In [1]:

```
import numpy as np
from tqdm import tqdm
import os
import cv2
import shutil
import random
import time
import matplotlib.pyplot as plt
%matplotlib inline
import tensorflow as tf
from tensorflow import keras
from keras import backend as K
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten,SpatialDropout2
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model
from tensorflow.keras.applications import imagenet_utils
from tensorflow. keras.applications import MobileNet
from tensorflow.keras.applications.mobilenet import preprocess_input
from IPython.display import Image
from sklearn.metrics import confusion_matrix
import itertools
import datetime
import time
```

In [2]:

```
mobile = tf.keras.applications.mobilenet.MobileNet()
```

In [3]:

```
mobile.summary()
```

```
Model: "mobilenet_1.00_224"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0         
                                                                 
 conv1 (Conv2D)              (None, 112, 112, 32)      864       
                                                                 
 conv1_bn (BatchNormalizatio  (None, 112, 112, 32)     128       
 n)                                                              
                                                                 
 conv1_relu (ReLU)           (None, 112, 112, 32)      0         
                                                                 
 conv_dw_1 (DepthwiseConv2D)  (None, 112, 112, 32)     288       
                                                                 
 conv_dw_1_bn (BatchNormaliz  (None, 112, 112, 32)     128       
 ation)                                                          
                                                                 
 conv_dw_1_relu (ReLU)       (None, 112, 112, 32)      0         
                                                                 
 conv_pw_1 (Conv2D)          (None, 112, 112, 64)      2048      
                                                                 
 conv_pw_1_bn (BatchNormaliz  (None, 112, 112, 64)     256       
 ation)                                                          
                                                                 
 conv_pw_1_relu (ReLU)       (None, 112, 112, 64)      0         
```

```
conv_pad_2 (ZeroPadding2D)  (None, 113, 113, 64)      0

conv_dw_2 (DepthwiseConv2D)  (None, 56, 56, 64)       576

conv_dw_2_bn (BatchNormaliz  (None, 56, 56, 64)       256
ation)

conv_dw_2_relu (ReLU)       (None, 56, 56, 64)        0

conv_pw_2 (Conv2D)          (None, 56, 56, 128)       8192

conv_pw_2_bn (BatchNormaliz  (None, 56, 56, 128)      512
ation)

conv_pw_2_relu (ReLU)       (None, 56, 56, 128)       0

conv_dw_3 (DepthwiseConv2D)  (None, 56, 56, 128)      1152

conv_dw_3_bn (BatchNormaliz  (None, 56, 56, 128)      512
ation)

conv_dw_3_relu (ReLU)       (None, 56, 56, 128)       0

conv_pw_3 (Conv2D)          (None, 56, 56, 128)       16384

conv_pw_3_bn (BatchNormaliz  (None, 56, 56, 128)      512
ation)

conv_pw_3_relu (ReLU)       (None, 56, 56, 128)       0

conv_pad_4 (ZeroPadding2D)  (None, 57, 57, 128)       0

conv_dw_4 (DepthwiseConv2D)  (None, 28, 28, 128)      1152

conv_dw_4_bn (BatchNormaliz  (None, 28, 28, 128)      512
ation)

conv_dw_4_relu (ReLU)       (None, 28, 28, 128)       0

conv_pw_4 (Conv2D)          (None, 28, 28, 256)       32768

conv_pw_4_bn (BatchNormaliz  (None, 28, 28, 256)      1024
ation)

conv_pw_4_relu (ReLU)       (None, 28, 28, 256)       0

conv_dw_5 (DepthwiseConv2D)  (None, 28, 28, 256)      2304

conv_dw_5_bn (BatchNormaliz  (None, 28, 28, 256)      1024
ation)

conv_dw_5_relu (ReLU)       (None, 28, 28, 256)       0

conv_pw_5 (Conv2D)          (None, 28, 28, 256)       65536

conv_pw_5_bn (BatchNormaliz  (None, 28, 28, 256)      1024
ation)

conv_pw_5_relu (ReLU)       (None, 28, 28, 256)       0

conv_pad_6 (ZeroPadding2D)  (None, 29, 29, 256)       0

conv_dw_6 (DepthwiseConv2D)  (None, 14, 14, 256)      2304
```

```
conv_dw_6_bn (BatchNormaliz  (None, 14, 14, 256)      1024
ation)

conv_dw_6_relu (ReLU)       (None, 14, 14, 256)       0

conv_pw_6 (Conv2D)          (None, 14, 14, 512)       131072

conv_pw_6_bn (BatchNormaliz  (None, 14, 14, 512)      2048
ation)

conv_pw_6_relu (ReLU)       (None, 14, 14, 512)       0

conv_dw_7 (DepthwiseConv2D)  (None, 14, 14, 512)      4608

conv_dw_7_bn (BatchNormaliz  (None, 14, 14, 512)      2048
ation)

conv_dw_7_relu (ReLU)       (None, 14, 14, 512)       0

conv_pw_7 (Conv2D)          (None, 14, 14, 512)       262144

conv_pw_7_bn (BatchNormaliz  (None, 14, 14, 512)      2048
ation)

conv_pw_7_relu (ReLU)       (None, 14, 14, 512)       0

conv_dw_8 (DepthwiseConv2D)  (None, 14, 14, 512)      4608

conv_dw_8_bn (BatchNormaliz  (None, 14, 14, 512)      2048
ation)

conv_dw_8_relu (ReLU)       (None, 14, 14, 512)       0

conv_pw_8 (Conv2D)          (None, 14, 14, 512)       262144

conv_pw_8_bn (BatchNormaliz  (None, 14, 14, 512)      2048
ation)

conv_pw_8_relu (ReLU)       (None, 14, 14, 512)       0

conv_dw_9 (DepthwiseConv2D)  (None, 14, 14, 512)      4608

conv_dw_9_bn (BatchNormaliz  (None, 14, 14, 512)      2048
ation)

conv_dw_9_relu (ReLU)       (None, 14, 14, 512)       0

conv_pw_9 (Conv2D)          (None, 14, 14, 512)       262144

conv_pw_9_bn (BatchNormaliz  (None, 14, 14, 512)      2048
ation)

conv_pw_9_relu (ReLU)       (None, 14, 14, 512)       0

conv_dw_10 (DepthwiseConv2D  (None, 14, 14, 512)      4608
)

conv_dw_10_bn (BatchNormali  (None, 14, 14, 512)      2048
zation)

conv_dw_10_relu (ReLU)      (None, 14, 14, 512)       0

conv_pw_10 (Conv2D)         (None, 14, 14, 512)       262144

conv_pw_10_bn (BatchNormali  (None, 14, 14, 512)      2048
```

```
 zation)

 conv_pw_10_relu (ReLU)      (None, 14, 14, 512)       0

 conv_dw_11 (DepthwiseConv2D  (None, 14, 14, 512)      4608
 )

 conv_dw_11_bn (BatchNormali  (None, 14, 14, 512)      2048
 zation)

 conv_dw_11_relu (ReLU)      (None, 14, 14, 512)       0

 conv_pw_11 (Conv2D)         (None, 14, 14, 512)       262144

 conv_pw_11_bn (BatchNormali  (None, 14, 14, 512)      2048
 zation)

 conv_pw_11_relu (ReLU)      (None, 14, 14, 512)       0

 conv_pad_12 (ZeroPadding2D)  (None, 15, 15, 512)      0

 conv_dw_12 (DepthwiseConv2D  (None, 7, 7, 512)        4608
 )

 conv_dw_12_bn (BatchNormali  (None, 7, 7, 512)        2048
 zation)

 conv_dw_12_relu (ReLU)      (None, 7, 7, 512)         0

 conv_pw_12 (Conv2D)         (None, 7, 7, 1024)        524288

 conv_pw_12_bn (BatchNormali  (None, 7, 7, 1024)       4096
 zation)

 conv_pw_12_relu (ReLU)      (None, 7, 7, 1024)        0

 conv_dw_13 (DepthwiseConv2D  (None, 7, 7, 1024)       9216
 )

 conv_dw_13_bn (BatchNormali  (None, 7, 7, 1024)       4096
 zation)

 conv_dw_13_relu (ReLU)      (None, 7, 7, 1024)        0

 conv_pw_13 (Conv2D)         (None, 7, 7, 1024)        1048576

 conv_pw_13_bn (BatchNormali  (None, 7, 7, 1024)       4096
 zation)

 conv_pw_13_relu (ReLU)      (None, 7, 7, 1024)        0

 global_average_pooling2d (G  (None, 1, 1, 1024)       0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1, 1, 1024)        0

 conv_preds (Conv2D)         (None, 1, 1, 1000)        1025000

 reshape_2 (Reshape)         (None, 1000)              0

 predictions (Activation)    (None, 1000)              0

=================================================================
Total params: 4,253,864
Trainable params: 4,231,976
```

```
Non-trainable params: 21,888
_________________________________________________________________
```

In [4]:

```
def prepare_image(file):
    img_path = ''
    img = image.load_img(img_path + file, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array_expanded_dims = np.expand_dims(img_array, axis=0)
    return tf.keras.applications.mobilenet.preprocess_input(img_array_expanded_dims)
```

In [5]:

```
num_of_train_samples = 30999
num_of_valid_samples = 13325
num_of_test_samples = 110
```

In [6]:

```
#Show sample image
Image(filename='C:/Users/GyasiEmmanuelKwabena/Desktop/Dataset/train/Cb/Cb-N116.JPG', wi
```

Out[6]:

In [7]:

```
train_path = 'C:/Users/GyasiEmmanuelKwabena/Desktop/Dataset/train'
valid_path = 'C:/Users/GyasiEmmanuelKwabena/Desktop/Dataset/valid'

train_batches = ImageDataGenerator(preprocessing_function=tf.keras.applications.mobilen
    directory=train_path, target_size=(224,224), batch_size=22)
valid_batches = ImageDataGenerator(preprocessing_function=tf.keras.applications.mobilen
    directory=valid_path, target_size=(224,224), batch_size=22)

batchX, batchy = train_batches.next()
print('Batch shape=%s, min=%.3f, max=%.3f' % (batchX.shape, batchX.min(), batchX.max())
```

```
Found 30999 images belonging to 11 classes.
Found 13325 images belonging to 11 classes.
Batch shape=(22, 224, 224, 3), min=-1.000, max=1.000
```

In [8]:

```
# Modify the model: Mobilenet

base_model=MobileNet(weights='imagenet',include_top=False) #imports the mobilenet model

x=base_model.output
x=GlobalAveragePooling2D()(x)
x=Dense(300,activation='relu')(x) # 500 we add dense layers so that the model can learn
x=Dense(200,activation='relu')(x) # 100 dense layer 2
```

```
x=Dense(512,activation='relu')(x) # 512 dense layer 3
preds=Dense(11,activation='softmax')(x) #final layer with softmax activation
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [128, 1
60, 192, 224]. Weights for input shape (224, 224) will be loaded as the default.
```

In [9]:

```
for i,layer in enumerate(base_model.layers):
    print(i,layer.name)
```

```
0 input_2
1 conv1
2 conv1_bn
3 conv1_relu
4 conv_dw_1
5 conv_dw_1_bn
6 conv_dw_1_relu
7 conv_pw_1
8 conv_pw_1_bn
9 conv_pw_1_relu
10 conv_pad_2
11 conv_dw_2
12 conv_dw_2_bn
13 conv_dw_2_relu
14 conv_pw_2
15 conv_pw_2_bn
16 conv_pw_2_relu
17 conv_dw_3
18 conv_dw_3_bn
19 conv_dw_3_relu
20 conv_pw_3
21 conv_pw_3_bn
22 conv_pw_3_relu
23 conv_pad_4
24 conv_dw_4
25 conv_dw_4_bn
26 conv_dw_4_relu
27 conv_pw_4
28 conv_pw_4_bn
29 conv_pw_4_relu
30 conv_dw_5
31 conv_dw_5_bn
32 conv_dw_5_relu
33 conv_pw_5
34 conv_pw_5_bn
35 conv_pw_5_relu
36 conv_pad_6
37 conv_dw_6
38 conv_dw_6_bn
39 conv_dw_6_relu
40 conv_pw_6
41 conv_pw_6_bn
42 conv_pw_6_relu
43 conv_dw_7
44 conv_dw_7_bn
45 conv_dw_7_relu
46 conv_pw_7
47 conv_pw_7_bn
48 conv_pw_7_relu
49 conv_dw_8
50 conv_dw_8_bn
51 conv_dw_8_relu
52 conv_pw_8
53 conv_pw_8_bn
54 conv_pw_8_relu
55 conv_dw_9
```

```
56 conv_dw_9_bn
57 conv_dw_9_relu
58 conv_pw_9
59 conv_pw_9_bn
60 conv_pw_9_relu
61 conv_dw_10
62 conv_dw_10_bn
63 conv_dw_10_relu
64 conv_pw_10
65 conv_pw_10_bn
66 conv_pw_10_relu
67 conv_dw_11
68 conv_dw_11_bn
69 conv_dw_11_relu
70 conv_pw_11
71 conv_pw_11_bn
72 conv_pw_11_relu
73 conv_pad_12
74 conv_dw_12
75 conv_dw_12_bn
76 conv_dw_12_relu
77 conv_pw_12
78 conv_pw_12_bn
79 conv_pw_12_relu
80 conv_dw_13
81 conv_dw_13_bn
82 conv_dw_13_relu
83 conv_pw_13
84 conv_pw_13_bn
85 conv_pw_13_relu
```

In [10]:

```
#we want to set the first 20 layers of the network to be non-trainable
for layer in base_model.layers[:20]:
    layer.trainable=False
```

In [11]:

```
CloudMobiNet_model=Model(inputs=base_model.input,outputs=preds)
```

