applied sciences

MDPI

Article

Robust Approach to Supervised Deep Neural Network Training for Real-Time Object Classification in Cluttered Indoor Environment

Bedada Endale [1], Abera Tullu [2], Hayoung Shi [1] and Beom-Soo Kang [1,*]

[1] Department of Aerospace Engineering, Pusan National University, Busan 46241, Korea; endale@pusan.ac.kr (B.E.); shy621@pusan.ac.kr (H.S.)
[2] Department of Aerospace Engineering, Sejong University, 209, Neungdong-ro, Gwangjin-gu, Seoul 05006, Korea; tuab@sejong.ac.kr
* Correspondence: bskang@pusan.ac.kr; Tel.: +82-51-510-2310

Abstract: Unmanned aerial vehicles (UAVs) are being widely utilized for various missions: in both civilian and military sectors. Many of these missions demand UAVs to acquire artificial intelligence about the environments they are navigating in. This perception can be realized by training a computing machine to classify objects in the environment. One of the well known machine training approaches is supervised deep learning, which enables a machine to classify objects. However, supervised deep learning comes with huge sacrifice in terms of time and computational resources. Collecting big input data, pre-training processes, such as labeling training data, and the need for a high performance computer for training are some of the challenges that supervised deep learning poses. To address these setbacks, this study proposes mission specific input data augmentation techniques and the design of light-weight deep neural network architecture that is capable of real-time object classification. Semi-direct visual odometry (SVO) data of augmented images are used to train the network for object classification. Ten classes of 10,000 different images in each class were used as input data where 80% were for training the network and the remaining 20% were used for network validation. For the optimization of the designed deep neural network, a sequential gradient descent algorithm was implemented. This algorithm has the advantage of handling redundancy in the data more efficiently than other algorithms.

Keywords: object classification; deep learning; convolutional neural network; network architecture

Citation: Endale, B.; Tullu, A.; Shi, H.; Kang, B.-S. Robust Approach to Supervised Deep Neural Network Training for Real-Time Object Classification in Cluttered Indoor Environment. *Appl. Sci.* **2021**, *11*, 7148. https://doi.org/10.3390/app11157148

Academic Editors: Sylvain Bertrand, Hyo-sang Shin and Seong-Ik Han

Received: 9 June 2021
Accepted: 28 July 2021
Published: 2 August 2021

1. Introduction

The emergence of artificial intelligence and computer vision technologies bring forth a wide range of applications. As a result, various unmanned systems are being deployed in both civilian and military domains. Equipped with these technologies, self-driving cars [1–4] and autonomously navigating UAVs [5–9] are being integrated into our daily life. All of these and other important applications of integrated artificial intelligence and computer vision technologies rely, in one way or another, on training neural networks, which is crucial for the classification of objects in images taken by visual sensors.

Moreover, the ability of a computing machine to autonomously detect and classify objects leveraged the autonomous navigation of unmanned aerial vehicles in cluttered environments. This capability further incites a wide range of applications of UAVs. UAV missions, such as door-to-door package delivery, search and rescue of victims in a collapsed building, indoor first aid, and target tracking in urban environments, demand that the UAV has environmental perception. To this end, training a companion computer onboard the UAV is mandatory. Training computing machines to perceive the surrounding environment is a state-of-the-art technology, generally known by the name "machine learning": a sub-discipline of artificial intelligence. For the process of machine learning, a network of

mathematical abstracts (neurons) are layered in structured way. The depth of the network is determined by its number of layers and training this deep network to hierarchically extract desired features in input data is referred to as "deep learning". Various definitions of deep learning were reviewed by Zhang et al. [10]. A comprehensive review of deep learning and its variants was presented by Yann et al. [11] and Jurgen Schmidhuber [12]. A survey of the wide range of applications of deep learning as well as its challenges and future directions was reported by Laith et al. [13]

There are various deep learning approaches where supervised deep learning is one of the well known and widely used approaches. In this type of neural network learning approach, the dataset is labeled manually for training the network. The three commonly known neural networks to which supervised deep learning is applied are the convolutional neural network (CNN), artificial neural network (ANN), and recurrent neural network (RNN). Ever since its conception, supervised deep learning is being implemented in various areas such as the autonomous navigation of both ground [14] and aerial vehicles [15], speech and pattern recognition [16], and medical image analysis [17].

The commonly utilized network that is often implemented in deep learning for object detection and classification is CNN [18–22]. This CNN is proved to outperform other networks on various tasks [23]. There are many variants of CNN and their differences are dictated by the network parameters, such as number of layers, number of intra-layer neurons and the types of inter-layer connections in the architecture. These parameters greatly affect the efficiency of the performance of a given CNN. An extensive survey of the recent architectures of deep CNN was reported by Asiffullah et al. [24].

Neha et al. [25] analyzed three variants of CNN—AlexNet, GoogleNet and ResNet50—and reported that the number of layers in a network architecture affects the performance of CNN. Karen et al. [26] found that the increase in the number of layers of CNN enhances the performance accuracy of the network. Christian et al. [27] also proposed that increasing the depth of the network remarkably improves the performance of the network. In their study on training deep convolutional neural networks using huge image data, Alex et al. [28] concluded that a very large network, such as ImageNet LSVRC-2010, is essential to achieving good results.

Indefinite increment in network depth, however, incurs a tiresome training process, requires high storage and computing capacities, and includes the difficulties of network architectural design that lead to performance degradation. Tiresome pre-training processes, such as input data labeling and the need for high performance computers, are common setbacks for deep network implementation. A companion computer on-board a UAV has to have enough memory to store huge amounts of activations and weights of the deep network and needs to conduct resource intensive image processing in real-time for safe navigation.

Challenges related to large network training were presented and explained by Michael [29]. Soumya et al. [30] also discussed the challenges of training CNNs. Many researchers suggested remedies to the challenges. Hugo et al. [31] suggested a deep neural network training procedure that leverages the performance of the network. Kaiming et al. [32] proposed a learning framework, named residual learning, that resolves the difficulty of training large networks and performance degradation. Stephan et al. [33] proposed a stability training approach that enhances network tolerance to small perturbations in input image data.

In addition to training difficulty, the work of selecting and designing a particular deep CNN architecture is not simple. This is because the optimization algorithms have to be gauged from the point of view of specific deep learning problems. There is, however, a most appropriate optimization for a particular problem which is the result of the "no free lunch theorem" of mathematical optimization reported by Tamás et al. [34]. Ivana et al. [35] took the aforementioned network parameters and parameters such as number of filters per layer and filter size in order to perform parameter optimization and obtain a network architecture with the best performance. Gao et al. [36] introduced DenseNets,

where each layer is connected to every other layer. They reported that this network reduces some of the problems that come with increasing the depth of CNN architecture.

The remainder of this work is organized into sections. Section 2 describes the objective of this research and the methodologies followed. Problem specific input data augmentation techniques are listed and augmentation strategies are explained in Section 3. The designed network architecture and its training procedure are presented in Section 4. Results and discussions are presented in Section 5 and finally the conclusion is drawn in Section 6.

2. Problem Statement and Methodology

The objective of this research is to enable a quadcopter UAV engaged in search and rescue operations to acquire the capability of classifying objects in a wreckage of collapsed building. For this to be realized, a companion computer onboard the quadcopter has to be well trained with plausible indoor objects. For the first phase of this work, we randomly picked plausible indoor and surrounding objects, such as windows, doors, walls, columns, pipes, poles, tables, fans, nets, and trees.

For training, we considered supervised deep learning of the convolutional neural network approach. The architecture of this CNN is problem specific. Therefore, we designed a custom CNN architecture specific to the aforementioned problem. Often, network training requires big input data. There are no big data for the stated scenario. In cases when the available input data are scarce, a common method to enhance the number input data is to augment the available data through various methods. Connor et al. [37] conducted an extensive survey on augmentation methods of input image data. In our case, we used mission specific augmentation methods.

