WIND TURBINE TOWER DETECTION USING FEATURE DESCRIPTORS AND DEEP LEARNING

Fereshteh Abedini¹, Mahdi Bahaghighat², Misak S’hoyan³

¹Department of Electrical Engineering Amirkabir University of Technology Tehran, Iran
²Department of Electrical Engineering Raja University Qazvin, Iran
³Department of Information Security National Polytechnic University of Armenia
Yerevan, Armenia

Abstract. Wind Turbine Towers (WTTs) are the main structures of wind farms. They are costly devices that must be thoroughly inspected according to maintenance plans. Today, existence of machine vision techniques along with unmanned aerial vehicles (UAVs) enable fast, easy, and intelligent visual inspection of the structures. Our work is aimed towards developing a vision-based system to perform Nondestructive tests (NDTs) for wind turbines using UAVs. In order to navigate the flying machine toward the wind turbine tower and reliably land on it, the exact position of the wind turbine and its tower must be detected. We employ several strong computer vision approaches such as Scale-Invariant Feature Transform (SIFT), Speeded Up Robust Features (SURF), Features from Accelerated Segment Test (FAST), Brute-Force, Fast Library for Approximate Nearest Neighbors (FLANN) to detect the WTT. Then, in order to increase the reliability of the system, we apply the ResNet, MobileNet, ShuffleNet, EffNet, and SqueezeNet pre-trained classifiers in order to verify whether a detected object is indeed a turbine tower or not. This intelligent monitoring system has auto navigation ability and can be used for future goals including intelligent fault diagnosis and maintenance purposes. The simulation results show the accuracy of the proposed model are 89.4% in WTT detection and 97.74% in verification (classification) problems.

Key words: Machine Vision, Object Detection, Vision Inspection, Wind Turbine, Deep Learning.
1 Introduction

Providing reliable and affordable electricity in order to face the increasing demand of energy in the near future is a worldwide concern. In this regard, developing renewable and clean energy sources such as wind turbine (WT) farms in smart grid (SG) infrastructures can play a crucial role in increasing the capacity of electricity production in many countries across the world. SG deploys widely information and communication technologies (ICT) [1] subsystems. There is almost unlimited number of possible applications of ICT subsystems within the smart grid. SGs with these infrastructures make it more possible to develop reliable systems through Artificial Intelligence (AI) [2–4].

On the other side, guaranteeing the reliability of wind turbines is of great importance. In case of failure and faulty operation, the grid will face interruptions in its service. Challenges and costly breakdowns such as mechanical deformations, surface defects, overheated components in rotor blades, nacelles, slip rings, yaw drives, bearings, gearbox, generators, and transformers are the ones which should be monitored to detect faults intelligently in a wind turbine farm [5–8].

Besides, WTs are costly devices that should have advanced maintenance systems [5]. In order to increase the lifetime of the WTs and reduce the maintenance cost, it is essential to improve the monitoring and maintenance approaches and reach solutions to avoid failure during in-service operation [9, 10]. Vision inspection paves the way toward generating reliable, efficient, and economical electrical energy in wind turbine farms. Image processing and machine learning methods have been widely employed to assist in monitoring and fault diagnostic solutions in energy systems [10–15]. The authors in [11] proposed a learning-based approach to inspect power line infrastructures. In [12], authors suggested a smart framework for system reliability, using machine learning algorithms, to predict failures for preventive maintenance of system components. The authors in [10], benefited from the image data and suggested a model to estimate the rotational velocity of the turbine blade. Estimating the velocity of blades helps to predict the amount of generated power by the WTs in the smart grid. This will ensure the grid to be a more reliable system.
In [13], the authors employed machine learning algorithms to manage the energy of loads and sources in smart grids. The problems of malicious activity prediction and intrusion detection have been analyzed using machine learning techniques in smart grid communication systems in [14, 15]. The authors in [14] detected malicious events and improved system reliability. In [15], a novel method was proposed to reliably warn and anticipate abnormalities and failures in distribution and communication systems.

Deep learning techniques in computer vision applications such as autonomous inspection and monitoring have had a tremendous impact in recent years [16]. Using convolutional neural networks (CNNs) have led computer vision to more advanced approaches. The main feature of a CNN is its deep architecture [17]. One of the common and effective approaches in deep learning is to use a pre-trained network. Several classification problems have been solved using pre-trained networks [17]. For instance, the authors in [16] used deep CNN architecture in the fault classification of power line insulators.

