







A Robust Convolutional Neural Network Model for Fruit Image Classification

Beman Hamidja Kamagate^{1*}, N'Diffon. Charlemagne Kopoin¹, Dagou Dangui Augustin Koffi¹,
Olivier Pascal Asseu²

¹ Department of Computer Science, Ecole Supérieure Africaine des TIC (ESATIC), Abidjan 18 BP 1501, Côte d'Ivoire

² Department of Electrical and Electronic Engineering, Institut National Polytechnique Félix Houphouët-Boigny (INPHB), Yamoussoukro BP 1093, Côte d'Ivoire

Corresponding Author Email: beman.kamagate@esatic.edu.ci

Copyright: ©2024 The authors. This article is published by IETA and is licensed under the CC BY 4.0 license (<http://creativecommons.org/licenses/by/4.0/>).

<https://doi.org/10.18280/isi.290504>

ABSTRACT

Received: 22 April 2024

Revised: 7 July 2024

Accepted: 1 August 2024

Available online: 24 October 2024

Keywords:

Convolutional Neural Network, deep learning, image recognition, KKDNet, smart agriculture

Modernization of agriculture is associated with increased use of energy and inputs (fertilizers, plant protection products, water), which today need to be better managed for optimization purpose and to limit risks for people and the environment. This issue has given rise to smart agriculture, a relative new approach to farming based on the integration of information and communication technologies, particularly Artificial Intelligent (AI) and Internet of Things, in which the environment (soil, relief, air, etc.) represents potential data source that should be exploited. In this work, we propose a service of smart agriculture that based on plant image recognition. A robust and simple Convolutional Neural Network (CNN) model called Kamagate, Kopoin and Dagou Neural Network (KKDNet) is developed for plant image classification. For our first experiment, we focused on tomato cultivation, which is very demanding in terms of energy and inputs. The proposed model KKDNet with less layer than other CNN architecture achieves approximately the same performance in metric like accuracy. As concerning the execution time, the other models use more execution time. we need to multiply KKDNet's execution time by a coefficient ranging from 25 to 52 to match the time required for the other architectures discussed in this study.

1. INTRODUCTION

Aiming to increase both quantity and quality, agriculture has undergone many technological advancements and changes over the years. However, this field still encounters significant obstacles in terms of quantities to cover global needs, as shown in the June 2017 United Nations report [1].

To effectively meet this need to sustain a growing and more selective society, the modernization of agriculture combined with increased use of energy and inputs is therefore necessary. However, such a practice can adversely affect the natural ecosystem. One way for mitigating the adverse impacts of productive yet intensive agriculture consists of using solutions offered by Artificial Intelligence (AI) [2] and the Internet of Things (IoT) [3], thereby implementing smart agriculture [3, 4].

Smart agriculture using AI and IoT techniques goes through various phases. These include the acquisition of environmental data from sensors, the recognition of acquired data for analysis, the data analysis phase and the decision-making phase. One of the foundations of smart agriculture is the use of computer vision and deep learning for pattern recognition and classification. Through these methods, relevant information such as plant species, presence of diseases, and presence of harmful species can be extracted [4]. In this context, image recognition can distinguish weeds from crops, thereby

promoting targeted application of herbicides and reducing chemical usage. Furthermore, fruit recognition allows for determining the optimal harvesting time, which can enhance yields and fruit quality. It also enables rapid identification of fruits affected by diseases or pests, facilitating timely corrective actions to minimize crop losses and damages. Additionally, in production and distribution chains, fruit recognition facilitates sorting and removal of defective or lower-quality products, ensuring that only the best fruits reach the market.

This work focuses on the recognition and analysis of captured data, specifically the classification of plant images. Convolutional Neural Networks are typically used for this task [5-8]. They are a form of Deep Learning (DL) model [9, 10] (see Figure 1). Originally presented by LeCun et al. [11] and enhanced by LeCun et al. [12], they design is based on the architecture of the biological neurons [11].

Indeed, LeCun et al. [12] created a Convolutional Neural Network known as LeNet-5, specifically designed for classifying handwritten digits. Its architecture and training principles are discussed in more detail by Hecht-Nielsen [13]. However, LeNet-5 is not equipped to handle more complex challenges, such as extensive image and video classification tasks, due to the constraints of training data and computational resources at that time. However, since 2006, numerous CNN methods have been developed that overcome these problems

[6, 7, 14, 15].

Krizhevsky proposed a classical CNN architecture called Alexnet [5] which showed good performance. Numerous studies such as ZFNet [16], VGGNet [17] and GoogleNet [14] have been suggested to increase its results.

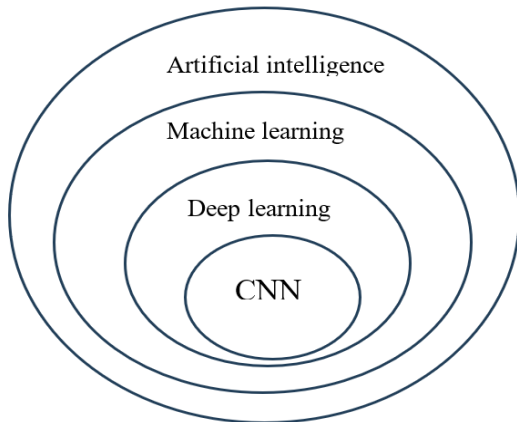


Figure 1. Subfields of artificial intelligence

Huang et al. [18] proposed DenseNet, in which every layer is directly connected to all preceding layers. In this architecture, each layer processes the attribute from all earlier layers as separate inputs, while its own attribute is given as inputs to all next layers. This scheme reaches optimal results on CIFAR-10/100 [19], whether or not data augmentation is applied. On the extensive dataset, DenseNet gets the same results to ResNet [20], when using less than half of the parameters. Traore et al. [21] proposed a CNN model for the classification of epidemic pathogens. The proposed CNN model comprises six convolution layers with the same architecture.

Guo et al. [6] introduced a straightforward Convolutional Neural Network. Authors (*They*) used the ReLU function [22] as the activation function and the dropout [23] as the regularization technique). The ReLU function leaves the value becomes 0 if the output value is less than 0; if not, it retains

the initial values. Thus, it can be used to force certain data to zero. Compared to some existing CNN models, such as those suggested by authors in references [24-26], the model proposed by Guo et al. [6] demonstrates inferior performance. However, the Guo et al. [6] model achieves relatively good recognition capabilities, benefits from a simple architecture, and requires minimal memory space for its parameters.

