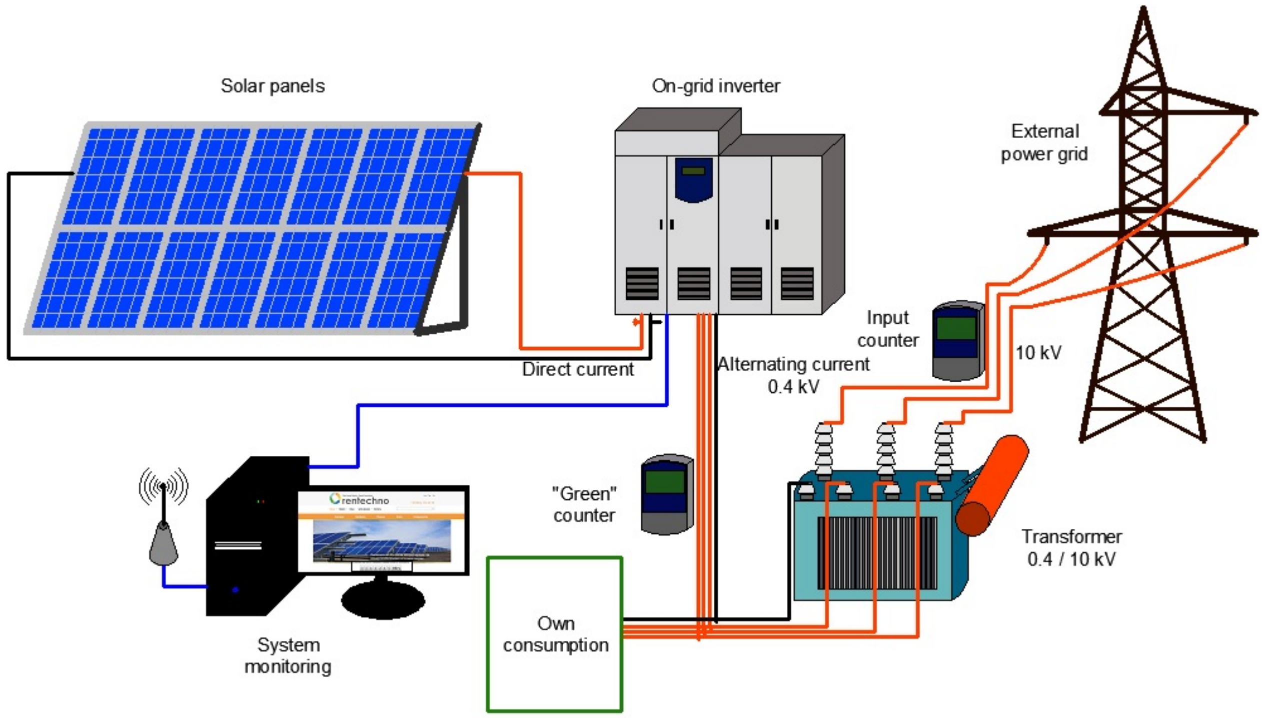
Dust Detection on solar panels by using InceptionV3



Objective of this project

1. **Solar panels work by converting sunlight into electricity. If dirt, dust, or other debris accumulates on the surface of the solar panels, it can reduce the amount of sunlight that is absorbed, which can lead to a decrease in the amount of electricity that is generated.**

2. **The amount of energy loss depends on the level of dirt and debris on the solar panels. According to the Solar Energy Power Association, dirty solar panels can lose up to 20% of their energy output. The National Renewable Energy Laboratory puts that figure even higher, at 25%.**

3. **In addition to reducing the amount of electricity that is generated, dirty solar panels can also shorten the lifespan of the solar panels. This is because the dirt and debris can trap moisture, which can cause corrosion and other damage to the solar panels.**

4. **For these reasons, it is important to clean solar panels regularly. The frequency of cleaning will depend on a number of factors, including the environment in which the solar panels are located. In general, however, most manufacturers recommend that solar panels be cleaned at least twice a year.**

