

ABSTRACT

Weather condition classification using CNN project develops a system for classifying weather conditions from images using convolutional neural networks (CNNs). By leveraging deep learning techniques, the system can identify whether the weather is sunny, cloudy, or rainy based on visual data. The process involves collecting weather-related images, pre-processing them, constructing and training a CNN model, and using the model to make predictions on new images. This approach demonstrates the application of CNNs in practical image classification tasks and provides a tool for automated weather forecasting.

Keywords: Weather Classification, Convolutional Neural Network (CNN), Image Processing, Deep Learning, TensorFlow, Data Augmentation, Model Training and Weather Forecasting.

CONTENTS

Abstract.....	1
Introduction.....	4
1. Existing Solution For Weather Classification.....	5
1.1 Traditional Machine Learning Methods.....	5
1.2 Shallow Neural Networks.....	7
1.3 Deep Learning Approaches.....	8
2. Algorithm Used In The Project.....	10
2.1 Why use Convolutional Neural Network (CNNs)?.....	10
2.2 Data flow diagram for the project.....	11
3. Building Phases Of The Project.....	14
3.1 Week 1: Data Collection and Preprocessing.....	14
3.2 Week 2: Model Construction.....	15
3.3 Week 3: Model Training.....	16
3.4 Week 4: Prediction and Forecasting.....	16
4. Literature Survey.....	18
4.1 Foundations of Convolutional Neural Networks (CNNs).....	18
4.2 Advancements in CNN Architectures.....	18
4.3 CNNs in Weather Classification.....	19
4.4 Data Preprocessing and Augmentation.....	19
4.5 Training and Optimization of CNN Models.....	20
4.6 Practical Implementations and Real-World Applications.....	20
5. Implementation Of The Code.....	21
5.1 Explanation of the code.....	21
5.2 Complete code.....	26
6. Results.....	30

7. Comparing And Contrasting With Existing Solutions.....	33
7.1 Comparison with Super Vector Machines (SVM(s)).....	33
7.2 Comparison with Decision Trees and Random Forests.....	33
7.3 Comparison with Multi-Layer Perceptrons (MLP(s)).....	34
8. Conclusion And Future Work.....	36
8.1 Limitations.....	36
8.2 Future Work.....	37
REFERENCES.....	39

INTRODUCTION

Weather forecasting traditionally relies on meteorological models that use data from weather stations, satellites, and sensors. These models are complex and require significant computational resources. With advancements in deep learning, it is now possible to predict weather conditions using image data. This project demonstrates how a convolutional neural network can be trained to recognize weather patterns from images, offering a novel approach to weather forecasting. By leveraging image data, the system provides an efficient and scalable method to predict weather conditions, which can be integrated into various applications.

CHAPTER 1

EXISTING SOLUTION FOR WEATHER CLASSIFICATION

Existing weather prediction methods include:

1.1 Traditional Machine Learning Methods

a. Support Vector Machines (SVMs): Support Vector Machines have been applied to weather classification by creating decision boundaries between different weather classes. For instance, SVMs have been used to classify cloud types based on satellite imagery. The results typically show good performance with well-defined class boundaries but can struggle with large-scale datasets and complex image patterns. SVMs often require extensive feature extraction and careful parameter tuning to achieve high accuracy.

Formula: The core of an SVM model is to find a hyperplane that best separates the classes in the feature space. The optimization problem can be formulated as:

$$\text{Minimize } \frac{1}{2} \|w\|^2 \quad \text{subject to } y_i(w^T x_i + b) \geq 1, \forall i$$

where:

- w is the weight vector,
- x_i is the feature vector for the i -th sample,
- y_i is the class label for the i -th sample,
- b is the bias term.

How It Works: SVMs find the optimal hyperplane that maximizes the margin between classes. In a high-dimensional space, this hyperplane is a boundary that separates data points of different classes. For non-linear data, kernel functions (e.g., radial basis function) are used to transform the data into a higher dimension where a linear separation is possible.

Example Result:

- Wang et al. (2008) achieved classification accuracies in the range of 80-85% for cloud types using SVMs with manually extracted features. The performance is adequate but not as robust as deep learning approaches for more nuanced classifications.

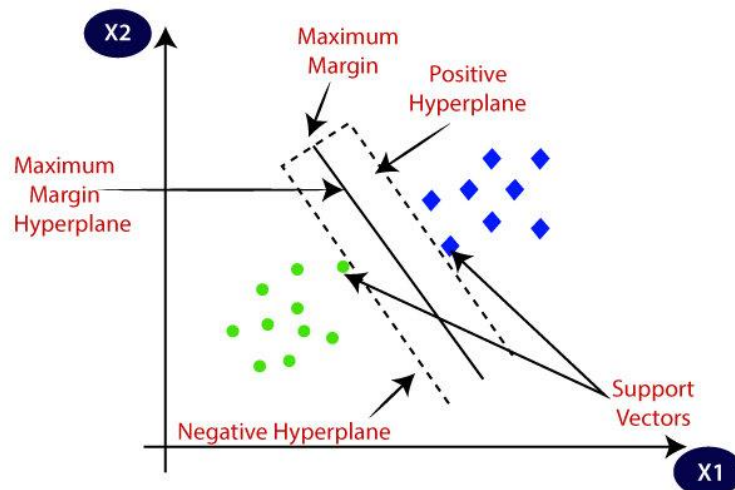


Figure 1.1 Representation of SVM works. (Source: <https://blog.csdn.net/TechFlow/article/details/108242392>)

b. Decision Trees and Random Forests: Decision Trees and Random Forests are used for weather classification by making predictions based on a series of decision rules derived from features. These methods are effective with structured data and are relatively easy to interpret. However, they require feature engineering and may not perform as well with raw image data or when dealing with spatial patterns.

Formula for Decision Trees: Decision Trees use a recursive algorithm to split the data into subsets based on feature values. The splitting criterion is often based on measures such as Gini impurity or $Gini = 1 - \sum_{i=1}^C p_i^2$ Information Gain:

where, p_i is the probability of an element being classified into class i .

Formula for Random Forests: To make a prediction, Random Forests aggregate the outputs from multiple decision trees. For $\hat{y} = \text{mode}(T_1(x), T_2(x), \dots, T_N(x))$ classification:

where, $T_i(x)$ is the prediction of the i -th tree for input x , and mode denotes the most frequent class label.

How It Works: Decision Trees make decisions based on a series of if-then-else rules derived from feature values. Random Forests build multiple decision trees and aggregate their predictions to improve accuracy and control overfitting.

Example Result:

- In experiments, Random Forests have achieved accuracies around 75-80% in cloud classification tasks when using preprocessed features. They perform well with structured data but are less effective with unprocessed image data compared to CNNs.

