MULTI-CLASS WEATHER CLASSIFICATION

Yashwanth Gajji

Abstract

Meteorology, agriculture, transportation, and other industries rely heavily on weather classification. Due of the widespread availability of high-quality cameras and the promise to give more detailed and accurate meteorological information, image-based weather classification has attracted growing attention in recent years. In this paper, we offer a deep learning-based approach for image-based weather classification. To extract characteristics from the input photos and train a classifier to forecast the weather state, we employ a pre-trained convolutional neural network architecture, such as EfficientNetB0. We also investigate data augmentation strategies like as rotation, scaling, and flipping to boost the model's generalization capability.

Introduction

Deep learning, a subfield of machine learning, has shown considerable promise in handling image-based weather categorization challenges. Convolutional neural networks (CNNs) are a type of deep neural network that has achieved state-of-the-art performance on a variety of computer vision tasks, including image classification, object identification, and segmentation. EfficientNetB0 is a popular design that is noted for its efficiency and efficacy in balancing model size and accuracy.

In this work, we present a deep learning-based approach for image-based weather classification. EfficientNetB0 is used to extract features from input photos and train a classifier to predict the weather. We also explore data augmentation strategies like as rotation, scaling, and flipping to boost the model's generalization capability. On a publicly available weather image dataset, we test the suggested approach and achieve excellent accuracy in predicting diverse weather situations such as bright, cloudy, rainy, and snowy. The proposed method could be employed in a variety of real-world applications, including automated weather monitoring systems and weather forecasting.

Related work

Several studies have investigated the use of machine learning methods for image-based weather classification. Sharma et al. (2018) suggested a convolutional neural network-based method for classifying the weather from satellite photos in one study. They classified four weather conditions cloudy, sunny, rainy, and snow with great accuracy.

Mazzarella et al. (2018) used ground-level camera images to construct a deep learning-based system for weather identification. They classified four meteorological conditions using a combination of convolutional neural networks and recurrent neural networks, including clear sky, partly cloudy, overcast, and rain.

Furthermore, various research has studied the application of several machine learning techniques, such as decision trees, support vector machines, and random forests, for weather categorization utilizing weather station data (Chandola et al., 2016; Ghosh et al., 2018).

Furthermore, some studies have investigated the use of data augmentation strategies to improve deep learning model performance in weather categorization tasks. Cao et al. (2020), for example, improved the accuracy of their deep learning model for weather categorization utilizing ground-level camera images by using data augmentation techniques such as rotation, scaling, and flipping.

In another study, Song et al. (2019) improved the robustness of their deep learning model for weather categorization using satellite photos by applying data augmentation techniques such as rotation, scaling, and translation.

Programming Language and Libraries:

- Python is used as the main programming language.
- Keras, a high-level neural networks API, is used to build and train the deep learning model.
- EfficientNetB0, a pre-trained convolutional neural network architecture, is used as the base model.
- NumPy and Pandas libraries are used for data manipulation and preprocessing.
- Scikit-learn library is used for evaluation metrics such as accuracy, precision, recall, and F1 score.
- Matplotlib and Seaborn libraries are used for data visualization.

Data Pre-processing:

- The input images are resized to 224 x 224 pixels, which is the input size of the EfficientNetB0 model.
- The pixel values are normalized to be between 0 and 1.
- The images are split into training, validation, and testing sets with a ratio of 70:15:15, respectively.
- Data augmentation techniques such as rotation, zooming, horizontal and vertical flipping are applied to the training set to increase the size of the dataset and improve the generalization capability of the model.

Model Architecture:

- The base model, EfficientNetB0, is used as the starting point for the weather classification model.
- A BatchNormalization layer is added to normalize the output of the previous layer.
- Two Dense layers with 256 and the number of classes neurons, respectively, are added with ReLU activation.
- A Dropout layer with a rate of 0.45 is added to prevent overfitting.
- The output layer is a Dense layer with the number of classes neurons and a softmax activation function.

