UNIVERSITÀ DEGLI STUDI DI PADOVA

Computer Vision 2023/2024
FINAL PROJECT: HOW'S THE WEATHER TODAY

AYSIMA MERVE AKKURT

2071495

TABLE OF CONTENTS

1. DATASETS

In this project we have 4 different datasets.These are :



Class	Cloudy	Sunshine	Rainy	Sunrise
Samples	300	235	215	357

1)THE DATASETS: MULTI-CLASS WEATHER DATASET (MWD)

- You can get the data from https://drive.google.com/openid=1zJtLsAtdDXKBgBpupqw4YhUH0TyB-GBP&usp=drive_fs
- Provides 1125 images divided into 4 classes as in the table
- The data has been divided into 75% for training and 25% for testing

2)THE DATASETS: ACDC

- You can get from




Class	Clear	Fog	Night	Rain	Snow
Samples	300	300	300	300	300

https://drive.google.com/file/d/1zLieQfvP_6C3pkt0FKC1kmMKtXv2Nimt/view?usp=sharing

- Provides 1500 images divided into 5 classes as in the table
- The data has been divided into 2/3 for training and 1/3 for testing

3)THE DATASETS: UAVID

- You can get from




Class	Clear	Fog	Night	Rain
Samples	30	30	30	30

https://drive.google.com/file/d/1zKCUXF7wltISTpxd9pW3MLHG6egICvYN/view?usp=sharing

- Provides 120 images divided into 4 classes as in the table
- The data has been divided into 2/3 for training and 1/3 for testing

4)THE DATASETS: SYNDRONE

- You can get from

Class	Clear	Fog	Night	Rain
Samples	300	300	300	300

https://drive.google.com/file/d/1zOevapI-HyJo_-fDnBqK2Viy1_9o4L0E/view?usp=sharing

- Provides 1200 images divided into 4 classes as in the table
- The data has been divided into 400 for test and 800 for training

2. LIBRARIES

- **matplotlib.pyplot:** Matplotlib is a popular data visualization library in Python. pyplot provides a useful interface for creating various types of charts and graphs.
- **numpy (as np):** NumPy is a powerful library for numerical computation in Python. It provides support for large, multidimensional arrays and matrices, as well as mathematical functions to operate on these arrays.
- **keras.layers:** Keras is a high-level neural networks API and these modules provide the building blocks for building neural networks.
- **keras.models:** This module in Keras helps define and train models. Model is a class that allows you to create a model layer by layer.
- **keras_preprocessing.image:** This module provides tools for working with images in the context of neural networks. ImageDataGenerator is often used to augment data during training.
- **sklearn.metrics:** Scikit-learn is a machine learning library and the metrics module in it provides information such as accuracy, precision, recall, etc. to evaluate machine learning models.
- **tensorflow.keras.applications:** TensorFlow is an open source machine learning library and keras.applications contains pre-trained models for various tasks. VGG16 is a deep convolutional neural network model pre-trained on the ImageNet dataset.
- **tensorflow.keras.optimizers:** This module provides optimization algorithms for training neural networks. Adam is a popular optimization algorithm.
- **os:** The os module provides a way to interact with the operating system. It can be used for tasks such as reading or writing to the file system.
- **cv2:** OpenCV (Open Source Computer Vision Library) is a library for computer vision and image processing. cv2 provides functions for working with images and videos.
- **tensorflow.keras.utils:** This module contains utility functions for working with Keras. to_categorical is often used to transform class vectors into binary class matrices for categorical classification tasks.

3. CODES

Function to extract images and labels

```python
def seperate_images_labels(data_dir, label_indices):
    data = [] # List to store preprocessed images
    labels = [] # List to store corresponding labels
    img_format = ['.jpg', '.png', '.jpeg'] # Supported image formats
    # Iterate through each file in the specified directory
    for img_name in os.listdir(data_dir):
        # Check if the file has a valid image format
        if not any([img_name.endswith(img) for img in img_format]):
            continue
        # Create the full path to the image file
        img_path = os.path.join(data_dir, img_name)
        # Extract the label from the image name based on a specific pattern
        label_name = img_name.split('_')[0]
        label = label_indices[label_name]
        # Read and resize the image using OpenCV
        img = cv2.imread(img_path)
        img = cv2.resize(img, (224, 224))
        # Append the preprocessed image and its label to the respective lists
        data.append(img)
        labels.append(label)
    # Convert the lists to numpy arrays and return
    return np.array(data), np.array(labels)
```

data and labels Lists: These lists are used to store the preprocessed images and their corresponding labels.

img_format List: This list contains valid image file extensions (.jpg, .png, and .jpeg). The function only considers files with these extensions.

Loop through Files in the Directory: The function iterates through each file (img_name) in the specified directory (data_dir).It checks whether the file has a valid image format. If not, it skips to the next iteration.

Image Path and Label Extraction: It constructs the full path to the image file (img_path).It extracts the label from the image name based on a specific pattern. In this case, it assumes that the label is the part of the image name before the first underscore (_).

Read and Resize Image: It uses OpenCV (cv2) to read the image from the file and resize it to a common size (224x224 pixels).