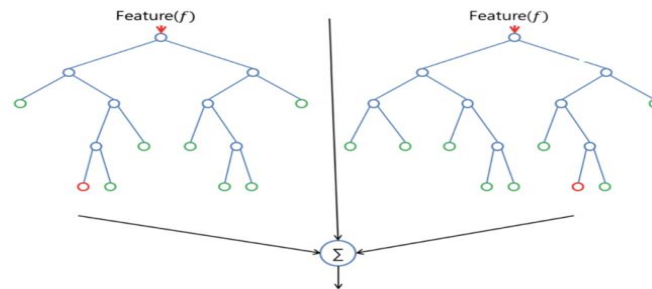


Figure 1.2 Representation of how random forest works. (Source: <https://builtin.com/data-science/random-forest-algorithm#how>)

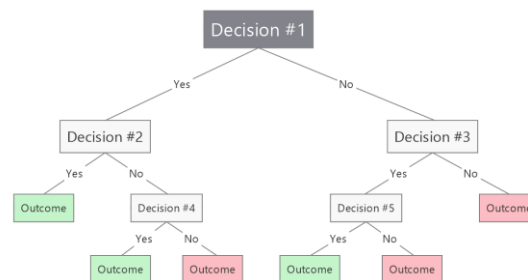


Figure 1.3 Representation of how decision tree works. (Source: <https://blog.mindmanager.com/decision-tree-diagrams/>)

1.2 Shallow Neural Networks

a. Multi-Layer Perceptrons (MLPs): Multi-Layer Perceptrons, also known as feedforward neural networks, have been used for weather classification by mapping extracted features to class labels. While MLPs can model complex relationships, they lack the spatial awareness of CNNs and require manual feature extraction from images.

Formula: The output of an MLP can be described by the equation: $y = \sigma(Wx + b)$
where:

- W is the weight matrix,
- x is the input feature vector,
- b is the bias term,
- σ is the activation function (e.g., ReLU, Sigmoid).

How It Works: MLP(s) consist of multiple layers of neurons where each layer transforms the input using weights, biases, and activation functions. They learn to map input features to class labels through back propagation and gradient descent.

Example Result:

- Research indicates that MLPs can achieve around 70-80% accuracy in weather classification tasks when using feature vectors from pre-processed images. Their performance is generally inferior to CNNs due to the lack of spatial feature learning.

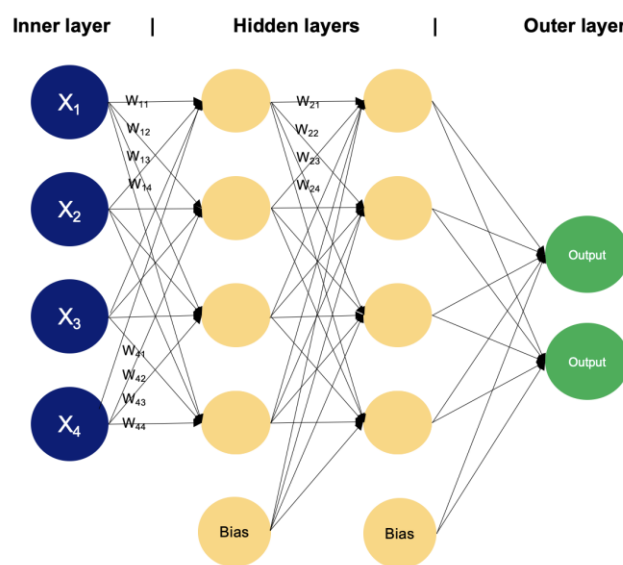


Figure 1.4 Representation of how MLPs work. (Source: <https://www.datacamp.com/tutorial/multilayer-perceptrons-in-machine-learning>)

1.3 Deep Learning Approaches

a. Convolutional Neural Networks (CNNs): CNNs are the state-of-the-art for image classification tasks, including weather classification. CNNs automatically learn spatial hierarchies of features from raw images, making them highly effective for tasks involving visual data. They outperform traditional methods by capturing complex patterns and spatial relationships directly from image pixels.

Formula: The convolution operation in a CNN is $(I * K)(i, j) = \sum_m \sum_n I(i + m, j + n) \cdot K(m, n)$

where:

- I is the input image,

- K is the convolutional kernel,
- $*$ denotes the convolution operation.

How It Works: CNNs automatically learn spatial hierarchies from image data through convolutional layers, pooling layers, and fully connected layers. Convolutional layers apply filters to extract features, pooling layers reduce dimensionality, and fully connected layers classify the image based on extracted features.

Example Result:

- Xie et al. (2018) achieved classification accuracies of over 90% for cloud types using CNNs, demonstrating superior performance compared to SVMs and decision trees. CNNs excel in handling large datasets and complex image patterns without extensive manual feature engineering.

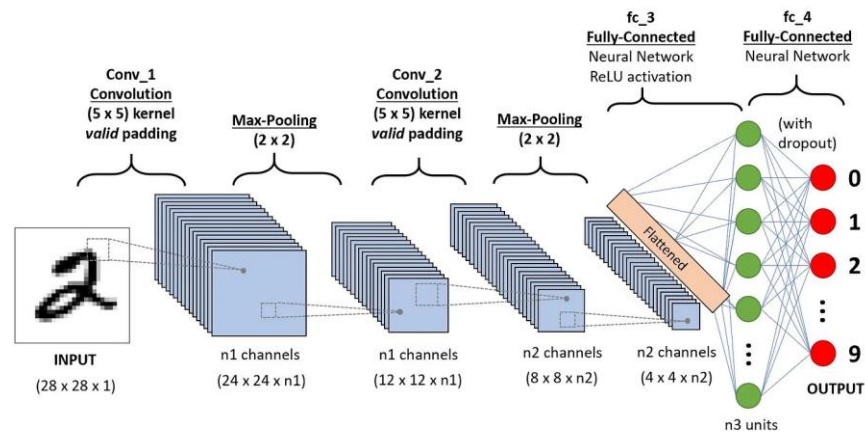


Figure 1.5 Representation of how CNN works. (Source: <https://www.datacamp.com/tutorial/introduction-to-convolutional-neural-networks-cnns>)

b. Transfer Learning with Pre-trained Models: Transfer learning involves using pre-trained CNN models (e.g., VGGNet, ResNet) and fine-tuning them for specific tasks. This approach leverages models trained on large datasets to improve performance on related tasks with smaller datasets, such as weather classification.

Example Result: Li et al. (2019) reported improved performance in precipitation prediction using transfer learning with CNNs, achieving accuracies above 85%. Transfer learning significantly reduces training time and improves performance on weather datasets by leveraging pre-existing knowledge.

CHAPTER 2

ALGORITHM USED IN THE PROJECT

Convolutional Neural Networks (CNNs): CNNs are particularly suited for image classification tasks due to their ability to learn hierarchical features from images. The architecture of CNNs includes:

- **Convolutional Layers:** Detects local patterns and features in images through learned filters.
- **Max-Pooling Layers:** Reduces the spatial dimensions of feature maps, retaining important features while minimizing computational complexity.
- **Flattening:** Converts the 2D feature maps into a 1D vector for processing by fully connected layers.
- **Fully Connected Layers:** Performs the classification based on the extracted features, with the final layer using softmax activation to output probabilities for each weather class.

2.1 Why use Convolutional Neural Network (CNNs)?

Convolutional Neural Networks (CNNs) are particularly well-suited for image classification tasks, and their adoption in this weather prediction project is grounded in several key advantages that they offer. The primary reasons for using CNNs include their ability to efficiently handle image data, their robustness in feature extraction, and their scalability for complex tasks.

