Computers and Electrical Engineering 102 (2022) 108271

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

Cloudy/clear weather classification using deep learning techniques with cloud images☆

Mürüvvet Kalkan [a,*], Gazi Erkan Bostancı [a], Mehmet Serdar Güzel [a], Buğrahan Kalkan [b], Şifa Özsarı [a], Ömürhan Soysal [a], Güven Köse [a]

[a] Computer Engineering Department, Ankara University, 06830 Ankara, Turkey
[b] Institute of Informatics, Hacettepe University, 06800 Ankara, Turkey

ARTICLE INFO

Keywords:
Weather forecast
Meteorology
Cloud coverage
Cloudiness
Cloudage
CNN

ABSTRACT

In recent years, the accessibility of weather forecasts has reached a point that checking it up on a smart device, like a smartphone, only takes a few seconds. Despite easy accessibility, false predictions of weather forecasts are commonly experienced, which this situation has yet to be put right. In meteorology institutions where this situation arises from, there is a need for many personnel working both on the fields and in the institution to make weather predictions as accurate as possible. As in all human-based systems, human mistakes constantly occur in weather forecasting systems. If human factor was to be minimized in weather forecasting systems, likewise human fallibility would diminish. The most feasible way to deal with this problem is to take advantage of the deep learning techniques, the pinnacle of modern software technologies, which requires almost no human effort on the domain they work, once developed. The deep learning methods have the capability of classifying big image datasets with image processing. With this feature and using the cloud pictures taken from the ground, they can be classified as clear/cloudy and the weather cloudiness can be determined as a numerical ratio. This study aims to reduce human-induced meteorology errors as much as possible with the application of deep learning techniques. The dataset that was preferred contains cloud pictures taken from the ground which are classified as either clear or cloudy. In order to compare different deep learning architectures and their efficiency on this subject, four particular pretrained models were selected. Among the models based on MobileNet V2, VGG-16, ResNet-152 V2, DenseNet-201: VGG-16 came as the best in terms of accuracy with 91.4%. In the future, it can be foreseen that all weather forecasting systems will prefer making their predictions based on modern artificial technologies like deep learning.

1. Introduction

Weather forecasting consists of three phases: observation, analysis and prediction. In these phases, various factors are taken into account, such as atmosphere pressure, air temperature, humidity, wind speed and cloud coverage. An important point among these factors is the cloud coverage, namely the mass of clouds that cover and obscure the sky also known as cloudage, cloud amount or cloudiness. The common practice in the traditional meteorology of utilization of the cloud coverage is at the observation phase sending personnel to the field and having them to take pictures from the ground, at the analysis phase having expert personnel in the

☆ This paper is for regular issues of CAEE. Reviews processed and recommended for publication to the Editor-in-Chief by Guest Editor Sunder Ali.

* Corresponding author.
E-mail address: muruvvetates@gmail.com (M. Kalkan).

https://doi.org/10.1016/j.compeleceng.2022.108271
Received 4 June 2022; Received in revised form 7 July 2022; Accepted 23 July 2022
Available online 6 August 2022

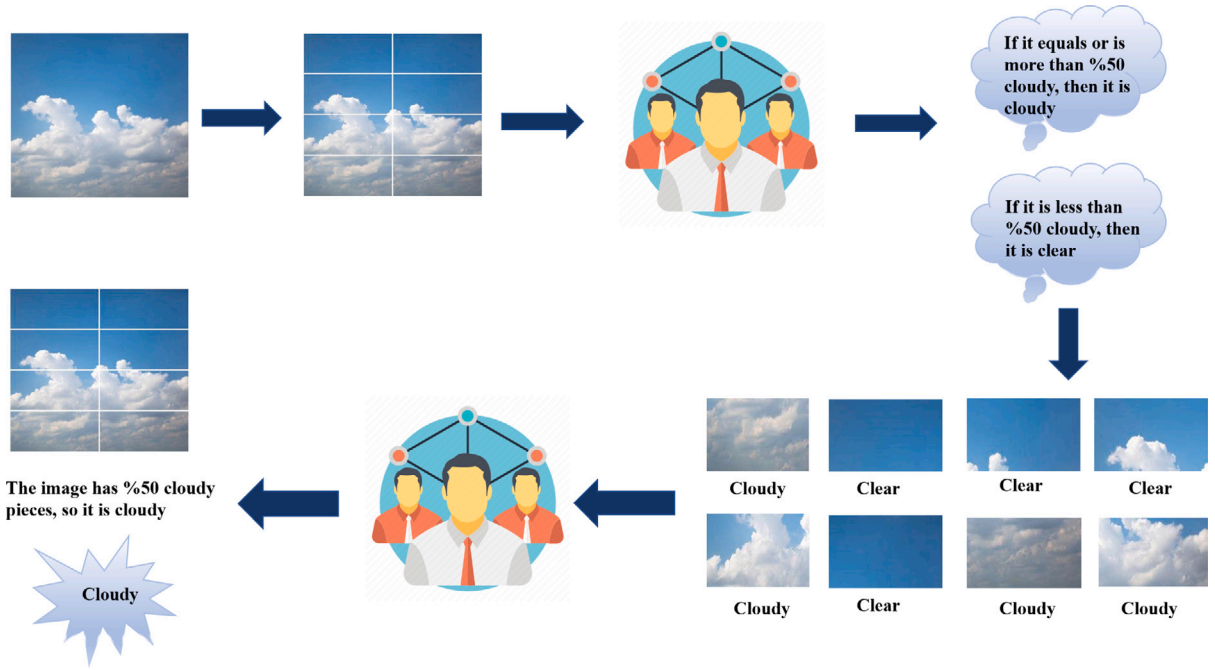

Fig. 1. Determining cloudiness in traditional meteorology.

stations to analyze the images taken during observation phase and finally at the prediction phase calculating the desired outcome as a weather forecast by making use of all other previously mentioned data including cloudage. Globally, it is acknowledged that a traditional meteorology institution can provide a forecast with different approximate accuracy for certain periods: 90 percent for a five days period, 80 percent for a seven days period and 50 percent for a ten days period. Although the equipment and methods of analysis and prediction involve advanced technologies and techniques, the final outcome is still far from near-zero errors, especially for longer periods. In addition, human based systems produce more errors than usual when people are affected by special circumstances, like in COVID-19 pandemic. During the pandemic, since both field workers and experts at stations of meteorology do not go to work regularly, weather forecasting took a hit and consequently meteorology errors spiked [1].

The method of determining manual cloud coverage in meteorology stations can vary depending on the country but in Turkey it is done as follows. First, observers on the field send cloud pictures taken from the ground, "from the ground" means not from satellites and "the ground" can be observation towers or balloons, to the stations. Next, collected pictures are sliced into eight equal pieces and sent to examination by human experts, each piece is considered independently. There are three possible outcomes for each piece: if experts decide that the sky part of the picture consists of clouds in a 50% rate or more, then it is cloudy, if not then it is clear, but if in a piece of picture there is no sky parts and it consists of irrelevant data to weather forecasting, it is considered an image with high noise and taken out of process. After determining the cloudiness of each piece, the non-noise pieces are reunited into their original picture and if among the pieces of the merged picture the cloudy pieces make up 50% or more of the all pieces, then that picture is classified as cloudy, if not then it is determined as clear. This flow is given in Fig. 1.

Rather than gathering daily or weekly image data and analyzing it with human experts, training a big dataset while regularly increasing its volume and processing it via a well-founded deep learning architecture would offer a modern perspective and boost the accuracy of predictions at such a rate that cannot be underestimated. The aid of big data and artificial intelligence (AI) technologies transform the human effort that requires expertise into a calculation job done by computers [2].

Deep learning based systems have the skill to tag a newcomer data with its learned experience from training itself on related big data. Deep learning, the evolution of machine learning, consists of multiple layers which train themselves by predicting on and processing a dataset. While the terms deep learning, machine learning and AI have different definitions, they are subgroups of each other, namely machine learning is an approach of AI, deep learning can be explained as a subset of machine learning with its unique techniques and architecture [3].

In this research, cloud pictures taken from the ground were divided into 4 equal pieces as in a grid and each single piece was classified as either clear or cloudy, while pieces which have a serious amount of undesired noise are deemed as unclassifiable and taken out of the dataset. After this operation, the dataset, with the intention of being trained and tested in a classification system based on deep learning, becomes the input for four distinct pretrained model based architectures: MobileNet V2, VGG-16, ResNet-152 V2, DenseNet-201.

This experiment will be examined in the next sections of this paper as follows: The literature review section presents the previous academic works related to the domain, techniques that either inspire or is similar to the proposed method and the deep learning

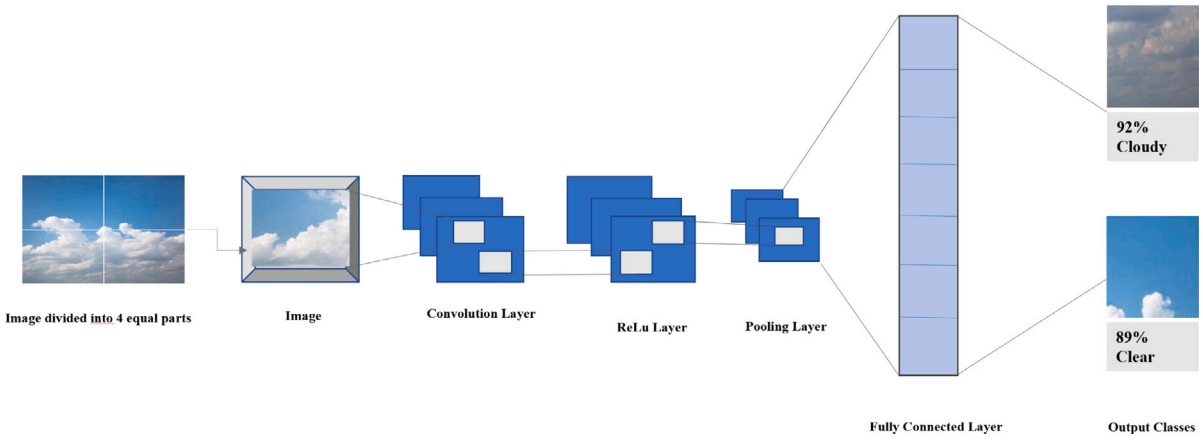

Fig. 2. Diagram of CNN.

architectures used in this research; the proposed method section displays the structure of this experimental software in a detailed way; in the Experimental Results section the outcome of the study is demonstrated and discussed; the conclusion section wraps all the academic work up.

The literature proves that DL based architectures, especially CNN based models, generate higher prediction accuracy for image based classification problems [4]. However, CNN based architectures have not been applied for estimating cloudage rate. This motivates authors to apply deep learning based architectures for both predictions of weather forecasts and cloudage rate.