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import PIL
from PIL import Image
```

In [2]:
```
from sklearn.utils.class_weight import compute_class_weight
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, GlobalAveragePooling2D
from keras.preprocessing.image import ImageDataGenerator
from keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping
from keras import regularizers
from keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.applications import VGG16, ResNet50, InceptionV3, MobileNetV2, DenseNet121
from itertools import chain
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, GlobalAveragePooling2D
from keras.preprocessing.image import ImageDataGenerator
from keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping
from keras import regularizers
from keras.callbacks import ReduceLROnPlateau
```

In [3]:
```
import warnings
warnings.filterwarnings("ignore", category=UserWarning, module="tensorflow_io")
```

In [4]:
```
BATCH_SIZE = 48

image_height = 299
image_width = 299
```

In [5]:
```
# Data agumentation and pre-processing using tensorflow
data_generator_1 = ImageDataGenerator(
                            rescale=1./255,
                            rotation_range=5,
                            width_shift_range=0.05,
                            height_shift_range=0.05,
                            shear_range=0.05,
                            zoom_range=0.05,
                            brightness_range = [0.95,1.05],
                            horizontal_flip=False,
                            vertical_flip=False,
                            fill_mode='nearest'
                        )

print('Data Augmentation 1 was created')

data_generator_2 = ImageDataGenerator(
                            rescale=1./255,
                            rotation_range=10,
                            width_shift_range=0.1,
                            height_shift_range=0.1,
                            shear_range=0.1,
                            zoom_range=0.1,
                            brightness_range = [0.9,1.1],
                            horizontal_flip=False,
                            vertical_flip=False,
                            fill_mode='nearest'
                        )
print('Data Augmentation 2 was created')

data_generator_3 = ImageDataGenerator (rescale=1./255)
```

```
Data Augmentation 1 was created
Data Augmentation 2 was created
```

In [6]:

```
# Read the image
train_generator1 = data_generator_1.flow_from_directory(
    directory = "/kaggle/input/solar-panels-dirt-detection/Detect_solar_dust/", # images data path / folder in 
#     subset = 'training',
    color_mode = "rgb",
    target_size = (image_height, image_width), # image height , image width
    class_mode = "categorical",
    batch_size = BATCH_SIZE,
    shuffle = True,
    seed = 1234)

test_generator = data_generator_2.flow_from_directory(
    directory = "/kaggle/input/solar-panels-dirt-detection/Detect_solar_dust/", # images data path / folder in 
#     subset = 'validation',
    color_mode = "rgb",
    target_size = (image_height, image_width), # image height , image width
    class_mode = "categorical",
    batch_size = BATCH_SIZE,
    shuffle = True,
    seed = 1234)
```

```
Found 1187 images belonging to 2 classes.
Found 1187 images belonging to 2 classes.
```

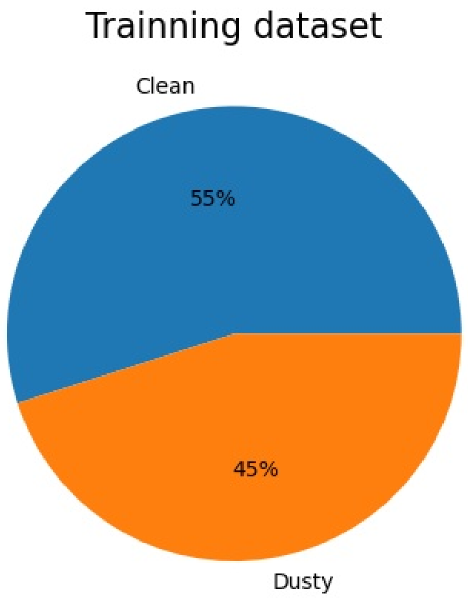
In [7]:

```
dict_class = train_generator1.class_indices
print('Dictionary: {}'.format(dict_class))
class_names = list(dict_class.keys())  # storing class/breed names in a list
print('Class labels: {}'.format(class_names))
```

```
Dictionary: {'Clean': 0, 'Dusty': 1}
Class labels: ['Clean', 'Dusty']
```

In [8]:

```
frequency = np.unique(train_generator1.classes, return_counts=True)

plt.title("Trainning dataset", fontsize='16')
plt.pie(frequency[1], labels = class_names, autopct='%1.0f%%');
```



In [9]:

```
# Dataset characteristics
print("Dataset Characteristics of Train Data Set:")
print("Number of images:", len(train_generator1.classes))
print("Number of normal images:", len([label for label in train_generator1.classes if label == 0]))
print("Number of pneumonia images:", len([label for label in train_generator1.classes if label == 1]))
print()

print("Dataset Characteristics of Test Data Set:")
print("Number of images:", len(test_generator.classes))
print("Number of normal images:", len([label for label in test_generator.classes if label == 0]))
print("Number of pneumonia images:", len([label for label in test_generator.classes if label == 1]))
print()
```

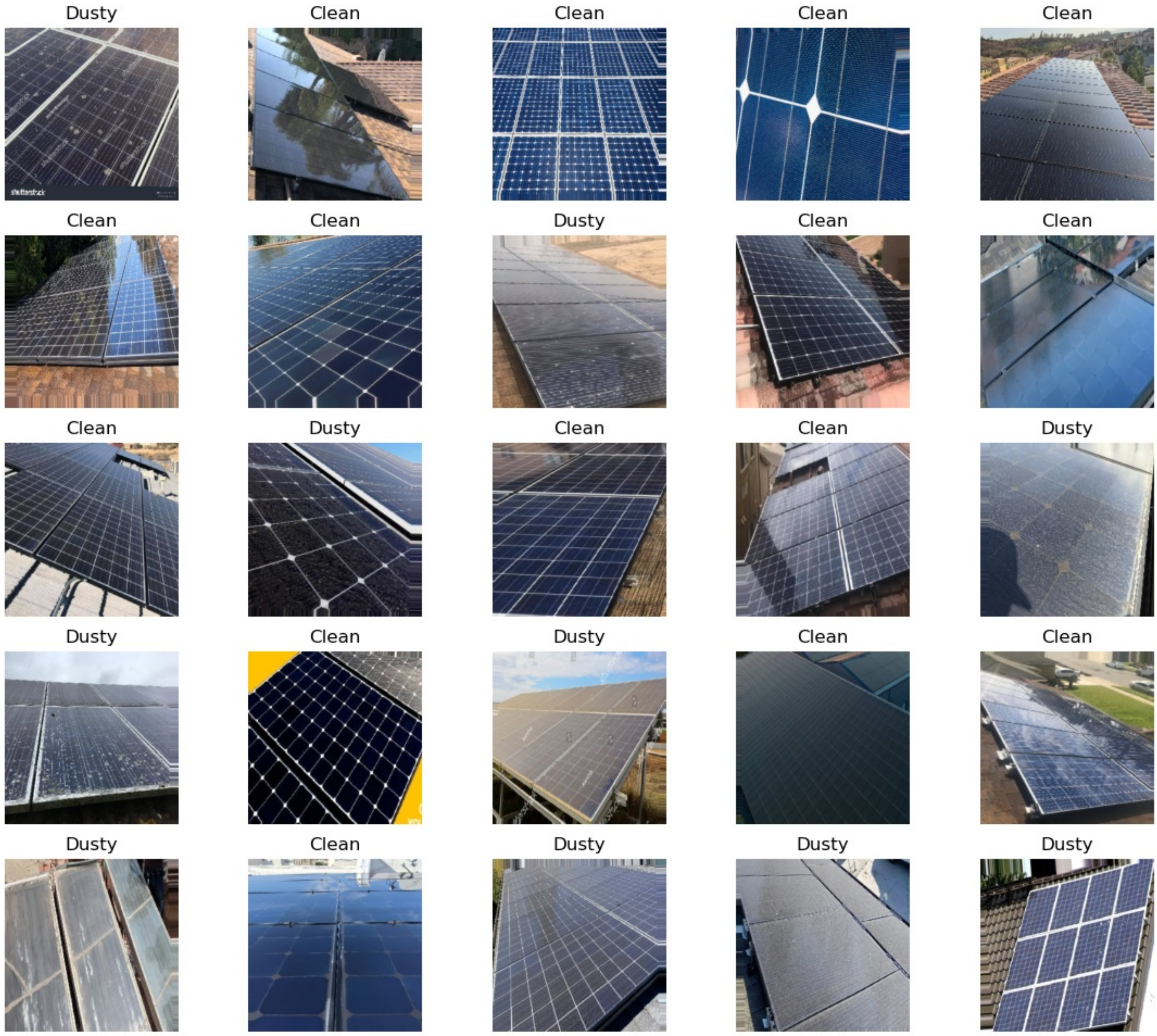
```
Dataset Characteristics of Train Data Set:
Number of images: 1187
Number of normal images: 652
Number of pneumonia images: 535

