cnn-deep-learing-model-training

May 25, 2023

Import all the Dependencies

[1]:
```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
from IPython.display import HTML
```

Set all the Constants

[15]:
```
BATCH_SIZE = 2
IMAGE_SIZE = 10
CHANNELS=3
EPOCHS=20
```

Import data into tensorflow dataset object

We will use image_dataset_from_directory api to load all images in tensorflow dataset:

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

[16]:
```
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "images",
    seed=123,
    shuffle=True,
    image_size=(IMAGE_SIZE,IMAGE_SIZE),
    batch_size=BATCH_SIZE
)
```

```
Found 20 files belonging to 2 classes.
```

[17]:
```
class_names = dataset.class_names
class_names
```

[17]:
```
['Dust', 'No_Dust']
```

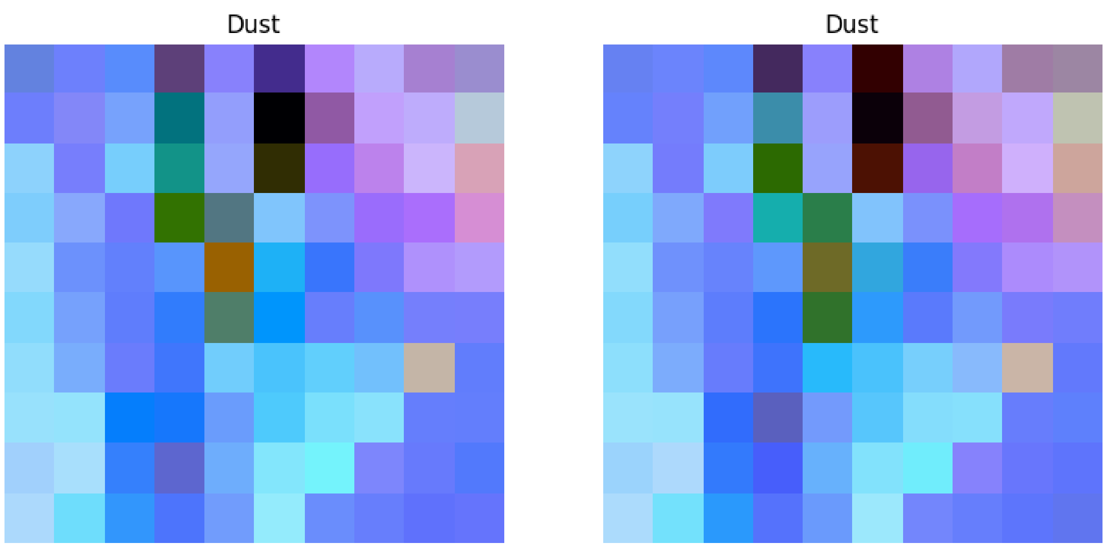
[18]:
```
for image_batch, labels_batch in dataset.take(1):
    print(image_batch.shape)
    print(labels_batch.numpy())
```

```
(2, 10, 10, 3)
[0 0]
```

As you can see above, each element in the dataset is a tuple. First element is a batch of 10 elements of images. Second element is a batch of 10 elements of class labels

Visualize some of the images from our dataset:

[69]:
```
plt.figure(figsize=(10, 10))
for image_batch, labels_batch in dataset.take(3):
    for i in range(2):
        ax = plt.subplot(1, 2, i + 1)
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.title(class_names[labels_batch[i]])
        plt.axis("off")
```



Function to Split Dataset

Dataset should be bifurcated into 3 subsets, namely:

1. Training: Dataset to be used while training:
2. Validation: Dataset to be tested against while training:
3. Test: Dataset to be tested against after we trained a model:

[19]:
```
len(dataset)
```

[19]: 10

[20]:
```
train_size = 0.8
len(dataset)*train_size
```

[20]: 8.0

[22]:
```python
train_ds = dataset.take(8)
len(train_ds)
```

[22]: 8

[23]:
```python
test_ds = dataset.skip(8)
len(test_ds)
```

[23]: 2

[24]:
```python
val_size=0.1
len(dataset)*val_size
```

[24]: 1.0

[26]:
```python
val_ds = test_ds.take(1)
len(val_ds)
```

[26]: 1

[27]:
```python
test_ds = test_ds.skip(1)
len(test_ds)
```

[27]: 1

function to split data into train,test and validation

[28]:
```python
def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1, test_split=0.
↪1, shuffle=True, shuffle_size=10000):
    assert (train_split + test_split + val_split) == 1

    ds_size = len(ds)

    if shuffle:
        ds = ds.shuffle(shuffle_size, seed=12)

    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)

    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)

    return train_ds, val_ds, test_ds
```

[29]:
```
train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)
```

[30]:
```
len(train_ds)
```

[30]: 8

[31]:
```
len(val_ds)
```

[31]: 1

[32]:
```
len(test_ds)
```

[32]: 1

Cache, Shuffle, and Prefetch the Dataset

[33]:
```
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
```

Building the Model

Creating a Layer for Resizing and Normalization

Before we feed our images to network, we should be resizing it to the desired size. Moreover, to improve model performance, we should normalize the image pixel value (keeping them in range 0 and 1 by dividing by 256). This should happen while training as well as inference. Hence we can add that as a layer in our Sequential Model.

You might be thinking why do we need to resize (256,256) image to again (256,256). You are right we don't need to but this will be useful when we are done with the training and start using the model for predictions. At that time somone can supply an image that is not (256,256) and this layer will resize it

[34]:
```
resize_and_rescale = tf.keras.Sequential([
  layers.experimental.preprocessing.Resizing(IMAGE_SIZE, IMAGE_SIZE),
  layers.experimental.preprocessing.Rescaling(1./255),
])
```

Data Augmentation

Data Augmentation is needed when we have less data, this boosts the accuracy of our model by augmenting the data.

[35]:
```
data_augmentation = tf.keras.Sequential([
  layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
  layers.experimental.preprocessing.RandomRotation(0.2),
])
```

Applying Data Augmentation to Train Dataset

[36]:
```
train_ds = train_ds.map(
    lambda x, y: (data_augmentation(x, training=True), y)
).prefetch(buffer_size=tf.data.AUTOTUNE)
```

```
WARNING:tensorflow:From C:\Users\ABDUL QADIR\.conda\envs\python\lib\site-
packages\tensorflow\python\autograph\pyct\static_analysis\liveness.py:83:
Analyzer.lamba_check (from
tensorflow.python.autograph.pyct.static_analysis.liveness) is deprecated and
will be removed after 2023-09-23.
Instructions for updating:
Lambda fuctions will be no more assumed to be used in the statement where they
are used, or at least in the same block.
https://github.com/tensorflow/tensorflow/issues/56089
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there
is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no
registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no
registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2
cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3
cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there
is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no
registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no
registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2
cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3
cause there is no registered converter for this op.
```

We are going to use convolutional neural network (CNN) here. CNN is popular for image classification tasks. Watch below video to understand fundamentals of CNN

[38]:
```
input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 2

model = models.Sequential([
    resize_and_rescale,
    layers.Conv2D(16, kernel_size = (3,3), activation='relu',␣
 ↪input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(16, activation='relu'),
    layers.Dense(n_classes, activation='softmax'),
```

```
])

model.build(input_shape=input_shape)
```

[39]:
```
model.summary()
```

```
Model: "sequential_3"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (2, 10, 10, 3)            0

 conv2d_3 (Conv2D)           (2, 8, 8, 16)             448

 max_pooling2d_3 (MaxPooling  (2, 4, 4, 16)            0
 2D)

 flatten_1 (Flatten)         (2, 256)                  0

 dense_2 (Dense)             (2, 16)                   4112

 dense_3 (Dense)             (2, 2)                    34

=================================================================
Total params: 4,594
Trainable params: 4,594
Non-trainable params: 0
_________________________________________________________________
```

Compiling the Model

We use `adam` Optimizer, `SparseCategoricalCrossentropy` for losses, `accuracy` as a metric