2. Literature review

In recent years, the AI technologies, while getting progressively advanced, are being applied on various domains. Correspondingly, different approaches of AI technologies are either tested or already adopted by some countries in the meteorology field. A precedent of practicing deep CNN, as a specialized AI technique, weather forecasts is the work of Jonathan A. Weyn et al. They employed CNN to predict basic atmospheric factors globally. Their research consists of collecting the temperature values of 2 meters above the ground all over the world, then transforming it, a spheric data, into a cube then handling it via CNN [2]. An inclusive method of implementing deep CNN for weather forecasts is proposed with more parameters like temperature, pressure, wind etc. The researchers of that study examine and compare numerical, hybrid and end-to-end DL solutions to weather forecasts and introduce an end-to-end DL workflow [5]. Another research, which is closer to the approach of this academic work but differs in the focus of classification, is the classification of the cloud types by cloud photos taken from the ground with image processing and deep CNN, named CloudNet [6].

For employing deep learning techniques with image processing there are diverse methods. The experiment evaluated by this paper can be differentiated with its steps and the organization of these steps are as follows. First of all, a model is constructed from layers. In the early layers as input layers loaded images will be put into preprocessing, which includes images' data augmentation, conversion to numeric 3D matrices. In the following layers as convolutional layers, a picked pretrained model extracts features, and a pooling layer computes the prediction vectors. In the last layers as output layers, the prediction layer with an activation function outputs the final predictions as single numbers from prediction vectors. After the model is formed it is trained twice with different configurations for the DL technique *fine tuning*. This application involving the usage of pretrained models is named *transfer learning* and this brand-new methodology is profoundly common in new academic works associated with deep CNN and image processing. The four pretrained CNN architectures that are selected in this study are: MobileNet V2, VGG-16, ResNet-152 V2 and DenseNet-201.

Image classification is the course of calculating then predicting the class of each image in a big multi-classed image set. Image classification can be carried out with diversified approaches. The most common practice for image classification proposes an architecture composed of neural layers and it is named *Convolutional Neural Networks* (CNN). Artificial neural networks, the infrastructure of CNNs, aim to achieve pattern recognition by gaining inspirations from and even imitating the neural structure, which is composed of neurons, of human and animal brains [7]. In order to imitate a natural brain, it is essential to create artificial neurons and connect them with each other for transferring data. CNNs focus on seeking and finding features of classes which are hidden and almost always incomprehensible to humans. These features play a distinguished role for classifying images, as they can be common features of a specific class or a differentiating feature for multiple classes. With features extracted, CNNs produce a prediction of classification on the input images and score themselves by their accuracy. Thus, each time they calculate they correct themselves a bit more, all of it is, as the name of this process suggests, a training.

CNNs extract features, then produce predictions, and they do it with a structure consisting of specialized layers which are convolutional, pooling and fully connected layers, shown in Fig. 2.

Convolution Layer: By hovering a filter over the input matrices, initially pixels of the input images, the features are extracted. The training of the architecture is mainly done on these types of layers.

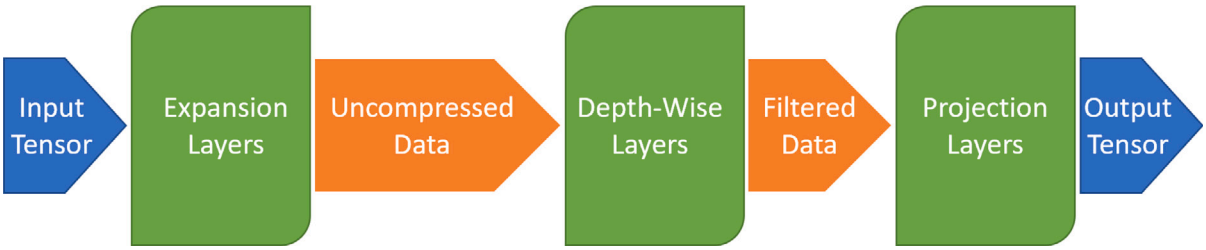

Fig. 3. The flow of MobileNets.

Non-Linearity Layer: This layer introduces non-linearity to the system. These types of layers also apply non-linear activation functions.

Pooling Layer: In pooling layers, there is also hovering of a filter over the matrices, like convolutional layers. However, they filter with pooling techniques like maximum or minimum pooling, which is taking the biggest or smallest, respectively, number in the pool, or average pooling, which is taking the arithmetic average of the numbers in the pool. After a pooling layer, input matrices become smaller and important features are extracted.

Flattening Layer: Flattening layers convert feature matrices to prediction vectors. They flatten multi sized matrices to one dimensional vectors. Thus, the inputs of the fully connected layers are prepared after flattening layers.

Fully Connected Layer: After multiple convolutional, multiple pooling layers and, often, a single flattening layer, the prediction vectors are sent to fully connected layers. The input prediction vector for a single image computed into a single prediction result in fully connected layers. The final predictions for image classifications are calculated here. They mostly use activation functions at the end, such as *sigmoid*, *softmax* etc.

The term *fine tuning* is commonly accepted as making adjustments in a CNN for achieving improved results during and after the training operation. In the science world, "fine tuning" originated from theoretical physics and can be described as adjusting the parameters of a designed system in order to make it produce results that can fit with certain observed results. Likewise in deep learning, the "fine tuning" term means altering the parameters like weights and biases for more desired results. There are numerous ways to fine tune a CNN, but the *freezing out* method is applied in the experiment of this academic work. Freezing a layer out of training means that the layer is not trainable, thus its weights do not change during a training, unfreezing a layer means that an untrainable layer becomes trainable again. It is observed that freezing certain layers out of training can actually yield superior outcomes [8]. There are various answers to give to which layers should be frozen out in order to produce the best results, for example one application could be training the batch normalization layers and freezing out convolutional layers, since weights of batch normalization layers are based on the pre-trained model's original training and they may not be generalized for transfer learning [9,10]. Freezing out technique along with transfer learning implemented in this experiment is as follows. First, all of the layers of the pre-trained model are frozen, then the actual model built with the base pre-trained model is trained. Once the training is completed, except for the earlier layers, approximately the first 2/3 of the all layers, the layers of the pre-trained model are unfrozen. After the unfreezing, the learning rate is slowed down to ten percent of its previous value, then the model is trained for a second time. The effects of fine tuning with layer freezing–unfreezing are displayed and discussed in the Experimental Results section.

MobileNet, developed by Google, is one of the pre-trained models of the experiment. It was trained with the famous dataset ImageNet that consists of 14 million images and 1000 classes. This network is particularly developed for, as the name suggests, mobile and embedded vision applications. The first version of it, MobileNet V, 1 presented a depth-wise convolution, meaning in contrast to standard convolution where input channels are mixed and connected to an equally deep filter layer, in the depth-wise convolution each input channel has a filter layer and they are separated from each other [11]. Currently, in the second version, MobileNet V2, there is a new system of blocks of layers where each block has expansion, depth-wise filtering and projection layers respectively where the data in turn uncompressed, filtered and compressed back, as given in Fig. 3. In the version "mobilenetv2 _1.00_160" of the network, which is practiced in this research, the layer structure starts with an input layer, followed by a standard 1×1 convolution layer group (Conv, Batch Normalization, ReLu). Then, there is one of each depth-wise and projection layer groups. Next, there are 16 blocks of aforementioned expansion-depthwise filter-projection layers in the middle and again one regular 1×1 convolution layer group with activation function as *softmax* at the end.

VGG models were first developed by K. Simonyan and A. Zisserman from Oxford University. They are also trained with ImageNet and their best accuracy was measured as 92.7 percent in ILSVRC-2014 (ImageNet Large Scale Visual Recognition Competition) in which VGGs came in first place in the localization field and second place in classification field. VGGs have a CNN structure with the aim of investigating the effect of the network depth on accuracy. In large scale image and video recognition, the Visual Geometry Group (VGG) researchers observed that by incrementing depth of the network with very small convolutional layers the accuracy of the model increased significantly [12]. Furthermore, VGG models were developed with different weight layers configurations. There are 11, 13, 16 or 19 weight layers options and 11 and 16 weight layers configurations have also two different settings. In the experiment inspected in this paper, VGG-16 which has 16 weight layers, as the name suggests, is picked and operated with. VGG-16 has a layer structure in order of one input layer, five varying blocks of two or three convolutional layers with the same size and

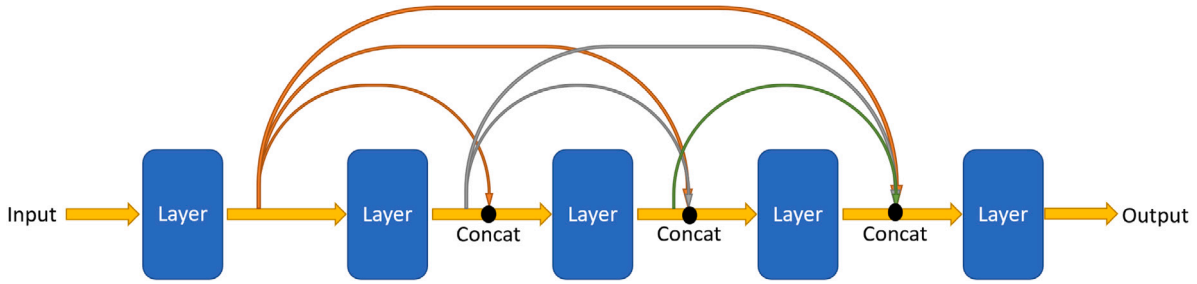

Fig. 4. Flow of a *dense block*.

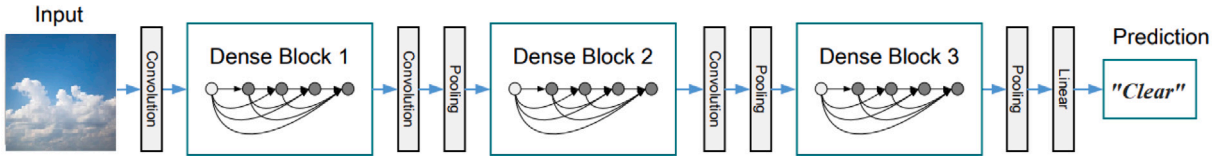

Fig. 5. DenseNets layer structure [14].

one max pooling layer, while each block having incrementing convolutional layer sizes, except the last block, three fully-connected layers and the softmax activation function at the end.

The ResNet (Residual Network) architecture was first propounded by Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun from Microsoft. It came first in the classification task of ILSVRC 2015. When a CNN gets deeper, it gets harder to train. However, researchers from Microsoft found a solution for this problem: residual mapping. ResNets have layers similar to VGG but they could go as deep as 152 layers while they are letting layers to fit a residual mapping rather than a basic mapping. As a result, the complexity of ResNets becomes even smaller than VGG networks, even though they could be almost eight times deeper [13]. There are ResNets with different depths, which are 18, 34, 50, 101, 152, and the 152 layered depth is chosen in this experiment, meaning ResNet-152 V2. ResNet-152 V2 has a deeper layer structure than the former two models and it consists of one input layer and 16 different blocks of layers. Each block has more than one convolution layer occasionally followed by changing batch normalization layers or activation functions and the last layer of the last block is the final activation function.