Precise monitoring and forecasting of emerging faults and failures of WTs are critical tasks and can be complex challenges. If the system problems are detected and notified accurately, they can be fixed as soon as possible to increase the reliability of the system. Intelligent vision inspection techniques can be employed to make these predictions and controls to be done automatically and reliably. Therefore, in this work, we propose an intelligent approach, which deploys computer vision techniques to detect wind turbine towers (WTTs). Detection of WTTs can ease the challenges in automatic fault detection and diagnostics process in wind farms through unmanned aerial vehicles (UAV) [18]. In [19], they used signal processing approaches and employed a combination of line and feature detection to locate the wind turbine towers. They started the wind turbine detection stage with Hough transform to detect lines but there are many objects in the background with line shapes, such as horizon, shadow, mountains, and power lines, which are not desired to be located. Detecting the lines and then removing the false detections can cause computational cost and decrease the overall accuracy.

In this research, we developed a new vision-based model to detect WTTs and verify it. The proposed verification step which is implemented using a deep learning classification method, is an extra phase to improve the reliability of the navigation system. This classifier decides between OK and NG (Not Good) detection results. Here, OK means a WTT is detected correctly and the UAV should update its navigation information while NG means a false detection has occurred and the UAV should keep its previous knowl-
edge. In our future works, the UAV will be embedded with thermal vision cameras for advanced visual Nondestructive Tests (NDT).

The remainder of this paper is organized as follows. Section II presents the methodology of the proposed model. The experiments and results are reported in Section III. Finally, in Section IV, we conclude the paper.

Fig. 1: Flowchart of the proposed model.

2 Methodology

In this work, we target developing an appropriate infrastructure to perform vision-based Nondestructive tests (NDT) for wind turbines using UAVs for future works. In order to have a precise navigation and guide the flying machine toward the wind turbine tower and reliably land on it, the position of the wind turbine and its tower must be estimated. To tackle this issue,
we have proposed our model based on the flowchart of Fig. 1. There are two main steps in our model, the first one probing a WTT location in an input image and the second one checking whether it is a real WTT or not using a deep learning classifier. This classifier decides between OK and NG detection results. The UAV updates the position information when the classifier output is OK.

2.1 Dataset

In our research, DB1 includes about 1500 images which have been captured by us in different angles, distances, and backgrounds in a real wind turbine farm. In this wind turbine farm more than 300 WTTs exist and in average 5 images/WTT are available in DB1. This dataset is used for object detection problem. In Fig. 2, some selected samples from DB1 are depicted. Beside this dataset, about 2000 images consisting of two different classes, Not Good (NG) as Class 1 and OK as Class 2 with equal distribution (1000 images for OK class and 1000 samples for NG), have been collected as DB2 to evaluate the performance of the proposed algorithm in the verification stage (classification problem). Fig. 3 illustrates several examples of two mentioned classes in DB2.
2.2 The WTT object detection

Object detection is a fundamental field of study in computer vision and image processing applications [20–27]. Recently, various algorithms have been suggested for object detection purposes [28–31]. These algorithms extract local interest features (key points) and describe them to identify the objects [28–31]. In [28], a well-known algorithm, Scale-Invariant Feature Transform (SIFT) was presented as a scheme for extracting highly distinctive invariant features, which can be used to match different views of objects. The advantage of SIFT is its invariance to scaling, rotation, and translation. The SIFT key point detectors and descriptors have reported to be remarkably effective in different applications [28].

SIFT is computationally expensive, especially for real-time systems. This
has led into thorough research toward an alternative algorithm with lower computational cost such as Speeded Up Robust Features (SURF) and Features from Accelerated Segment Test (FAST) [28–31]. In [29], the authors implemented a new detector and descriptor called SURF which is invariant to scaling and rotation. It is competitive and often superior in terms of repeatability, distinctiveness, and robustness to SIFT, and can be calculated and compared much faster.

SIFT and SURF are both based on detectors and descriptors. Once key points are extracted, a template-matching algorithm must be applied to describe the features. Here, we adopt Brute-Force [28] and FLANN (Fast Library for Approximate Nearest Neighbors) based matcher [32] to test the similarity between the descriptors for the training and test images. The Brute-Force matcher is simple; it takes the descriptor in the first set and compares it to all other descriptors in the second set using a distance calculation. Then the closest ones are returned as the best matches. FLANN contains a collection of optimized algorithms for fast nearest neighbor searches in large datasets and for features which are high dimensional.

