Add libraries

In [1]:
```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3' #  Turn off some unimportant messages
```

In [2]:
```
import tensorflow as tf
from keras.applications.vgg19 import VGG19
from keras.applications.resnet import ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten, Dense, Dropout
import numpy as np
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.regularizers import l1_l2
from tensorflow.keras.optimizers import Adam
import seaborn as sns
```

```
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
E0000 00:00:1739952653.869597    1035 cuda_dnn.cc:8310] Unable to register cuDNN factory: Attempting to register factory for pl
ugin cuDNN when one has already been registered
E0000 00:00:1739952653.892811    1035 cuda_blas.cc:1418] Unable to register cuBLAS factory: Attempting to register factory for
plugin cuBLAS when one has already been registered
```

Path system

In [3]:
```
# you need the current working directory NB: works both windows and linux
current_working_directory = os.getcwd()
current_working_directory = os.path.dirname(current_working_directory)

# get the directory where I want to download the dataset
path_of_download = os.path.join(*['..', current_working_directory,'Assignment2','weather_dataset'])
print(f"[DIR] The directory of the current dataset is {path_of_download}")
```

```
[DIR] The directory of the current dataset is /mnt/c/UTU/2025/Computer Vision and Sensor Fusion/Assignment2/weather_dataset
```

function for data loading

In [4]:
```python
# here let s do some functions that we can re-use also for other assignment
def load_the_data_and_the_labels(data_set_path: str, target_size: tuple or None = None):
    try:
        dataset, labels, name_of_the_labels = list(), list(), list()
        # let s loop here and we try to discover how many class we have
        for class_number, class_name in enumerate(os.listdir(data_set_path)):
            full_path_the_data = os.path.join(data_set_path, class_name)
            print(f"[WALK] I am walking into {full_path_the_data}")

            # add the list to nam _list
            name_of_the_labels.append(class_name)

            for single_image in os.listdir(f"{full_path_the_data}"):
                full_path_to_image = os.path.join(*[full_path_the_data, single_image])

                # add the class number
                labels.append(class_number)

                if target_size is None:
                    # let s load the image
                    image = tf.keras.utils.load_img(full_path_to_image)
                else:
                    image = tf.keras.utils.load_img(full_path_to_image, target_size=target_size)

                # transform PIL object in image
                image = tf.keras.utils.img_to_array(image)

                # add the image to the ds list
                dataset.append(image)

        return np.array(dataset, dtype='uint8'), np.array(labels, dtype='int'), name_of_the_labels
    except Exception as ex:
        print(f"[EXCEPTION] load the data and the labels throws exceptions {ex}")
```

OHE function

In [5]:
```
# here we have to one hot encode the labes
def make_the_one_hot_encoding(labels_to_transform):
    try:
        enc = OneHotEncoder(handle_unknown='ignore')
        # this is a trick to figure the array as 2d array instead of list
        temp = np.reshape(labels_to_transform, (-1, 1))
        labels_to_transform = enc.fit_transform(temp).toarray()
        print(f'[ONE HOT ENCODING] Labels are one-hot-encoded: {(labels_to_transform.sum(axis=1) - np.ones(labels_to_transform
        return labels_to_transform
    except Exception as ex:
        print(f"[EXCEPTION] Make the one hot encoding throws exception {ex}")
```

load the data and labels

In [6]:
```
# Resize the image to the correct size for VGG19
target_size = (224, 224, 3) # 224 x 224 pixels, 3 RGB color channels

# Call function to load data
X, y, class_names = load_the_data_and_the_labels(path_of_download, target_size)

# Print data info after loading
print(f"Dataset shape: {X.shape}")
print(f"Labels shape: {y.shape}")
print(f"Classes: {class_names}")
```

```
[WALK] I am walking into /mnt/c/UTU/2025/Computer Vision and Sensor Fusion/Assignment2/weather_dataset/Cloudy
[WALK] I am walking into /mnt/c/UTU/2025/Computer Vision and Sensor Fusion/Assignment2/weather_dataset/Rain
[WALK] I am walking into /mnt/c/UTU/2025/Computer Vision and Sensor Fusion/Assignment2/weather_dataset/Shine
[WALK] I am walking into /mnt/c/UTU/2025/Computer Vision and Sensor Fusion/Assignment2/weather_dataset/Sunrise
Dataset shape: (1125, 224, 224, 3)
Labels shape: (1125,)
Classes: ['Cloudy', 'Rain', 'Shine', 'Sunrise']
```

split the dataset in train and test set (ratio 0.3)

In [7]:
```
#  Divide data into training set (70%) and validation set (30%)
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
# Print train, and test size
print(f"Train size: {X_train.shape}, Validation size: {X_val.shape}")
print(f"Validation size: {X_val.shape}")
```

```
Train size: (787, 224, 224, 3), Validation size: (338, 224, 224, 3)
Validation size: (338, 224, 224, 3)
```

normalize the data

In [8]:
```
X_train = X_train / 255.0
X_val = X_val / 255.0
```

create the CNN and set all parameters to trainable

a. Input layer b. As base model use VGG19: i. Weights: imagenet ii. Include_top: False iii. Input_shape the target shape described in point 1. c. Add a flatten layer d. Add a Dense layer with 512 units and a dropout layer with 0.1 unit. e. Add a Dense layer with 256 units and a dropout layer with 0.1 unit. f. Add the final classifier with the correct number of units and the suitable activation.

In [9]:
```
# Set trainable (True, False)
isTrainable = True

baseModel = VGG19(input_shape=target_size, weights='imagenet', include_top=False) # Input model use VGG19 use imagenet weight

# Freeze all layers of VGG19
for layer_ctn, layer in enumerate(baseModel.layers[:]):
    layer.trainable = isTrainable

# Flatten layer to convert from 4D tensor -> 1D vector
x = Flatten()(baseModel.output)

# Fully Connected Layers (Dense + Dropout)
x = Dense(512, activation='relu', kernel_regularizer=l1_l2(l1=0, l2=0.01))(x) # Add a Dense layer with 512 units
x = Dropout(0.1)(x) # Dropout layer with 0.1 unit
x = Dense(256, activation='relu', kernel_regularizer=l1_l2(l1=0, l2=0.01))(x) # Add a Dense layer with 256 units
x = Dropout(0.1)(x) # Dropout layer with 0.1 unit

# Output layer with 4 classes for the final classifier
x = Dense(4, activation='softmax')(x)  # 4 classes: ['Cloudy', 'Rain', 'Shine', 'Sunrise']
```

```
# Combine base_model and Fully Connected layers into a final model
model = Model(inputs=baseModel.input, outputs=x)

model.summary() # Print mode summary
```

```
I0000 00:00:1739952668.428653    1035 gpu_device.cc:2022] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 5520 MB memory:  -> device: 0, name: NVIDIA GeForce RTX 4070 Laptop GPU, pci bus id: 0000:01:00.0, compute capability: 8.9
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv4 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv4 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808

block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv4 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 512)	12,845,568
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131,328
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 4)	1,028

Total params: 33,002,308 (125.89 MB)

Trainable params: 33,002,308 (125.89 MB)

Non-trainable params: 0 (0.00 B)

compile the model with adam

In [10]:
```
# Compile model with Adam
model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='categorical_crossentropy', # Use categorical_crossentrop for multi-class classification
    metrics=['accuracy']
)
```

Fit the model with batch size 32 and 15 epochs (This take 15 - 20 minutes with the CPU)

In [11]:
```
# Use one_hot_encoding to convert each label into a binary vector for training
y_train = make_the_one_hot_encoding(y_train)
y_val = make_the_one_hot_encoding(y_val)
```

```
[ONE HOT ENCODING] Labels are one-hot-encoded: True
[ONE HOT ENCODING] Labels are one-hot-encoded: True
```

In [12]:
```python
# Train model with epochs = 15, batch_size = 32
history = model.fit(X_train, y_train,
                    epochs=15, batch_size=32,
                    validation_data=(X_val, y_val))
```

```
Epoch 1/15
```

```
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1739952672.729650    1208 service.cc:148] XLA service 0x7f9b0c01e130 initialized for platform CUDA (this does not guarantee that XLA will be used). Devices:
I0000 00:00:1739952672.730724    1208 service.cc:156]   StreamExecutor device (0): NVIDIA GeForce RTX 4070 Laptop GPU, Compute Capability 8.9
I0000 00:00:1739952673.194164    1208 cuda_dnn.cc:529] Loaded cuDNN version 90300
I0000 00:00:1739952698.588621    1208 device_compiler.h:188] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
```