DenseNets, abbreviation of Dense Networks, were first proposed by Gao Huang, Zhuang Liu and Laurens van der Maaten. The academic paper of DenseNets, won the "Best Paper Award" at the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2017. With the same goal as ResNets, namely proper manageability of deeper CNNs, DenseNets offer a different approach, thus a solution to the problem. While ResNets transfer the information of backward propagation in each block with residual mapping as shown in Fig. 4. In each "dense block" of a DenseNet, layers concatenate the outputs of previous layers [14]. The layer architecture of the DenseNet is constructed as follows. Convolutional and pooling layers are added at the beginning and the end of the dense blocks, along with in between the dense blocks, lastly a fully connected layer with softmax activation function at the end is added to the model, Fig. 5. DenseNets have different architectures like 121, 169, 201 and 264, DenseNet-201 is picked in this academic work for transfer learning.

3. Methodology

The main goal of this study is to predict weather cloud coverage from an image by computing the probability of belonging to the cloudy class. For instance, a cloud image that is in the cloudy class with probability of 57% can be estimated as 57% cloudy. By being able to predict cloudage weather forecasting would be much more protected from human mistakes. The method proposed in this paper is to compute these estimations with pre-trained model based models.

In terms of processing images and classifying them with machine learning algorithms or structures, there are differences between approaches. Every approach has advantages and disadvantages depending on the goal of the experiment, the input dataset of images and the approach itself. For the goal of this experiment and the input cloud images CNN architectures with transfer learning are the most fit as the approach. By having trained on a different dataset, like ImageNet that has 14 million images, and being capable of controlling layer structure, depth and breadth, the CNNs can produce quality predictions. Their only setback is being a little slower than majority of classical ML algorithms and artificial neural networks [15]. Therefore, CNN architecture with transfer learning is selected for this experiment. In order to reach a better result along with comparisons of pretrained models, four aforementioned models are selected.

Since different architectures are being put to the test for this task, they must be analyzed and compared with the same techniques. The flow of each model that is based on a different pre-trained model starts with building the main model that is composed of, in order, preprocessing/input layers, base model, global average pooling layer and prediction layer. The main model is trained

Table 1
The sizes of the datasets.

Class	Training	Validation
Clear	1612	977
Cloudy	1627	981

Table 2
The sizes of the datasets after partition of the validation dataset.

Class	Training	Validation	Test
Clear	1612	782	195
Cloudy	1627	785	196

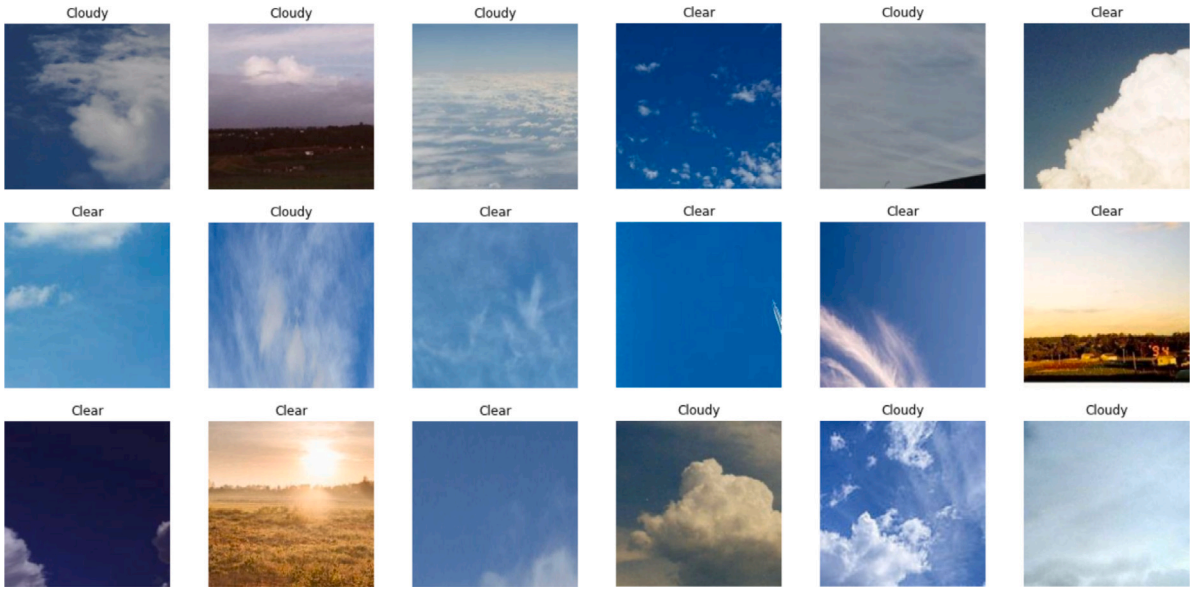

Fig. 6. Sample images from the training dataset.

twice according to fine tuning with freezing out. During training sessions there are epochs, thus it is possible to observe learning capabilities and extract learning curves of a model. After training, final results are produced from a test dataset. The train, validation and test datasets are essential for all these processes, and these three datasets are fragments of the source dataset. The source dataset is a wide and balanced dataset that consists of cloud images taken from the ground.

The dataset of the experiment, consisting of more than 1500 cloud images taken from the ground, was used as 5197 images by dividing each image into 4 equal parts. Fragments of images with high noise (images irrelevant to classification) were removed from the dataset. A separate program was written and run to split the pictures into parts. Then, 5197 cloud images were divided into two classes by hand as clear and cloudy with each class having its own folder, which some samples can be seen in Fig. 6. This process was picked, because this is the method that is similar to currently used in meteorology stations. Slicing pictures into 4 parts also contributes to the total volume of the dataset. The image fragments were divided into classes with the intent of training, validating and testing the models. In almost all machine learning techniques, the machine must learn, namely train itself, for deep learning networks there is also a need for validation steps, epochs, in training. Therefore, after division of the images to the classes, the distribution of Training–Validation datasets were done and it is as in Table 1.

Test datasets are mandatory for almost all DL models, since a model's efficiency on various factors like accuracy must be measured after training it. Thus, just before processing of the images, 20 percent of the validation dataset was chosen randomly and balanced and determined as the test dataset, Table 2.

To concretely examine the solution proposed in this study, distinct software mediums are required. The instruments of experimenting in this program are as follows. First, Python, which is one of the most used programming languages for practicing deep learning, is picked. Second, Google Colab platform which allows running Python code from a browser is preferred. Next, the Python libraries NumPy, PIL are chosen for their capability of array calculations and image processing, respectively. Finally, the TensorFlow software library which is one of the few essential options for building, configuring deep neural networks and expediting transfer learning is selected.

The study is based on transfer learning and for pre-trained models MobileNet V2, VGG-16, ResNet-152 V2 and DenseNet-201 are chosen. Each pre-trained model must be trained and tested independently from the others; thus, the experiment must be repeated four times with the same steps but with different pre-trained models. For a pre-trained model, which can be named **base model**, there are also additional layers both before and after them. The additional layers are the same for all experiments. By adding required extra layers to a base model, a **main model** is built. Training and testing are all done on main models. The equivalent steps of the experimental program that are followed for all four main models are given as pseudocode in Algorithms 1 and 2, also explanation of these steps are as follows.

Algorithm 1 Building the experimental model

```
procedure MODEL CONSTRUCTION
    Fetch the dataset
    Split them as training, test and validation datasets
    Approximately 61% training, 31% validation, 8% test

    baseModel ←get the pretrained model (MobileNet        V2, VGG-16, ResNet-152 V2, DenseNet-201)

    baseModel.trainable ← False

    model ← create the experimental model
    model.layers ←empty, initial value

    model.layers.insert(inputLayer)
    model.layers.insert(dataAugmentationLayer)
    model.layers.insert(preprocessLayer)
    model.layers.insert(baseModel)
    model.layers.insert(globalAveragePoolingLayer)
    model.layers.insert(predictionLayer)

    metrics ← [accuracy, loss, precision,
               recall, f1Score, roc]
    model.compile(metrics)
end procedure
```

Algorithm 2 Model training with fine tuning and evaluation

```
procedure MODEL TRAINING AND EVALUATION
    Train the model
    baseModel.trainable ← True
    fineTuneAt ← approximately 2/3 of the number of        layers in the model
    for k ∈ {0, ..., fineTuneAt} do
        baseModel.layers[k].trainable ← False
    end for
    metrics ← [accuracy, loss, precision,
               recall, f1Score, roc]
    model.compile(metrics)
    Train the model
    Draw the learning curves
    Evaluate model by the test dataset
end procedure
```

1. Training and validation datasets are loaded into the program.
2. The validation dataset is divided randomly, but balanced, into two smaller datasets with the sizes of 20 and 80 percent of the original. The 80% stays in the validation dataset, the rest, 20%, is moved to create a new dataset, the test dataset.
3. A number of configurations are applied to the datasets, mostly for performance purposes. Each image is set to be resized as 160–160 pixels. Batch size is selected as 32, a common ideal batch size.
4. In order to employ data augmentation, increasing data volume, a layer is created. The data augmentation layer of an image dataset rotates images with different angles to create new images with the same label, class as in Fig. 7. Thus, the size of the dataset rises significantly.

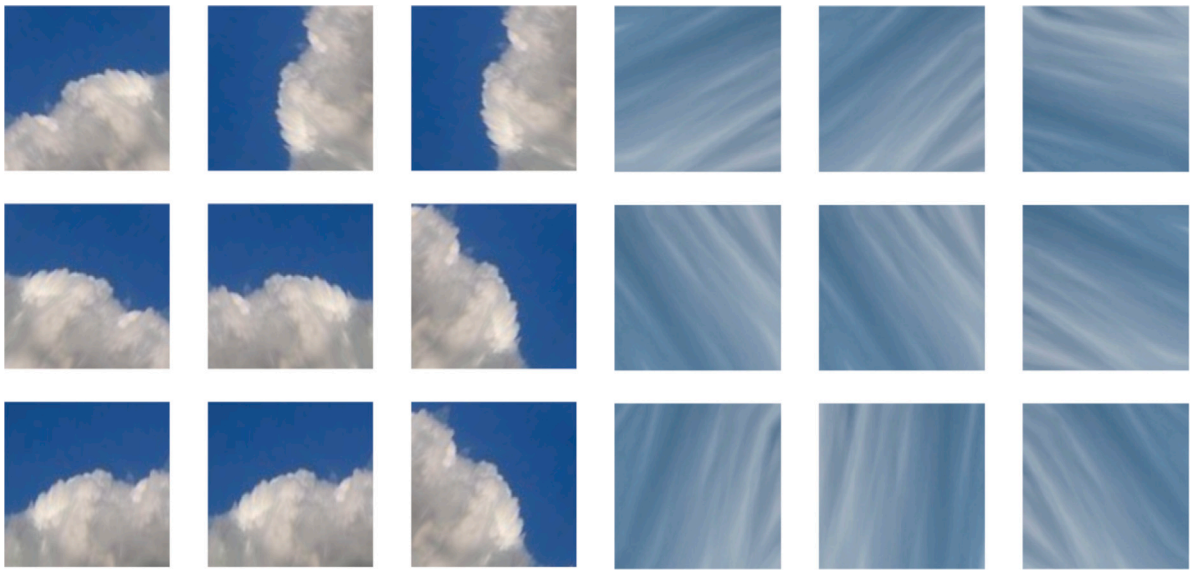

Fig. 7. Sample images from data augmentation by rotation.