2.3 The verification step based on deep learning classifiers

We used SIFT, SURF, FAST algorithms to extract features and detect WTs. Since the object detection stage plays a crucial role in reliability of navigating toward the correct target, we must verify our detection results. If the towers are not detected reliably, UAVs and thermal cameras may hit the blades or land on wrong objects. As a result, the UAVs and cameras may be damaged or the inspection and estimation data may be erroneous.

There are a lot of classical approaches for classification problems such as Random Forest (RF), AdaBoost, k-Nearest Neighbor (kNN), and Support Vector Machine (SVM) [33–38] but to verify the object detection output accurately, we propose use of a pre-trained CNN as the classifier. The authors in [17] presented MobileNet as a class of more efficient models for mobile and embedded vision applications. As indicated in Fig. 4, MobileNets are based on an architecture that uses depth wise separable convolutions to build lightweight deep neural networks [17, 39–41]. In [17], the authors introduced two simple global hyper-parameters that effectively compensate for latency and accuracy. These hyper-parameters allow the model builder to choose the right sized model for their application according to the limitations of the problem. They have presented many experiments on resource and accuracy trade-offs and have demonstrated better performance compared to
In this work, we applied MobileNet [17], ShuffleNet (an extremely efficient convolutional neural network for mobile devices) [42], EffNet (an efficient structure for convolutional neural network) [43], SqueezeNet [44], and ResNet [45, 46] pre-trained classifiers with two added fully connected layers \textit{Dense 1} and \textit{Dense 2}. Dense 2 is the output layer and fixed for our binary classifier while Dense 1’s parameters are targeted as optimization parameters.

ShuffleNet is a practical CNN architecture with high computational efficiency. It provides more feature map channels to encode more information. This is an important point for the performance of very small networks. ShuffleNet is well designed and developed for embedded devices such as mobile phones with very low computing power [42]. SqueezeNet tries to reduce time cost and parameters noticeably while holding on the accuracy [44]. Residual Neural Network so-called ResNet utilizes the bottleneck architecture efficiently to obtain impressive performance [45, 46]. In this model an innovative structure with skip connections and features heavy batch normalization was introduced. Such skip connections are also known as gated...
units or gated recurrent units and have a strong similarity to recent successful elements applied in RNNs. ResNet has proven to be powerful in a lot of applications but one major disadvantage is that the deeper networks usually need several weeks for training, making it impractical in real-world applications. In addition it has large size for most embedded devices. In comparison to ResNet, ShuffleNet has the lower complexity with the same settings. MobileNet and ShuffleNet are favourite models for embedded and mobile systems but EffNet is the optimized model that can be replaced with them in the same applications. EffNet deploys spatial separable convolution, which is simply a depthwise convolution splitted along the x and y axis with a separable pooling between them. It has been shown that it has the same capacity even when applied to narrow and shallow architectures [43]. EffNet block is developed to guarantee the safe replacement of the vanilla convolution layers in mobile hardware applications. Therefore, it has two main advantages, first is the quicker inference and second the application of a larger, deeper network becoming possible [43]. A comparison of MobileNet and ShuffleNet with EffNet is shown in Fig. 5. In this figure, \(dw\) means depthwise convolution, \(mp\) means max-pooling, \(ch\) is for the number of output channels and \(gc\) is for group convolutions [43].

![Diagram of MobileNet, ShuffleNet, and EffNet blocks](image)

**Fig. 5:** A comparison of MobileNet and ShuffleNet with EffNet [43]

(a) An EffNet block  (b) A MobileNet block  (c) A ShuffleNet block

In the next section, we elaborate experimental results in our research.

### 3 Experiments and Results

In our work, all implementations and simulations have been done using Python programming language and TensorFlow by a Core i7 CPU and
NVIDIA GTX 1050 GPU with 16 GB DDR4 RAM memory. As mentioned before in Section 2.1, we carried out our experiments and simulations using images which were captured by us in a real wind farm.