Append to Lists: The preprocessed image (img) is appended to the data list.The corresponding label (label) is appended to the labels list.

Conversion to Numpy Arrays: Finally, the function converts the lists data and labels into numpy arrays using np.array() and returns them.

Function to create model

```
class CreateModel:
    def __init__(self, pre_train_model, num_classes=2):
        # Freeze the weights of the pre-trained model
        pre_train_model.trainable = False
        # Get the output of the pre-trained model
        last_output = pre_train_model.output
        # Flatten the output and add additional layers
        x = Flatten()(last_output)
        x = Dense(512, activation='relu')(x)
        x = Dense(256, activation='relu')(x)
        x = Dense(128, activation='relu')(x)
        x = Dropout(0.1)(x)
        x = Dense(64, activation='relu')(x)
        x = Dropout(0.2)(x)
        # Final output layer with softmax activation for classification
        output = Dense(num_classes, activation='softmax')(x)

        # Define optimizer, learning rate and loss function
        self.model = Model(pre_train_model.input, output)
        self.model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['acc'])
    # Train the model using the fit method
    def fit(self, train_generator, validation_generator, step_size_train, step_size_validation, epochs=20):
        history = self.model.fit(
            train_generator,
            steps_per_epoch=step_size_train,
            epochs=epochs,
            shuffle=False,
            verbose=1,
            validation_data=validation_generator,
            validation_steps=step_size_validation,
        )
        # Return the training history and the trained model
        return history, self.model
```

(__init__) Method:

pre_train_model: The pre-trained model, and its weights are frozen (trainable = False).

num_classes: The number of classes for the final classification layer (default is 2).

Flatten and Additional Dense Layers: The output of the pre-trained model is flattened. Additional dense layers with ReLU activation are added for feature extraction and non-linearity. Dropout layers are used for regularization.

Output Layer: The final output layer has num_classes nodes and uses softmax activation for multi-class classification.

Model Compilation: The new model is created using the input of the pre-trained model and the custom output. The model is compiled with the Adam optimizer, a specified learning rate, categorical cross-entropy loss, and accuracy as the evaluation metric.

Fit Method: The fit method is defined to train the model using data generators (train_generator and validation_generator). Training parameters such as the number of epochs, steps per epoch, and validation steps are specified.The training history and the trained model are returned.

Function to plot accuracy and loss

```
def plot_graphs(history, string):
    plt.plot(history.history[string])
    plt.plot(history.history["val_" + string])
    plt.xlabel("Epochs")
    plt.ylabel(string)
    plt.legend([string, "val_" + string])
    plt.show()
```

This function is used to plot the training and validation metrics (e.g., accuracy or loss) over epochs. It takes the training history (history) and the metric to be plotted (string) as parameters. The training and validation metrics are plotted on the same graph for comparison.

Function to plot confusion matrix

```
def confusion_matrix(test_generator, y_pred, class_labels=None):
    if class_labels is None:
        class_labels = list(test_generator.class_indices.keys())
        disp = metrics.ConfusionMatrixDisplay.from_predictions(test_generator.classes, y_pred, display_labels=class_labels,
                                                               xticks_rotation='vertical', cmap='Blues')
    else:
        disp = metrics.ConfusionMatrixDisplay.from_predictions(test_generator, y_pred, display_labels=class_labels,
                                                               xticks_rotation='vertical', cmap='Blues')
    fig = disp.ax_.get_figure()
    fig.set_figwidth(10)
    fig.set_figheight(10)
    plt.show()
```

This function uses scikit-learn's ConfusionMatrixDisplay to plot a confusion matrix. It takes the test data generator (test_generator), predicted labels (y_pred), and optional class labels. The confusion matrix is displayed with labels and color mapping.

Function to calculate report

```
def report_model(test_generator, y_pred, class_labels=None):
    if class_labels is None:
        class_labels = list(test_generator.class_indices.keys())
        true_classes = test_generator.classes
        report = metrics.classification_report(true_classes, y_pred, target_names=class_labels)
    else:
        report = metrics.classification_report(test_generator, y_pred, target_names=class_labels)
    print(report)
```

This function uses scikit-learn's classification_report to generate a text report of precision, recall, and F1-score for each class. It takes the test data generator (test_generator), predicted labels (y_pred), and optional class labels.

4. ACDC Dataset

This dictionary maps the class labels ('clear', 'fog', 'night', 'rain', 'snow') to numerical indices.
Data Loading:The separate_images_labels function is likely used to load and preprocess the images and their corresponding labels from the specified directories.
One-Hot Encoding:Convert the categorical labels to one-hot encoding.
Image Dimensions and Batch Size:These variables define the dimensions of the input images and the batch size for training.
Data Augmentation:Data augmentation is applied to the training and validation datasets using ImageDataGenerator. It includes rescaling, horizontal flipping, rotation, and brightness adjustments.

```
train_generator = train_datagen.flow(
    training_images, training_labels,
    batch_size=batch_size,
    shuffle=True,
    seed=19
)

validation_generator = val_datagen.flow(
    validation_images, validation_labels,
    batch_size=batch_size,
    shuffle=False,
    seed=19
)

test_generator = val_datagen.flow(
    test_images, test_labels,
    batch_size=1,
    shuffle=False,
    seed=19
)
```

```
step_size_train = train_generator.n // train_generator.batch_size
step_size_validation = validation_generator.n // validation_generator.batch_size
step_size_test = test_generator.n // test_generator.batch_size
```

```
pre_train_model_VGG16 = VGG16(
    include_top=False,
    weights="imagenet",
    input_shape=(img_width, img_height, 3)
)

model = CreateModel(pre_train_model_VGG16, num_classes=training_labels.shape[1])
history_acdc, model_acdc = model.fit(train_generator, validation_generator, step_size_train, step_size_validation, epochs=20)
```