```
25/25 ━━━━━━━━━━━━━━━━━━━━ 74s 2s/step - accuracy: 0.2724 - loss: 63.2936 - val_accuracy: 0.2249 - val_loss: 5.5560
Epoch 2/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 9s 354ms/step - accuracy: 0.2610 - loss: 5.1254 - val_accuracy: 0.2959 - val_loss: 4.2792
Epoch 3/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 9s 344ms/step - accuracy: 0.3210 - loss: 4.1248 - val_accuracy: 0.2959 - val_loss: 3.7886
Epoch 4/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 8s 338ms/step - accuracy: 0.3339 - loss: 3.7067 - val_accuracy: 0.2959 - val_loss: 3.5262
Epoch 5/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 8s 337ms/step - accuracy: 0.3312 - loss: 3.4679 - val_accuracy: 0.2959 - val_loss: 3.3636
Epoch 6/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 8s 337ms/step - accuracy: 0.3121 - loss: 3.3261 - val_accuracy: 0.2959 - val_loss: 3.2502
Epoch 7/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 8s 339ms/step - accuracy: 0.3179 - loss: 3.2147 - val_accuracy: 0.2959 - val_loss: 3.1630
Epoch 8/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 9s 339ms/step - accuracy: 0.3281 - loss: 3.1316 - val_accuracy: 0.2959 - val_loss: 3.0930
Epoch 9/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 8s 340ms/step - accuracy: 0.3157 - loss: 3.0686 - val_accuracy: 0.2959 - val_loss: 3.0309
Epoch 10/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 9s 340ms/step - accuracy: 0.3537 - loss: 2.9908 - val_accuracy: 0.2959 - val_loss: 2.9789
Epoch 11/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 8s 338ms/step - accuracy: 0.3113 - loss: 2.9547 - val_accuracy: 0.2959 - val_loss: 2.9307
Epoch 12/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 9s 341ms/step - accuracy: 0.3554 - loss: 2.8950 - val_accuracy: 0.2959 - val_loss: 2.8884
Epoch 13/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 9s 358ms/step - accuracy: 0.3125 - loss: 2.8626 - val_accuracy: 0.2959 - val_loss: 2.8490
Epoch 14/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 9s 346ms/step - accuracy: 0.3444 - loss: 2.8210 - val_accuracy: 0.2959 - val_loss: 2.8120
Epoch 15/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 9s 341ms/step - accuracy: 0.3233 - loss: 2.7853 - val_accuracy: 0.2959 - val_loss: 2.7789
```

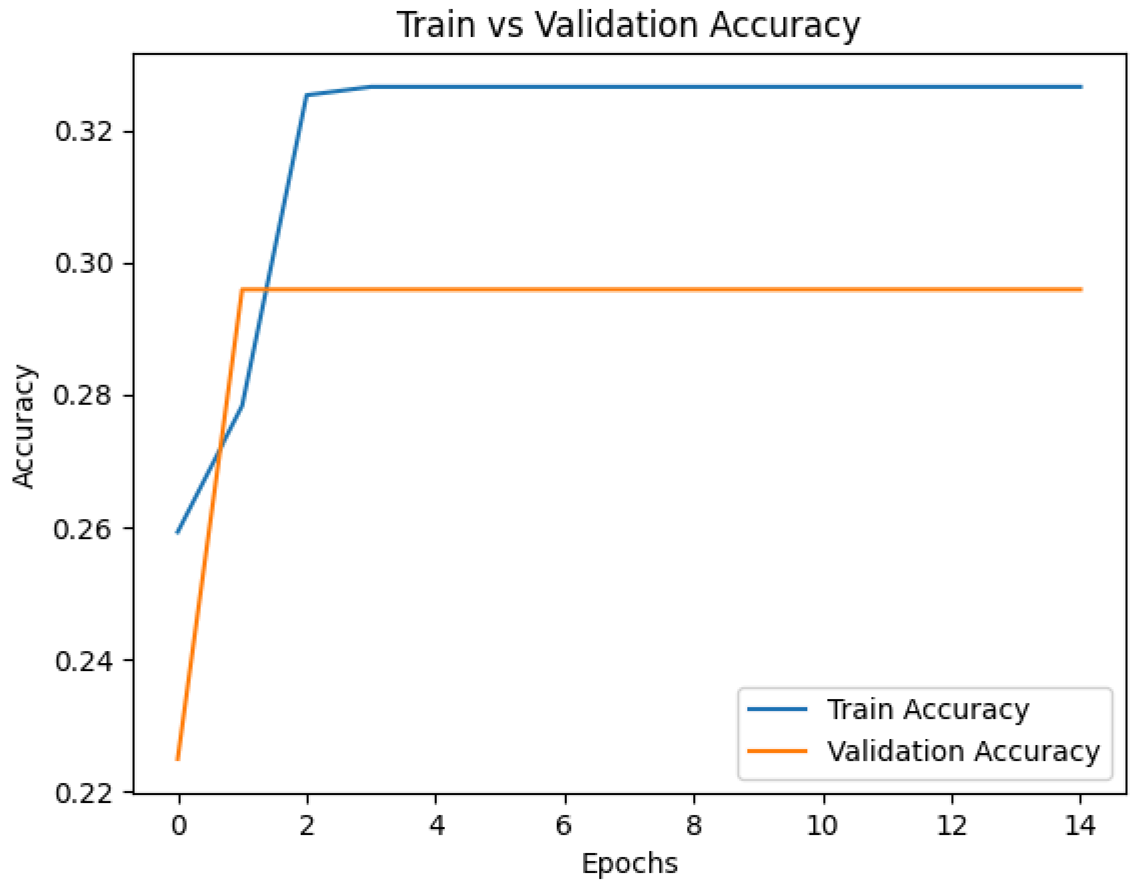
Evaluate the model

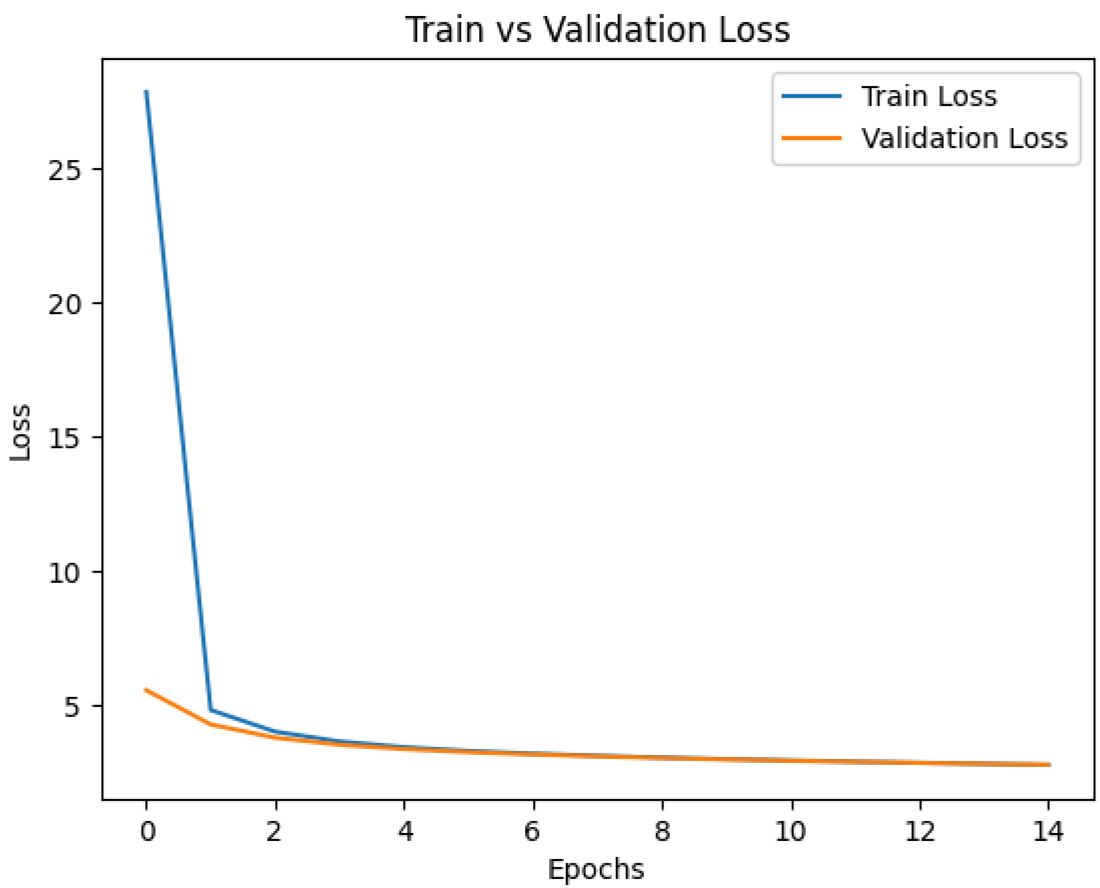
In [13]:
```python
test_loss, test_acc = model.evaluate(X_val, y_val)
print(f"Accuracy: {test_acc*100:.2f}%")
```