3.1 Using SIFT, SURF, and FAST for WTT object detection

In order to detect the wind turbine towers, firstly, the features and descriptors are extracted using SIFT, SURF, and FAST schemes and then, by applying Brute-Force and FLANN template matching algorithms, a bounding box is predicted for the wind towers. To evaluate the performance of our proposed object detection method, we used the Intersection over Union (IoU) [47]. IoU can be calculated having the ground-truth (GT) bounding box and predicted bounding box of the model. Fig. 6 illustrates different examples of the simulation results in our dataset. The features key points are drawn in blue, predicted bounding box is in red and the ground-truth bounding box is in green.

The goal was to compute the intersection over union of detected bounding box and ground-truth box based on Equation (1).

\[
\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}
\]  
(1)

We considered the IoU as a scoring factor and will decide on the performance of the suggested model based on Equation (2).

\[
\text{Prediction} = \begin{cases} 
0 & \text{IoU} \leq \Lambda \\
1 & \text{IoU} \geq \Lambda 
\end{cases}
\]  
(2)

If Prediction = 0, it actually means a detected object is not acceptable. Fig. 6.(a) demonstrates a poor detection with IoU = 0.0198, while Fig. 6.(b) is an example of a less good prediction with IoU = 0.3129. Fig. 6.(c) is a good detection with IoU = 0.5734, and Fig. 6.(d) demonstrates an excellent bounding box, IoU = 0.9708.

The detection rate of the WTT, \( dr_{WTT} \), can be calculated as introduced in Equation (3). In this equation, \( N \) is the total number of images in the experiment.

\[
dr_{WTT} = \frac{\sum_{i=1}^{N} Prediction}{N} \times 100
\]  
(3)
3.1 Using SIFT, SURF, and FAST for WTT object detection

In order to detect the wind turbine towers, firstly, the features and descriptors are extracted using SIFT, SURF, and FAST schemes and then, by applying Brute-Force and FLANN template matching algorithms, a bounding box is predicted for the wind towers. To evaluate the performance of our proposed object detection method, we used the Intersection over Union (IoU) [47]. IoU can be calculated having the ground-truth (GT) bounding box and predicted bounding box of the model. Fig. 6 illustrates different examples of the simulation results in our dataset. The features key points are drawn in blue, predicted bounding box is in red and the ground-truth bounding box is in green.

The goal was to compute the intersection over union of detected bounding box and ground-truth box based on Equation (1).

\[
\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}
\] (1)

We considered the IoU as a scoring factor and will decide on the performance of the suggested model based on Equation (2).

\[
\text{Prediction} = \begin{cases} 0 & \text{IoU} \leq \Lambda \\ 1 & \text{IoU} \geq \Lambda \end{cases}
\] (2)

If \( \text{Prediction} = 0 \), it actually means a detected object is not acceptable. Fig. 6.(a) demonstrates a poor detection with \( \text{IoU} = 0.0198 \), while Fig. 6.(b) is an example of a less good prediction with \( \text{IoU} = 0.3129 \). Fig. 6.(c) is a good detection with \( \text{IoU} = 0.5334 \), and Fig. 6.(d) demonstrates an excellent bounding box, \( \text{IoU} = 0.9708 \).

To evaluate the performance of the proposed method, we measured the accuracy using Equation (4) and the following metrics: TP (True Positive) indicates the number of correctly classified samples in the OK class, FN (False Negative) indicates the number of samples for which the classification is NG class, but misclassified, TN (True Negative) indicates the number of samples that properly classified as not belonging to the OK class, and FP (False Positive) indicates the number of samples belongs to the class OK but misclassified [48]

\[
\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}
\] (4)

The results of applying SIFT, SURF, and FAST feature extractors with Brute-Force and FLANN template matching algorithms are summarized in Table 1. According to this table, FAST outperforms other feature extractors in term of accuracy. It extracts more features than SIFT and SURF, that makes FAST more powerful in the detection of wind towers. We also observe that SURF is more accurate in extracting and describing features.
in comparison to SIFT. As it is mentioned in Table 1, FLANN template matching works slightly better than Brute-Force.