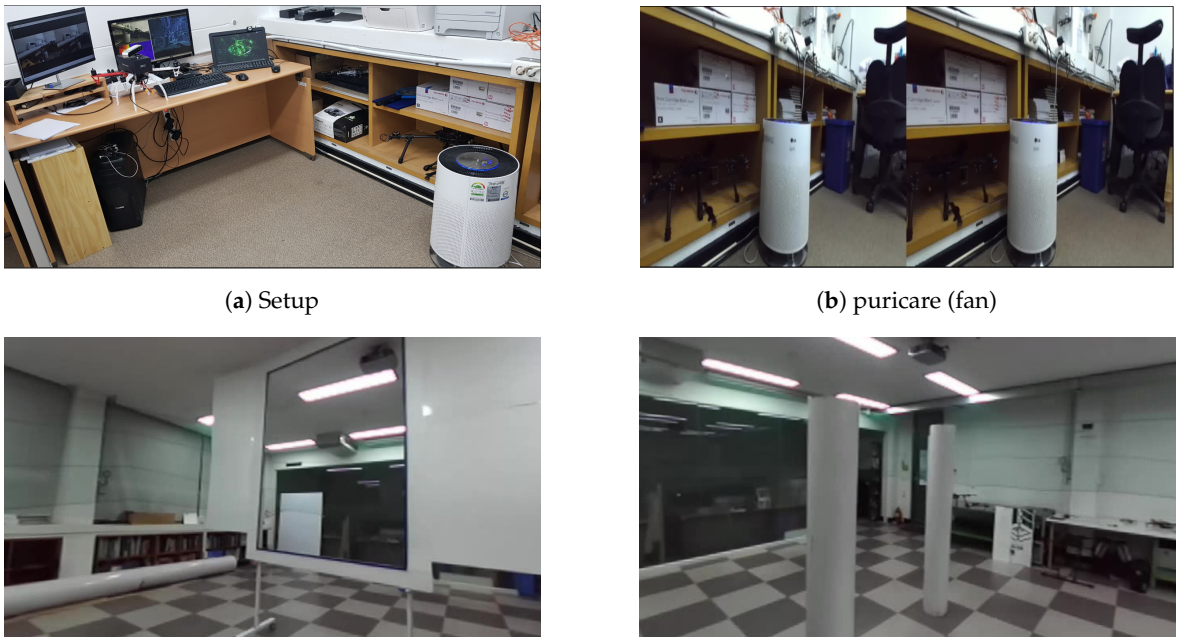
A mission specific augmented input data generation approach is pertinent to the quadcopter UAV engaged in search and rescue operations, in a collapsed building. Under such an operation, the quadcopter has to enter the collapsed building through any hole available and avoid collision with obstacles as it searches for targets. The quadcopter will possibly encounters environments with variable brightness as well as laid or inverted objects. While the quadcopter is taking images of the scene, the images can be blurred due to the vibration of the quadcopter. If images are taken during the rolling of the quadcopter, objects in the images might appear rotated. If images are taken at a glance, objects in the images appear sheared. These possibilities are taken into consideration during image data augmentation. In designing the neural network model, data cleaning and code development took most of the time and computing resources. Data training frameworks and code development are carried out with Python libraries. Image preparation and manipulation is performed with the Image Processing Toolbox™ from MathWorks®. Units' dropout is randomly implemented in a ratio of 0.1 to 0.4 on the fully connected (dense) layer of applied layers.

3. Data Preparation and Pre-Processing Methodology

A quadcopter UAV based mission for the search and rescue of a target in a collapsed building was taken into consideration during augmentation of the input image data. We used a stereo camera to collect the 3D image information in SVO format. Sample images are presented in Figure 1. The image information is then converted to 2D stereo images for training the supervised deep neural network.

Before the data were analyzed, they were organized into an appropriate form. The raw image data were prepared in such a way that they retain complete information about the environment they represent: the aspect of the environment that the data are describing. Random sample image inputs of the actual experimental indoor environment are shown in Figure 2. The integrity, completeness, validity, consistency and uniqueness of the dataset are maintained as much as possible.

The whole preparation process contains tasks such as refining, integrating and transforming input data. For our small dataset, we applied an image augmentation technique to train the network.



(**a**) Setup (**b**) puricare (fan)

(**c**) window (**d**) pole

Figure 1. Experimental setup and sample stereo images of objects used to train the network.

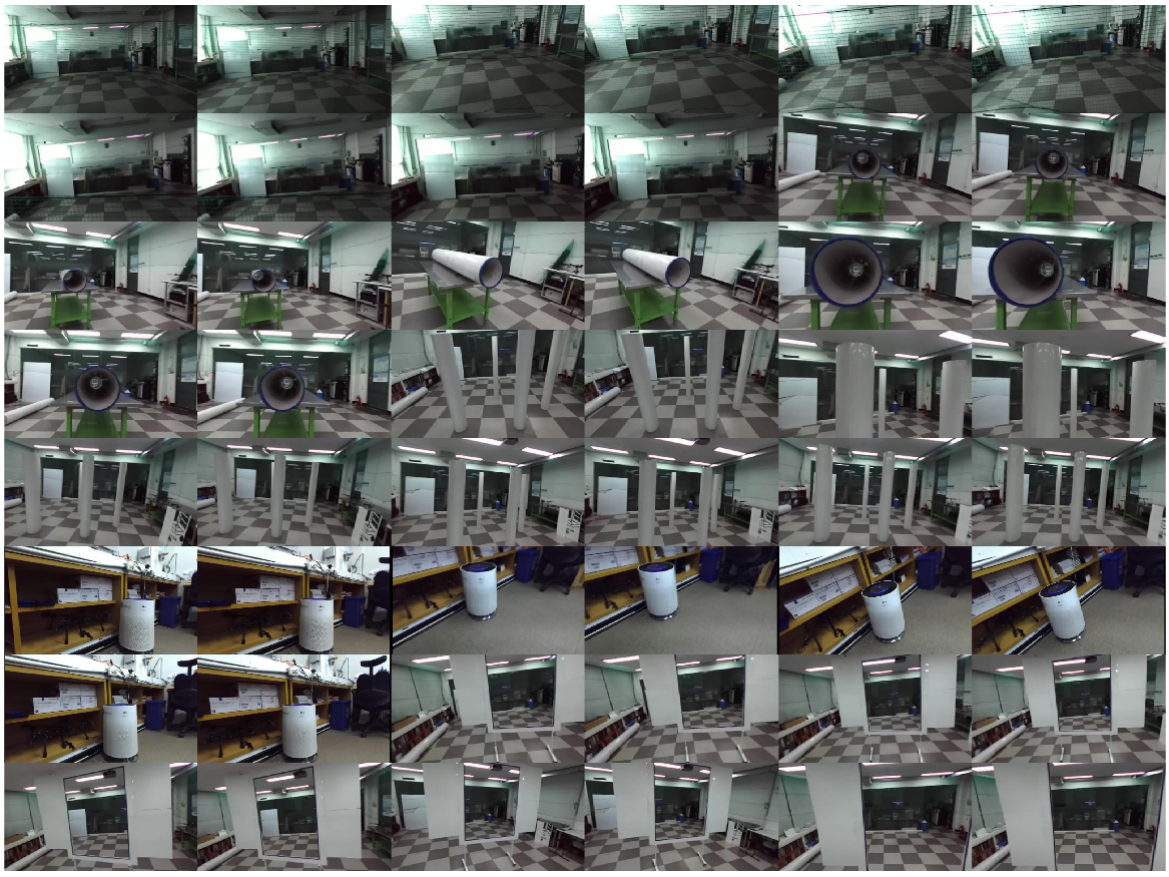


Figure 2. Sample random stereo images used to train our deep neural network.

This augmentation technique enhances the size of input data by zooming, shearing, rotating, blurring, flipping and changing the color of already existing scarce input data. The technique performs transformations to yield believable-looking images in the scene. The network model makes use of this technique to perceive wider aspects of the input

data. Deliberately introducing imperfections into our dataset was essential to making our model more resilient to the harsh realities it will encounter in real world situations. Degrading image quality by applying Gaussian blurring was one we applied to our images for this purpose. Figure 3 shows a random sample of the blurred images used for training.

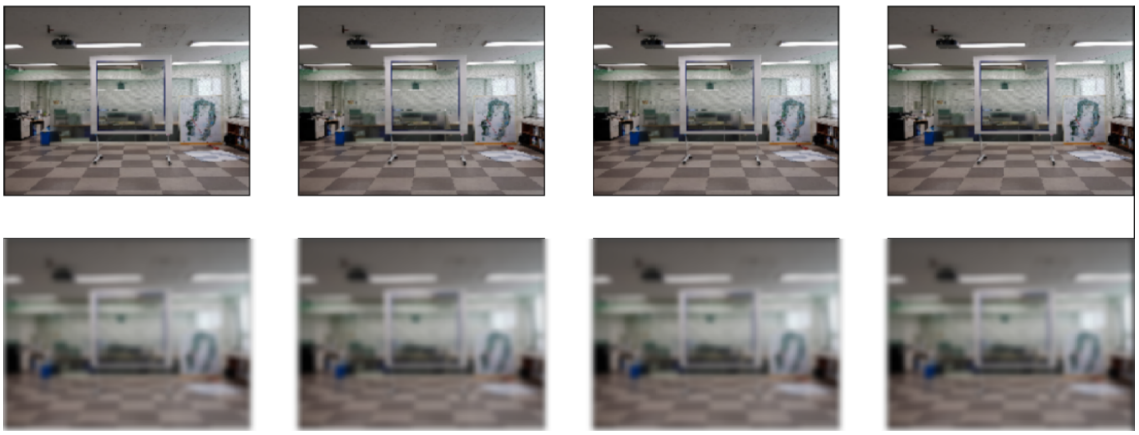
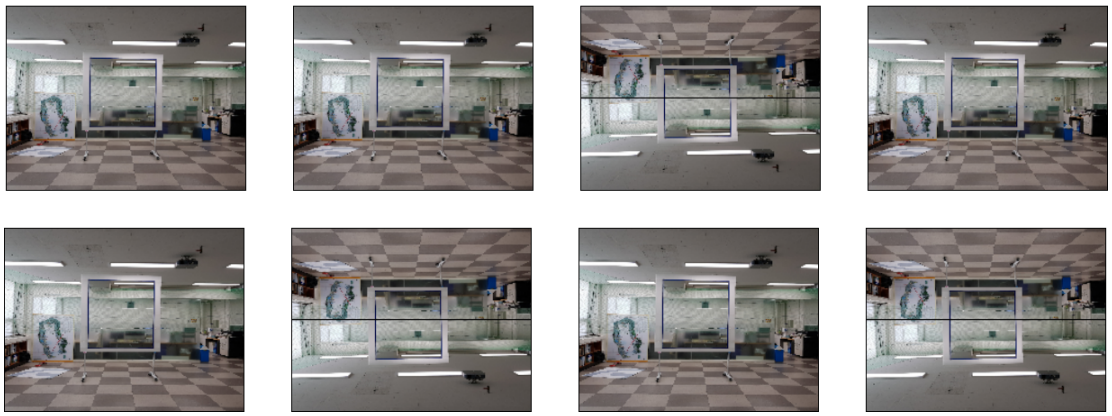


Figure 3. Original (**top**) and blurred (**bottom**) sample images.

