ENEL 525:
Weather Classification Project

Created by: Abdelrahman Abbas
UCIDs: 30110374
Lab Section: B01
Tutorial Section: N/A
Due Date of Report: Friday, December 17, 2021

Note: most of this labs' written work is a rough rephrasing from the CNN Explainer website and the project handout, with some original thoughts, I say this so as to not impersonate this work as completely mine.

Introduction:

In this project, I shall develop a model to classify weather conditions given an input image. I will consider the model to be one commonly used in deep learning, namely convolutional neural network (CNN). In machine learning, a classifier assigns a class label to a data point. For example, an *image classifier* produces a class label (e.g, bird, plane) for what objects exist within an image. A *convolutional neural network*, or CNN for short, is a type of classifier, which excels at solving this problem! A CNN is a neural network: an algorithm used to recognize patterns in data. Neural Networks in general are composed of a collection of neurons that are organized in layers, each with their own learnable weights and biases.

Preprocessing:

I used the following load_pics function to be able to read and resize the images given to us through the project handout:

```
def load_pics(dataset_path):
    dataset = []
    targets = []
    imgs = []

    for filename in os.listdir(dataset_path):

        try:
            img = cv2.imread(os.path.realpath(
                os.path.join(dataset_path, filename)))

            imgs.append(img)

            # apply preprocessing
            img_adjusted = cv2.resize(img, (IMG_HEIGHT, IMG_WIDTH))

            dataset.append(img_adjusted)

            ##saving the index of the file, which will later be used for checking our results
            pic_name = os.path.basename(os.path.normpath(filename))
            if pic_name.find(CLASSES[0]) != -1:
                targets.append(0)
            elif pic_name.find(CLASSES[1]) != -1:
                targets.append(1)
```

```
            elif pic_name.find(CLASSES[2]) != -1:
                targets.append(2)
            elif pic_name.find(CLASSES[3]) != -1:
                targets.append(3)
        except:
            pass

    # convert the python lists to numpy arrays for usability
    dataset = np.array(dataset)
    targets = np.array(targets)
    imgs = np.array(imgs)

    return [dataset, targets, imgs]
```

This function just basically runs through every image from the database directory, and saves the original image in the *imgs* list, while saving the preprocessed images to the *dataset* list. Lastly, since the dataset of images was provided with the label of the weather as part of each images' name, I just checked for each of the labels in the image's name and gave them a respective integer from 0-3 in their corresponding index of the *targets* list.

After loading the full dataset and targets of images let's use 80 % of the images for training, and 20 % for validation. However, will use a random permutation to choose random pics for the training and test data, as follows:

```
permuted_idx = np.random.permutation(dataset.shape[0])
X_train = dataset[permuted_idx[0:(int)(len(dataset)*0.8)]]
y_train = targets[permuted_idx[0:(int)(len(targets)*0.8)]]
X_test = dataset[permuted_idx[((int)(len(dataset)*0.8) + 1):]]
y_test = targets[permuted_idx[((int)(len(targets)*0.8) + 1):]]
```

As a last step in our preprocessing phase, I will convert the label train and test arrays into a one_hot encoding since will be using softmax in our output layer:

```
y_train = tf.one_hot(y_train, NUM_CLASSES)
y_test_onehot = tf.one_hot(y_test, NUM_CLASSES)
```

Approach/Discussion:

As for the CNN architecture, I used a sequential model that consists of 3 convolutional layers, that are activated by a ReLU activation function, along with a max pooling layer that corresponds with each of convolutional layers. Furthermore, there's a fully-connected layer with 32 units on top of these convolutional blocks that is also activated by a ReLU activation function. Lastly, I used a 4 neuron softmax output layer, which helps by making sure the CNN outputs sum to 1. Because of this, softmax operations are useful to scale model outputs into probabilities. Here is the code for this model:

```
model = Sequential([
  layers.Rescaling(1./127.5,offset=-1),
  layers.Conv2D(filters=8, kernel_size=3, activation='relu',input_shape=[IMG_HEIGHT,
IMG_WIDTH, 3]),
  layers.MaxPooling2D(pool_size=2, strides=2),
  layers.Conv2D(filters=16, kernel_size=3, activation='relu'),
  layers.MaxPooling2D(pool_size=2, strides=2),
  layers.Conv2D(filters=32, kernel_size=3, activation='relu'),
  layers.MaxPooling2D(pool_size=2, strides=2),
  layers.Dropout(0.5),
  layers.Flatten(),
  layers.Dense(units=32, activation='relu'),
  layers.Dropout(0.5),
  layers.Dense(units=NUM_CLASSES, activation='softmax')
])
```

Furthermore, since he RGB channel values are in the[0, 255] range. This is not ideal for a neural network, because in general you should seek to make your input values small. Here, I standardized the values to be in the[-1, 1] range by passing scale=1./127.5, offset=-1 into the rescaling layer.

When I first set up my model with just the convolutional blocks and the neural network layer, I realized that there was a lot of overfittings. To reduce overfitting, I introduced dropout regularization to the network. When I apply dropout to a layer, it randomly drops out (by setting the activation to zero) a number of output units from the layer during the training process. This could be seen in the model above, where dropout takes a fractional number as its input value, in the form such as 0.1, 0.2, 0.4, etc. This means dropping out 10 %, 20 % or 40 % of the output units randomly from the applied layer.