Training and Evaluation:

- The model is trained using the Adam optimizer with a learning rate of 0.0001, categorical cross-entropy loss function, and a batch size of 32 for 50 epochs.
- The model is evaluated on the validation set after each epoch, and the best model based on the validation accuracy is saved.
- The final performance of the model is evaluated on the test set using evaluation metrics such as accuracy, precision, recall, and F1 score.

Model and Algorithm

A high-level algorithm for this project is:

1. Load the dataset of ground-level camera images.
2. Pre-process the images:
    a. Resize the images to 224 x 224 pixels.
    b. Normalize the pixel values to be between 0 and 1.
    c. Split the dataset into training, validation, and testing sets with a ratio of 70:15:15.
    d. Apply data augmentation techniques such as rotation, zooming, and flipping to the training set.
3. Load the pre-trained EfficientNetB0 model as the base model.
4. Add a BatchNormalization layer to normalize the output of the previous layer.
5. Add two Dense layers with 256 and the number of classes neurons, respectively, with ReLU activation.
6. Add a Dropout layer with a rate of 0.45 to prevent overfitting.
7. Add an output layer with the number of classes neurons and a SoftMax activation function.
8. Compile the model using the Adam optimizer with a learning rate of 0.0001, categorical cross-entropy loss function, and a batch size of 32.
9. Train the model for 50 epochs, evaluating the model's performance on the validation set after each epoch and saving the best model based on validation accuracy.

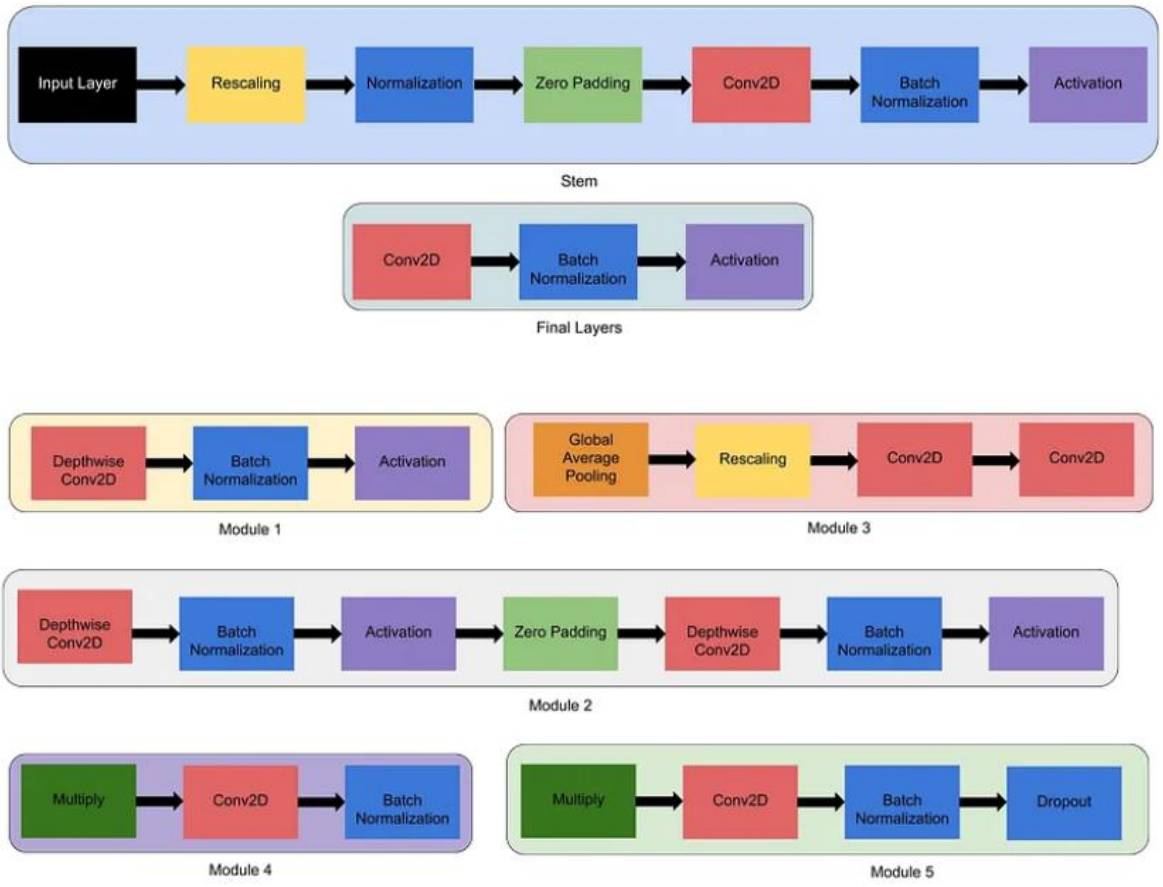
Evaluate the final performance of the model on the test set using evaluation metrics such as accuracy, precision, recall, and F1 score.