In [12]:

```
CloudMobiNet_model.summary()
```

```
Model: "model"

_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_2 (InputLayer)        [(None, None, None, 3)]   0         
                                                                 
 conv1 (Conv2D)              (None, None, None, 32)    864       
                                                                 
 conv1_bn (BatchNormalizatio  (None, None, None, 32)   128       
 n)                                                              
                                                                 
 conv1_relu (ReLU)           (None, None, None, 32)    0         
                                                                 
 conv_dw_1 (DepthwiseConv2D)  (None, None, None, 32)   288       
                                                                 
 conv_dw_1_bn (BatchNormaliz  (None, None, None, 32)   128       
 ation)                                                          
                                                                 
 conv_dw_1_relu (ReLU)       (None, None, None, 32)    0         
                                                                 
 conv_pw_1 (Conv2D)          (None, None, None, 64)    2048      
```

```
conv_pw_1_bn (BatchNormaliz  (None, None, None, 64)   256
ation)

conv_pw_1_relu (ReLU)       (None, None, None, 64)    0

conv_pad_2 (ZeroPadding2D)  (None, None, None, 64)    0

conv_dw_2 (DepthwiseConv2D)  (None, None, None, 64)   576

conv_dw_2_bn (BatchNormaliz  (None, None, None, 64)   256
ation)

conv_dw_2_relu (ReLU)       (None, None, None, 64)    0

conv_pw_2 (Conv2D)          (None, None, None, 128)   8192

conv_pw_2_bn (BatchNormaliz  (None, None, None, 128)  512
ation)

conv_pw_2_relu (ReLU)       (None, None, None, 128)   0

conv_dw_3 (DepthwiseConv2D)  (None, None, None, 128)  1152

conv_dw_3_bn (BatchNormaliz  (None, None, None, 128)  512
ation)

conv_dw_3_relu (ReLU)       (None, None, None, 128)   0

conv_pw_3 (Conv2D)          (None, None, None, 128)   16384

conv_pw_3_bn (BatchNormaliz  (None, None, None, 128)  512
ation)

conv_pw_3_relu (ReLU)       (None, None, None, 128)   0

conv_pad_4 (ZeroPadding2D)  (None, None, None, 128)   0

conv_dw_4 (DepthwiseConv2D)  (None, None, None, 128)  1152

conv_dw_4_bn (BatchNormaliz  (None, None, None, 128)  512
ation)

conv_dw_4_relu (ReLU)       (None, None, None, 128)   0

conv_pw_4 (Conv2D)          (None, None, None, 256)   32768

conv_pw_4_bn (BatchNormaliz  (None, None, None, 256)  1024
ation)

conv_pw_4_relu (ReLU)       (None, None, None, 256)   0

conv_dw_5 (DepthwiseConv2D)  (None, None, None, 256)  2304

conv_dw_5_bn (BatchNormaliz  (None, None, None, 256)  1024
ation)

conv_dw_5_relu (ReLU)       (None, None, None, 256)   0

conv_pw_5 (Conv2D)          (None, None, None, 256)   65536

conv_pw_5_bn (BatchNormaliz  (None, None, None, 256)  1024
ation)

conv_pw_5_relu (ReLU)       (None, None, None, 256)   0
```

```
 conv_pad_6 (ZeroPadding2D)  (None, None, None, 256)   0

 conv_dw_6 (DepthwiseConv2D)  (None, None, None, 256)  2304

 conv_dw_6_bn (BatchNormaliz  (None, None, None, 256)  1024
 ation)

 conv_dw_6_relu (ReLU)       (None, None, None, 256)   0

 conv_pw_6 (Conv2D)          (None, None, None, 512)   131072

 conv_pw_6_bn (BatchNormaliz  (None, None, None, 512)  2048
 ation)

 conv_pw_6_relu (ReLU)       (None, None, None, 512)   0

 conv_dw_7 (DepthwiseConv2D)  (None, None, None, 512)  4608

 conv_dw_7_bn (BatchNormaliz  (None, None, None, 512)  2048
 ation)

 conv_dw_7_relu (ReLU)       (None, None, None, 512)   0

 conv_pw_7 (Conv2D)          (None, None, None, 512)   262144

 conv_pw_7_bn (BatchNormaliz  (None, None, None, 512)  2048
 ation)

 conv_pw_7_relu (ReLU)       (None, None, None, 512)   0

 conv_dw_8 (DepthwiseConv2D)  (None, None, None, 512)  4608

 conv_dw_8_bn (BatchNormaliz  (None, None, None, 512)  2048
 ation)

 conv_dw_8_relu (ReLU)       (None, None, None, 512)   0

 conv_pw_8 (Conv2D)          (None, None, None, 512)   262144

 conv_pw_8_bn (BatchNormaliz  (None, None, None, 512)  2048
 ation)

 conv_pw_8_relu (ReLU)       (None, None, None, 512)   0

 conv_dw_9 (DepthwiseConv2D)  (None, None, None, 512)  4608

 conv_dw_9_bn (BatchNormaliz  (None, None, None, 512)  2048
 ation)

 conv_dw_9_relu (ReLU)       (None, None, None, 512)   0

 conv_pw_9 (Conv2D)          (None, None, None, 512)   262144

 conv_pw_9_bn (BatchNormaliz  (None, None, None, 512)  2048
 ation)

 conv_pw_9_relu (ReLU)       (None, None, None, 512)   0

 conv_dw_10 (DepthwiseConv2D  (None, None, None, 512)  4608
 )

 conv_dw_10_bn (BatchNormali  (None, None, None, 512)  2048
 zation)
```

```
 conv_dw_10_relu (ReLU)      (None, None, None, 512)   0

 conv_pw_10 (Conv2D)         (None, None, None, 512)   262144

 conv_pw_10_bn (BatchNormali  (None, None, None, 512)  2048
 zation)

 conv_pw_10_relu (ReLU)      (None, None, None, 512)   0

 conv_dw_11 (DepthwiseConv2D  (None, None, None, 512)  4608
 )

 conv_dw_11_bn (BatchNormali  (None, None, None, 512)  2048
 zation)

 conv_dw_11_relu (ReLU)      (None, None, None, 512)   0

 conv_pw_11 (Conv2D)         (None, None, None, 512)   262144

 conv_pw_11_bn (BatchNormali  (None, None, None, 512)  2048
 zation)

 conv_pw_11_relu (ReLU)      (None, None, None, 512)   0

 conv_pad_12 (ZeroPadding2D)  (None, None, None, 512)  0

 conv_dw_12 (DepthwiseConv2D  (None, None, None, 512)  4608
 )

 conv_dw_12_bn (BatchNormali  (None, None, None, 512)  2048
 zation)

 conv_dw_12_relu (ReLU)      (None, None, None, 512)   0

 conv_pw_12 (Conv2D)         (None, None, None, 1024)  524288

 conv_pw_12_bn (BatchNormali  (None, None, None, 1024)  4096
 zation)

 conv_pw_12_relu (ReLU)      (None, None, None, 1024)  0

 conv_dw_13 (DepthwiseConv2D  (None, None, None, 1024)  9216
 )

 conv_dw_13_bn (BatchNormali  (None, None, None, 1024)  4096
 zation)

 conv_dw_13_relu (ReLU)      (None, None, None, 1024)  0

 conv_pw_13 (Conv2D)         (None, None, None, 1024)  1048576

 conv_pw_13_bn (BatchNormali  (None, None, None, 1024)  4096
 zation)

 conv_pw_13_relu (ReLU)      (None, None, None, 1024)  0

 global_average_pooling2d_1   (None, 1024)             0
 (GlobalAveragePooling2D)

 dense (Dense)               (None, 300)               307500

 dense_1 (Dense)             (None, 200)               60200

 dense_2 (Dense)             (None, 512)               102912
```

```
 dense_3 (Dense)             (None, 11)                5643      
                                                                 
=================================================================
Total params: 3,705,119
Trainable params: 3,669,215
Non-trainable params: 35,904
_________________________________________________________________
```

In [13]:

```
train_datagen=ImageDataGenerator(preprocessing_function=preprocess_input) #included in

train_generator=train_datagen.flow_from_directory(train_path,
                                                 target_size=(224, 224),
                                                 color_mode='rgb',
                                                 batch_size=22,
                                                 class_mode='categorical',
                                                 shuffle=True)



test_datagen = ImageDataGenerator()


validation_generator = test_datagen.flow_from_directory(valid_path,
                                                        target_size=(224, 224),
                                                        color_mode='rgb',
                                                        batch_size=22,
                                                        class_mode='categorical',
                                                        shuffle=True)
```

```
Found 30999 images belonging to 11 classes.
Found 13325 images belonging to 11 classes.
```

In [14]:

```
#Images Classes with index
print(train_generator.class_indices)
```

```
{'Ac': 0, 'As': 1, 'Cb': 2, 'Cc': 3, 'Ci': 4, 'Cs': 5, 'Ct': 6, 'Cu': 7, 'Ns': 8, 'Sc': 
9, 'St': 10}
```

In [15]:

```
CloudMobiNet_model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_cros
```