Many types of imperfections can make their way into an image: blur, poor contrast, noise, joint photographic experts group (JPEG) compression, and more. Of these, blurring is among the most detrimental to image classification. Blurring an image is taking neighboring pixels and averaging them, in effect reducing detail and creating what can be perceived as blur. When we implement different amounts of blur, we are determining how many neighboring pixels to include. We measured this spread from a single pixel as the standard deviation in both the horizontal and vertical directions. The larger the standard deviation, the more blur an image receives. In this work, we filtered the image with a Gaussian filter of 4 standard deviations. Image flipping is one of the commonly used approaches in image augmentation techniques. Flipping images vertically or horizontally does not alter the classes of objects in the images. Sample horizontally flipped (left and right) images used in the model training are displayed in Figure 4a.

(**a**) left and right flipped

Figure 4. *Cont.*



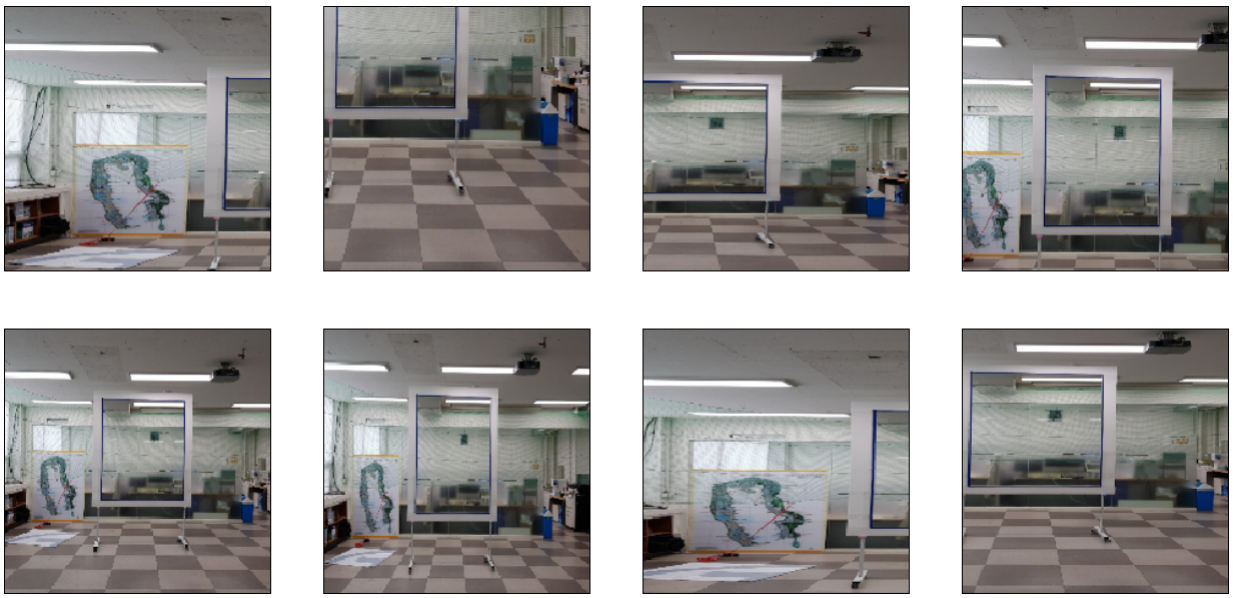
(**b**) top and bottom flipped

Figure 4. Left and right and top and bottom flipped images used for training.

Even though it is not as common as horizontal flipping, we flipped input images vertically (up-side-down) and the sample is displaced in Figure 4b. Both of these horizontal and vertical flipping techniques align with the scenario of objects under a collapsed building.

To make the network model invariant to the change in positions of objects in images, input images were randomly cropped. Each input image was randomly cropped from 10% to 100% of its original area, and the ratio of width to height of the region was randomly selected between 0.5 and 2. Sample cropped images are shown in Figure 5a with the width and height scaled to 180 pixels.

Varying the colors of input images is another image augmentation technique. We varied the brightness, contrast, saturation and hue of input images. Brightness is randomly varied between 50% to 150% of the original image and sample images are as shown in Figure 5b. Similarly, the hues of the input images were randomly varied. Sample images are displayed in Figure 6a.



(**a**) random cropped

Figure 5. *Cont.*

(**b**) random brightness

Figure 5. Randomly cropped and brightened images used for training.

We also created random color jitter instances and randomly changed the brightness, contrast, saturation and hue of the images as shown in Figure 6b. Finally, a multiple image augmentation technique was applied. All the aforementioned techniques were overlaid and applied to each input image used in the model training. Figure 6c shows the different augmentation techniques on sample images.

(**a**) random hue

(**b**) random color jitters

Figure 6. *Cont.*

(**c**) multiple augmentation

Figure 6. Images with random hue, color jitters and multiple augmentation used for training.

4. Network Architecture Design

There are two components in artificial neural network models. These are filters and network architecture. The filter defines the weight parameters and the network architecture defines parameters such as how many layers, how many neurons per layer, type of intra-layer and inter-layer neurons connections in the network. Network architecture design is problem specific. What is accurate for one problem may not perform well for the others. Therefore, designing a problem specific deep neural network architecture is a common practice with the intention of reducing computational burdens and enhancing the accuracy or speed of the network for the objective it is designed for.

In this section, our designed deep neural network architecture for object classification is presented. The designed architecture is based on CNN. As an objective, since we have very few classes of objects to classify, it is better to have very few but an effective number of layers in the network to make it light-weight for fast object classification to ensure the real-timeness of the classification processes. In this light-weight CNN design, we followed a common CNN architecture design trend [38], where the sequence of layers are with few convolutional layers and activation functions followed by pooling layers as shown in Figure 7.

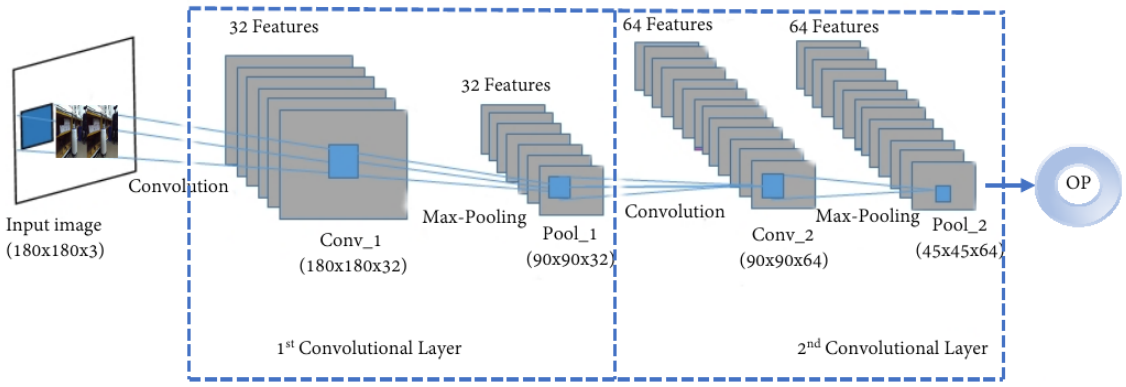
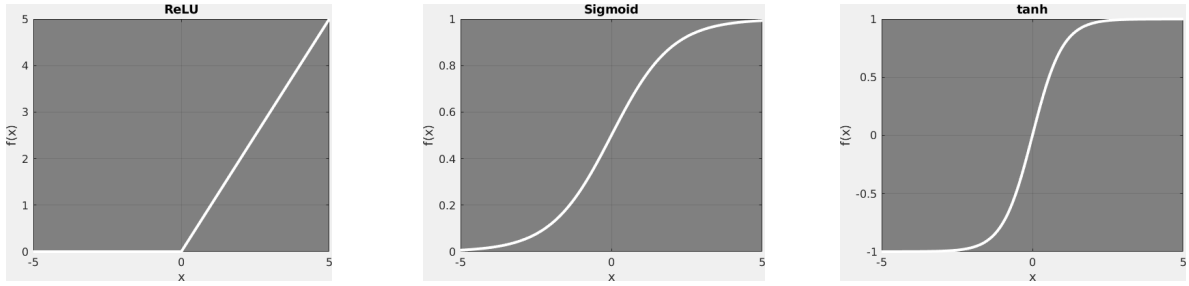


Figure 7. Data flow in the first two layers of a convolutional neural network (CNN).

