Course Project Report

Detection in Adverse Weather Conditions for Autonomous Vehicles via Deep Learning

Submitted By

Vijay Kumar B (221AI043)
Adya N A (221AI006)

as part of the requirements of the course

Data Science (IT258) [Dec 2023 - Apr 2024]

in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology in Artificial Intelligence

under the guidance of

Dr. Sowmya Kamath S, Dept of IT, NITK Surathkal

undergone at

DEPARTMENT OF INFORMATION TECHNOLOGY

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA, SURATHKAL

DEC 2023 - APR 2024

DEPARTMENT OF INFORMATION TECHNOLOGY

National Institute of Technology Karnataka, Surathkal

C E R T I F I C A T E

This is to certify that the Course project Work Report entitled **"Detection in Adverse Weather Conditions for Autonomous Vehicles via Deep Learning"** is submitted by the group mentioned below -

Details of Project Group

Name of the Student	Register No.	Signature with Date
Vijay Kumar B	221AI043	
Adya N A	221AI006	

this report is a record of the work carried out by them as part of the course **Data Science (IT258)** during the semester **Dec 2023 - Apr 2024**. It is accepted as the Course Project Report submission in the partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Artificial Intelligence.**

(Name and Signature of Course Instructor)
Dr. Sowmya Kamath S
Associate Professor, Dept. of IT, NITK

DECLARATION

We hereby declare that the project report entitled **"Detection in Adverse Weather Conditions for Autonomous Vehicles via Deep Learning"** submitted by us for the course **Data Science (IT258)** during the semester **Dec 2023 - Apr 2024**, as part of the partial course requirements for the award of the degree of Bachelor of Technology in Artificial Intelligence at NITK Surathkal is our original work. We declare that the project has not formed the basis for the award of any degree, associateship, fellowship or any other similar titles elsewhere.

Details of Project Group

Name of the Student	Register No.	Signature with Date
1. Vijay Kumar B	221AI043	
2. Adya N A	221AI006	

Place: NITK, Surathkal
Date: March 26, 2024

Detection in Adverse Weather Conditions for Autonomous Vehicles via Deep Learning

Vijay Kumar B, Adya N A

***Abstract*— Accurate real-time classification of adverse weather conditions is crucial for ensuring the safe and reliable operation of autonomous vehicles. This report presents a novel deep learning approach for classifying diverse weather conditions from images, leveraging state-of-the-art computer vision techniques and advanced data preprocessing methods. The study utilizes two up-to-date weather imaging datasets, DAWN and MCWRD, which are combined to create a comprehensive dataset comprising nearly 2000 images across seven weather classes: cloudy, foggy, rainy, snowy, sandy, sunrise, and shine. To enhance the performance and generalization capabilities of the deep learning models, a robust data preprocessing pipeline is employed. Basic preprocessing techniques include image resizing, data augmentation through random rotations, horizontal flips, and center cropping, as well as advanced methods such as color channel splitting, Principal Component Analysis (PCA), and kernel filtering. PCA is applied to each color channel separately, enabling dimensionality reduction while preserving essential information, resulting in a significant reduction in computational complexity without compromising classification accuracy. The report presents a detailed methodology encompassing data preparation, model development, and evaluation stages. Various convolutional neural network (CNN) architectures are explored, and their performance is evaluated using accuracy and loss. The findings demonstrate the effectiveness of the proposed approach in accurately classifying adverse weather conditions, with potential applications in enhancing the safety and reliability of autonomous vehicle systems. By combining cutting-edge deep learning techniques with innovative data preprocessing strategies, this research contributes to the growing body of knowledge in computer vision and autonomous vehicle technology, paving the way for more robust and adaptable perception systems in dynamic weather environments.**

Keywords: **deep learning (DL), convolutional neural networks (CNNs) , ResNet-50, EfficientNet, image classification, weather conditions, autonomous vehicle.**

link to GitHub Repo: `https://github.com/Vijayvkb007/Adverse-weather-classification.git`

link to GitHub Repo: `https://www.overleaf.com/read/bpdxbttthytg#f66f5a`

I. INTRODUCTION

Weather detection systems(WDS) are crucial for various applications, including autonomous vehicles, navigation, and infrastructure management. In case of autonomous vehicles adverse weather conditions pose significant challenges on safety of passengers in the vehicles, different weather conditions require different driving strategies creating the necessity of WDS, which enable it to adapt its behavior to maintain safety and efficiency. Deep learning techniques have emerged as promising solutions in building a robust weather detection in challenging conditions. Our objective is to build a DL-based weather detection system capable of classifying different images of weather conditions to its original labels.