In [16]:

```
start = datetime.datetime.now()

history = CloudMobiNet_model.fit(train_generator,
            steps_per_epoch=len(train_batches)//train_generator.batch_size,
            validation_data=valid_batches,
            validation_steps=len(valid_batches)//valid_batches.batch_size,
            epochs=130,
            verbose=2
            )

end= datetime.datetime.now()
elapsed= end-start
print ('Time: ', elapsed)
```

```
Epoch 1/130
64/64 - 132s - loss: 2.1058 - accuracy: 0.2927 - val_loss: 2.1467 - val_accuracy: 0.2458
- 132s/epoch - 2s/step
Epoch 2/130
64/64 - 131s - loss: 1.6764 - accuracy: 0.4382 - val_loss: 1.7775 - val_accuracy: 0.3973
```

```
- 131s/epoch - 2s/step
Epoch 3/130
64/64 - 132s - loss: 1.4733 - accuracy: 0.5028 - val_loss: 1.6209 - val_accuracy: 0.4512
- 132s/epoch - 2s/step
Epoch 4/130
64/64 - 132s - loss: 1.3195 - accuracy: 0.5455 - val_loss: 1.4848 - val_accuracy: 0.4933
- 132s/epoch - 2s/step
Epoch 5/130
64/64 - 132s - loss: 1.2639 - accuracy: 0.5675 - val_loss: 1.3493 - val_accuracy: 0.5556
- 132s/epoch - 2s/step
Epoch 6/130
64/64 - 131s - loss: 1.1417 - accuracy: 0.6250 - val_loss: 1.1267 - val_accuracy: 0.6111
- 131s/epoch - 2s/step
Epoch 7/130
64/64 - 131s - loss: 1.1474 - accuracy: 0.6143 - val_loss: 1.1028 - val_accuracy: 0.6313
- 131s/epoch - 2s/step
Epoch 8/130
64/64 - 132s - loss: 1.0466 - accuracy: 0.6435 - val_loss: 1.1998 - val_accuracy: 0.5909
- 132s/epoch - 2s/step
Epoch 9/130
64/64 - 131s - loss: 0.9849 - accuracy: 0.6697 - val_loss: 1.0519 - val_accuracy: 0.6347
- 131s/epoch - 2s/step
Epoch 10/130
64/64 - 131s - loss: 0.8747 - accuracy: 0.7038 - val_loss: 1.0734 - val_accuracy: 0.6347
- 131s/epoch - 2s/step
Epoch 11/130
64/64 - 131s - loss: 0.7763 - accuracy: 0.7422 - val_loss: 0.9012 - val_accuracy: 0.7003
- 131s/epoch - 2s/step
Epoch 12/130
64/64 - 133s - loss: 0.8291 - accuracy: 0.7195 - val_loss: 0.9216 - val_accuracy: 0.6953
- 133s/epoch - 2s/step
Epoch 13/130
64/64 - 133s - loss: 0.7676 - accuracy: 0.7464 - val_loss: 0.8668 - val_accuracy: 0.7104
- 133s/epoch - 2s/step
Epoch 14/130
64/64 - 132s - loss: 0.7233 - accuracy: 0.7443 - val_loss: 0.9766 - val_accuracy: 0.6633
- 132s/epoch - 2s/step
Epoch 15/130
64/64 - 132s - loss: 0.6970 - accuracy: 0.7628 - val_loss: 0.8618 - val_accuracy: 0.7172
- 132s/epoch - 2s/step
Epoch 16/130
64/64 - 132s - loss: 0.6439 - accuracy: 0.7841 - val_loss: 0.9038 - val_accuracy: 0.6852
- 132s/epoch - 2s/step
Epoch 17/130
64/64 - 131s - loss: 0.6509 - accuracy: 0.7791 - val_loss: 0.8481 - val_accuracy: 0.7357
- 131s/epoch - 2s/step
Epoch 18/130
64/64 - 132s - loss: 0.5377 - accuracy: 0.8175 - val_loss: 0.8421 - val_accuracy: 0.7323
- 132s/epoch - 2s/step
Epoch 19/130
64/64 - 131s - loss: 0.5051 - accuracy: 0.8317 - val_loss: 0.6238 - val_accuracy: 0.7963
- 131s/epoch - 2s/step
Epoch 20/130
64/64 - 131s - loss: 0.5084 - accuracy: 0.8295 - val_loss: 0.6728 - val_accuracy: 0.7912
- 131s/epoch - 2s/step
Epoch 21/130
64/64 - 133s - loss: 0.5101 - accuracy: 0.8402 - val_loss: 0.6787 - val_accuracy: 0.7593
- 133s/epoch - 2s/step
Epoch 22/130
64/64 - 131s - loss: 0.4774 - accuracy: 0.8395 - val_loss: 0.6839 - val_accuracy: 0.7677
- 131s/epoch - 2s/step
Epoch 23/130
64/64 - 132s - loss: 0.4374 - accuracy: 0.8473 - val_loss: 0.6310 - val_accuracy: 0.7946
- 132s/epoch - 2s/step
Epoch 24/130
```

```
64/64 - 132s - loss: 0.4446 - accuracy: 0.8409 - val_loss: 0.5570 - val_accuracy: 0.7946
- 132s/epoch - 2s/step
Epoch 25/130
64/64 - 132s - loss: 0.4577 - accuracy: 0.8452 - val_loss: 0.6505 - val_accuracy: 0.7710
- 132s/epoch - 2s/step
Epoch 26/130
64/64 - 132s - loss: 0.3961 - accuracy: 0.8693 - val_loss: 0.5918 - val_accuracy: 0.8013
- 132s/epoch - 2s/step
Epoch 27/130
64/64 - 131s - loss: 0.3938 - accuracy: 0.8771 - val_loss: 0.4794 - val_accuracy: 0.8266
- 131s/epoch - 2s/step
Epoch 28/130
64/64 - 132s - loss: 0.3552 - accuracy: 0.8786 - val_loss: 0.7028 - val_accuracy: 0.7795
- 132s/epoch - 2s/step
Epoch 29/130
64/64 - 132s - loss: 0.3552 - accuracy: 0.8885 - val_loss: 0.7102 - val_accuracy: 0.7542
- 132s/epoch - 2s/step
Epoch 30/130
64/64 - 131s - loss: 0.3284 - accuracy: 0.8949 - val_loss: 0.4370 - val_accuracy: 0.8653
- 131s/epoch - 2s/step
Epoch 31/130
64/64 - 131s - loss: 0.3365 - accuracy: 0.8857 - val_loss: 0.4079 - val_accuracy: 0.8519
- 131s/epoch - 2s/step
Epoch 32/130
64/64 - 131s - loss: 0.3000 - accuracy: 0.8942 - val_loss: 0.4432 - val_accuracy: 0.8519
- 131s/epoch - 2s/step
Epoch 33/130
64/64 - 131s - loss: 0.2965 - accuracy: 0.8977 - val_loss: 0.3968 - val_accuracy: 0.8805
- 131s/epoch - 2s/step
Epoch 34/130
64/64 - 132s - loss: 0.2725 - accuracy: 0.9155 - val_loss: 0.5398 - val_accuracy: 0.8165
- 132s/epoch - 2s/step
Epoch 35/130
64/64 - 132s - loss: 0.2664 - accuracy: 0.9119 - val_loss: 0.5580 - val_accuracy: 0.8131
- 132s/epoch - 2s/step
Epoch 36/130
64/64 - 131s - loss: 0.3085 - accuracy: 0.8970 - val_loss: 0.4771 - val_accuracy: 0.8316
- 131s/epoch - 2s/step
Epoch 37/130
64/64 - 132s - loss: 0.2787 - accuracy: 0.9091 - val_loss: 0.4895 - val_accuracy: 0.8468
- 132s/epoch - 2s/step
Epoch 38/130
64/64 - 132s - loss: 0.2336 - accuracy: 0.9240 - val_loss: 0.3722 - val_accuracy: 0.8620
- 132s/epoch - 2s/step
Epoch 39/130
64/64 - 132s - loss: 0.3211 - accuracy: 0.8906 - val_loss: 0.3512 - val_accuracy: 0.8855
- 132s/epoch - 2s/step
Epoch 40/130
64/64 - 131s - loss: 0.2284 - accuracy: 0.9190 - val_loss: 0.4868 - val_accuracy: 0.8283
- 131s/epoch - 2s/step
Epoch 41/130
64/64 - 132s - loss: 0.2619 - accuracy: 0.9148 - val_loss: 0.5344 - val_accuracy: 0.8367
- 132s/epoch - 2s/step
Epoch 42/130
64/64 - 131s - loss: 0.2074 - accuracy: 0.9318 - val_loss: 0.5965 - val_accuracy: 0.7912
- 131s/epoch - 2s/step
Epoch 43/130
64/64 - 132s - loss: 0.2407 - accuracy: 0.9276 - val_loss: 0.3261 - val_accuracy: 0.8973
- 132s/epoch - 2s/step
Epoch 44/130
64/64 - 132s - loss: 0.2090 - accuracy: 0.9339 - val_loss: 0.3557 - val_accuracy: 0.8939
- 132s/epoch - 2s/step
Epoch 45/130
64/64 - 132s - loss: 0.2254 - accuracy: 0.9290 - val_loss: 0.6574 - val_accuracy: 0.8300
- 132s/epoch - 2s/step
```

```
Epoch 46/130
64/64 - 132s - loss: 0.2261 - accuracy: 0.9283 - val_loss: 0.5331 - val_accuracy: 0.8384
- 132s/epoch - 2s/step
Epoch 47/130
64/64 - 132s - loss: 0.2229 - accuracy: 0.9261 - val_loss: 0.5187 - val_accuracy: 0.8401
- 132s/epoch - 2s/step
Epoch 48/130
64/64 - 132s - loss: 0.2136 - accuracy: 0.9382 - val_loss: 0.3531 - val_accuracy: 0.9007
- 132s/epoch - 2s/step
Epoch 49/130
64/64 - 131s - loss: 0.1893 - accuracy: 0.9382 - val_loss: 0.3899 - val_accuracy: 0.8788
- 131s/epoch - 2s/step
Epoch 50/130
64/64 - 131s - loss: 0.1976 - accuracy: 0.9425 - val_loss: 0.2834 - val_accuracy: 0.9057
- 131s/epoch - 2s/step
Epoch 51/130
64/64 - 132s - loss: 0.1799 - accuracy: 0.9347 - val_loss: 0.3807 - val_accuracy: 0.8788
- 132s/epoch - 2s/step
Epoch 52/130
64/64 - 132s - loss: 0.1759 - accuracy: 0.9432 - val_loss: 0.4007 - val_accuracy: 0.8704
- 132s/epoch - 2s/step
Epoch 53/130
64/64 - 132s - loss: 0.2271 - accuracy: 0.9297 - val_loss: 0.3641 - val_accuracy: 0.8822
- 132s/epoch - 2s/step
Epoch 54/130
64/64 - 131s - loss: 0.1756 - accuracy: 0.9453 - val_loss: 0.2993 - val_accuracy: 0.9007
- 131s/epoch - 2s/step
Epoch 55/130
64/64 - 130s - loss: 0.1854 - accuracy: 0.9438 - val_loss: 0.4212 - val_accuracy: 0.8636
- 130s/epoch - 2s/step
Epoch 56/130
64/64 - 131s - loss: 0.2278 - accuracy: 0.9219 - val_loss: 0.6634 - val_accuracy: 0.7896
- 131s/epoch - 2s/step
Epoch 57/130
64/64 - 132s - loss: 0.1832 - accuracy: 0.9453 - val_loss: 0.3681 - val_accuracy: 0.8788
- 132s/epoch - 2s/step
Epoch 58/130
64/64 - 131s - loss: 0.1762 - accuracy: 0.9425 - val_loss: 0.4339 - val_accuracy: 0.8653
- 131s/epoch - 2s/step
Epoch 59/130
64/64 - 131s - loss: 0.2045 - accuracy: 0.9325 - val_loss: 0.3876 - val_accuracy: 0.8737
- 131s/epoch - 2s/step
Epoch 60/130
64/64 - 131s - loss: 0.1759 - accuracy: 0.9368 - val_loss: 0.3223 - val_accuracy: 0.8939
- 131s/epoch - 2s/step
Epoch 61/130
64/64 - 132s - loss: 0.1421 - accuracy: 0.9538 - val_loss: 0.3640 - val_accuracy: 0.8973
- 132s/epoch - 2s/step
Epoch 62/130
64/64 - 132s - loss: 0.1844 - accuracy: 0.9396 - val_loss: 0.4474 - val_accuracy: 0.8687
- 132s/epoch - 2s/step
Epoch 63/130
64/64 - 131s - loss: 0.1366 - accuracy: 0.9531 - val_loss: 0.2716 - val_accuracy: 0.9091
- 131s/epoch - 2s/step
Epoch 64/130
64/64 - 132s - loss: 0.1678 - accuracy: 0.9510 - val_loss: 0.3233 - val_accuracy: 0.8990
- 132s/epoch - 2s/step
Epoch 65/130
64/64 - 131s - loss: 0.1508 - accuracy: 0.9496 - val_loss: 0.3612 - val_accuracy: 0.8771
- 131s/epoch - 2s/step
Epoch 66/130
64/64 - 131s - loss: 0.1427 - accuracy: 0.9567 - val_loss: 0.2512 - val_accuracy: 0.9141
- 131s/epoch - 2s/step
Epoch 67/130
64/64 - 131s - loss: 0.1596 - accuracy: 0.9446 - val_loss: 0.2841 - val_accuracy: 0.9057
```

```
- 131s/epoch - 2s/step
Epoch 68/130
64/64 - 132s - loss: 0.1521 - accuracy: 0.9560 - val_loss: 0.2807 - val_accuracy: 0.9040
- 132s/epoch - 2s/step
Epoch 69/130
64/64 - 131s - loss: 0.1526 - accuracy: 0.9460 - val_loss: 0.2871 - val_accuracy: 0.9074
- 131s/epoch - 2s/step
Epoch 70/130
64/64 - 129s - loss: 0.1524 - accuracy: 0.9510 - val_loss: 1.2978 - val_accuracy: 0.6380
- 129s/epoch - 2s/step
Epoch 71/130
64/64 - 132s - loss: 0.2141 - accuracy: 0.9290 - val_loss: 0.3641 - val_accuracy: 0.8939
- 132s/epoch - 2s/step
Epoch 72/130
64/64 - 132s - loss: 0.1526 - accuracy: 0.9489 - val_loss: 0.3875 - val_accuracy: 0.8822
- 132s/epoch - 2s/step
Epoch 73/130
64/64 - 131s - loss: 0.1503 - accuracy: 0.9531 - val_loss: 0.3018 - val_accuracy: 0.9125
- 131s/epoch - 2s/step
Epoch 74/130
64/64 - 131s - loss: 0.1624 - accuracy: 0.9496 - val_loss: 0.3674 - val_accuracy: 0.8889
- 131s/epoch - 2s/step
Epoch 75/130
64/64 - 131s - loss: 0.1621 - accuracy: 0.9425 - val_loss: 0.3774 - val_accuracy: 0.9024
- 131s/epoch - 2s/step
Epoch 76/130
64/64 - 132s - loss: 0.1433 - accuracy: 0.9517 - val_loss: 0.3527 - val_accuracy: 0.8805
- 132s/epoch - 2s/step
Epoch 77/130
64/64 - 131s - loss: 0.1442 - accuracy: 0.9482 - val_loss: 0.4003 - val_accuracy: 0.8788
- 131s/epoch - 2s/step
Epoch 78/130
64/64 - 132s - loss: 0.1334 - accuracy: 0.9588 - val_loss: 0.3178 - val_accuracy: 0.9007
- 132s/epoch - 2s/step
Epoch 79/130
64/64 - 131s - loss: 0.1120 - accuracy: 0.9595 - val_loss: 0.3861 - val_accuracy: 0.8973
- 131s/epoch - 2s/step
Epoch 80/130
64/64 - 132s - loss: 0.1507 - accuracy: 0.9538 - val_loss: 0.2691 - val_accuracy: 0.9226
- 132s/epoch - 2s/step
Epoch 81/130
64/64 - 131s - loss: 0.1349 - accuracy: 0.9517 - val_loss: 0.3382 - val_accuracy: 0.8939
- 131s/epoch - 2s/step
Epoch 82/130
64/64 - 132s - loss: 0.1417 - accuracy: 0.9560 - val_loss: 0.2655 - val_accuracy: 0.9158
- 132s/epoch - 2s/step
Epoch 83/130
64/64 - 131s - loss: 0.1300 - accuracy: 0.9503 - val_loss: 0.3167 - val_accuracy: 0.9074
- 131s/epoch - 2s/step
Epoch 84/130
64/64 - 131s - loss: 0.1265 - accuracy: 0.9624 - val_loss: 0.4131 - val_accuracy: 0.8956
- 131s/epoch - 2s/step
Epoch 85/130
64/64 - 133s - loss: 0.1410 - accuracy: 0.9553 - val_loss: 0.2897 - val_accuracy: 0.9192
- 133s/epoch - 2s/step
Epoch 86/130
64/64 - 130s - loss: 0.1963 - accuracy: 0.9438 - val_loss: 0.4191 - val_accuracy: 0.8636
- 130s/epoch - 2s/step
Epoch 87/130
64/64 - 132s - loss: 0.1323 - accuracy: 0.9574 - val_loss: 0.2988 - val_accuracy: 0.9226
- 132s/epoch - 2s/step
Epoch 88/130
64/64 - 132s - loss: 0.1248 - accuracy: 0.9631 - val_loss: 0.3291 - val_accuracy: 0.9108
- 132s/epoch - 2s/step
Epoch 89/130
```

```
64/64 - 130s - loss: 0.2033 - accuracy: 0.9402 - val_loss: 0.3383 - val_accuracy: 0.8973
- 130s/epoch - 2s/step
Epoch 90/130
64/64 - 132s - loss: 0.1224 - accuracy: 0.9588 - val_loss: 0.3174 - val_accuracy: 0.9074
- 132s/epoch - 2s/step
Epoch 91/130
64/64 - 131s - loss: 0.1206 - accuracy: 0.9666 - val_loss: 0.3652 - val_accuracy: 0.8855
- 131s/epoch - 2s/step
Epoch 92/130
64/64 - 131s - loss: 0.1169 - accuracy: 0.9616 - val_loss: 0.3052 - val_accuracy: 0.9108
- 131s/epoch - 2s/step
Epoch 93/130
64/64 - 132s - loss: 0.0987 - accuracy: 0.9673 - val_loss: 0.3171 - val_accuracy: 0.9175
- 132s/epoch - 2s/step
Epoch 94/130
64/64 - 131s - loss: 0.1456 - accuracy: 0.9560 - val_loss: 0.2751 - val_accuracy: 0.9040
- 131s/epoch - 2s/step
Epoch 95/130
64/64 - 132s - loss: 0.1102 - accuracy: 0.9645 - val_loss: 0.3257 - val_accuracy: 0.9091
- 132s/epoch - 2s/step
Epoch 96/130
64/64 - 132s - loss: 0.0920 - accuracy: 0.9723 - val_loss: 0.4270 - val_accuracy: 0.8906
- 132s/epoch - 2s/step
Epoch 97/130
64/64 - 131s - loss: 0.0995 - accuracy: 0.9645 - val_loss: 0.2515 - val_accuracy: 0.9242
- 131s/epoch - 2s/step
Epoch 98/130
64/64 - 132s - loss: 0.1240 - accuracy: 0.9609 - val_loss: 0.2359 - val_accuracy: 0.9242
- 132s/epoch - 2s/step
Epoch 99/130
64/64 - 132s - loss: 0.1062 - accuracy: 0.9652 - val_loss: 0.2522 - val_accuracy: 0.9226
- 132s/epoch - 2s/step
Epoch 100/130
64/64 - 131s - loss: 0.1205 - accuracy: 0.9638 - val_loss: 0.2524 - val_accuracy: 0.9276
- 131s/epoch - 2s/step
Epoch 101/130
64/64 - 132s - loss: 0.1409 - accuracy: 0.9616 - val_loss: 0.3322 - val_accuracy: 0.8973
- 132s/epoch - 2s/step
Epoch 102/130
64/64 - 132s - loss: 0.0887 - accuracy: 0.9737 - val_loss: 0.2346 - val_accuracy: 0.9242
- 132s/epoch - 2s/step
Epoch 103/130
64/64 - 132s - loss: 0.0937 - accuracy: 0.9716 - val_loss: 0.2509 - val_accuracy: 0.9276
- 132s/epoch - 2s/step
Epoch 104/130
64/64 - 132s - loss: 0.1213 - accuracy: 0.9609 - val_loss: 0.3402 - val_accuracy: 0.8990
- 132s/epoch - 2s/step
Epoch 105/130
64/64 - 132s - loss: 0.1268 - accuracy: 0.9609 - val_loss: 0.3253 - val_accuracy: 0.8990
- 132s/epoch - 2s/step
Epoch 106/130
64/64 - 131s - loss: 0.1166 - accuracy: 0.9631 - val_loss: 0.2930 - val_accuracy: 0.9040
- 131s/epoch - 2s/step
Epoch 107/130
64/64 - 131s - loss: 0.1152 - accuracy: 0.9624 - val_loss: 0.4834 - val_accuracy: 0.8704
- 131s/epoch - 2s/step
Epoch 108/130
64/64 - 132s - loss: 0.1341 - accuracy: 0.9553 - val_loss: 0.3910 - val_accuracy: 0.8737
- 132s/epoch - 2s/step
Epoch 109/130
64/64 - 132s - loss: 0.1029 - accuracy: 0.9638 - val_loss: 0.4579 - val_accuracy: 0.8687
- 132s/epoch - 2s/step
Epoch 110/130
64/64 - 131s - loss: 0.1292 - accuracy: 0.9638 - val_loss: 0.3574 - val_accuracy: 0.9007
- 131s/epoch - 2s/step
```

```
Epoch 111/130
64/64 - 129s - loss: 0.1178 - accuracy: 0.9603 - val_loss: 0.5041 - val_accuracy: 0.8569
- 129s/epoch - 2s/step
Epoch 112/130
64/64 - 132s - loss: 0.1224 - accuracy: 0.9595 - val_loss: 0.2953 - val_accuracy: 0.9158
- 132s/epoch - 2s/step
Epoch 113/130
64/64 - 132s - loss: 0.1012 - accuracy: 0.9652 - val_loss: 0.4072 - val_accuracy: 0.8704
- 132s/epoch - 2s/step
Epoch 114/130
64/64 - 131s - loss: 0.0997 - accuracy: 0.9624 - val_loss: 0.3128 - val_accuracy: 0.9209
- 131s/epoch - 2s/step
Epoch 115/130
64/64 - 132s - loss: 0.1507 - accuracy: 0.9482 - val_loss: 0.2503 - val_accuracy: 0.9226
- 132s/epoch - 2s/step
Epoch 116/130
64/64 - 131s - loss: 0.0997 - accuracy: 0.9680 - val_loss: 0.2788 - val_accuracy: 0.9057
- 131s/epoch - 2s/step
Epoch 117/130
64/64 - 135s - loss: 0.1060 - accuracy: 0.9624 - val_loss: 0.3079 - val_accuracy: 0.9057
- 135s/epoch - 2s/step
Epoch 118/130
64/64 - 131s - loss: 0.1167 - accuracy: 0.9595 - val_loss: 0.4111 - val_accuracy: 0.8721
- 131s/epoch - 2s/step
Epoch 119/130
64/64 - 132s - loss: 0.0861 - accuracy: 0.9695 - val_loss: 0.2236 - val_accuracy: 0.9192
- 132s/epoch - 2s/step
Epoch 120/130
64/64 - 131s - loss: 0.0919 - accuracy: 0.9688 - val_loss: 0.2695 - val_accuracy: 0.9226
- 131s/epoch - 2s/step
Epoch 121/130
64/64 - 132s - loss: 0.0946 - accuracy: 0.9723 - val_loss: 0.3165 - val_accuracy: 0.9209
- 132s/epoch - 2s/step
Epoch 122/130
64/64 - 132s - loss: 0.0908 - accuracy: 0.9666 - val_loss: 0.3157 - val_accuracy: 0.9057
- 132s/epoch - 2s/step
Epoch 123/130
64/64 - 130s - loss: 0.1439 - accuracy: 0.9524 - val_loss: 0.5036 - val_accuracy: 0.8552
- 130s/epoch - 2s/step
Epoch 124/130
64/64 - 131s - loss: 0.0957 - accuracy: 0.9702 - val_loss: 0.3756 - val_accuracy: 0.9007
- 131s/epoch - 2s/step
Epoch 125/130
64/64 - 133s - loss: 0.1203 - accuracy: 0.9673 - val_loss: 0.3178 - val_accuracy: 0.9040
- 133s/epoch - 2s/step
Epoch 126/130
64/64 - 133s - loss: 0.0956 - accuracy: 0.9688 - val_loss: 0.2325 - val_accuracy: 0.9327
- 133s/epoch - 2s/step
Epoch 127/130
64/64 - 132s - loss: 0.0888 - accuracy: 0.9744 - val_loss: 0.3974 - val_accuracy: 0.8923
- 132s/epoch - 2s/step
Epoch 128/130
64/64 - 132s - loss: 0.1059 - accuracy: 0.9645 - val_loss: 0.3708 - val_accuracy: 0.8990
- 132s/epoch - 2s/step
Epoch 129/130
64/64 - 132s - loss: 0.1028 - accuracy: 0.9680 - val_loss: 0.3336 - val_accuracy: 0.9074
- 132s/epoch - 2s/step
Epoch 130/130
64/64 - 132s - loss: 0.1001 - accuracy: 0.9645 - val_loss: 0.3006 - val_accuracy: 0.9108
- 132s/epoch - 2s/step
Time:  4:45:11.459956
```