EfficientNetB0

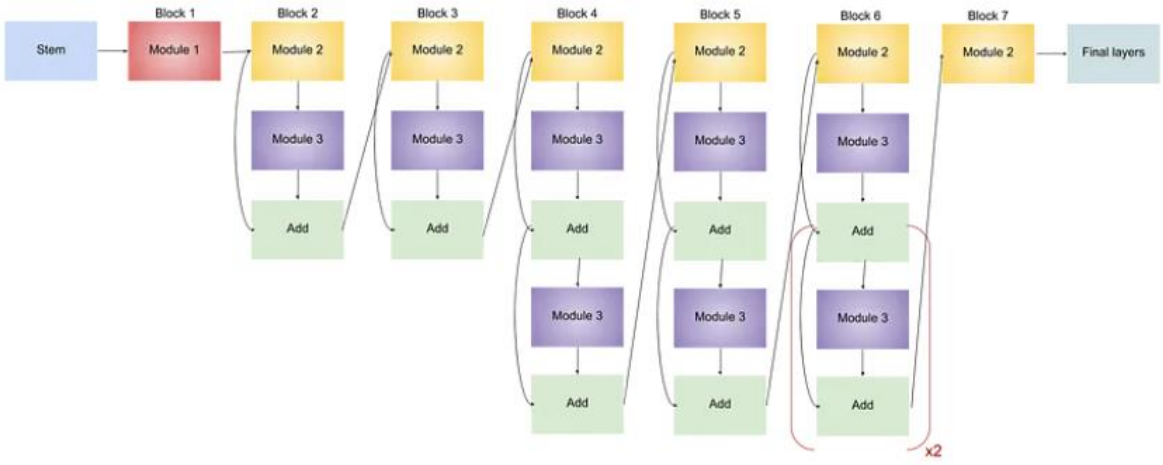
EfficientNetB0 is a convolutional neural network architecture proposed by Google researchers in 2019. It is part of the EfficientNet family of models, which are intended to outperform prior state-of-the-art models in terms of accuracy and efficiency.

EfficientNetB0 is distinguished by a scaling strategy that balances model depth, width, and resolution using compound scaling to enhance performance while minimizing computational cost. This method involves scaling the network's depth, width, and resolution all at once, rather than separately, allowing for more efficient use of computational resources.

EfficientNetB0's architecture is made up of a sequence of convolutional layers, followed by global average pooling and a fully connected layer for classification. It also incorporates several innovative features, such as squeeze-and-excitation blocks, which increase the model's performance by helping it to focus on relevant elements in the input images.



5 modules we will use to make the architecture.



Architecture for EfficientNet-B0. (x2 means that modules inside the bracket are repeated twice)

Dataset Description

Multi-class Weather Dataset for Image Classification is image dataset consisting of a total of 1125 images. Images are of four categories, Cloudy, Rain, Shine, Sunrise.