<table>
<thead>
<tr>
<th>Feature extractor &amp; descriptor</th>
<th>Matcher</th>
<th>Runtime (s)</th>
<th>IoU=0.5</th>
<th>0.3</th>
<th>0.25</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIFT</td>
<td>Brute-Force</td>
<td>0.3955</td>
<td>45.4</td>
<td>74.8</td>
<td>79.8</td>
</tr>
<tr>
<td>SURF</td>
<td>Brute-Force</td>
<td>0.3021</td>
<td>46.6</td>
<td>76.9</td>
<td>82.0</td>
</tr>
<tr>
<td>FAST &amp; SIFT</td>
<td>Brute-Force</td>
<td>0.0728</td>
<td>51.1</td>
<td>81.8</td>
<td>84.1</td>
</tr>
<tr>
<td>FAST &amp; SURF</td>
<td>Brute-Force</td>
<td>0.0623</td>
<td>52.7</td>
<td>83.6</td>
<td>87.3</td>
</tr>
<tr>
<td>SIFT</td>
<td>FLANN</td>
<td>0.3666</td>
<td>47.0</td>
<td>78.5</td>
<td>83.2</td>
</tr>
<tr>
<td>SURF</td>
<td>FLANN</td>
<td>0.2309</td>
<td>49.5</td>
<td>82.7</td>
<td>84.9</td>
</tr>
<tr>
<td>FAST &amp; SIFT</td>
<td>FLANN</td>
<td>0.0821</td>
<td>52.6</td>
<td>82.8</td>
<td>87.3</td>
</tr>
<tr>
<td>FAST &amp; SURF</td>
<td>FLANN</td>
<td>0.0542</td>
<td>54.8</td>
<td>83.8</td>
<td>89.4</td>
</tr>
</tbody>
</table>

### 3.2 Applying deep learning classifiers to verify object detection results

To verify the object detection performance, firstly we used the pre-trained MobileNet classifier with MobileNets Body Architecture [17] that is presented in Table 2. In addition, we deployed the following parameter setting:

The optimizer used in our work is named Adam [49]. We set learning rate to be $3 \times 10^{-7}$ for the optimizer. At two last layers, we used Fully Connected (FC) networks. In these layers, each neuron reads the neurons output in the previous layer and processes the information it needs, and produces the outputs for the next layer [50–52]. The general formula is the following, where $b$ is the BIAS; weights of connections are $w_i$, $f$ is a nonlinear activation function.

$$f(W^t x) = f(\sum_{i=1}^{3} W_i x_i + b)$$

The most common activation functions are Sigmoid function, hyperbolic tangent function (Tanh), and rectified linear function (ReLU). Their formulas
are as follows ([50–52]):

\[ f(W^t x) = Sigmoid(W^t x) = \frac{1}{1 + e^{W^t x}} \] (6)

\[ f(W^t x) = tanh(W^t x) = \frac{e^{W^t x} - e^{-W^t x}}{e^{W^t x} + e^{-W^t x}} \] (7)

\[ f(W^t x) = Relu(W^t x) = \max(0, W^t x) \] (8)

The Sigmoid function receives a value range between 0 and 1, and a real-valued number as the firing rate of a neuron: 0 for not firing or 1 for firing. The hyperbolic tangent functions as a zero-centered output range and uses \([-1, 1]\) instead of \([0, 1]\). For Relu function, if the input is less than 0, its activation will be thresholded at zero. The Softmax function also can be used as the output neuron function and is a logistic function. The function definition is as follows ([50–52]):

\[ \sigma(x) = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k} X_i} \text{ for } j = 1, ..., K \] (9)

In our work, Softmax is used for the final classification at the final layer of the NN.

In the verification problem, the detected objects should be classified into two categories: OK and NG. As a result, the number of neurons in FC output layer (Dense 2) would be equal to two. The classification results are presented in Table 3. Based on the information given in this table, we could achieve the accuracy of 96.01% in classifying the objects in the 5th experiment (E5) as the best result of MobileNet with runtime around 0.034857 (s). Then we conducted our new experiments (E7 to E12) using ResNet50 rather than MobileNet. According to E11, it could result in the higher accuracy about 98.92% in 0.061942 (s). Although ResNet50 achieved the better result in term of the average accuracy, the runtime was doubled. Afterwards, E13 to E18 were implemented for EffNet, ShuffleNet, and SqueezeNet. The obtained results can be evident that EffNet with 1000 iterations and 100 neurons before the output layer (E14) can lead to 97.74% average accuracy in 0.039351 (s) as the best scenario. The validation accuracy and the validation loss for E14 are depicted in Fig. 7. In Fig. 7.(a), the network has been trained for 1000 epochs and we have obtained a validation accuracy of 97.74% and as it is apparent in Fig. 7.(b) the validation loss follows the training loss which is very low.
Table 2: MobileNets Body Architecture [17].