II. PROBLEM STATEMENT

Fast deep learning approach to classify a diverse set of adverse weather conditions from images to support autonomous vehicle operation.

III. DATASETS

To evaluate our models built, we are using two up-to-date weather imaging datasets, namely, DAWN2020 and MCWRD2018. We combined both DAWN2020 and MCWRD2018 datasets to create a dataset of 2015 images across 7 weather classes, namely, Cloudy (312 images), Foggy (300 images), Rainy (413 images), Snowy (204 images), Sandy (323 images), Sunrise (376 images) and Shine (250 images). The DL-based models used takes images as input with standard dimensions of 224x224x3 pixels. Different Extensions files were converted to model's standard, into jpeg format. Several augmentation techniques were applied on the dataset to get a final dataset containing 4394 images.

IV. METHODOLOGY

The main goal of this research is the development of a system that can detect and classify weather conditions in adverse or normal situations using deep learning methods. The intended system is assumed to be deployed with autonomous vehicles and can effectively identify real-time outdoor weather conditions. In building models mentioned in this paper, we follow this methodology consisting of three system of interest (SOI):

1. Data preparation
2. Learning models
3. Evaluation

A. Data Preparation subsystem

This subsystems includes all the methods used for data pre-processing.

Combining two weather imaging datasets, resulted in combined dataset with around 2015 images of different weather conditions with 7 classes (**Fig 1** shows sample images from each class). All the three CNN models take images as input and each images with a standard dimension of 224x224x3 pixels, so each images are resized accordingly, If different format/extensions of images are present, then they are converted into jpg format which are then feed to the convolution neural network.

Fig. 1: Images from different classes

Number of images were increased from around 2015 images to around 4300 images by applying augmentation techniques. **Fig 2** illustrates some new images resulting from applying data augmentation pre-processing operations.

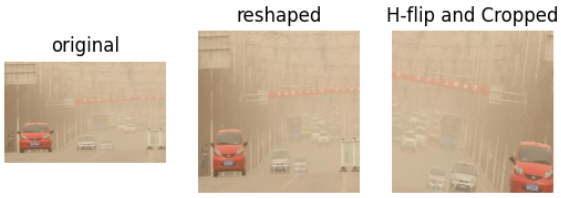

Fig. 2: Comparison of images before and after augmentation

Data processing is done in two folds, namely, Basic data processing, and Advanced data processing. This includes experimentation on datasets.

1) Basic data pre-processing: To ensure our deep learning model performs optimally for image classification, we implemented a specific data preprocessing pipeline. This process is crucial for preparing the images in a way that allows the model to effectively extract meaningful features.

A key consideration during preprocessing was minimizing information loss. Since the task might rely on the original intensity values within the images, we avoided techniques like normalization, which typically scales pixel values to a specific range. Similarly, converting images to grayscale discards color information altogether, potentially hindering object distinction. We also opted out of excessive color jitters, which introduce random variations in brightness, contrast, or saturation. While these jitters can improve generalizability in some cases, they might introduce unrealistic color changes that confuse the model, especially if color is important for classification.