```
11/11 ━━━━━━━━━━━━━━━━━━━━ 1s 84ms/step - accuracy: 0.2940 - loss: 2.7851
Accuracy: 29.59%
```

Draw a chart of the training process to check overfitting/underfitting

In [14]:
```
# Plot accuracy graph
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Train vs Validation Accuracy')
plt.show()

# Plot loss graph
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Train vs Validation Loss')
plt.show()
```


Train vs Validation Accuracy
Accuracy
0.32
0.30
0.28
0.26
0.24
0.22
0
2
4
6
8
10
12
14
Epochs
Train Accuracy
Validation Accuracy



Train vs Validation Accuracy

- Trend: Accuracy increases gradually with the number of epochs
- Low train accuracy (~32%): The model is learning badly on the training set
- Low validation accuracy (~30%): The model generates badly
- Train-validation distance: The gap is not too far; but the training and validation scores are very bad, which illustrates the model does not generate well

Train vs Validation Loss

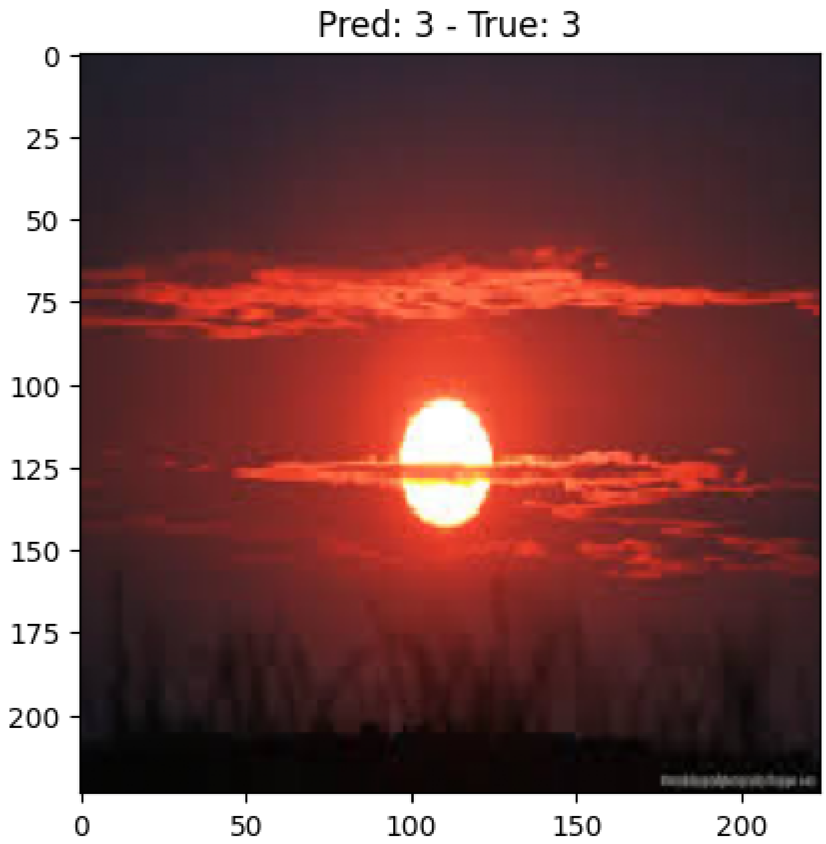
- Trend: Loss of both training set and validation set are gradually decreasing.

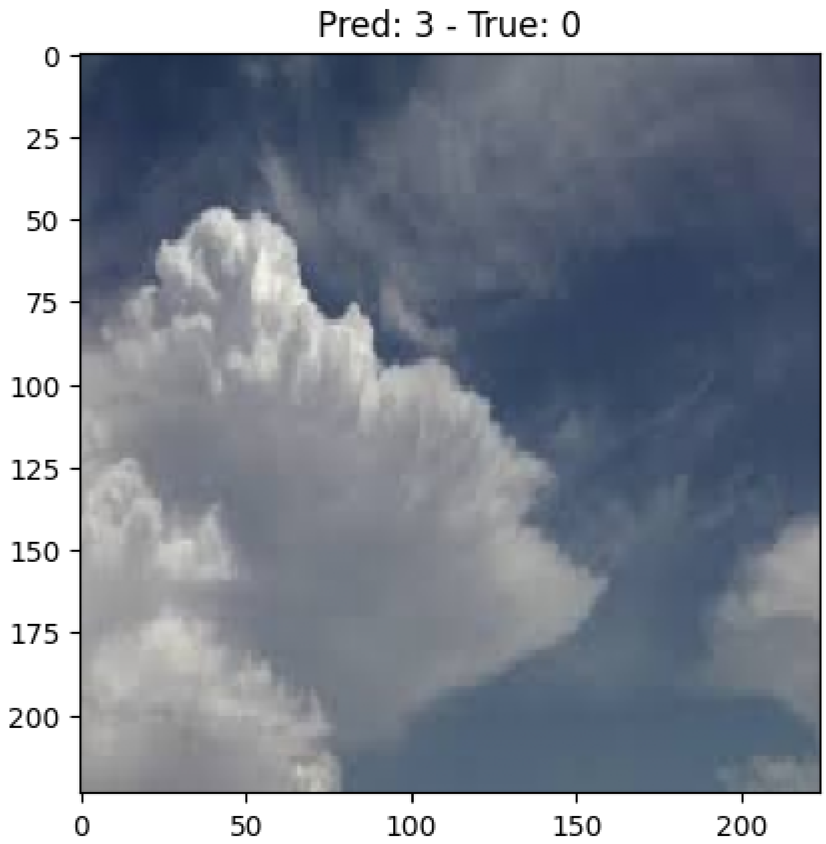
- Loss distance: The gap is quite close to each other, which shows the model is not overfitting.
- Loss reduction: There are no large fluctuations; providing that training is stable

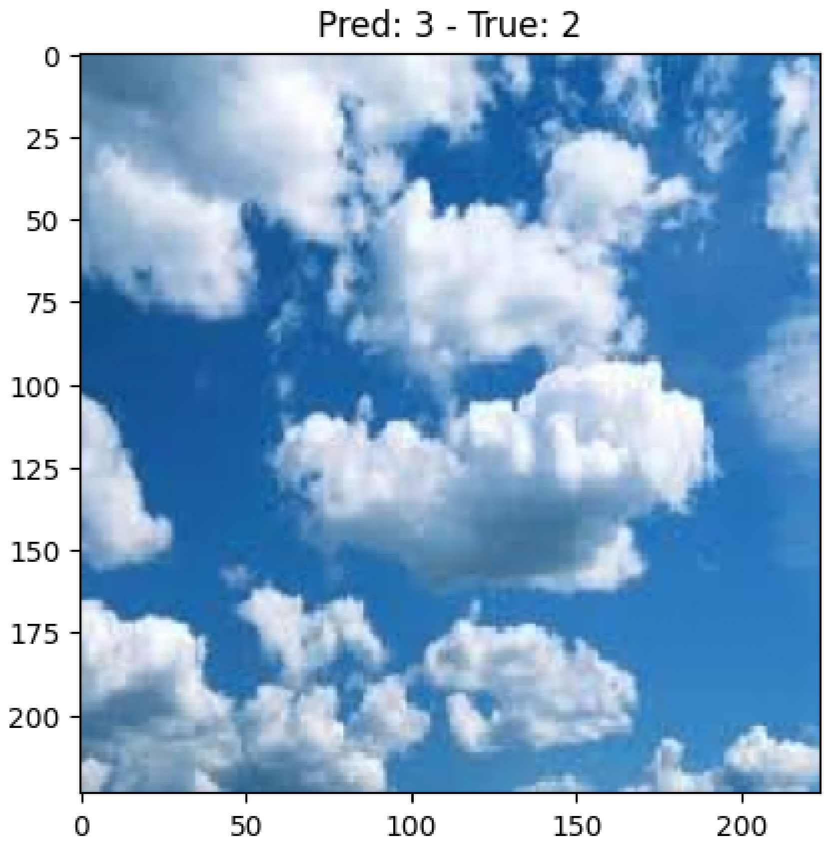
Make and show predictions

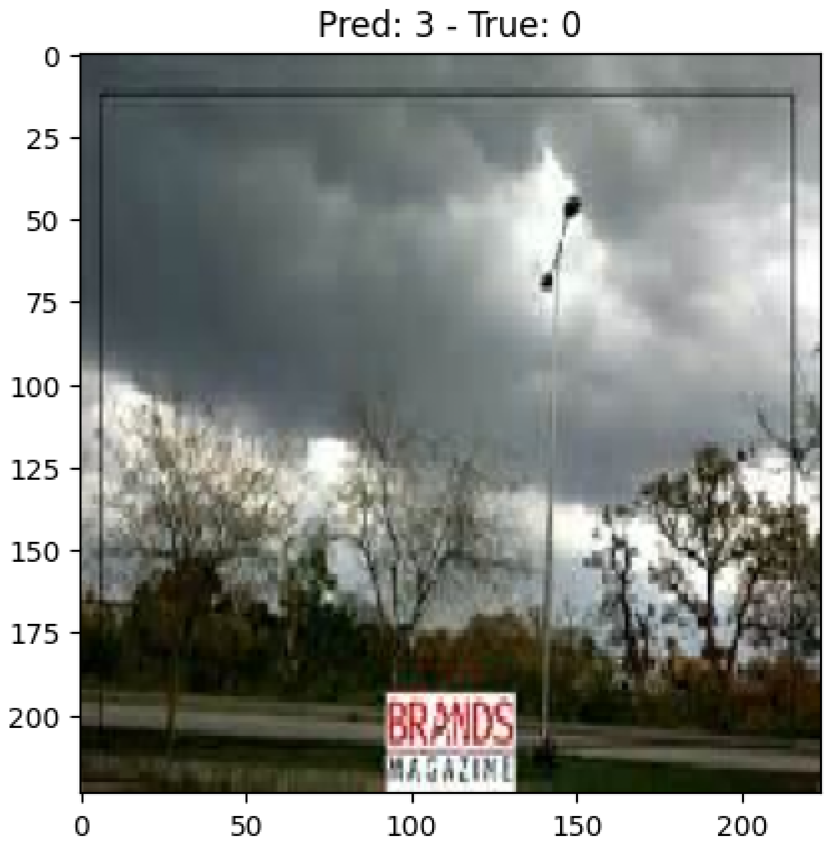
In [15]:
```
# Chose random 5 samples from validation
num_samples = 5
indices = np.random.choice(len(X_val), num_samples, replace=False)
sample_images = X_val[indices]
sample_labels = y_val[indices]