In [17]:

```
score = CloudMobiNet_model.evaluate(train_generator, verbose=0)
print("Accuracy: %.2f%%" % (score[1]*100))
```

```
Accuracy: 97.45%
```

In [18]:
```
CloudMobiNet_model.save('Cloud_MOBINET.h5')
```

In [19]:
```
CloudMobiNet_model.save('Cloud_MOBINET.hdf5')
```

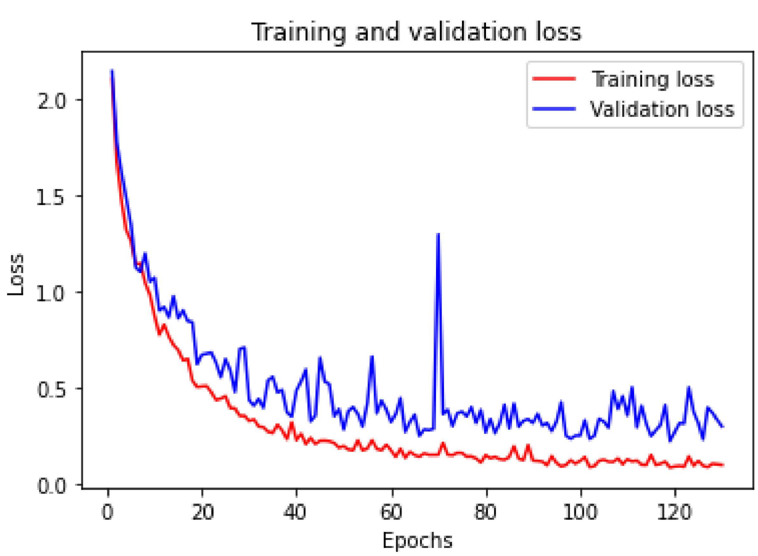
In [20]:
```
CloudMobiNet_model.metrics_names
score
```

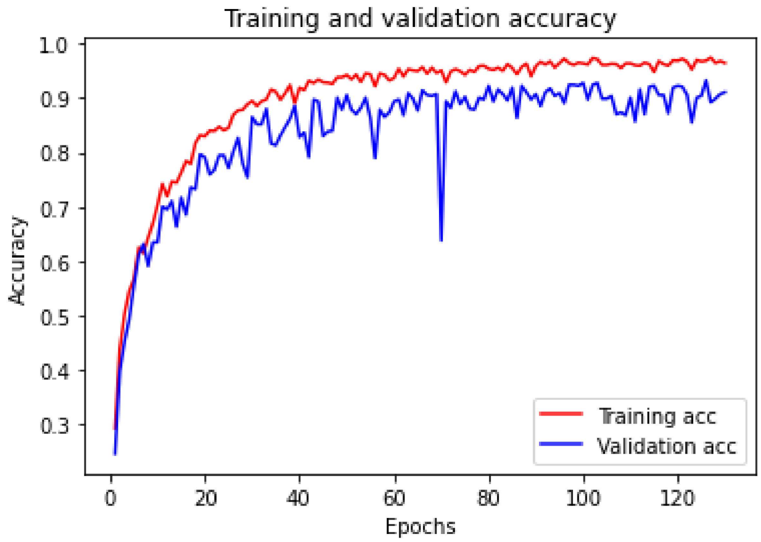
Out[20]:
```
[0.07623911648988724, 0.974483072757721]
```

In [21]:
```
score
```

Out[21]:
```
[0.07623911648988724, 0.974483072757721]
```

In [22]:
```
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

acc =history.history['accuracy']
val_acc = history.history['val_accuracy']
plt.plot(epochs, acc, 'r', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```





In [23]:
```
test_path = 'C:/Users/GyasiEmmanuelKwabena/Desktop/Dataset/test'
```

In [24]:
```
test_batches = ImageDataGenerator(preprocessing_function=tf.keras.applications.mobilene
    directory=test_path, target_size=(224,224), batch_size=110, shuffle=False)
```

```
Found 110 images belonging to 11 classes.
```

In [25]:
```
class_labels =["Ac","As","Cb", "Cc", "Ci", "Cs", "Ct", "Cu", "Ns", "Sc", "St" ]
from sklearn.metrics import classification_report
test_labels = test_batches.classes
predictions = CloudMobiNet_model.predict(test_batches, steps=len(test_batches), verbose
y_pred = np.argmax (predictions, axis= -1)
print(classification_report(test_labels, y_pred, target_names=class_labels))
```

```
              precision    recall  f1-score   support

          Ac       1.00      1.00      1.00        10
          As       1.00      1.00      1.00        10
          Cb       1.00      0.90      0.95        10
          Cc       1.00      0.90      0.95        10
          Ci       0.91      1.00      0.95        10
          Cs       0.91      1.00      0.95        10
          Ct       1.00      1.00      1.00        10
          Cu       1.00      1.00      1.00        10
          Ns       1.00      0.80      0.89        10
          Sc       1.00      1.00      1.00        10
          St       0.83      1.00      0.91        10

    accuracy                           0.96       110
   macro avg       0.97      0.96      0.96       110
weighted avg       0.97      0.96      0.96       110
```

In [26]:
```
test_labels = test_batches.classes
```

In [27]:
```
predictions = CloudMobiNet_model.predict(x=test_batches, steps=len(test_batches), verbo
```

In [28]:

```
cm = confusion_matrix(y_true=test_batches.classes, y_pred= np.argmax(predictions,axis=1
```

In [29]:

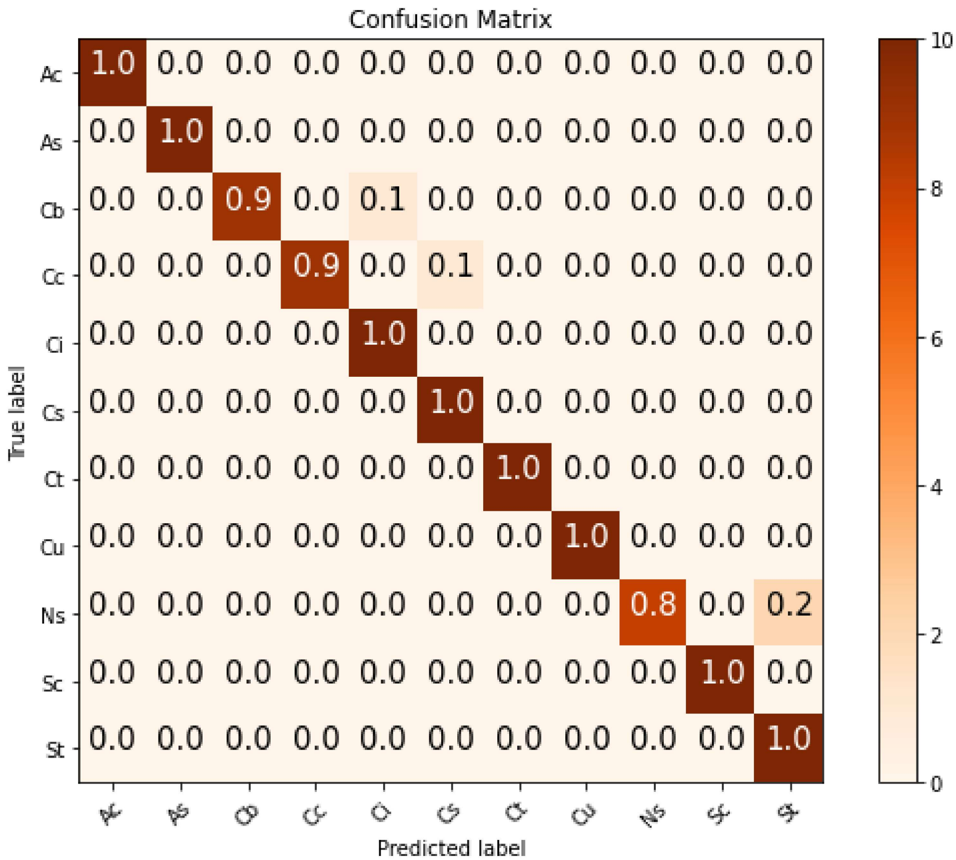
```
#Plot the confusion matrix. Set Normalize = True/False


def plot_confusion_matrix(cm, classes, normalize=True, title='Confusion matrix', cmap=p

    """

    This function prints and plots the confusion matrix.

    Normalization can be applied by setting `normalize=True`.

    """

    plt.figure(figsize=(8,6))


    plt.imshow(cm, interpolation='nearest', cmap=cmap)

    plt.title(title)

    plt.colorbar()


    tick_marks = np.arange(len(classes))

    plt.xticks(tick_marks, classes, rotation=45)

    plt.yticks(tick_marks, classes)



    if normalize:

        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        cm = np.around(cm, decimals=2)
        cm[np.isnan(cm)] = 0.0

        print("Normalized confusion matrix")

    else:

        print('Confusion matrix, without normalization')

    thresh = cm.max() / 2.

    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

        plt.text(j, i, cm[i, j],

                 horizontalalignment="center",
                 fontsize="15",

                 color="white" if cm[i, j] > thresh else "black")
```

```
    plt.tight_layout()

    plt.ylabel('True label')

    plt.xlabel('Predicted label')
```

In [30]:

```
cm_plot_labels = ["Ac", "As", "Cb", "Cc", "Ci", "Cs", "Ct", "Cu", "Ns", "Sc", "St"]
plot_confusion_matrix(cm=cm, classes=cm_plot_labels, title='Confusion Matrix')
```

```
Normalized confusion matrix
```



In [31]:

```
testX, testy = test_batches.next()
print('test shape=%s, min=%.3f, max=%.3f' % (testX.shape, testX.min(), testX.max()))
```

```
test shape=(110, 224, 224, 3), min=-1.000, max=1.000
```

In [32]:

```
#making a prediction about our test data

#checking the prediction shape
predictions.shape
```

Out[32]:

```
(110, 11)
```

In [33]:

```
#checking the batch shape
testX.shape
```

Out[33]:

```
(110, 224, 224, 3)
```

In [34]:
```
predictions= CloudMobiNet_model.predict(testX)
```

```
4/4 [==============================] - 3s 622ms/step
```

In [35]:
```
test_loss, test_accuracy = CloudMobiNet_model.evaluate(testX, testy)
```

```
4/4 [==============================] - 2s 548ms/step - loss: 0.1594 - accuracy: 0.9636
```

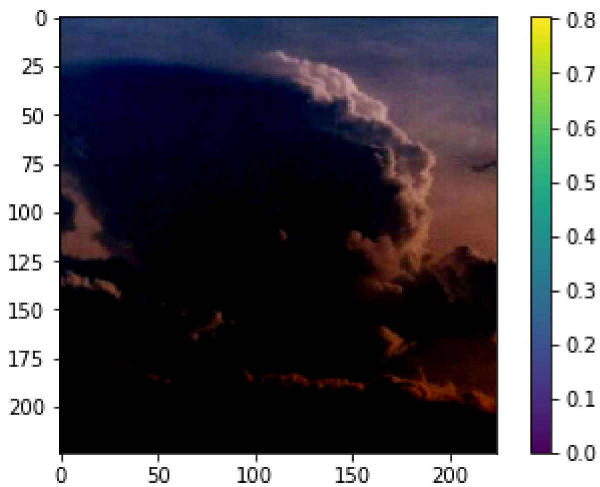
In [36]:
```
predictions[28]
```

Out[36]:
```
array([8.1877010e-07, 2.3667071e-05, 3.0261282e-02, 1.4088984e-04,
       9.5752704e-01, 5.6759247e-05, 2.0696471e-06, 6.3395048e-03,
       3.7166319e-04, 4.5917022e-05, 5.2305181e-03], dtype=float32)
```

In [37]:
```
import matplotlib.image as mpimg
```

In [38]:
```
plt.figure()
plt.imshow(testX[28])
plt.colorbar()
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```



In [39]:
```
class_names = ["Altocumulus", "Altostratus", "Cumulonimbus", "Cirrocumulus", "Cirrus",
```

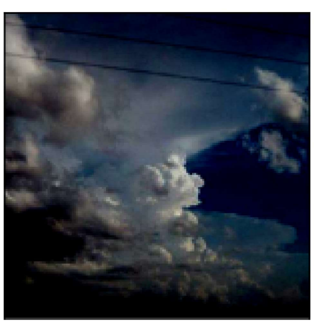
In [40]:
```
# function to plot an image
def plot_image(i, predictions_array, true_label, img):
 predictions_array, true_label, img=predictions_array, true_label[i], img[i]
 plt.grid(False)
 plt.xticks([])
 plt.yticks([])

 plt.imshow(img, cmap=plt.cm.gray)

 predicted_label = np.argmax(predictions_array)
 if predicted_label==true_label:
    color='green'
```