Dataset Characteristics of Test Data Set:
Number of images: 1187
Number of normal images: 652
Number of pneumonia images: 535
```

In [10]:
```python
class_weights = compute_class_weight(class_weight = "balanced", classes= np.unique(train_generator1.classes), y
class_weights = dict(zip(np.unique(train_generator1.classes), class_weights))
class_weights
```

Out[10]:
```
{0: 0.9102760736196319, 1: 1.1093457943925233}
```

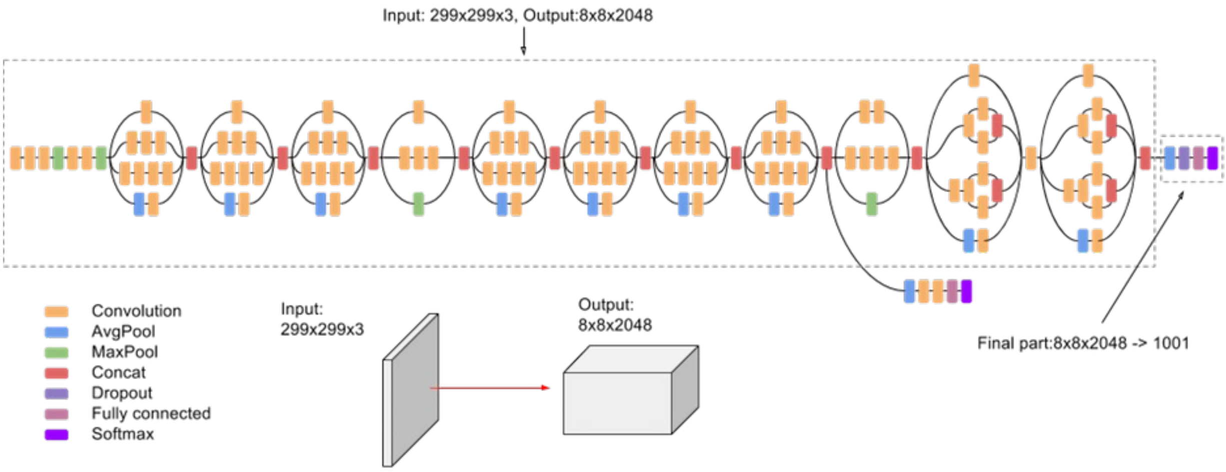
In [11]:
```python
# Image Samples
print('Train image data from Data Augmentation 1')
img, label = next(train_generator1)
# print(len(label))

plt.figure(figsize=[14, 12])
for i in range(25):
    plt.subplot(5, 5, i+1)
    plt.imshow(img[i])
    plt.axis('off')
    plt.title(class_names[np.argmax(label[i])])
plt.show()
```

```
Train image data from Data Augmentation 1
```



In [12]:
```python
EPOCHS = 50
num_gpus = 2
early_stopping = EarlyStopping(monitor='val_accuracy', patience=3, verbose=1, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_accuracy', factor=0.001, patience=10, verbose=1)
train_data = train_generator1
```

InceptionV3 Model



Inception V3 is a convolutional neural network (CNN) architecture developed by Google in 2015. It is a deep learning model that is used for image classification and object detection. Inception V3 is a successor to the Inception V1 and Inception V2 models, and it has several advantages over other deep learning models.

Advantages of Inception V3

1. Higher accuracy: Inception V3 has achieved state-of-the-art accuracy on the ImageNet dataset, which is a benchmark dataset for image classification.
2. Efficiency: Inception V3 is a computationally efficient model, which means that it can be trained and used on a variety of hardware platforms.
3. Flexibility: Inception V3 is a versatile model that can be used for a variety of image classification tasks.

Inception V3 Architecture

The Inception V3 architecture is composed of a series of Inception modules. Each Inception module is a combination of different convolutional layers, pooling layers, and normalization layers. This combination of layers allows Inception V3 to extract features from images at different scales.

Applications of Inception V3

1. Inception V3 has been used for a variety of image classification tasks, including:

2. ImageNet classification: Inception V3 achieved a top-5 error rate of 23.1% on the ImageNet dataset.

3. Object detection: Inception V3 has been used to train object detection models, such as the Faster R-CNN model.
4. Semantic segmentation: Inception V3 has been used to train semantic segmentation models, which can be used to identify objects and their boundaries in images.

In [13]:
```
strategy = tf.distribute.MirroredStrategy(devices=['/gpu:0', '/gpu:1'])

with strategy.scope():