Therefore, our chosen preprocessing techniques focus on preparing the images for the model while preserving the information critical for the task. Images are first resized to a standard dimension (224x224 pixels) to ensure compatibility with the model's input requirements. To make the model more robust to variations in object orientation, we introduce random rotations during preprocessing. Additionally, random horizontal flips help the model learn that objects can appear on either side of the image. Finally, we employ center cropping, which randomly extracts a central square region from each resized image. This prevents the model from overfitting to specific backgrounds or unimportant areas within the images. Optionally, a slight Gaussian blur can be applied to introduce some tolerance to noise and variations in image quality. By combining these techniques, we aim to strike a balance between preparing the data for the model and maintaining the information essential for accurate image classification.

```
# This is code snippet for augmentation functional
# -ity which uses torch.transforms
def augmentation(imf, otf):
    data_transforms = transforms.Compose([
    transforms.Resize((300, 300)),
    transforms.RandomRotation(degrees=15),
    transforms.CenterCrop(224),
    transforms.RandomHorizontalFlip(p=1.0),
    transforms.ToTensor(), ])

# to apply this on images use
img_ =  data_transforms(img)
img_ = transforms.ToPILImage()(img_)
```

2) Advanced data processing: Image Splitting One of the fundamental techniques in image processing is splitting an image into its individual color channels. In this study, the input image is split into its Blue, Green, and Red channels. By separating the image into these channels, it becomes possible to analyze and manipulate each channel independently, allowing for a deeper understanding of the image's color composition and characteristics. Fig 3. shows splitting of color channels for an image.

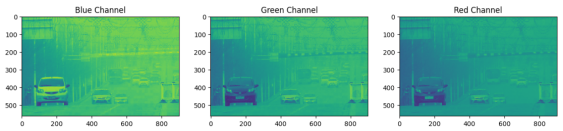

Fig. 3: Splitting of color channels

Principal Component Analysis (PCA) is a widely used dimensionality reduction technique in image processing and computer vision. Normally images have a lot of pixels to retain their clarity, but that significantly increases its size and slows down the performance of the system when it has to process multiple images. To overcome this situation we use this dimensionality reduction technique which comes under Unsupervised Machine Learning. Here, PCA is applied to each color channel (Blue, Green, and Red) separately to reduce the dimensionality of the data while preserving the most relevant information. The explained variance ratio is calculated for each color channel, which represents the amount of variance in the original data captured by the principal components. The eigenvalues of the principal components are visualized using bar plots, providing insight into

the contribution of each principal component to the overall variance. Fig.4 shows bar plot of eigenvalues of PCA.

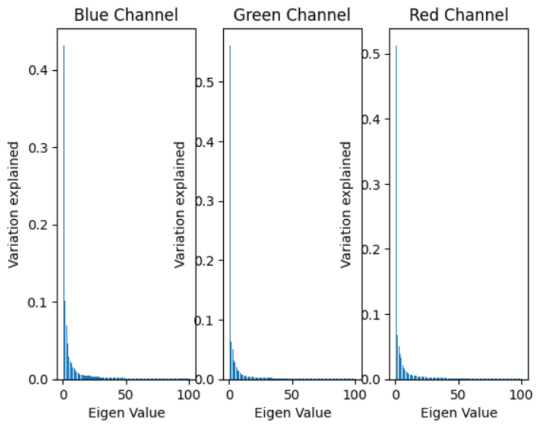

Fig. 4: Barplots of eigenvalues for each channels

After performing PCA on each color channel, the image is reconstructed using the reduced set of principal components. This reconstructed image is then compared with the original image to assess the quality and information loss resulting from the dimensionality reduction process as shown in Fig.5. With only keeping 50 dimensions we can retain almost 90% of the image data. Fig.6 shows for n_components=10.

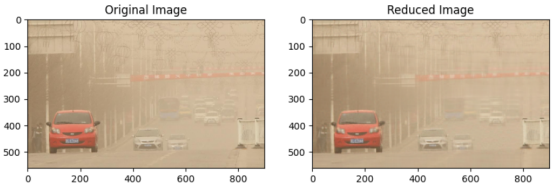

Fig. 5: Original v/s reconstructed image (50 components)

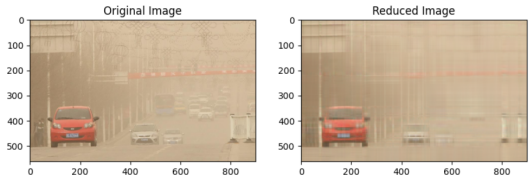

Fig. 6: Original v/s reconstructed image (10 components)