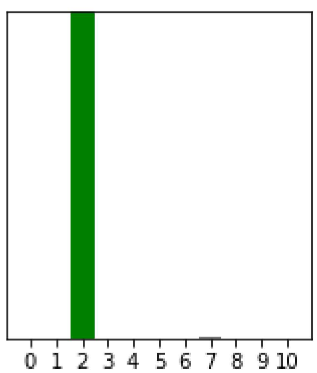
```
        plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                          100*np.max(predictions_array),
                                          class_names[true_label]),
                                          color=color)
    else:
        color='red'
        plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                          100*np.max(predictions_array),
                                          class_names[true_label]),
                                          color=color)

#functions to create bar plot of the predictions
def plot_value_array(i, predictions_array, true_label):
    predictions_array, true_label = predictions_array, true_label[i]
    plt.grid(False)
    plt.xticks(range(11))
    plt.yticks([])
    thisplot = plt.bar(range(11), predictions_array, color="#777777")
    plt.ylim([0, 1])
    predicted_label=np.argmax(predictions_array)

    thisplot[predicted_label].set_color('red')
    thisplot[true_label].set_color('green')
```

In [41]:

```
i=27
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, testX)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i], test_labels)
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```


Cumulonimbus 100% (Cumulonimbus)



In [42]:

```
predictions[35]
```

Out[42]:

```
array([5.3586769e-03, 8.9163244e-02, 5.3298902e-07, 8.9823407e-01,
       6.7059938e-03, 2.2159831e-04, 3.5701366e-06, 1.0728238e-06,
       3.7971606e-07, 2.4805779e-05, 2.8608745e-04], dtype=float32)
```

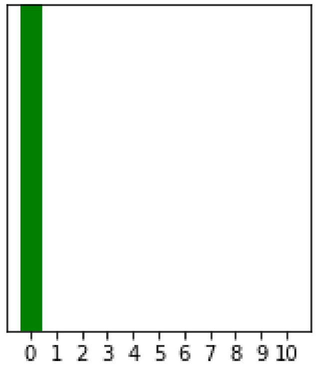
In [43]:

```
for i in range(11):
    plt.figure(figsize=(6,3))
    plt.subplot(1,2,1)
    plot_image(i, predictions[i], test_labels, testX)
```

```
    plt.subplot(1,2,2)
    plot_value_array(i, predictions[i], test_labels)
    plt.show()
```

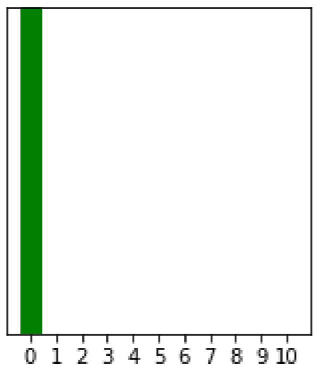
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

Altocumulus 100% (Altocumulus)

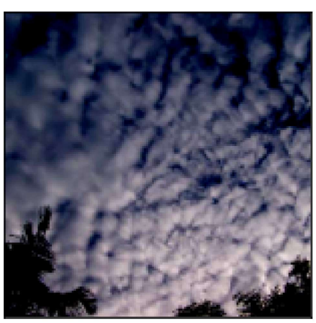



```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

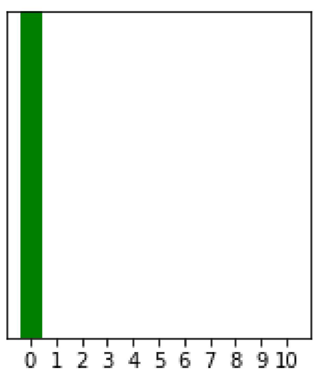
Altocumulus 100% (Altocumulus)




```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

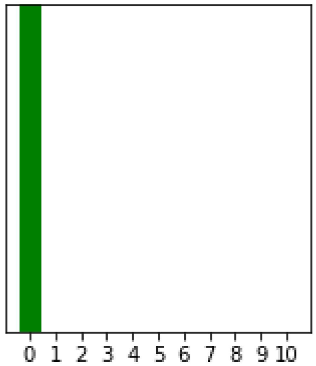


Altocumulus 100% (Altocumulus)

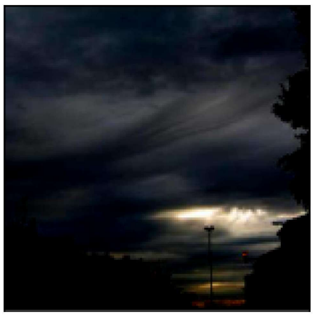



```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

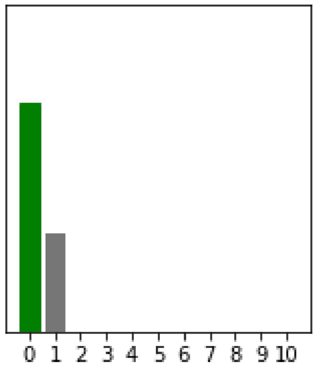
Altocumulus 100% (Altocumulus)




```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

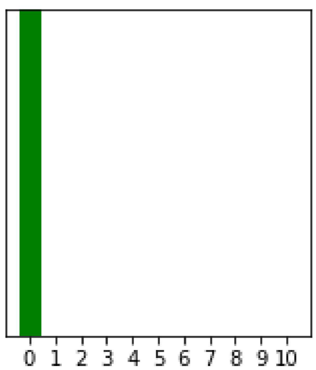


Altocumulus 70% (Altocumulus)

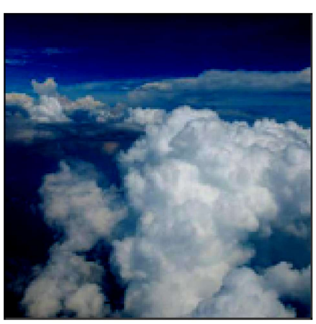



```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

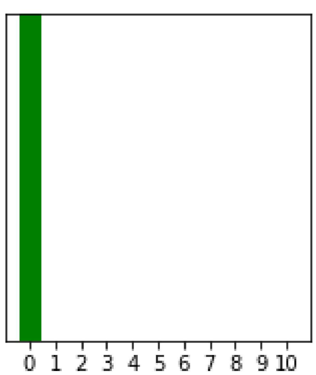
Altocumulus 100% (Altocumulus)



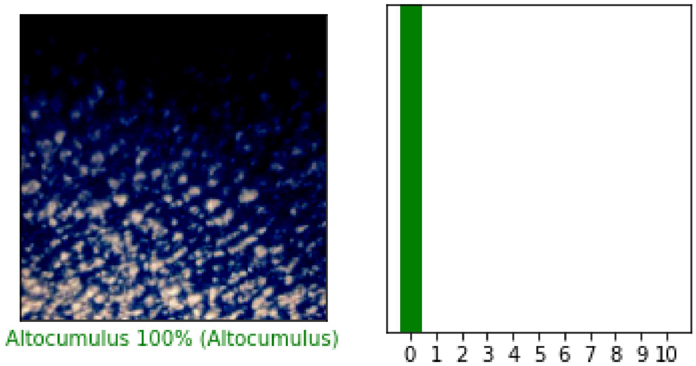

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```



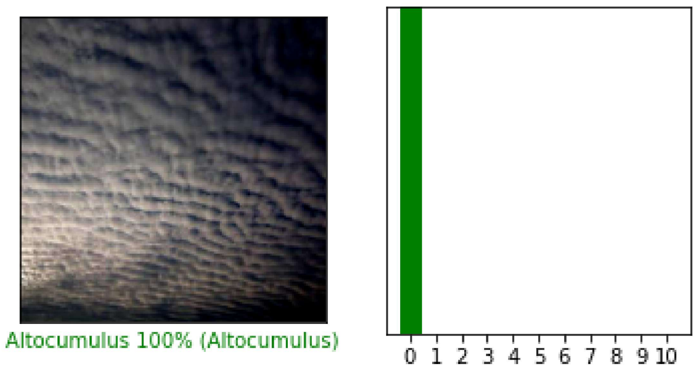
Altocumulus 100% (Altocumulus)



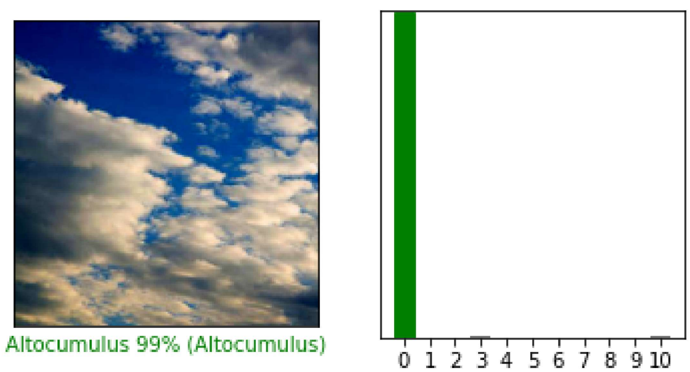

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```



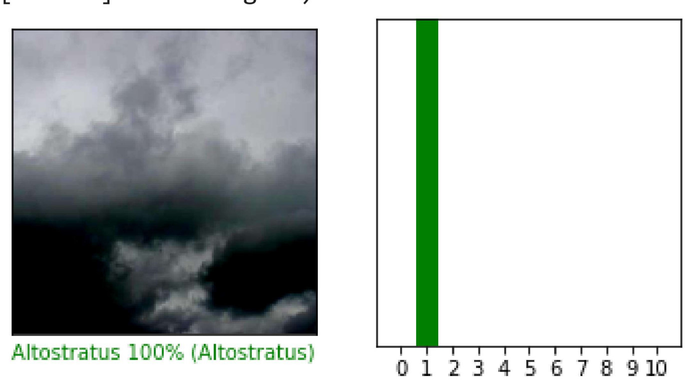

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```




```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```




```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

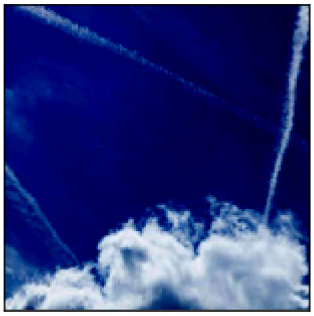


In [44]:

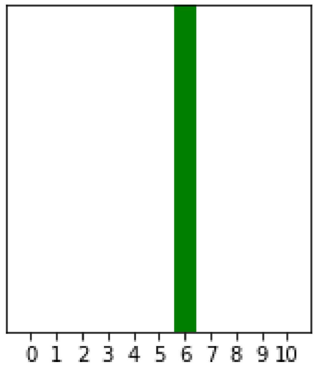
```
i=65
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
```

```
    plot_image(i, predictions[i], test_labels, testX)
    plt.subplot(1,2,2)
    plot_value_array(i, predictions[i], test_labels)
    plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```



Contrails 100% (Contrails)



In [45]:

```
predicted_classes = CloudMobiNet_model.predict(x=test_batches, steps=len(test_batches),
predicted_classes = np.argmax(np.round(predicted_classes),axis=1)
predicted_classes.shape, y_pred.shape
#print(type(predictions), predictions.shape)
```

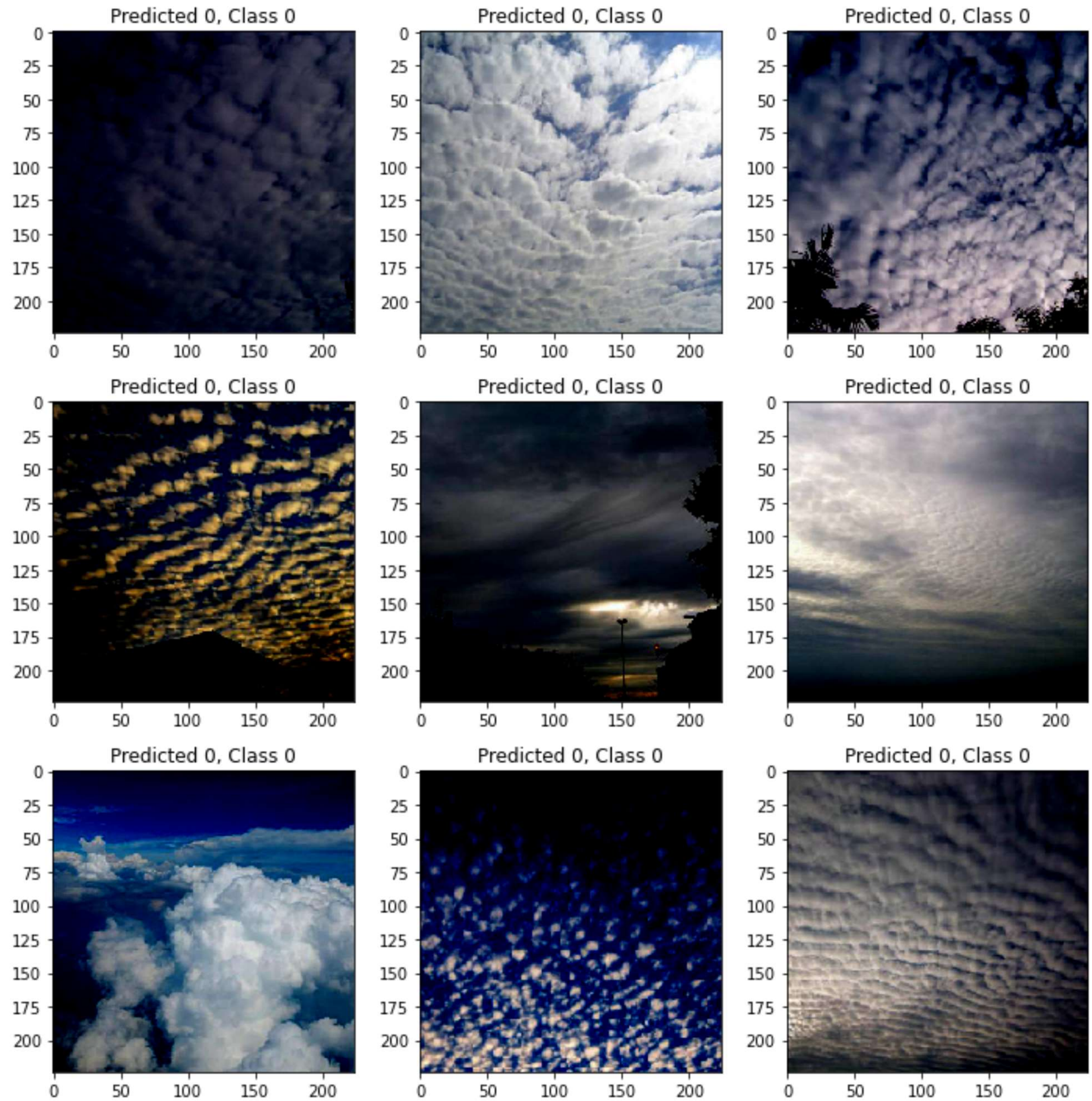
Out[45]:

```
((110,), (110,))
```

In [46]:

```
plt.figure(figsize=(10, 10))
correct = np.where(predicted_classes==y_pred)[0]
print (("Found %d correct labels") % len(correct))
for i, correct in enumerate(correct[:9]):
    plt.subplot(3,3,i+1)
    plt.imshow(testX[correct], cmap='gray', interpolation='none')
    plt.title("Predicted {}, Class {}".format(predicted_classes[correct], y_pred[correc
    plt.tight_layout()
```

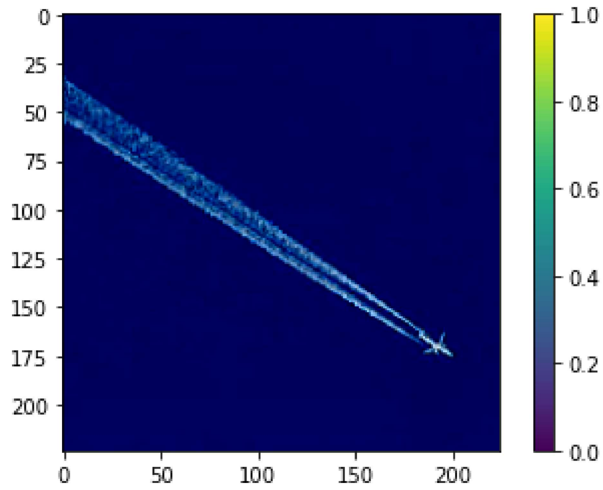
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Found 110 correct labels
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```



In [47]:

```
plt.figure()
plt.imshow(testX[60])
plt.colorbar()
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

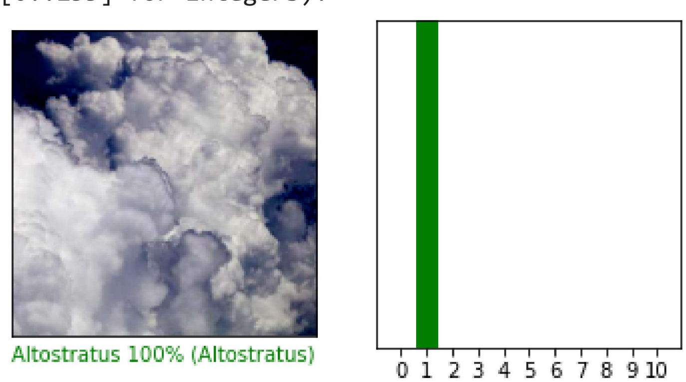


In [48]:
```
#printing the first element from predicted data
pred=CloudMobiNet_model.predict(test_batches)
print(pred[60])
#printing the index of
print('Index:',np.argmax(pred[60]))
```

```
1/1 [==============================] - 3s 3s/step
[1.6036435e-12 2.3826804e-13 4.7321495e-13 2.5269540e-09 3.1627598e-12
 8.7350249e-12 1.0000000e+00 7.8760999e-12 1.7956898e-13 1.5294960e-11
 4.9321798e-08]
Index: 6
```

In [49]:
```
i=16
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, testX)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i], test_labels)
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

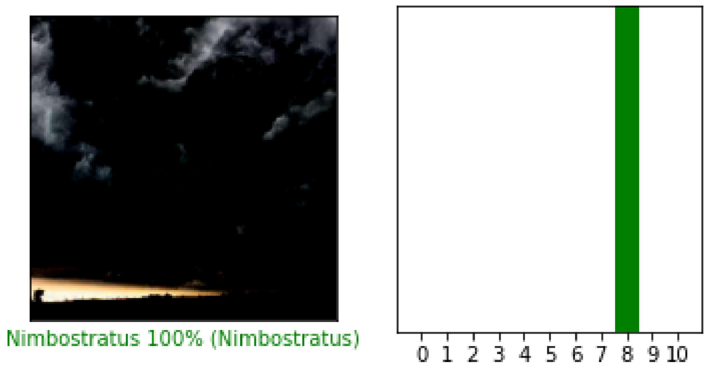


In [50]:
```
predictions[16]
```

Out[50]:
```
array([2.3733778e-04, 9.9904412e-01, 1.9174904e-04, 6.2474978e-06,
       2.0500877e-09, 7.5997235e-07, 6.6262295e-08, 1.9544762e-04,
       1.1830035e-06, 8.7253291e-08, 3.2299891e-04], dtype=float32)
```

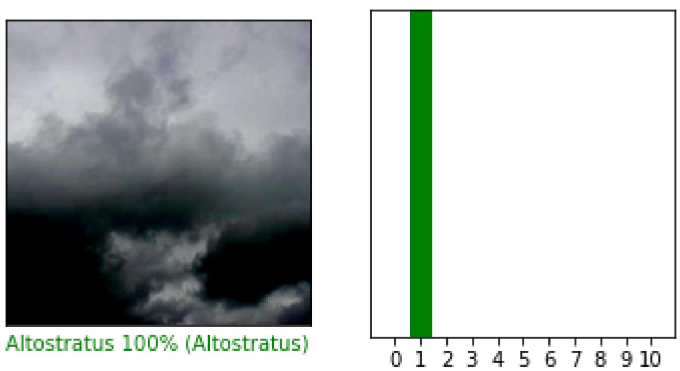
In [51]:
```
i=85
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, testX)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i], test_labels)
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```



In [52]:
```
i=10
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, testX)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i], test_labels)
plt.show()
```

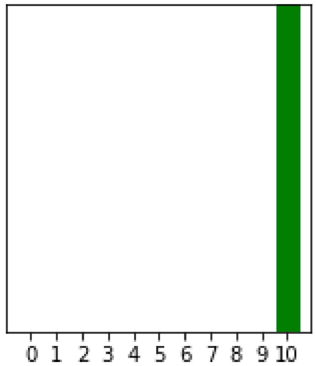
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```



In [53]:
```
i=109
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, testX)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i], test_labels)
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

Stratus 100% (Stratus)



In [54]:

```
predictions = CloudMobiNet_model.predict(test_batches)
for i in range(len(test_batches)):
 print("X=%s, Predicted=%s" % (test_batches[i], predictions[i]))
```

```
1/1 [==============================] - 3s 3s/step
X=(array([[[[-0.09019607, -0.0745098 ,  0.09019613],
         [-0.09803921, -0.08235294,  0.082353  ],
         [-0.11372548, -0.09803921,  0.06666672],
         ...,
         [ 0.14509809,  0.1686275 ,  0.28627455],
         [ 0.14509809,  0.1686275 ,  0.28627455],
         [ 0.13725495,  0.16078436,  0.27843142]],

        [[-0.09019607, -0.0745098 ,  0.09019613],
         [-0.09803921, -0.08235294,  0.082353  ],
         [-0.10588235, -0.09019607,  0.07450986],
         ...,
         [ 0.14509809,  0.1686275 ,  0.28627455],
         [ 0.14509809,  0.1686275 ,  0.28627455],
         [ 0.13725495,  0.16078436,  0.27843142]],

        [[-0.09803921, -0.08235294,  0.082353  ],
         [-0.09803921, -0.08235294,  0.082353  ],
         [-0.09803921, -0.08235294,  0.082353  ],
         ...,
         [ 0.14509809,  0.1686275 ,  0.28627455],
         [ 0.13725495,  0.16078436,  0.27843142],
         [ 0.14509809,  0.1686275 ,  0.28627455]],

        ...,

        [[-0.372549  , -0.4980392 , -0.69411767],
         [-0.27058822, -0.36470586, -0.5529412 ],
         [-0.09019607, -0.14509803, -0.29411763],
         ...,
         [-0.18431371, -0.1372549 ,  0.03529418],
         [-0.17647058, -0.12156862,  0.00392163],
         [-0.11372548, -0.05098039,  0.05098045]],

        [[-0.372549  , -0.4980392 , -0.69411767],
         [-0.21568626, -0.3098039 , -0.4980392 ],
         [-0.0745098 , -0.12941176, -0.27058822],
         ...,
         [-0.26274508, -0.19999999, -0.01960784],
         [-0.02745098,  0.04313731,  0.17647064],
         [-0.09019607, -0.01176471,  0.082353  ]],

        [[-0.04313725, -0.16862744, -0.36470586],
         [-0.06666666, -0.1607843 , -0.3490196 ],
         [-0.0745098 , -0.12941176, -0.27058822],
```

```
  ...,
  [-0.6392157 , -0.5764706 , -0.3960784 ],
  [-0.654902  , -0.58431375, -0.45098037],
  [-0.75686276, -0.6784314 , -0.58431375]]],


[[[ 0.3803922 ,  0.45098042,  0.4901961 ],
  [ 0.38823533,  0.45882356,  0.49803925],
  [ 0.35686278,  0.427451  ,  0.4666667 ],
  ...,
  [ 0.6156863 ,  0.7490196 ,  0.9529412 ],
  [ 0.60784316,  0.7411765 ,  0.94509804],
  [ 0.5764706 ,  0.70980394,  0.9137255 ]],

 [[ 0.4039216 ,  0.47450984,  0.5137255 ],
  [ 0.39607847,  0.4666667 ,  0.5058824 ],
  [ 0.34901965,  0.41960788,  0.45882356],
  ...,
  [ 0.5921569 ,  0.7254902 ,  0.92941177],
  [ 0.58431375,  0.7176471 ,  0.92156863],
  [ 0.5764706 ,  0.70980394,  0.9137255 ]],

 [[ 0.43529415,  0.5058824 ,  0.54509807],
  [ 0.39607847,  0.4666667 ,  0.5058824 ],
  [ 0.33333337,  0.4039216 ,  0.4431373 ],
  ...,
  [ 0.58431375,  0.7176471 ,  0.92156863],
  [ 0.5764706 ,  0.70980394,  0.9137255 ],
  [ 0.5686275 ,  0.7019608 ,  0.90588236]],

 ...,

 [[ 0.26274514,  0.30196083,  0.33333337],
  [ 0.27843142,  0.3176471 ,  0.34901965],
  [ 0.2941177 ,  0.33333337,  0.36470592],
  ...,
  [ 0.62352943,  0.6627451 ,  0.6862745 ],
  [ 0.6156863 ,  0.654902  ,  0.6784314 ],
  [ 0.6       ,  0.6392157 ,  0.6627451 ]],

 [[ 0.24705887,  0.28627455,  0.3176471 ],
  [ 0.26274514,  0.30196083,  0.33333337],
  [ 0.27843142,  0.3176471 ,  0.34901965],
  ...,
  [ 0.52156866,  0.58431375,  0.6       ],
  [ 0.52156866,  0.58431375,  0.6       ],
  [ 0.5294118 ,  0.5921569 ,  0.60784316]],

 [[ 0.24705887,  0.28627455,  0.3176471 ],
  [ 0.254902  ,  0.2941177 ,  0.32549024],
  [ 0.26274514,  0.30196083,  0.33333337],
  ...,
  [ 0.4666667 ,  0.5294118 ,  0.54509807],
  [ 0.427451  ,  0.4901961 ,  0.5058824 ],
  [ 0.43529415,  0.49803925,  0.5137255 ]]],


[[[ 0.15294123,  0.21568632,  0.3176471 ],
  [ 0.12941182,  0.19215691,  0.2941177 ],
  [ 0.10588241,  0.1686275 ,  0.27058828],
  ...,
  [-0.32549018, -0.20784312,  0.0196079 ],
  [-0.34117645, -0.2235294 ,  0.03529418],
  [-0.29411763, -0.18431371,  0.10588241]],
```

```
 [[ 0.14509809,  0.20784318,  0.30980396],
  [ 0.12156868,  0.18431377,  0.28627455],
  [ 0.09803927,  0.15294123,  0.27843142],
  ...,
  [-0.3490196 , -0.23137254, -0.00392157],
  [-0.3098039 , -0.19215685,  0.06666672],
  [-0.27843136, -0.16862744,  0.12156868]],

 [[ 0.12941182,  0.19215691,  0.2941177 ],
  [ 0.11372554,  0.17647064,  0.27843142],
  [ 0.09803927,  0.15294123,  0.27843142],
  ...,
  [-0.20784312, -0.09803921,  0.12941182],
  [-0.21568626, -0.09803921,  0.16078436],
  [-0.34117645, -0.23137254,  0.04313731]],

 ...,

 [[-0.9764706 , -0.9529412 , -1.        ],
  [-1.        , -0.9843137 , -1.        ],
  [-1.        , -0.9843137 , -1.        ],
  ...,
  [-1.        , -0.9843137 , -1.        ],
  [-0.9843137 , -0.96862745, -0.99215686],
  [-0.99215686, -0.9764706 , -1.        ]],

 [[-0.64705884, -0.62352943, -0.69411767],
  [-0.94509804, -0.92156863, -0.9764706 ],
  [-0.9843137 , -0.96862745, -1.        ],
  ...,
  [-0.99215686, -0.9764706 , -1.        ],
  [-1.        , -0.9843137 , -1.        ],
  [-0.9764706 , -0.9607843 , -0.9843137 ]],

 [[-0.6627451 , -0.6392157 , -0.70980394],
  [-0.5294118 , -0.5058824 , -0.56078434],
  [-0.84313726, -0.827451  , -0.8666667 ],
  ...,
  [-0.9843137 , -0.96862745, -0.99215686],
  [-1.        , -0.99215686, -1.        ],
  [-0.99215686, -0.9764706 , -1.        ]]],


...,


[[[-0.47450978, -0.29411763, -0.15294117],
  [-0.47450978, -0.29411763, -0.15294117],
  [-0.46666664, -0.2862745 , -0.14509803],
  ...,
  [-0.34117645, -0.21568626, -0.11372548],
  [-0.38039213, -0.25490195, -0.15294117],
  [-0.38823527, -0.26274508, -0.1607843 ]],

 [[-0.47450978, -0.29411763, -0.15294117],
  [-0.47450978, -0.29411763, -0.15294117],
  [-0.46666664, -0.2862745 , -0.14509803],
  ...,
  [-0.35686272, -0.23137254, -0.12941176],
  [-0.38039213, -0.25490195, -0.15294117],
  [-0.38039213, -0.25490195, -0.15294117]],

 [[-0.47450978, -0.29411763, -0.15294117],
  [-0.47450978, -0.29411763, -0.15294117],
  [-0.47450978, -0.29411763, -0.15294117],
```

```
          ...,
          [-0.34117645, -0.21568626, -0.11372548],
          [-0.35686272, -0.23137254, -0.12941176],
          [-0.32549018, -0.19999999, -0.09803921]],

         ...,

         [[-0.64705884, -0.6313726 , -0.6392157 ],
          [-0.64705884, -0.6313726 , -0.6392157 ],
          [-0.654902  , -0.6392157 , -0.64705884],
          ...,
          [-0.6156863 , -0.6       , -0.62352943],
          [-0.6156863 , -0.6       , -0.62352943],
          [-0.6156863 , -0.6       , -0.62352943]],

         [[-0.6313726 , -0.6156863 , -0.62352943],
          [-0.6313726 , -0.6156863 , -0.62352943],
          [-0.6313726 , -0.6156863 , -0.62352943],
          ...,
          [-0.6313726 , -0.6156863 , -0.6392157 ],
          [-0.62352943, -0.60784316, -0.6313726 ],
          [-0.60784316, -0.5921569 , -0.6156863 ]],

         [[-0.6392157 , -0.62352943, -0.6313726 ],
          [-0.6392157 , -0.62352943, -0.6313726 ],
          [-0.64705884, -0.6313726 , -0.6392157 ],
          ...,
          [-0.6392157 , -0.62352943, -0.64705884],
          [-0.62352943, -0.60784316, -0.6313726 ],
          [-0.6       , -0.58431375, -0.60784316]]],


        [[[ 0.427451  ,  0.45098042,  0.4901961 ],
          [ 0.41960788,  0.4431373 ,  0.48235297],
          [ 0.41960788,  0.4431373 ,  0.48235297],
          ...,
          [ 0.41960788,  0.41960788,  0.48235297],
          [ 0.427451  ,  0.427451  ,  0.4901961 ],
          [ 0.4431373 ,  0.4431373 ,  0.5058824 ]],

         [[ 0.43529415,  0.45882356,  0.49803925],
          [ 0.427451  ,  0.45098042,  0.4901961 ],
          [ 0.41960788,  0.4431373 ,  0.48235297],
          ...,
          [ 0.33333337,  0.33333337,  0.39607847],
          [ 0.34901965,  0.34901965,  0.41176474],
          [ 0.37254906,  0.37254906,  0.43529415]],

         [[ 0.4431373 ,  0.4666667 ,  0.5058824 ],
          [ 0.41960788,  0.4431373 ,  0.48235297],
          [ 0.41960788,  0.4431373 ,  0.48235297],
          ...,
          [ 0.3176471 ,  0.3176471 ,  0.3803922 ],
          [ 0.35686278,  0.35686278,  0.41960788],
          [ 0.38823533,  0.38823533,  0.45098042]],

         ...,

         [[-0.6313726 , -0.69411767, -0.78039217],
          [-0.6392157 , -0.7019608 , -0.7882353 ],
          [-0.6392157 , -0.7019608 , -0.7882353 ],
          ...,
          [-0.4588235 , -0.6       , -0.77254903],
          [-0.49019605, -0.6313726 , -0.8039216 ],
          [-0.5058824 , -0.64705884, -0.81960785]],
```

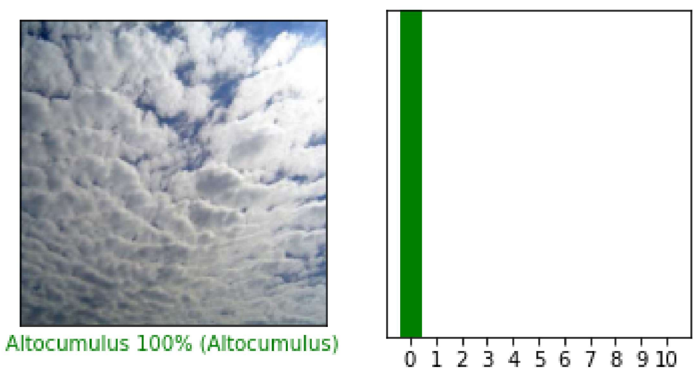
```
 [[ 0.06666672,  0.03529418, -0.03529412],
  [ 0.05098045,  0.0196079 , -0.05098039],
  [ 0.04313731,  0.01176476, -0.05882353],
  ...,
  [-0.4588235 , -0.5529412 , -0.6627451 ],
  [-0.4823529 , -0.5764706 , -0.6862745 ],
  [-0.49019605, -0.58431375, -0.69411767]],

 [[ 0.7254902 ,  0.7019608 ,  0.64705884],
  [ 0.70980394,  0.6862745 ,  0.6313726 ],
  [ 0.69411767,  0.67058825,  0.6156863 ],
  ...,
  [-0.06666666, -0.09803921, -0.18431371],
  [-0.0745098 , -0.10588235, -0.19215685],
  [-0.05882353, -0.09019607, -0.1607843 ]]],


[[[-0.54509807, -0.1607843 ,  0.27843142],
  [-0.54509807, -0.1607843 ,  0.27843142],
  [-0.54509807, -0.1607843 ,  0.27843142],
  ...,
  [ 0.03529418,  0.24705887,  0.58431375],
  [ 0.04313731,  0.254902  ,  0.5921569 ],
  [ 0.07450986,  0.28627455,  0.62352943]],

 [[-0.5294118 , -0.14509803,  0.2941177 ],
  [-0.5294118 , -0.14509803,  0.2941177 ],
  [-0.52156866, -0.1372549 ,  0.30196083],
  ...,
  [ 0.05098045,  0.26274514,  0.6       ],
  [ 0.04313731,  0.254902  ,  0.5921569 ],
  [ 0.04313731,  0.254902  ,  0.5921569 ]],

 [[-0.5372549 , -0.15294117,  0.28627455],
  [-0.5372549 , -0.15294117,  0.28627455],
  [-0.5294118 , -0.14509803,  0.2941177 ],
  ...,
  [ 0.09803927,  0.30980396,  0.64705884],
  [ 0.082353  ,  0.2941177 ,  0.6313726 ],
  [ 0.06666672,  0.27843142,  0.6156863 ]],

 ...,

 [[-0.32549018, -0.30196077, -0.5294118 ],
  [-0.41176468, -0.4588235 , -0.6627451 ],
  [-0.58431375, -0.7254902 , -0.8980392 ],
  ...,
  [-0.58431375, -0.7411765 , -0.8117647 ],
  [-0.6313726 , -0.8117647 , -0.8745098 ],
  [-0.7019608 , -0.88235295, -0.94509804]],

 [[-0.81960785, -0.7411765 , -0.92941177],
  [-0.34117645, -0.3490196 , -0.49019605],
  [-0.654902  , -0.77254903, -0.92156863],
  ...,
  [-0.58431375, -0.7411765 , -0.79607844],
  [-0.73333335, -0.8901961 , -0.94509804],
  [-0.6627451 , -0.827451  , -0.8666667 ]],

 [[-0.85882354, -0.75686276, -0.8980392 ],
  [-0.6       , -0.5764706 , -0.70980394],
  [-0.5058824 , -0.6       , -0.7254902 ],
  ...,
  [-0.6392157 , -0.79607844, -0.8666667 ],
```

```
         [-0.69411767, -0.8509804 , -0.90588236],
         [-0.62352943, -0.78039217, -0.8352941 ]]]], dtype=float32), array([[1., 0., 0.,
..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.]], dtype=float32)), Predicted=[9.9999166e-01 9.68315
93e-07 1.5438092e-06 3.1837392e-06 2.6253348e-09
 5.8913634e-09 7.3930245e-08 4.3575041e-09 3.6041396e-07 2.9845236e-08
 2.0578241e-06]
```

In [55]:

```
i=1
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, testX)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i], test_labels)
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```



In [56]:

```
#printing the first element from predicted data
pred=CloudMobiNet_model.predict(test_batches)
print(pred[1])
#printing the index of
print('Index:',np.argmax(pred[1]))
```

```
1/1 [==============================] - 3s 3s/step
[9.9999940e-01 2.8567024e-07 6.9118293e-08 8.0745348e-09 7.9743567e-10
 9.3173025e-09 1.4445129e-08 9.0659498e-09 8.3250541e-08 1.5470862e-09
 9.1152835e-08]
Index: 0
```

In [57]:

```
y_classes = [np.argmax(element) for element in pred]
print('Predicted_values:',pred[:110])
print('Actual_values:',y_pred[:110])
```

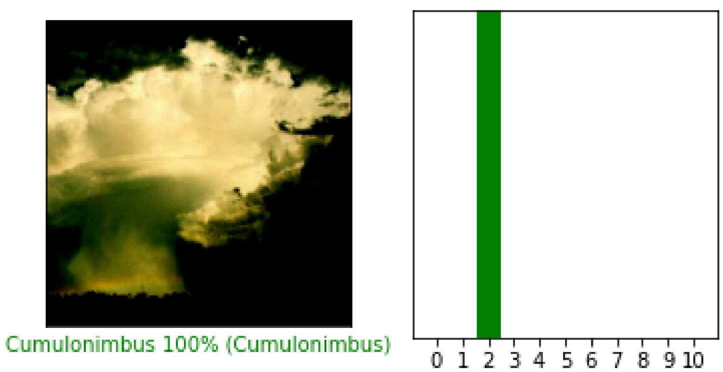
```
Predicted_values: [[9.9999166e-01 9.6831593e-07 1.5438092e-06 ... 3.6041396e-07
  2.9845236e-08 2.0578241e-06]
 [9.9999940e-01 2.8567024e-07 6.9118293e-08 ... 8.3250541e-08
  1.5470862e-09 9.1152835e-08]
 [9.9999869e-01 8.6377771e-12 7.9065351e-13 ... 1.3093185e-08
  4.1794167e-13 1.3357350e-06]
 ...
```

```
[4.5870058e-10 3.7879904e-08 3.2199129e-08 ... 6.2749854e-07
 4.3797286e-08 9.9999917e-01]
[2.9766650e-04 4.3526511e-03 2.9316428e-04 ... 1.6459420e-01
 8.6311381e-03 8.1986427e-01]
[6.6972603e-08 6.0949361e-08 3.2543997e-07 ... 2.5278161e-05
 1.0236427e-03 9.9869210e-01]]
Actual_values: [ 0  0  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1  1  1  2  2  2  2
  2  2  2  2  4  2  3  5  3  3  3  3  3  3  3  3  4  4  4  4  4  4  4  4
  4  4  5  5  5  5  5  5  5  5  5  5  6  6  6  6  6  6  6  6  6  6  7  7
  7  7  7  7  7  7  7  7  8  8 10  8 10  8  8  8  8  8  9  9  9  9  9  9
  9  9  9  9 10 10 10 10 10 10 10 10 10 10]
```

In [59]:

```python
i=21
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, testX)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i], test_labels)
plt.show()
```

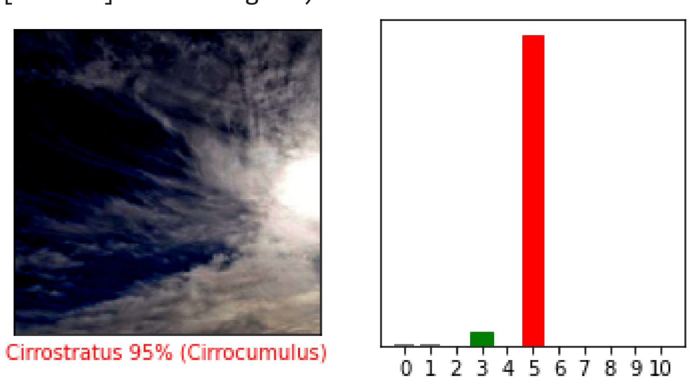
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```



In [60]:

```python
i=31
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, testX)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i], test_labels)
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

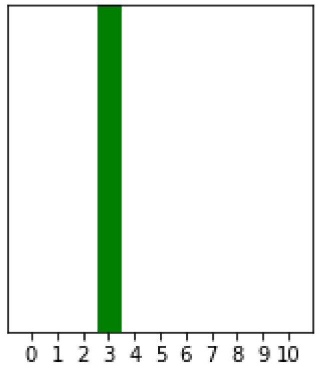


In [63]:

```
i=38
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, testX)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i], test_labels)
plt.show()
```

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

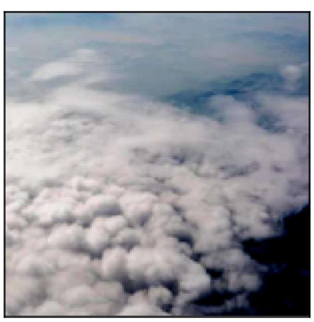
Cirrocumulus 100% (Cirrocumulus)



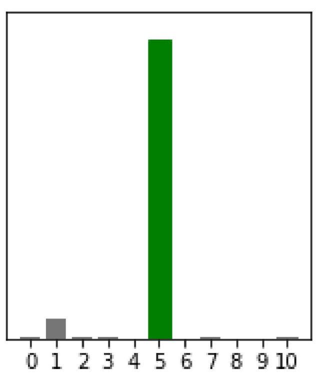
In [64]:

```
i=54
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, testX)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i], test_labels)
plt.show()
```

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).



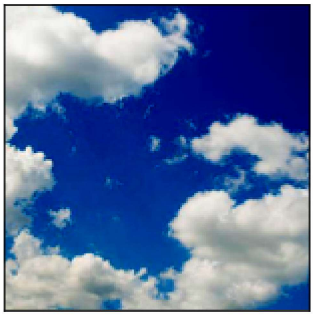
Cirrostratus 91% (Cirrostratus)



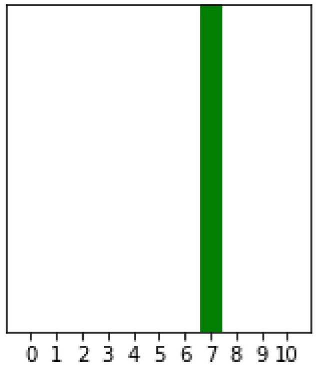
In [65]:

```
i=71
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, testX)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i], test_labels)
plt.show()
```

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).



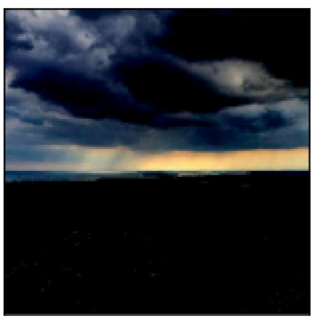
Cumulus 100% (Cumulus)



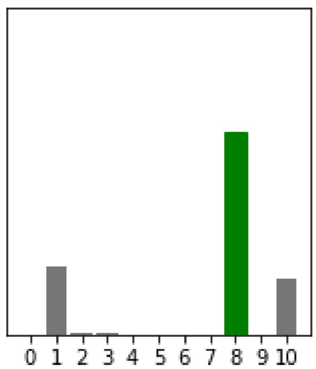
In [66]:

```
i=81
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, testX)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i], test_labels)
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```



Nimbostratus 62% (Nimbostratus)

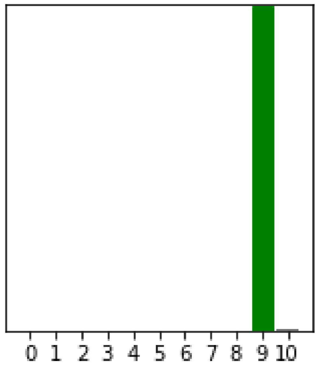


In [67]:

```
i=90
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, testX)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i], test_labels)
plt.show()
```

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Stratocumulus 99% (Stratocumulus)

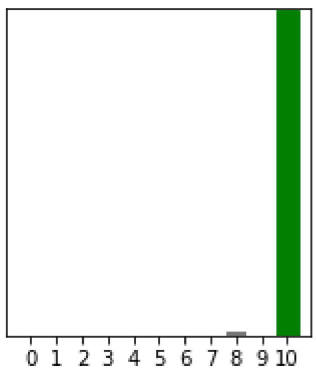


In [69]:

```
i=105
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, testX)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i], test_labels)
plt.show()
```

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

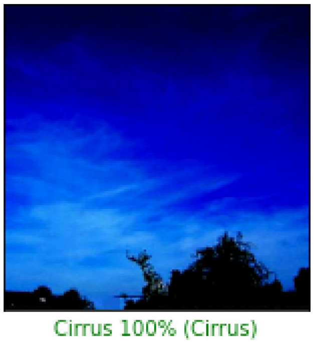
Stratus 99% (Stratus)

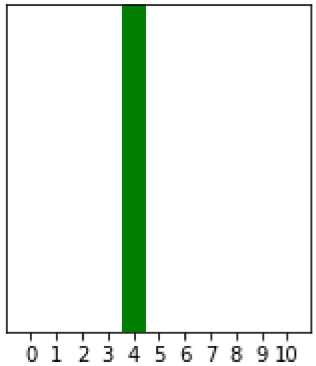


In [70]:

```
i=47
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, testX)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i], test_labels)
plt.show()
```

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

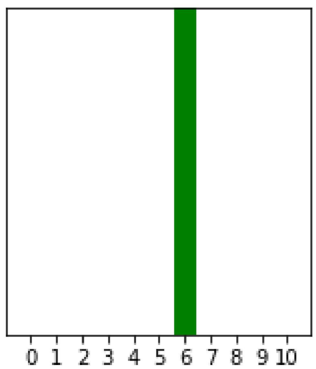




In [71]:

```
i=60
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, testX)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i], test_labels)
plt.show()
```

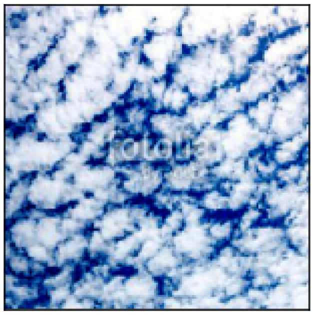
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```



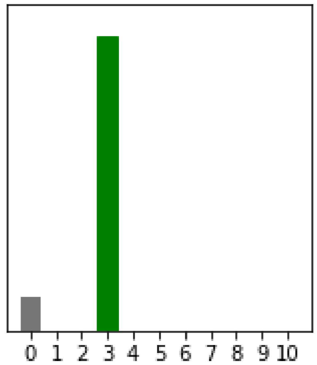
In [72]:

```
i=33
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, testX)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i], test_labels)
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```



Cirrocumulus 90% (Cirrocumulus)



In [58]:

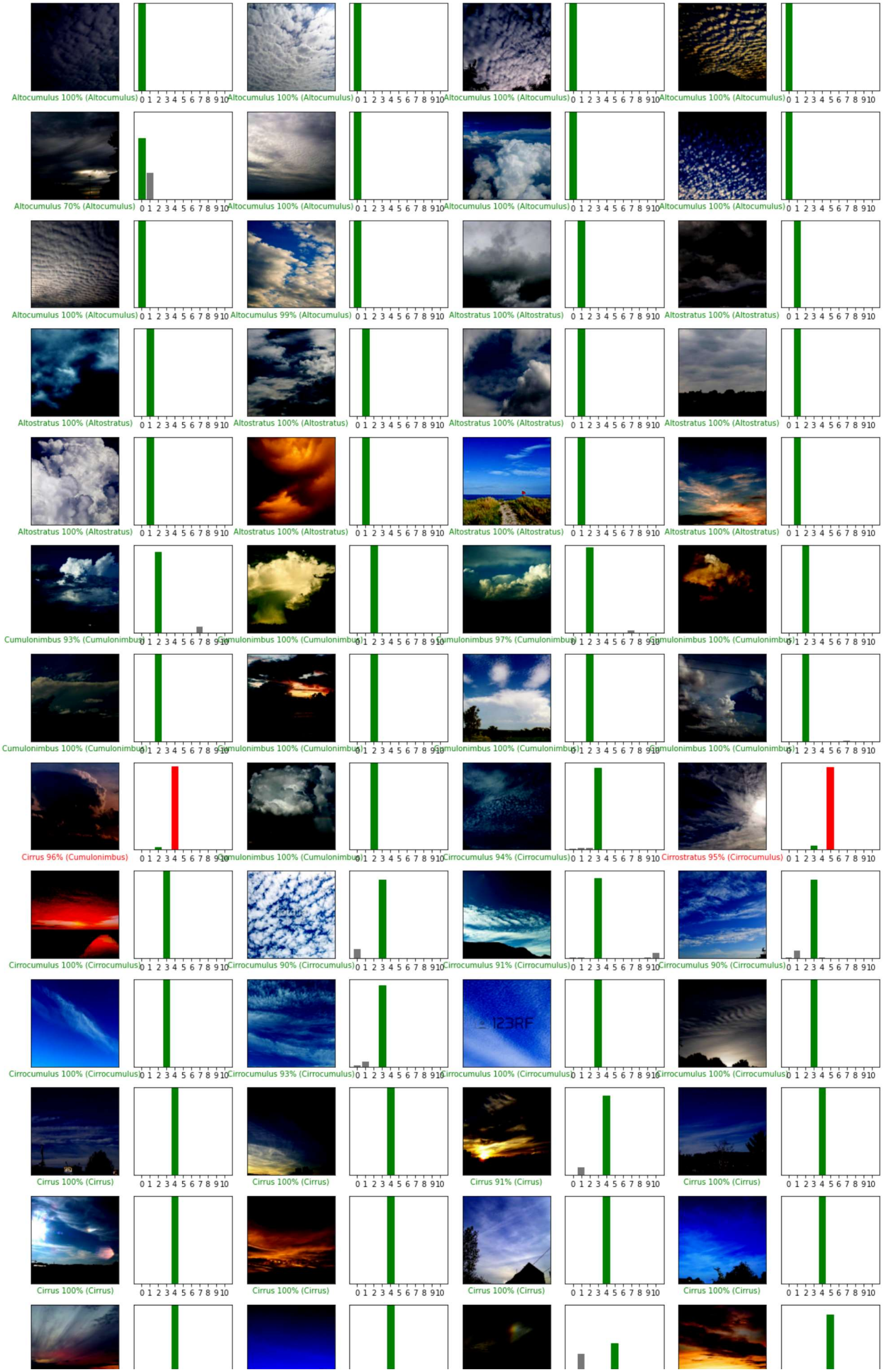
```
rows=27
cols=4

num_images = rows * cols
plt.figure(figsize=(2*2*cols, 2*rows))

for i in range(num_images):
    plt.subplot(rows, 2*cols, 2*i+1)
    plot_image(i, predictions[i], test_labels, testX)
    plt.subplot(rows, 2*cols, 2*i+2)
    plot_value_array(i, predictions[i],test_labels)
plt.tight_layout()
plt.show()
```

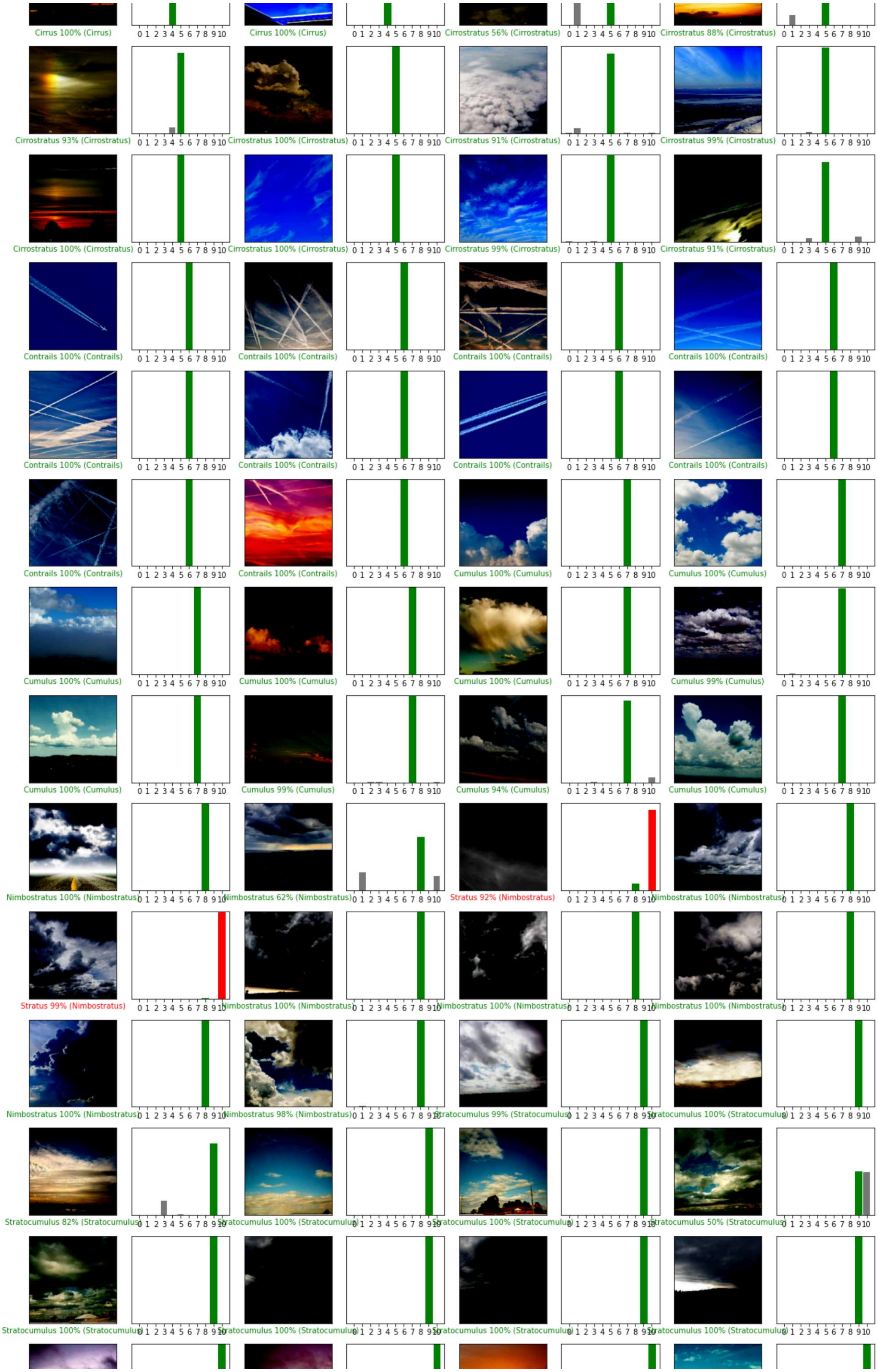
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

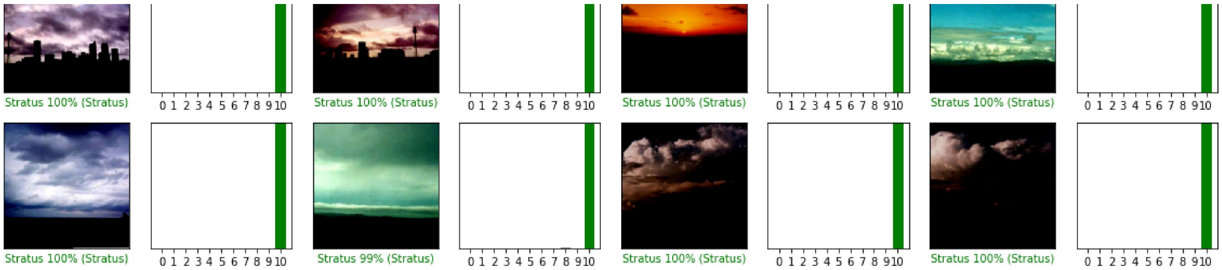
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
```

```
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```


Altocumulus 100% (Altocumulus)
Altocumulus 100% (Altocumulus)
Altocumulus 100% (Altocumulus)
Altocumulus 100% (Altocumulus)
Altocumulus 70% (Altocumulus)
Altocumulus 100% (Altocumulus)
Altocumulus 100% (Altocumulus)
Altocumulus 100% (Altocumulus)
Altocumulus 100% (Altocumulus)
Altocumulus 99% (Altocumulus)
Altostratus 100% (Altostratus)
Altostratus 100% (Altostratus)
Altostratus 100% (Altostratus)
Altostratus 100% (Altostratus)
Altostratus 100% (Altostratus)
Altostratus 100% (Altostratus)
Altostratus 100% (Altostratus)
Altostratus 100% (Altostratus)
Altostratus 100% (Altostratus)
Altostratus 100% (Altostratus)
Cumulonimbus 93% (Cumulonimbus
Cumulonimbus 97% (Cumulonimbus
Cumulonimbus 100% (Cumulonimbus
Cirrus 96% (Cumulonimbus)
Cirrocumulus 94% (Cirrocumulus)
Cirrostratus 95% (Cirrocumulus)
Cirrocumulus 100% (Cirrocumulus)
Cirrocumulus 90% (Cirrocumulus)
Cirrocumulus 91% (Cirrocumulus)
Cirrocumulus 90% (Cirrocumulus)
Cirrocumulus 100% (Cirrocumulus
Cirrocumulus 93% (Cirrocumulus)
Cirrus 100% (Cirrus)
Cirrus 100% (Cirrus)
Cirrus 91% (Cirrus)
Cirrus 100% (Cirrus)
Cirrus 100% (Cirrus)
Cirrus 100% (Cirrus)
Cirrus 100% (Cirrus)
Cirrus 100% (Cirrus)


Cirrus 100% (Cirrus)
Cirrus 100% (Cirrus)
Cirrostratus 56% (Cirrostratus)
Cirrostratus 88% (Cirrostratus)
Cirrostratus 93% (Cirrostratus)
Cirrostratus 100% (Cirrostratus)
Cirrostratus 91% (Cirrostratus)
Cirrostratus 99% (Cirrostratus)
Cirrostratus 100% (Cirrostratus)
Cirrostratus 100% (Cirrostratus)
Cirrostratus 99% (Cirrostratus)
Cirrostratus 91% (Cirrostratus)
Contrails 100% (Contrails)
Contrails 100% (Contrails)
Contrails 100% (Contrails)
Contrails 100% (Contrails)
Contrails 100% (Contrails)
Contrails 100% (Contrails)
Contrails 100% (Contrails)
Contrails 100% (Contrails)
Contrails 100% (Contrails)
Contrails 100% (Contrails)
Cumulus 100% (Cumulus)
Cumulus 100% (Cumulus)
Cumulus 100% (Cumulus)
Cumulus 100% (Cumulus)
Cumulus 100% (Cumulus)
Cumulus 99% (Cumulus)
Cumulus 100% (Cumulus)
Cumulus 99% (Cumulus)
Cumulus 94% (Cumulus)
Cumulus 100% (Cumulus)
Nimbostratus 100% (Nimbostratus)
Nimbostratus 62% (Nimbostratus)
Stratus 92% (Nimbostratus)
Nimbostratus 100% (Nimbostratus)
Stratus 99% (Nimbostratus)
Nimbostratus 100% (Nimbostratus)
Nimbostratus 100% (Nimbostratus)
Nimbostratus 100% (Nimbostratus)
Nimbostratus 100% (Nimbostratus)
Nimbostratus 98% (Nimbostratus)
Stratocumulus 99% (Stratocumulus)
Stratocumulus 100% (Stratocumulus)
Stratocumulus 82% (Stratocumulus)
Stratocumulus 100% (Stratocumulus)
Stratocumulus 100% (Stratocumulus)
Stratocumulus 50% (Stratocumulus)
Stratocumulus 100% (Stratocumulus)
Stratocumulus 100% (Stratocumulus)
Stratocumulus 100% (Stratocumulus)
Stratocumulus 100% (Stratocumulus)


Stratus 100% (Stratus)
0 1 2 3 4 5 6 7 8 9 10
Stratus 100% (Stratus)
0 1 2 3 4 5 6 7 8 9 10
Stratus 100% (Stratus)
0 1 2 3 4 5 6 7 8 9 10
Stratus 100% (Stratus)
0 1 2 3 4 5 6 7 8 9 10
Stratus 100% (Stratus)
0 1 2 3 4 5 6 7 8 9 10
Stratus 99% (Stratus)
0 1 2 3 4 5 6 7 8 9 10
Stratus 100% (Stratus)
0 1 2 3 4 5 6 7 8 9 10
Stratus 100% (Stratus)
0 1 2 3 4 5 6 7 8 9 10