Data Generators:Data generators are created for training, validation, and testing using the configured ImageDataGenerator instances.
Model Initialization (VGG16):The VGG16 pre-trained model is loaded with weights from ImageNet and is configured for the specified input shape.
Model Creation and Training:An instance of the CreateModel class is created, and the model is trained using the data generators for a specified number of epochs.

5. ADCD - RESULTS

Epoch 1/20
31/31 [==============================] - 22s 563ms/step - loss: 1.0668 - acc: 0.5968 - val_loss: 0.6157 - val_acc: 0.7750
Epoch 2/20
31/31 [==============================] - 19s 630ms/step - loss: 0.5318 - acc: 0.8054 - val_loss: 0.5222 - val_acc: 0.7854
.
.
.
Epoch 19/20
31/31 [==============================] - 19s 619ms/step - loss: 0.0757 - acc: 0.9768 - val_loss: 0.3754 - val_acc: 0.8938
Epoch 20/20
31/31 [==============================] - 20s 658ms/step - loss: 0.0610 - acc: 0.9778 - val_loss: 0.3847 - val_acc: 0.8833

This is a training log that shows the progress of the model during training. It includes information such as the number of batches processed (31/31), time taken per batch (20s), loss (0.0610 for training and 0.3847 for validation), and accuracy (0.9778 for training and 0.8833 for validation).

1

```
y_pred_tmp = model_acdc.predict(test_generator, step_size_test)
y_pred_acdc = np.argmax(y_pred_tmp, axis=1)
```

```
500/500 [==============================] - 8s 14ms/step
```

2

```
# Test model
loss = model_acdc.evaluate(test_generator, steps=test_generator.n)
print("loss test: {:.2f}\n".format(loss[0]), "\raccuracy test: {:0.2f}".format(loss[1]))
```

```
500/500 [==============================] - 8s 15ms/step - loss: 0.3825 - acc: 0.8740
loss test: 0.38
accuracy test: 0.87
```

3

```
plot_graphs(history_acdc, 'acc')
plot_graphs(history_acdc, 'loss')
```

1) The predict method is used to obtain raw predictions (probability scores) from the model for the test dataset.

model_acdc: This is the trained model obtained from the previous code. The model's predict method is used to generate predictions.

test_generator: This is a data generator providing the test images and labels.

step_size_test: It specifies the number of steps (batches) the generator should iterate over the test dataset.

The result (y_pred_tmp) is a numpy array containing probability scores for each class for each sample in the test dataset.

Class Label Determination:

y_pred_acdc = np.argmax(y_pred_tmp, axis=1)

The argmax function is used to determine the predicted class labels for each sample based on the highest probability score. The axis=1 argument indicates that the maximum value along axis 1 (class dimension) should be considered.

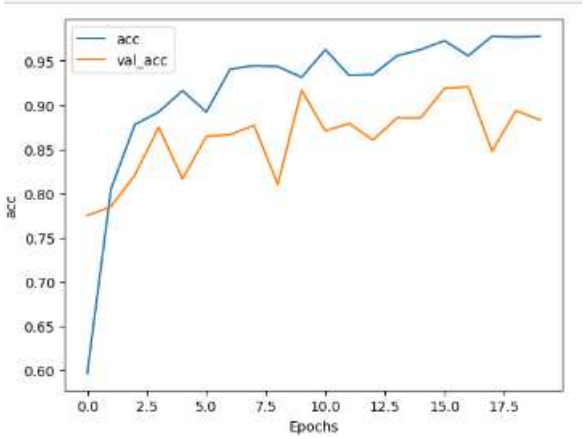
y_pred_tmp: This is the array of raw predictions obtained from the model.

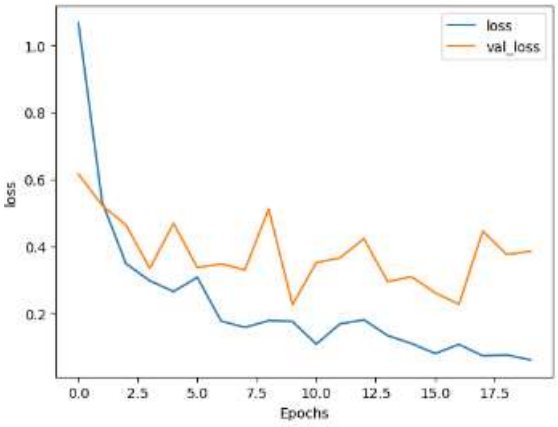
y_pred_acdc: This variable holds the final predicted class labels for each sample in the test dataset.