We need score the reconstruction of the image after applying PCA, which is discussed after the pipeline for applying PCA on images. below is the code used for preparation:

```
# transform and reconstruct image
from sklearn.decomposition import PCA
import cv2

# convert it to RGB later
img = cv2.imread(img_path)
blue,green,red = cv2.split(img)
# scale down pixels 225 -> 1

# fit and transform
pca = PCA(n_components=50)
pca.fit(df_blue)
trans_pca = pca.transform(df_blue)
# inverse transfrom
arr = pca.inverse_transform(trans_pca)

# merge channels
img_reduced= (cv2.merge((b_arr, g_arr, r_arr)))
```

The Pipeline to obtain dimensionally reduced image is as follows, split the images into different color channels (RGB) , apply PCA on each channel-separated images by transforming them, get top principle components and apply inverse transformations to all images, finally merge all the channels to get the images with reduced dimension. Now the image is dimensionally reduced and can be resized to 224x224 (standard input size for most CNN models), this kind of reduction by resizing is called spatial reduction. Spatial reduction process effectively reduces the total number of pixels in the image which is different than dimensionality reduction.

While few classes gave a very good reconstruction quality score after applying Visual Information Fidelity (VIF) on the reconstructed images through PCA, the table below shows the results obtained. We also measured score structural similarity using universal quality index (uqi) in which all images a good reconstruction, and the table belows shows the results obtained.

TABLE I: VIF score for PCA reconstructed images

Class	VIF score
sand	0.45
cloud	0.80
fog	0.79
rain	0.14
shine	0.37
snow	0.45
sunrise	0.72

TABLE II: UQI score for PCA reconstructed images

Class	UQI score
sand	0.99
cloud	0.99
fog	0.99
rain	0.78
shine	0.99
snow	0.99
sunrise	0.97

The results above indicate that reconstruction quality of rainy images are not so good, so PCA avoided on raniy images.

Kernel filtering is a fundamental technique in image processing that involves convolving an image with a predefined

kernel or filter. Here, various kernel filters are applied to the input image, including: Small and large blur filters: These filters apply a smoothing effect to the image, reducing noise and enhancing overall image quality. Sharpening filter: This filter enhances the edges and details in the image, making them appear sharper and more defined. Laplacian filter: The Laplacian filter is commonly used for edge detection, as it highlights the regions in an image where there are rapid intensity changes. Sobel filters (x and y directions): The Sobel filters are gradient-based edge detection filters that compute the approximate gradient magnitude and orientation at each pixel in the image. The effects of applying each kernel filter are visualized by displaying the original and filtered images side by side, allowing for a visual comparison and analysis of the filtering results.