Next, I choose the Adam optimizer and categorical_crossentropy loss function, as these were the parameters used in the lesson exercises. To view training and validation accuracy for each training epoch, I passed the accuracy metric argument to Model.compile:

```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

#View all the layers of the network using the model's Model.summary method:
model.build(input_shape=(None,IMG_HEIGHT, IMG_WIDTH, 3))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 150, 150, 3)       0

 conv2d (Conv2D)             (None, 148, 148, 8)       224

 max_pooling2d (MaxPooling2D  (None, 74, 74, 8)        0
 )

 conv2d_1 (Conv2D)           (None, 72, 72, 16)        1168

 max_pooling2d_1 (MaxPooling  (None, 36, 36, 16)       0
 2D)

 conv2d_2 (Conv2D)           (None, 34, 34, 32)        4640

 max_pooling2d_2 (MaxPooling  (None, 17, 17, 32)       0
 2D)

 dropout (Dropout)           (None, 17, 17, 32)        0

 flatten (Flatten)           (None, 9248)              0

 dense (Dense)               (None, 32)                295968

 dropout_1 (Dropout)         (None, 32)                0

 dense_1 (Dense)             (None, 4)                 132

=================================================================
Total params: 302,132
Trainable params: 302,132
Non-trainable params: 0
_________________________________________________________________
```

One other good measure that I took to better the performance of my model was to use data augmentation in my design. Overfitting generally occurs when there are a small number of training examples. Data augmentation takes the approach of generating additional training data

from the existing examples/dataset by augmenting them using random transformations that yield believable-looking images. This helps expose the model to more aspects of the data and generalize better. In this case, I introduced three transformations: zoom, shear, and brightness augmentations, as can be seen in the code below:

```
datagen = ImageDataGenerator(
   zoom_range=0.2,
   shear_range=0.2,
   brightness_range=[0.8,1.3],)

epochs = 25
history = model.fit(datagen.flow(X_train, y_train, batch_size=32),
        validation_data=(X_test, y_test_onehot), epochs=epochs)
```

Finally, when it came to choosing the "best" parameters and architecture-structure of this aforementioned model, all I did was use a baseline algorithm/structure to which I tweaked the parameter of this structure while using a constant random seed to assure a reproducibility between different parameters, and ensure that the baseline performance was truly constant between every new iteration of the model:

```
seed = 1
tf.random.set_seed(seed)
np.random.seed(seed)
```

Furthermore, the baseline model which I started with was the model used in the "image classification" tutorial from the TensorFlow website:

```
model = Sequential([
  data_augmentation,
  layers.Rescaling(1./255),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

Results:

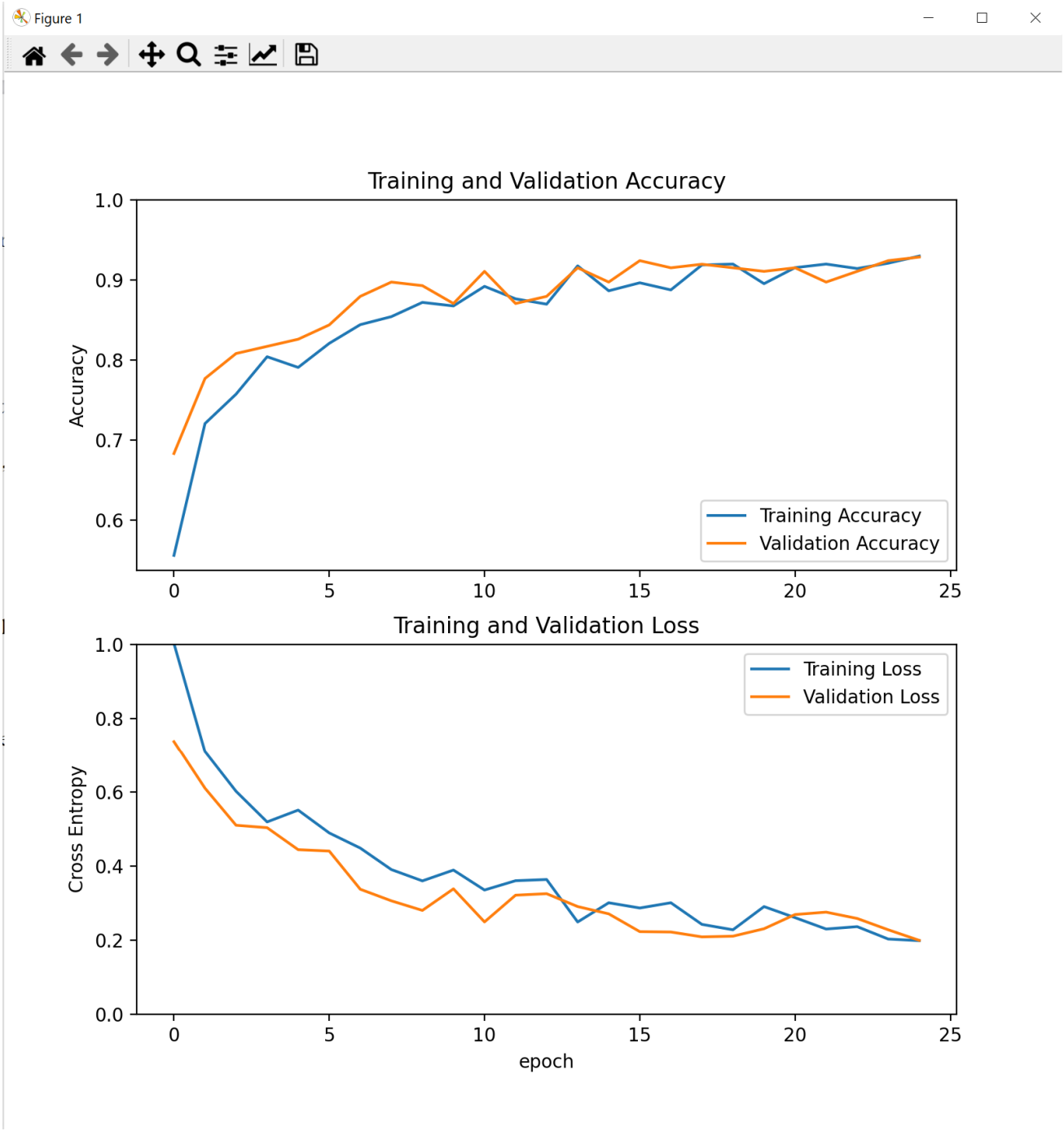
Model's Accuracies and Losses:

```
_________________________________________________________________
Epoch 1/25
29/29 [==============================] - 6s 196ms/step - loss: 1.0028 - accuracy: 0.5557 - val_loss: 0.7375 - val_accuracy
: 0.6830
Epoch 2/25
29/29 [==============================] - 5s 187ms/step - loss: 0.7097 - accuracy: 0.7205 - val_loss: 0.6102 - val_accuracy
: 0.7768
Epoch 3/25
29/29 [==============================] - 6s 189ms/step - loss: 0.6019 - accuracy: 0.7572 - val_loss: 0.5102 - val_accuracy
: 0.8080
Epoch 4/25
29/29 [==============================] - 5s 175ms/step - loss: 0.5192 - accuracy: 0.8040 - val_loss: 0.5036 - val_accuracy
: 0.8170
Epoch 5/25
29/29 [==============================] - 5s 180ms/step - loss: 0.5511 - accuracy: 0.7906 - val_loss: 0.4443 - val_accuracy
: 0.8259
Epoch 6/25
29/29 [==============================] - 5s 186ms/step - loss: 0.4895 - accuracy: 0.8207 - val_loss: 0.4406 - val_accuracy
: 0.8438
Epoch 7/25
29/29 [==============================] - 5s 182ms/step - loss: 0.4486 - accuracy: 0.8441 - val_loss: 0.3375 - val_accuracy
: 0.8795
Epoch 8/25
29/29 [==============================] - 5s 186ms/step - loss: 0.3906 - accuracy: 0.8541 - val_loss: 0.3062 - val_accuracy
: 0.8973
Epoch 9/25
29/29 [==============================] - 5s 185ms/step - loss: 0.3600 - accuracy: 0.8719 - val_loss: 0.2804 - val_accuracy
: 0.8929
Epoch 10/25
29/29 [==============================] - 5s 187ms/step - loss: 0.3892 - accuracy: 0.8675 - val_loss: 0.3385 - val_accuracy
: 0.8705
Epoch 11/25
29/29 [==============================] - 5s 187ms/step - loss: 0.3354 - accuracy: 0.8920 - val_loss: 0.2493 - val_accuracy
: 0.9107
Epoch 12/25
29/29 [==============================] - 5s 184ms/step - loss: 0.3605 - accuracy: 0.8764 - val_loss: 0.3215 - val_accuracy
: 0.8705
Epoch 13/25
29/29 [==============================] - 6s 189ms/step - loss: 0.3638 - accuracy: 0.8697 - val_loss: 0.3255 - val_accuracy
: 0.8795
Epoch 14/25
29/29 [==============================] - 5s 187ms/step - loss: 0.2491 - accuracy: 0.9176 - val_loss: 0.2907 - val_accuracy
: 0.9152
Epoch 15/25
29/29 [==============================] - 6s 191ms/step - loss: 0.3010 - accuracy: 0.8864 - val_loss: 0.2711 - val_accuracy
: 0.8973
```

```
29/29 [==============================] - 6s 191ms/step - loss: 0.3010 - accuracy: 0.8864 - val_loss: 0.2711 - val_accuracy
: 0.8973
Epoch 16/25
29/29 [==============================] - 6s 195ms/step - loss: 0.2868 - accuracy: 0.8964 - val_loss: 0.2230 - val_accuracy
: 0.9241
Epoch 17/25
29/29 [==============================] - 6s 188ms/step - loss: 0.3010 - accuracy: 0.8875 - val_loss: 0.2221 - val_accuracy
: 0.9152
Epoch 18/25
29/29 [==============================] - 6s 188ms/step - loss: 0.2426 - accuracy: 0.9187 - val_loss: 0.2089 - val_accuracy
: 0.9196
Epoch 19/25
29/29 [==============================] - 6s 189ms/step - loss: 0.2280 - accuracy: 0.9198 - val_loss: 0.2108 - val_accuracy
: 0.9152
Epoch 20/25
29/29 [==============================] - 6s 201ms/step - loss: 0.2906 - accuracy: 0.8953 - val_loss: 0.2308 - val_accuracy
: 0.9107
Epoch 21/25
29/29 [==============================] - 6s 200ms/step - loss: 0.2608 - accuracy: 0.9154 - val_loss: 0.2692 - val_accuracy
: 0.9152
Epoch 22/25
29/29 [==============================] - 6s 197ms/step - loss: 0.2301 - accuracy: 0.9198 - val_loss: 0.2757 - val_accuracy
: 0.8973
Epoch 23/25
29/29 [==============================] - 6s 197ms/step - loss: 0.2365 - accuracy: 0.9143 - val_loss: 0.2586 - val_accuracy
: 0.9107
Epoch 24/25
29/29 [==============================] - 6s 193ms/step - loss: 0.2030 - accuracy: 0.9209 - val_loss: 0.2280 - val_accuracy
: 0.9241
Epoch 25/25
29/29 [==============================] - 6s 191ms/step - loss: 0.1987 - accuracy: 0.9298 - val_loss: 0.1994 - val_accuracy
: 0.9286
```

As can be seen from the model summary above, the model reached both an accuracy and validation-accuracy of very close to 93% correct, after going through 25 epochs. Furthermore, both the final categorical_crossentropy loss and categorical_crossentropy validation-loss are very close to being 0.2, of the last epoch.

After applying data augmentation and Dropout, there is less overfitting than before, and training and validation accuracy are closer aligned, as is displayed by the following two graphs:

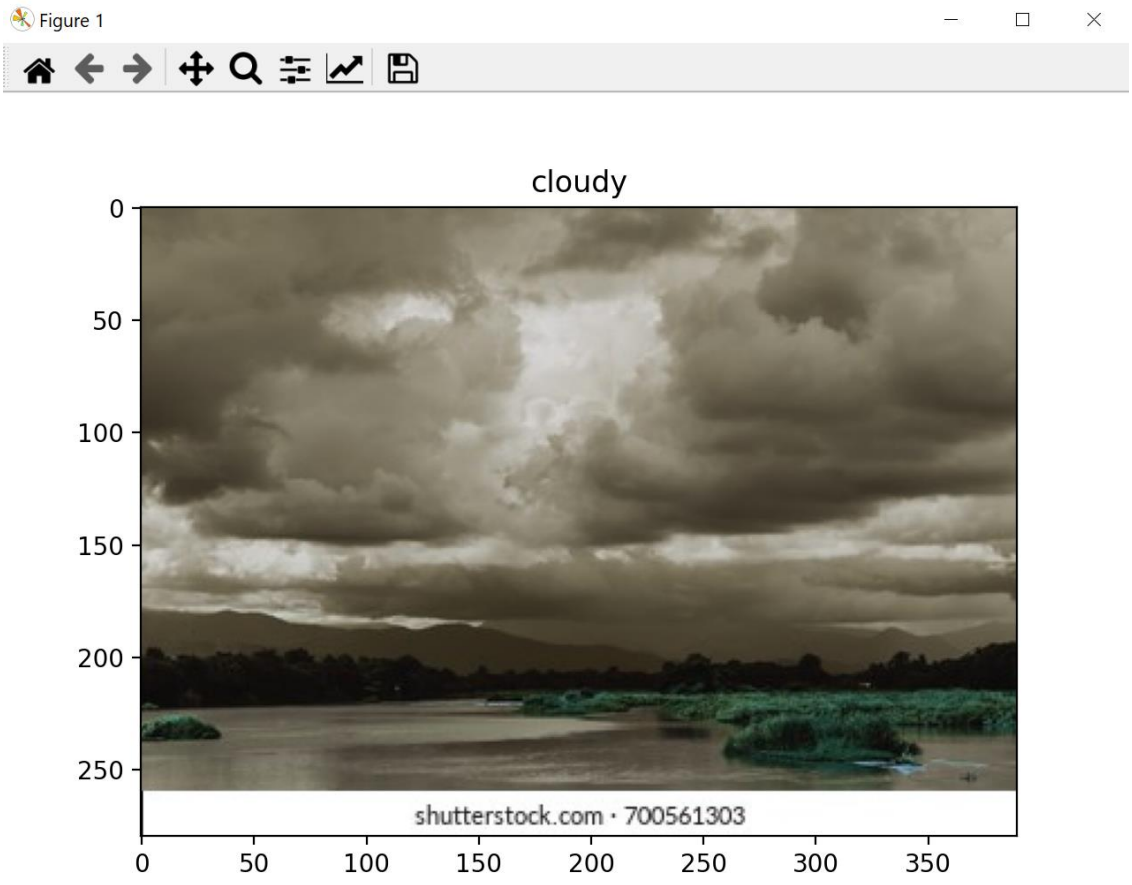


Confusion Matrix for Test Data:

```
CONFUSION MATRIX OF TEST DATA:
[[54.  1.  4.  1.]
 [ 3. 36.  3.  0.]
 [ 2.  0. 43.  0.]
 [ 0.  0.  2. 75.]]
7/7 [==============================] - 0s 29ms/step - loss: 0.1994 - accuracy: 0.9286
PERCENT ACCURACY OF TEST DATA: 92.85714030265808
```

As is demonstrated by the confusion matrix above, after the model has been trained on the training dataset, the model correctly predicts 208 out of the 224 pictures in the test dataset, which resulted in a percent accuracy of about 93% for the model, on the test dataset. Also, the evaluated categorical_crossentropy loss of the test data is around 0.2.

Predicted Output and Corresponding Input Images for The 10 Archived Images:



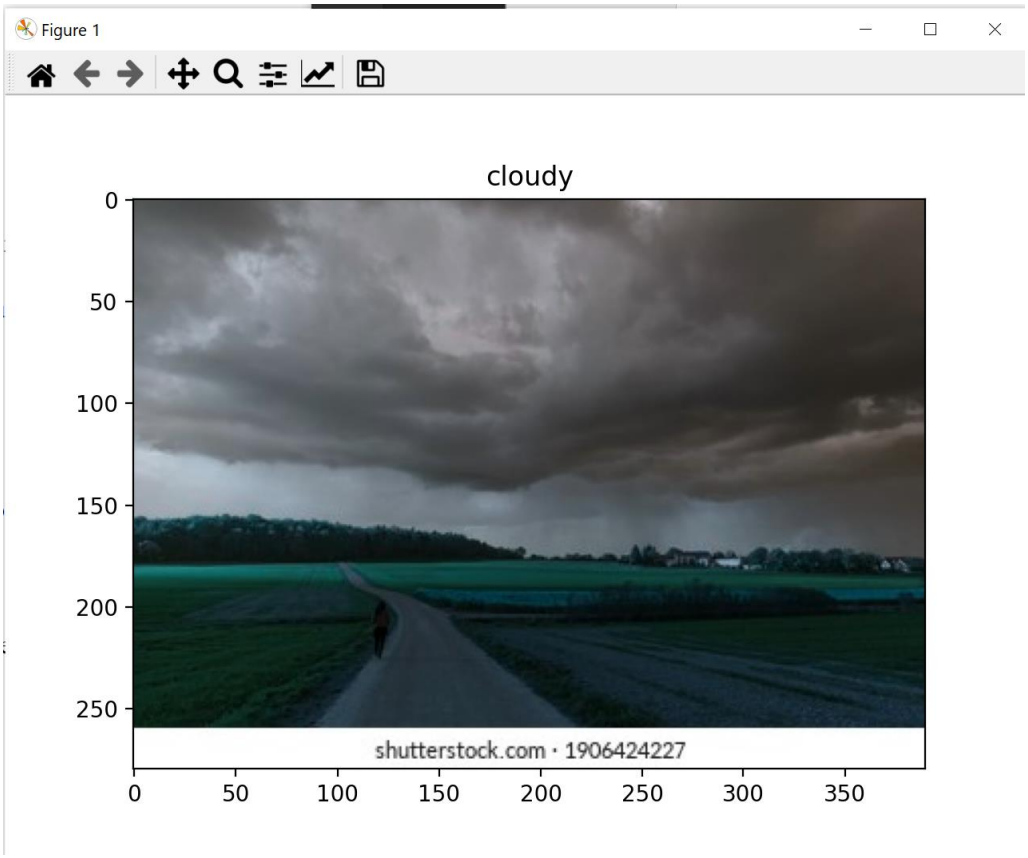


Figure 1
cloudy
shutterstock.com · 1906424227

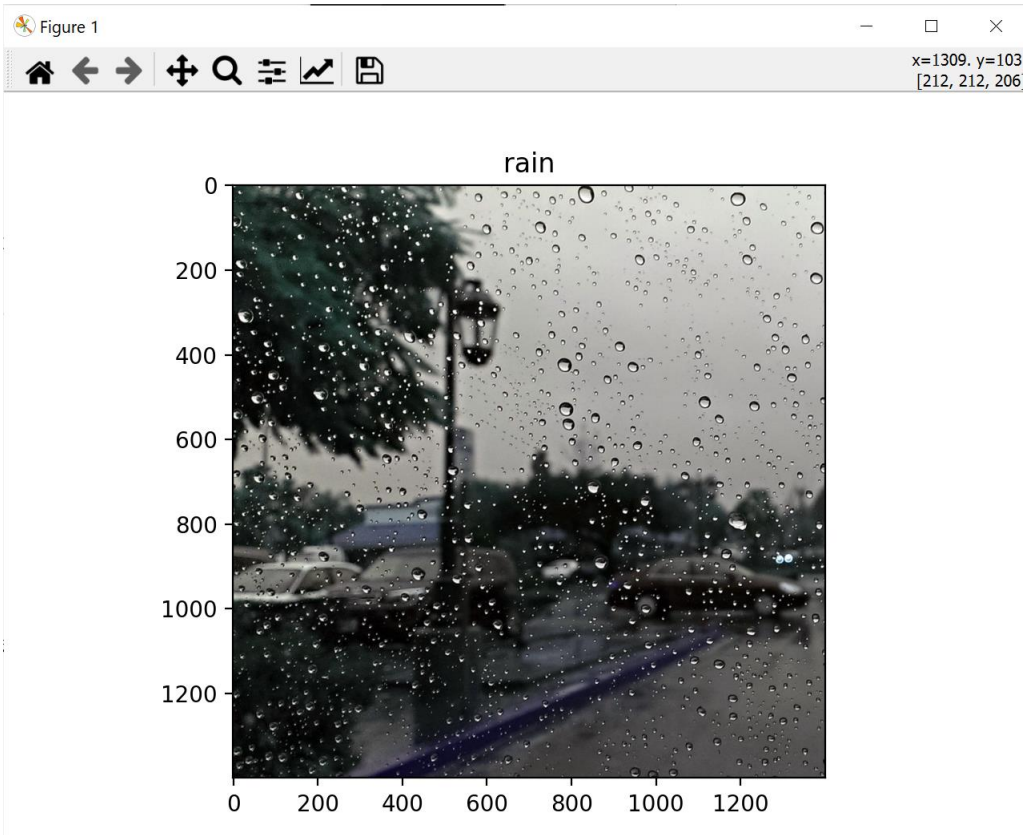


Figure 1
x=1309. y=103
[212, 212, 206