    # Load the pre-trained InceptionV3 model without the top classification layer
    base_model_Inception = InceptionV3(weights='imagenet', include_top=False, input_shape=(image_height, image_

    # Set the layers of the base model as non-trainable (freeze them)
    for layer in base_model_Inception.layers:
        layer.trainable = False

    # Create a new model and add the InceptionV3 base model
    model_Inception = Sequential()
    model_Inception.add(base_model_Inception)

    # Add a global average pooling layer and output layer for classification
    model_Inception.add(GlobalAveragePooling2D())
```

```python
    model_Inception.add(Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.001)))
    model_Inception.add(Dropout(0.4))
    model_Inception.add(Dense(64, activation='relu', kernel_regularizer=regularizers.l2(0.001)))
    model_Inception.add(Dropout(0.2))
    model_Inception.add(Dense(2, activation='softmax'))

    # Model summary
    print("Model Summary (InceptionV3):")
    model_Inception.summary()
    print()

    # Compile the model
    model_Inception.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

    # Train the model with EarlyStopping
    history_Inception = model_Inception.fit(train_data, epochs=EPOCHS, validation_data=test_generator, callback

    # Validate the model
    val_loss_Inception, val_accuracy_Inception = model_Inception.evaluate(test_generator, steps=len(test_genera
    print(f'Validation Loss: {val_loss_Inception:.4f}')
    print(f'Validation Accuracy: {val_accuracy_Inception:.4f}')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_we
ights_tf_dim_ordering_tf_kernels_notop.h5
87910968/87910968 [==============================] - 1s 0us/step
Model Summary (InceptionV3):
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 inception_v3 (Functional)   (None, 8, 8, 2048)        21802784  
                                                                 
 global_average_pooling2d (G  (None, 2048)             0         
 lobalAveragePooling2D)                                          
                                                                 
 dense (Dense)               (None, 128)               262272    
                                                                 
 dropout (Dropout)           (None, 128)               0         
                                                                 
 dense_1 (Dense)             (None, 64)                8256      
                                                                 
 dropout_1 (Dropout)         (None, 64)                0         
                                                                 
 dense_2 (Dense)             (None, 2)                 130       
                                                                 
=================================================================
Total params: 22,073,442
Trainable params: 270,658
Non-trainable params: 21,802,784
_________________________________________________________________