5. The rescaling layer which aims to increase the time efficiency of the program is created. An image is composed of pixels as a 2D matrix and each pixel consists of three bytes, RGB (red, green, blue) color values. An RGB value is in the interval of [0, 255], but in big matrix computations high value limits like 255 causes slowdowns. Thus, each RGB value of each pixel of each picture must be rescaled to the interval of [−1, 1] to maximize the speed of calculations.
6. The pre-trained model is loaded into the program via TensorFlow and defined as a base model. (MobileNet V2, VGG-16, ResNet-152 V2 or DenseNet-201)
7. All layers of the base model are set to untrainable, namely frozen, to implement the freeze out fine tuning method.
8. For the outputs of the base model, a feature batch is prepared and used to create a global average pooling layer. This pooling layer has the functionality of getting the feature batches from the base model's output layer and pooling them into prediction vectors (1D matrices).
9. The producer of the final prediction number, the prediction layer, a dense layer, is created. The prediction layer takes the outputs of the previous pooling layer, prediction vectors, and computes the final prediction as a number. Since the classification of this program has two classes, namely binary classification, the activation function *sigmoid* is chosen in the prediction layer.
10. All the aforementioned layers are constructed together in the same order including the base model, thus the main model is created and compiled with a constant learning rate, main model layer structures are shown in Figs. 8, 9, 10, 11.
11. Metrics are set to accuracy, cross entropy loss, precision, recall, f1-score and area under the ROC curve.
12. With the metrics chosen and ten epochs are set for the first training is done with training and validation datasets.
13. After the training, results of each epoch are saved and used for an immediate visual presentation with the metrics. Thus, the first learning curves are drawn.
14. Fine tuning is employed on base model's near output layers, approximately one third of all layers. The near output layers of the base model become trainable again, namely unfrozen.
15. The main model is compiled again but this time the learning rate is dropped to ten percent of its former value. The reason for it is that there are new layers that would join a training and they must adapt to the previously trained layers.
16. The second training is done on the main model with the same given metrics and another ten epochs.
17. The results of each epoch of the second training are concatenated with the results of the first training.
18. Then, merged results are again displayed visually as learning curves.
19. To measure the final success of the model, the trained main model is evaluated on the test dataset, which has never been in any training, with the same metrics. Final results on the test dataset are displayed. These results are shown and discussed in the following section.
20. After measuring the final result of the main model, some tangible examples are displayed visually on randomly selected sample images. In this display, an image with its real class and predicted class by the program along with its calculated probability rate are shown, like in Fig. 14.

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 160, 160, 3)]	0
sequential (Sequential)	(None, 160, 160, 3)	0
tf.math.truediv (TFOpLambda)	(None, 160, 160, 3)	0
tf.math.subtract (TFOpLambda	(None, 160, 160, 3)	0
mobilenetv2_1.00_160 (Functi	(None, 5, 5, 1280)	2257984
global_average_pooling2d (Gl	(None, 1280)	0
dropout (Dropout)	(None, 1280)	0
dense (Dense)	(None, 1)	1281

Fig. 8. MobileNet V2 based main model.

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 160, 160, 3)]	0
sequential (Sequential)	(None, 160, 160, 3)	0
tf.__operators__.getitem (Sl	(None, 160, 160, 3)	0
tf.nn.bias_add (TFOpLambda)	(None, 160, 160, 3)	0
vgg16 (Functional)	(None, 5, 5, 512)	14714688
global_average_pooling2d (Gl	(None, 512)	0
dropout (Dropout)	(None, 512)	0
dense (Dense)	(None, 1)	513

Fig. 9. VGG-16 based main model.

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 160, 160, 3)]	0
sequential (Sequential)	(None, 160, 160, 3)	0
tf.math.truediv (TFOpLambda)	(None, 160, 160, 3)	0
tf.math.subtract (TFOpLambda	(None, 160, 160, 3)	0
resnet152v2 (Functional)	(None, 5, 5, 2048)	58331648
global_average_pooling2d (Gl	(None, 2048)	0
dropout (Dropout)	(None, 2048)	0
dense (Dense)	(None, 1)	2049

Fig. 10. ResNet-152 V2 based main model.

4. Experimental results

In this section the experiment's results are displayed and discussed. The experiment was run on Google Colab environment with its presented hardware. Though there can be different GPU models, NVIDIA Tesla K80 is allocated for this experiment by Google Colab. NVIDIA Tesla K80 has 12 GB of GDDR5 memory and 240 GB/s memory bandwidth. In time efficiency, the models differ in runtime for the same dataset. Six evaluation metrics are preferred for examining and comparing the results. The metrics that are chosen for comparisons are: accuracy, precision, recall, F1-score, the area under the receiver operating characteristic (ROC) curve and cross entropy loss. All metrics except cross entropy loss are calculated from positive and negative prediction values. In a prediction system where a prediction set aims to include all positives and exclude all negatives, the terms for these prediction values are (Fig. 12):

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 160, 160, 3)]	0
sequential (Sequential)	(None, 160, 160, 3)	0
tf.math.truediv (TFOpLambda)	(None, 160, 160, 3)	0
tf.nn.bias_add (TFOpLambda)	(None, 160, 160, 3)	0
tf.math.truediv_1 (TFOpLambd	(None, 160, 160, 3)	0
densenet201 (Functional)	(None, 5, 5, 1920)	18321984
global_average_pooling2d (Gl	(None, 1920)	0
dropout (Dropout)	(None, 1920)	0
dense (Dense)	(None, 1)	1921

Fig. 11. DenseNet-201 based main model.

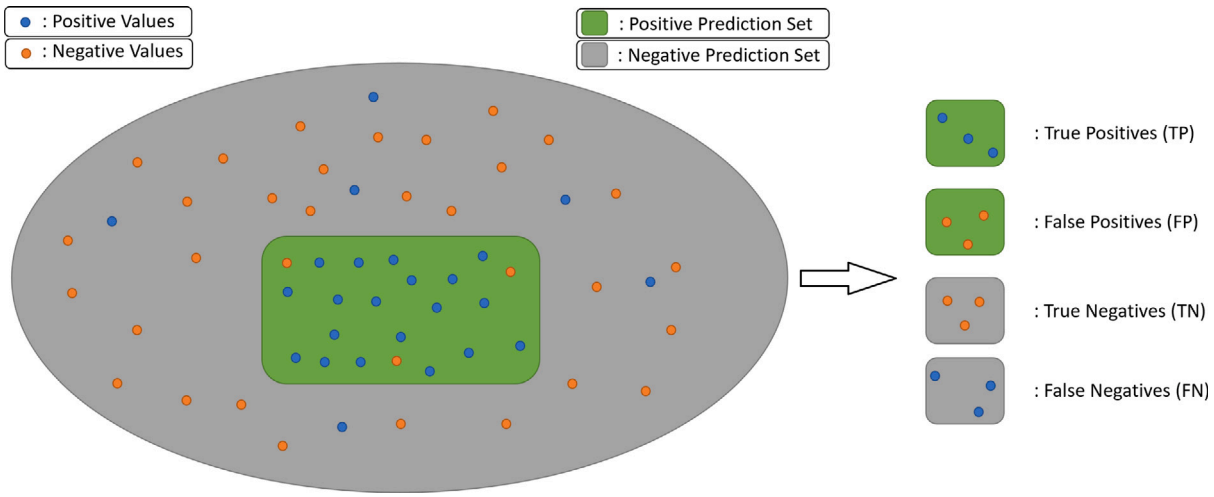

Fig. 12. Prediction value definitions.

- **TP:** Number of correctly predicted positive values, *True Positives*
- **TN:** Number of correctly predicted negative values, *True Negatives*
- **FP:**Number of incorrectly predicted as positive, but actually negative values, *False Positives*
- **FN:** Number of incorrectly predicted negative, but actually positive values, *False Negatives*

The accuracy is probably the most common and accepted measurement of success for both machine learning algorithms and deep neural networks. The more accurate a model, the more successful it is. Mathematically it is the rate of all true predictions over all values and shown in Eq. (1).

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{1}$$

Precision focuses on predictions of positive values. It is the rate of correctly predicted positives over all predicted positives and given in Eq. (2).

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

Recall also focuses on the positives, but it is the ratio between true positives and all actual positives and displayed in Eq. (3).

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

Recall and precision are contrasting measurements so there is often a need for compromise. It depends on the requirement of the domain. There might be cases where a high precision-low recall is desirable, for instance a product's being advertisable to the users is being predicted in a program, where a binary classification is conducted, then the advertiser would not want to waste resources on "not-likely" customers, so a high precision becomes mandatory. However, the opposite, low precision-high recall, can

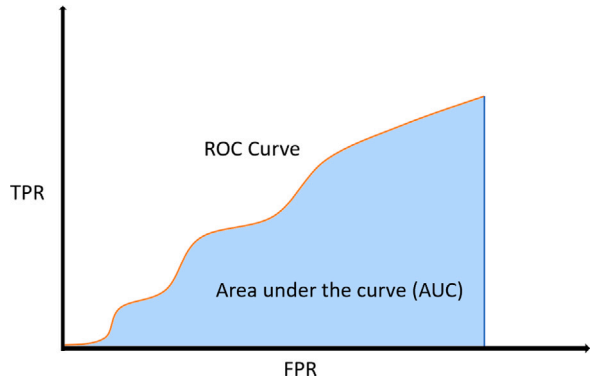

Fig. 13. Area under the curve of receiver operating characteristic (ROC AUC).

also be possible for different tasks, for example a program that detects people with serious illnesses by classification, requires high recall because the owners of the program would not want to miss a person under potential danger, namely all actual positives are important. Nevertheless, a balanced precision–recall could also be preferred, like in the program of this experiment.

To measure balance between precision and recall, the F-Score metrics are calculated. The general mathematical formula of all F-Scores is in Eq. (4).

$$F_\beta = (1+\beta^2) \times \frac{precision \times recall}{(\beta^2 \times precision) + recall} \tag{4}$$

In binary classifications F1 score must be applied, so it becomes a harmonic mean of precision and recall as in Eqs. (5) and (6).

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{5}$$

$$F_1 = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \tag{6}$$

With F1 score the success of a model becomes measurable for precision–recall balance, the higher it is, the higher quality results will be produced.