Although several CNN models have been proposed in the image classification context, architectures vary depending on the context of each study, which makes it difficult to establish a general architecture applicable to all domains. Additional issue is accessing extensive datasets such as ImageNet [5], encompassing approximately 14,000,000 images, and the MNIST database [27], which comprises 60,000 training images and 10,000 test images. This paper presents a robust and computationally efficient Convolutional Neural Network designed for the classification of tomato images.

2. CLASSIFICATION WITH CNN

2.1 CNN and visual data

Convolutional Neural Networks (CNNs) are highly effective for assignment that require the analysis of visual information, for example identifying images, detecting objects, and segmenting images. CNNs have the ability to learn hierarchical representations of visual data, allowing them to recognize intricate patterns and make precise predictions based on both low-level and high-level features.

2.2 Different layers of the CNN

A CNN typically comprises multiple layers (see Figure 2), every one serving a distinct function within the network (A CNN generally consists of several layers (Figure 2), each with a specific function in the network). According to some authors [28-30], several key types of layers in Convolutional Neural Networks should be. Table 1 shows these layers and their function in the network.

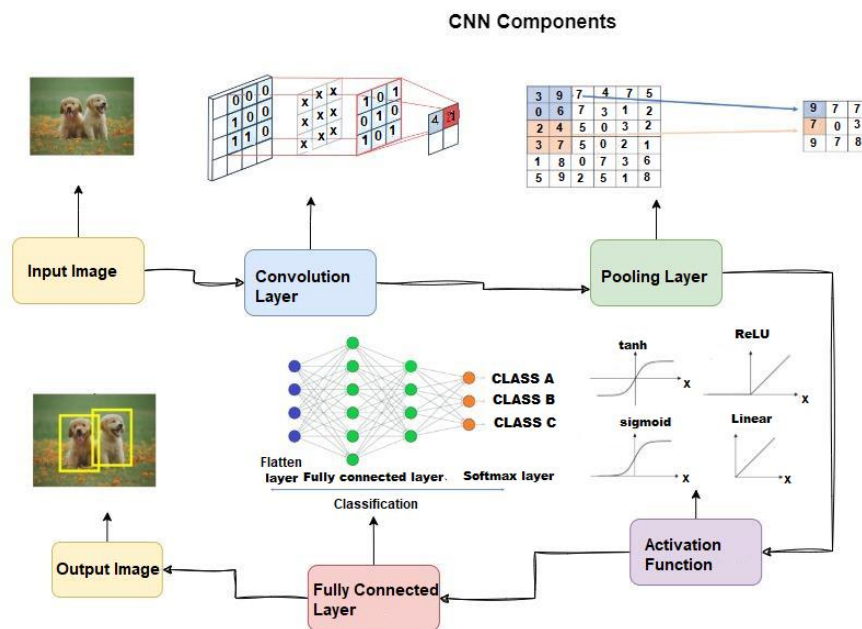


Figure 2. The different phases of the CNN [28]

Table 1. CNN layers (adapted from Krichen [29])

Layer	Function
Convolution layer	Generate feature maps by applying a convolution operation.
Pooling layer	Reduce the size of feature maps while retaining the most important information.
Activation layer	Introduce non-linearity in the model.
Full connected layer	Links each neuron in the preceding layer to every neuron in the current layer.

2.2.1 Convolutional layer

Considered as the heart of CNN [28, 29], convolution is a mathematical operation that combines two signals to generate a third signal, representing the effect of one signal on the other, modulated by the shape of the second signal. It can be calculated as follows:

$$(p \times q)[s] = \sum_{r=-\infty}^{\infty} p[r]q[s - r] \tag{1}$$

where, p and q are functions that can be either discrete or continuous, and s is the position of the output signal. For discrete signals, the equation above transforms to:

$$(p \times q)[s] = \sum_{r=-\infty}^{\infty} p[r]q[s - r]\Delta r \tag{2}$$

with Δr the sampling interval. For continuous one, the equation takes the form:

$$(p \times q)[u] = \int_{-\infty}^{\infty} p(\sigma)q(u - \sigma)d\sigma \tag{3}$$

Here, u is the position of the output signal.

Convolution is used to extract various features. The primary layer identifies basic characteristics such as contours, edges and angles. The upper layers, on the other hand, extract features more advanced (Figure 3). In the case presented in Figure 3, the input dimensions are: $N \times N \times D$. It is convolved with H filters, which size is $k \times k \times D$. Each operation on the input with a kernel produces a single output feature, and using H independent kernels results in H unique features.

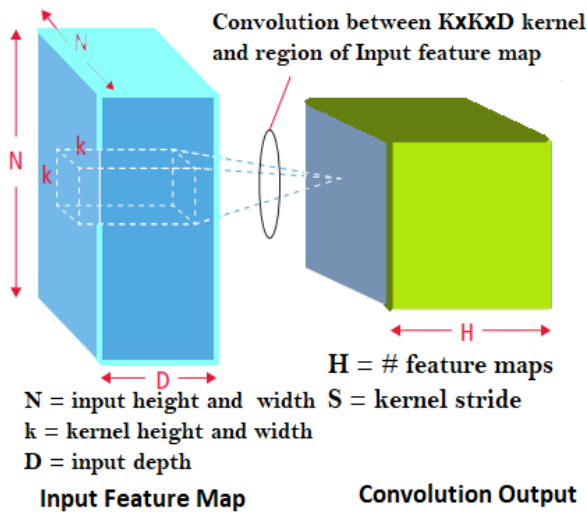


Figure 3. Graphical representation of the convolution process (adapted from Hijazi et al. [31])

Beginning at the top left corner of the input, each core shifts to the right, one element at a time. When it reaches the top right corner, the core moves down one element and then

resumes moving left to right, processing one element at a time. This procedure persists as far as the kernel reaches the bottom right corner. While N equal to 32 and k equal to 5, the kernel can occupy 28 distinct positions horizontally and 28 distinct positions vertically. As a result, each element of the output will contain 28×28 elements.