rain

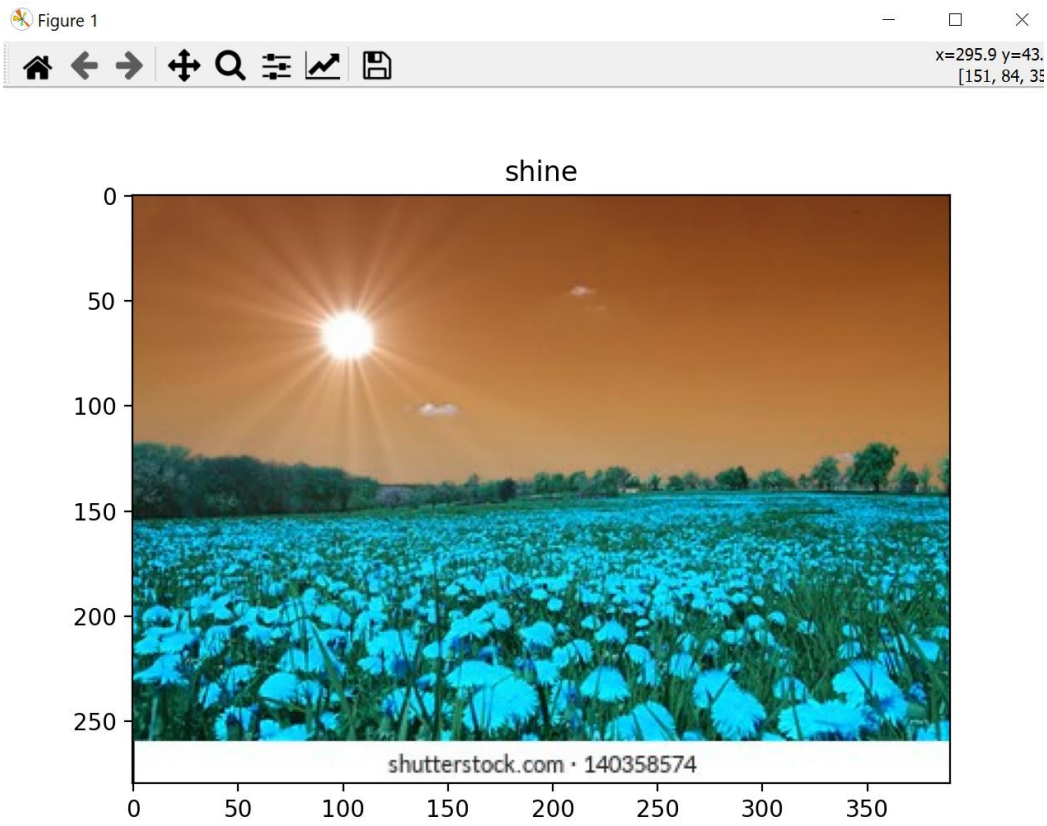


Figure 1
x=295.9 y=43.
[151, 84, 3
shine
0
50
100
150
200
250
shutterstock.com · 140358574
0
50
100
150
200
250
300
350

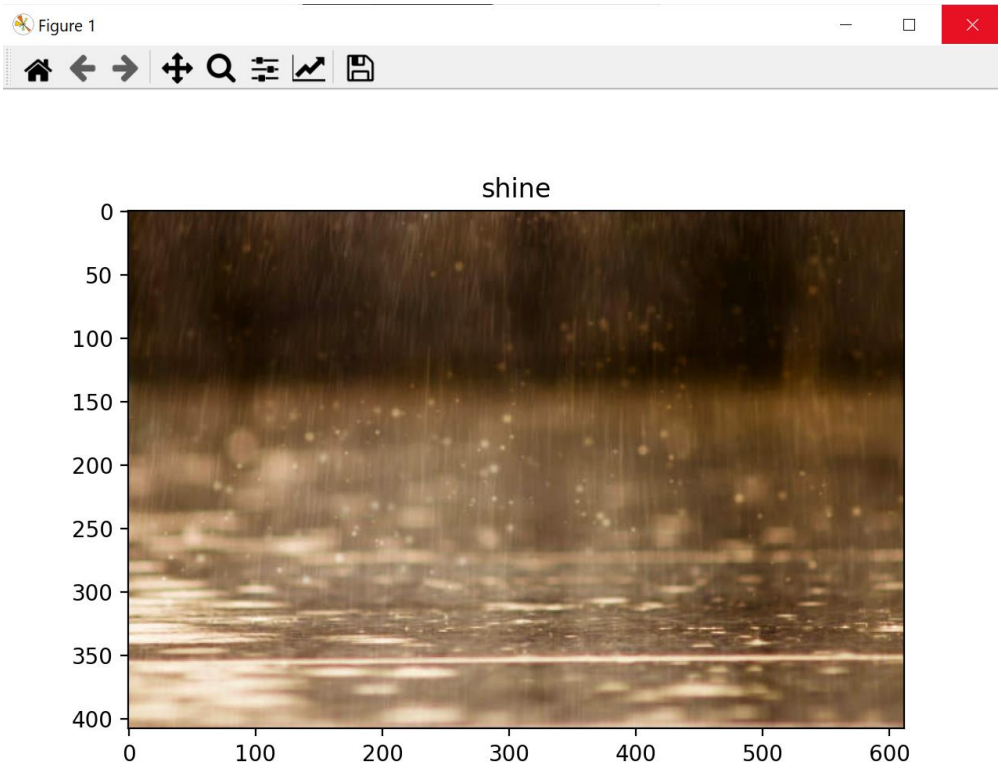


Figure 1
shine
0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600

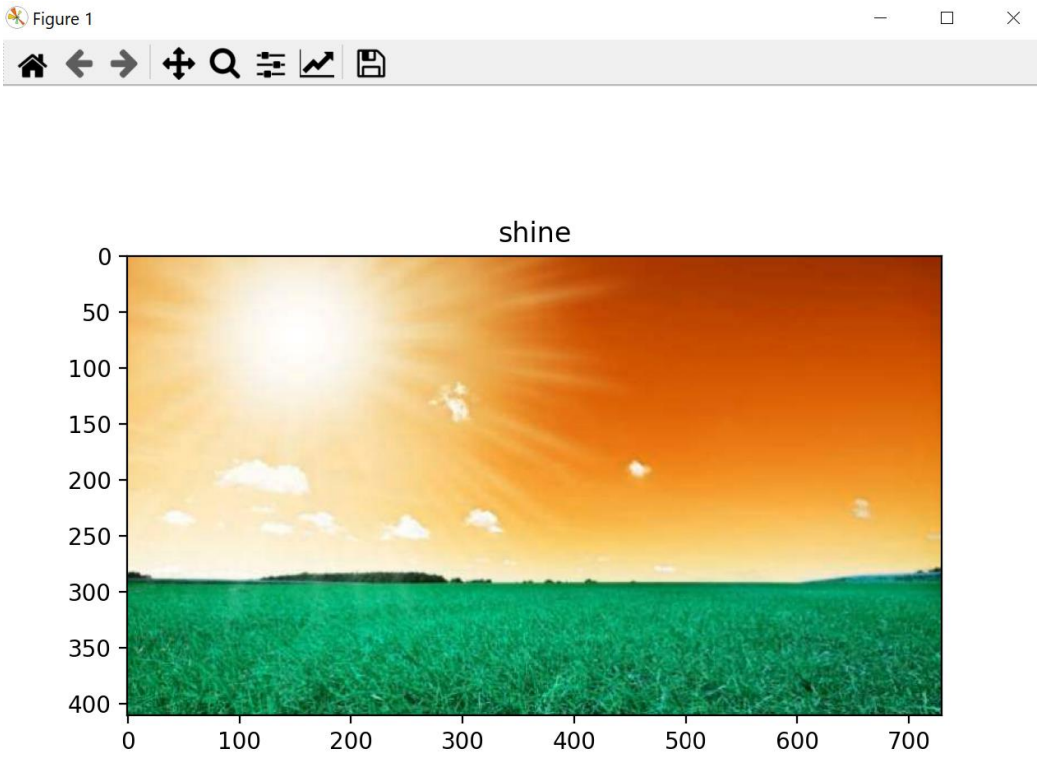


Figure 1
shine
0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600
700