Epoch 1/50
10/25 [===========>..................] - ETA: 18s - loss: 1.0042 - accuracy: 0.6060
```

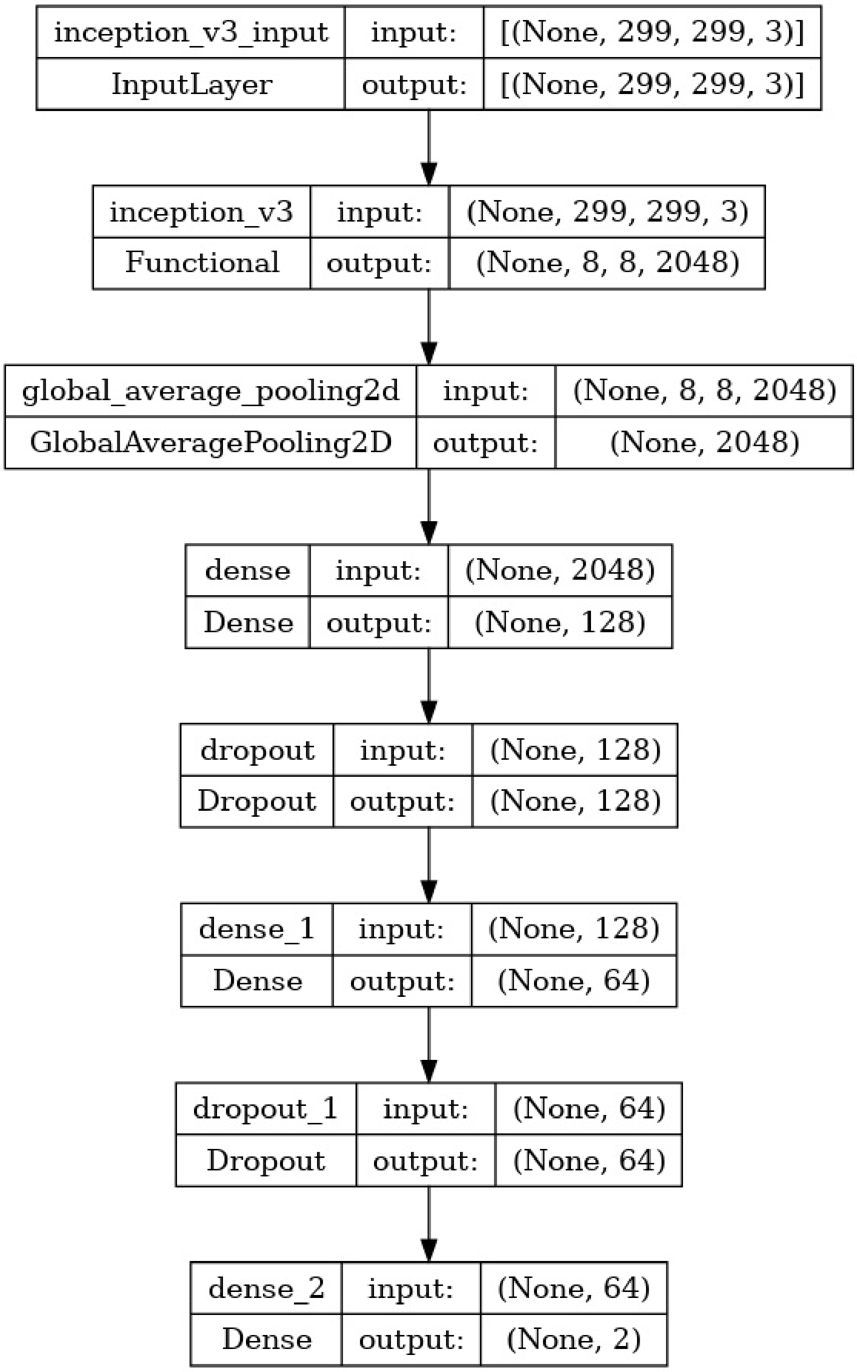
```
/opt/conda/lib/python3.10/site-packages/PIL/Image.py:992: UserWarning: Palette images with Transparency express
ed in bytes should be converted to RGBA images
  warnings.warn(
```