<table>
<thead>
<tr>
<th>Type/Stride</th>
<th>Filter Shape</th>
<th>Input Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv/s2</td>
<td>$3 \times 3 \times 3 \times 32$</td>
<td>$224 \times 224 \times 3$</td>
</tr>
<tr>
<td>Conv dw/s1</td>
<td>$3 \times 3 \times 32$</td>
<td>$112 \times 112 \times 32$</td>
</tr>
<tr>
<td>Conv/s1</td>
<td>$1 \times 1 \times 32 \times 64$</td>
<td>$112 \times 112 \times 32$</td>
</tr>
<tr>
<td>Conv dw/s2</td>
<td>$3 \times 3 \times 64$</td>
<td>$112 \times 112 \times 64$</td>
</tr>
<tr>
<td>Conv/s1</td>
<td>$1 \times 1 \times 64 \times 128$</td>
<td>$56 \times 56 \times 64$</td>
</tr>
<tr>
<td>Conv dw/s1</td>
<td>$3 \times 3 \times 128$</td>
<td>$56 \times 56 \times 128$</td>
</tr>
<tr>
<td>Conv/s1</td>
<td>$1 \times 1 \times 128 \times 256$</td>
<td>$28 \times 28 \times 128$</td>
</tr>
<tr>
<td>Conv dw/s1</td>
<td>$3 \times 3 \times 256$</td>
<td>$28 \times 28 \times 256$</td>
</tr>
<tr>
<td>Conv/s1</td>
<td>$1 \times 1 \times 256 \times 512$</td>
<td>$14 \times 14 \times 256$</td>
</tr>
<tr>
<td>Conv dw/s1</td>
<td>$3 \times 3 \times 512$</td>
<td>$14 \times 14 \times 512$</td>
</tr>
<tr>
<td>Conv/s1</td>
<td>$1 \times 1 \times 512 \times 1024$</td>
<td>$7 \times 7 \times 512$</td>
</tr>
<tr>
<td>Conv dw/s1</td>
<td>$3 \times 3 \times 1024$</td>
<td>$7 \times 7 \times 1024$</td>
</tr>
<tr>
<td>Conv/s1</td>
<td>$1 \times 1 \times 1024 \times 1024$</td>
<td>$7 \times 7 \times 1024$</td>
</tr>
<tr>
<td>Avg Pool/s1</td>
<td>$7 \times 7$</td>
<td>$7 \times 7 \times 1024$</td>
</tr>
<tr>
<td>FC/s1</td>
<td>$1024 \times 1000$</td>
<td>$1 \times 1 \times 1024$</td>
</tr>
<tr>
<td>Softmax/s1</td>
<td>Classifier</td>
<td>$1 \times 1 \times 1000$</td>
</tr>
</tbody>
</table>

3.3 Decision making and updating the positions

Remembering the Fig. 1, the last stage of our model is the decision making. In order to update the location of the WTT, Equations (10) and (11) are used for X-axis and Y-axis, respectively.

\[
X_i^* = \begin{cases} 
X_{i-1}, & D_i = NG \\
X_P^i, & D_i = OK 
\end{cases}
\]  

(10)

\[
Y_i^* = \begin{cases} 
Y_{i-1}, & D_i = NG \\
Y_P^i, & D_i = OK 
\end{cases}
\]  

(11)

where in these equations, $D_i$ denotes the status of the detected object in current image frame, $i$. Here, $X_{i-1}, X_P^i$ and $X_i^*$ are the X positions for
previous frame \((i - 1)\), predicted X position for current frame \(i\), and the new value of X position for the current frame, respectively (for \(i = 1, 2\)). The same rule applies for \(Y_{i-1}, Y_i^p\) and \(Y_i^*\), in Y positions (for \(i = 1, 2\)). According to the result of the classifier and these equations, the UAV can decide on updating its information about the location of the WTT.