- Cloudy – 300 images
- Rain – 215 images
- Sunrise – 357 images
- Shine – 253 images

Experiment

Data Preprocessing

Dataset is divided into training and test set using with a ratio of 4:1. Further test data is split into validation set and test set in the equal ratio. Training set contains 900 images, Validation set contains 112 images and test set contains 113 images. ImageDataGenerator is a Keras library class that helps to pre-process and enhance image data during deep learning model training. During the model training process, it generates batches of augmented picture data in real-time, which can help to improve the trained model's accuracy and resilience. The ImageDataGenerator class enables picture augmentation techniques like as rotation, zooming, horizontal and vertical flips, brightness modifications, and more to be applied to training data. It also enables the use of data normalization and scaling to image data. ImageDataGenerator is used to augment the training and testing data.

Keras Callback functionality

MyCallback is a subclass of the Keras library's keras.callbacks.Callback class. It enables you to create custom behavior during Keras model training and assessment. You can specify methods that will be called at various moments of the training and evaluation process by subclassing keras.callbacks.Callback, such as at the start or finish of each epoch, or at the start or end of the training process as a whole. The __init__ function takes in several parameters, including the neural network model model, the number of epochs to wait before reducing the learning rate patience, the number of times to reduce the learning rate without improvement before stopping training stop_patience, the threshold accuracy for adjusting the learning rate based on validation loss threshold, the factor by which to reduce the learning rate factor, the number of training batches to run per epoch batches, the total number of epochs to run epochs, and the epoch at which to start asking the user for input ask_epoch. The callback function monitors a number of parameters, such as the number of times the learning rate has been reduced without improvement count, the number of times the learning rate has been reduced without improvement that has caused training to stop stop_count, the epoch with the lowest validation loss best_epoch, the initial learning rate of the model initial_lr, the highest training accuracy highest_tracc, the lowest validation loss lowest_vloss, and the best_weights.

The callback determines the length of an epoch at its conclusion and extracts a number of metrics from the logs, such as the training and validation accuracy and loss, the current learning rate, and the improvement in the metric of interest (either the training accuracy or the validation loss) since the previous epoch. The callback modifies the learning rate based on the training accuracy if it falls below a predetermined threshold. The callback increases the

learning rate by a specific factor (determined by the factor parameter) if the training accuracy has not increased for a predetermined number of epochs (specified by the patience parameter). The callback saves the model weights if the validation loss is lower than the lowest validation loss recorded thus far. The callback terminates training if the learning rate has been altered a predetermined number of times without result (determined by the stop_patience argument). The callback modifies the learning rate depending on the validation loss if the training accuracy is greater than the threshold. The callback stores the model weights if the validation loss has decreased since the previous epoch. The callback terminates training if the learning rate has been altered a predetermined number of times without producing any improvement. The callback will request the user's consent to carry on with training after the specified epoch if the ask_epoch argument is set to a certain value. The callback will keep requesting at the chosen epoch until it receives an input if the user does not provide a list of epochs to execute.

Model

The EfficientNetB0 model serves as the foundation. The model accepts 224x224 pictures with three color channels. The final dense layer's class count is determined by counting the number of class indices in the training data. The base model is loaded with ImageNet pre-trained weights, and the top layer is deleted to allow for the addition of new layers for our specific assignment. After that, a batch normalization layer is added, followed by a dense layer of 256 neurons, L1 and L2 regularization, and the ReLU activation function. To avoid overfitting, a dropout layer is applied. Finally, the predicted probabilities for each class are output by a dense layer with softmax activation.

```
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb0_notop.h5
16705208/16705208 [==============================] - 0s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 efficientnetb0 (Functional)  (None, 1280)             4049571

 batch_normalization (BatchN  (None, 1280)             5120
 ormalization)

 dense (Dense)               (None, 256)               327936

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 4)                 1028

=================================================================
Total params: 4,383,655
Trainable params: 4,339,072
Non-trainable params: 44,583
_________________________________________________________________
```

Training parameters are set as follows:

- batch_size = 16
- epochs = 60
- patience = 1

- stop_patience = 3
- threshold = 0.9
- factor = 0.5
- ask_epoch = 5

Model is trained using the training set. Trained model is evaluated on validation and test set.

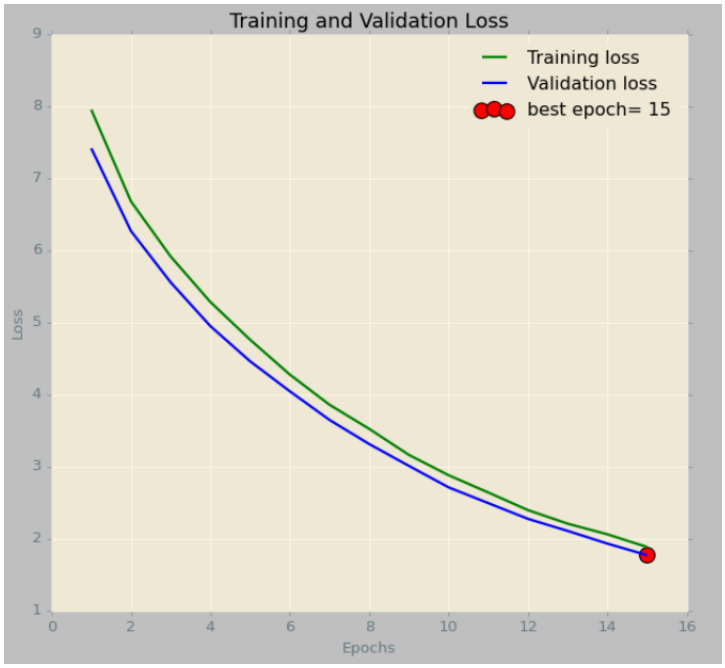
Results and Accuracy

Model is trained on the training set for 15 epochs with a batch size of 16. and halted as there is not much improvement in the accuracy. Below image shows the training values.

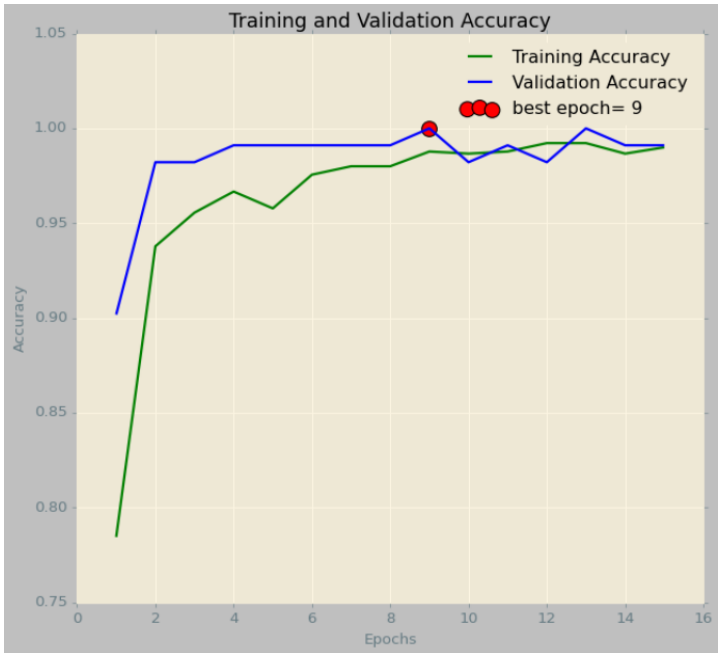
```
Do you want model asks you to halt the training [y/n] ?
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
 1 /60     7.950   78.444   7.41413  90.179   0.00100  0.00100  accuracy     0.00    343.13
 2 /60     6.676   93.778   6.26531  98.214   0.00100  0.00100  val_loss    15.49    294.43
 3 /60     5.911   95.556   5.55325  98.214   0.00100  0.00100  val_loss    11.37    289.63
 4 /60     5.281   96.667   4.94575  99.107   0.00100  0.00100  val_loss    10.94    294.33
 5 /60     4.757   95.778   4.45834  99.107   0.00100  0.00100  val_loss     9.86    287.78
enter H to halt training or an integer for number of epochs to run then ask again
 training will continue until epoch 15
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  % Improv  Duration
 6 /60     4.274   97.556   4.04174  99.107   0.00100  0.00100  val_loss     9.34    293.45
 7 /60     3.855   98.000   3.64405  99.107   0.00100  0.00100  val_loss     9.84    291.20
 8 /60     3.520   98.000   3.30941  99.107   0.00100  0.00100  val_loss     9.18    293.69
 9 /60     3.159   98.778   3.00646  100.000  0.00100  0.00100  val_loss     9.15    288.83
10 /60     2.877   98.667   2.70821  98.214   0.00100  0.00100  val_loss     9.92    294.33
11 /60     2.637   98.778   2.48928  99.107   0.00100  0.00100  val_loss     8.08    288.93
12 /60     2.392   99.222   2.27085  98.214   0.00100  0.00100  val_loss     8.77    286.70
13 /60     2.204   99.222   2.10370  100.000  0.00100  0.00100  val_loss     7.36    291.32
14 /60     2.057   98.667   1.92789  99.107   0.00100  0.00100  val_loss     8.36    288.37
15 /60     1.880   99.000   1.76667  99.107   0.00100  0.00100  val_loss     8.36    287.31
enter H to halt training or an integer for number of epochs to run then ask again
H
training has been halted at epoch 15 due to user input
training elapsed time was 1.0 hours, 26.0 minutes, 55.61 seconds)
```

At the end of 15th epoch, Accuracy is 99% on training set and 99.107% on validation set. Final Validation loss is 1.766. Best epoch recorded is epoch 9.

Graph plotted on train and validation loss:



Graph plotted on Train and Validation accuracies:



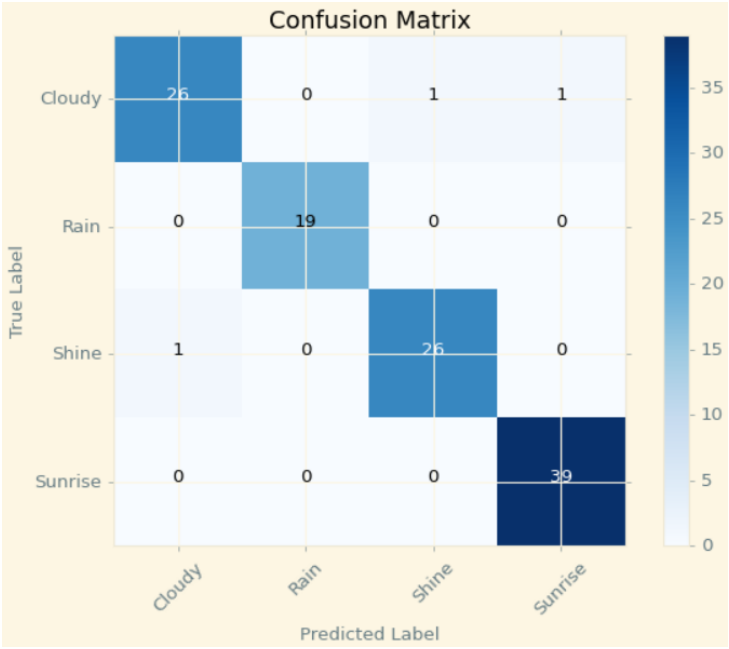
Trained model is evaluated on all three of the datasets and the results are as follows:

```
Train Loss:  1.742717981338501
Train Accuracy:  1.0
--------------------
Validation Loss:  1.7666683197021484
Validation Accuracy:  0.9910714030265808
--------------------
Test Loss:  1.7924755811691284
Test Accuracy:  0.9734513163566589
```

Evaluation

Like any classification problem, in this project we use confusion matrix, precision, recall evaluating the trained model.

Confusion matrix for the four classes on the test set is as follows:



Precision, recall and F1 score for each class on the test set is as follows:

	precision	recall	f1-score	support
Cloudy	0.96	0.93	0.95	28
Rain	1.00	1.00	1.00	19
Shine	0.96	0.96	0.96	27
Sunrise	0.97	1.00	0.99	39
accuracy			0.97	113
macro avg	0.98	0.97	0.97	113
weighted avg	0.97	0.97	0.97	113

Result Analysis

The report displays the precision, recall, F1-score, and support metrics for each class, as well as the overall macro and weighted average. The accuracy metric counts how many of the projected samples for a class are true positive (TP) samples, indicating how accurate the model is when predicting a specific class. Recall, on the other hand, measures how many actual positive (TP + false negative (FN)) samples the model properly predicts. The F1-score is a balanced measure that takes the harmonic mean of precision and recall. Finally, the number of samples in the dataset that belong to each class is represented by support.

Looking at the report, we can see that the model's overall accuracy is 0.97 or 97%. This means that out of the 113 samples in the dataset, the model correctly predicted the class of 110 samples. We can also see that the model performed very well for all the classes. For Cloudy, the model achieved a precision of 0.96, meaning that 96% of the predicted Cloudy samples

were actually Cloudy. The recall for Cloudy was 0.93, indicating that the model correctly predicted 93% of the actual Cloudy samples in the dataset. The F1-score for Cloudy was 0.95, which is a good balance between precision and recall. The model had a precision and recall of 1.0 for Rain, suggesting that it accurately predicted all of the Rain samples in the dataset. Rain's F1-score was also 1.0, indicating flawless performance. The model correctly predicted 96% of the actual Shine samples in the dataset with a precision of 0.96 and a recall of 0.96 for Shine. Shine's F1-score was 0.96, indicating an excellent mix of precision and recall.

Finally, the model had a precision of 0.97 and a recall of 1.0 for Sunrise, suggesting that it accurately predicted 97% of the predicted Sunrise samples and all of the actual Sunrise samples in the dataset. Sunrise's F1-score was 0.99, suggesting that the model performed admirably for all classes.

Conclusion

In conclusion, developing a deep learning model for weather classification can yield valuable insights for weather monitoring and forecast systems. We started with the pre-trained EfficientNetB0 model and added additional layers to fine-tune the model for weather categorization in this project. We also used data augmentation techniques to expand the dataset and improve the model's generalization capability. The resulting model has a high accuracy rate, confirming the efficacy of the proposed method.

This project can be expanded in the future by combining other weather conditions or by putting real-time weather data into the model for more precise predictions. Other deep learning techniques, such as transfer learning and ensemble learning, can also be investigated to increase the model's performance. Overall, this effort proves the use of deep learning models for weather classification and has implications for agriculture, transportation, and renewable energy. We can extend the project using pre-trained models, which may be more efficient than a neural network models.

References

Sharma, A., Kumar, V., & Bhattacharya, A. (2018). A convolutional neural network-based approach for weather classification from satellite images. Geocarto International, 33(11), 1179-1190.

Mazzarella, F., Boccia, L., & Riccio, D. (2018). A deep learning approach to weather identification from ground-level camera images. Journal of Atmospheric and Solar-Terrestrial Physics, 175, 73-81.

Chandola, C., Sharma, R., & Kumar, A. (2016). Weather classification using machine learning algorithms. In Proceedings of the International Conference on Advances in Computing, Communications and Informatics (pp. 1688-1692). IEEE.

Ghosh, S., Das, S., & Bhowmick, P. (2018). Machine learning approaches for weather prediction: a review. In Proceedings of the 2nd International Conference on Inventive Communication and Computational Technologies (pp. 1321-1325). IEEE.

Cao, Y., Chen, J., & Liu, J. (2020). Weather recognition of ground-level camera images based on deep learning. In Proceedings of the 2nd International Conference on Computing and Artificial Intelligence (pp. 16-20). ACM.

Song, H., Wu, Y., & Han, J. (2019). A robust weather classification method using convolutional neural network. In Proceedings of the IEEE International Conference on Multimedia and Expo (pp. 290-295). IEEE.

Tan, M., & Le, Q. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In Proceedings of the 36th International Conference on Machine Learning (ICML). http://proceedings.mlr.press/v97/tan19a/tan19a.pdf