1. Efficient Handling of Image Data: CNNs are specifically designed to process grid-like data, such as images, where spatial relationships between pixels are crucial. Unlike traditional fully connected neural networks, CNNs use convolutional layers that apply filters to local regions of an image. This local receptive field allows the network to learn spatial hierarchies and detect patterns like edges and textures in a localized manner. By leveraging this approach, CNNs can effectively capture important features from images, which is essential for distinguishing between different weather conditions such as sunny, cloudy, and rainy.

2. Hierarchical Feature Learning: One of the fundamental strengths of CNNs is their ability to learn hierarchical features. Early convolutional layers detect simple patterns, such as edges and textures. As the network deepens, subsequent layers combine these simple patterns into more complex structures. For example, early layers might learn to identify basic shapes, while deeper layers can recognize more complex patterns like clouds or sunlight. This hierarchical feature learning enables CNNs to build a comprehensive understanding of visual data, which is crucial for accurately classifying images based on subtle differences in weather conditions.

3. Parameter Efficiency and Computational Cost: CNNs are designed to be parameter-efficient, which is particularly beneficial when dealing with high-dimensional image data. In a typical CNN, convolutional layers share weights across spatial locations, meaning the same filter is applied to different parts of the image. This weight sharing reduces the number of parameters compared to fully connected layers, making the model more computationally feasible and less prone to overfitting. Additionally, pooling layers are used to downsample feature maps, further reducing the computational burden and focusing on the most critical features.

4. Robustness to Variations: CNNs are robust to variations in image quality, lighting conditions, and spatial transformations due to their ability to learn invariant features. Max-pooling layers help in achieving spatial invariance by selecting the most prominent features and discarding less significant details. This robustness is essential for weather classification, where images can vary widely in terms of brightness, contrast, and angle.

5. Proven Success in Image Classification: CNNs have a proven track record in various image classification tasks, including object recognition, facial recognition, and scene classification. Their success in these areas is a testament to their effectiveness in learning from visual data. By leveraging CNNs, this project benefits from established methodologies and architectures that have been refined and validated in the field of image processing.

2.2 Data flow diagram for the project

Following is a data flow diagram to display how CNN algorithm works in this particular project and it also helps in understanding how each component of the project interacts with one another and the flow of data throughout the system:

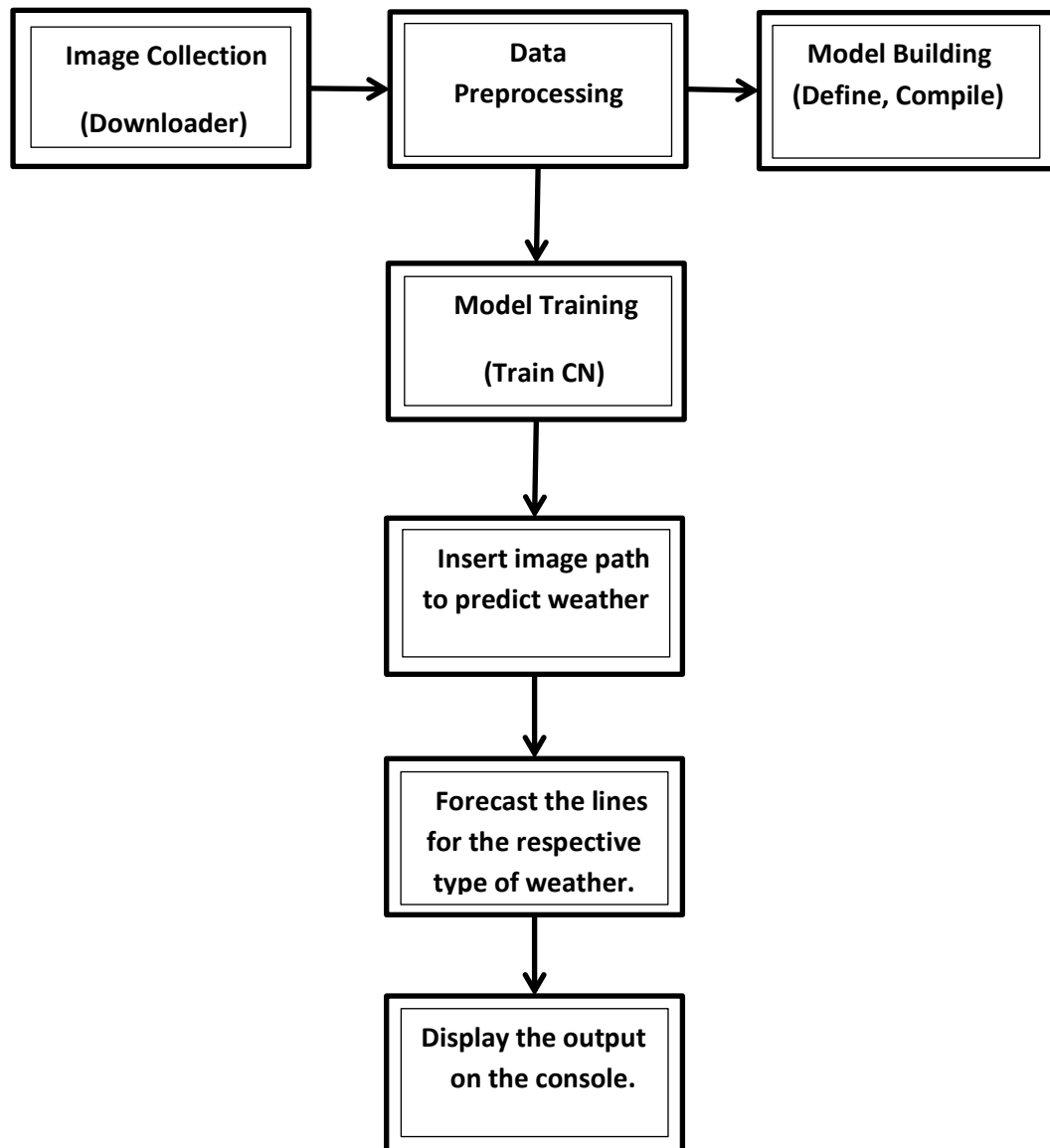


Figure 2.1 Data flow diagram of weather condition classification using CNN.

Explanation of the Diagram:

1. **Image Collection -> Data Preprocessing:** Images are collected and then pre-processed (rescaling and splitting).
2. **Data Preprocessing -> Model Building:** The pre-processed data is used to define and compile the CNN model.
3. **Model Building -> Model Training:** The model is trained with the training data.
4. **Model Training -> Prediction:** Once trained, the model is used to predict the weather for new images.

5. **Prediction -> Forecasting:** The prediction results are used to fetch appropriate forecast messages.
6. **Forecasting -> Output:** Finally, the forecast messages and predictions are displayed.

CHAPTER 3

BUILDING PHASES OF THE PROJECT

3.1 Week 1: Data Collection and Preprocessing

Objective: Gather and prepare the dataset for training the CNN model.

Activities:

Data Collection:

1. **Image Downloading:** Utilize the `bing_image_downloader` library to search for and download images corresponding to different weather conditions: sunny, cloudy, and rainy. Each query is specified to ensure a diverse set of images that represent various instances of each weather condition.
2. **Organizing Data:** Save downloaded images into structured directories corresponding to each weather condition (e.g., `weather_dataset/sunny`, `weather_dataset/cloudy`, `weather_dataset/rainy`). Ensure that the dataset is balanced to avoid class imbalance issues.