Deep neural learning networks and deep learning are popular algorithms. Most of the outcomes of these algorithms depend on their cautious architectural design and the

choice of appropriate activation functions. Commonly deployed activation functions in neural network design are shown in Figure 8.



(**a**) rectified linear unit (**b**) sigmoid (**c**) tanhyperbolic

Figure 8. Different activation functions.

In Figure 7, a susceptive area of a part with a specified weight vector (a filter) is transformed bit by bit over a two-dimensional arrangement of process parameters, which are constituent of an image. The resulting arrangement of following activation incidents of this part then provide parameters to higher-level parts and so forth. The basic parts in each convolutional block are a convolutional layer, a rectified linear unit (ReLU) activation function, and a succeeding max-pooling operation.

Each layer has a particular goal. The layers may be replicated with inconsistent variables as part of the convolutional network. The types of layers we deployed in our artificial neural network design are listed as follows:

- image input layer
- convolution 2D layer
- batch normalization layer
- relu layer
- max pooling 2D layer
- fully connected layer
- softmax layer
- classification layer

There can be various layers of each kind of layer. Some convolutional nets have hundreds of layers. Convolution is the process of highlighting expected features in an image. This layer applies sliding convolutional filters to an image to extract features. A batch normalization layer normalizes each input channel across a mini-batch. It automatically divides up the input channel into batches. This reduces the sensitivity to the initialization. Relu layer is a layer that uses the rectified linear unit activation function.

Maxpooling 2D Layer creates a layer that breaks the 2D input into rectangular pooling regions and outputs the maximum value of each region. The input pool size specifies the width and height of a pooling region. Pool size can have one element (for square regions) or two for rectangular regions. The fully connected layer connects all of the inputs to the outputs with weights and biases. Softmax finds a maximum of a set of values using the logistic function. A classification layer computes the cross-entropy loss for multi-class classification problems with mutually exclusive classes.

The layers of the network are summarized in the Table 1. The neural network design contains of three convolutional structures each of which has a layer that calculates the maximum value. There is a fully connected layer with 256 units. The neural network model uses a rectified linear unit (relu) activation function. The image-quantity is a variable quantity of configuration (32, 180, 180, 3). That is a quantity of 32 images of the configuration of $180 \times 180 \times 3$, where the last feature represents the colors red, blue and green channels.

Table 1. Some layers of the neural network used for object classification training.

Layer (Type)	Output Shape	Parameter
Rescaling	(None, 180, 180, 3)	0
conv1 (Conv2D)	(None, 180, 180, 32)	896
max-pooling1 (MaxPooling2D)	(None, 90, 90, 32)	0
conv2 (Conv2D)	(None, 90, 90, 64)	18,496
max-pooling2 (MaxPooling2D)	(None, 45, 45, 64)	0
conv3 (Conv2D)	(None, 45, 45, 128)	73,856
max-pooling3 (MaxPooling2D)	(None, 22, 22, 128)	0
Flatten	(None, 61,952)	0
dense (Dense)	(None, 256)	15,859,968
dense-1 (Dense)	(None, 5)	1285

4.1. Training the Neural Network

The task of training a deep neural network is the biggest setback of all the processes therein. This setback worsens as the depth of the network increases. Xavier et al. [39] discussed the difficulty of training deep neural networks and suggested the work around procedures to be taken during training. In this section, different methods are explored to optimize the network output. The general network training flowchart is shown in Figure 9.

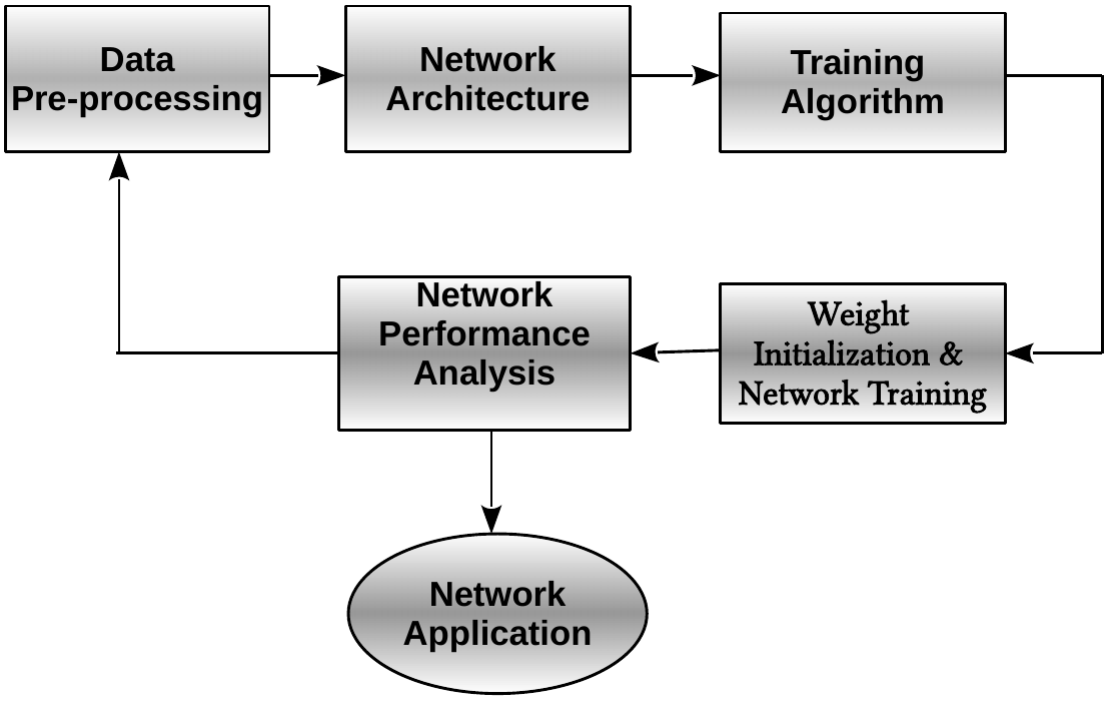


Figure 9. Flowchart of neural network training, analysis and application process.

Network training is all about determining the optimal values of network parameters for which the network performs best. Neural networks are a common category of parametric nonlinear functions of the form that transforms a vector $\mathbf{u}$ of input variables to a vector $\mathbf{v}$ of output variables. A simple way to obtain network parameters is to make the similarity of curve fitting, and thus minimize the sum of squares error function. Given a training set containing a set of input vectors $\mathbf{u}_n$, where $n = 1, \ldots, N$, together with a corresponding set of target vectors $\top_n$, we calculate the minimum of the error function.

$$\xi(\mathbf{\Omega}) = \frac{1}{2}\sum_{n=1}^{N}\|\mathbf{v}(\mathbf{u}_n, \mathbf{\Omega}) - \top_n\|^2. \tag{1}$$

Considering a single target variable k that can take any real value and that k has a Gaussian distribution with an x-dependent mean, which is given by the output of the neural network, so that

$$p(k|\mathbf{u}, \mathbf{\Omega}) = \mathbb{N}(\top|v(\mathbf{u}, \mathbf{\Omega}), \alpha^{-1}), \tag{2}$$

where α is the precision of Gaussian noise. For N independent data $\mathbf{\Theta} = \{\mathbf{u}_1, ..., \mathbf{u}_N\}$ with corresponding target values $\top = \{\top_1, ..., \top_N\}$, we can build the consistent probability function.

$$p(\top|\mathbf{\Theta}, \mathbf{\Omega}, \alpha) = \prod_{n=1}^{N} p(\top_n|\mathbf{u}_n, \mathbf{\Omega}, \alpha). \tag{3}$$

We can evaluate the error function by calculating the negative logarithm of the probability function.

$$\frac{\alpha}{2}\sum_{n=1}^{N}\{v(\mathbf{u}_n, \mathbf{\Omega}) - \top_n\}^2 - \frac{N}{2}\ln\alpha + \frac{N}{2}\ln(2\pi). \tag{4}$$

Equation (4) is used to learn parameters $\mathbf{\Omega}$ and α. In neural networks design literature, it is customary to take into account the minimization of the error function more than the maximization of the probability function. We are following the same approach here. First we obtain $\mathbf{\Omega}$ by maximizing the probability function (or minimizing the sum-of-squares error function).

$$\xi(\mathbf{\Omega}) = \frac{1}{2}\sum_{n=1}^{N}\{v(\mathbf{u}_n, \mathbf{\Omega}) - \top_n\}^2, \tag{5}$$

where operational constants are rejected. The value of $\mathbf{\Omega}$ obtained by minimizing $\xi(\mathbf{\Omega})$ represented by $\mathbf{\Omega}_{mxl}$ corresponds to the maximum probability solution. Having obtained $\mathbf{\Omega}_{mxl}$, α can be found by minimizing the negative logarithm of probability

$$\frac{1}{\alpha_{mxl}} = \frac{1}{N}\sum_{n=1}^{N}\{v(\mathbf{u}_n, \mathbf{\Omega}_{mxl}) - \top_n\}^2. \tag{6}$$