[40]:
```
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
```

[ ]:
```

```

[41]:
```
history = model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
    verbose=1,
    epochs=50,
)
```

```
Epoch 1/50
8/8 [==============================] - 1s 61ms/step - loss: 0.7346 - accuracy:
0.5000 - val_loss: 0.6438 - val_accuracy: 1.0000
Epoch 2/50
8/8 [==============================] - 0s 3ms/step - loss: 0.6988 - accuracy:
0.5000 - val_loss: 1.0308 - val_accuracy: 0.0000e+00
Epoch 3/50
8/8 [==============================] - 0s 5ms/step - loss: 0.6631 - accuracy:
0.5625 - val_loss: 1.0310 - val_accuracy: 0.0000e+00
Epoch 4/50
8/8 [==============================] - 0s 5ms/step - loss: 0.6984 - accuracy:
0.5625 - val_loss: 0.8145 - val_accuracy: 0.0000e+00
Epoch 5/50
8/8 [==============================] - 0s 7ms/step - loss: 0.6439 - accuracy:
0.5625 - val_loss: 0.9262 - val_accuracy: 0.0000e+00
Epoch 6/50
8/8 [==============================] - 0s 5ms/step - loss: 0.6150 - accuracy:
0.5625 - val_loss: 0.8522 - val_accuracy: 0.0000e+00
Epoch 7/50
8/8 [==============================] - 0s 4ms/step - loss: 0.6072 - accuracy:
0.6250 - val_loss: 0.8623 - val_accuracy: 0.0000e+00
Epoch 8/50
8/8 [==============================] - 0s 6ms/step - loss: 0.5974 - accuracy:
0.6875 - val_loss: 0.8645 - val_accuracy: 0.0000e+00
Epoch 9/50
8/8 [==============================] - 0s 3ms/step - loss: 0.6129 - accuracy:
0.6875 - val_loss: 0.8734 - val_accuracy: 0.0000e+00
Epoch 10/50
8/8 [==============================] - 0s 5ms/step - loss: 0.5797 - accuracy:
0.8750 - val_loss: 0.8253 - val_accuracy: 0.0000e+00
Epoch 11/50
8/8 [==============================] - 0s 3ms/step - loss: 0.5772 - accuracy:
0.8125 - val_loss: 0.9131 - val_accuracy: 0.0000e+00
Epoch 12/50
8/8 [==============================] - 0s 5ms/step - loss: 0.5714 - accuracy:
0.6250 - val_loss: 0.9068 - val_accuracy: 0.0000e+00
Epoch 13/50
8/8 [==============================] - 0s 5ms/step - loss: 0.5731 - accuracy:
0.6250 - val_loss: 0.9035 - val_accuracy: 0.0000e+00
Epoch 14/50
8/8 [==============================] - 0s 5ms/step - loss: 0.5752 - accuracy:
0.8125 - val_loss: 0.6659 - val_accuracy: 0.5000
Epoch 15/50
8/8 [==============================] - 0s 5ms/step - loss: 0.5684 - accuracy:
0.9375 - val_loss: 0.8188 - val_accuracy: 0.5000
Epoch 16/50
8/8 [==============================] - 0s 6ms/step - loss: 0.5090 - accuracy:
0.8750 - val_loss: 0.8846 - val_accuracy: 0.5000
```

```
Epoch 17/50
8/8 [==============================] - 0s 3ms/step - loss: 0.5323 - accuracy:
0.8750 - val_loss: 0.7933 - val_accuracy: 0.5000
Epoch 18/50
8/8 [==============================] - 0s 5ms/step - loss: 0.5056 - accuracy:
0.9375 - val_loss: 0.7813 - val_accuracy: 0.5000
Epoch 19/50
8/8 [==============================] - 0s 4ms/step - loss: 0.4773 - accuracy:
0.9375 - val_loss: 0.8087 - val_accuracy: 0.5000
Epoch 20/50
8/8 [==============================] - 0s 3ms/step - loss: 0.4870 - accuracy:
0.8750 - val_loss: 0.8567 - val_accuracy: 0.5000
Epoch 21/50
8/8 [==============================] - 0s 4ms/step - loss: 0.4772 - accuracy:
0.8750 - val_loss: 0.7362 - val_accuracy: 0.5000
Epoch 22/50
8/8 [==============================] - 0s 4ms/step - loss: 0.4587 - accuracy:
0.8750 - val_loss: 0.6157 - val_accuracy: 0.5000
Epoch 23/50
8/8 [==============================] - 0s 3ms/step - loss: 0.4771 - accuracy:
0.7500 - val_loss: 0.7416 - val_accuracy: 0.5000
Epoch 24/50
8/8 [==============================] - 0s 3ms/step - loss: 0.5096 - accuracy:
0.7500 - val_loss: 0.9310 - val_accuracy: 0.5000
Epoch 25/50
8/8 [==============================] - 0s 5ms/step - loss: 0.4422 - accuracy:
0.8750 - val_loss: 0.7288 - val_accuracy: 0.5000
Epoch 26/50
8/8 [==============================] - 0s 5ms/step - loss: 0.4730 - accuracy:
0.8125 - val_loss: 0.5720 - val_accuracy: 0.5000
Epoch 27/50
8/8 [==============================] - 0s 5ms/step - loss: 0.3893 - accuracy:
0.8750 - val_loss: 0.6707 - val_accuracy: 0.5000
Epoch 28/50
8/8 [==============================] - 0s 10ms/step - loss: 0.4381 - accuracy:
0.8125 - val_loss: 0.9700 - val_accuracy: 0.5000
Epoch 29/50
8/8 [==============================] - 0s 3ms/step - loss: 0.3904 - accuracy:
0.9375 - val_loss: 0.7758 - val_accuracy: 0.5000
Epoch 30/50
8/8 [==============================] - 0s 5ms/step - loss: 0.3871 - accuracy:
0.9375 - val_loss: 0.6170 - val_accuracy: 0.5000
Epoch 31/50
8/8 [==============================] - 0s 4ms/step - loss: 0.3575 - accuracy:
0.9375 - val_loss: 0.7775 - val_accuracy: 0.5000
Epoch 32/50
8/8 [==============================] - 0s 4ms/step - loss: 0.3286 - accuracy:
0.9375 - val_loss: 0.7778 - val_accuracy: 0.5000
```

```
Epoch 33/50
8/8 [==============================] - 0s 5ms/step - loss: 0.3476 - accuracy:
0.8750 - val_loss: 0.7600 - val_accuracy: 0.5000
Epoch 34/50
8/8 [==============================] - 0s 4ms/step - loss: 0.4343 - accuracy:
0.8125 - val_loss: 0.7962 - val_accuracy: 0.5000
Epoch 35/50
8/8 [==============================] - 0s 5ms/step - loss: 0.3599 - accuracy:
0.8750 - val_loss: 0.5675 - val_accuracy: 0.5000
Epoch 36/50
8/8 [==============================] - 0s 5ms/step - loss: 0.3277 - accuracy:
0.9375 - val_loss: 0.6416 - val_accuracy: 0.5000
Epoch 37/50
8/8 [==============================] - 0s 4ms/step - loss: 0.3041 - accuracy:
0.9375 - val_loss: 0.9248 - val_accuracy: 0.5000
Epoch 38/50
8/8 [==============================] - 0s 5ms/step - loss: 0.2676 - accuracy:
0.9375 - val_loss: 0.5116 - val_accuracy: 0.5000
Epoch 39/50
8/8 [==============================] - 0s 5ms/step - loss: 0.2896 - accuracy:
0.9375 - val_loss: 0.5502 - val_accuracy: 0.5000
Epoch 40/50
8/8 [==============================] - 0s 4ms/step - loss: 0.2465 - accuracy:
0.9375 - val_loss: 0.9112 - val_accuracy: 0.5000
Epoch 41/50
8/8 [==============================] - 0s 3ms/step - loss: 0.2426 - accuracy:
0.9375 - val_loss: 0.9225 - val_accuracy: 0.5000
Epoch 42/50
8/8 [==============================] - 0s 4ms/step - loss: 0.2827 - accuracy:
0.9375 - val_loss: 0.5696 - val_accuracy: 0.5000
Epoch 43/50
8/8 [==============================] - 0s 5ms/step - loss: 0.1882 - accuracy:
1.0000 - val_loss: 0.9291 - val_accuracy: 0.5000
Epoch 44/50
8/8 [==============================] - 0s 4ms/step - loss: 0.1809 - accuracy:
1.0000 - val_loss: 0.9048 - val_accuracy: 0.5000
Epoch 45/50
8/8 [==============================] - 0s 5ms/step - loss: 0.1699 - accuracy:
1.0000 - val_loss: 0.8929 - val_accuracy: 0.5000
Epoch 46/50
8/8 [==============================] - 0s 5ms/step - loss: 0.1496 - accuracy:
1.0000 - val_loss: 0.8392 - val_accuracy: 0.5000
Epoch 47/50
8/8 [==============================] - 0s 4ms/step - loss: 0.1526 - accuracy:
1.0000 - val_loss: 0.8395 - val_accuracy: 0.5000
Epoch 48/50
8/8 [==============================] - 0s 5ms/step - loss: 0.1571 - accuracy:
1.0000 - val_loss: 0.9083 - val_accuracy: 0.5000
```

```
Epoch 49/50
8/8 [==============================] - 0s 4ms/step - loss: 0.1129 - accuracy:
1.0000 - val_loss: 0.9196 - val_accuracy: 0.5000
Epoch 50/50
8/8 [==============================] - 0s 5ms/step - loss: 0.1471 - accuracy:
1.0000 - val_loss: 0.7480 - val_accuracy: 0.5000
```

Saving the Model

[58]:
```
model.save('AiModeL.h5')
```

[59]:
```
scores = model.evaluate(test_ds)
```

```
1/1 [==============================] - 0s 24ms/step - loss: 0.4886 - accuracy:
0.5000
```

You can see above that we get 71.167% accuracy for our test dataset. This is considered to be a pretty good accuracy

[60]:
```
scores
```

[60]:
```
[0.48860958218574524, 0.5]
```

Scores is just a list containing loss and accuracy value:

Plotting the Accuracy and Loss Curves

[ ]:
```
history
```

You can read documentation on history object here: https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/History

[ ]:
```
history.params
```

[ ]:
```
history.history.keys()
```

loss, accuracy, val loss etc are a python list containing values of loss, accuracy etc at the end of each epoch

[43]:
```
type(history.history['loss'])
```

[43]:
```
list
```

[44]:
```
len(history.history['loss'])
```

[44]:
```
50
```

[45]:
```
history.history['loss'][:5] # show loss for first 5 epochs
```

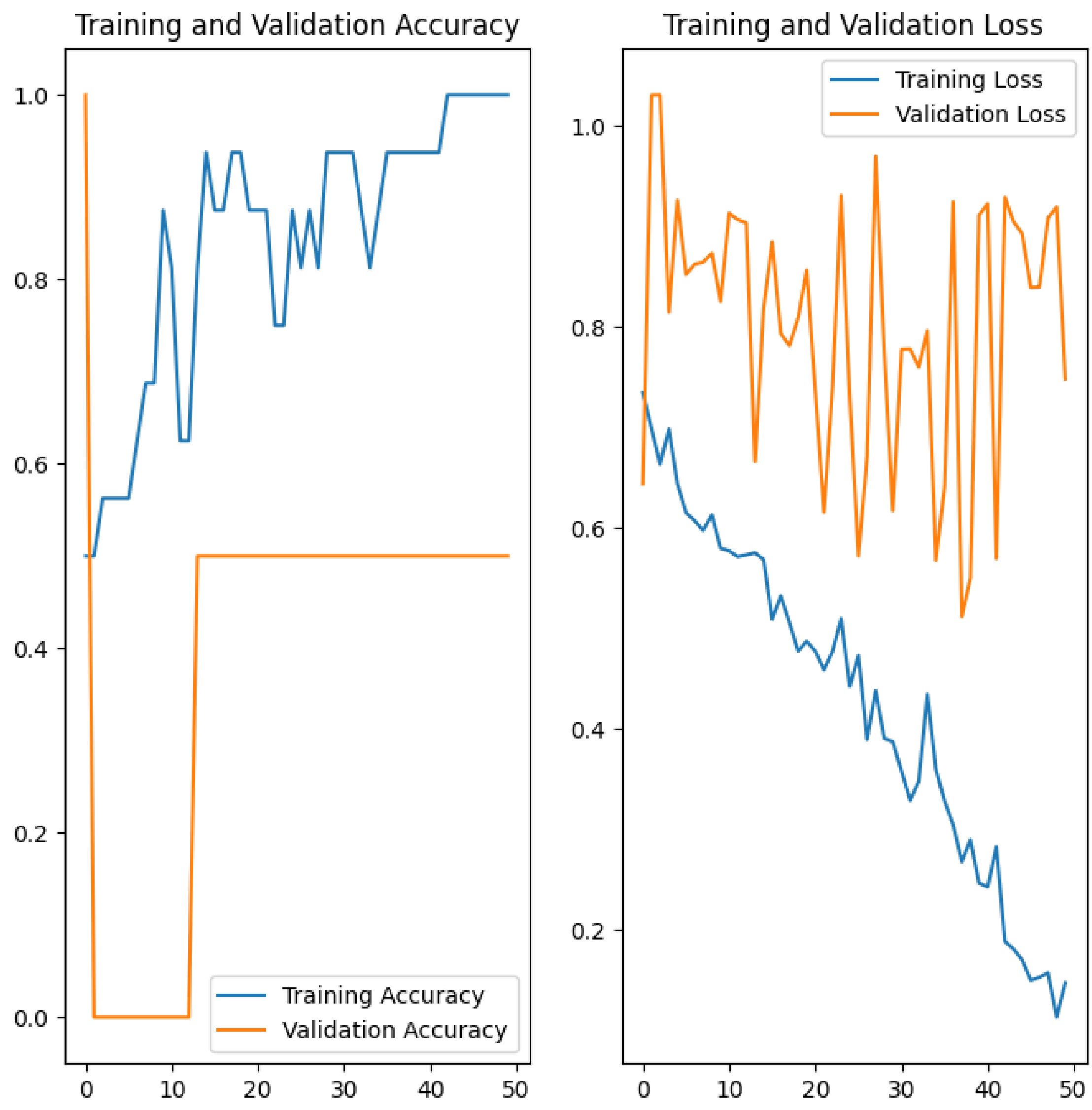
[45]: [0.7346152067184448,
 0.6988462805747986,
 0.663068413734436,
 0.698443591594696,
 0.6438900828361511]

[46]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']
```

[47]:
```
EPOCHS=50
```

Training accuracy vs Epcohs , validation accuracy vs Epochs , Training and validation loss vs Epochs

[48]:
```
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(range(EPOCHS), acc, label='Training Accuracy')
plt.plot(range(EPOCHS), val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(range(EPOCHS), loss, label='Training Loss')
plt.plot(range(EPOCHS), val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