# Prediction
predictions = model.predict(sample_images)

# Plot the result, 4 classes: ['Cloudy', 'Rain', 'Shine', 'Sunrise']
for i in range(num_samples):
    plt.imshow(sample_images[i])
    plt.title(f'Pred: {np.argmax(predictions[i])} - True: {np.argmax(sample_labels[i])}')
    plt.show()
```

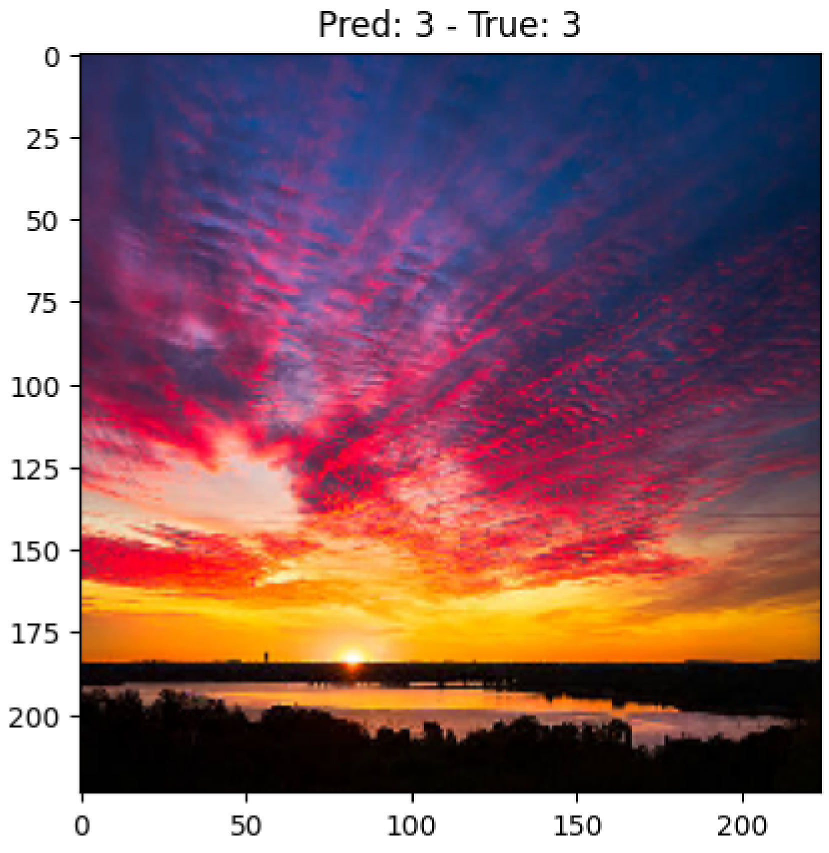
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
```


Pred: 3 - True: 3
0
25
50
75
100
125
150
175
200
0
50
100
150
200


Pred: 3 - True: 0
0
25
50
75
100
125
150
175
200
0
50
100
150
200


Pred: 3 - True: 2
0
25
50
75
100
125
150
175
200
0
50
100
150
200


Pred: 3 - True: 0
0
25
50
75
100
125
150
175
200
0
50
100
150
200
BRANDS
MAGAZINE