In a sliding window process, at each position of the core, the elements $k \times k \times D$ are multiplied, then accumulated element by element. Thus, to create a characteristic output element, $k \times k \times D$ elementary multiplications and accumulations are required.

2.2.2 Pooling layer

The pooling layer is designed to diminish the spatial dimensions of the feature maps generated by the convolutional layer [6, 28]. It helps to improve the robustness of feature extraction and reduces the likelihood of overfitting.

Typically placed in the middle of convolutional layers, the size of the feature maps in the pooling layer is influenced by the stride of the kernels. Commonly used pooling operations include average pooling, which mathematically corresponds to $f(x) = \frac{1}{n} \sum_{i=1}^n x_i$ and max pooling, which mathematically corresponds to $\max_i x_i$ [32]. Max pooling, one of the most frequently adopted methods, divides the image into rectangular areas, returning only the maximum value found in each area. A common maximum pooling size is 2×2 . An essential element of pooling, aimed at reducing the complexity of the upper layers, is subsampling, which can be likened to a reduction in resolution in image processing. The number of filters is not altered by pooling. As illustrated in Figure 4, using pooling on 2-by-2 blocks in the top left corner shifts focus to the top right corner, with a shift of two steps. Thus, a step of 2 is used for pooling. It is possible, but not very common, to use a step of 1 to avoid under sampling. Here, APG means the average of the window in green and maxG the maximum of the window in green.

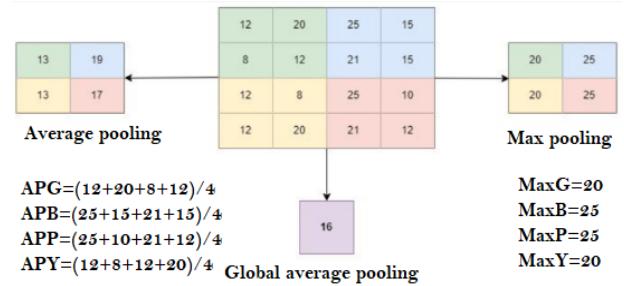


Figure 4. Pooling layers (adapted from Taye [28])

2.2.3 Activation layer

Activation functions are an essential element of CNNs, as they introduce nonlinearity [28, 33]. This non-linearity is essential as it allows the model to capture the intricate and non-linear patterns present in many real-world situations.

In the literature, activation functions frequently employed in CNNs are the rectified linear unit (ReLU), the sigmoid function, and the hyperbolic tangent function (tanh). The ReLU is particularly popular in contemporary CNNs because of its simplicity and its effectiveness in mitigating the vanishing gradient issue during training. Mathematically, the ReLU function is defined as follows:

$$ReLU(x) = \max(0, x) \tag{4}$$

x represents neuron's input. This function calculates the maximum between 0 and x , which neutralizes negative values while preserving positive values unchanged. ReLU helps Convolutional Neural Networks (CNNs) learn complex features, and helps avoid neuron saturation during learning. Given its simplicity, several derivatives of the ReLU activation function have been proposed to improve it. Examples include ELU, LeakyReLU, SELU and ReLU6 [30, 33]. This function is often used in neural networks for specific tasks where activation is required but where it is also desirable to limit the output range, according to the study by Jiang et al. [33].

2.2.4 Fully connected layer

The fully connected layer is a type of layer where each neuron is connected to every neuron in the previous layer [34, 35]. In neural network architectures, they are typically employed in the last stages to generate the ultimate output. Here are some of the main roles of the fully connected layer, it is in charge of integrating features extracted from input data by previous layers. Fully connected layers can also learn non-linear combinations of input features, allowing them to capture intricate relationships between different variables. Once features have been appropriately combined, the fully connected layer produces outputs that are used for classification (classification tasks) or for the prediction of continuous values (regression tasks).

3. PROPOSED CNN MODEL

When observing the development of a CNN in depth, evidence suggests that the capacity for expression increases with the network's depth. However, as the network becomes deeper, its memory consumption increases and there is a risk that its performance will not improve [6, 18].

With this in mind, we proposed a novel architecture of CNN called *Kamagate, Kopoin and Dagou Neural Network* (KKDNet), which is intended to be robust and does not rely on layer depth. Unlike many Convolutional Neural Networks (CNNs), which place great emphasis on network depth, our approach with KKDNet focuses more on the quality of extracted features and the stability of the model in different situations. The choice not to base our model on layer depth stems from several considerations. Firstly, a shallower architecture can reduce the model's complexity and hence its sensitivity to overlearning, while enabling faster convergence during training. Furthermore, by focusing on features that are relevant and meaningful to the given task, our approach aims to promote efficient generalization, even using a limited network depth.

In developing KKDNet, we focused on building an architecture that leverages the power of Convolutional Neural Networks while avoiding the pitfalls associated with excessive depth. By judiciously adjusting hyperparameters and using appropriate regularization techniques, we sought to create a model capable of competing with deeper architectures while retaining high efficiency and generalizability.

3.1 Architecture of KKDNet

3.1.1 Convolution part

KKDNet's architecture is a classic CNN with 4 convolution layers and 4 pooling layers (see Figure 5). We experimented

with different architectures and selected the one with 4 convolutional layers (see Table 2), because starting from this number of layers, the accuracy reaches its maximum value. Therefore, there is no need to go beyond this number of layers. Moreover, adding additional layers does not improve performance but could rather increase execution time. The same reasoning led us to choose the 3×3 kernel. We utilized a pooling layer with 2×2 filters and a stride of 2. The goal is to reduce the input to a quarter of its initial size. This allows the image to be scaled by a factor of $\frac{1}{2}$, which means that a window of 2×2 is used and the height of the output corresponds to half the height of the input.

The first layer is a convolutional layer that uses 16 filters of size. Each filter here corresponds to a matrix of 3 pixels by 3 pixels. In this layer, we applied max pooling with a 2×2 filter and a stride of 2, ensuring that the pooled regions do not intersect. The second layer applies 32 sizes 3×3 filters, producing 32 activation maps. We applied the same type of maximum grouping to this layer as to the first layer, shape 2×2 and stride 2. The third one applies 64 filters of 3×3 , followed by another layer of maximum shape pooling 2×2 and stride. The four max pooling layers decrease the dimensions of the display by a factor of 32. This layer feeds two other fully connected layers with 1024 and 512 inputs respectively. Using 3×3 filters allow the model to capture local patterns and details in the input image. Fundamental features like edges and textures are efficiently extracted.

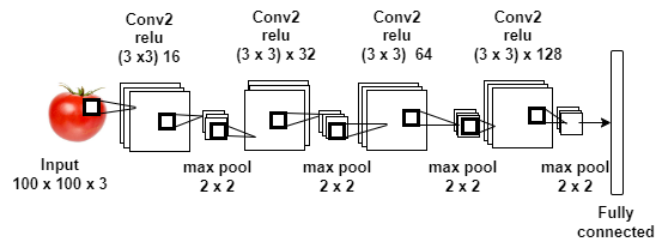


Figure 5. Convolution part of KKDNet

3.1.2 Fully connected architecture

In a fully connected layer, each neuron is connected to all the outputs of the previous layer. Note that the operations carried out in a fully connected layer are analogous to those in a convolutional layer, making it possible to transition between the two. Loss layers are employed to penalize the network when it produces an output different from that expected, and are usually located at the last stage of the network. Different loss functions exist, for example the SoftMax function, which is used to predict one class from a set of disjoint classes. In this study, we have also concentrated on the fully connected portion [36], which makes the network more robust. For greater efficiency in discriminating features extracted in the convolutional phases, the fully connected part is learned under two distinct sub-networks (Figure 6).

The first subnetwork has 1024 neurons as input, followed by 512, 256 and 128 neurons respectively in the next three layers. The second subnetwork has 512 neurons as input, followed by 128 neurons in each of the three successive layers. Both subnetworks have an identical number of fully connected layers and *They outputs are concatenated in the fourth layer to extract the most discriminating features learned.* This distribution makes it possible to search for high-dimensionality features across two different architectures. Each fully connected layer performs complex calculations to

transform inputs and produce meaningful outputs, contributing to its ability to learn and generalize from training data. The final layer is a SoftMax loss layer with 128 inputs.

We should point out that several architectures were tested in this work. For example, architectures with more than 4 layers gave slightly better performance results than the one we adopted. However, the execution time was a little longer for no great difference.

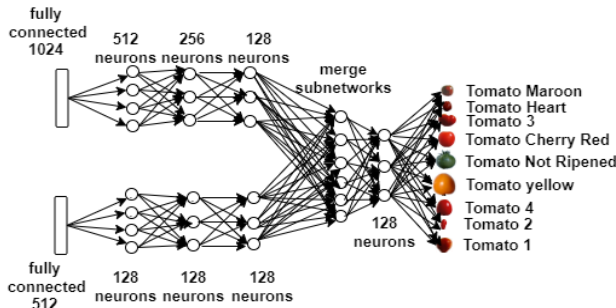


Figure 6. Fully connected part of KKDNNet

Table 2. Different CNN architectures

Name	Interval	Recommendation
Convolution layers	3, 4, 5, 6, 7, 8	4
kernel	(3 × 3), (5 × 5), (7 × 7)	(3 × 3)
Stride	1 × 1, 2 × 2	2 × 2
Activation function	ReLU, Sigmoid, Softmax, Tanh	ReLU

3.2 Network optimization

Optimizing convolutional networks is one of the most important tasks in setting up such a system. The weights of the Convolutional Neural Network must be optimized with the gradient descent algorithm. Throughout this iterative process, it is crucial to fine-tune key parameters, such as the base learning rate. To achieve better validation accuracy in a reduced number of epochs, we used the cyclic learning rate method [37], which involves varying the learning rate within a range of values for a specific number of iterations. In our case, the Adadelta optimizer with a learning rate whose values are between $1 \cdot e^{-1}$ and $1 \cdot e^{-6}$ was applied. The factor $\rho = 0.5$ was used to reduce the learning rate.

Table 3. Core KKDNNet network parameters and recommended settings

Name	Interval	Recommendation
Epoch	16, 25, 32, 64, 50	25
Batch-size	25, 50, 75, 100	50
Per-parameter adaptive learning rate methods	SGD, RMSprop, Adagrad, Adadelta, Adam, Adamax, Nadam	Adadelta
Dropout rate	0.1, 0.2, 0.3, 0.4	0.1

As far as the other hyperparameters are concerned, it has to be said that the selection of the best hyperparameters depends on the data and the application, requiring the formation of models with different configurations and the evaluation of their performance on a validation set. It is almost impossible to try them all, given the exponential number of configurations

and hyperparameters. Some authors [36] instead recommend optimizing hyperparameters, such as learning rate and batch size, by experimenting with various values while keeping the other hyperparameters fixed. Next, the specified hyperparameter space can be explored in more detail through random sampling, and the parameters yielding the best results on the validation set can be selected. Based on this principle, we have listed the other hyperparameter values used in the network in Table 3.

4. EXPERIMENTATION AND RESULTS

4.1 Dataset

The image dataset we utilized is 'Fruits-360', derived from the work by Muresan and Oltean [38]. 'Fruits-360' is a comprehensive dataset of fruit images that has been employed in various studies to assess the proposed models [39-41]. The images were captured by filming the fruit as it was rotated by a motor. The fruits were positioned along the motor's axis at a low speed (3 rpm), and a brief 20-second video was recorded.

To obtain the background images, a diffusion fill algorithm was employed. Initially, all pixels along the edges of the image are marked, as well as any pixels in close proximity to these marked pixels, provided the color distance is below a specified threshold. This process is repeated until no additional pixels can be marked. The marked pixels are identified as the background, while the remaining pixels are classified as part of the object. The maximum allowable distance between neighboring pixels is a parameter of the algorithm, adjusted through trial and error for each video. The fruits were resized to fit a 100×100 pixel image. This dataset includes 90,483 images of fruits and vegetables, categorized into 131 different classes. In this study, we focused primarily on tomato images, which are classified into 9 types: Tomato 1, Tomato 2, Tomato 3, Tomato 4, Cherry Red Tomato, Heart Tomato, Brown Tomato, Unripe Tomato, and Yellow Tomato (see Figure 7). Note that the tomato whose scientific or popular name has not been found is labelled with numbers: tomato 1, tomato 2, tomato 3, tomato 4.

The total number of tomato images (data set 1) is 6810, of which 5103 used for training consist of sets of each tomato type, and 1707 used for testing. We also observed that in each training set, one-fifth of the images for each tomato type were allocated for validation, while the remaining four-fifths were used for training.

To address the issue of small data sizes, the image data augmentation technique known as 'Augmentor' [42] was employed. This involved using a pipeline-based approach to stochastic image augmentation. This method allows the user to chain operations such as shears, rotations and crops, and pass images through this pipeline to create new data. All pipeline operations are applied randomly, both in terms of the probability of operations being applied to each image as it passes through the pipeline, and in terms of the parameters of each operation, which are also chosen randomly from user-specified ranges. This makes it possible to sample from a distribution of possible images, generated by the pipeline at runtime. To create data set 2, comprising 10,000 images, we used the 'Augmentor' tool package available in Python. The main parameters selected include a random rotation of images between -5 degrees and +5 degrees, horizontal flipping with a probability of 50%, and random zooming with a 50%

probability, enlarging or reducing the image within an area representing 80% of its original size. For vertical flipping, a probability of 50% has been set.

Elastic distortions allow you to distort an image while maintaining its proportions. Table 4 shows the number of training images from data sets 1 and 2, as well as the test images for each type of tomato.

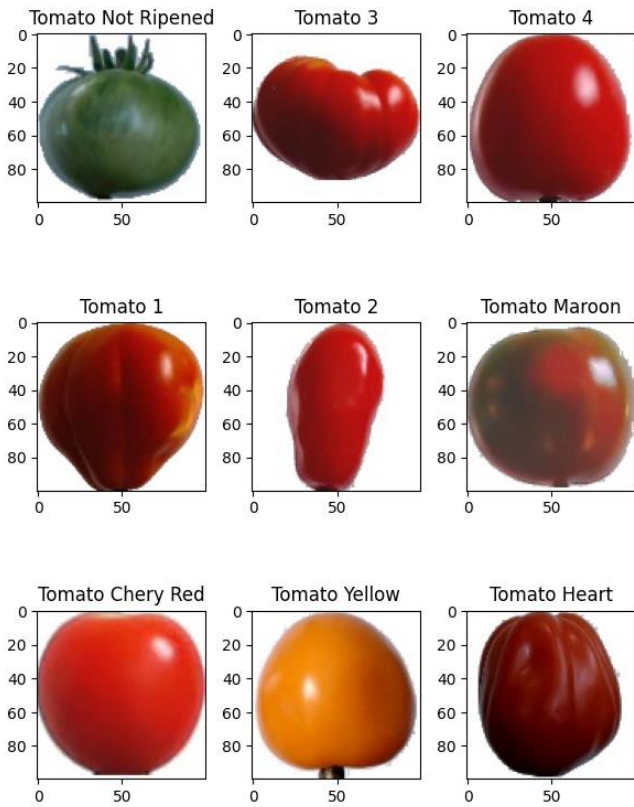


Figure 7. Image of each tomato by category

Table 4. Number of images of each tomato for each training set

Tomato Type	Training Set 1	Training Set 2	Testing Set
Tomato 1	738	1440	246
Tomato 2	672	1316	225
Tomato 3	738	1419	246
Tomato 4	479	938	160
Tomato Cherry Red	492	989	164
Tomato Maroon	367	788	127
Tomato Heart	684	1325	228
Tomato Not Ripened	474	895	158
Tomato Yellow	459	890	153
Total	5103	10000	1707

4.2 Results for the “fruits-360” dataset

4.2.1 Results on tomatoes dataset

Here are experimental results carried out on a core i7 computer and 16GB RAM with python packages like Scikit-learn, Keras. Experimental results are relative to our proposed architecture KKDNet and other CNN architecture like GoogLeNet, ShuffleNet, ResNet and DenseNet. The main metrics performance we look for are accuracy both in training phase and test phase, we also look for the execution times.

Figure 8 shows the validation and training loss curves for our model. We trained our model with 8.1 million parameters

for 25 epochs and a batch size of 50 images. As illustrated in Figure 8, our model quickly (before epoch 5) achieves higher validation accuracy, with much lower validation loss, thanks to the cyclic learning rate we applied. Before epoch 12, we could say that we were under learning, since the validation loss curve was below the training loss curve. From this point onwards, a higher cyclic learning rate was applied, reducing the gap between the two curves, which subsequently tend to decrease together towards 0. At the end of the 25 epochs, we have obtained a validation accuracy of 100%, while our learning accuracy is also equal to 100%, reflecting an effective level of learning and generalization of our model.

In the following, we evaluate the achievement of our KKDNet model against several models from the literature, such as GoogLeNet [43], ShuffleNet [44], ResNet [20], and DenseNet-121 [39], which we have implemented.

It should be noted that all these networks have been trained with the hyperparameter values that give the best performance. GoogLeNet, consisting of 22 convolution layers and it is based on the inception architecture [45]. It employs inception modules, allowing the network to choose from various sizes of convolutional filters within each block. An inception network stacks these modules sequentially, occasionally incorporating maximum pooling layers with a stride of 2 to halve the grid resolution. ShuffleNet is particularly designed for mobile devices with very limited processing capabilities. Its architecture uses two process point-group convolution and channel shuffling, to significantly decrease computing costs while maintaining accuracy.

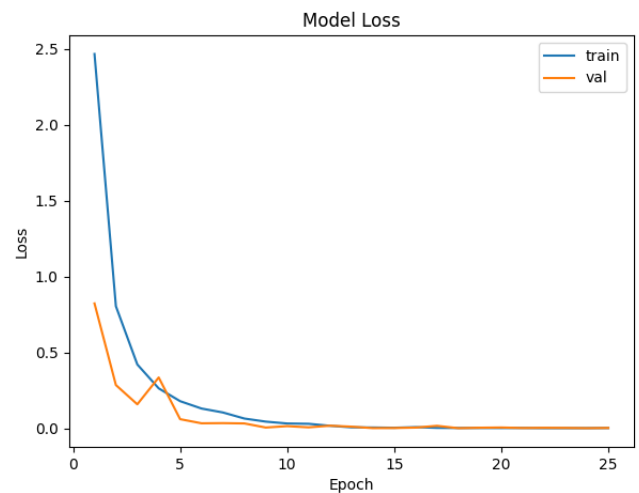


Figure 8. KKDNet training and validation loss curves

Regarding the calculation of complexity, it should be noted that pooling layers and activation functions generally have negligible complexity and are therefore not taken into account. The algorithmic complexity here is primarily determined by the number of operations required to perform the convolutions, which is the most computationally expensive part [46]. Let C denote the complexity, it can be calculated as follows:

$$C = \left(\sum_{l=1}^L K_l \times K_l \times I_l \times H_l \times W_l \times S_l \right) + P \quad (5)$$

where, L represents the total number of convolutional layers in the network, K denotes the filter size, I stand for the number of input channels, H indicates the height of the feature map, W

signifies the width of the feature map, S refers to the number of output channels, and P represents the number of model parameters. The computation of algorithmic complexity can be expressed in FLOPs (Floating Point Operations), which is the total number of floating-point operations needed to execute a model. Table 5 presents the parameters and algorithmic complexities of the various models.

We also compare our model with KKDNet-one, which retains the same architecture as KKDNet but has a single fully connected layer. The KKDNet-one architecture thus consists of 4 convolutional layers with filters 3×3 and a shape 2×2 stride. For the fully connected part, we have a first layer of 1024 neurons, followed by 512, 256 and 128 neurons in the next three layers, the last layer being the output layer with 9 neurons.

The training performance results obtained from training set 1, composed of tomato images, on different models are presented in Table 6. Table 7 presents similar results obtained on training set 2. In addition, we have included in these tables a column relating to computational complexity, corresponding to the results in Table 5.

What we can say about the results shown in Table 6 and Table 7 is that for all models overall, the training performance values in the accuracy metric are very close to the test ones, and in addition they are above 94%. This shows that all models are able to extract and learn features efficiently. We have the Resnet, DenseNet and our KKDNet models, which show 100% performance in the accuracy metric on both the validation and test sides. This shows that all three models have learned the features well and are able to generalize to new data accurately. Regarding the accuracy metric (Table 6), the

GoogLeNet model shows a validation performance of 98.61% and a test performance of 99.05%. The ShuffleNet model has a training performance of 95.07% and a test performance of 95.80%. As for KDDNet-one, performance was 95.90% in training and 96.09% in test. With the data set 2.

We observe on both two table (Table 6 and Table 7) that KKDNet achieves similar or even better performance than the GoogLeNet, ShuffleNet, and ResNet architectures, both in training and testing. The advantage is that these performances are achieved with a significantly lower computation time compared to the other architectures. Indeed, computation time is closely related to complexity, evaluated in FLOPS. For example, it takes GoogleNet 25 times longer than KKDNet to achieve the same results. ShuffleNet takes 51 times longer than KKDNet.

4.2.2 Others result

We conducted further experiments using the remaining images from the "Fruits-360" database, excluding the tomato images, to evaluate KKDNet's generalization capability. The "Fruits-360" dataset comprises around 80 types of fruits across 131 classes, including avocados, bananas, pineapples, and mangoes. By excluding the tomato classes, we retained 83,643 images across 122 classes. KKDNet achieved an accuracy of 98.20%, compared to 97.50% for GoogleNet, 96.53% for ShuffleNet, 99.58% for ResNet, and 99.10% for DenseNet. Notably, KKDNet demonstrates strong generalization with an accuracy above 94% and outperforms GoogleNet, ShuffleNet, and DenseNet in this aspect, though ResNet achieves slightly higher accuracy. Additionally, KKDNet offers a superior execution time advantage (Table 5).

Table 5. Parameters number and computational complexity of the models

Models	Parameters Number	Computational Complexity (FLOPs)
GoogleNet	6,056,505	1,221,195,310
ShuffleNet	968,337	2,450,507,844
ResNet	23,606,153	6,347,705,993
DenseNet	7,052,617	3,212,015,689
KKDNet -one	5,507,385	44,924,217
KKDNet	8,064,441	47,704,745

Table 6. Performance comparison on data set 1

Models	Computational Complexity (FLOPs)	Training Accuracy (%)	Test Accuracy (%)
GoogleNet	1,221,195,310	98.61	99.05
ShuffleNet	2,450,507,844	95.07	95.78
ResNet	6,347,705,993	100	100
DenseNet	3,212,015,689	100	100
KKDNet -one	44,924,217	95.80	96.09
KKDNet	47,704,745	100	100

Table 7. Performance comparison on data set 2

Models	Computational Complexity (FLOPs)	Training Accuracy (%)	Test Accuracy (%)
GoogleNet	1,221,195,310	99.79	100
ShuffleNet	2,450,507,844	98.69	99.88
ResNet	6,347,705,993	99.89	100
DenseNet	3,212,015,689	100	100
KKDNet -one	44,924,217	98.92	100
KKDNet	47,704,745	100	100

5. DISCUSSION

This study intends to present a CNN network model that

offers good performance without being computationally intensive or having many layers. Our proposed CNN network, named KKDNet, features just 4 convolutional layers, with a

kernel size of 3×3 , a stride of 2×2 , and utilizes the ReLU activation function, as outlined in its architecture in Table 2. In addition, with the use of cyclic learning rate in training phase achieve approximately same performance in training and test accuracy as shown in Table 6 and Table 7. We believe that using a cyclic learning rate, which periodically increases the learning rate, helps explore a larger portion of the parameter space and avoid local minima, thereby facilitating the discovery of better global solutions more quickly.

Concerning the execution time based on the number of operations in FLOPS, as presented in Table 5, we observed that we need to multiply its execution time by a coefficient ranging from 25 to 52 to match the time required for the other architectures discussed in this study. This efficiency is primarily due to the use of 4 layers and the 3×3 kernel size, which require less computation time and maintain similar performance like other mentioned CNN architectures.

However, we note that for other classes in the Fruits 360 database, our model (KKDNet), as well as the GoogLeNet, DenseNet, and ResNet models, exhibit error rates between 1% and 2%. Indeed, these errors could result from the variety of classes and the fact that some classes are less represented in the database. Regularization techniques such as L1/L2 could address this issue and help the model generalize better. We could also add the batch normalization technique, which, although not strictly a regularization technique, can help achieve a more generalizable model.