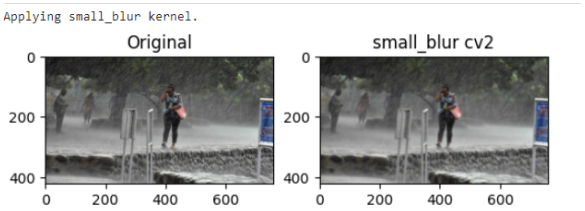

Fig. 7: Image before and after applying blur filter

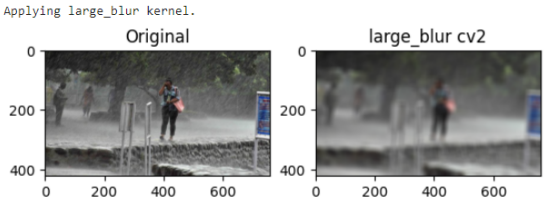

Fig. 8: Image before and after applying blur filter

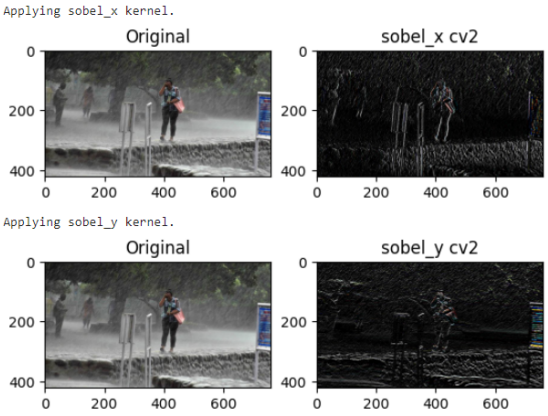

Fig. 9: Image with sobel filter in x and y direction

B. Learning Model Subsystem

The learning model subsystem employs a transfer learning approach, leveraging pre-trained deep convolutional neural network (CNN) models for image classification. Transfer learning is a technique that involves using a model that has been pre-trained on a large dataset, such as ImageNet, and then fine-tuning it on a new dataset for a specific task. This approach allows the model to take advantage of the learned features from the pre-training dataset, which can significantly improve performance and reduce the amount of training data required for the new task.

The images of size 224X224 are feed and needs to be processed by techniques like normalizing the pixel values, performing data augmentation techniques like random flips, rotations, or crops to increase the diversity of the training data and improve the model's generalization ability. Some processing like normalization and colour jitter reduced our model accuracy, so after proper training and testing, those preprocessing techniques were removed.

```
def augmentation(image_folder, output_folder):
    data_transforms = transforms.Compose([
    transforms.Resize((300, 300)),
    transforms.RandomRotation(degrees=20),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    ])
    image = Image.open(image_path)
    img = data_transforms(image)
    img = transforms.ToPILImage()(img)
    # store it in some place ..
```

Two pre-trained deep CNN models are employed for transfer learning: ResNet-50 and EfficientNet-B0. ResNet-50 is a 50-layer deep residual network architecture that has been pre-trained on the ImageNet dataset, a large dataset consisting of over 1 million images spanning 1,000 different classes. EfficientNet-B0 is a lightweight and efficient CNN architecture that is part of the EfficientNet family, also pre-trained on ImageNet. These models have shown remarkable performance on various computer vision tasks and serve as strong starting points for transfer learning.

The ResNet-50 architecture is known for its use of skip connections and residual blocks, which allow for effective training of very deep networks by mitigating the vanishing gradient problem. The EfficientNet-B0 architecture, on the other hand, employs mobile-inverted bottleneck convolutions and specialized scaling methods for efficient model scaling All these details specific to our techniques are as follows.

1) **Input Layer:** Images in RGB color format, with dimensions of 224 pixels by 224 pixels by 3 channels (red, green and blue), are used as inputs for convolutional neural networks model.
2) **Processing Layer:** Transfer learning is employed using two pre-trained deep CNN models:
   - ResNet-50 CNN
   - EfficientNet-b0 CNN

```
# loading pre-trained efficientnetB0 model ,
# discarding the top layers
effi_model = EfficientNetB0(
```

```
                    weights="imagenet",
                    include_top=False,
                    input_shape=(224, 224, 3)
                    )
resnet_model = ResNet50(
                    weights="imagenet",
                    include_top=False,
                    input_shape=(224, 224, 3)
                    )
```

3) **Output Layer:** Added 2 fully connected layers to the top of both the models, with first fully connected layers for
   - ResNet-50 : 2048 Nodes,
   - EfficientNet-B0 : 1280 Nodes

   with ReLU activation function, final output layer consists of 7 nodes implying 7 class output with SoftMax activation. The class with highest probability will be returned as prediction.

```
# Add custom layers for classification
x1 = effi_model.output
x1 = tf.keras.layers.GlobalAveragePooling2D()(x1)
x1 = Dense(1280, activation="relu")(x1)
predictions1 = Dense(num_classes,
                     activation="softmax")(x1)

# Add custom layers for classification
x2 = resnet_model.output
x2 = tf.keras.layers.GlobalAveragePooling2D()(x2)
x2 = Dense(2048, activation="relu")(x2)
predictions2 = Dense(num_classes,
                     activation="softmax")(x2)
```

4) **Model Architecture:**
   - ResNet-50:
     a) It uses skip connections and residual blocks to improve training of very deep networks.
     b) A Global average polling layer before output layer.