The practically local maximum of the probability may be obtained that corresponds to the local minimum of the error function. If there are multiple target variables, which are assumed to be independent depending on $\mathbf{u}$ and $\mathbf{\Omega}$ with common α, then the conditional distribution of the target values is obtained by:

$$p(\top|\mathbf{u}, \mathbf{\Omega}) = \mathbb{N}(\top|\mathbf{v}(\mathbf{u}, \mathbf{\Omega}), \alpha^{-1}\mathbf{I}). \tag{7}$$

With the same argument as for a single target variable, we obtain the maximum likelihood weights by minimizing the sum-of-squares error function. The noise precision is given by

$$\frac{1}{\alpha_{mxl}} = \frac{1}{NK}\sum_{n=1}^{N}\|\mathbf{v}(\mathbf{u}_n, \mathbf{\Omega}_{mxl}) - \top_n\|^2, \tag{8}$$

where K is the number of target variables. There is a real matching of the error function and the output unit activation function. In regression, the network can be viewed as having an output activation function that is the unitary, so that $v_k = b_k$. The corresponding sum-of-squares error function can be expressed as

$$\frac{\partial\xi}{\partial b_\top} = v_k - \top_k, \tag{9}$$

which is used for error backpropagation. In the case of binary classification, in which there is a single target variable k so that $k = 1$ represents class $\mathtt{CL}_1$ and $k = 0$ denotes class $\mathtt{CL}_2$. Assuming a network to have a single output whose activation function is a logistic sigmoid,

$$v = \sigma(b) = \frac{1}{1 + exp(-b)}. \tag{10}$$

In this instance, $0 \leq v(\mathbf{u}, \mathbf{\Omega}) \leq 1$. $v(\mathbf{u}, \mathbf{\Omega})$ is explained as the conditional probability $p(\mathtt{CL}_1|\mathbf{u})$ given by $1 - v(\mathbf{u}, \mathbf{\Omega})$. The conditional distribution of targets given inputs is then a Bernoulli distribution with the form

$$p(\top|\mathbf{u}, \mathbf{\Omega}) = v(\mathbf{u}, \mathbf{\Omega})^{\top}\{1 - v(\mathbf{u}, \mathbf{\Omega})\}^{1-\top}. \tag{11}$$

Considering a training set of unconventional results, then the error function, which is given by the negative logarithm probability, is a *measure of the performance the classifier or log loss* error function which is given by the form

$$\xi(\mathbf{\Omega}) = -\sum_{n=1}^{N}\{\top_n \ln v_n + (1 - \top_n)\ln(1 - v_n)\}, \tag{12}$$

where v_n denotes $v(\mathbf{u}, \mathbf{\Omega})$. Using the log loss error function in place of the sum-of-squares for a classifier improves training and generalization. If there are K separate binary classifications to be done, we can use a network that has K outputs, each of which has a logistic sigmoid activation function. Associated with each output is a binary class label $k_t \in \{0, 1\}$, where $t = 1, \ldots, \top$. Assuming that the class labels are individualistic, given the input vectors, the conditional distribution of the target is

$$p(\top|\mathbf{u}, \mathbf{\Omega}) = \prod_{k=1}^{K}(\mathbf{u}, \mathbf{\Omega})^{\top_k}[1 - v_{\top}(\mathbf{u}, \mathbf{\Omega})]^{1-\top_k}. \tag{13}$$

Evaluating the negative logarithm of the corresponding probability function then gives the following error function

$$\xi(\mathbf{\Omega}) = -\sum_{n=1}^{N}\sum_{k=1}^{K}\{k_{n\top} \ln v_{n\top} + (1 - k_{n\top})\ln(1 - v_{n\top})\}, \tag{14}$$

where $v_{n\top}$ represents $v_{\top}(\mathbf{u}_n, \mathbf{\Omega})$. Considering the standard multi-class classification problem in which each input is assigned to one of the $\top$ mutually exclusive classes. The binary target variables $k_{\top} \in \{0, 1\}$ have a 1-of-$\top$ coding strategy showing the class, and the network outputs are explained as $v_{\top}(\mathbf{u}, \mathbf{\Omega}) = p(k_{\top} = 1|\mathbf{u})$, which results in the following error function

$$\xi(\mathbf{\Omega}) = -\sum_{n=1}^{N}\sum_{\top=1}^{K} k_{\top n} \ln v_{\top}(\mathbf{u}_n, \mathbf{\Omega}). \tag{15}$$

The return unitary activation function, which correlates with the canonical link, is then represented by the softmax function

$$v_{\top}(\mathbf{u}, \mathbf{\Omega}) = \frac{exp(b_t(\mathbf{u}, \mathbf{\Omega}))}{\sum_j exp(b_j(\mathbf{u}, \mathbf{\Omega}))}, \tag{16}$$

which fulfills $0 \leq v_{\top} \leq 1$ and $\sum_{\top} v_{\top} = 1$. In general, there is a simple option of both output part activation function and complement error function, in accordance with the type of solution being sought. In the case of regression, linear outputs and a sum-of-squares error are used. For multiple unconstrained duplicate classifications we use logistic sigmoid outputs with a log-loss error function. In the case of multi-class classification we implement softmax outputs with the comparable multi-class log-loss error function. For

a classifier problem concerning two classes, a single logistic sigmoid output can be used, or optionally network with two outputs having a softmax output activation function can be implemented.

Parameter Optimization

We will find a weight vector $\mathbf{\Omega}$ that decreases the selected function $\xi(\mathbf{\Omega})$. If we make a small pace in weight span from $\mathbf{\Omega}$ to $\mathbf{\Omega} + \delta\mathbf{\Omega}$, then the pace in the error function is $\delta\xi \simeq \delta\mathbf{\Omega}^T \nabla\xi(\mathbf{\Omega})$, where the vector $\nabla\xi(\mathbf{\Omega})$ shows in the direction of the largest rate of the pace of the error function. Because the error is an even, continuous function of $\mathbf{\Omega}$, its least value will happen at a point in weight span such that the slope of the error function disappears, therefore

$$\nabla\xi(\mathbf{\Omega}) = 0, \tag{17}$$

or else we could make a small pace in the direction of $-\nabla\xi(\mathbf{\Omega})$ and thereby further decrease the error. Points at which the slope disappears are known as static points, and may be categorized into minimum, maximum, and saddle points. Our aim is to find a vector $\mathbf{\Omega}$ such that $\xi(\mathbf{\Omega})$ takes its least value. Nevertheless, the error function always has an extremely nonlinear dependence on the weights and bias parameters. So there will be many points in weight span where the slope disappears (or is numerically very small). For any point $\mathbf{\Omega}$ that is a local minimum, there will be other points in the weight span that are similar minima.

Additionally, there will always be many different static points and in particular multiple different minima. A minimum that corresponds to the least value of the error function for any weight vector is called a global minimum. Any other minima corresponding to larger values of the error function are called local minima. For a better application of neural networks, it may not be important to find the global minimum but it may be good to compare many local minima to find an acceptably good result. because there is no expectation of obtaining an analytical result to the equation $\nabla\xi(\mathbf{\Omega}) = 0$, we seek iterative numerical methods. Most numerical methods include selecting some initial value $\mathbf{\Omega}^0$ for the weight vector and then operating through the weight span in sequence of pace of the following form,

$$\mathbf{\Omega}^{(\lambda+1)} = \mathbf{\Omega}^{(\lambda)} + \Delta\mathbf{\Omega}^{(\lambda)}, \tag{18}$$

where λ labels the iteration step. Separate algorithms use many alternatives for the weight vector improvement $\Delta\mathbf{\Omega}^{(\lambda)}$. There are methods that make use of slope detail and hence need, after each improvement, the value of $\nabla\xi(\mathbf{\Omega})$ to be calculated at the new weight vector $\mathbf{\Omega}^{(\lambda+1)}$. It is viable to calculate the slope of an error function effectively by means of the backpropagation procedure. The use of this gradient detail can lead to better improvements in the speed with which the minima of the error function can be detected.

The simplest way to use gradient detail is to choose the weight update in Equation (18) to include a small pace in the direction of the negative slope so that

$$\mathbf{\Omega}^{(\lambda+1)} = \mathbf{\Omega}^{(\lambda)} - \mu\nabla\xi(\mathbf{\Omega}^{(\lambda)}), \tag{19}$$

where the specification $\mu > 0$ is called the learning rate. Following each such improvement, the slope is re-calculated for the new weight vector and the computation is repeated. The error function is explained regarding a training set, and so each pace needs all the training sets to be computed in order to estimate $\nabla\xi$. Methods that use the whole dataset at once are named batch methods. At each pace, the weight vector is moved in the direction of the largest rate of decrease of the error function and so this approach is called the gradient descent or steepest descent. Even though this approach might look sensible, in fact it turns out to be a poor method. For batch optimization, there are more effective methods, such as conjugate gradients and quasi-Newton methods, which are stronger and faster than the simple gradient descent. Far from gradient descent, these algorithms have the effects that the error function always diminishes at each iteration except for the weight vector has reached a local or global minimum.