Along with F1 score there is another metric to measure precision–recall balance and that is area under the receiver operating characteristic (ROC) curve. TP rate (TPR, Eq. (7)) as *y*-axis, FP rate (FPR, Eq. (8)) as *x*-axis a curve is drawn in the 2D space, which is ROC curve (Fig. 13). Furthermore, rather than the curve itself, the area under the curve (AUC) is the significant part. AUC of ROC determines the skill of a model, the bigger it is the more skilled a model at getting true positives rather than false positives.

$$TPR = \frac{TP}{TP + FN} \tag{7}$$

$$FPR = \frac{FP}{FP + FN} \tag{8}$$

The cross entropy loss is the representation of distance between probability distributions in a prediction system. Since the program does a binary classification, it becomes binary cross entropy loss. If an entity that belongs to a class and its calculated probability of belonging to that class is *p*, then the cross entropy loss of the binary prediction system would be *-log(p)*. Also, a model is considered better in contrast to others, if it has less cross entropy loss.

After fully training a model by training and validation datasets, it becomes ready to generate predictions and be evaluated by the test dataset. Therefore, different models can be compared against each other with evaluation. As for produced prediction results, a number is produced for each image and this number points to the image's predicted probability of belonging to a class. Furthermore, one minus this probability gives its probability of belonging to the other class, because the classification is binary. This capability provides the opportunity to interpret the outputs of the program as cloud coverage on image. Some samples from these produced predicted probabilities are shown in Fig. 14.

Because the training is done with epochs and each epoch produces evaluation metric results, it is possible to see the learning process of the models by drawing learning curves, thanks to these metrics. There are both training and validation dataset evaluations by metrics.

From the learning curves of the MobileNet V2 based model given in Fig. 15, it can be seen that fine-tuning causes a positive or negative spike in most evaluations. In addition, validation dataset is affected much more than training dataset. However, the impact of fine tuning is rather minor than sharp in terms of increasing more refined results.

By the learning curves of the model based on VGG-16 in Fig. 16, it can clearly be seen that fine tuning played a major role for classification quality performance of the model in all aspects. Sharply raising accuracy, precision, recall, F1-score and AUC of ROC and dropping cross entropy loss. VGG-16 would not be able to produce such high quality results without fine tuning .

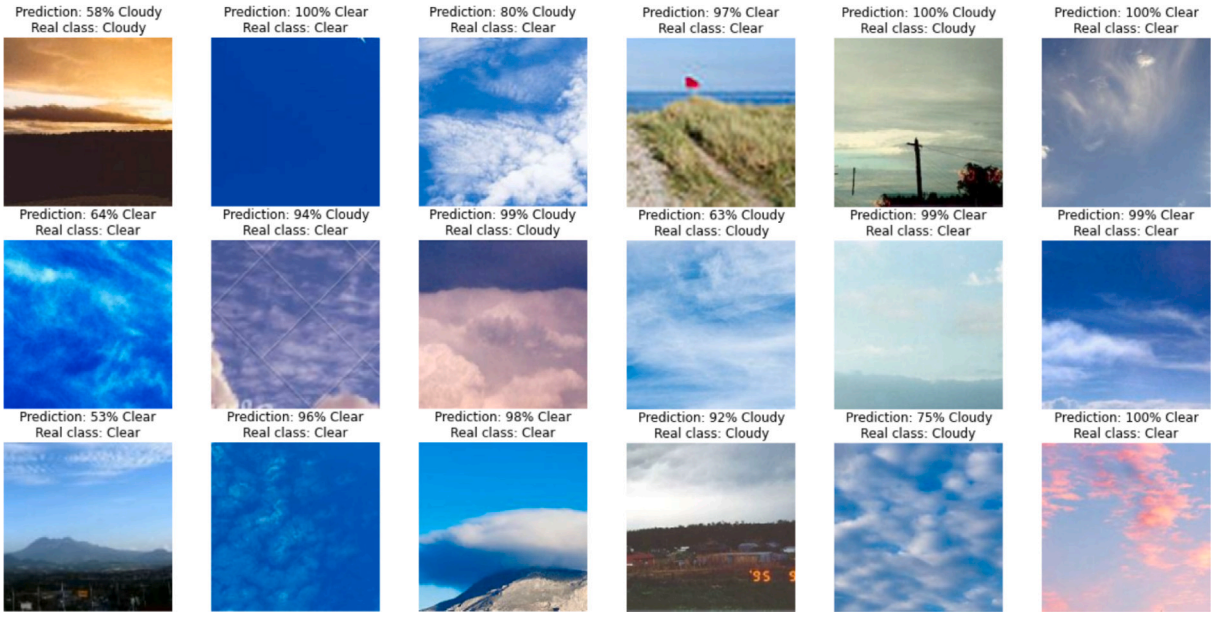

Fig. 14. Sample prediction results.

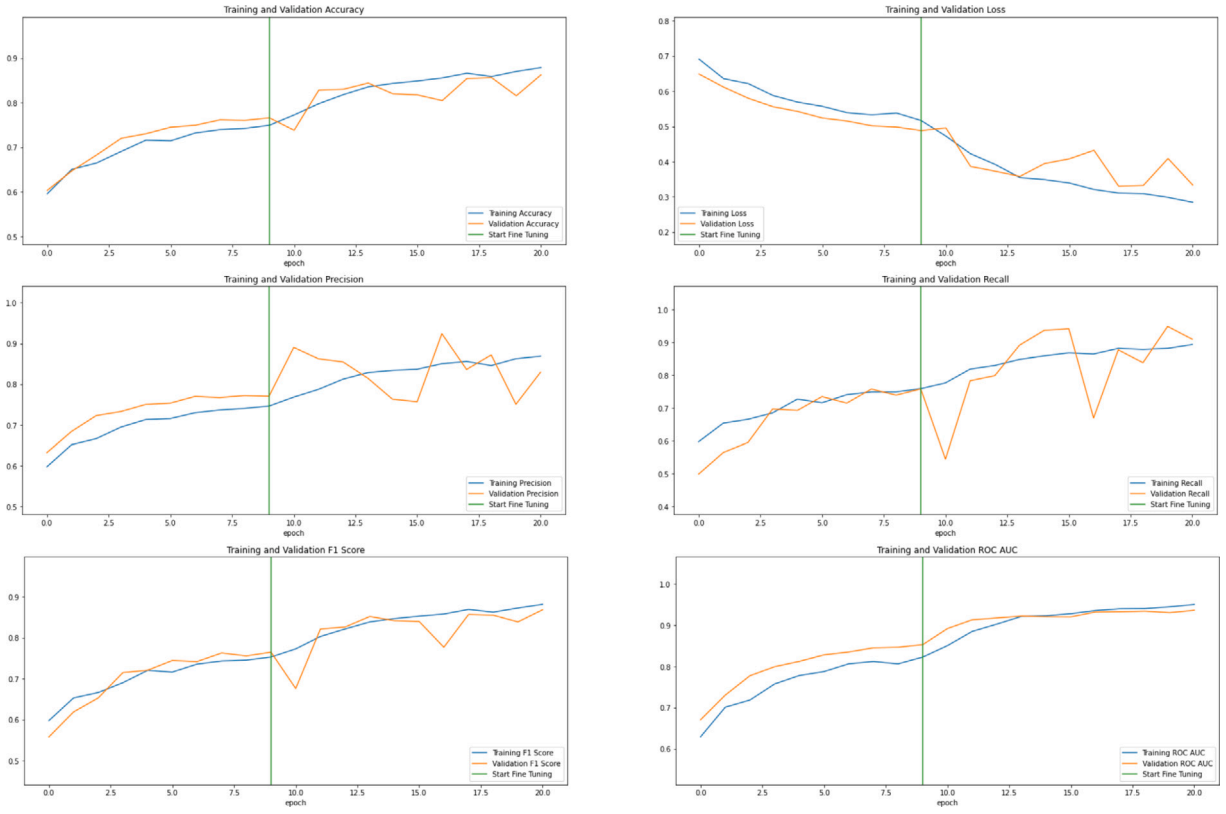

Fig. 15. Learning curves of MobileNet V2 based model.

ResNet-152 V2 based model's learning curves in Fig. 17 show that although fine tuning affected positively at first, the effect did not last and it is not evidently spiked. The evaluations on the training dataset are fine but the ones on the validation dataset are fluxional.

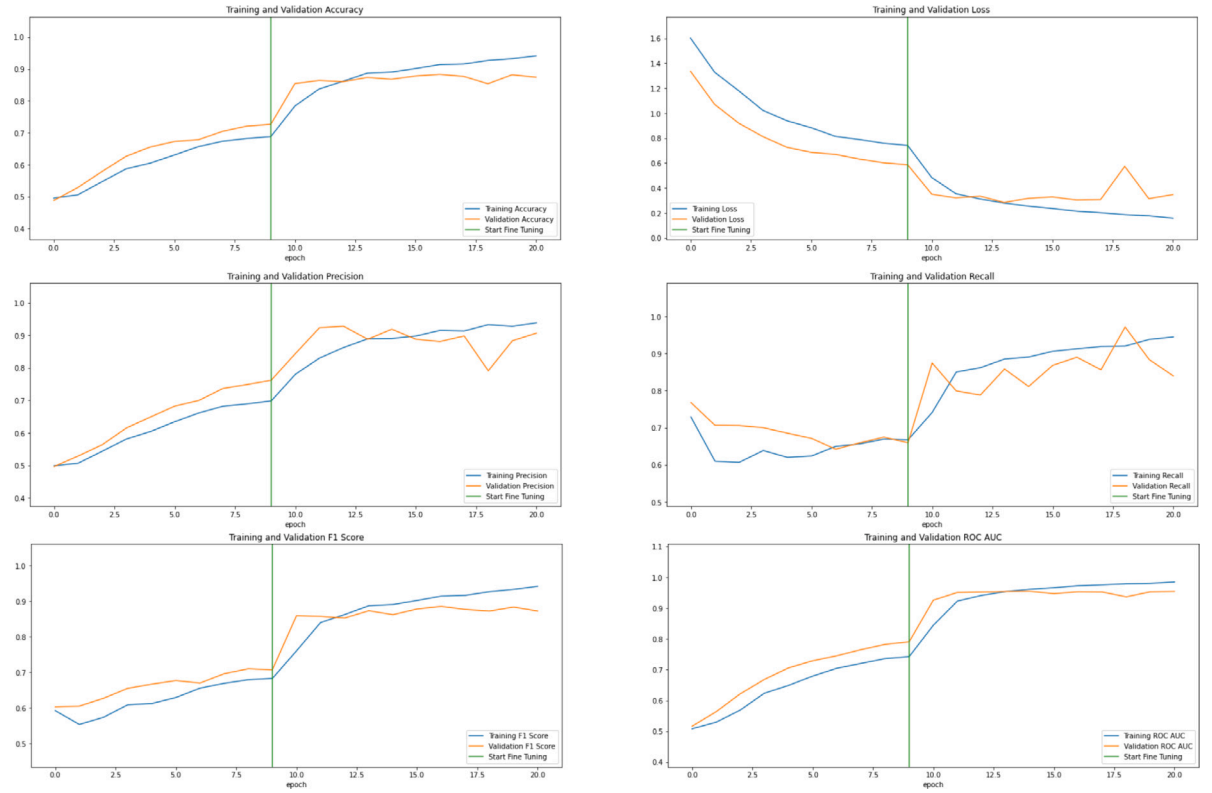

Fig. 16. Learning curves of VGG-16 based model.