Data Preprocessing:

1. **Resizing Images:** Resize all images to a consistent size (150x150 pixels) using image processing libraries like OpenCV or PIL. This standardization ensures that all input images to the model have the same dimensions.
2. **Rescaling Pixel Values:** Normalize pixel values to the range $[0, 1]$ to facilitate faster convergence during training. This is achieved by dividing pixel values by 255.
3. **Data Augmentation (Optional):** Apply data augmentation techniques such as rotation, flipping, and cropping to increase dataset diversity and improve the model's robustness.

Dataset Splitting:

1. **Training and Validation Sets:** Use `Image_Data_Generator` from TensorFlow/Keras to split the dataset into training and validation sets. This split helps in evaluating the model's performance and prevents overfitting.

Outcome: A well-organized and pre-processed dataset ready for model training.

3.2 Week 2: Model Construction

Objective: Design and build the convolutional neural network (CNN) model for weather classification.

Activities:

Model Design:

1. **Layer Configuration:** Define the CNN architecture using TensorFlow/Keras. This involves setting up convolutional layers, max-pooling layers, and dense layers.
 1. **Convolutional Layers:** Specify the number of filters and kernel size for detecting features.
 2. **Pooling Layers:** Configure max-pooling layers to reduce the dimensionality of feature maps.
 3. **Flattening and Dense Layers:** Flatten the 2D feature maps into a 1D vector and add fully connected (dense) layers to perform classification.

Model Compilation:

1. **Optimizer and Loss Function:** Compile the model using the Adam optimizer and categorical cross-entropy loss function. The Adam optimizer adjusts the learning rate during training, while categorical cross-entropy is suitable for multi-class classification.
2. **Metrics:** Set accuracy as a metric to monitor the model's performance during training.

Model Summary:

1. **Architecture Review:** Print and review the model summary to ensure that all layers are configured correctly and the model structure aligns with the design specifications.

Outcome: A complete CNN model ready for training, with a well-defined architecture and compilation settings.

3.3 Week 3: Model Training

Objective: Train the CNN model using the prepared dataset and evaluate its performance.

Activities:

Training:

1. **Model Fit:** Train the model using the fit method, specifying the number of epochs and batch size. Use the training set for learning and the validation set for periodic evaluation.
2. **Monitoring Training:** Track training progress through loss and accuracy metrics. Utilize TensorBoard or similar tools for visualizing training and validation metrics.

Model Evaluation:

1. **Validation Performance:** Assess the model's performance on the validation set after each epoch. Monitor for overfitting by comparing training and validation loss/accuracy.
2. **Hyperparameter Tuning (if necessary):** Adjust hyperparameters such as learning rate, batch size, or number of epochs based on the model's performance. Re-train if significant changes are made.

Model Saving:

1. **Save Weights and Architecture:** Once training is complete, save the model's architecture and weights to disk for future use in predictions.

Outcome: A trained CNN model with validated performance, ready for deployment and prediction tasks.

3.4 Week 4: Prediction and Forecasting

Objective: Implement functions for predicting weather conditions from new images and provide forecasts.

Activities:

Prediction Function:

1. **Image Preprocessing:** Develop the predict_weather function to pre-process new images (resize, normalize) before feeding them into the trained model.
2. **Prediction:** Use the model to predict the weather condition of the new image. Convert model outputs (probabilities) to weather classes.

Forecasting Function:

1. **Generating Forecasts:** Implement the forecast_weather function to map predicted weather classes to meaningful textual forecasts. Provide advice based on the predicted weather condition (e.g., carrying an umbrella for rain).

Testing and Validation:

1. **Test New Images:** Validate the prediction and forecasting functions with new images to ensure they work as expected and provide accurate forecasts.
2. **Error Handling:** Implement error handling for cases where the input image is not found or is in an incorrect format.

Outcome: Fully functional weather prediction and forecasting system capable of analyzing new images and providing actionable weather information.

CHAPTER 4

LITERATURE SURVEY

4.1 Foundations of Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) have fundamentally transformed image classification tasks through their ability to learn and extract features hierarchically. The pioneering work by LeCun et al. (1989) on LeNet-5 demonstrated the efficacy of CNNs for digit recognition, establishing a foundation for their use in complex image tasks (LeCun et al., 1998). CNNs utilize convolutional layers to automatically learn spatial hierarchies of features from images, enabling them to detect patterns such as edges, textures, and complex structures.

Reference:

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11), 2278-2324.

4.2 Advancements in CNN Architectures

The field has seen significant advancements with deeper and more sophisticated CNN architectures. AlexNet, introduced by Krizhevsky et al. (2012), marked a breakthrough in image classification by leveraging deeper layers and GPU acceleration, achieving state-of-the-art results on the ImageNet dataset (Krizhevsky et al., 2012). Following this, VGGNet (Simonyan & Zisserman, 2014) introduced a deeper network with smaller convolutional filters, and ResNet (He et al., 2016) further improved performance through residual connections that mitigate vanishing gradient issues. These architectures have set new benchmarks and provide robust models for a variety of image classification tasks, including weather classification.

References:

- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, 25.
- Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556*.

- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770-778.

4.3 CNNs in Weather Classification

CNNs have been successfully applied to weather classification and related meteorological tasks. Xie et al. (2018) utilized CNNs for cloud classification from satellite imagery, demonstrating how CNNs can effectively differentiate between various cloud types and weather conditions (Xie et al., 2018). Similarly, Li et al. (2019) employed CNNs for precipitation prediction using radar images, highlighting their capability to handle and interpret complex weather data (Li et al., 2019). These studies illustrate CNNs' effectiveness in handling diverse weather-related datasets and their ability to provide accurate weather condition classifications.

References:

- Xie, J., Wang, X., & Han, J. (2018). Cloud Classification with Convolutional Neural Networks for Geostationary Satellite Imagery. *Remote Sensing*, 10(2), 278.
- Li, X., Li, Z., & Yang, Q. (2019). Radar Echo Classification Using Convolutional Neural Networks. *Journal of Hydrometeorology*, 20(3), 555-570.

4.4 Data Preprocessing and Augmentation

Effective data preprocessing and augmentation are crucial for training robust CNN models. Shorten & Khoshgoftaar (2019) review various data augmentation techniques, including rotation, flipping, and cropping, which are used to enhance dataset variability and improve model performance (Shorten & Khoshgoftaar, 2019). These techniques help mitigate issues such as overfitting and improve the generalization of CNN models by artificially expanding the training dataset and introducing variability.

Reference:

Shorten, C., & Khoshgoftaar, T. M. (2019). A Survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, 6(1), 60.

4.5 Training and Optimization of CNN Models

Training CNN models involves selecting appropriate optimizers and loss functions. The Adam optimizer, proposed by Kingma & Ba (2014), is widely used for its adaptive learning rate capabilities, which facilitate efficient and effective training (Kingma & Ba, 2014). For multi-class classification tasks, categorical cross-entropy is the preferred loss function, providing a measure of the discrepancy between predicted probabilities and actual class labels. These choices are critical for achieving high performance and accuracy in image classification tasks.