To find an acceptably good minimum, it may be mandatory to run a gradient-based algorithm many times, each time using a different randomly selected starting point, and comparing the resulting performance on an independent validation set. However, there is an on-line sort of gradient descent that has shown convenience in practice for training neural networks. Error functions based on maximum probability for a set of independent observations include a sum of terms, one for every data point,

$$\xi(\mathbf{\Omega}) = \sum_{n=1}^{N} \xi_n(\mathbf{\Omega}). \tag{20}$$

On-line gradient descent, also known as *sequential gradient descent* or *stochastic gradient descent*, makes an update to the weight vector based on one data point at a time so that

$$\mathbf{\Omega}^{(\lambda+1)} = \mathbf{\Omega}^{(\lambda)} - \mu \nabla \xi_n(\mathbf{\Omega}^{(\lambda)}). \tag{21}$$

In this way, the update is redone by repeating through the data either in succession or by random selection of points and replacing. The power of on-line methods handle superfluity in the data much more effectively in contrast to the batch methods.

One of the challenges with training networks with a limited number of input data is over fitting which greatly degrades the performance of the network. A recommended way of reducing this overfitting is through randomly dropping a certain number of neurons from layers [40]. We implemented this regularization method and were able to achieve an improvement in the performance of our network.

5. Results and Discussion

The datasets used for this study are problem specific and a neural network is designed for this specific purpose that does not rely on the use of other pre-trained networks and transfer learning. So network training was done from scratch. Network training was performed on NVIDIA® AGX Xavier Developer Kit for 50 epochs. The accuracies for mini-batch, validation and the corresponding losses are summarized in Table 2 for some selected epochs.

Using sequential gradient descent (SGD) with momentum (0.9) and a piece wise adjustment to the learning rate schedule, training was run on a 512-core Volta GPU with Tensor Cores. For GPUs with less memory, it may be necessary to reduce the batch size. The choice of SGD optimization has advantages for small datasets. Based on the SGD optimization scheme, ten classes of indoor objects were used for training the model. The validation precision of the architecture to predict an object which was not contained in training data was 76.5%. Furthermore, the performance of the network was improved by fine tuning the weights (parameters) and learning rate. More importantly, the performance of the network improved by about 10.5% using data augmentation and dropout techniques.

As we observe from our experimental results, training and validation precision are separated by a large boundary in the case where data augmentation was not implemented. The model in this case attained only around 68.5% precision on the validation dataset. Shown in Figure 10, the training precision improves linearly over time, but the validation precision stalls around 68.5% in the learning process. Moreover, the difference in precision between training and validation losses is clearly observed which is an indication of overfitting.

Table 2. Network training on 512-core Volta GPU result summary.

Epoch	Mini-Batch Accuracy	Vald. Accuracy	Mini-Batch Loss	Vald. Loss
1	37.54%	57.22%	1.48	1.10
4	68.53%	67.03%	0.82	0.94
8	74.08%	70.44%	0.68	0.74
12	79.09%	72.62%	0.55	0.68
16	82.30%	75.20%	0.50	0.69
20	83.55%	73.98%	0.41	0.75
24	88.20%	74.52%	0.40	0.77
27	88.09%	74.39%	0.32	0.84
31	91.62%	74.39%	0.26	0.78
35	92.04%	76.84%	0.23	0.83
39	93.08%	76.29%	0.18	1.06
43	94.56%	72.62%	0.16	1.14
47	93.89%	76.98%	0.18	0.90
50	94.48%	76.16%	0.15	1.09

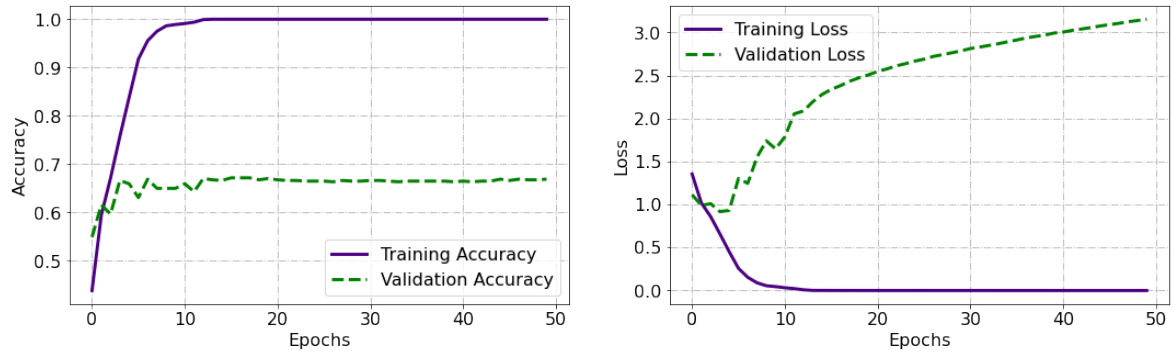


Figure 10. Model performance without data augmentation.

When there is a small number of training datasets, the model sometimes learns from unwanted details from the training dataset that negatively impact the performance of the model on new datasets. This means that the model will not be able to generalize on a new dataset. To reduce the problem of overfitting, we implemented image augmentation and dropout techniques. Image augmentation produces more training images from our existing image by augmenting them using random transformations that yield believable-looking images. This helps expose the model to more aspects of the image and generalize better. Image augmentation as depicted in Figure 11 improved the model performance to well above 74.5%. At the same time, the loss during model training is much reduced, indicating a better model performance.

In the dropout scheme, we randomly decreased the number of output units in the dense layer in a range of 10% to 40%. Despite the fact that there was not much noticeable improvement as a result of dropout in training and validation accuracy, there was a progressive improvement of the validation and loss functions as shown in Figure 12a–d. Increasing the number of dropouts from the output units again is not a guarantee for better model performance as it may degrade model performance. In this study, the best model performance is achieved for both validation precision and validation loss at 30% dropout of the output units.

The performance of the trained model network is evaluated using validation datasets. Validation data are not used to train the network. Our experimental results reveal that the model classified the data in the training dataset with a success rate of 94.5% and validated results with a success rate of 76.5%. As shown in Figure 12, the training loss reveals how well the model is fitting the training data, while the validation loss reveals how well the

model fits new data. Very successful rates on both training and validation loss are achieved at a 30% dropout, which are well below 0.3 and 0.8, respectively.

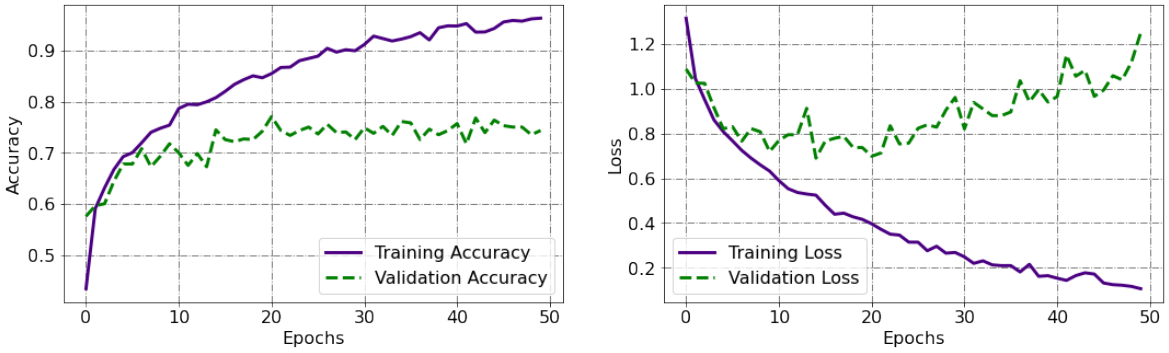
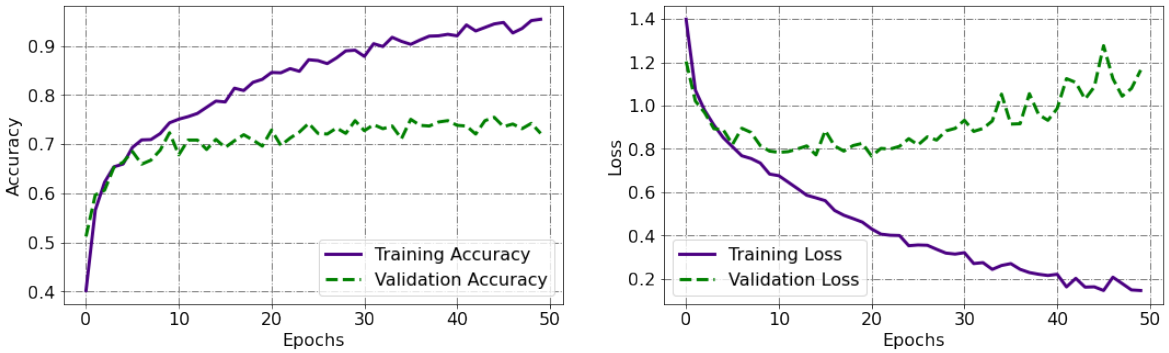
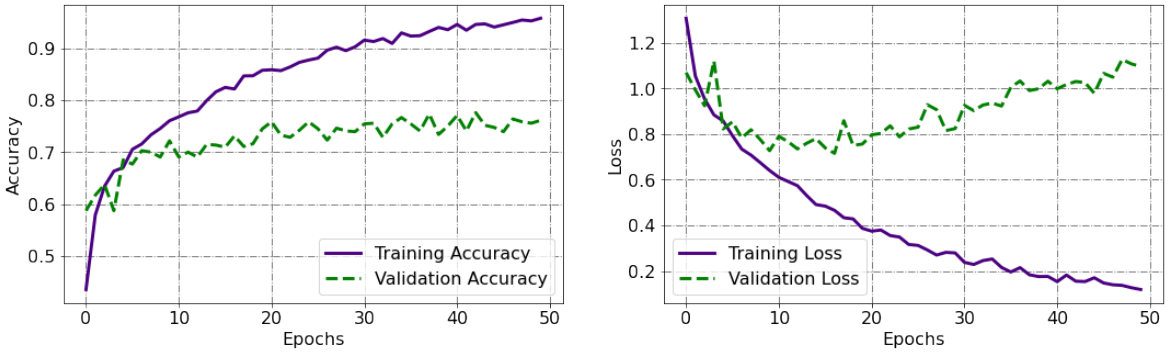