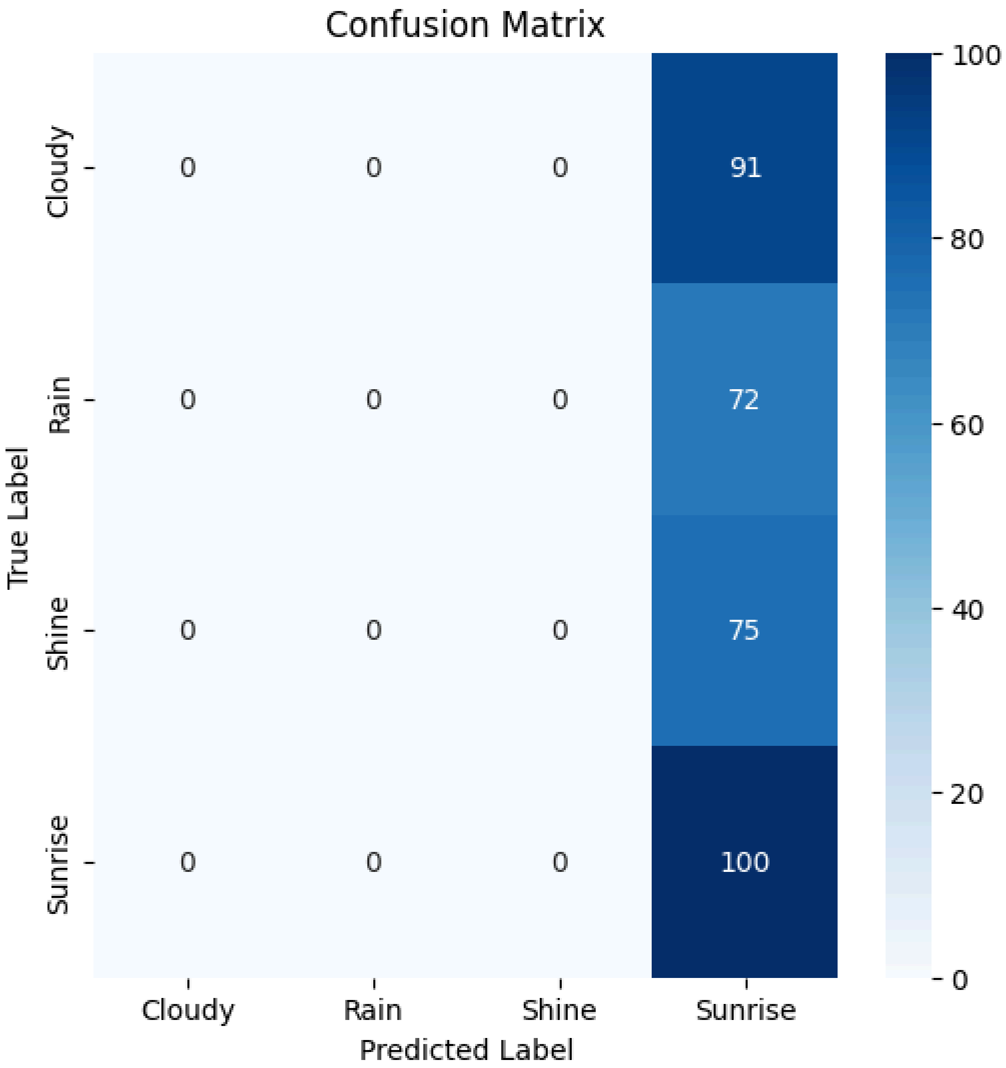


make confusion matrix

In [16]:
```
# Predict the probabilities of classes on the validation set
y_pred_probs = model.predict(X_val)

# Convert to a label by taking the value with the highest probability
y_pred = np.argmax(y_pred_probs, axis=1)

# Convert it back to label format when y_val is in one-hot encoding
y_true = np.argmax(y_val, axis=1)

# Definition of class labels
class_labels = ['Cloudy', 'Rain', 'Shine', 'Sunrise']
```

```
# Create confusion matrix
cm = confusion_matrix(y_true, y_pred)

# Plot confusion matrix
plt.figure(figsize=(6,6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

```
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 126ms/step
```



True Label / Predicted label	Cloudy	Rain	Shine	Sunrise
Cloudy	0	0	0	91
Rain	0	0	0	72
Shine	0	0	0	75

True Label / Predicted label	Cloudy	Rain	Shine	Sunrise
Sunrise	0	0	0	100

Main diagonal (0, 0, 0, 100) → True Positives

- 0 Cloudy photos were correctly predicted.
- 0 Rain photos were correctly.
- 0 Shine photos were correctly predicted.
- 100 Sunrise photos were correctly predicted (perfect accuracy for Sunrise class).
- This means that only the "Sunrise" class is recognized correctly, all other classes are confused.

Values out off the main diagonal → False Positives

- 91 Cloudy photos were mistaken for Sunrise.
- 72 photos of Rain were mistaken for Sunrise.
- 75 photos of Shine were mistaken for Sunrise.
- This shows that the model is tending to predict all images as "Sunrise".

Evaluate the model

- Poor classification performance overall.
- Cloudy, Rain, Shine have 0% accuracy.
- Extreme class bias toward Sunrise.

Load again the cnn but this time set the parameters to NOT TRAINABLE

In [17]:
```
# Set trainable (True, False)
isTrainable = False

baseModel = VGG19(input_shape=target_size, weights='imagenet', include_top=False) # Input model use VGG19 use imagenet weight

# Freeze all layers of VGG19
for layer_ctn, layer in enumerate(baseModel.layers[:]):
    layer.trainable = isTrainable
```

```
# Flatten layer to convert from 4D tensor -> 1D vector
x = Flatten()(baseModel.output)

# Fully Connected Layers (Dense + Dropout)
x = Dense(512, activation='relu', kernel_regularizer=l1_l2(l1=0, l2=0.01))(x) # Add a Dense layer with 512 units
x = Dropout(0.1)(x) # Dropout layer with 0.1 unit
x = Dense(256, activation='relu', kernel_regularizer=l1_l2(l1=0, l2=0.01))(x) # Add a Dense layer with 256 units
x = Dropout(0.1)(x) # Dropout layer with 0.1 unit

# Output layer with 4 classes for the final classifier
x = Dense(4, activation='softmax')(x)  # 4 classes: ['Cloudy', 'Rain', 'Shine', 'Sunrise']

# Combine base_model and Fully Connected layers into a final model
model = Model(inputs=baseModel.input, outputs=x)

model.summary() # Print mode summary
```

Model: "functional_1"

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv4 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv4 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808

block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv4 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten_1 (Flatten)	(None, 25088)	0
dense_3 (Dense)	(None, 512)	12,845,568
dropout_2 (Dropout)	(None, 512)	0
dense_4 (Dense)	(None, 256)	131,328
dropout_3 (Dropout)	(None, 256)	0
dense_5 (Dense)	(None, 4)	1,028

Total params: 33,002,308 (125.89 MB)

Trainable params: 12,977,924 (49.51 MB)

Non-trainable params: 20,024,384 (76.39 MB)

Fit the model with batch size 32 and 15 epochs (This is fsaster)

In [18]:
```
# Compile model with Adam
model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='categorical_crossentropy', # Use categorical_crossentrop for multi-class classification
    metrics=['accuracy']
)

# Train model with epochs = 15, batch_size = 32
history = model.fit(X_train, y_train,
                    epochs=15, batch_size=32,
                    validation_data=(X_val, y_val))
```

