King Saud University

College of Computer and Information Sciences

Computer Science Department

CSC462

Multi-class Weather Image Recognition Using CNN (VGG16) Model

2023 – 1445

First term

Name	ID	Section #
Lama Al-Abdulkarim	442200432	44277
Hessah Al-Dhubaib	44201355	
Dalal Al-Sadoun	44201417	

Table of Content:

1. Abstract

This study outlines developing and implementing a Convolutional Neural Network (CNN) model, specifically leveraging the VGG16 architecture, to classify images into various weather phenomena. We used "The Weather Phenomenon Database (WEAPD)" to train and test the model. The model achieved notable accuracy in classifying diverse weather conditions through careful hyperparameter experimentation and employing 5-fold cross-validation, achieving an accuracy of 88%.

2. Introduction

Computer vision is a subset of AI and machine learning. Image classification, a technique used in computer vision, trains machines to categorize pixels into predefined classes. Deep learning has been instrumental in image classification, specifically using CNNs. Image classification has provided insightful analysis in various fields. Our project focuses on meteorology, using image classification to enhance weather analysis through automated interpretation of weather images. By leveraging the capabilities of CNNs, we aim to develop a model that can identify and categorize different weather conditions based on visual cues in images, providing valuable insights for meteorological studies and practical applications in various sectors affected by weather conditions.

3. Problem Statement

Extreme weather conditions always pose potential risks to people's lives and property. Therefore, the automatic recognition of extreme weather plays a crucial role in various applications, such as highway traffic condition warnings [1] and automobile auxiliary driving [2]. Old methods of weather recognition mainly relied on multiple sensors, which required a significant workforce and material consumption for installation and maintenance, which may result in low accuracy. Additionally, these methods only use a small dataset for training and recognition, resulting in a lack of robustness. To address these challenges, new methods for weather recognition based on computer vision techniques have been proposed using deep learning.

In this report, we will first introduce the dataset in (Section 4). Then, we will review papers that have utilized computer vision and convolutional neural networks (CNN) to recognize different weather conditions (Section 5). Next, we will discuss the pre-processing methods that our model employs (Section 6). Following that, we will go over the methodology that we have used (Section 7). Finally, we will present our results (Section 8). The strengths, limitations, and future improvements are discussed in (Section 9) and (Section 10), respectively. All the code and outputs were provided in (Section 11).

4. Dataset

In this project, we will use a weather Harvard dataverse dataset to develop a multi-class CNN-based model. This dataset contains 6,877 images of different types of weather phenomena, it can be used to implement different classification models when receiving an image that showcases some weather phenomenon. The dataset is divided into 11 classes: dew, fog/smog, frost, glaze, hail, lightning, rain, rainbow, rime, sandstorm, and snow [3].

5. Literature Review

Zhu et al. [4] introduced a novel approach for recognizing extreme weather conditions through computer vision using CNN. The reason they chose CNN is that the weather is influenced by several factors, making it challenging to extract features that can precisely represent different weather characteristics. CNN architecture (Fig.1) consists of a sequence of convolution and downsampling layers that are applied alternately to input images. This results in a hierarchical structure of increasingly complex features. Finally, one or more fully connected layers are added to this structure.

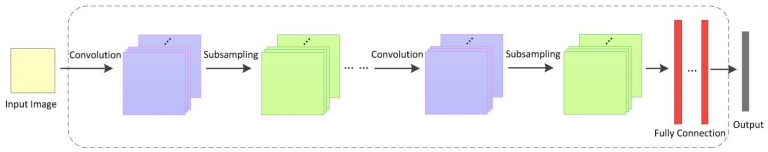


Fig.1: CNN architecture [4].

In addition, the authors collected a large-scale weather dataset named "WeatherDataset". The dataset includes images collected from different sources such as aerial photography, cameras, traffic accidents, and automobile data recorders. The dataset comprises 16,635 images divided into four weather types: sunny, rainstorm, blizzard, and fog. The dataset also covers complex scenes such as urban roads and highways. The training process consists of two key steps: pre-training and fine-tuning. They used GoogLeNet to achieve weather recognition. GoogLeNet is a deep CNN, which has 22 layers [5]. During the pre-training phase, the imageNet large-scale visual recognition challenge (ILSVRC-2012) dataset [6] trains the GoogLeNet model for ILSVRC. The model is then fine-tuned on the WeatherDataset to improve its accuracy in recognizing extreme weather conditions. For the training set, images are randomly selected from the WeatherDataset and scaled to 224×224 before inputting them into the GoogLeNet. To evaluate the effectiveness of the proposed method, experiments have been conducted on three different network structures: AlexNet [7] and the modified AlexNet, which are both considered as CNNs, along with the GoogLeNet. All three network models are pre-trained on the ILSVRC-2012 dataset. The recognition results of GoogLeNet are then compared with those of AlexNet and the modified AlexNet to determine its effectiveness. GoogLeNet can obtain the best recognition accuracy of 94.5% with fine-tuning.

In the paper [8], a weather image recognition model was introduced, categorizing images into four classes: hazy, rainy, snowy, and an additional category for images not belonging to these classes. These categories were initially proposed in [9], which used a dataset of 20,000 outdoor images known as a multi-class weather image (MWI) dataset [9]. To accomplish this task, the authors employed GoogLeNet, a CNN with 22 layers, as mentioned before –27 layers when counting pooling layers–[5]. The inception model, inspired by the paper in [10], is the core of GoogLeNet and is designed to improve the representational power of neural networks. The authors made some modifications to GoogLeNet to adjust the output size of the network to four, so it matches the four class weather classifications, whereas the original had 1,000 image classes. To evaluate the performance of their model, the authors compared it with existing approaches such as AlexNet, the winner of the competition ILSVRC2012 [6]. The GoogLeNet accuracy slightly outperformed AlexNet with 92.0%. Moreover, the authors compared GoogLeNet with another existing approach, MKL-based, which heavily relied on manual feature extraction, and the result showed that GoogLeNet significantly outperformed MKL-based.

A novel deep CNN named MeteCNN was proposed in [11] for weather phenomena classification. The authors created their own dataset –"The Weather Phenomenon Database" (WEAPD) [3]. MeteCNN model has thirteen convolutional layers (Fig. 2), six pooling layers, five of which were max-pooling layers, and one was a global average pooling layer instead of a fully connected layer.

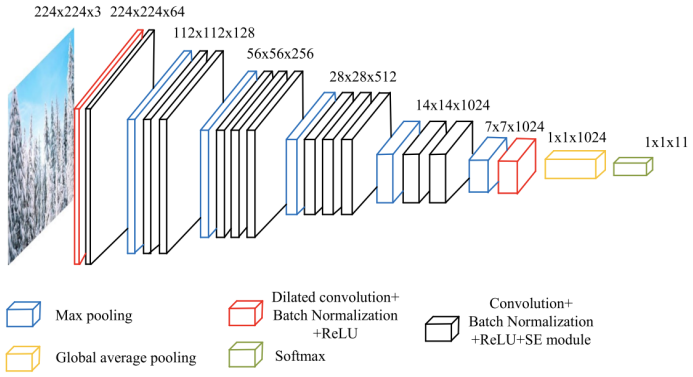


Fig 2: MeteCNN architecture [11]

In the convolutional layers, the authors used a squeeze and excitation module (SE) to improve the representational power of CNNs. Additionally, dilated convolution is also used to introduce gaps between the values in a filter, which enlarges the receptive field. The SE module, batch normalization, and the rectified linear units (ReLU) function are combined in all convolution layers except the first and last, where dilated convolution is used instead of SE. MeteCNN receives a batch of fixed-size weather phenomenon images as input. The convolutional layers function as a feature extractor, with each layer generating an array of feature maps. Following batch normalization, the ReLU function performs non-linear mapping of the convolutional layer output results. The pooling layers help reduce training parameters. The convolutional layer's local feature outputs are combined into global features by the global average pooling layer. Finally, the feature map class probability is computed by the softmax layer. The classification accuracy of the MeteCNN model on the weather phenomena is above 84%.

6. Preprocessing

Preprocessing steps are used before feeding the dataset into the VGG16 model to ensure the data is in a suitable format for effective training and feature extraction. First, we resized the images to 224×224 pixels in size. Resizing images to a standard size ensures that they all have the same dimensions, which is necessary for the images to fit into the VGG16 model. The pixel values of the resized images can be normalized using Z-score normalization to scale the pixel values. This normalization procedure is critical for stabilizing and accelerating the training procedure, it rescales data so that they have the properties of a standard normal distribution with $\mu = 0$ and $\sigma = 1$. The model's convergence during training is optimized by ensuring uniform scales for the features via Z-score normalization, allowing faster and more accurate learning [12].

7. Methodology

7.1 VGG16 Model

We will employ a CNN architecture known as VGG16 to perform image classification on a dataset of weather images. The choice of VGG16 was motivated by its widespread use and high performance in various computer vision tasks. Furthermore, the model was pre-trained on a massive dataset, which enabled it to extract meaningful features from the weather images with minimal training. The VGG16 comprises 16 layers, including 13 convolutional layers and 3 fully connected layers. The convolutional layers extract features from the input image, while the fully connected layers are used for classification. The convolutional layers are arranged in blocks, each containing two or three convolutional layers followed by a max pooling layer. The output of the final convolutional layer is flattened and fed into the fully connected layers, which make the final classification decision. The model is trained using a variant of stochastic gradient descent (SGD), where the weights are updated iteratively to minimize the classification error. The model is also regularized using dropout to prevent overfitting on the training dataset [13].

8. Experimental Results

We conducted systematic experiments on various hyperparameters, including learning rates and neuron counts in additional dense layers, as illustrated in Table 1, to optimize the model's performance. To ensure a comprehensive and fair evaluation of the model's ability to handle class imbalances, we utilized multiple evaluation metrics such as accuracy, precision, recall, and F1-Score. We employed a 5-fold cross-validation methodology for training and testing the model. After analyzing the results presented in Table 2, we determined that the optimal learning rate for the model is 0.01, and the optimal number of neurons is 1024. We trained the model again using the best learning rate and number of neurons for a total of 30 epochs, achieving an accuracy of 88%, as shown in Table 3.

Table 1: Experiments with different hyperparameters.

	Number of epochs	**Learning rate**	**Number of neurons**
Case1	10	0.01	512
Case2	10	0.01	1024
Case3	10	0.001	512
Case4	10	0.001	1024
Case5	10	0.0001	512
Case6	10	0.0001	1024

Table 2: Results of Accuracy for 5-Fold Cross-Validation.

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Case 1	84%	87%	84%	86%	85.5%
Case 2	87.5%	87.1%	86.2%	86%	86.5%
Case 3	82.1%	84.2%	83%	83.8%	82.8%
Case 4	84.5%	84.9%	84.1%	84.6%	82.5%
Case 5	61.5%	60.7%	61.5%	60.6%	60.5%
Case 6	64.3%	64%	61.2%	66.6%	63.5%

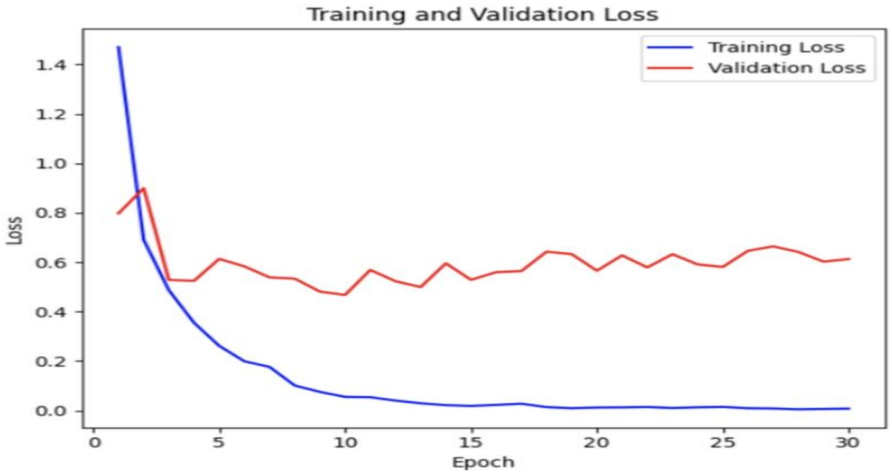


Fig 3: The training and validation loss.

Fig 4: the training and validation accuracy.

Table 3: Evaluation Metrics Results for Different Classes.

	Precision	Recall	F1-score	Support
dew	0.94	0.95	0.95	140
fogsmog	0.87	0.92	0.90	168
frost	0.81	0.77	0.79	96
glaze	0.81	0.75	0.78	128
hail	0.93	0.97	0.95	116
lightning	1.00	0.99	0.99	75
rain	0.86	0.92	0.89	104
rainbow	0.94	0.94	0.94	48
rime	0.85	0.88	0.86	232
sandstorm	0.94	0.86	0.90	136
snow	0.78	0.73	0.75	122
accuracy			0.88	1365
macro avg	0.88	0.88	0.88	1365
Weighted avg	0.88	0.88	0.88	1365

9. Strengths:

The proposed approach carefully preprocessed the data to ensure its quality and used the WEAPD for thorough representation of weather conditions. It applied transfer learning with the VGG16 model from ImageNet to capture intricate features from images and accelerate the training process. The model was systematically tuned with different learning rates and neuron counts, along with K-Fold Cross-Validation. We employed multiple Comprehensive evaluation metrics.

10. Limitations and Potential Future Improvements:

The current approach for weather phenomenon classification is limited by the imbalanced class distribution across different weather conditions, which poses a significant challenge. Additionally, the fixed image size during resizing may lead to information loss or distortion, adding to the problem. To overcome these limitations and enhance the proposed approach, it is recommended to experiment with advanced models such as ResNet, Inception, or EfficientNet. Furthermore, employing ensemble learning by combining multiple models with different architectures can further improve overall performance and robustness, resulting in a more precise and resilient weather phenomenon classification model.

11. Code:

```
import os
from PIL import Image
import shutil
from sklearn.model_selection import train_test_split
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import KFold
import warnings
warnings.filterwarnings("ignore")
```

```
def resize_images_in_folder(main_folder_path, new_size=(224, 224)):
    #for each subdirectory in the directory specified 'main_folder_path'
    for subdir in os.listdir(main_folder_path):
        #create a path for the subdirectory
        subdir_path = os.path.join(main_folder_path, subdir)
        #check if the location we are in is a directory
        if os.path.isdir(subdir_path):
            #for each file in the subdirectory:
            for file in os.listdir(subdir_path):
                #check the extentions of the image files just incase
                if file.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.gif')):
                    #create a path for the file
                    file_path = os.path.join(subdir_path, file)
                    #try block to open, resize, and save the images
                    try:
                        #opens the file in the file_path
                        img = Image.open(file_path)
                        #resizes the opened file
                        img = img.resize(new_size, Image.ANTIALIAS)
                        #save the resized images
                        img.save(file_path)
                    #any file that causes any issues will be deleted
                    except Exception as e:
                        os.remove(file_path)
```

```
#find the path of the WEAPD dataset:
main_folder_path ='/Users/hessahaldhubaib/Desktop/Machine Learning/ML_Project/dataset(WEAPD)'
#call the resize_images_in_folder
resize_images_in_folder(main_folder_path)
```

```
# We defined the paths of the train, test, and validation files
source_dir = '/Users/hessahaldhubaib/Desktop/Machine Learning/ML_Project/dataset(WEAPD)'
train_dir = '/Users/hessahaldhubaib/Desktop/Machine Learning/ML_Project/Data/train'
test_dir = '/Users/hessahaldhubaib/Desktop/Machine Learning/ML_Project/Data/test'
val_dir = '/Users/hessahaldhubaib/Desktop/Machine Learning/ML_Project/Data/val'
```

```
def count_files(startpath):
    file_count = 0
    for root, dirs, files in os.walk(startpath):
        file_count += len(files)
    return file_count

# Set the path to the folder
folder_path = '/Users/hessahaldhubaib/Desktop/Machine Learning/ML_Project/dataset(WEAPD)'

# Get the count of files in the folder and its subfolders
num_files = count_files(folder_path)

# Print the result
print(f'The number of files in the folder and its subfolders is: {num_files}')
```

```
The number of files in the folder and its subfolders is: 6786
```

```python
# Create destination directories with the paths that we defined
#for each of the paths:
for dir in [train_dir, test_dir, val_dir]:
    #Check if dir doesn't exist.
    #The os.path.exists function returns True if the path exists, so we added the 'not'
    if not os.path.exists(dir):
        # if the dir doesn't exist, we make it.
        os.makedirs(dir)
```

```python
# Get list of class directories
classes = os.listdir(source_dir)
#prints the classes
print(classes)

for cls in classes:
    if '.DS_Store' in cls:
      continue
    # Create subdirectories in train, test, and val directories
    # 'exist_ok' ensures that the directories are created even if they already exist.
    os.makedirs(os.path.join(train_dir, cls), exist_ok=True)
    os.makedirs(os.path.join(test_dir, cls), exist_ok=True)
    os.makedirs(os.path.join(val_dir, cls), exist_ok=True)

    # Get a list of images in each class
    images = os.listdir(os.path.join(source_dir, cls))

    # Split the images into train, test, and validation sets using the train_test_split function from scikit-learn.
    train_imgs, test_imgs = train_test_split(images, test_size=0.3, random_state=42)
    val_imgs, test_imgs = train_test_split(test_imgs, test_size=(2/3), random_state=42)

    # Function to copy files. Takes the file (image), source, and destination.
    # It copies the images into their respective training, testing, and validation directories.
    def copy_files(files, source, destination):
        for file in files:
            src_path = os.path.join(source, file)
            dst_path = os.path.join(destination, file)
            shutil.copy(src_path, dst_path)

    # Copy images to their respective directories
    copy_files(train_imgs, os.path.join(source_dir, cls), os.path.join(train_dir, cls))
    copy_files(test_imgs, os.path.join(source_dir, cls), os.path.join(test_dir, cls))
    copy_files(val_imgs, os.path.join(source_dir, cls), os.path.join(val_dir, cls))
```

```
['lightning', 'sandstorm', 'glaze', 'rain', '.DS_Store', 'rime', 'frost', 'fogsmog', 'hail', 'dew', 'rainbow', 'snow']
```

```
data_dir = '/Users/hessahaldhubaib/Desktop/Machine Learning/ML_Project/Data'
train_data_dir = os.path.join(data_dir, 'train')
val_data_dir = os.path.join(data_dir, 'val')
best_model_path = '/Users/hessahaldhubaib/Desktop/Machine Learning/ML_Project/best_model'
```

```
def prepare_data_generators(data_dir, val_data_dir, batch_size=32):
    #scale pixel values to the range [0, 1]
    #'datagen' is then used to create data generators for training and validation data
    datagen = ImageDataGenerator(rescale=1./255)

    #A generator for training data
    #It will load and preprocess images from the specified 'data_dir'
    #It is configured with a target image size of (224, 224)
    train_generator = datagen.flow_from_directory(
        data_dir,
        target_size=(224, 224),
        batch_size=batch_size,
        class_mode='categorical')

    val_generator = datagen.flow_from_directory(
        val_data_dir,
        target_size=(224, 224),
        batch_size=batch_size,
        class_mode='categorical')

    return train_generator, val_generator
```

```
def create_model(learning_rate, num_neurons):
    # Load the pre-trained VGG16 model
    base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

    # Freeze the layers except the last 4 layers
    for layer in base_model.layers[:-4]:
        layer.trainable = False

    # Add custom layers
    x = base_model.output
    x = Flatten()(x)
    x = Dense(num_neurons, activation='relu')(x)
    x = Dropout(0.5)(x)
    predictions = Dense(11, activation='softmax')(x)  # Assuming 11 classes

    # Define the model
    model = Model(inputs=base_model.input, outputs=predictions)

    # Compile the model
    model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=learning_rate),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

```python
# use diffrent lr and num_neurons and apply k-fold
# Hyperparameter space
learning_rates = [0.01, 0.001, 0.0001]
num_neurons = [512, 1024]

num_folds = 5
best_accuracy = 0
best_lr = 0
best_neuron = 0

for lr in learning_rates:
    for neuron in num_neurons:
        fold_accuracies = []

        for fold in range(num_folds):
            print(f"Training on fold {fold + 1}/{num_folds}...")

            # Prepare data generators for the current fold
            train_generator, val_generator = prepare_data_generators(train_data_dir, val_data_dir)

            # Create the model
            model = create_model(lr, neuron)

            # Train the model
            history = model.fit(
                train_generator,
                epochs=10,
                validation_data=val_generator
            )

            # Evaluate the model
            accuracy = model.evaluate(val_generator)[1]
            fold_accuracies.append(accuracy)
            print("lr", lr, "neuron", neuron, "fold", fold, "accuracy", accuracy)

        # Average accuracy for current hyperparameters
        avg_accuracy = np.mean(fold_accuracies)
        print("avg_accuracy", avg_accuracy)

        if avg_accuracy > best_accuracy:
            best_accuracy = avg_accuracy
            best_lr = lr
            best_neuron = neuron
            model.save(best_model_path)

print(f'Best Accuracy: {best_accuracy}, Best Learning Rate: {best_lr}, Best Number of Neurons: {best_neuron}')
```

```
Epoch 8/10
149/149 [==============================] - 28s 190ms/step - loss: 1.3988 - accuracy: 0.5359 - val_loss: 1.2339 - val_accuracy: 0.6169
Epoch 9/10
149/149 [==============================] - 28s 189ms/step - loss: 1.3009 - accuracy: 0.5625 - val_loss: 1.1598 - val_accuracy: 0.6361
Epoch 10/10
149/149 [==============================] - 26s 175ms/step - loss: 1.2419 - accuracy: 0.5825 - val_loss: 1.0962 - val_accuracy: 0.6627
22/22 [==============================] - 3s 122ms/step - loss: 1.0962 - accuracy: 0.6627
lr 0.0001 neuron 1024 fold 3 accuracy 0.6627218723297119
Training on fold 5/5...
Found 4745 images belonging to 11 classes.
Found 676 images belonging to 11 classes.
Epoch 1/10
149/149 [==============================] - 27s 174ms/step - loss: 2.4669 - accuracy: 0.1429 - val_loss: 2.1671 - val_accuracy: 0.2322
Epoch 2/10
149/149 [==============================] - 26s 172ms/step - loss: 2.2303 - accuracy: 0.2341 - val_loss: 2.0009 - val_accuracy: 0.3491
Epoch 3/10
149/149 [==============================] - 28s 187ms/step - loss: 2.0521 - accuracy: 0.3009 - val_loss: 1.8618 - val_accuracy: 0.3935
Epoch 4/10
149/149 [==============================] - 28s 186ms/step - loss: 1.8998 - accuracy: 0.3591 - val_loss: 1.7240 - val_accuracy: 0.4630
Epoch 5/10
149/149 [==============================] - 26s 170ms/step - loss: 1.7589 - accuracy: 0.4110 - val_loss: 1.5940 - val_accuracy: 0.5104
Epoch 6/10
149/149 [==============================] - 28s 187ms/step - loss: 1.6430 - accuracy: 0.4516 - val_loss: 1.4844 - val_accuracy: 0.5370
Epoch 7/10
149/149 [==============================] - 26s 175ms/step - loss: 1.5429 - accuracy: 0.4778 - val_loss: 1.3865 - val_accuracy: 0.5695
Epoch 8/10
149/149 [==============================] - 26s 175ms/step - loss: 1.4507 - accuracy: 0.5062 - val_loss: 1.2985 - val_accuracy: 0.5932
Epoch 9/10
149/149 [==============================] - 26s 174ms/step - loss: 1.3565 - accuracy: 0.5458 - val_loss: 1.2236 - val_accuracy: 0.6036
Epoch 10/10
149/149 [==============================] - 26s 173ms/step - loss: 1.2973 - accuracy: 0.5572 - val_loss: 1.1601 - val_accuracy: 0.6302
22/22 [==============================] - 3s 123ms/step - loss: 1.1601 - accuracy: 0.6302
lr 0.0001 neuron 1024 fold 4 accuracy 0.6301774978637695
avg_accuracy 0.6378698229789734
Best Accuracy: 0.8653846144676208, Best Learning Rate: 0.01, Best Number of Neurons: 1024
```

```python
#Now we can train the model using lr=0.01 and neuron=1024
#We also use number of ebochs=30
lr=0.01
neuron=1024
num_ebochs=30

train_generator, val_generator = prepare_data_generators(train_data_dir, val_data_dir)

# Create the model
model = create_model(lr, neuron)

# Train the model
history = model.fit(
    train_generator,
    epochs=num_ebochs,
    validation_data=val_generator
)

# Evaluate the model
accuracy = model.evaluate(val_generator)[1]
model.save(best_model_path)
print("accuracy", accuracy)
```

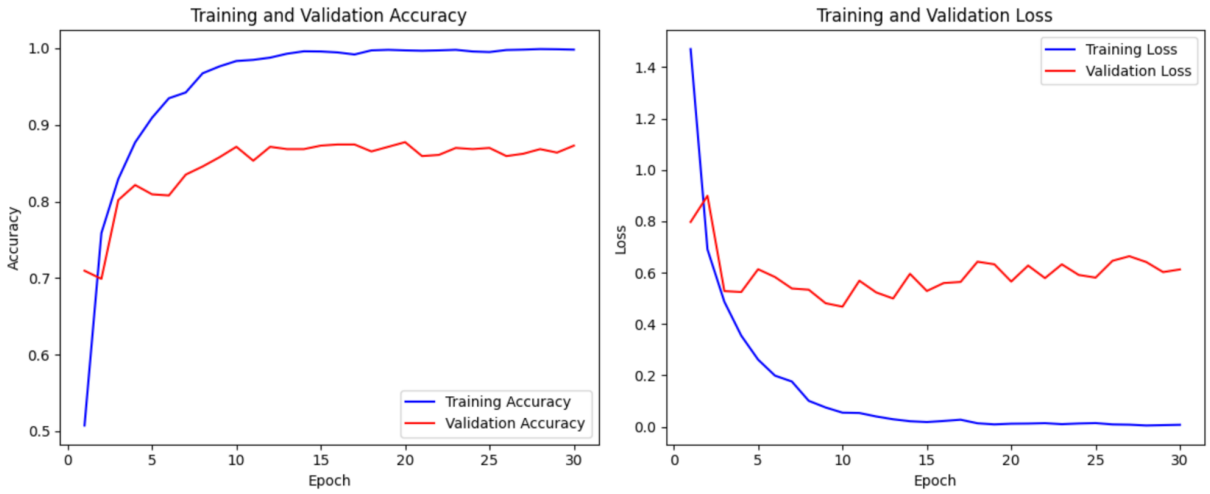
```
129/129 [==============================] - 24s 186ms/step - loss: 0.0295 - accuracy: 0.9929 - val_loss: 0.4997 - val_accuracy: 0.8684
Epoch 14/30
129/129 [==============================] - 25s 190ms/step - loss: 0.0218 - accuracy: 0.9961 - val_loss: 0.5956 - val_accuracy: 0.8684
Epoch 15/30
129/129 [==============================] - 24s 187ms/step - loss: 0.0186 - accuracy: 0.9959 - val_loss: 0.5290 - val_accuracy: 0.8729
Epoch 16/30
129/129 [==============================] - 24s 189ms/step - loss: 0.0228 - accuracy: 0.9946 - val_loss: 0.5599 - val_accuracy: 0.8744
Epoch 17/30
129/129 [==============================] - 27s 207ms/step - loss: 0.0277 - accuracy: 0.9920 - val_loss: 0.5642 - val_accuracy: 0.8744
Epoch 18/30
129/129 [==============================] - 24s 188ms/step - loss: 0.0138 - accuracy: 0.9973 - val_loss: 0.6429 - val_accuracy: 0.8654
Epoch 19/30
129/129 [==============================] - 24s 187ms/step - loss: 0.0095 - accuracy: 0.9981 - val_loss: 0.6329 - val_accuracy: 0.8714
Epoch 20/30
129/129 [==============================] - 24s 187ms/step - loss: 0.0122 - accuracy: 0.9973 - val_loss: 0.5658 - val_accuracy: 0.8775
Epoch 21/30
129/129 [==============================] - 25s 189ms/step - loss: 0.0128 - accuracy: 0.9968 - val_loss: 0.6279 - val_accuracy: 0.8593
Epoch 22/30
129/129 [==============================] - 25s 190ms/step - loss: 0.0143 - accuracy: 0.9973 - val_loss: 0.5791 - val_accuracy: 0.8608
Epoch 23/30
129/129 [==============================] - 25s 190ms/step - loss: 0.0104 - accuracy: 0.9981 - val_loss: 0.6326 - val_accuracy: 0.8699
Epoch 24/30
129/129 [==============================] - 24s 189ms/step - loss: 0.0131 - accuracy: 0.9959 - val_loss: 0.5914 - val_accuracy: 0.8684
Epoch 25/30
129/129 [==============================] - 25s 190ms/step - loss: 0.0146 - accuracy: 0.9951 - val_loss: 0.5811 - val_accuracy: 0.8699
```

```python
import matplotlib.pyplot as plt
# Extracting the metrics
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
train_loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, num_ebochs + 1)

