

```
In [ ]: import os
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import splitfolders
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, AveragePooling2D
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.layers import LeakyReLU
```

```
In [ ]: # Function to extract features from images
def extract_features(image_path):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    resized_image = cv2.resize(image, (32, 32)) # Resize the image to a fixed size
    flattened_image = resized_image.flatten() # Flatten the image into a 1D array
    return flattened_image
```

```
In [ ]: # Split dataset into training and testing sets
input_dataset = "datasets/solarpanels/Faulty_solar_panel"
output_dataset = "datasets/solarpanels/Faulty_solar_panel"
splitfolders.ratio(input_dataset, output=output_dataset, seed=42, ratio=(.8, .2), group_prefix="train")

# Load training and testing datasets
train_dataset = os.path.join(output_dataset, "train")
test_dataset = os.path.join(output_dataset, "val")
```

Copying files: 555 files [00:00, 573.22 files/s]

```
In [ ]: # Load training data
X_train = []
y_train = []
for root, dirs, files in os.walk(train_dataset):
    for filename in files:
        image_path = os.path.join(root, filename)
        label = os.path.basename(root)
        X_train.append(extract_features(image_path))
        y_train.append(label)
```

```
In [ ]: # Load testing data
X_test = []
y_test = []
for root, dirs, files in os.walk(test_dataset):
    for filename in files:
        image_path = os.path.join(root, filename)
        label = os.path.basename(root)
        X_test.append(extract_features(image_path))
        y_test.append(label)
```

```
In [ ]: # Convert lists to numpy arrays
X_train = np.array(X_train)
y_train = np.array(y_train)
X_test = np.array(X_test)
y_test = np.array(y_test)
```

```
In [ ]: # Define hyperparameters to tune
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
    'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
}

# Initialize and train Logistic Regression model
model = LogisticRegression(max_iter=10000)
```

```
In [ ]: # Initialize GridSearchCV
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='a
```

```
In [ ]: # Perform grid search to find the best hyperparameters
grid_search.fit(X_train, y_train)
```

```
Out[ ]: ▶      GridSearchCV      ⓘ ⓘ
          ▶ estimator: LogisticRegression
              ▶ LogisticRegression ⓘ
```

```
In [ ]: # Get the best hyperparameters and the corresponding accuracy
best_params = grid_search.best_params_
best_accuracy = grid_search.best_score_

print("Best Hyperparameters:", best_params)
print("Best Accuracy:", best_accuracy)

# Use the best model for prediction
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

# Evaluate model performance
accuracy = accuracy_score(y_test, y_pred)
print("Test Accuracy:", accuracy)
```

```
Best Hyperparameters: {'C': 0.001, 'solver': 'lbfgs'}
Best Accuracy: 0.5642236976506639
Test Accuracy: 0.5982142857142857
```

Not great, lets try other models, starting with Random Forest

```
In [ ]: # Define hyperparameters to tune
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
}

# Initialize Random Forest model
model = RandomForestClassifier(random_state=42)
```

```
In [ ]: # Initialize GridSearchCV
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='accuracy')

# Perform grid search to find the best hyperparameters
grid_search.fit(X_train, y_train)
```

```
Out[ ]: ▶      GridSearchCV      ⓘ ⓘ
        ▶ estimator: RandomForestClassifier
            ▶ RandomForestClassifier ⓘ
```

```
In [ ]: # Get the best hyperparameters and the corresponding accuracy
best_params = grid_search.best_params_
best_accuracy = grid_search.best_score_

print("Best Hyperparameters:", best_params)
print("Best Accuracy:", best_accuracy)

# Use the best model for prediction
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

# Evaluate model performance
accuracy = accuracy_score(y_test, y_pred)
print("Test Accuracy:", accuracy)
```

```
Best Hyperparameters: {'max_depth': 10, 'min_samples_split': 2, 'n_estimators': 50}
Best Accuracy: 0.6747957099080695
Test Accuracy: 0.7410714285714286
```

Lets try Support Vector Machines

```
In [ ]: # Define hyperparameters to tune
param_grid = {
    'C': [0.1, 1, 10],
    'gamma': [0.01, 0.1, 1],
    'kernel': ['linear', 'rbf'],
}

# Initialize SVM model
model = SVC()
```

```
In [ ]: # Initialize GridSearchCV
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='a'

# Perform grid search to find the best hyperparameters
grid_search.fit(X_train, y_train)
```

Out[]:

- ▶ GridSearchCV ⓘ ⓘ ?
- ▶ estimator: SVC
 - ▶ SVC ⓘ ?

```
In [ ]: # Get the best hyperparameters and the corresponding accuracy
best_params = grid_search.best_params_
best_accuracy = grid_search.best_score_

print("Best Hyperparameters:", best_params)
print("Best Accuracy:", best_accuracy)

# Use the best model for prediction
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

# Evaluate model performance
accuracy = accuracy_score(y_test, y_pred)
print("Test Accuracy:", accuracy)
```

```
Best Hyperparameters: {'C': 0.1, 'gamma': 0.01, 'kernel': 'linear'}
Best Accuracy: 0.5414964249233912
Test Accuracy: 0.5625
```