```
Epoch 1/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 13s 442ms/step - accuracy: 0.3825 - loss: 12.1266 - val_accuracy: 0.7219 - val_loss: 5.0827
Epoch 2/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 132ms/step - accuracy: 0.8157 - loss: 4.2797 - val_accuracy: 0.7249 - val_loss: 3.2858
Epoch 3/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 127ms/step - accuracy: 0.8272 - loss: 2.8125 - val_accuracy: 0.8905 - val_loss: 2.1537
Epoch 4/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 124ms/step - accuracy: 0.9541 - loss: 1.9234 - val_accuracy: 0.9083 - val_loss: 1.7115
Epoch 5/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 125ms/step - accuracy: 0.9436 - loss: 1.5715 - val_accuracy: 0.8876 - val_loss: 1.5914
Epoch 6/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 124ms/step - accuracy: 0.8951 - loss: 1.4987 - val_accuracy: 0.9053 - val_loss: 1.4403
Epoch 7/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 127ms/step - accuracy: 0.9292 - loss: 1.3445 - val_accuracy: 0.8846 - val_loss: 1.3637
Epoch 8/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 128ms/step - accuracy: 0.9547 - loss: 1.1552 - val_accuracy: 0.8905 - val_loss: 1.2334
Epoch 9/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 129ms/step - accuracy: 0.9816 - loss: 0.9886 - val_accuracy: 0.9053 - val_loss: 1.0947
Epoch 10/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 126ms/step - accuracy: 0.9612 - loss: 0.9302 - val_accuracy: 0.8254 - val_loss: 1.2628
Epoch 11/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 131ms/step - accuracy: 0.9159 - loss: 1.0628 - val_accuracy: 0.8462 - val_loss: 1.2218
Epoch 12/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 129ms/step - accuracy: 0.9358 - loss: 1.0028 - val_accuracy: 0.8521 - val_loss: 1.2416
Epoch 13/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 128ms/step - accuracy: 0.9331 - loss: 0.9527 - val_accuracy: 0.8787 - val_loss: 1.0426
Epoch 14/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 126ms/step - accuracy: 0.9699 - loss: 0.8234 - val_accuracy: 0.8698 - val_loss: 0.9953
Epoch 15/15
25/25 ━━━━━━━━━━━━━━━━━━━━ 3s 128ms/step - accuracy: 0.9272 - loss: 0.8641 - val_accuracy: 0.7781 - val_loss: 1.3017
```

Evaluate the model

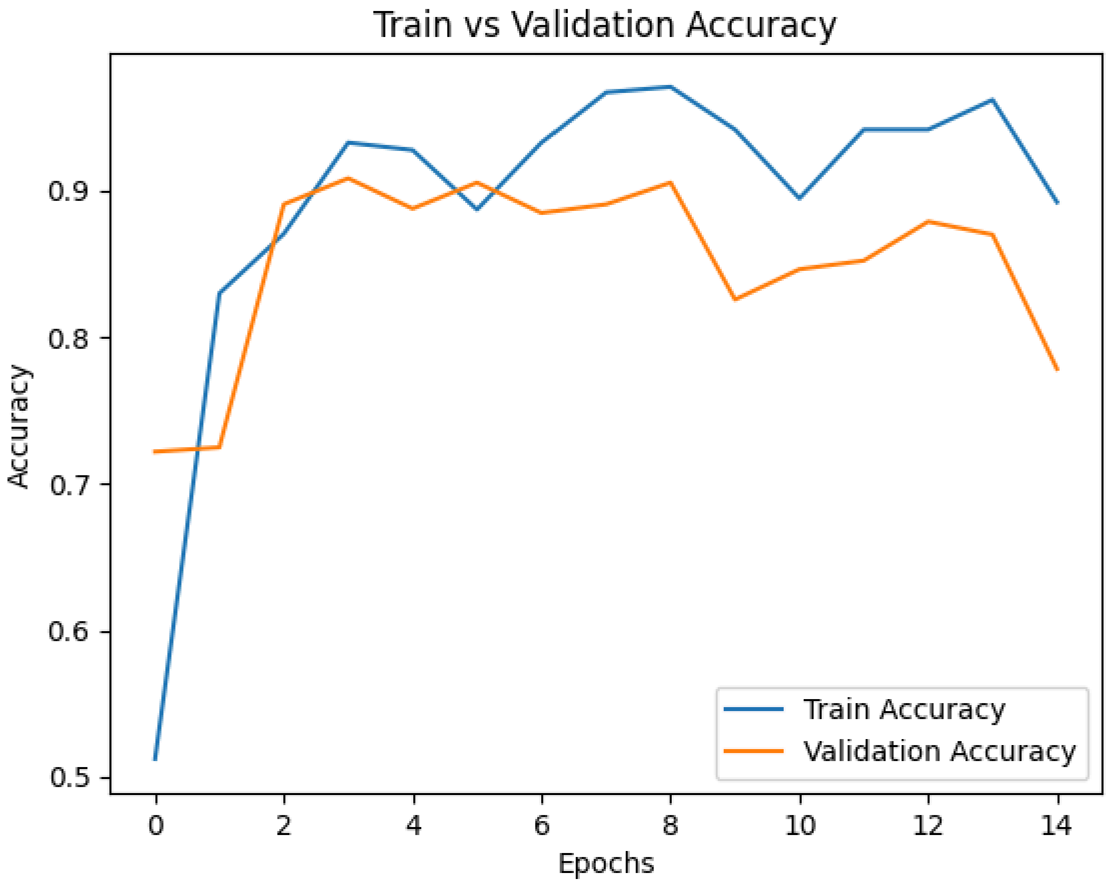
In [19]:

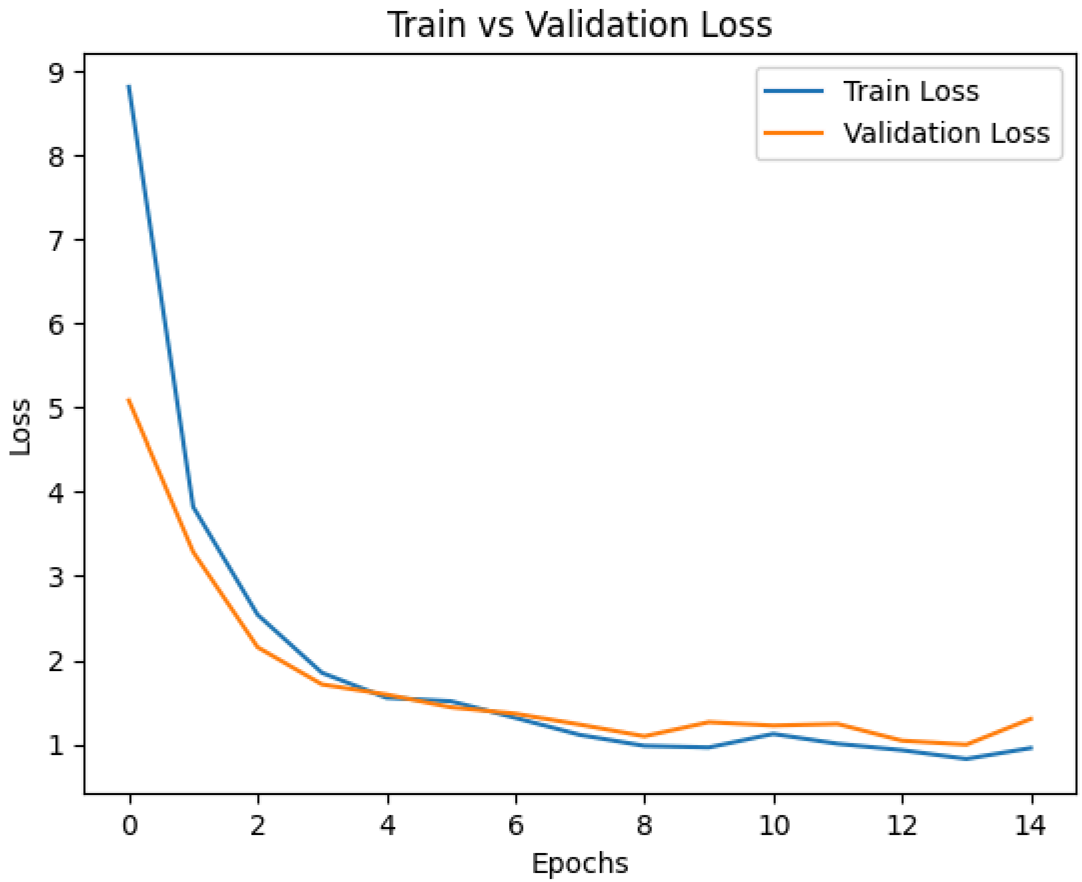
```python
test_loss, test_acc = model.evaluate(X_val, y_val)
print(f"Accuracy: {test_acc*100:.2f}%")
```

```
11/11 ━━━━━━━━━━━━━━━━━━━━ 1s 84ms/step - accuracy: 0.7701 - loss: 1.2972
Accuracy: 77.81%
```

Draw a chart of the training process to check overfitting/underfitting

In [20]:
```
# Plot accuracy graph
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Train vs Validation Accuracy')
plt.show()