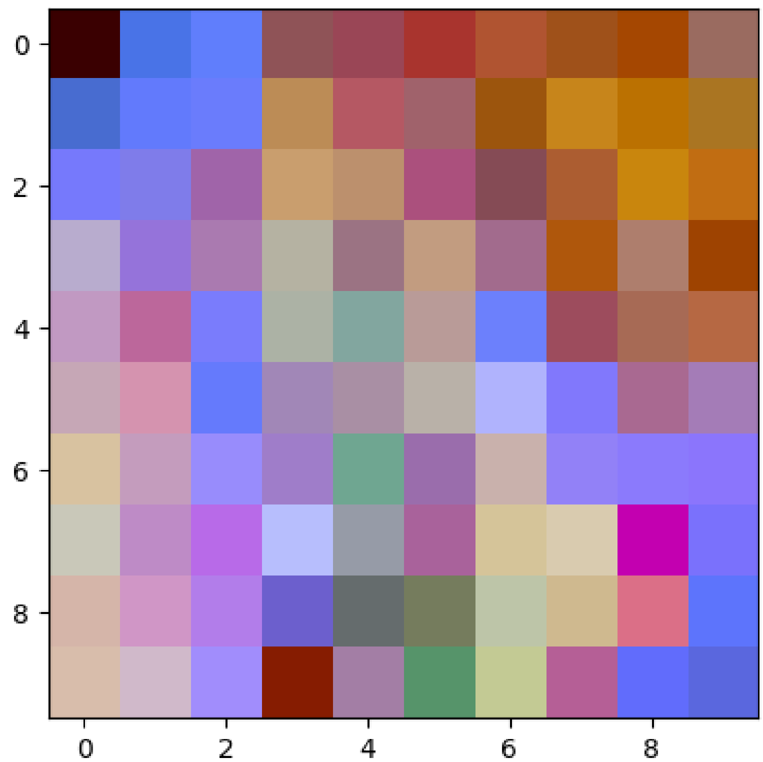


Run prediction on a sample image

[49]:
```
import numpy as np
for images_batch, labels_batch in test_ds.take(1):

    first_image = images_batch[0].numpy().astype('uint8')
    first_label = labels_batch[0].numpy()

    print("first image to predict")
    plt.imshow(first_image)
    print("actual label:",class_names[first_label])

    batch_prediction = model.predict(images_batch)
```

```
    print("predicted label:",class_names[np.argmax(batch_prediction[0])])
```

```
first image to predict
actual label: No_Dust
1/1 [==============================] - 0s 110ms/step
predicted label: No_Dust
```



Write a function for inference

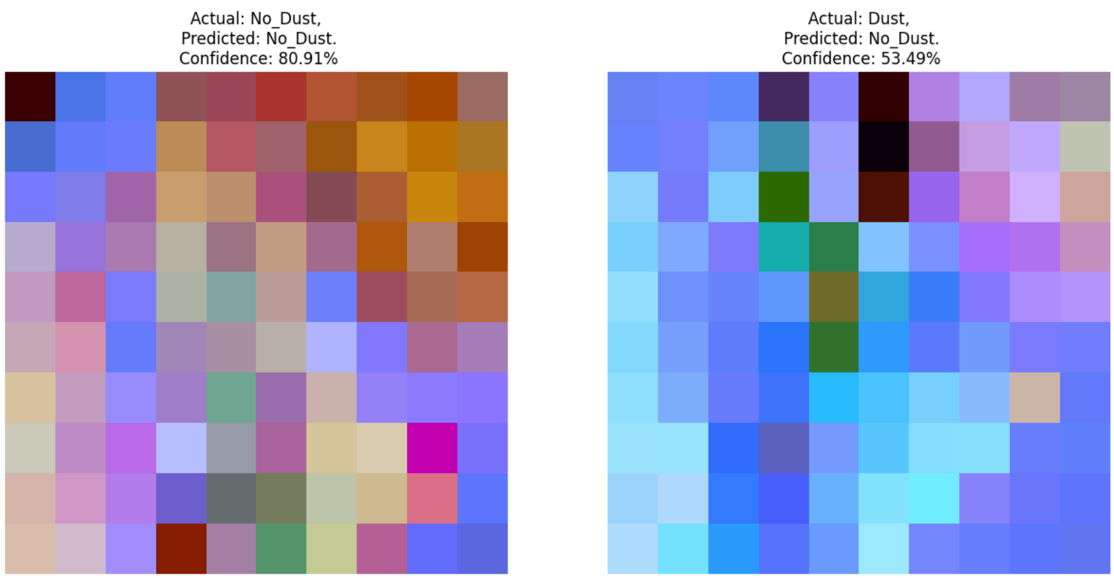
```
[50]: def predict(model, img):
          img_array = tf.keras.preprocessing.image.img_to_array(images[i].numpy())
          img_array = tf.expand_dims(img_array, 0)

          predictions = model.predict(img_array)

          predicted_class = class_names[np.argmax(predictions[0])]
          confidence = round(100 * (np.max(predictions[0])), 2)
          return predicted_class, confidence
```