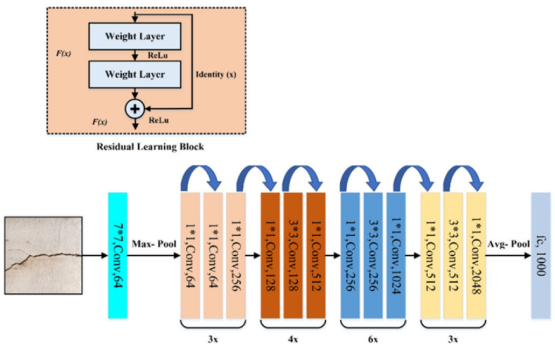

Fig. 10: Architecture of ResNet-50

   - EfficientNet-B0:
     a) It uses mobile-inverted bottleneck convolutions and specialized scaling methods for efficient model scaling.
     b) The base model was loaded without the top classification layers. Custom head with GlobalAveragePooling2D, Dense(1280, relu), Dense(7, softmax) was added.

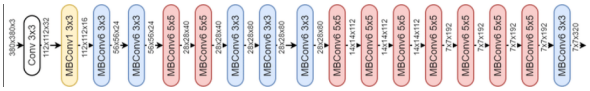

Fig. 11: Architecture of EfficientNet-B0

5) **Implementation:**
   There is implementation for both EfficientNet-B0 and ResNet-50 models, both implemented through tensorflow/keras DL framework. Each model are pretrained on images net dataset which create an opportunity to use these pretrained models retaining the weights. The top layers are removed to change the number of output classes, the custom top layer includes a global average pooling layer with fully connected layers following it. The model is compiled with Adam optimizer as optimizer and Sparse Categorical Cross-Entropy loss as loss function.
   Tensorflow provides many functions to handle datasets, we are making use of tensorflow.keras.utils.image_dataset_from_directory to split data into train, test, and validation dataset. Both the models are trained for 100 epochs and test loss and accuracy are evaluated
   Data imbalance are removed by creating more image by augmentation technique and by setting a fixed number of image intake from each class, after generating augmented images we take around 600 images per class to train our data.

C. Evaluation Subsystem

1) **Train/Validation/Test Split:**
   The two separate datasets doesn't come in a structured format, it is essential to make it structured to access data from tf.keras.utils.image_dataset_from_directory function provided by tensorflow to make train, test, validation splits. So we segregated images of different classes into their respective labeled files. Dataset is divided into 80:20 ratio for training and testing data, train dataset is again split into training and validation dataset in 80:20 ratio.
2) **Test Evaluation:**
   - Model in trained using model.fit() function, which takes training and validation dataset as input, and 100 as number of epochs.
   - Similarly there is a function to evaluate the loss and accuracy, model.evaluate() returns the loss and accuracy for the test dataset.
   - Different runs resulted slight different accuracy and loss, considering the best out of all the EfficientNet-B0 gave an accuracy of 0.9530 and ResNet-50 gave an accuracy of 0.9345, which is comparable to other previous works.