Lets try Gradient Boosting

```
In [ ]: # Define hyperparameters to tune
param_grid = {
    'n_estimators': [50, 100, 150],
    'learning_rate': [0.05, 0.1, 0.2],
    'max_depth': [3, 4, 5],
}

# Initialize GBM model
model = GradientBoostingClassifier()
```

```
In [ ]: # Initialize GridSearchCV
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='a'

# Perform grid search to find the best hyperparameters
grid_search.fit(X_train, y_train)
```

```
Out[ ]: > GridSearchCV ⓘ ⓘ
      > estimator: GradientBoostingClassifier
          > GradientBoostingClassifier ⓘ
```

```
In [ ]: # Get the best hyperparameters and the corresponding accuracy
best_params = grid_search.best_params_
best_accuracy = grid_search.best_score_

print("Best Hyperparameters:", best_params)
print("Best Accuracy:", best_accuracy)

# Use the best model for prediction
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

# Evaluate model performance
accuracy = accuracy_score(y_test, y_pred)
print("Test Accuracy:", accuracy)
```

```
Best Hyperparameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 100}
Best Accuracy: 0.649923391215526
Test Accuracy: 0.6696428571428571
```

Now lets try a CNN and see if that does better

```
In [ ]: # Function to extract features from images
def extract_features(image_path):
    image = cv2.imread(image_path)
    resized_image = cv2.resize(image, (32, 32)) # Resize the image to a fixed size
    return resized_image
```

```
In [ ]: # Load training data
X_train = []
y_train = []
for root, dirs, files in os.walk(train_dataset):
    for filename in files:
        image_path = os.path.join(root, filename)
        label = os.path.basename(root) # Assuming class labels are encoded as fold
        X_train.append(extract_features(image_path))
        y_train.append(label)
```

```
In [ ]: # Load testing data
X_test = []
y_test = []
for root, dirs, files in os.walk(test_dataset):
    for filename in files:
        image_path = os.path.join(root, filename)
        label = os.path.basename(root) # Assuming class labels are encoded as fold
        X_test.append(extract_features(image_path))
        y_test.append(label)
```

```
In [ ]: # Convert lists to numpy arrays
X_train = np.array(X_train)
X_test = np.array(X_test)
```

```
In [ ]: # Normalize pixel values to the range [0, 1]
X_train = X_train / 255.0
X_test = X_test / 255.0
```

```
In [ ]: # Initialize LabelEncoder
label_encoder = LabelEncoder()

# Fit and transform the training labels
y_train_encoded = label_encoder.fit_transform(y_train)

# Transform the testing labels
y_test_encoded = label_encoder.transform(y_test)

# One-hot encode the numerical labels
num_classes = len(np.unique(y_train_encoded))
y_train = to_categorical(y_train_encoded, num_classes)
y_test = to_categorical(y_test_encoded, num_classes)
```

```
In [ ]: # Define the CNN architecture
model = Sequential([
    Conv2D(32, (3, 3), activation='tanh', padding='same', input_shape=(32, 32, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='tanh', padding='same'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='tanh', padding='same'),
    MaxPooling2D((2, 2)),
    Conv2D(256, (3, 3), activation='tanh', padding='same'),
    MaxPooling2D((2, 2)),
    Conv2D(512, (3, 3), padding='same'),
    LeakyReLU(alpha=0.1),
```

```
    MaxPooling2D((2, 2)),  
    Flatten(),  
    Dense(64, activation='relu'),  
    Dense(num_classes, activation='softmax')  
])
```

```
In [ ]: # Compile the model  
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])  
  
# Train the model  
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_te
```

```
Epoch 1/10  
14/14 [=====] - 2s 98ms/step - loss: 1.2288 - accuracy: 0.4  
470 - val_loss: 0.9967 - val_accuracy: 0.5714  
Epoch 2/10  
14/14 [=====] - 1s 73ms/step - loss: 1.0459 - accuracy: 0.5  
688 - val_loss: 0.8944 - val_accuracy: 0.6250  
Epoch 3/10  
14/14 [=====] - 1s 72ms/step - loss: 0.9628 - accuracy: 0.5  
914 - val_loss: 0.8631 - val_accuracy: 0.6607  
Epoch 4/10  
14/14 [=====] - 1s 71ms/step - loss: 0.8425 - accuracy: 0.6  
975 - val_loss: 0.8286 - val_accuracy: 0.7232  
Epoch 5/10  
14/14 [=====] - 1s 67ms/step - loss: 0.7164 - accuracy: 0.7  
178 - val_loss: 0.8044 - val_accuracy: 0.6786  
Epoch 6/10  
14/14 [=====] - 1s 67ms/step - loss: 0.6417 - accuracy: 0.7  
540 - val_loss: 0.7402 - val_accuracy: 0.7411  
Epoch 7/10  
14/14 [=====] - 1s 71ms/step - loss: 0.5117 - accuracy: 0.8  
397 - val_loss: 0.7173 - val_accuracy: 0.7321  
Epoch 8/10  
14/14 [=====] - 1s 71ms/step - loss: 0.4072 - accuracy: 0.8  
623 - val_loss: 0.6708 - val_accuracy: 0.7768  
Epoch 9/10  
14/14 [=====] - 1s 69ms/step - loss: 0.3176 - accuracy: 0.8  
826 - val_loss: 0.7782 - val_accuracy: 0.7321  
Epoch 10/10  
14/14 [=====] - 1s 74ms/step - loss: 0.2595 - accuracy: 0.9  
074 - val_loss: 0.7041 - val_accuracy: 0.7679
```

```
Out[ ]: <keras.callbacks.History at 0x25dceece5c0>
```

```
In [ ]: # Evaluate model performance  
test_loss, test_accuracy = model.evaluate(X_test, y_test)  
print("Test Loss:", test_loss)  
print("Test Accuracy:", test_accuracy)
```

```
4/4 [=====] - 0s 13ms/step - loss: 0.7041 - accuracy: 0.767  
9  
Test Loss: 0.7041270136833191  
Test Accuracy: 0.7678571343421936
```