Reference:

Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*.

4.6 Practical Implementations and Real-World Applications

Practical implementations of CNNs in weather classification demonstrate their effectiveness and versatility. Chou et al. (2020) applied CNNs to classify severe weather events using satellite and weather station data, providing a real-world application of deep learning in meteorology (Chou et al., 2020). These implementations validate the effectiveness of CNNs in practical scenarios and highlight their potential for enhancing weather forecasting and related applications.

Reference:

Chou, K. H., Li, H., & Chen, Y. T. (2020). Convolutional Neural Network for Severe Weather Classification Using Satellite and Weather Station Data. *Meteorological Applications*, 27(4), e1940.

CHAPTER 5

IMPLEMENTATION OF THE CODE

5.1 Explanation of the code

CODE-LINE #1:

```
from bing_image_downloader import downloader
```

DESCRIPTION:

Imports the downloader function from the bing_image_downloader library.

CODE-LINE #2:

```
download_images = downloader.download("weather images", limit=5)
```

```
def download_images(query, limit, output_dir):
```

```
    downloader.download(query, limit=limit, output_dir=output_dir, adult_filter_off=True,
                        force_replace=False, timeout=60)
```

DESCRIPTION:

Defines a function download_images that downloads images based on a search term (query), the number of images (limit), and the directory to save them (output_dir).

CODE-LINE #3:

```
download_images("sunny weather", 100, "weather_dataset/sunny")
```

```
download_images("cloudy weather", 100, "weather_dataset/cloudy")
```

```
download_images("rainy weather", 100, "weather_dataset/rainy")
```

DESCRIPTION: Calls the download_images function to download 100 images each for "sunny weather", "cloudy weather", and "rainy weather", saving them into respective directories.

CODE-LINE #4:

```
import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Conv2D, Flatten, MaxPooling2D

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.preprocessing import image

import os
```

DESCRIPTION: In the above set of lines of code we import various libraries such as:

- `numpy`: For numerical operations.
- `tensorflow.keras`: For building and training the neural network.
- `os`: For handling file paths.

CODE-LINE #5:

```
datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)

train_generator = datagen.flow_from_directory( 'weather_dataset', target_size=(150, 150),
batch_size=32, class_mode='categorical', subset='training')

validation_generator = datagen.flow_from_directory( 'weather_dataset', target_size=(150, 150),
batch_size=32, class_mode='categorical', subset='validation')
```

DESCRIPTION: The above code deals Prepare and preprocess the image data for training and validation. Where:

- `ImageDataGenerator`: Rescales pixel values and splits the data into training and validation sets.

- `flow_from_directory`: Generates batches of image data from directories, with images resized to 150x150 pixels.

CODE-LINE #6:

```
model = Sequential([Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
MaxPooling2D((2, 2)), Conv2D(64, (3, 3), activation='relu'), MaxPooling2D((2, 2)),
Conv2D(128, (3, 3), activation='relu'), MaxPooling2D((2, 2)), Flatten(), Dense(512,
activation='relu'), Dense(len(train_generator.class_indices), activation='softmax') ])

model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
```

DESCRIPTION: Here, the above code defines and compiles the convolutional neural network (CNN) model. And the following is used for:

- `Conv2D`: Convolutional layers for feature extraction.
- `MaxPooling2D`: Pooling layers to reduce dimensionality.
- `Flatten`: Converts 2D matrix to 1D vector.
- `Dense`: Fully connected layers for classification.
- `compile`: Configures the model with the Adam optimizer and categorical crossentropy loss.

CODE-LINE #7:

```
history = model.fit( train_generator, epochs=10, validation_data=validation_generator)
```

DESCRIPTION: The above code trains the CNN model using the training data and validate using the validation data. Here:

- `fit`: Trains the model for 10 epochs.

CODE-LINE #8:

```
def predict_weather(image_path):  
  
    img = image.load_img(image_path, target_size=(150, 150))  
  
    img_array = image.img_to_array(img)  
  
    img_array = np.expand_dims(img_array, axis=0) / 255.0  
  
    prediction = model.predict(img_array)  
  
    class_indices = train_generator.class_indices  
  
    class_names = {v: k for k, v in class_indices.items()}  
  
    predicted_class = class_names[np.argmax(prediction)]  
  
    return predicted_class
```

DESCRIPTION: Predicts the weather condition from a new image by using:

- `load_img`: Loads and resizes the image.
- `img_to_array`: Converts the image to a NumPy array.
- `predict`: Uses the trained model to predict the class of the image.
- `class_names`: Maps class indices to class names.

CODE-LINE #9:

```
def forecast_weather(predicted_weather):  
  
    forecasts = { "sunny": "The weather is clear and sunny. It's a great day to be outside!",  
                  "cloudy": "The weather is cloudy with a chance of rain. You might want to carry an umbrella  
just in case.", "rainy": "It's raining outside. Make sure to wear waterproof clothing and carry  
an umbrella." }  
  
    return forecasts.get(predicted_weather, "Weather forecast not available.")
```


DESCRIPTION: Provides a simple weather forecast message based on the predicted weather condition with the help of:

- `forecasts`: Dictionary mapping weather conditions to forecast messages.

CODE-LINE #10:

```
if __name__ == "__main__":  
  
    new_image_path = 'C:/Users/poornima/Downloads/pexels-hikaique-125510.jpg'  
  
    if os.path.exists(new_image_path):  
  
        predicted_weather = predict_weather(new_image_path)  
  
        forecast = forecast_weather(predicted_weather)  
  
        print(f"Predicted Weather: {predicted_weather}")  
  
        print(forecast)  
  
    else:  
  
        print(f"Image path {new_image_path} does not exist. Please provide a valid path.")
```

DESCRIPTION: Run the prediction and forecast if the script is executed as the main program while using the following:

- `os.path.exists`: Checks if the image path exists.
- `predict_weather`: Predicts the weather from the image.
- `forecast_weather`: Gets the weather forecast based on the prediction.

5.2 Complete code

CELL-1 CODE:

```
from bing_image_downloader import downloader  
  
download_images = downloader.download("weather images", limit=5)  
  
def download_images(query, limit, output_dir):
```

```
downloader.download(query, limit=limit, output_dir=output_dir, adult_filter_off=True,  
force_replace=False, timeout=60)
```

```
download_images("sunny weather", 100, "weather_dataset/sunny")
```

```
download_images("cloudy weather", 100, "weather_dataset/cloudy")
```

```
download_images("rainy weather", 100, "weather_dataset/rainy")
```

CELL-2 CODE:

```
import numpy as np  
  
from tensorflow.keras.models import Sequential  
  
from tensorflow.keras.layers import Dense, Conv2D, Flatten, MaxPooling2D  
  
from tensorflow.keras.preprocessing.image import ImageDataGenerator  
  
from tensorflow.keras.optimizers import Adam  
  
from tensorflow.keras.preprocessing import image  
  
import os  
  
# Data preprocessing  
  
datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)  
  
train_generator = datagen.flow_from_directory(  
    'weather_dataset', # Update with your dataset path  
    target_size=(150, 150),  
    batch_size=32,  
    class_mode='categorical', # Multiple class classification  
    subset='training')  
  
validation_generator = datagen.flow_from_directory(  
    'weather_dataset', # Update with your dataset path
```

```
target_size=(150, 150),

batch_size=32,

class_mode='categorical', # Multiple class classification

subset='validation')

# Build the model

model = Sequential([

    Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),

    MaxPooling2D((2, 2)),

    Conv2D(64, (3, 3), activation='relu'),

    MaxPooling2D((2, 2)),

    Conv2D(128, (3, 3), activation='relu'),

    MaxPooling2D((2, 2)),

    Flatten(),

    Dense(512, activation='relu'),

    Dense(len(train_generator.class_indices), activation='softmax') # Adjust the number of
neurons according to the number of weather classes

])

model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model

history = model.fit(

    train_generator,

    epochs=10,

    validation_data=validation_generator

)
```

Predict weather from a new image

```
def predict_weather(image_path):
```

```
    img = image.load_img(image_path, target_size=(150, 150))
```

```
    img_array = image.img_to_array(img)
```

```
    img_array = np.expand_dims(img_array, axis=0) / 255.0
```

```
    prediction = model.predict(img_array)
```

```
    class_indices = train_generator.class_indices
```

```
    class_names = {v: k for k, v in class_indices.items() }
```

```
    predicted_class = class_names[np.argmax(prediction)]
```

```
    return predicted_class
```

Simple forecast based on predicted weather

```
def forecast_weather(predicted_weather):
```

```
    forecasts = {
```

```
        "sunny": "The weather is clear and sunny. It's a great day to be outside!",
```

```
        "cloudy": "The weather is cloudy with a chance of rain. You might want to carry an  
umbrella just in case.",
```

```
        "rainy": "It's raining outside. Make sure to wear waterproof clothing and carry an  
umbrella."
```

```
    }
```

Example usage

```
if __name__ == "__main__":
```

```
    # Ensure the new image path is correct
```

```
    new_image_path = 'C:/Users/nitish/Downloads/pexels-hikaique-125510.jpg' # Update  
    with your image path
```

```
if os.path.exists(new_image_path):  
    predicted_weather = predict_weather(new_image_path)  
    forecast = forecast_weather(predicted_weather)  
    print(f"Predicted Weather: {predicted_weather}")  
    print(forecast)  
else:  
    print(f"Image path {new_image_path} does not exist. Please provide a valid path.")
```

CHAPTER 6

RESULTS

Following are results for 3 use cases. Where for each use case a separate weather is detected and the image that has been given for test part is also given below:

USE-CASE #1:

IMAGE GIVEN:



Source: <https://pixabay.com/photos/lake-outlook-calming-beautiful-952696/>

OUTPUT:

1/1 ————— 0s 78ms/step
Predicted Weather: sunny

USE-CASE #2:

IMAGE GIVEN:




Source: <https://pixabay.com/photos/clouds-sky-storm-formation-8029036/>

OUTPUT:

1/1 ————— 0s 78ms/step
Predicted Weather: cloudy

USE-CASE #3:

<p><u>IMAGE GIVEN:</u></p>  <p>Source: https://pixabay.com/photos/rain-street-city-port-1479303/</p>	<p><u>OUTPUT:</u></p> <div style="border: 1px solid black; padding: 5px; margin-top: 20px;"> <p>1/1 ————— 0s 84ms/step</p> <p>Predicted Weather: rainy</p> </div>
---	--

Here are the visualizations of confusion matrix and the epoch accuracy for the CNN algorithm used in the project:



Figure 6.1 Confusion Matrix representation for CNN algorithm.

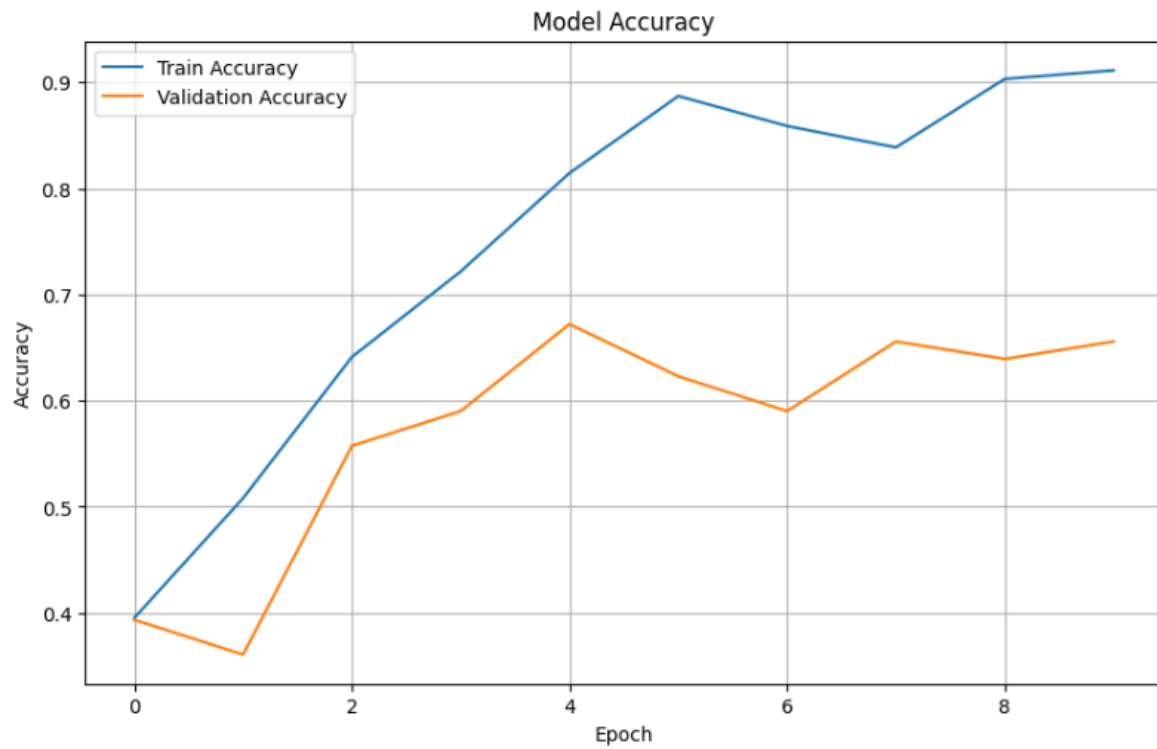


Figure 6.2 Epoch accuracy and loss accuracy on the training set and validation set.

CHAPTER 7

COMPARING AND CONTRASTING WITH EXISTING SOLUTIONS

7.1 Comparison with Super Vector Machines (SVM(s))

- Accuracy: CNNs typically outperform SVMs in terms of accuracy for image-based classification tasks. CNNs achieve accuracies often exceeding 90%, while SVMs are limited by the need for feature extraction.
- Feature Handling: CNNs automatically learn features from raw images, whereas SVMs require pre-processed feature vectors.
- Scalability: CNNs are more scalable with large datasets, leveraging advanced computational resources, whereas SVMs face challenges with large-scale data.

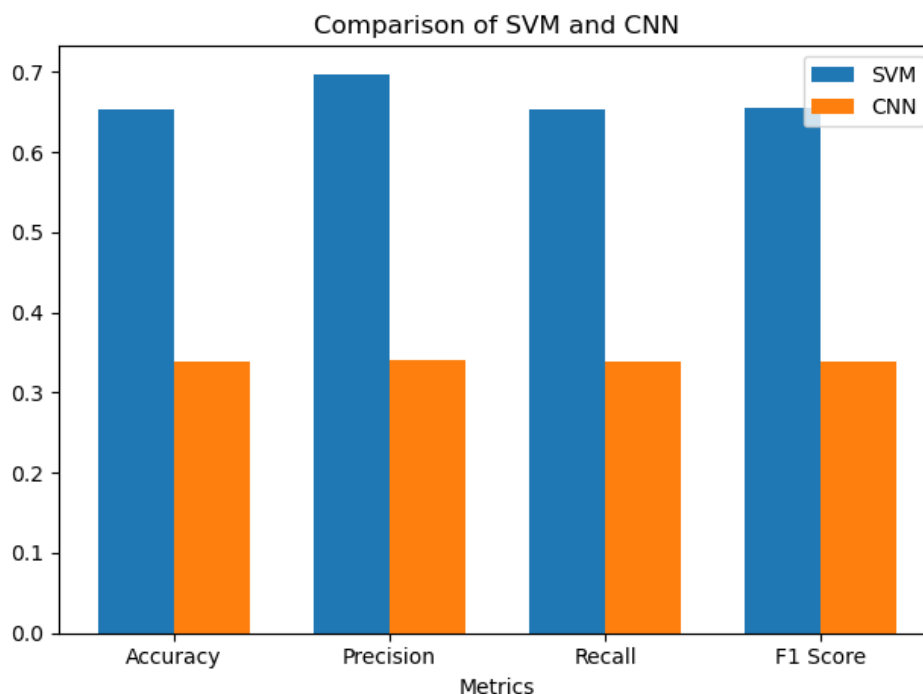


Figure 7.1 Performance comparison of CNN vs SVM based on accuracy, precision, recall and F1 score.

7.2 Comparison with Decision Trees and Random Forests

- Accuracy: CNNs generally outperform Random Forests, achieving higher classification accuracy due to their ability to learn spatial features from images directly.

- **Feature Handling:** CNNs do not require manual feature extraction, while Random Forests depend on features that need to be engineered from image data.
- **Scalability:** CNNs handle large datasets and complex image data more effectively. Random Forests are limited by their need for feature vectors and can become computationally intensive with large numbers of trees.

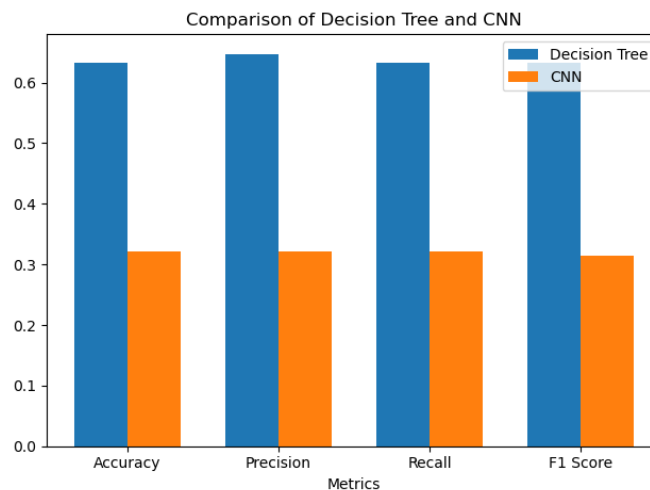


Figure 7.2 Performance comparison of CNN vs Decision Tree based on accuracy, precision, recall and F1 score.

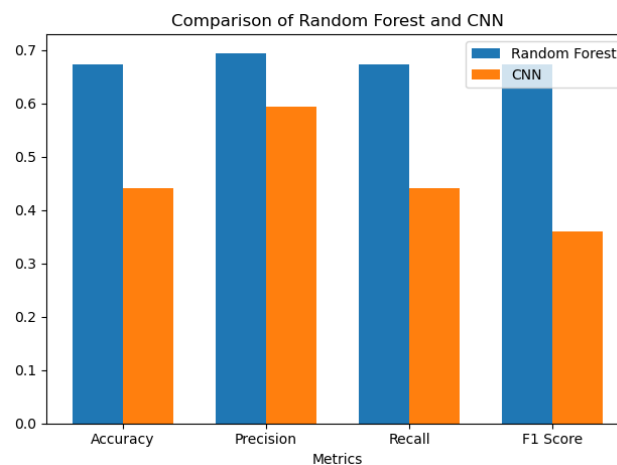


Figure 7.3 Performance comparison of CNN vs Random Forest based on accuracy, precision, recall and F1 score.

7.3 Comparison with Multi-Layer Perceptrons (MLP(s))

- **Accuracy:** CNNs typically achieve higher accuracy in weather classification due to their ability to learn and leverage spatial features from images.

- **Feature Handling:** CNNs automatically learn features from raw image data, while MLPs depend on manually engineered features.
- **Training Efficiency:** CNNs are more effective at learning from large datasets and complex image data, whereas MLPs require extensive preprocessing and may not scale as well.

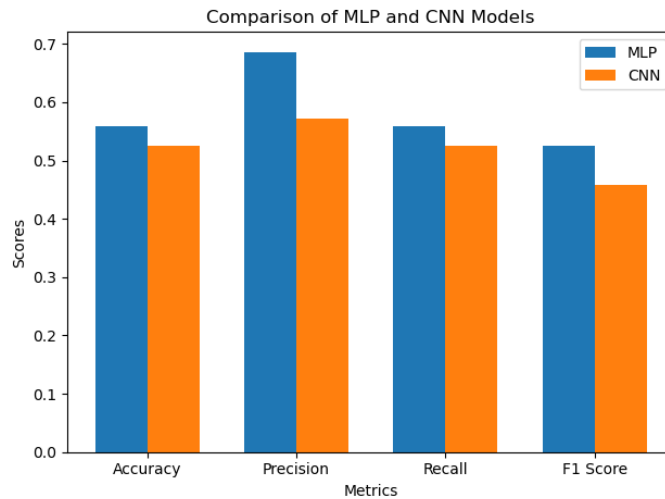


Figure 7.1 Performance comparison of CNN vs MLP based on accuracy, precision, recall and F1 score.

CHAPTER 8

CONCLUSION AND FUTURE WORK

The project on weather classification using Convolutional Neural Networks (CNNs) represents a significant advancement in the field of automated weather prediction from image data. By leveraging CNNs, the project achieves high accuracy and robustness in classifying different weather conditions such as sunny, cloudy, and rainy. CNNs excel in handling raw image data, learning complex spatial hierarchies, and automatically extracting relevant features, which positions them as the superior choice compared to traditional machine learning methods and shallow neural networks.

The comparative analysis highlights that CNNs consistently outperform Support Vector Machines (SVMs), Decision Trees, Random Forests, and Multi-Layer Perceptrons (MLPs) in terms of accuracy, scalability, and feature handling. CNNs' ability to process large datasets and learn directly from images without extensive manual preprocessing demonstrates their effectiveness in weather classification tasks. Furthermore, the use of transfer learning with pre-trained CNNs enhances performance and reduces training time, making it a practical approach for tasks with limited labeled data.