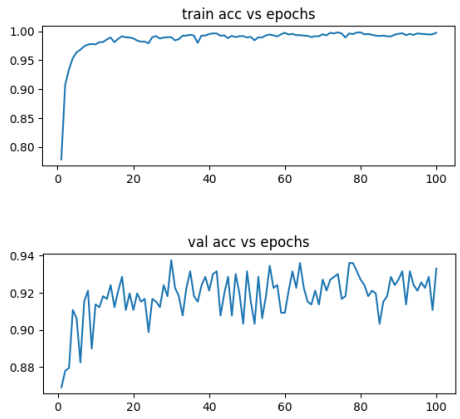

Fig. 12: Accuracy v/s epochs for EfficientNet-B0 model

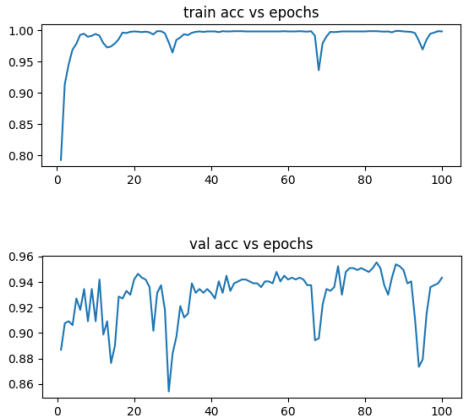

Fig. 13: Accuracy v/s epochs for ResNet-50 model

TABLE III: Accuracy for each class

Class Name	Accuracy (EffiecientNet-B0) test accuracy : 91%	Accuracy (ResNet-50) test accuracy : 93%
Cloudy	0.8512	0.8524
Fog	0.8536	08762
Rainy	0.8750	0.8607
Sand	0.8643	0.8429
Shine	0.8560	0.8595
Snow	0.8452	0.8488
Sunrise	0.8548	0.8595

TABLE IV: Evaluation Metrics (best scores from all runs)

Refernce Number	Model Year	Number of classes	Classification Models	Accuracy
[4]	2019	4 classes	Stacked ensemble Algorithms	86.00%
[2]	2021	11 classes	DeepMete CNN	92.00 %
[1]	2022	6 classes	SqueezeNet CNN	96.00 %
proposed	2024	7 classes	EfficientNet-B0 CNN	95.30%
proposed	2024	7 classes	ResNet-50 CNN	93.45%

V. CONCLUSIONS

This project aimed to develop an effective deep learning approach for classifying adverse weather conditions from images to support autonomous vehicle operations. To achieve this, two up-to-date weather imaging datasets, DAWN2020 and MCWRD2018, were combined, resulting in a dataset of 2015 images across seven weather classes. Various data preprocessing techniques were employed, including image resizing, data augmentation, color channel splitting, Principal Component Analysis (PCA), and kernel filtering.

The data preprocessing pipeline played a crucial role in preparing the images for the deep learning models. Basic preprocessing techniques, such as resizing, random rotations, horizontal flips, and center cropping, were applied to enhance the model's robustness and prevent overfitting. Advanced data processing methods, including color channel splitting and PCA, were used to reduce the dimensionality of the data while preserving essential information. Kernel filtering techniques, including blur, sharpening, Laplacian, and Sobel filters, were also explored to enhance image quality and edge detection.

The dimensionality reduction approach using PCA demonstrated promising results, retaining approximately 90% of the image data while reducing the number of components to 50. This significant reduction in dimensionality can potentially improve the computational efficiency of the deep learning models without compromising the accuracy of weather condition classification.

Overall, the combination of data preprocessing techniques and dimensionality reduction methods laid a solid foundation for developing effective deep learning models for classifying adverse weather conditions in real-time scenarios. This project contributes to the advancement of autonomous vehicle technology by enabling more accurate and reliable detection of various weather conditions, ultimately enhancing the safety and performance of these systems.

REFERENCES

[1] Qasem Abu Al-Haija, Manaf Gharaibeh, and Ammar Odeh, "Detection in Adverse Weather Conditions for Autonomous Vehicles via Deep Learning", April 2022

[2] Xiao, H.; Zhang, F.; Shen, Z.; Wu, K.; Zhang, J. Classification of Weather Phenomenon from Images by Using Deep Convolutional Neural Network. Earth Space Sci. 2021, 8, e2020EA001604.

[3] Abu Al-Haija, Q.; Smadi, M.A.; Zein-Sabatto, S. Multi-Class Weather Classification Using ResNet-18 CNN for Autonomous IoT and CPS Applications. In Proceedings

of the International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 16–18 December 2020; pp. 1586–1591

[4] Oluwafemi, A.G.; Zenghui, W. Multi-Class Weather Classification from Still Image Using Said Ensemble Method.South Africa, 28–30 January 2019; pp. 135–140.