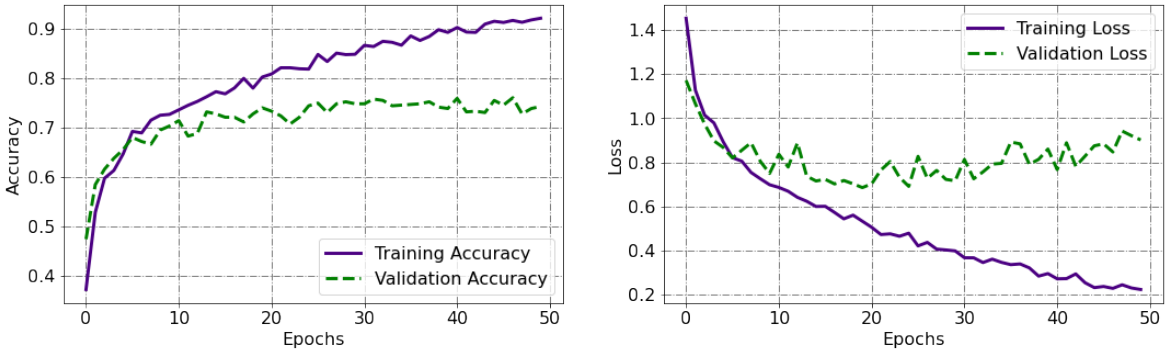
Figure 11. Model performance with data augmentation.



(**a**) 10% dropout

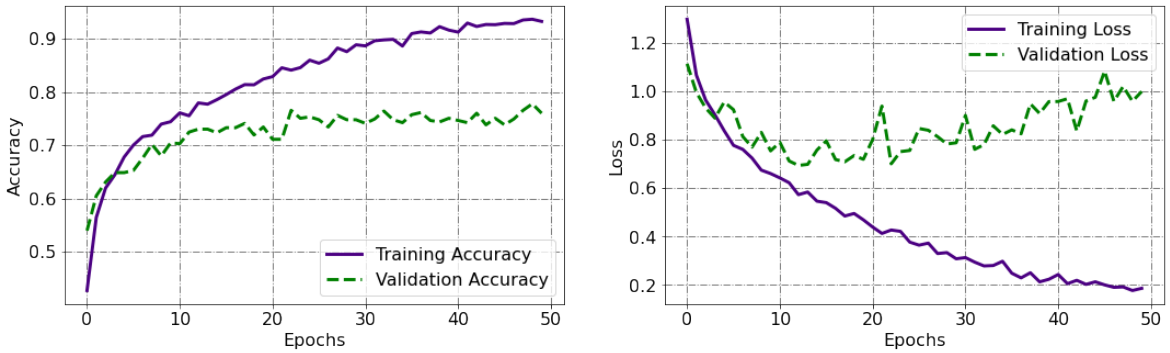


(**b**) 20% dropout



(**c**) 30% dropout

Figure 12. *Cont.*



(**d**) 40% dropout

Figure 12. Improvement in validation accuracy and loss function of the model architecture.

To study the precision over various label groups, the confusion matrix (error matrix) is plotted, which is shown in Figure 13. The label imbalance noted in the training set is an issue in the classification accuracy. The confusion chart (matrix) illustrates higher precision and recall for walls and confuses most tables with doors. Even if there are very few miss-predicted objects, the overall performance of the model to predict an object that was not contained in the training or validation sets is successful. Since the purpose of this study is to demonstrate a basic classification network training approach with raw data, possible next steps that could be taken to improve classification performance, such as re-sampling the training set or achieving better label balance or using a loss function more robustly to label imbalance (e.g., weighted cross-entropy,) will be explored in future studies.

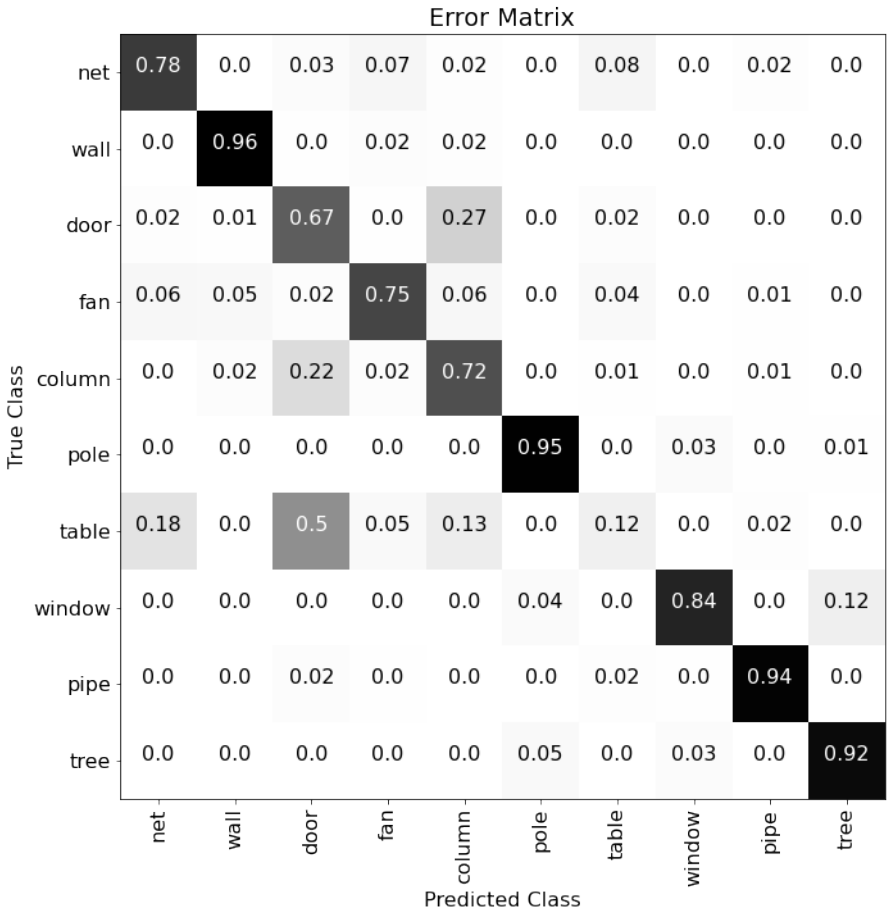


Figure 13. Confusion chart illustrating precision for each class.

6. Conclusions

A wide diversity of design and optimization techniques are used to design neural networks and to analyze their performance. The goal of these novel approaches is not to restrain the expertise architect but is rather to provide hands-on ways and mechanisms that can help obtain less complex, more robust and high-performance designs. In this study, we designed and presented a supervised deep neural network architecture for accurate real-time classification of objects in cluttered indoor environments. We have performed an in-depth analysis of augmentation in image classification for the case where there is not a large dataset. Our experimental results revealed that model performance is enhanced by enforcing different augmentation techniques provided that the level of noise remains reasonable.

To overcome data constraints in the development of the model that require large datasets, we implemented a data augmentation technique for each class of objects. Moreover, we implemented a dropout technique to further improve the performance of the model and reduce the validation loss. In this case, the model achieved a success rate of 94.5% and 76.5% for the training and validation datasets, respectively. Experimental results reveal that the model architecture performs very well in predicting new data that are not included in either the training or validation datasets. In our future work, we will deploy our model on different problems related to images and explore optimal ways to hasten the computational time.