Now, y_pred_acdc contains the predicted class labels, and each value represents the predicted class for the corresponding test sample.

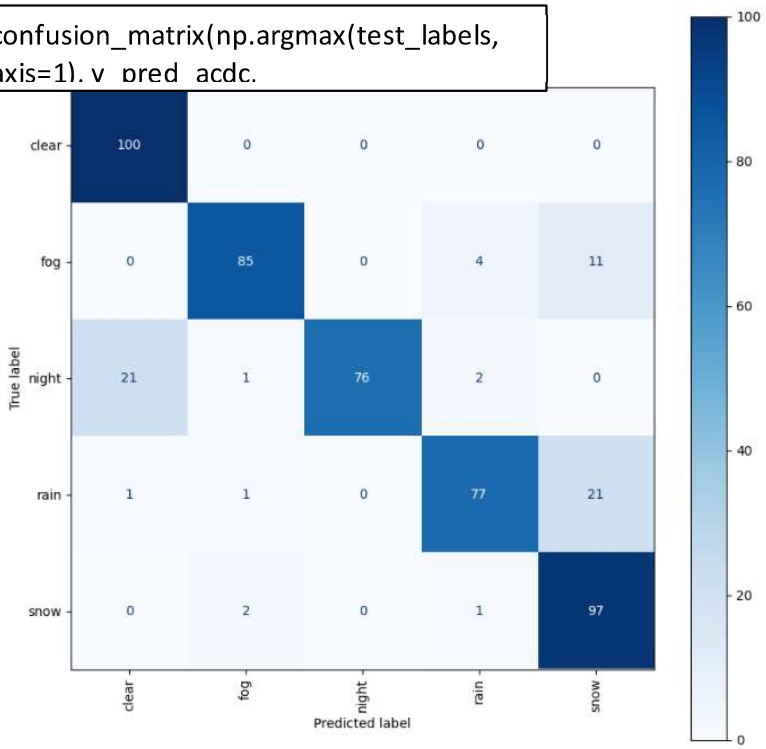
2) The evaluate method is used to compute the loss and accuracy on the test dataset. The results are printed, showing the loss and accuracy on the test set.

3)






```
confusion_matrix(np.argmax(test_labels,
axis=1), y_pred_acdc,
```



6. CONCLUSION – ADCD

This line generates and plots a confusion matrix, comparing the true labels (test_labels) against the predicted labels (y_pred_acdc). It uses the confusion_matrix function.

```
report_model(np.argmax(test_labels, axis=1), y_pred_acdc, class_labels=label_indices.keys())
```

	precision	recall	f1-score	support
clear	0.82	1.00	0.90	100
fog	0.96	0.85	0.90	100
night	1.00	0.76	0.86	100
rain	0.92	0.77	0.84	100
snow	0.75	0.97	0.85	100
accuracy			0.87	500
macro avg	0.89	0.87	0.87	500
weighted avg	0.89	0.87	0.87	500

For these dataset (ACDC) :

Precision : for the "clear" category, the precision is 0.82. This means that when the model predicts a sample as "clear," it is correct 82% of the time.

Recall : for the "night" category, the recall is 0.76. This means that the model correctly identifies 76% of the actual "night" samples.

F1-score : for the "fog" category, the F1-score is 0.90. This is a balanced measure that considers both precision and recall.

Support : is the number of actual occurrences of the class. In this case, there are 100 instances for each weather condition.

The overall accuracy of the model is 0.87 or 87%. This means that, overall, the model correctly predicts the weather conditions for 87% of the samples.

Strengths:

+ High Precision for Clear, Fog, and Rain:

The model has high precision for the "clear," "fog," and "rain" categories (0.82, 0.96, and 0.92, respectively). This indicates that when the model predicts these conditions, it is often correct.

+ High Recall for Snow:

The model has a high recall for the "snow" category (0.97). This suggests that the model is good at capturing most of the actual instances of snowy weather.

+ Balanced F1-Scores:

The F1-scores for all classes are relatively balanced, indicating a good compromise between precision and recall. This suggests that the model performs well across multiple weather conditions.

Weaknesses:

- Lower Recall for Night:

The model has a lower recall for the "night" category (0.76). This suggests that the model might be missing some instances of nighttime conditions.

- Lower Precision for Snow:

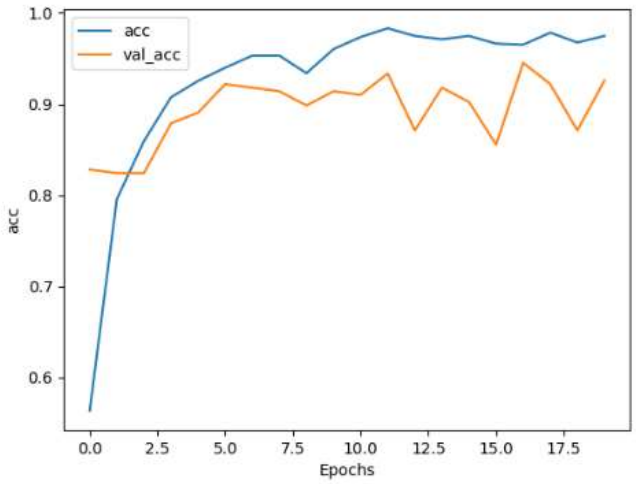
The precision for the "snow" category is 0.75. This indicates that when the model predicts snowy weather, it has a higher chance of making a false positive.