Now run inference on few sample images

```
[61]: plt.figure(figsize=(15, 15))
      for images, labels in test_ds.take(1):
```

```python
    for i in range(2):
        ax = plt.subplot(1,2,i+ 1)
        plt.imshow(images[i].numpy().astype("uint8"))

        predicted_class, confidence = predict(model, images[i].numpy())
        actual_class = class_names[labels[i]]

        plt.title(f"Actual: {actual_class},\n Predicted: {predicted_class}.\n Confidence: {confidence}%")

        plt.axis("off")
```

```
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 16ms/step
```



Load the saved Model

[62]:
```python
from tensorflow.keras.models import load_model
```

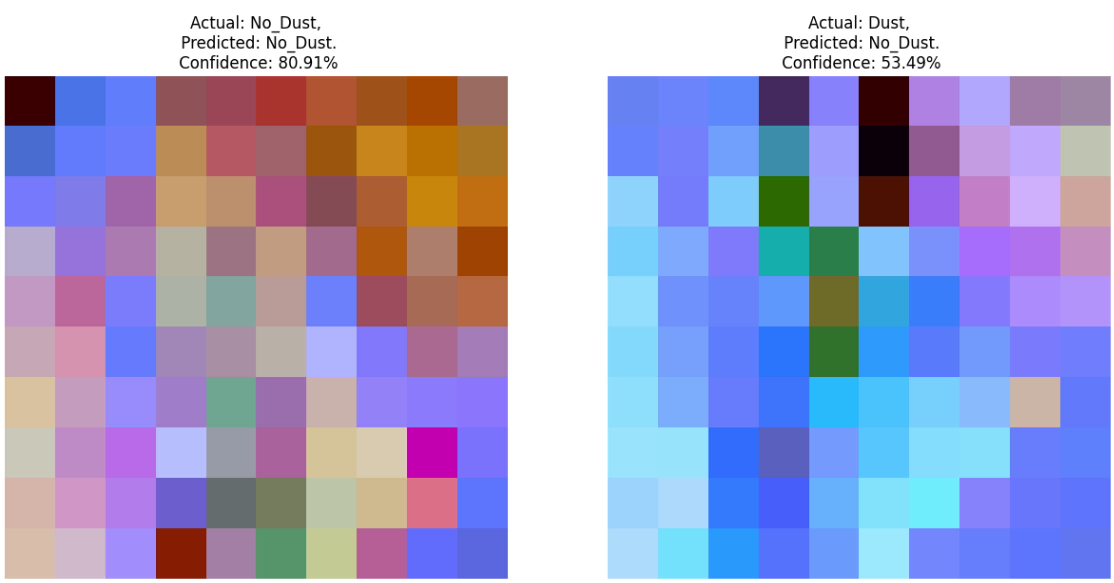
[63]:
```python
model = load_model('AiModeL.h5')
```

[64]:
```python
import cv2
from tensorflow.keras.preprocessing import image

import os
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.models import load_model
```

```python
[65]: def predict(model, img):
          img_array = tf.keras.preprocessing.image.img_to_array(images[i].numpy())
          img_array = tf.expand_dims(img_array, 0)

          predictions = model.predict(img_array)

          predicted_class = class_names[np.argmax(predictions[0])]
          confidence = round(100 * (np.max(predictions[0])), 2)
          return predicted_class, confidence
```

```python
[68]: plt.figure(figsize=(15, 15))
      for images, labels in test_ds.take(3):
          for i in range(2):
              ax = plt.subplot(1 ,2, i + 1)
              plt.imshow(images[i].numpy().astype("uint8"))

              predicted_class, confidence = predict(model, images[i].numpy())
              actual_class = class_names[labels[i]]

              plt.title(f"Actual: {actual_class},\n Predicted: {predicted_class}.\n Confidence: {confidence}%")

              plt.axis("off")
```

```
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
```



1 END