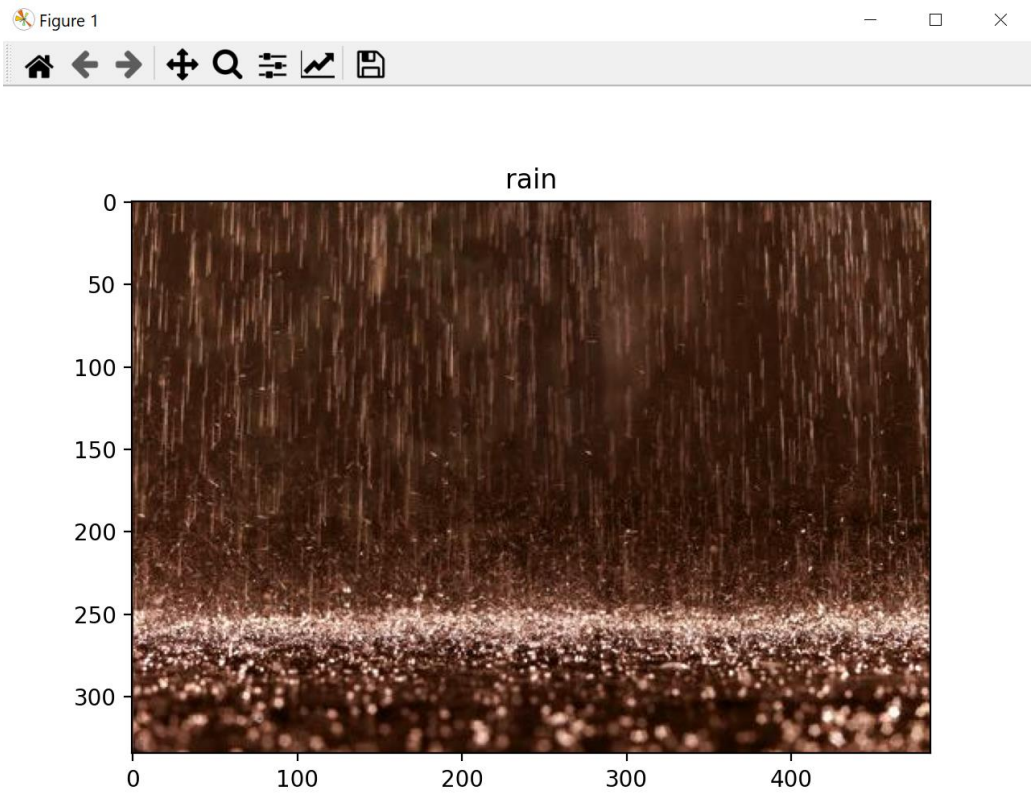


Figure 1
rain
0
50
100
150
200
250
300
0
100
200
300
400

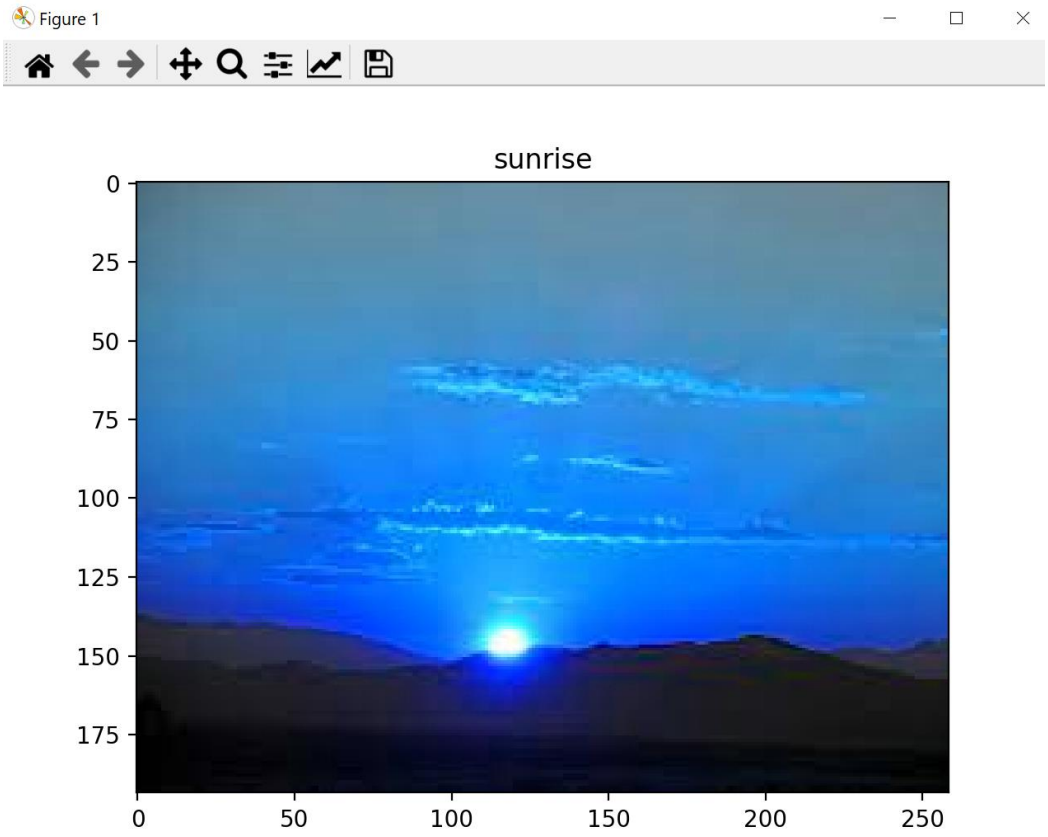


Figure 1
sunrise
0
25
50
75
100
125
150
175
0
50
100
150
200
250

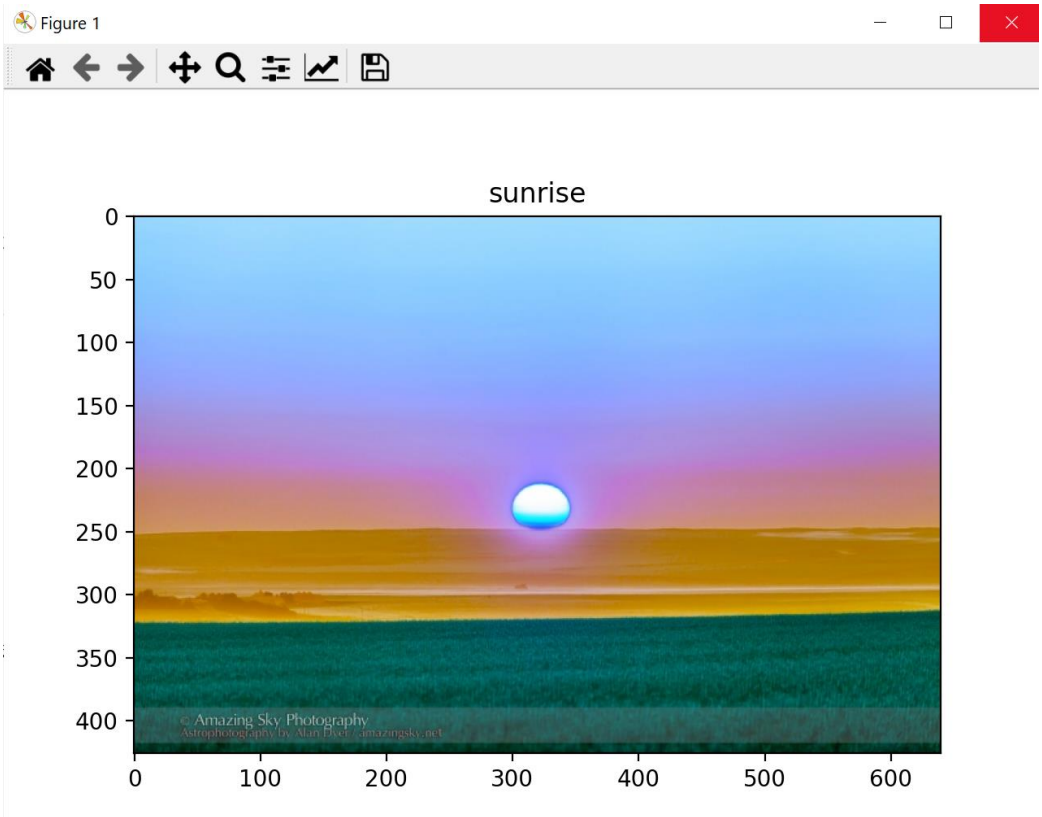
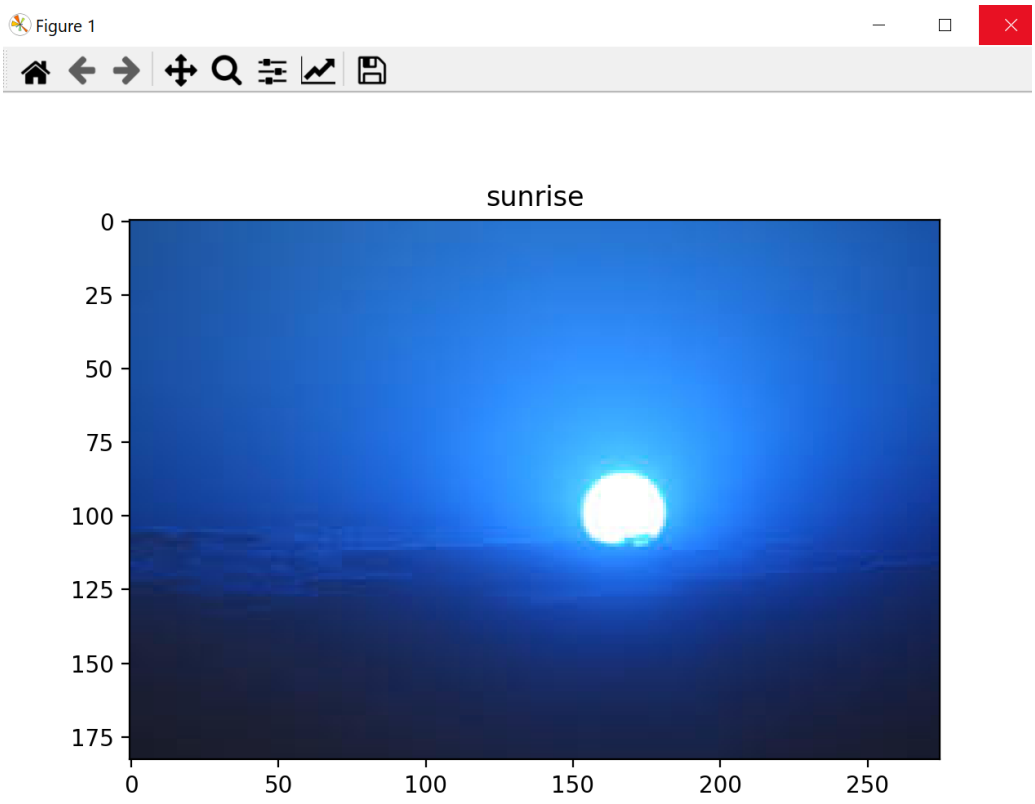


Figure 1
sunrise
0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600
© Amazing Sky Photography



Confusion Matrix for 10 Archived Pictures:

```
CONFUSION MATRIX ON 10 PICS:
[[2. 0. 0. 0.]
 [0. 2. 1. 0.]
 [0. 0. 2. 0.]
 [0. 0. 0. 3.]]
1/1 [==============================] - 0s 115ms/step - loss: 0.2999 - accuracy: 0.9000
PERCENT ACCURACY OF 10 PICS: 89.99999761581421
```

As is demonstrated by the confusion matrix above, after the model has been trained on the training dataset, the model correctly predicts 9 out of my 10 archived pictures, which resulted in a percent accuracy of 90% for the model on my 10 archived pictures. Also, the evaluated categorical_crossentropy loss of the 10 pictures is around 0.3.

Conclusion:

In this lab I completed the following tasks:

1) Downloaded the dataset from the link provided in the Project Handout
2) Read the dataset and applied preprocessing techniques (normalization, resize)
3) Split the dataset into a “80%” training set and a 20% test set

4) Developed a model based on convolutional neural network and trained it for the training data
5) Tested my model's performance on the test dataset
6) Tested my model performance on 10 images from your own archives/camera, and show the output with the corresponding image.
7) Modified my model/approach/training/preprocessing as in iterations to enhance my model's performance

I also learned the following skills:

- Designed a deep neural network architecture for a practical application in image processing
- Implemented a workflow for image processing and classification
- Evaluated results and assessed the model's performance
- Understood libraries such as Tensorflow, OpenCV, numpy in Python as data analytic tools

References:

- *CNN Explainer*. (n.d.). Poloclub.github.io. https://poloclub.github.io/cnn-explainer/
- *Image classification | TensorFlow Core*. (n.d.). TensorFlow. Retrieved December 14, 2021, from https://www.tensorflow.org/tutorials/images/classification#configure_the_dataset_for_performance
- Convolutional Neural Network (CNN. (2019). *Convolutional Neural Network (CNN) | TensorFlow Core*. TensorFlow. https://www.tensorflow.org/tutorials/images/cnn
- ENEL 525 Project Handout. | https://d2l.ucalgary.ca/d2l/le/content/400184/viewContent/5026116/View