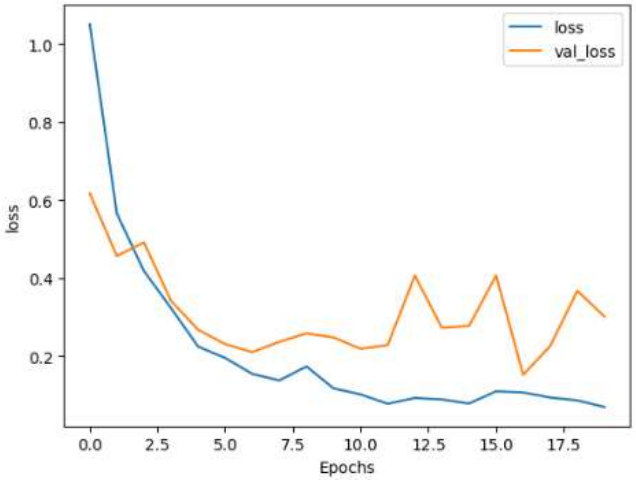
7. MWD Dataset - RESULTS

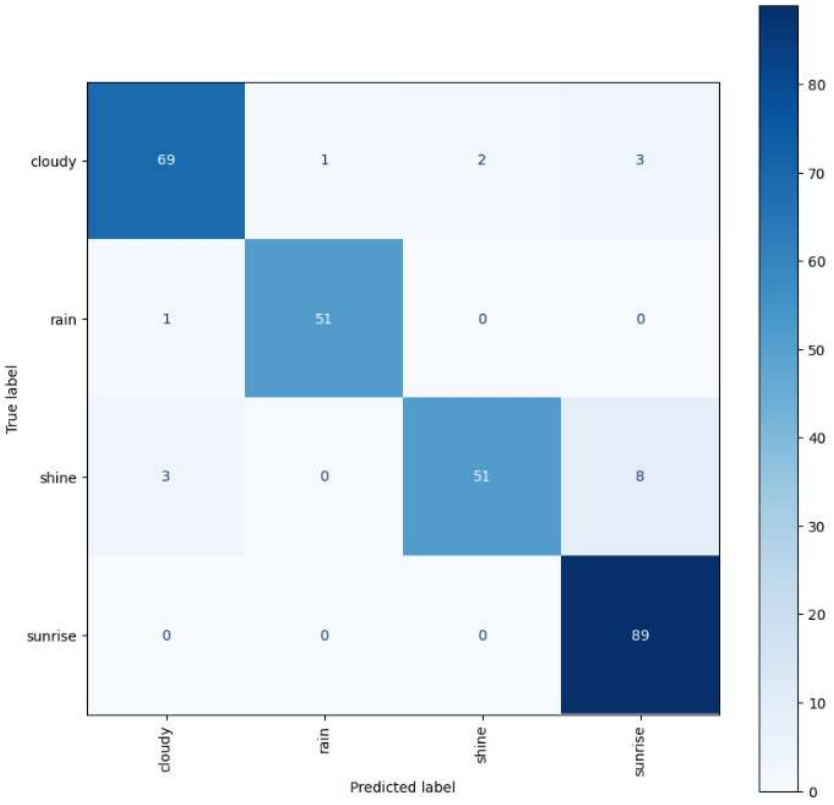
```
Epoch 1/20
26/26 [==============================] - 26s 985ms/step - loss: 1.0504 - acc: 0.5637 -
val_loss: 0.6167 - val_acc: 0.8281
Epoch 2/20
26/26 [==============================] - 15s 583ms/step - loss: 0.5651 - acc: 0.7957 -
val_loss: 0.4567 - val_acc: 0.8242
.

.

.
Epoch 19/20
26/26 [==============================] - 18s 697ms/step - loss: 0.0865 - acc: 0.9675 -
val_loss: 0.3676 - val_acc: 0.8711
Epoch 20/20
26/26 [==============================] - 17s 649ms/step - loss: 0.0697 - acc: 0.9748 -
val_loss: 0.3012 - val_acc: 0.9258
```


acc
val_acc
acc
Epochs
1.0
0.9
0.8
0.7
0.6
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


loss
val_loss
loss
Epochs
1.0
0.8
0.6
0.4
0.2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


cloudy
rain
shine
sunrise
True label
69
1
2
3
1
51
0
0
3
0
51
8
0
0
0
89
cloudy
rain
shine
sunrise
Predicted label
80
70
60
50
40
30
20
10
0

8. CONCLUSION – MWD

```
: report_model(test_generator, y_pred_mwd)
```

	precision	recall	f1-score	support
cloudy	0.95	0.92	0.93	75
rain	0.98	0.98	0.98	52
shine	0.96	0.82	0.89	62
sunrise	0.89	1.00	0.94	89
accuracy			0.94	278
macro avg	0.94	0.93	0.94	278
weighted avg	0.94	0.94	0.93	278

Strengths:

+ High Precision and Recall for Rain:

The model has very high precision (0.98) and recall (0.98) for the "rain" category. This indicates that the model is highly accurate at identifying rainy conditions, with very few false positives or false negatives.