Author Contributions: In this manuscript, model design for the object classifier, hardware integration for collecting data, and model validation and visualization were done by B.E. The overall conceptual design of the methodology, advisory work on pre-training data augmentation and manuscript editing were done by A.T. Data collection, profiling, labeling and arrangement was done by H.-Y.S. and B.-S.K. was responsible for the technical layout of the manuscript and the supervision of the research work in progress. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Korea Institute for Advancement of Technology (KIAT) grant funded by the Korean Government (MOTIE) (N0002431, The Competency Development Program for Industry Specialist).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: Authors mentioned in this manuscript have strongly involved in this study from the start to the end. The manuscript were thoroughly reviewed by the authors before it's submission to journal of Applied Science. This manuscript has not been submitted to another journal for publication.

References

1. Naghavi, S.H.; Avaznia, C.; Talebi, H. Integrated real-time object detection for self-driving vehicles. In Proceedings of the 10th Iranian Conference on Machine Vision and Image Processing, Isfahan, Iran, 22–23 November 2017.
2. Liu, D.; Cui, Y.; Chen, Y.; Zhang, J.; Fan, B. Video object detection for autonomous driving: Motion-aid feature calibration. *Neurocomputing* **2020**, *409*, 1–11. [CrossRef]
3. Wang, Y.; Liu, D.; Jeon, H.; Chu, Z.; Matson, E. End-to-end learning approach for autonomous driving: A convolutional neural network model. In Proceedings of the International Conference on Agents and Artificial Intelligence 2019, Prague, Czech Republic, 19–21 February 2019.
4. Saloni, W. The Role of Autonomous Unmanned Ground Vehicle Technologies in Defense Applications. Aerospace & Defense Technology Magazine. 2020. Available online: https://www.aerodefensetech.com/component/content/article/adt/features/articles/37888 (accessed on 1 April 2021).
5. Niu, H.; Gonzalez-Prelcic, N.; Heath, R.W. A uav-based traffic monitoring system-invited paper. In Proceedings of the IEEE 87th Vehicular Technology Conference, Porto, Portugal, 3–6 June 2018.
6. Sarthak, B.; Sujit, P. UAV Target Tracking in Urban Environments Using Deep Reinforcement Learning. *arXiv* **2020**. arXiv:2007.10934.
7. Zhen, J.; Balasuriya, A.; Subhash, C. Autonomous vehicles navigation with visual target tracking: Technical approaches. *Algorithms* **2008**, *1*, 153–182.

8. Aguilar, W.G.; Luna, M.A.; Moya, J.F.; Abad, V.; Parra, H.; Ruiz, H. Pedestrian detection for UAVs using cascade classifiers with meanshift. In Proceedings of the IEEE 11th International Conference on Semantic Computing, San Diego, CA, USA, 30 January–1 February 2017.
9. Gageik, N.; Benz, P.; Montenegro, S. Obstacle detection and collision avoidance for a uav with complementary low-cost sensors. *IEEE Access* **2015**, *3*, 599–609. [CrossRef]
10. Zhang, W.J.; Yang, G.; Lin, Y.; Gupta, M.M.; Ji, C. On the definition of deep learning. In Proceedings of the 2018 World Automation Congress (WAC), Stevenson, WA, USA, 3–6 June 2018.
11. Yann, L.; Yoshua, B.; Geoffrey, H. Deep learning. *Nature* **2015**, *521*, 436–444.
12. Jurgen, S. Deep learning in neural networks: An overview. *Neural Netw.* **2015**, *61*, 85–117.
13. Alzubaidi, L.; Zhang, J.; Humaidi, A.J.; Al-Dujaili, A.; Duan, Y.; Al-Shamma, O.; Santamaría, J.; Fadhel, M.A.; Al-Amidie, M.; Farhan, L. Review of deep learning: Concepts, CNN architecture, challenges, applications, future directions. *J. Big Data* **2021**, *8*, 1–74. [CrossRef]
14. Jefferson, S.; Gustavo, P.; Fernando, O.; Denis, W. Vision-Based Autonomous Navigation Using Supervised Learning Techniques. In Proceedings of the 12th Engineering Applications of Neural Networks and 7th Artificial Intelligence Applications and Innovations, Corfu, Greece, 15–18 September 2011.
15. Fabrice, R.N. Machine Learning and a Small Autonomous Aerial Vehicle Part 1: Navigation and Supervised Deep Learning. *Tech. Rep.* **2018**. [CrossRef]
16. Wang, D.; Chen, J. Supervised Speech Separation Based on Deep Learning: An Overview. Available online: https://arxiv.org/pdf/1708.07524.pdf (accessed on 1 April 2021).
17. Yang, X.; Kwitt, R.; Niethammer, M. Fast predictive image registration. In *Deep Learning and Data Labeling for Medical Applications*; Springer: Cham, Switzerland, 2016; pp. 48–57.
18. Louati, H.; Bechikh, S.; Louati, A.; Hung, C.C.; Said, L.B. Deep convolutional neural network architecture design as a bi-level optimization problem. *Neurocomputing* **2021**, *439*, 44–62. [CrossRef]
19. Bayot, R.; Gonalves, T. A Survey on Object Classfication Using Convolutional Neural Networks. 2015. Available online: https://core.ac.uk/download/pdf/62473376.pdf (accessed on 1 April 2021).
20. Thumu, K.; Gurrala, N.R.; Srinivasan, N. Object Classification and Detection using Deep Convolution Neural Network Architecture. *Int. J. Recent Technol. Eng.* **2020**. [CrossRef]
21. Rikiya, Y.; Mizuho, N.; Richard, K.G.D.; Kaori, T. Convolutional neural networks: An overview and application in radiology. *Insights Imaging* **2018**, *9*, 611–629.
22. Richard, L.; Roberto, F. Space Object Classification Using Deep Convolutional Neural Networks. In Proceedings of the 19th International Conference on Information Fusion, Big Sky, MT, USA, 3–10 March 2018.
23. Bai, S.; Kolter, J.Z.; Koltun, V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv* **2018**, arXiv:1803.01271.
24. Asifullah, K.; Anabia, S.; Umme, Z.; Aqsa, S.Q. A survey of the Recent Architectures of Deep Convolutional Neural Networks. *Artif. Intell. Rev.* **2020**, *53*, 5455–5516.
25. Neha, S.; Vibhor, J.; Anju, M. An Analysis Of Convolutional Neural Networks For Image Classification. *Procedia Comput. Sci.* **2018**, *132*, 377–384.
26. Karen, S.; Andrew, Z. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv* **2015**, arXiv:1409.1556.
27. Christian, S.; Wei, L.; Yangqing, J.; Pierre, S.; Scott, R.; Dragomir, A.; Dumitru, E.; Vincent, V.; Andrew, R. Going Deeper with Convolutions. *arXiv* **2015**, arXiv:1409.4842.
28. Alex, K.; Ilya, S.; Geoffrey, E.H. ImageNet Classification with Deep Convolutional Neural Networks. *Imagenet Compet.* **2012**, *25*, 1097–1105.
29. Nielsen, M. Neural Networks and Deep Learning. Free Online Book, Michael Nielsen, 2019; Chapter 5. Available online: http://neuralnetworksanddeeplearning.com/about.html (accessed on 1 April 2021).
30. Soumya, J.; Dhirendra, K.V.; Gaurav, S.; Amit, P. Issues in Training a Convolutional Neural Network Model for Image Classification. *Adv. Comput. Data Sci.* **2019**, doi10.1007/978-981-13-9942-8_27. [CrossRef]
31. Hugo, L.; Yoshua, B.; Jerome, L.; Pascal, L. Exploring Strategies for Training Deep Neural Networks. *J. Mach. Learn. Res.* **2009**. [CrossRef]
32. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. Computer Vision and Pattern Recognition. *arXiv* **2015**, arXiv:1512.03385.
33. Stephan, Z.; Yang S.; Thomas, L.; Ian G. Improving the Robustness of Deep Neural Networks via Stability Training. *arXiv* 2016, arXiv:1604.04326.
34. Tamás, O.; Anton, R.; Ants, K.; Pedro, A.; David, P.; Jan, K.; Pavel, K. Robust Design Optimization and Emerging Technologies for Electrical Machines: Challenges and Open Problems. *Appl. Sci.* **2020**, *10*, 6653.
35. Ivana, S.; Eva, T.; Nebojsa, B.; Miodrag, Z.; Marko, B.; Milan, T. Designing Convolutional Neural Network Architecture by the Firefly Algorithm. *Int. Young Eng. Forum* **2019**. [CrossRef]
36. Gao, H.; Zhuang, L.; van der Laurens, M.; Kilian, Q.W. Densely Connected Convolutional Networks. *arXiv* **2016**, arXiv:1608.06993.
37. Connor, S.; Taghi, K.M. A survey on Image Data Augmentation of Deep Learning. *J. Big Data* **2019**, *6*, 1–48.

38. Keiron, O.; Ryan, N. An Introduction to Convolutional Neural Networks. Neural and Evolutionary Computing. *arXiv* **2015**, arXiv:1511.08458.
39. Xavier, G.; Yoshua, B. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of Machine Learning Research, Sardinia, Italy, 13–15 May 2010.
40. Geoffrey, E.H.; Nitish, S.; Alex, K.; Ilya, S.; Ruslan, R.S. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv* **2012**, arXiv:1207.0580.