Table 3
The evaluation of models on test dataset.

Metric	MobileNet V2	VGG-16	ResNet-152 V2	DenseNet-201
Accuracy (%)	88.8	**91.4**	89.58	90.89
Binary cross entropy loss	0.3	0.25	0.3	**0.2**
Precision (%)	84.68	92.86	91.26	**94.22**
Recall (%)	**95.43**	89.42	89.52	86.7
F1-Score (%)	89.74	**91.11**	90.38	90.3
AUC of ROC (%)	94.83	97.18	95.89	**97.77**

The learning curves of the DenseNet-201 based model in Fig. 18 display that fine tuning influenced the results in a better way, especially the ones on the training dataset. However, the evaluations on the validation dataset are not as stable as the others.

In addition, the final evaluation metric values of each model should be compared. Since the test dataset images were never exposed to models during trainings, they allow unbiased predictions to be produced. Thus, this final evaluation is done against the test dataset. These results can be seen on the Table 3 and graph in Fig. 19.

In terms of accuracy, while the worst model is based on MobileNet V2 with 88.8%, which is not far from ResNet-152 V2, the best model is based on VGG-16 with 91.4%, close to DenseNet-201. All binary cross entropy loss values are on the small interval of [0.2, 0.3] and DenseNet-201 has the smallest loss. Moreover, DenseNet-201 has the best precision overall. Since precision and recall are often inversely correlated, despite having the lowest precision MobileNet V2 has the highest recall. As for F1-score, VGG-16 has the best rate. The model based on DenseNet-201 has the biggest area under the receiver operating characteristic curve (AUC of ROC) and the near second one is based on VGG-16.

If all models are being appraised for their success, they also should be judged for their runtime performance. To statistically determine runtime performance of deep neural networks, clocking their training time should suffice. However, aforementioned there are two trainings, before and after fine tuning where unfreezing certain layers of pre-trained models occurs. These runtime durations are given in Table 4 and graph in Fig. 20. The order of models according to their time efficiency from the highest to the lowest is: MobileNet V2, VGG-16, DenseNet-201, ResNet-152 V2.

By accuracy the best model results come from the model based on VGG-16, and the close second is based on DenseNet-201. The most correct predictions are calculated in these two models. Binary cross entropy loss, distance between probability distributions is also the least in these two models, but DenseNet-201 has a lower loss, meaning the probability distributions are closer to each other compared to the other models.

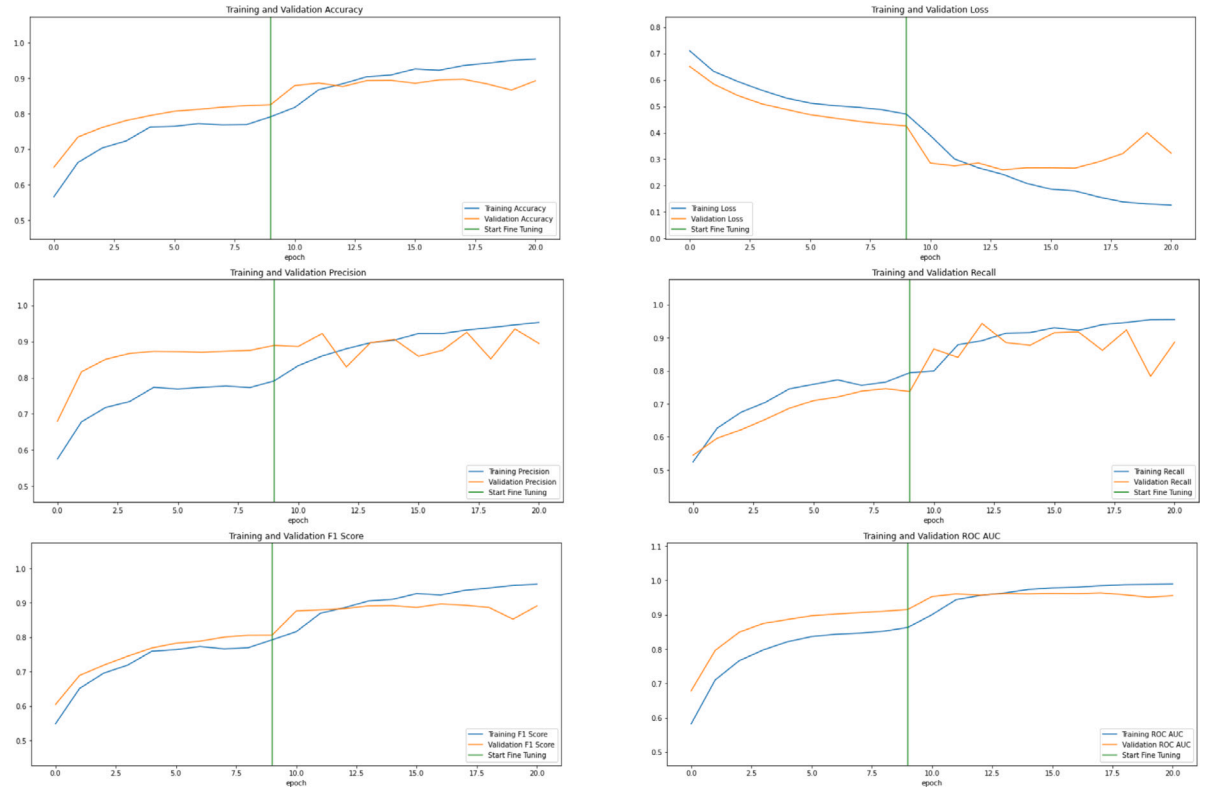

Fig. 17. Learning curves of ResNet-152 V2 based model.

Table 4
Training times of models.

Runtime	MobileNet V2	VGG-16	ResNet-152 V2	DenseNet-201
Time of training before fine tuning (s)	**446**	536	872	615
Time of training after fine tuning (s)	**179**	359	951	639
Total time of the trainings (s)	**625**	895	1823	1254

As for precision–recall, while the MobileNet V2 based model has higher recall-lower precision, for the other models precision over recall is clearly paramount. Also, DenseNet-201 has the highest precision among the models, while VGG-16 and ResNet-152 V2 have more balanced precision–recall values. If these results are considered, it can be deduced that MobileNet V2 based model predictions contain more true positives at the cost of containing more false positives, meaning it misses fewer positive values but guesses more negatives as positives incorrectly in its predictions. The model based on DenseNet-201 is doing the opposite of MobileNet V2 based one, it calculates the positive purest prediction sets, with the least incorrectly guessed negatives as positives but it misses the most positive values as well. In terms of F1-scores, the most quality predictions come from VGG-16, as in accuracy, ResNet-152 V2 being the second, but very closely followed by DenseNet-201 and lastly MobileNet V2 with lowest F1-score. By examining these F1-scores it is possible to reach some important points in the experiment. Firstly, even though the domain of the program does not prioritize precision or recall over each other, the models with more precision than recall have higher F1-score, but this is an observation from this experiment not a general rule. However, there can be no deduction that near precision–recall produces higher F1-score, since it is harmonic mean, and it can also be seen that VGG-16 has the highest among all, although it does not have the closest of the two values. Furthermore, the closest precision–recall results are generated from ResNet-152 V2, which is the second in F1-score, but it is still behind DenseNet-201 in terms of accuracy. The harmonic mean value F1-score does require a specific balance between contrasting precision and recall parity, and it affects accuracy but not directly, like in the example of ResNet-152 V2, better F1-score, and DenseNet-201, better accuracy.

Previously mentioned, AUC of ROC value represents a model's putting an actual positive value into the positive prediction set rather than the negative prediction set, also accepted as a measurement of the skill of a model. DenseNet-201 has the highest value among all, and it is followed by VGG-16, ResNet-152 V2 and MobileNet V2, respectively. Despite having the most skill on predicting a positive value DenseNet-201 fails to become a second to VGG-16, as it can be observed from precision–recall, DenseNet-201 have

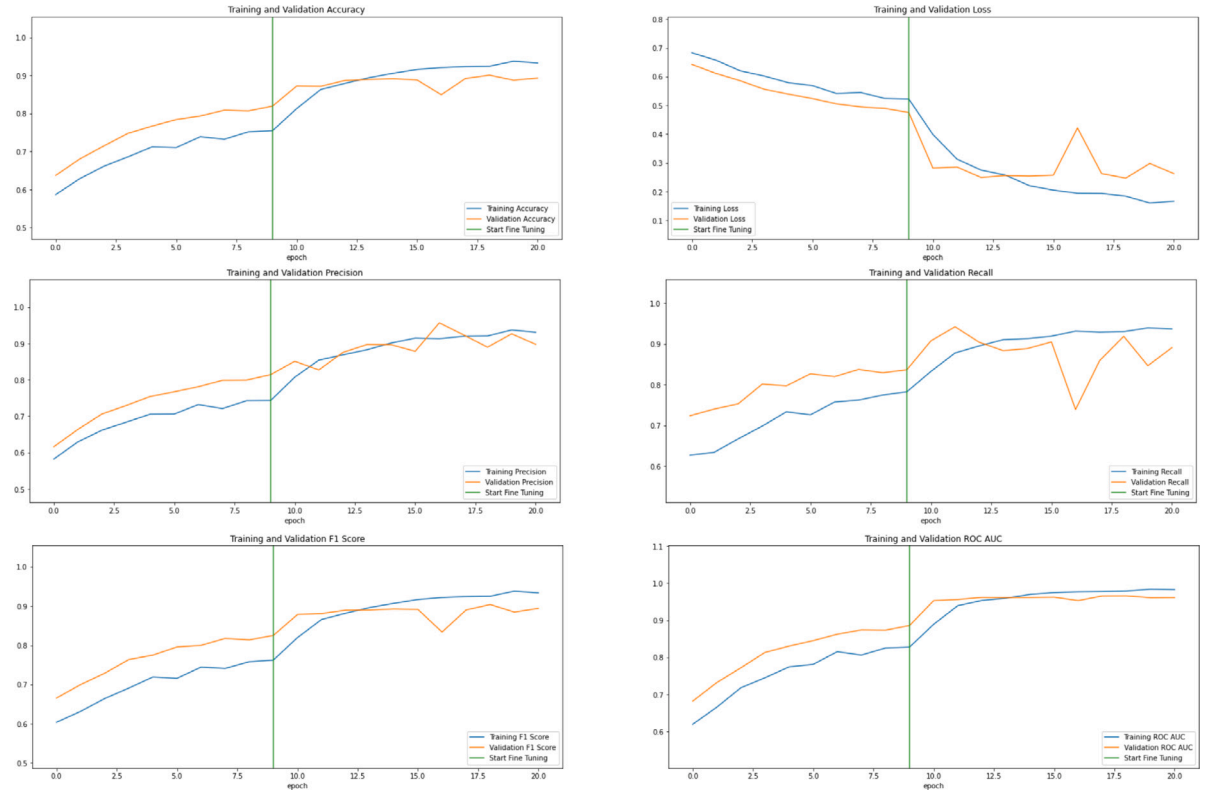

Fig. 18. Learning curves of DenseNet-201 based model.

a tendency to miss positive values more than VGG-16. Therefore, despite these qualities of DenseNet-201, best results are produced from VGG-16 overall and it is followed by DenseNet-201, ResNet-152 V2 and MobileNet V2, respectively. Also, if learning curves are considered, it is really apparent that the most efficient learner is VGG-16 compared to its peers.