+ High Precision for Cloudy:

The model has high precision (0.95) for the "cloudy" category, indicating that when it predicts cloudy conditions, it is correct 95% of the time.

+ High Recall for Sunrise:

The model has high recall (1.00) for the "sunrise" category, meaning it successfully captures all instances of sunrise conditions.

+ High Overall Accuracy:

The overall accuracy of the model is 94%, suggesting that it performs well across all classes.

+ Balanced F1-Scores

Weaknesses:

- Lower Recall for Shine:

The model has lower recall (0.82) for the "shine" category. This suggests that the model may miss some instances of sunny conditions.

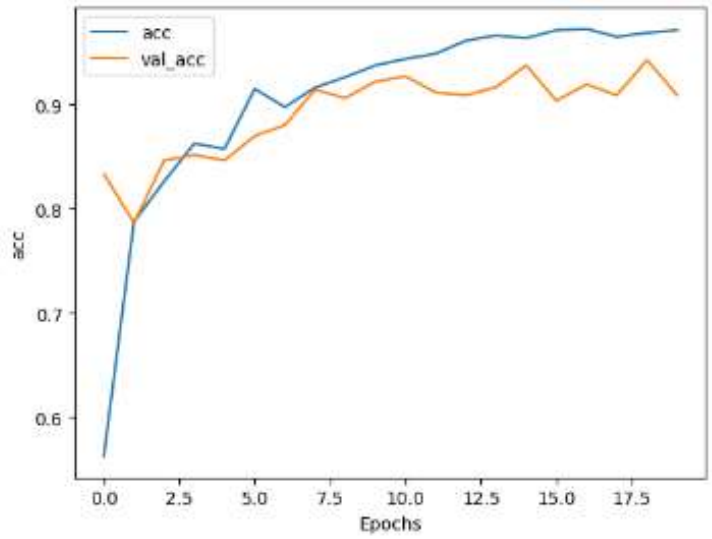
9. Syndrone Dataset – RESULTS

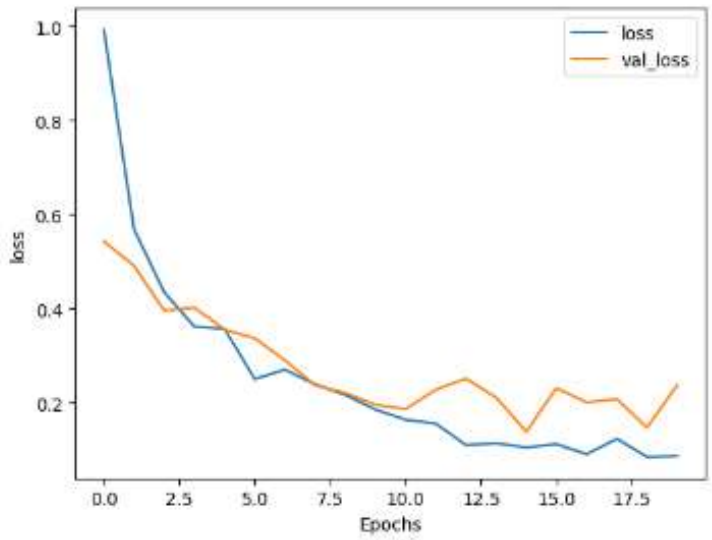
label_indices = {'ClearNight':0, 'ClearNoon':1, 'HardRainNoon':2, 'MidFoggyNoon':3}

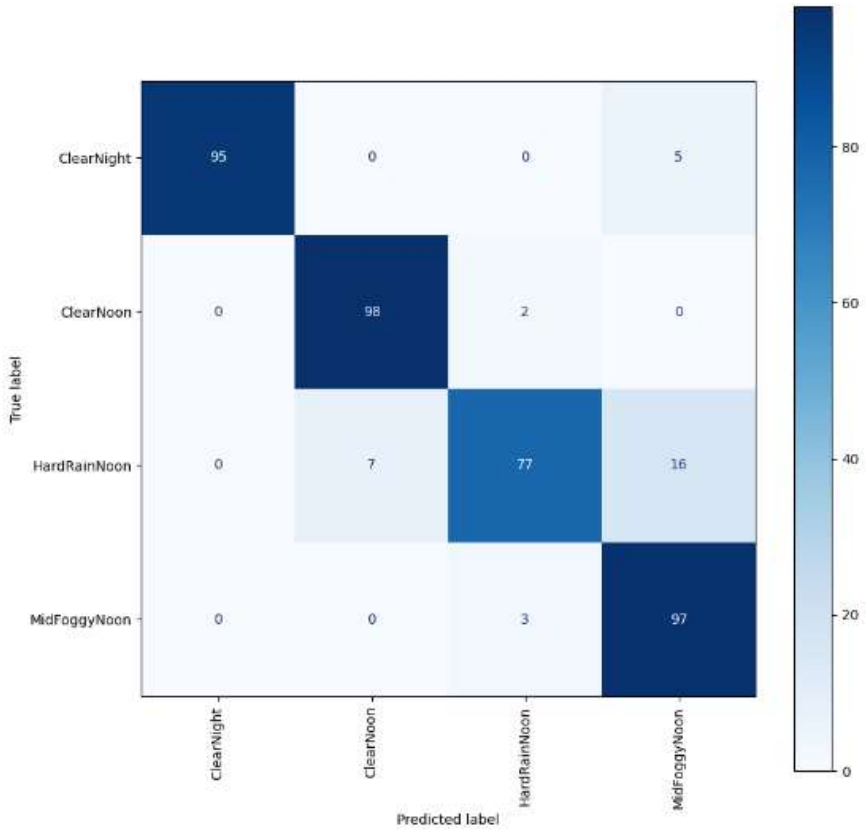
Epoch 1/20
25/25 [==============================] - 15s 562ms/step - loss: 0.9929 - acc: 0.5625 - val_loss: 0.5428 - val_acc: 0.8333
Epoch 2/20
25/25 [==============================] - 16s 630ms/step - loss: 0.5686 - acc: 0.7875 - val_loss: 0.4907 - val_acc: 0.7865
.