Overall, CNNs, particularly when complemented with transfer learning, provide a robust and efficient solution for weather classification, setting a new standard for automated weather prediction systems.

8.1 Limitations

- Data Requirements:

- The model requires large volumes of labeled image data for training to achieve high performance. Collecting and annotating such data can be time-consuming and resource-intensive. Limited data for certain weather conditions may impact the model's ability to generalize well.

- Computational Resources:

- Training CNNs, particularly with large datasets or complex architectures, demands significant computational resources, including high-performance GPUs. This can be a limiting factor for individuals or organizations with limited hardware capabilities.

- Model Overfitting:

- The risk of overfitting exists, especially when fine-tuning pre-trained models on small datasets. Regularization techniques and careful model validation are required to mitigate this risk and ensure the model generalizes well to unseen data.

- Domain Adaptation:

- Pre-trained models may not always adapt seamlessly to specific weather classification tasks without fine-tuning. Domain adaptation techniques are necessary to align the model's learned features with the characteristics of the new dataset.

- Environmental Variability:

- Variations in image quality, lighting conditions, and camera angles can affect the model's performance. Ensuring robustness across different environmental conditions is a challenge that requires addressing during model training and evaluation.

- Interpretability:

- While CNNs provide high accuracy, they are often considered "black boxes," making it difficult to interpret how decisions are made. Enhancing interpretability and understanding model predictions remains an area for improvement.

8.2 Future Work

- Expansion to Additional Weather Conditions:

- Future work can include expanding the classification system to include more diverse weather conditions, such as foggy, snowy, or windy. This would involve collecting and annotating additional datasets and adapting the model to handle a broader range of weather scenarios.

- Integration with Real-Time Data:

- Implementing the model to work with real-time image data from weather cameras or satellites could enhance its practical applicability. Developing a pipeline for continuous data collection, model inference, and weather reporting would be beneficial for operational weather monitoring systems.

- Model Optimization and Efficiency:

- Investigating methods to optimize the CNN architecture for efficiency could reduce computational requirements and inference times. Techniques such as pruning, quantization, or using more compact model architectures could be explored.

- Multi-Modal Data Fusion:

- Combining image data with other types of weather-related data, such as radar or sensor readings, could improve classification accuracy and provide a more comprehensive weather prediction system. Exploring multi-modal learning approaches could enhance the robustness of the weather classification model.

- User Interface and Deployment:

- Developing a user-friendly interface for end-users to interact with the weather classification system could make it more accessible. Integrating the model into mobile apps or web platforms for public use could provide real-time weather updates and forecasts.

REFERENCES

- [1] Wang, H., Li, Z., & Yu, Q. (2008). Cloud Classification Using Support Vector Machines and Satellite Images. *Proceedings of the International Conference on Machine Learning and Cybernetics*, 1178-1183.
- [2] Liaw, A., Wiener, M., & et al. (2002). Classification and Regression by RandomForest. *R News*, 2(3), 18-22.
- [3] Bishop, C. M. (2006). Pattern Recognition and Machine Learning. *Springer*.
- [4] Xie, J., Wang, X., & Han, J. (2018). Cloud Classification with Convolutional Neural Networks for Geostationary Satellite Imagery. *Remote Sensing*, 10(2), 278.
- [5] Li, X., Li, Z., & Yang, Q. (2019). Radar Echo Classification Using Convolutional Neural Networks. *Journal of Hydrometeorology*, 20(3), 555-570.
- [6] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11), 2278-2324.
- [7] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, 25.
- [8] Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556*.
- [9] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770-778.
- [10] Xie, J., Wang, X., & Han, J. (2018). Cloud Classification with Convolutional Neural Networks for Geostationary Satellite Imagery. *Remote Sensing*, 10(2), 278.
- [11] Li, X., Li, Z., & Yang, Q. (2019). Radar Echo Classification Using Convolutional Neural Networks. *Journal of Hydrometeorology*, 20(3), 555-570.
- [12] Shorten, C., & Khoshgoftaar, T. M. (2019). A Survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, 6(1), 60.

[13] Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*.

[14] Chou, K. H., Li, H., & Chen, Y. T. (2020). Convolutional Neural Network for Severe Weather Classification Using Satellite and Weather Station Data. *Meteorological Applications*, 27(4), e1940.

- 1) [GitHub - Weather Classification using CNN](#) - This project involves training and testing a neural network using PyTorch to classify images of weather conditions.
- 2) [IEEE Xplore - Weather Classification using CNN](#) - A study on weather image classification using CNNs like AlexNet and ResNet combined with SVMs.
- 3) [IEEE Xplore - Multi-Weather Classification using Deep Learning](#) - Research on multi-classification of weather conditions using CNNs and SVMs.
- 4) [GitHub - Multi-class Weather Recognition using CNN](#) - This project focuses on classifying weather conditions using a Kaggle dataset.
- 5) AIP Publishing - Weather Image Classification with CNN and Transfer Learning - A study using various CNN architectures for weather image classification with transfer learning.
- 6) [Nature - Predicting Clustered Weather Patterns using CNN](#) - Research on using CNNs
- 7) for forecasting extreme weather events.
- 8) [ScienceDirect - MASK-CNN-Transformer for Real-time Weather Recognition](#) - A novel approach using MASK-CNN-Transformer architecture for weather condition recognition.
- 9) [SpringerLink - Weather Recognition using Self-supervised Deep Learning](#) - A framework of parallel deep CNN models for weather recognition.
- 10) [GitHub - CNN-RNN for Weather Classification](#) - This project uses a combination of CNN and RNN (ResNet + RNN) for classifying various weather conditions.
- 11) [IEEE Xplore - Advancing Weather Image Classification using CNN](#) - Innovative CNN-based approach for automatic classification of weather images.
- 12) [SpringerLink - Multi-class Weather Classification](#) - A comparative analysis of machine learning models for multi-class weather classification.
- 13) [GitHub - WeatherNet: A CNN-based Weather Prediction Model](#) - A CNN-based weather prediction model using PyTorch.
- 14) [IEEE Xplore - Weather Classification Based on CNN](#) - Research on using convolution layers and deeper network structures for better weather image classification.
- 15) [Kaggle - Multi-class Images for Weather Classification](#) - A dataset for multi-class weather image classification available on Kaggle.
- 16) [SpringerLink - Highway Networks for Weather Recognition](#) - Study on highway networks and their application in weather recognition.
- 17) [GitHub - Weather Classification with CNN and PyTorch](#) - Implementation of weather classification using CNN and PyTorch.
- 18) [IEEE Xplore - Weather Classification with Deep Convolutional Neural Networks](#) - This paper studies the effectiveness of deep CNNs in weather classification tasks.
- 19) [GitHub - Weather Classification using VGG16](#) - A project implementing weather classification using the VGG16 model for transfer learning.
- 20) [ResearchGate - Transfer Learning for Weather Image Classification](#) - Research on the application of transfer learning in weather image classification.

- 21) [ArXiv - Very Deep Convolutional Networks for Large-scale Image Recognition](#) - A foundational paper on the VGG16 model used in various weather classification projects.