**Table 3:** Classification results for the verification problem based on MobileNet, ShuffleNet, EffNet, SqueezeNet, and ResNet

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Base model</th>
<th>Epochs</th>
<th>Dense 1</th>
<th>Run time (s)</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>MobileNet</td>
<td>20</td>
<td>40</td>
<td>0.027058</td>
<td>51.11</td>
</tr>
<tr>
<td>E2</td>
<td>MobileNet</td>
<td>50</td>
<td>40</td>
<td>0.030192</td>
<td>54.43</td>
</tr>
<tr>
<td>E3</td>
<td>MobileNet</td>
<td>100</td>
<td>100</td>
<td>0.032696</td>
<td>76.69</td>
</tr>
<tr>
<td>E4</td>
<td>MobileNet</td>
<td>300</td>
<td>100</td>
<td>0.032752</td>
<td>86.22</td>
</tr>
<tr>
<td>E5</td>
<td>MobileNet</td>
<td>1000</td>
<td>100</td>
<td>0.034857</td>
<td>96.01</td>
</tr>
<tr>
<td>E6</td>
<td>MobileNet</td>
<td>1000</td>
<td>40</td>
<td>0.033198</td>
<td>85.00</td>
</tr>
<tr>
<td>E7</td>
<td>ResNet50</td>
<td>20</td>
<td>40</td>
<td>0.055964</td>
<td>56.01</td>
</tr>
<tr>
<td>E8</td>
<td>ResNet50</td>
<td>50</td>
<td>40</td>
<td>0.054390</td>
<td>56.13</td>
</tr>
<tr>
<td>E9</td>
<td>ResNet50</td>
<td>100</td>
<td>100</td>
<td>0.059588</td>
<td>88.21</td>
</tr>
<tr>
<td>E10</td>
<td>ResNet50</td>
<td>300</td>
<td>100</td>
<td>0.060267</td>
<td>95.78</td>
</tr>
<tr>
<td>E11</td>
<td>ResNet50</td>
<td>1000</td>
<td>100</td>
<td>0.061942</td>
<td>98.92</td>
</tr>
<tr>
<td>E12</td>
<td>ResNet50</td>
<td>1000</td>
<td>40</td>
<td>0.060906</td>
<td>89.46</td>
</tr>
<tr>
<td>E13</td>
<td>EffNet</td>
<td>100</td>
<td>100</td>
<td>0.036546</td>
<td>79.12</td>
</tr>
<tr>
<td>E14</td>
<td>EffNet</td>
<td>1000</td>
<td>100</td>
<td>0.039351</td>
<td>97.74</td>
</tr>
<tr>
<td>E15</td>
<td>ShuffleNet</td>
<td>100</td>
<td>100</td>
<td>0.033497</td>
<td>77.43</td>
</tr>
<tr>
<td>E16</td>
<td>ShuffleNet</td>
<td>1000</td>
<td>100</td>
<td>0.035001</td>
<td>95.89</td>
</tr>
<tr>
<td>E17</td>
<td>SqueezeNet</td>
<td>100</td>
<td>100</td>
<td>0.040951</td>
<td>73.41</td>
</tr>
<tr>
<td>E18</td>
<td>SqueezeNet</td>
<td>1000</td>
<td>100</td>
<td>0.041828</td>
<td>93.36</td>
</tr>
</tbody>
</table>

4 Conclusion

Wind turbine tower (WTT) as a main component in a farm is a mechanical structure where its components are formed and constructed using carbon
fiber reinforced plastic (CFRP) [9]. In order to have intelligent and proactive maintenance services for the farm, it is essential to develop monitoring in-
frastructure based on Vision Inspection (VI) technologies. This can increase the lifetime of the WT farm and reduce the maintenance cost, provided that accurate faults and failures predictions are available via advanced Nondestructive tests approaches such as intelligent VI and thermal visions [6–10].

In this paper, We suggested a scheme to detect the wind turbine towers to facilitate monitoring, controlling, and maintenance tasks in smart grids. We deployed machine learning techniques with vision inspection proposes to navigate a flying machine in a wind turbine farm precisely. We used SIFT, SURF, and FAST as feature extractors and Brute-Force and FLANN as matchers to detect wind turbines. Our simulation results have shown that FAST as the feature extractor with SURF as the descriptor along with FLANN matcher outperforms in object detection task with the 89.4% accuracy. Besides, in order to improve navigation reliability, an additional binary classification step was considered based on ResNet, MobileNet, ShuffleNet, EffNet, and SqueezeNet. Among all mentioned classifiers, ResNet50 obtained the highest average accuracy about 98.92% in 0.061942(s). Its runtime was almost two times compared to EffNet with average accuracy around 97.74%. Therefore, from practical points of view such as computational-efficiency and memory restrictions for an embedded device, our well-tuned pre-trained EffNet was considered as the best classifier among all mentioned models in our research.

References