.

.
Epoch 19/20
25/25 [==============================] - 14s 570ms/step - loss: 0.0847 - acc: 0.9688 - val_loss: 0.1470 - val_acc: 0.9427
Epoch 20/20
25/25 [==============================] - 15s 598ms/step - loss: 0.0867 - acc: 0.9712 - val_loss: 0.2373 - val_acc: 0.9089







10. CONCLUSION – Syndrone

```
report_model(np.argmax(test_labels, axis=1), y_pred_syndrone, class_labels=label_indices.keys())
```

	precision	recall	f1-score	support
ClearNight	1.00	0.95	0.97	100
ClearNoon	0.93	0.98	0.96	100
HardRainNoon	0.94	0.77	0.85	100
MidFoggyNoon	0.82	0.97	0.89	100
accuracy			0.92	400
macro avg	0.92	0.92	0.92	400
weighted avg	0.92	0.92	0.92	400

Strengths:

+ Perfect Precision for ClearNight:

The model has perfect precision (1.00) for the "ClearNight" category, indicating that when it predicts ClearNight conditions, it is always correct.

+ High Precision for ClearNoon:

The model has high precision (0.93) for the "ClearNoon" category, suggesting that it correctly identifies ClearNoon conditions with a high degree of accuracy.

+ High Recall for MidFoggyNoon:

The model has high recall (0.97) for the "MidFoggyNoon" category, meaning it successfully captures most instances of MidFoggyNoon conditions.

+ High Overall Accuracy:

The overall accuracy of the model is 92%, indicating that it performs well across all classes.

+ Balanced F1-Scores

Weaknesses:

– Lower Recall for HardRainNoon:

The model has lower recall (0.77) for the "HardRainNoon" category. This suggests that the model may miss some instances of HardRainNoon conditions.

11. UAVid Dataset – RESULTS

label_indices = {'day':0, 'fog':1, 'night':2, 'rain':3}

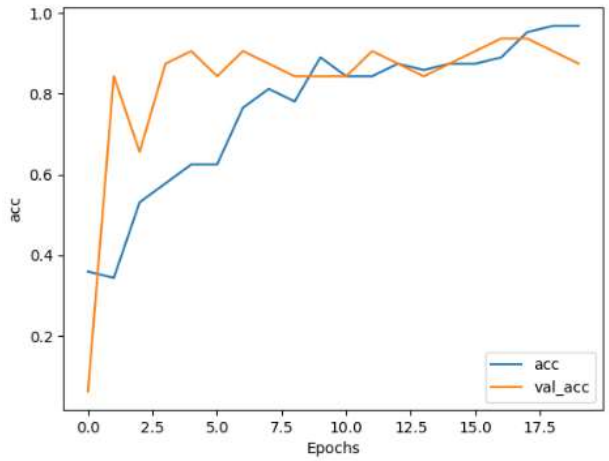
history_uavid, model_uavid = model.fit(train_generator, validation_generator, step_size_train, step_size_validation, epochs=20)

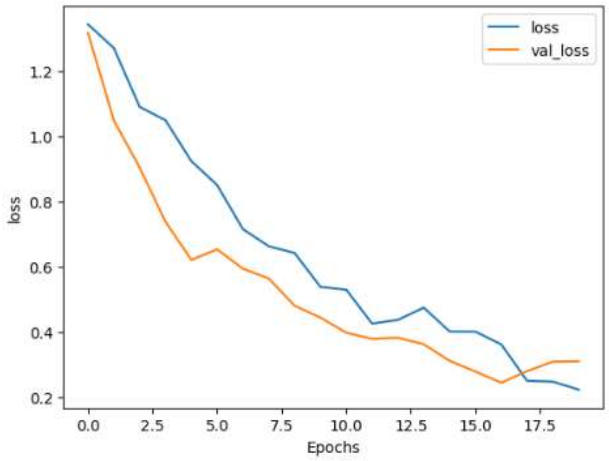
Epoch 1/20
2/2 [==============================] - 3s 1s/step - loss: 1.3421 - acc: 0.3594 - val_loss: 1.3159 - val_acc: 0.0625
Epoch 2/20
2/2 [==============================] - 1s 997ms/step - loss: 1.2697 - acc: 0.3438 - val_loss: 1.0478 - val_acc: 0.8438
.