# Plot loss graph
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Train vs Validation Loss')
plt.show()
```


Train vs Validation Accuracy
Accuracy
0.5
0.6
0.7
0.8
0.9
0
2
4
6
8
10
12
14
Epochs
Train Accuracy
Validation Accuracy



Train vs Validation Accuracy

- Trend: Accuracy increases gradually with the number of epochs
- High train accuracy (~90%): The model is learning well on the training set
- High validation accuracy (~80%): The model generates well
- Train-validation distance: The gap is not too far (10%); it illustrates no signs of serious overfitting

Train vs Validation Loss

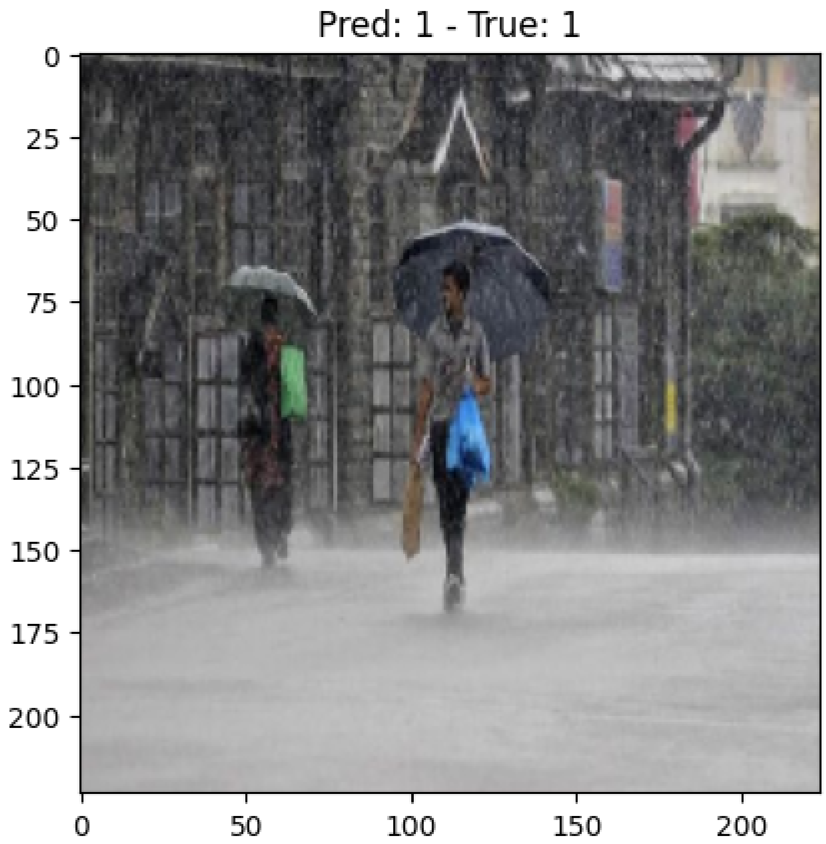
- Trend: Loss of both training set and validation set are gradually decreasing.
- Loss distance: The gap is quite close to each other, which shows the model is not overfitting.

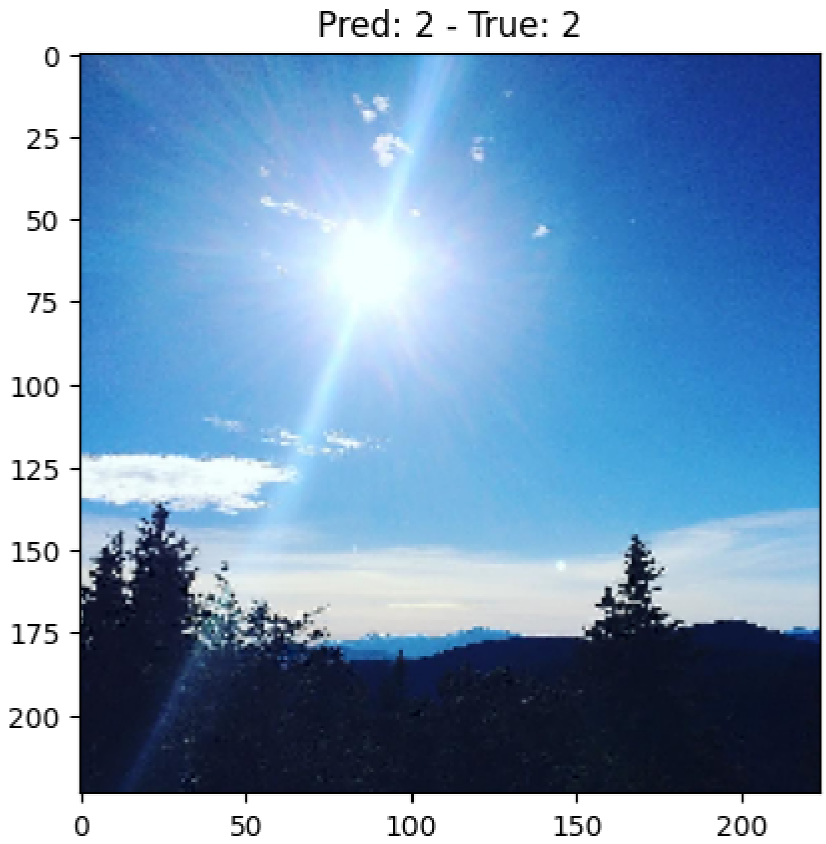
- Loss reduction: There are no large fluctuations; providing that training is stable

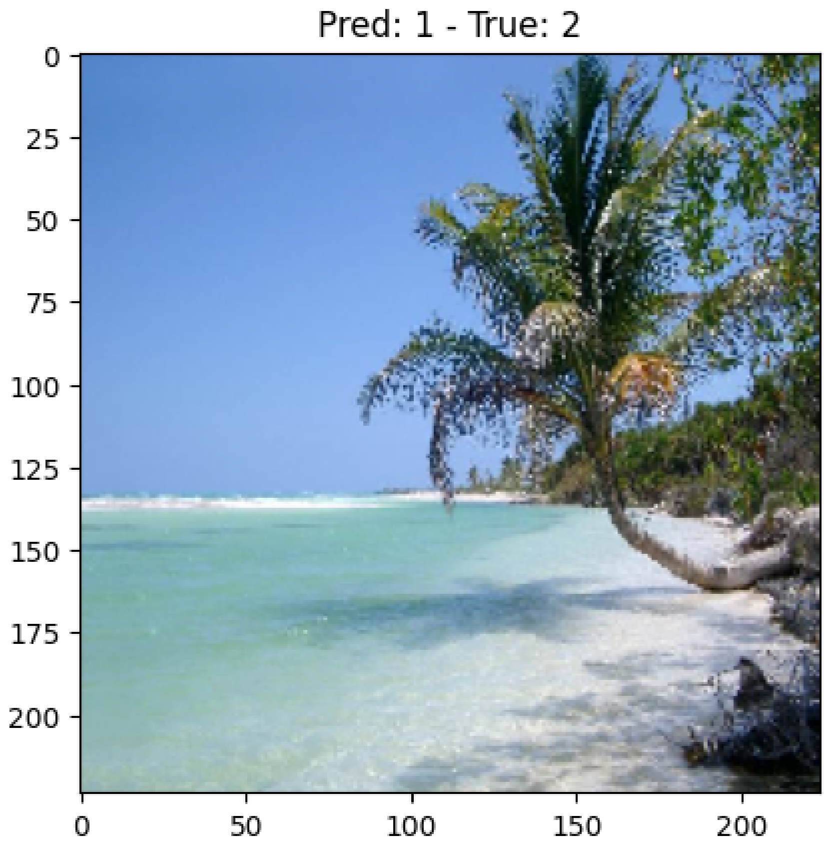
Make and show some predictions

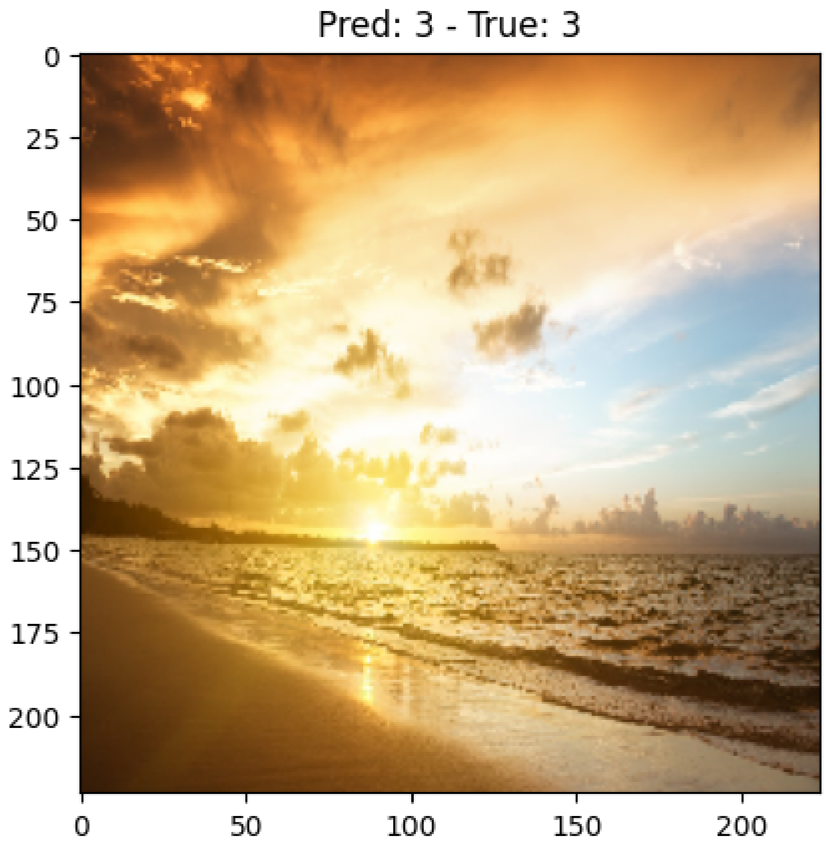
In [21]:
```
# Chose random 5 samples from validation
num_samples = 5
indices = np.random.choice(len(X_val), num_samples, replace=False)
sample_images = X_val[indices]
sample_labels = y_val[indices]