# Plotting training and validation accuracy
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs, train_acc, 'b', label='Training Accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Plotting training and validation loss
plt.subplot(1, 2, 2)
plt.plot(epochs, train_loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'r', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```



```
best_model = tf.keras.models.load_model(best_model_path)

test_data_dir = os.path.join(data_dir, '/Users/hessahaldhubaib/Desktop/Machine Learning/ML_Project/Data/test')

test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_directory(
    test_data_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    shuffle=False)  # Set shuffle to False for consistent results
```

```
ound 1365 images belonging to 11 classes.
```

```
# Make predictions
predictions = best_model.predict(test_generator)
predicted_classes = np.argmax(predictions, axis=1)
```

```
3/43 [==============================] - 394s 9s/step
```

```
true_classes = test_generator.classes
class_labels = list(test_generator.class_indices.keys())
```

```
from sklearn.metrics import classification_report

print(classification_report(true_classes, predicted_classes, target_names=class_labels))
```

```
              precision    recall  f1-score   support

         dew       0.94      0.95      0.95       140
     fogsmog       0.87      0.92      0.90       168
       frost       0.81      0.77      0.79        96
       glaze       0.81      0.75      0.78       128
        hail       0.93      0.97      0.95       116
   lightning       1.00      0.99      0.99        75
        rain       0.86      0.92      0.89       104
     rainbow       0.94      0.94      0.94        48
        rime       0.85      0.88      0.86       232
   sandstorm       0.94      0.86      0.90       136
        snow       0.78      0.73      0.75       122

    accuracy                           0.88      1365
   macro avg       0.88      0.88      0.88      1365
weighted avg       0.88      0.88      0.88      1365
```

References

[1] C. Oh, J.-S. Oh, and S. G. Ritchie, “Real-Time Hazardous Traffic Condition Warning System: Framework and Evaluation,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 6, no. 3, pp. 265–272, Sep. 2005.

[2] Z. Liu, Y. Cai, H. Wang, L. Chen, H. Gao, Y. Jia, and Y. Li, "Robust Target Recognition and Tracking of Self-Driving Cars With Radar and Camera Information Fusion Under Severe Weather Conditions," in IEEE Transactions on Intelligent Transportation Systems, vol. 23, no. 7, pp. 6640-6653, Jul. 2022.

[3] H. Xiao, F. Zhang, Z. Shen, K. Wu, and J. Zhang, "The Weather Phenomenon Database (WEAPD)." November 10, 2021. Distributed by Harvard Dataverse. https://doi.org/10.7910/DVN/M8JQCR.

[4] Z. Zhu, L. Zhuo, P. Qu, K. Zhou and J. Zhang, "Extreme Weather Recognition Using Convolutional Neural Networks," *2016 IEEE International Symposium on Multimedia (ISM)*, San Jose, CA, USA, Dec. 11-13, 2016, pp. 621-625.

[5] C. Szegedy, W. Liu, Y, Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, “Going Deeper with Convolutions, “ in *IEEE Conference Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA, Jun. 07-12, 2015, pp. 1-9.

[6] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," IEEE International Journal of Computer Vision, vol. 115, pp. 221-252, 2015.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in Advances in Neural Information Processing Systems, 2012, pp. 1097-1105.

[8] L. W. Kang, K. L. Chou and R. H. Fu, "Deep Learning-based Weather Image Recognition, " in *International Symposium on Computer, Consumer and Control (IS3C)*, Taichung, Taiwan, Dec. 06-08, 2018, pp. 348-387

[9] Z. Zhang and H. Ma, "Multi-class weather classification on single images," *in IEEE International Conference Image Processing (ICIP)*, Quebec, Canada, Sep. 27-30, 2015, pp. 4396-4400.

[10] M. Lin, Q. Chen and S. Yan, "Network in Network, " in *International Conference on Learning Representation (ICLR)*, Banff, Canada, April 14-16, 2014.

[11] H. Xiao, F. Zhang, Z. Shen, K. Wu, and J. Zhang, "Classification of Weather Phenomenon From Images by Using Deep Convolutional Neural Network," *Earth and Space Science*, vol. 8, no. 5, May 2021.

[12] A. Burkov, " The Hundred-Page Machine Learning, " in eBook, Quebec, Canada, 2019.

[13] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *International Conference on Learning Representation (ICLR)*, San Diego, CA, USA, May 07-09, 2015.

Appendix

Table of distribution:

Name	Work distribution	
Lama Al-Abdulkarim	Paper[8]	The rest was done over Zoom (including the code), but we left the run until it finished.
Hessah Al-Dhubaib	Paper [11]	
Dalal Al-Sadoun	Paper [4]	