101 video dataset to observe the behavior of the DCON under real-world conditions.

Using Resnet-18 (Table 16), the best results for Apply Makeup were obtained by adding the deformation to the stage 5 with stride 1 or to the stages 3,4,5 with a result of 0.552 which is too low. In Human2, it was better to apply in stage 2 or in stage 5 than in the others. Then, with ResNet-34 (Table 17) a combination of stages 2 and 5 results in 0.592 in Apply Makeup and in Human2 is better to apply in the stage 2. The results suggest that the features are relatively closer between each class and probably the model and datasets are still insufficient for capturing time dimension to differentiate between classes (specially in Human2). Also, using deformable convolutions combining stages tends to saturate the model and not give a good result.

Table 16. Parameters, the quantity of Flops, and Accuracy in ResNet-18. The number of parameters increases when using DCNNs but the number of Flops diminishes. The best result is shown in bold.

	Parameters	**Training Quantity of Flops**	**Apply Makeup Accuracy**	**Human2 Accuracy**
Vanilla	**33.21 M**	25.00 G	0.526	0.666
Stride 1 Conv2	33.205 M	66.55 G	0.408	**1**
Stride 1 Conv3	34.32 M	24.51 G	0.500	0.833
Stride 1 Conv4	35.44 M	24.103 G	**0.552**	0.833
Stride 1 Conv5	37.68 M	24.20 G	0.487	**1**
Stride 1 Conv2,4	36.191 M	79.404 G	0.382	0.833
Stride 1 Conv2,5	38.43 M	79.666 G	0.382	0.667
Stride 1 Conv3,5	38.80 4M	63.105 G	0.368	0.667
Stride 1 Conv4,5	39.92 M	**23.25 G**	0.395	0. 666
Stride 1 Conv3,4,5	-	-	-	-
Stride 1 Conv2,3,4,5	41.79 M	28.53 G	0.474	**1**
Stride 2 Conv3,4,5	34.51 M	24.64 G	**0.552**	0.833

Table 17. Parameters, the quantity of Flops, and Accuracy in ResNet-34. The number of parameters increases when using DCNNs but the number of Flops diminishes. The best result is shown in bold.

	Parameters	**Training Quantity of Flops**	**Apply Makeup Accuracy**	**Human2 Accuracy**
Vanilla	**63.52 M**	38.19 G	0.368	**0.833**
Stride 1 Conv2	64.634 M	0.125 T	0.474	**1**
Stride 1 Conv3	66.127 M	36.992 G	0.5	0.5
Stride 1 Conv4	71.726 M	34.884 G	0.474	0.5
Stride 1 Conv5	70.98 M	36.85 G	0.513	0.833
Stride 1 Conv2,4	72.846 M	0.116 T	0.526	0.333
Stride 1 Conv2,5	72.099 M	0.121 T	**0.592**	0.5

Table 17. *Cont.*

	Parameters	Training Quantity of Flops	Apply Makeup Accuracy	Human2 Accuracy
Stride 1 Conv3,5	73.592 M	95.07 G	0.382	0.667
Stride 1 Conv4,5	79.19 M	**33.54 G**	0.421	0.667
Stride 1 Conv3,4,5	-	-	-	-
Stride 1 Conv2,3,4,5	-	-	-	-
Stride 2 Conv4,5	82.92 M	40.99 G	0.487	0.667

With deeper ResNets, the accuracy in Apply Makeup reaches 0.618 and 0.608 when deformable convolution was applied to stage 5 with depth 50 (Table 18) and 101 (Table 19) respectively. Then, the highest value for Human2 dataset is still 1, applying deformation in stage 2. More experiments are needed, but we can observe some tendency related to the last section. As classes show more visually different the deformation is better in the last layers.

Table 18. Total parameters, the quantity of Flops, and Accuracy for ResNet-50. With Stride 1 the parameters and the quantity of Flops are not improving. The best result is shown in bold.

	Parameters	Training Quantity of Flops	Apply Makeup Accuracy	Human2 Accuracy
Vanilla	**46.20 M**	30.46 G	0.513	0.833
Stride 1 Conv2	46.763 M	92.264 G	0.408	0.667
Stride 1 Conv3	47.32 M	29.97 G	0.474	**1**
Stride 1 Conv4	49.94 M	28.96 G	0.474	0.833
Stride 1 Conv5	49.19 M	29.92 G	**0.618**	0.500
Stride 1 Conv2,4	50.495 M	88.254 G	0.474	0.667
Stride 1 Conv2,5	49.749 M	90.834 G	0.434	0.833
Stride 1 Conv3,5	50.309 M	78.089 G	0.382	0.833
Stride 1 Conv4,5	52.92 M	**28.42 G**	0.566	0.666
Stride 1 Con3,4,5	-	-	-	-
Stride 1 Conv2,3,4,5	54.60 M	32.22 G	0.421	0.500
Stride 2 Conv3,4,5	48.82 M	29.73 G	0.500	0.333

Table 19. Parameters, the quantity of Flops, and Accuracy in ResNet-101. The best result is shown in bold.

	Parameters	Training Quantity of Flops	Apply Makeup Accuracy	Human2 Accuracy
Vanilla	**85.249 M**	41.962 G	0.355	0.833
Stride 1 Conv2	85.809 M	0.123 T	0.460	0.833
Stride 1 Conv3	86.37 M	41.48 G	**0.697**	0.666
Stride 1 Conv4	102 M	**35.35 G**	0.421	0.5
Stride 1 Conv5	88.24 M	41.43 G	0.605	0.666
Stride 1 Conv2,4	0.102 G	0.105 T	0.421	0.5
Stride 1 Conv2,5	88.795 M	0.121 T	0.473	0.667
Stride 1 Conv3,5	89.355 M	0.163 T	0.5	0.667
Stride 1 Conv4,5	-	-	-	-
Stride 1 Conv3,4,5			-	-
Stride 2 Conv3,4,5	87.862 M	41.231 G	0.592	0.5
Stride 2 Conv4,5	87.489 M	41.394 G	0.474	0.666

5. Discussion

In this study, we focused on analyzing how to use the DCON. First, on a small CNN model applied to image classification. After that, to a ResNet with different depths, extending the study to 3D-ResNet. From image classification to action recognition, we analyzed the feasibility of DCNNs. The implementation of DCONs applied in this study where tvdcn and Pytorch-Deformable-Convolution-v2. Both implementations produce similar results.

In the small model, the highest accuracy was reached by replacing the last two stages of the standard convolution with its deformable counterpart improving the accuracy by 0.003 and diminishing the quantity of Flops by 38.69%. During training, it was identified that when deformable convolution is employed in the last layers, the model can converge faster, as shown in Figure 9. The execution time is shorter when using the vanilla model, which reaches 0.33 min, while when combining the deformable in all stages, the execution time increases to 0.52 min. In the stage 4 configuration, 0.35 min is consumed, which is the shortest time achieved when incorporating the deformable convolution into the small model.

When introducing the deformable convolution to the small model in the first stage, the computational cost is higher than the vanilla model, as the deformable convolution is introduced in deeper stages, the computational cost decreases. The lowest computational cost is obtained by combining stage 3,4, where 0.763 GFlops is achieved.

Based on the experiments in the small model, the validation accuracy that uses transformations is better when the model includes DCONs, denoting gained robustness in the model. Nonetheless, DCON operations are hyper-resource-thirsty; paying attention to its implementation will be a key to acquiring benefits. In Figure 11, it is illustrated that when applying deformable convolutions in later layers (preferably in two stages), the accuracy is higher and the quantity of Flops diminishes which is the best scenario.

The kernel with dimension 3×3 suits to be the optimal to use for deformable convolution. The accuracy decreases if the kernel size is reduced to size 1×1, because the receptive field displacement mechanism is nullified while increasing the kernel to size 5×5 does not present a significant improvement in accuracy. On the contrary, the computational cost increases.

When introducing deformable convolution in deeper models using ResNet architectures, we noticed that the behavior is similar to the small model, as the best accuracy results were generated by introducing deformable convolution in deeper stages. In ResNet-18 and ResNet-34, for all datasets, better accuracy results are achieved by combining two stages incrementing the accuracy by 0.117 and 0.115.

Extending the research to video action recognition tasks using the 3D ResNet models, we noticed that the behavior is similar to that of their 2D counterparts but with some differences. The introduction of deformable convolution in the last layers helped to increase the accuracy about 0.224. Nevertheless, the combination of consecutive stages in 3D ResNet models tends to get saturated.

The best accuracy results in all evaluated datasets were usually obtained in the smaller depth models, ResNet-18 and ResNet-34 0.842 and 0.849 respectively. Poor performance was observed in deeper models, such as ResNet-50 and ResNet-101 0.820 and 0.816 respectivelyin 2D. We hypothesize that this drawback is caused by accumulating many deformable layers at very deep levels, where probably the bottleneck mechanism may contribute to the poor performance of these deeper architectures.

It is also observed that the depth of the network does not necessarily imply better representations (as indicated by [34]). Only a few blocks learn relevant information, while the other blocks and stages may just be confusing the network and decreasing its performance.

As deformable convolution is incorporated into the evaluated ResNet architectures, the training time increases. The highest training time using the stage-by-stage configuration is obtained when deformable convolution is incorporated in the conv1 stage; as deformable convolution is incorporated in deeper layers, the training time decreases. When

incorporating deformable convolution at all stages, the training time exceeds the training time of the vanilla configuration by up to three times.

The computational cost required to execute the modified ResNet models with the deformable convolution measured in the quantity of Flops is lower than that of the vanilla models. The lowest number of Flops (up to half) is achieved by combining all stages, while the worst case is when deformable convolution is incorporated in the conv2 stage, where the number of Flops is marginally lower than the vanilla configuration. DCNN helps to focus on significant information. Then, the quantity of Flops will decrease when the accuracy increases, regardless of whether the parameters increase. Therefore, their flexibility helps to focus the resources automatically.

We collected evidence that more layers of DCNNs do not necessarily lead to better results. The accumulation of deformable convolutional layers will slow down the network, and may even saturate the network, preventing it from achieving any result. So the guideline proposed is: before applying the DCNN to improve the model features maps, we suggest that the model uses a standard convolution layer. Specifically, in ResNets, we highly recommend to use two stages of deformable convolution. The fourth stage consists of the convolutions incremented depending on the depth selected, we suggest to include de deformable convolution in the third and the fifth stages. On the other hand, in different models, it will depend on the structure but we still suggest to look for the later layers. Nonetheless, more studies about the data involved are necessary to understand the influence and needs of the application involved.

6. Conclusions

The implementation of deformable convolution can help to improve the performance of a model, allowing the generalization of geometrical transformations such as scaling. As DCNNs alleviate geometric transformations, we encourage the readers to use DCNNs mainly if their resources are limited to work with data augmentation.

According to the obtained accuracy, training time, and the model performance results, it is optimal to introduce the deformable convolution in the last stages of the CNN since the training time and computational cost are lower than the ones obtained with vanilla models, and the accuracy is superior. When working with 3D-ResNets, the same trend can be observed. Nevertheless, we believe that the DCNN may lose its advantages with greater depth; therefore, more experiments are required.

In future work, we are seeking more experiments using DCNNs in the field of action recognition, which is a promising branch of computer vision where the application of deformable networks has not been explored as much. DCNNs improve the feature extraction capability and adapt to complex changes in the input, which is becoming a fundamental part of a recognition system. However, we propose to look for a quantitative paradigm that permits measuring datasets to interpret the information learned by the networks and to evaluate the opportunities that deformable networks can offer for action recognition.

Author Contributions: Conceptualization, A.B.M.; methodology, A.B.M.; software, A.B.M. and V.R.B.; validation, A.B.M. and V.R.B.; formal analysis, A.B.M., V.R.B. and L.A.R.; investigation, A.B.M.; resources, A.B.M.; data curation, A.B.M.; writing—original draft preparation, A.B.M. and V.R.B.; writing—review and editing, A.B.M., V.R.B., L.A.R. and R.D.H.; visualization, A.B.M.; supervision, L.A.R.; Project administration, L.A.R.; funding acquisition, A.B.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by CONHACyT through a doctoral scholarship granted to Andrea Burgos Madrigal CVU 702730.

Data Availability Statement: The research data are open to Public by the authors themselves. Human2 is a dataset created by us and it is available, the reader is invited to contact the authors of this article to request it.

Acknowledgments: We are grateful for the resources provided by the supercomputer LNS "Laboratorio Nacional de Supercómputo del Sureste de México" to carry out part of this research.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

DCNN	Deformable Convolutional Network
DCON	Deformable Convolution
Flops	Floating-type operations

References

1. Liu, Z.; Mao, H.; Wu, C.; Feichtenhofer, C.; Darrell, T.; Xie, S. A ConvNet for the 2020s. In Proceedings of the 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), New Orleans, LA, USA, 18–24 June 2022; pp. 11966–11976.
2. Krichen, M. Convolutional Neural Networks: A Survey. *Computers* **2023**, *12*, 151. [CrossRef]
3. Purwono, P.; Ma'arif, A.; Rahmaniar, W.; Fathurrahman, H.I.K.; Frisky, A.Z.K.; ul Haq, Q.M. Understanding of Convolutional Neural Network (CNN): A Review. *Int. J. Robot. Control Syst.* **2023**, *2*, 739–748. [CrossRef]
4. Younesi, A.; Ansari, M.; Fazli, M.; Ejlali, A.; Shafique, M.; Henkel, J. A Comprehensive Survey of Convolutions in Deep Learning: Applications, Challenges, and Future Trends. *IEEE Access* **2024**, *12*, 41180–41218. [CrossRef]
5. Ding, X.; Zhang, X.; Han, J.; Ding, G. Scaling up your kernels to 31 × 31: Revisiting large kernel design in cnns. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, New Orleans, LA, USA, 18–24 June 2022; pp. 11963–11975.
6. Liu, S.; Chen, T.; Chen, X.; Chen, X.; Xiao, Q.; Wu, B.; Kärkkäinen, T.; Pechenizkiy, M.; Mocanu, D.; Wang, Z. More convnets in the 2020s: Scaling up kernels beyond 51 × 51 using sparsity. *arXiv* **2022**, arXiv:2207.03620.
7. Chen, Y.; Liu, J.; Qi, X.; Zhang, X.; Sun, J.; Jia, J. Scaling up kernels in 3d cnns. *arXiv* **2022**, arXiv:2206.10555.
8. Khalfaoui-Hassani, I.; Pellegrini, T.; Masquelier, T. Dilated convolution with learnable spacings. *arXiv* **2021**, arXiv:2112.03740.
9. Sun, Z.; Ozay, M.; Okatani, T. Design of kernels in convolutional neural networks for image classification. In Proceedings of the Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, 11–14 October 2016; pp. 51–66.
10. He, K.; Li, C.; Yang, Y.; Huang, G.; Hopcroft, J.E. Integrating large circular kernels into cnns through neural architecture search. *arXiv* **2021**, arXiv:2107.02451.
11. Gao, H.; Zhu, X.; Lin, S.; Dai, J. Deformable kernels: Adapting effective receptive fields for object deformation. *arXiv* **2019**, arXiv:1910.02940.
12. Dai, J.; Qi, H.; Xiong, Y.; Li, Y.; Zhang, G.; Hu, H.; Wei, Y. Deformable convolutional networks. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 764–773.
13. Zhu, X.; Hu, H.; Lin, S.; Dai, J. Deformable convnets v2: More deformable, better results. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 9308–9316.
14. Wang, W.; Dai, J.; Chen, Z.; Huang, Z.; Li, Z.; Zhu, X.; Hu, X.; Lu, T.; Lu, L.; Li, H.; et al. Internimage: Exploring large-scale vision foundation models with deformable convolutions. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Vancouver, BC, Canada, 17–24 June 2023; pp. 14408–14419.
15. Xiong, Y.; Li, Z.; Chen, Y.; Wang, F.; Zhu, X.; Luo, J.; Wang, W.; Lu, T.; Li, H.; Qiao, Y.; et al. Efficient deformable convnets: Rethinking dynamic and sparse operator for vision applications. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 17–21 June 2024; pp. 5652–5661.
16. Chen, F.; Wu, F.; Xu, J.; Gao, G.; Ge, Q.; Jing, X.Y. Adaptive deformable convolutional network. *Neurocomputing* **2021**, *453*, 853–864. [CrossRef]
17. Lai, S.C.; Tan, H.K.; Lau, P.Y. 3D deformable convolution for action classification in videos. In Proceedings of the International Workshop on Advanced Imaging Technology (IWAIT), Online, 5–6 January 2021; Volume 11766, pp. 149–154.
18. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016. Available online: http://www.deeplearningbook.org (accessed on 20 July 2024).
19. Bishop, C.M.; Bishop, H. *Deep Learning: Foundations and Concepts*; Springer Nature: Cham, Switzerland, 2023.
20. Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; et al. An image is worth 16 × 16 words: Transformers for image recognition at scale. *arXiv* **2020**, arXiv:2010.11929.
21. Xia, Z.; Pan, X.; Song, S.; Li, L.E.; Huang, G. Vision transformer with deformable attention. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, New Orleans, LA, USA, 18–24 June 2022; pp. 4794–4803.
22. Chollet, F. Xception: Deep learning with depthwise separable convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 1251–1258.
23. Pominova, M.; Kondrateva, E.; Sharaev, M.; Pavlov, S.; Bernstein, A.V.; Burnaev, E. 3D Deformable Convolutions for MRI Classification. In Proceedings of the 18th IEEE International Conference on Machine Learning and Applications (ICMLA), Boca Raton, FL, USA, 16–19 December 2019; pp. 1710–1716.
24. Hoang, T. tvdcn. 2023. Available online: https://pypi.org/project/tvdcn/ (accessed on 10 March 2024).

25. Kwon, Y. PyTorch-Deformable-Convolution-v2. 2022. Available online: https://github.com/developer0hye/PyTorch-Deformable-Convolution-v2 (accessed on 25 March 2024).
26. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
27. Marsland, S.R. Machine Learning—An Algorithmic Perspective. In *Chapman and Hall/CRC Machine Learning and Pattern Recognition Series*; Routledge: London, UK, 2009.
28. Cat and Dog. Available online: https://www.kaggle.com/datasets/tongpython/cat-and-dog (accessed on 15 February 2024).
29. Kiefer, R. Glaucoma Dataset: EyePACS-AIROGS-light-V2. 2024. Available online: https://www.kaggle.com/datasets/deathtrooper/glaucoma-dataset-eyepacs-airogs-light-v2 (accessed on 22 February 2024).
30. Steen, J.; Kiefer, R.; Ardali, M.; Abid, M.; Amjadian, E. Standardized and Open-Access Glaucoma Dataset for Artificial Intelligence Applications. *Investig. Ophthalmol. Vis. Sci.* **2023**, *64*, 384.
31. Animals-10. Available online: https://www.kaggle.com/datasets/alessiocorrado99/animals10?rvi=1 (accessed on 20 February 2024).
32. Geometric Shapes Dataset. Available online: https://www.kaggle.com/datasets/dineshpiyasamara/geometric-shapes-dataset (accessed on 15 March 2024).
33. Soomro, K.; Zamir, A.R.; Shah, M. UCF101: A dataset of 101 human actions classes from videos in the wild. *arXiv* **2012**, arXiv:1212.0402.
34. Zagoruyko, S.; Komodakis, N. Wide residual networks. *arXiv* **2016**, arXiv:1605.07146.