Time efficiency should also be taken into consideration; however, CNN architectures do not have a time complexity dependent on the input size, namely they are not input-sensitive because of how they work. Just as some algorithms, such as convex hull algorithms like Chan's algorithm and Graham scan [16,17], are output-sensitive not input-sensitive, deep neural networks have a time complexity whose parameters are the aspects of its layers. These aspects are the number, type, order, depth or breadth of the layers. In contrast to the clear expression of the input-sensitive algorithms, like big O notation showing the relation between time complexity and input sizes, deep learning structures do not have a simple relation between the time complexity and its parameters. Despite the unavailability of time complexity expressions, it is viable to measure runtime of training the models. When transfer learning is applied, it is certain that the runtime of the models will change based on the core pretrained model. Thus, the runtimes of trainings are measured and model speeds are deducted in this study. The models have completed their trainings: MobileNet V2 in 625 s, VGG-16 in 895 s, DenseNet-201 in 1254 and ResNet-152 V2 in 1823 s. In other words, MobileNet V2 is the fastest while ResNet-152 V2 is the slowest.

5. Discussion

Experimental results in terms of all metrics show that VGG-16 is the best among all models. The reason for this is it has the highest accuracy and F1-Score, close second best loss and AUC of ROC. VGG-16 has deep architecture but its convolutional layers are very small, so this pattern allows it to get optimum results in all evaluations. Along with the most accurate predictions and it has the most balanced precision–recall by F1-Score, namely VGG-16 prediction set has the least combined compromise between false positives and false negatives. Furthermore, the second least binary cross entropy loss 0.25 proves that VGG-16 estimates do not go that far from actual results.

DenseNet-201 can be considered the second best model with the second highest accuracy, the best ROC AUC and loss and a close third F1-Score. DenseNet-201 is the deepest model of all, so its good quality results come from this feature. It handles this deepness with the concatenation method and, most likely, similar cloud pictures processed by this technique produce satisfactory but not the best results in this experiment. Nevertheless, DenseNet-201 has the closest estimation, least loss, and highest true positive false-positive rate, by AUC of ROC, but it lacks in precision–recall balance with the biggest precision and the least recall.

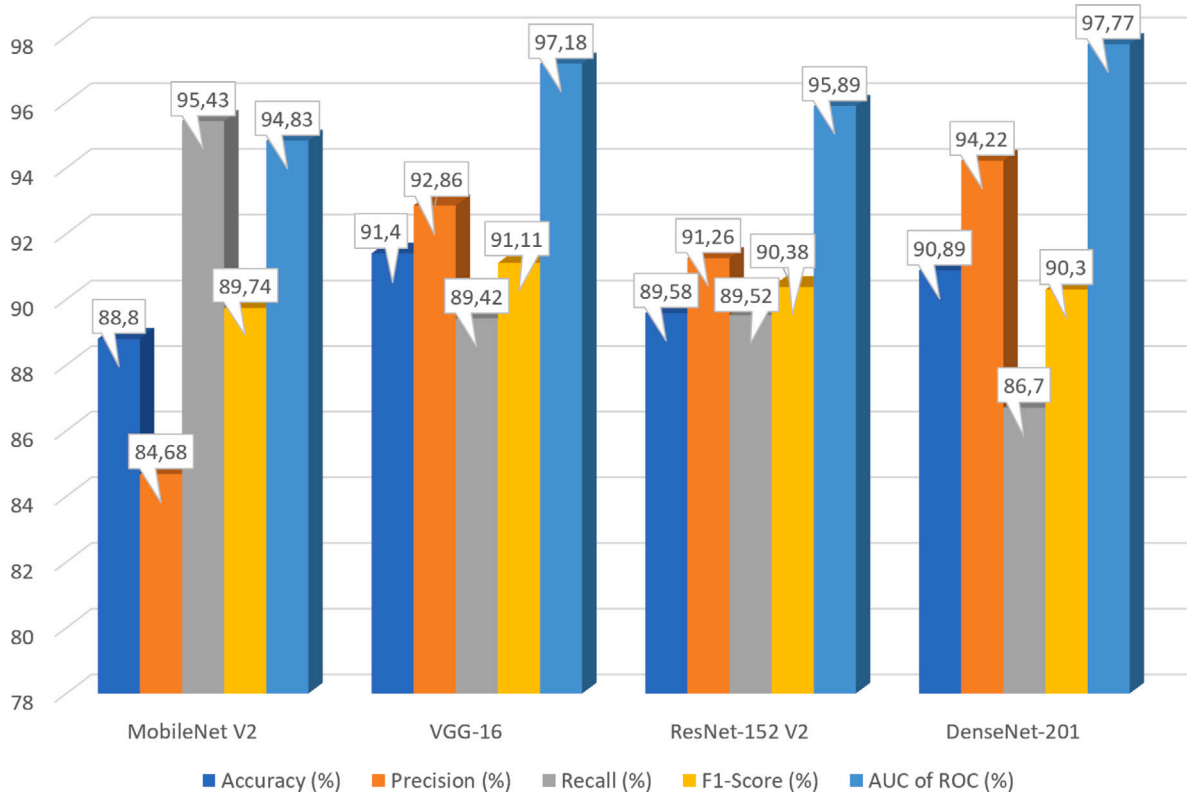

Fig. 19. Model evaluations on test dataset.

ResNet-152 V2 has results which can be thought as average among its peers. The third biggest accuracy and ROC AUC, a shared third place in loss and the second but close to the third F1-Score are yielded from this deep neural network. These results are lower than expected, because ResNets are deep structures, like VGG-16 and DenseNets, so this deepness should have had produced much better results than MobileNet V2 that is not a deep architecture. The most probable reason of this situation is that the method ResNets handle the setbacks of deepness which is residual mapping is not suitable for this task, since the dataset is composed of images resembling one other and contains close pixel values. Although the precision–recall balance is sufficient, in all other areas ResNet-152 V2 does not produce expected outcomes.

As it can be observed that the outcomes of the MobileNet V2 based model is the lowest of all. This model produced the smallest accuracy, F1-Score and ROC AUC and a shared highest loss. It has the highest recall at the cost of a compromise on precision, which is the smallest, consequently it is the least balanced model measured by F1-Score. Namely, this model has the most correct predictions of positive values and the most inaccurate predictions of negative values. Despite the lowest metric results, this was entirely expected of a MobileNet architecture, because it is the only "not too deep" CNN among all the models.

The model speed is in the order of: MobileNet V2, VGG-16, DenseNet-201, ResNet-152 V2. MobileNet V2 is the least deep structure, due to its emphasis on the layer breadth rather than the depth, so this contributed to its time efficiency. The other three are all very deep architectures with their own methods to handle the minuses of having too much depth. A point which can be interpreted as interesting is that although DenseNet-201 can be considered the deepest among the models, it has much better time efficiency than ResNet-152 V2. The reason for it is most probably the residual mapping method used by ResNets is not as effective as the concatenation of DenseNets in this task of classifying similar sky/cloud pictures. The final observable remark on time complexity is that VGG-16 which has a special pattern of very small convolutional layers with high depths is the second fastest model not too behind the first, thanks to its architecture.

If the training runtimes is considered together with the evaluation results, MobileNet V2 which has the poorest results among all makes up for it in training speed. ResNet-152 V2 is disappointing due to its worst time efficiency and undistinguished results among its peers. DenseNet-201 which can be thought as a bit slow, can produce promising results. Finally, VGG-16 along with the most accurate, quality and skillful predictions, is very fast, second to MobileNet V2, in terms of training speed. Therefore, the structure of VGG-16 is the most fit choice for this task of classifying cloud pictures which are not so different from one another.

If all these results are considered, then it can be seen the outcome is pretty decent and agreeable. The current methods used in some countries are completely done by human experts, so artificial intelligence makes a huge difference in accuracy with more than 90% rate. In that way the proposed method produced results as expected. Also, the deep learning models are proved to be

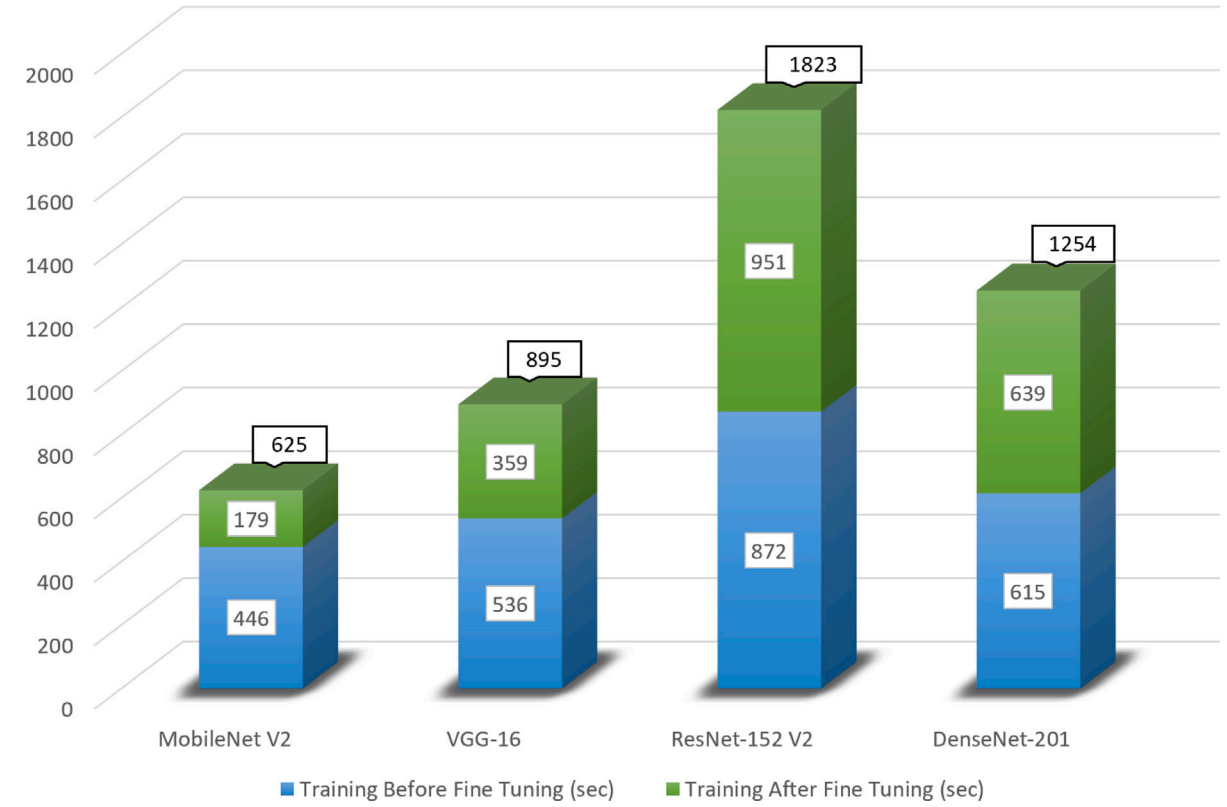

Fig. 20. Model training runtime durations.

applicable in the field by their observable benefits. A dataset of cloud images was given as input to the models and a classification of these images were estimated as outputs, which the tasks with these types of inputs are excellent for applying deep neural networks. Although the produced results are fine in contrast to human-based systems, they came short when compared to application of CNNs on different fields, since well refined CNNs can predict image classes with easily more than 95% accuracy, in datasets of similar sizes. If the test dataset was composed of millions of images, then evaluation results of the built models would plummet. This minus is caused by the size of the dataset, nearly 5000, which can be considered small in deep learning and it can be solved by increasing the size of the dataset. Currently, this is the biggest dataset available that could be used by this research. In the future, if a dataset with the size of millions were to be created, this setback of the proposed model also could be overcome. Finally, other than the dataset, different pretrained models could be chosen to test their accuracy on the field to further improve the proposed method.