6. CONCLUSION

In this research, we introduced the KKDNet model, an efficient CNN network model, while being enhanced for computational effectiveness. Experiments were performed on tomato image datasets from the "fruits 360" database. The results obtained showed that our KKDnet model offers comparable performance to recent models in the literature, while requiring less computation time to run. It should be noted that we applied techniques such as the cyclic learning rate, which not only accelerated learning but also made the model more efficient. Ultimately, the results showed that our KKDNet model with fewer layers achieved superior performance in the identification and classification of tomato images, and could therefore serve as a general system for smart categorization of plant images in practical use. We presume that our model because of using cyclic learning rate will perform well in generalization.

In the future, to make our model even more robust, we plan to test it on other types of image datasets such as MNIST dataset. We also intend to employ regularization techniques like L1/L2 regularization and batch normalization to enhance the model's generalizability. Additionally, we plan to integrate the model into a software application for use in agriculture.

REFERENCES

- [1] Nations, U. World population projected to reach 9.8 billion in 2050, and 11.2 billion in 2100. United Nations Department of Economic and Social Affairs. <https://www.un.org/development/desa/en/news/population/world-population-prospects-2017.html>, accessed on Jan. 5, 2024.
- [2] Rothman, D. (2018). Artificial intelligence by example: Develop machine intelligence from scratch using real artificial intelligence use cases. Packt Publishing Ltd. <https://www.pdfdrive.com/artificial-intelligence-by-example-develop-machine-intelligence-from-scratch-using-real-artificial-intelligence-use-cases-e176504308.html>, accessed on Oct. 30, 2019.
- [3] Ayaz, M., Ammad-Uddin, M., Sharif, Z., Mansour, A., Aggoune, E.H.M. (2019). Internet-of-Things (IoT)-based smart agriculture: Toward making the fields talk. *IEEE Access*, 7: 129551-129583. <https://doi.org/10.1109/ACCESS.2019.2932609>
- [4] Issad, H.A., Aoudjit, R., Rodrigues, J.J. (2019). A comprehensive review of Data Mining techniques in smart agriculture. *Engineering in Agriculture, Environment and Food*, 12(4): 511-525. <https://doi.org/10.1016/j.eaef.2019.11.003>
- [5] Krizhevsky, A., Sutskever, I., Hinton, G.E. (2012). Imagenet classification with deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, 25. <https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html>, accessed on Jan. 4, 2024.
- [6] Guo, T., Dong, J., Li, H., Gao, Y. (2017). Simple Convolutional Neural Network on image classification. In 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA), Beijing, China, pp. 721-724. <https://doi.org/10.1109/ICBDA.2017.8078730>
- [7] Sharma, N., Jain, V., Mishra, A. (2018). An analysis of Convolutional Neural Networks for image classification. *Procedia Computer Science*, 132: 377-384. <https://doi.org/10.1016/j.procs.2018.05.198>
- [8] LeCun, Y., Bengio, Y., Hinton, G. (2015). Deep learning. *Nature*, 521(7553): 436-444. <https://doi.org/10.1038/nature14539>
- [9] Cao, C., Liu, F., Tan, H., Song, D., Shu, W., Li, W., Zhou, Y., Bo, X., Xie, Z. (2018). Deep learning and its applications in biomedicine. *Genomics, Proteomics and Bioinformatics*, 16(1): 17-32. <https://doi.org/10.1016/j.gpb.2017.07.003>
- [10] Müller, V.C., Bostrom, N. (2016). Future progress in artificial intelligence: A survey of expert opinion. *Fundamental Issues of Artificial Intelligence*, Springer, Cham, 555-572. https://doi.org/10.1007/978-3-319-26485-1_33
- [11] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278-2324. <https://doi.org/10.1109/5.726791>
- [12] LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., Jackel, L. (1989). Handwritten digit recognition with a back-propagation network. *Advances in Neural Information Processing Systems*, 2: 396-404.
- [13] Hecht-Nielsen, R. (1992). III.3-Theory of the backpropagation neural network. *Neural Networks for Perception*, 1992: 65-93. <https://doi.org/10.1016/B978-0-12-741252-8.50010-8>
- [14] Lee, H., Kwon, H. (2017). Going deeper with contextual CNN for hyperspectral image classification. *IEEE Transactions on Image Processing*, 26(10): 4843-4855. <https://doi.org/10.1109/TIP.2017.2725580>
- [15] Chen, G., Chen, Q., Long, S., Zhu, W., Yuan, Z., Wu, Y. (2023). Quantum Convolutional Neural Network for image classification. *Pattern Analysis and Applications*,