# Prediction
predictions = model.predict(sample_images)

# Plot the result, 4 classes: ['Cloudy', 'Rain', 'Shine', 'Sunrise']
for i in range(num_samples):
    plt.imshow(sample_images[i])
    plt.title(f'Pred: {np.argmax(predictions[i])} - True: {np.argmax(sample_labels[i])}')
    plt.show()
```

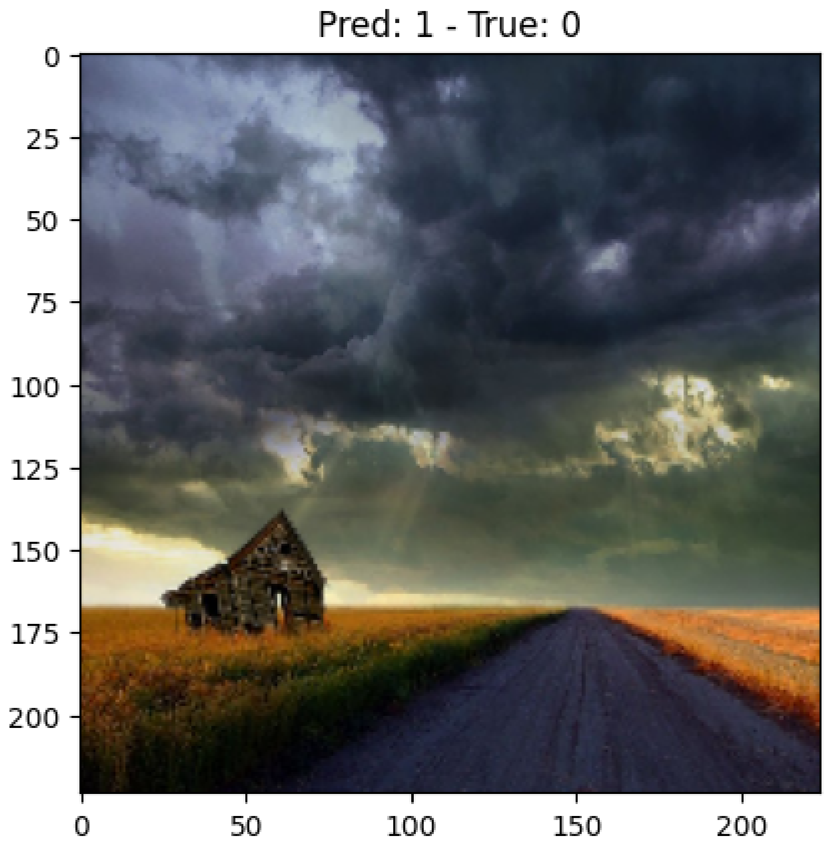
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 567ms/step
```


Pred: 1 - True: 1
0
25
50
75
100
125
150
175
200
0
50
100
150
200


Pred: 2 - True: 2
0
25
50
75
100
125
150
175
200
0
50
100
150
200


Pred: 1 - True: 2
0
25
50
75
100
125
150
175
200
0
50
100
150
200


Pred: 3 - True: 3
0
25
50
75
100
125
150
175
200
0
50
100
150
200

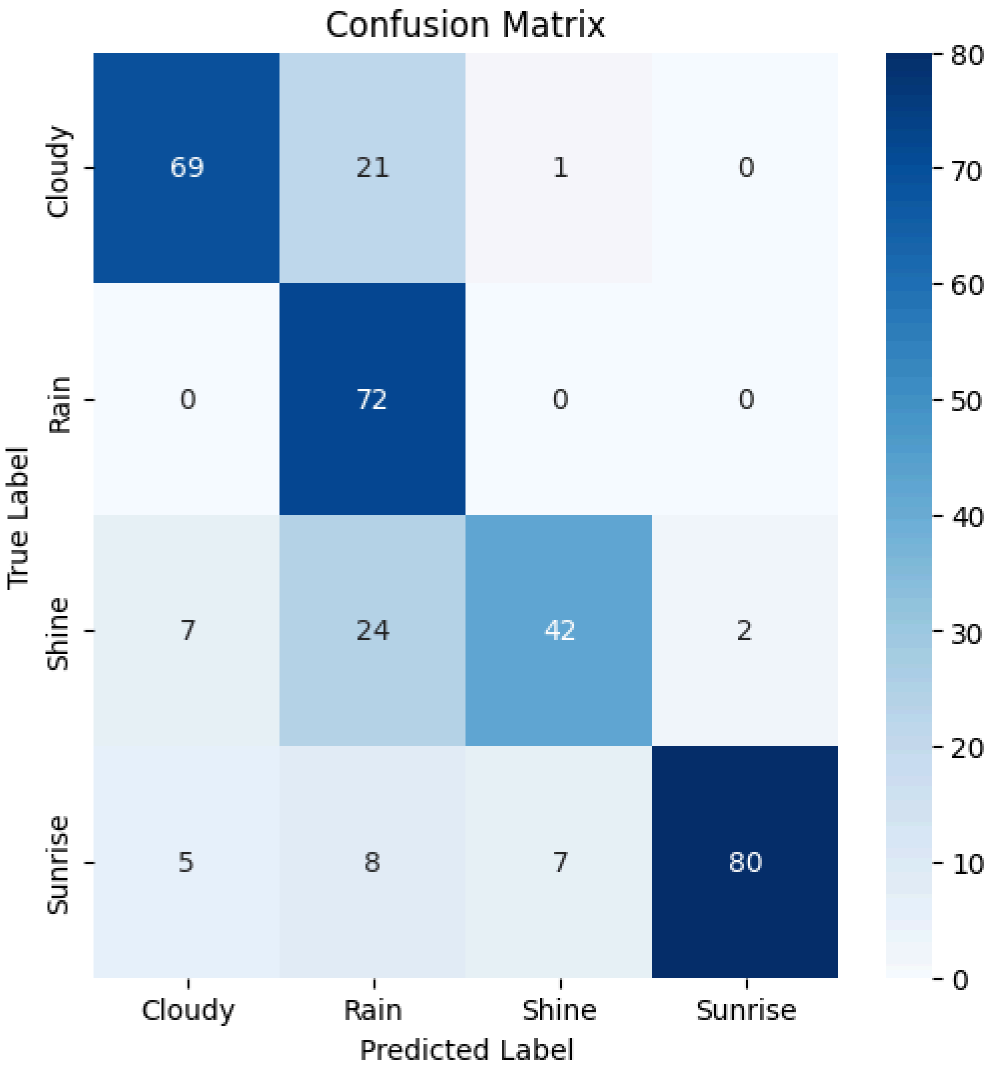


make confusion matrix

In [22]:
```
# Predict the probabilities of classes on the validation set
y_pred_probs = model.predict(X_val)

# Convert to a label by taking the value with the highest probability
y_pred = np.argmax(y_pred_probs, axis=1)

# Convert it back to label format when y_val is in one-hot encoding
y_true = np.argmax(y_val, axis=1)

# Definition of class labels
class_labels = ['Cloudy', 'Rain', 'Shine', 'Sunrise']
```

```python
# Create confusion matrix
cm = confusion_matrix(y_true, y_pred)

# Plot confusion matrix
plt.figure(figsize=(6,6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

```
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 120ms/step
```



True Label / Predicted label	Cloudy	Rain	Shine	Sunrise
Cloudy	69	21	1	0
Rain	0	72	0	0
Shine	7	24	42	2

True Label / Predicted label	Cloudy	Rain	Shine	Sunrise
Sunrise	5	8	7	80

Main diagonal (60, 72, 42, 80) → True Positives

- 69 Cloudy photos were correctly predicted.
- 72 Rain photos were correctly.
- 42 Shine photos were correctly predicted.
- 80 Sunrise photos were correctly predicted.

Values out off the main diagonal → False Positives

- 21 Cloudy photos were mistaken for Rain.
- 7 photos of Shine were mistaken for Cloudy.
- 24 photos of Shine were mistaken for Rain.
- 2 photos of Shine were mistaken for Sunrise.
- 5 photos of Sunrise were mistaken for Cloudy.
- 8 photos of Sunrise were mistaken for Rain.
- 7 photos of Sunrise were mistaken for Shine.

Evaluate the model

- Shine and Cloudy had the most confusion with other classes.
- Sunrise had the highest prediction accuracy (80%).

Compare two models

Method	Train Accuracy	Validation Accuracy	Test Accuracy	Evaluation
(trainable = True)	32%	30%	29.59%	Poor classification performance overall.
(trainable = False)	90%	80%	77.81%	Good performance in overall, a small sign of overfitting but it is not significant

Trainable model = True (32% train - 30% validation)

- VGG19 has been trained on ImageNet with millions of images, helping the model have very strong image recognition features.
- When trainable = True, there are updating all the weights of the model from the beginning.
- Insufficient data diversity makes it easy for the model to remember the patterns of the training set, but fails to generalize when predicting on new data.
- This leads to random guessing of a single class ("Sunrise" in the Confusion Matrix).

Trainable = False (90% train - 80% validation)

- Learn well because It takes advantage of ImageNet features.

Conclusion

Train Method	When to be used?
Trainable = True	When there is a large data set, it helps the model learn more optimally. Needs a more powerful GPU to train.
Trainable = False	When the dataset is small, avoid overfitting but may suffer from underfitting. Train faster.