6. Conclusion

Meteorological errors attract more attention than before and shake the confidence in weather forecasts, because people can easily access weather forecasts from their smart devices. As long as meteorological forecasts are human-based, error rates will be high. Especially in special circumstances, errors increase dramatically when people's participation decreases, like it happened in the COVID-19 pandemic [1]. Dependency on humans can only be eliminated with modern artificial intelligence technologies. In order to achieve it, the step taken by this study is aiming to detect cloud pictures taken from the ground as clear or cloudy and estimate their cloudiness by deep learning techniques. The proposed method is to build deep neural network architectures that have pretrained models as bases along with the preprocessing and output layers, then training these built models with applying the freezing out fine tuning method on certain number of layers in the architectures. Afterwards, each trained model is measured by the test dataset in terms of result quality metrics that are accuracy, precision, recall, F1 score, ROC AUC and the cross entropy loss. Consequently, this academic work displayed that within the proposed method, the best results are computed from the VGG-16 based model. Although the results of the program are mostly promising, there are also areas where it is lacking. The main reason for these shortcomings is thought to be the inadequate number of images in the dataset. It is recommended for future studies that the same processes should be done with much bigger datasets. Also in the future experiments, all can be done with modifying fine-tuning of the proposed method or changing the pretrained models. Finally, other key factors on meteorology, like the classification of cloud types can be studied in a similar fashion in the future.

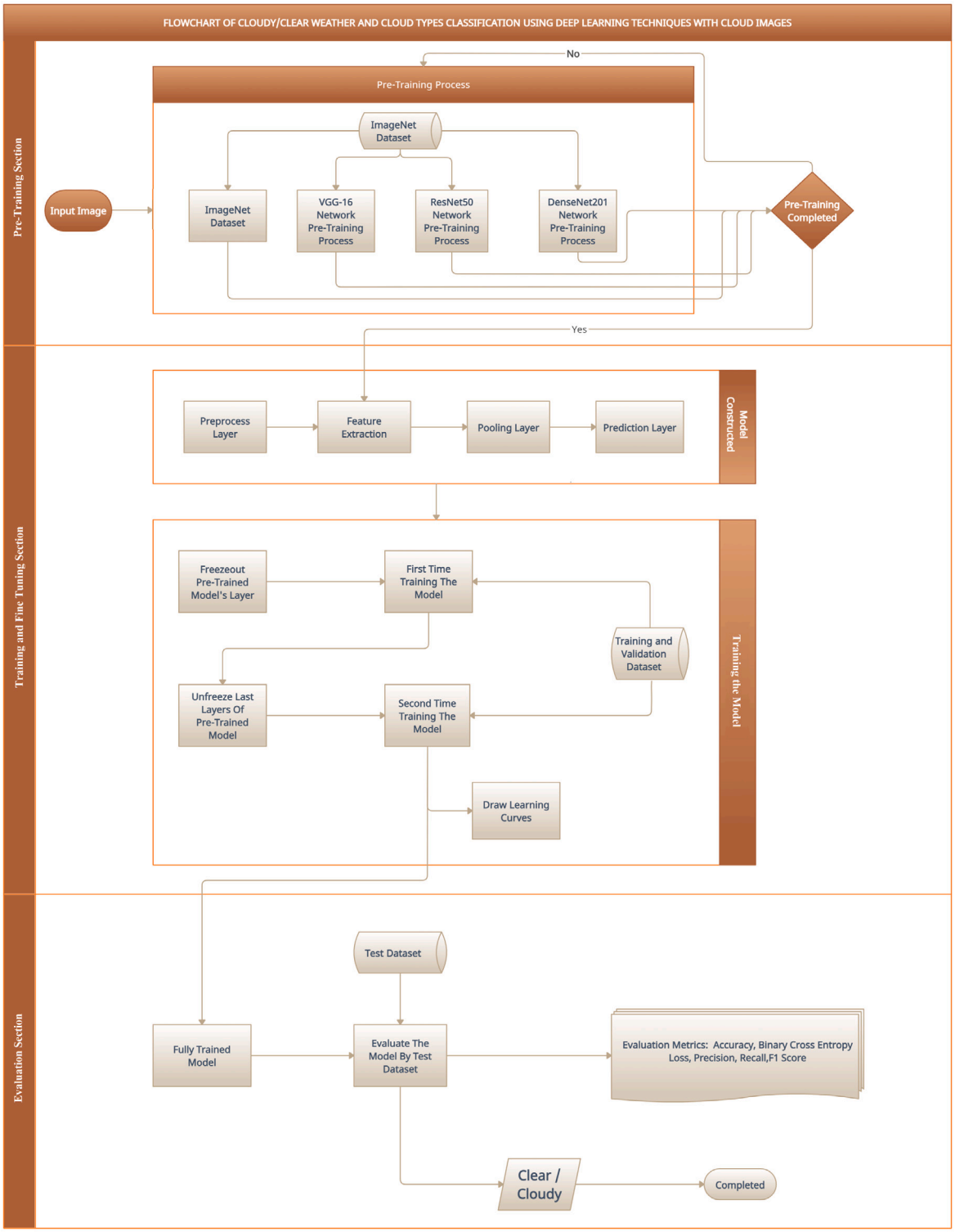

Fig. 21. Flowchart representation of the training and testing processes.

CRediT authorship contribution statement

Mürüvvet Kalkan: Conceptualization, Methodology, Software, Writing – original draft. **Gazi Erkan Bostancı:** Methodology, Writing – review & editing, Supervision. **Mehmet Serdar Güzel:** Writing – original draft, Writing – review & editing, Supervision. **Buğrahan Kalkan:** Software. **Şifa Özsarı:** Validation. **Ömürhan Soysal:** Validation. **Güven Köse:** Conceptualization, Supervision.

Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.compeleceng.2022.108271.

Data availability

For the original source of the experimental data see [18]. The original data has been modified to meet the requirements of the experiment. The experimental data will be made available on request.

Acknowledgment

All authors approved the version of the manuscript to be published.

Appendix

See Fig. 21.

References

[1] Viglione G. How COVID-19 could ruin weather forecasts and climate records. Nature 2020;580(7804):440–1. http://dx.doi.org/10.1038/d41586-020-00924-6.
[2] Weyn JA, Durran DR, Caruana R. Improving data-driven global weather prediction using deep convolutional neural networks on a cubed sphere. J Adv Modelling Earth Syst 2020;12(9). http://dx.doi.org/10.1029/2020MS002109, e2020MS002109.
[3] Arel I, Rose DC, Karnowski TP. Deep machine learning - a new frontier in artificial intelligence research [research frontier]. IEEE Comput Intell Mag 2010;5(4):13–8. http://dx.doi.org/10.1109/MCI.2010.938364.
[4] Yilmaz AA, Guzel MS, Bostanci E, Askerzade I. A novel action recognition framework based on deep-learning and genetic algorithms. IEEE Access 2020;8:100631–44. http://dx.doi.org/10.1109/ACCESS.2020.2997962.
[5] Schultz MG, Betancourt C, Gong B, Kleinert F, Langguth M, Leufen LH, et al. Can deep learning beat numerical weather prediction? Phil Trans R Soc A 2021;379(2194):20200097. http://dx.doi.org/10.1098/rsta.2020.0097.
[6] Zhang J, Liu P, Zhang F, Song Q. CloudNet: ground-based cloud classification with deep convolutional neural network. Geophys Res Lett 2018;45(16):8665–72. http://dx.doi.org/10.1029/2018GL077787.
[7] Alzubaidi L, Zhang J, Humaidi A, Al-Dujaili A, Duan Y, Al-Shamma O, et al. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. J Big Data 2021;8. http://dx.doi.org/10.1186/s40537-021-00444-8.
[8] Brock A, Lim T, Ritchie JM, Weston N. Freezeout: Accelerate training by progressively freezing layers. 2017, arXiv:1706.04983, arXiv preprint arXiv:1706.04983.
[9] Li Z, Hoiem D. Learning without forgetting. IEEE Trans Pattern Anal Mach Intell 2017;40(12):2935–47.
[10] Peng P, Wang J. How to fine-tune deep neural networks in few-shot learning? 2020, arXiv preprint arXiv:2012.00204.
[11] Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. 2017, arXiv:1704.04861, arXiv preprint arXiv:1704.04861.
[12] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. 2014, arXiv:1409.1556, arXiv preprint arXiv:1409.1556.
[13] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2016, p. 770–8.
[14] Huang G, Liu Z, Van Der Maaten L, Weinberger KQ. Densely connected convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2017, p. 4700–8.
[15] Krizhevsky A, Sutskever I, Hinton GE. ImageNet classification with deep convolutional neural networks. In: Pereira F, Burges C, Bottou L, Weinberger K, editors. Advances in neural information processing systems. vol. 25, Curran Associates, Inc.; 2012, URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.
[16] Chan T. Optimal output-sensitive convex hull algorithms in two and three dimensions. Discrete Comput Geom 1996;16(4):361–8. http://dx.doi.org/10.1007/BF02712873.
[17] Graham R. An efficient algorith for determining the convex hull of a finite planar set. Inform Process Lett 1972;1(4):132–3. http://dx.doi.org/10.1016/0020-0190(72)90045-2.
[18] Liu P. Cirrus cumulus stratus nimbus (CCSN) database. Harvard Dataverse; 2019, http://dx.doi.org/10.7910/DVN/CADDPD.