```
25/25 [==============================] - 130s 4s/step - loss: 0.9185 - accuracy: 0.6639 - val_loss: 0.7113 - va
l_accuracy: 0.7995
Epoch 2/50
25/25 [==============================] - 130s 5s/step - loss: 0.7271 - accuracy: 0.7835 - val_loss: 0.6214 - va
l_accuracy: 0.8366
Epoch 3/50
25/25 [==============================] - 95s 4s/step - loss: 0.6496 - accuracy: 0.8121 - val_loss: 0.5873 - val
_accuracy: 0.8500
Epoch 4/50
25/25 [==============================] - 96s 4s/step - loss: 0.5962 - accuracy: 0.8425 - val_loss: 0.5319 - val
_accuracy: 0.8576
Epoch 5/50
25/25 [==============================] - 94s 4s/step - loss: 0.5619 - accuracy: 0.8408 - val_loss: 0.5469 - val
_accuracy: 0.8382
Epoch 6/50
25/25 [==============================] - 128s 5s/step - loss: 0.5109 - accuracy: 0.8618 - val_loss: 0.5070 - va
l_accuracy: 0.8551
Epoch 7/50
25/25 [==============================] - 93s 4s/step - loss: 0.5167 - accuracy: 0.8593 - val_loss: 0.4491 - val
_accuracy: 0.8778
Epoch 8/50
25/25 [==============================] - 129s 5s/step - loss: 0.4807 - accuracy: 0.8694 - val_loss: 0.4522 - va
l_accuracy: 0.8719
Epoch 9/50
25/25 [==============================] - 94s 4s/step - loss: 0.4766 - accuracy: 0.8669 - val_loss: 0.4300 - val
_accuracy: 0.8787
Epoch 10/50
25/25 [==============================] - 95s 4s/step - loss: 0.4514 - accuracy: 0.8762 - val_loss: 0.4051 - val
_accuracy: 0.8837
Epoch 11/50
25/25 [==============================] - 95s 4s/step - loss: 0.4348 - accuracy: 0.8652 - val_loss: 0.3809 - val
_accuracy: 0.8964
Epoch 12/50
25/25 [==============================] - 95s 4s/step - loss: 0.4064 - accuracy: 0.8880 - val_loss: 0.3828 - val
_accuracy: 0.8913
Epoch 13/50
25/25 [==============================] - 94s 4s/step - loss: 0.3955 - accuracy: 0.8880 - val_loss: 0.3688 - val
_accuracy: 0.8905
Epoch 14/50
25/25 [==============================] - 94s 4s/step - loss: 0.3884 - accuracy: 0.8880 - val_loss: 0.3645 - val
_accuracy: 0.9006
Epoch 15/50
25/25 [==============================] - 94s 4s/step - loss: 0.3845 - accuracy: 0.8964 - val_loss: 0.3305 - val
_accuracy: 0.9090
Epoch 16/50
25/25 [==============================] - 94s 4s/step - loss: 0.3494 - accuracy: 0.9124 - val_loss: 0.3234 - val
_accuracy: 0.9073
Epoch 17/50
25/25 [==============================] - 93s 4s/step - loss: 0.3497 - accuracy: 0.9031 - val_loss: 0.3378 - val
_accuracy: 0.8930
Epoch 18/50
25/25 [==============================] - 94s 4s/step - loss: 0.3269 - accuracy: 0.9183 - val_loss: 0.3151 - val
_accuracy: 0.9158
Epoch 19/50
25/25 [==============================] - 93s 4s/step - loss: 0.3313 - accuracy: 0.9090 - val_loss: 0.3050 - val