- 26(2): 655-667. <https://doi.org/10.1007/s10044-022-01113-z>
- [16] Zeiler, M.D., Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Computer Vision-ECCV 2014: 13th European Conference, Zurich, Switzerland, Proceedings, Part I*. Springer International Publishing. Springer, Cham, 13: 818-833. https://doi.org/10.1007/978-3-319-10590-1_53
- [17] Simonyan, K. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv Preprint arXiv: 1409.1556*. <https://doi.org/10.48550/arXiv.1409.1556>
- [18] Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q. (2017). Densely connected convolutional networks. In *Proceedings 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA*, pp. 4700-4708. <https://doi.org/10.1109/CVPR.2017.243>
- [19] Hu, K., Zou, A., Wang, Z., Leino, K., Fredrikson, M. (2024). Unlocking deterministic robustness certification on imagenet. *Advances in Neural Information Processing Systems*, 36. https://proceedings.neurips.cc/paper_files/paper/2023/hash/863da9d40547f1d1b18859519ce2dee4-Abstract-Conference.html
- [20] Targ, S., Almeida, D., Lyman, K. (2016). Resnet in resnet: Generalizing residual architectures. *arXiv Preprint arXiv: 1603.08029*. <https://doi.org/10.48550/arXiv.1603.08029>
- [21] Traore, B.B., Kamsu-Foguem, B., Tangara, F. (2018). Deep convolution neural network for image recognition. *Ecological Informatics*, 48: 257-268. <https://doi.org/10.1016/j.ecoinf.2018.10.002>
- [22] Nair, V., Hinton, G.E. (2010). Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), Haifa, Israel*, pp. 807-814.
- [23] Wang, S., Manning, C. (2013). Fast dropout training. In *International Conference on Machine Learning, PMLR*, 28(2): 118-126. <https://proceedings.mlr.press/v28/wang13a.html>, accessed on Jan. 5, 2024.
- [24] Wan, L., Zeiler, M., Zhang, S., Le Cun, Y., Fergus, R. (2013). Regularization of neural networks using dropconnect. In *International Conference on Machine Learning. PMLR*, pp. 1058-1066. <https://proceedings.mlr.press/v28/wan13.html>, accessed on Jan. 6, 2024.
- [25] Lee, C.Y., Gallagher, P.W., Tu, Z. (2016). Generalizing pooling functions in Convolutional Neural Networks: Mixed, gated, and tree. In *Artificial Intelligence and Statistics. PMLR*, pp. 464-472. <https://proceedings.mlr.press/v51/lee16a.html>, accessed on Jan. 6, 2024.
- [26] Lee, C.Y., Xie, S., Gallagher, P., Zhang, Z., Tu, Z. (2015). Deeply-supervised nets. In *Artificial Intelligence and Statistics. PMLR*, pp. 562-570. <https://proceedings.mlr.press/v38/lee15a.html>, accessed on Jan. 06, 2024.
- [27] LeCun, Y., Kavukcuoglu, K., Faret, C. (2010). Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems, Paris, France*, pp. 253-256. <https://doi.org/10.1109/ISCAS.2010.5537907>
- [28] Taye, M.M. (2023). Theoretical understanding of Convolutional Neural Network: Concepts, architectures, applications, future directions. *Computation*, 11(3): 52. <https://doi.org/10.3390/computation11030052>
- [29] Krichen, M. (2023). Convolutional Neural Networks: A survey. *Computers*, 12(8): 151. <https://doi.org/10.3390/computers12080151>
- [30] Li, Z., Liu, F., Yang, W., Peng, S., Zhou, J. (2021). A survey of Convolutional Neural Networks: Analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 33(12): 6999-7019. <https://doi.org/10.1109/TNNLS.2021.3084827>
- [31] Hijazi, S., Kumar, R., Rowen, C. (2015). Using Convolutional Neural Networks for image recognition. *Cadence Design Systems Inc.: San Jose, CA, USA*, 9(1).
- [32] Bieder, F., Sandkühler, R., Cattin, P.C. (2021). Comparison of methods generalizing max-and average-pooling. *arXiv Preprint arXiv: 2103.01746*. <https://doi.org/10.48550/arXiv.2103.01746>
- [33] Jiang, Y., Xie, J., Zhang, D. (2022). An adaptive offset activation function for CNN image classification tasks. *Electronics*, 11(22): 3799. <https://doi.org/10.3390/electronics11223799>
- [34] Durairajah, V., Gobe, S., Muneer, A. (2018). Automatic vision based classification system using DNN and SVM classifiers. In *2018 3rd International Conference on Control, Robotics and Cybernetics (CRC), Penang, Malaysia*, pp. 6-14. <https://doi.org/10.1109/CRC.2018.00011>
- [35] Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., Alsaadi, F.E. (2017). A survey of deep neural network architectures and their applications. *Neurocomputing*, 234: 11-26. <https://doi.org/10.1016/j.neucom.2016.12.038>
- [36] Du, X., Sun, S., Hu, C., Yao, Y., Yan, Y., Zhang, Y. (2017). DeepPPI: boosting prediction of protein-protein interactions with deep neural networks. *Journal of Chemical Information and Modeling*, 57(6): 1499-1510. <https://doi.org/10.1021/acs.jcim.7b00028>
- [37] Yu, T., Zhu, H. (2020). Hyper-parameter optimization: A review of algorithms and applications. *arXiv Preprint arXiv: 2003.05689*. <https://doi.org/10.48550/arXiv.2003.05689>
- [38] Muresan, H., Oltean, M. (2018). Fruit recognition from images using deep learning. *Acta Universitatis Sapientiae, Informatica*, 10(1): 26-42. <https://doi.org/10.2478/ausi-2018-0002>
- [39] Lu, T., Han, B., Chen, L., Yu, F., Xue, C. (2021). A generic intelligent tomato classification system for practical applications using DenseNet-201 with transfer learning. *Scientific Reports*, 11(1): 15824. <https://doi.org/10.1038/s41598-021-95218-w>
- [40] Siddiqi, R. (2019). Effectiveness of transfer learning and fine tuning in automated fruit image classification. In *Proceedings of the 2019 3rd International Conference on Deep Learning Technologies. In ICDLT '19*. New York, NY, USA: Association for Computing Machinery, Xiamen, China, pp. 91-100. <https://doi.org/10.1145/3342999.3343002>
- [41] Kodors, S., Laci, G., Zhukov, V., Bartulsons, T. (2020). Pear and apple recognition using deep learning and mobile. *Engineering For Rural Development*, 20: 1795-1800. <https://doi.org/10.22616/ERDev.2020.19.TF476>
- [42] Bloice, M.D., Stocker, C., Holzinger, A. (2017). Augmentor: An image augmentation library for machine

- learning. arXiv Preprint arXiv: 1708.04680. <https://doi.org/10.48550/arXiv.1708.04680>
- [43] Anand, R., Shanthi, T., Nithish, M.S., Lakshman, S. (2020). Face recognition and classification using GoogleNET architecture. In *Soft Computing for Problem Solving: SocProS 2018*, Springer Singapore, 1: 261-269. https://doi.org/10.1007/978-981-15-0035-0_20
- [44] Zhang, X., Zhou, X., Lin, M., Sun, J. (2018). ShuffleNet: An extremely efficient Convolutional Neural Network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6848-6856. <https://doi.org/10.1109/CVPR.2018.00716>
- [45] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1-9. https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Szegedy_Going_Deeper_With_2015_CVPR_paper.html, accessed on Mar. 18, 2024.
- [46] Dumoulin, V., Visin, F. (2016). A guide to convolution arithmetic for deep learning. arXiv Preprint arXiv: 1603.07285. <https://doi.org/10.48550/arXiv.1603.07285>