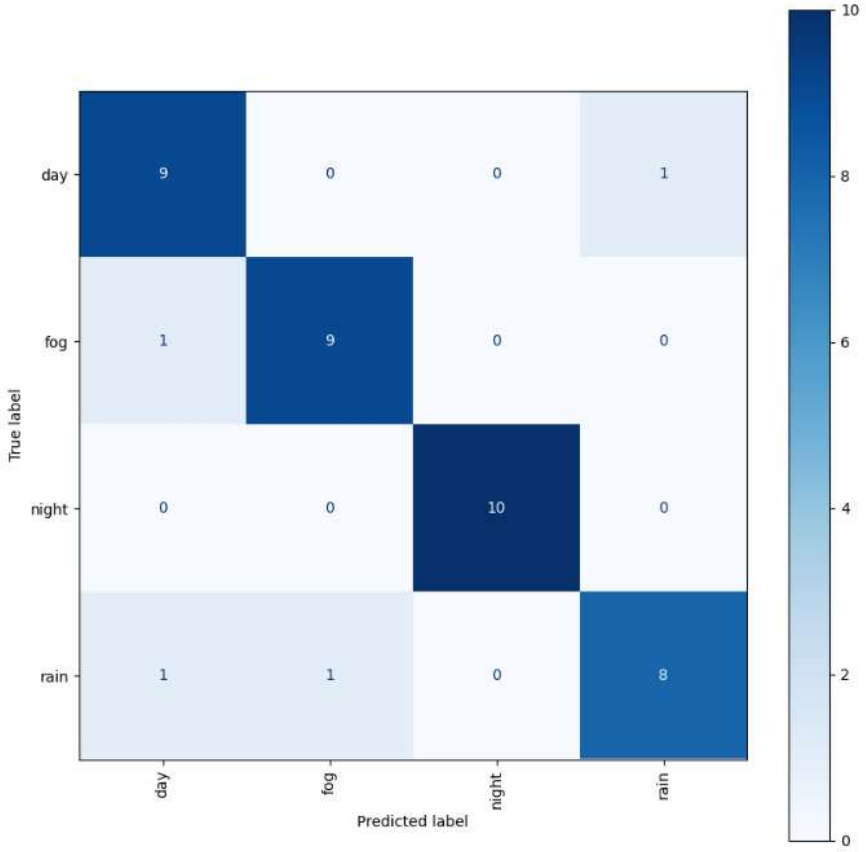
.

.

Epoch 19/20
2/2 [==============================] - 1s 794ms/step - loss: 0.2469 - acc: 0.9688 - val_loss: 0.3083 - val_acc: 0.9062
Epoch 20/20
2/2 [==============================] - 1s 808ms/step - loss: 0.2226 - acc: 0.9688 - val_loss: 0.3096 - val_acc: 0.8750


acc
val_acc
Epochs
1.0
0.8
0.6
0.4
0.2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


loss
val_loss
Epochs
1.2
1.0
0.8
0.6
0.4
0.2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


day
fog
night
rain
True label
Predicted label
9
0
0
1
1
9
0
0
0
0
10
0
1
1
0
8
10
8
6
4
2
0

12. CONCLUSION – UAVid

```
report_model(np.argmax(test_labels, axis=1), y_pred_uavid, class_labels=label_indices.keys())
```

	precision	recall	f1-score	support
day	0.82	0.90	0.86	10
fog	0.90	0.90	0.90	10
night	1.00	1.00	1.00	10
rain	0.89	0.80	0.84	10
accuracy			0.90	40
macro avg	0.90	0.90	0.90	40
weighted avg	0.90	0.90	0.90	40

Strengths:

+ Perfect Precision and Recall for Night:

The model has perfect precision (1.00) and recall (1.00) for the "night" category, indicating that it correctly identifies all instances of nighttime conditions without any false positives or false negatives.

+ High Precision and Recall for Fog:

The model has high precision (0.90) and recall (0.90) for the "fog" category, suggesting accurate identification of foggy conditions.

+ Balanced F1-Scores
+ High Overall Accuracy:

The overall accuracy of the model is 90%, indicating that it performs well across all classes.

Weaknesses:

– Lower Recall for Rain:

The model has lower recall (0.80) for the "rain" category. This suggests that the model may miss some instances of rainy conditions.

– Day Class Precision:

The precision for the "day" category is 0.82, indicating that the model has some false positives for daytime conditions.

13. REFERENCES

https://stackoverflow.com/

https://www.geeksforgeeks.org/

https://www.tensorflow.org/api_docs/python/tf/keras/utils

https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image

https://www.tensorflow.org/api_docs/python/tf/keras/layers

https://matplotlib.org/3.5.3/api/_as_gen/matplotlib.pyplot.html

https://www.tensorflow.org/api_docs/python/tf/keras/Model

https://www.tensorflow.org/api_docs/python/tf/keras/applications

https://www.tensorflow.org/api_docs/python/tf/keras/optimizers