_accuracy: 0.9149
Epoch 20/50
25/25 [==============================] - 93s 4s/step - loss: 0.3355 - accuracy: 0.9073 - val_loss: 0.2992 - val
_accuracy: 0.9166
Epoch 21/50
25/25 [==============================] - 93s 4s/step - loss: 0.3137 - accuracy: 0.9082 - val_loss: 0.3081 - val
_accuracy: 0.9040
Epoch 22/50
25/25 [==============================] - 93s 4s/step - loss: 0.3171 - accuracy: 0.9132 - val_loss: 0.2825 - val
_accuracy: 0.9250
Epoch 23/50
25/25 [==============================] - 94s 4s/step - loss: 0.2985 - accuracy: 0.9107 - val_loss: 0.2806 - val
_accuracy: 0.9225
Epoch 24/50
25/25 [==============================] - 93s 4s/step - loss: 0.3178 - accuracy: 0.9099 - val_loss: 0.2921 - val
_accuracy: 0.9115
Epoch 25/50
25/25 [==============================] - ETA: 0s - loss: 0.3389 - accuracy: 0.9006Restoring model weights from
the end of the best epoch: 22.
25/25 [==============================] - 93s 4s/step - loss: 0.3389 - accuracy: 0.9006 - val_loss: 0.2839 - val
_accuracy: 0.9233
Epoch 25: early stopping
25/25 [==============================] - 47s 2s/step - loss: 0.2923 - accuracy: 0.9166
Validation Loss: 0.2923
Validation Accuracy: 0.9166
```

In [14]:

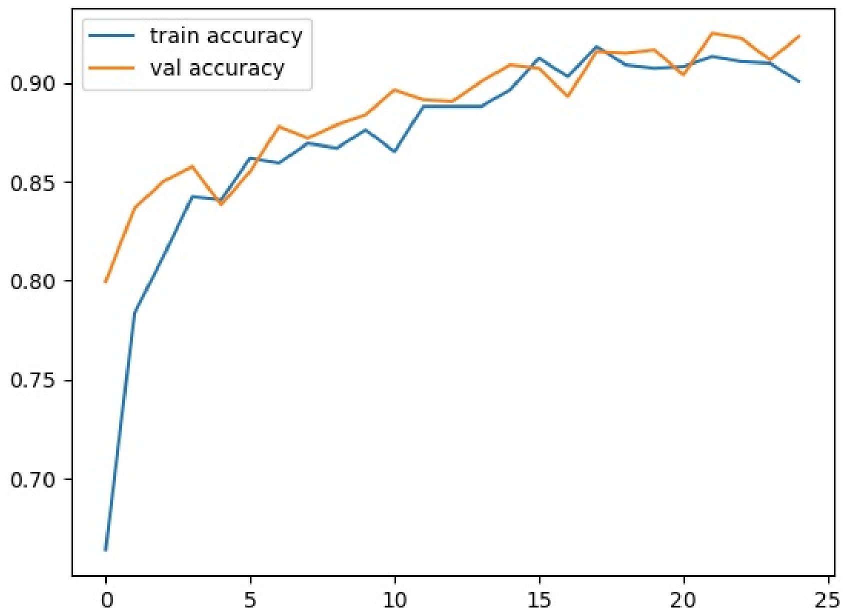
```
from tensorflow.keras.utils import plot_model
plot_model(model_Inception, show_shapes=True, show_layer_names=True)
```

Out[14]:

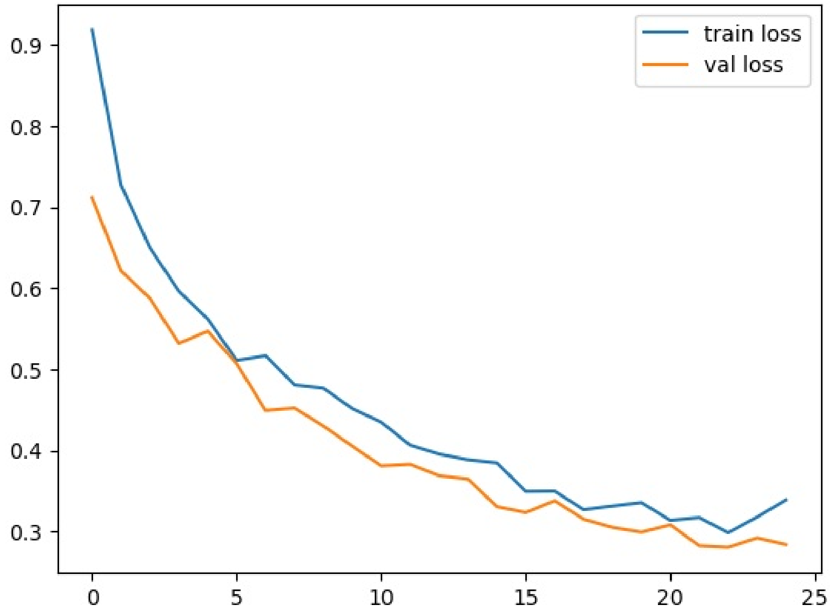


In [15]:

```
plt.plot(history_Inception.history['accuracy'],label='train accuracy')
plt.plot(history_Inception.history['val_accuracy'],label='val accuracy')
plt.legend()
plt.show()
```



In [16]:
```
plt.plot(history_Inception.history['loss'],label='train loss')
plt.plot(history_Inception.history['val_loss'],label='val loss')
plt.legend()
plt.show()
plt.savefig("LossVal_loss")
```




```
<Figure size 640x480 with 0 Axes>
```

Result Classification

In [17]:
```
test_generator.reset()
img, label = next(test_generator)

prediction = model_Inception.predict(img)
test_pred_classes = np.argmax(prediction, axis=1)

plt.figure(figsize=[14, 14])
for i in range(20):
    plt.subplot(5, 4, i+1)
    plt.imshow(img[i])
    plt.axis('off')
    plt.title("Label : {}\n Prediction : {} {:.1f}%".format(class_names[np.argmax(label[i])], class_names[test_
plt.show()
```

```
2/2 [==============================] - 8s 2s/step
```

Conclusion

Inception V3 is a powerful deep learning model that is used for solar panel dusty image classification tasks. It is a versatile and efficient model. It has achieved an accuracy of 91% on the solar panel dataset.

In [ ]: