assignment-3

May 18, 2023

[1]:

```
!pip install Augmentor
```

```
Collecting Augmentor
  Downloading Augmentor-0.2.12-py2.py3-none-any.whl (38 kB)
Requirement already satisfied: tqdm>=4.9.0 in c:\users\sakth\anaconda3\lib\site-
packages (from Augmentor) (4.64.0)
Requirement already satisfied: Pillow>=5.2.0 in
c:\users\sakth\anaconda3\lib\site-packages (from Augmentor) (9.0.1)
Requirement already satisfied: numpy>=1.11.0 in
c:\users\sakth\anaconda3\lib\site-packages (from Augmentor) (1.22.4)
Requirement already satisfied: colorama in c:\users\sakth\anaconda3\lib\site-
packages (from tqdm>=4.9.0->Augmentor) (0.4.4)
Installing collected packages: Augmentor
Successfully installed Augmentor-0.2.12
```

[6]:

```
import Augmentor

# Specify the path to the directory containing your animal image dataset
data_path = 'C:/Users/sakth/OneDrive/Desktop/animals/animals'

# Create an Augmentor pipeline
pipeline = Augmentor.Pipeline(data_path)

# Add augmentation operations to the pipeline
pipeline.rotate(probability=0.5, max_left_rotation=10, max_right_rotation=10)
pipeline.flip_left_right(probability=0.5)
pipeline.flip_top_bottom(probability=0.5)
pipeline.zoom(probability=0.5, min_factor=1.1, max_factor=1.5)

# Specify the desired number of augmented images
num_augmented_images = 1000

# Generate augmented images
pipeline.sample(num_augmented_images)
```

```
Initialised with 5400 image(s) found.
Output directory set to C:/Users/sakth/OneDrive/Desktop/animals/animals\output.
```

```
Processing <PIL.Image.Image image mode=RGB size=669x446 at 0x281D42A95E0>:
100%|      | 1000/1000 [00:32<00:00, 30.90 Samples/s]
```

[9]:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Define the input shape of the images
input_shape = (64, 64, 3)  # Assuming images are RGB and have a size of 64x64
num_classes=0

# Create a Sequential model
model = Sequential()

# Add the input layer
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',␣
 ↪input_shape=input_shape))

# Add a pooling layer
model.add(MaxPooling2D(pool_size=(2, 2)))

# Add a flatten layer
model.add(Flatten())

# Add hidden layers
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))

# Add the output layer
model.add(Dense(num_classes, activation='softmax'))  # Replace num_classes with␣
 ↪the actual number of output classes

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

# Print the model summary
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 62, 62, 32)        896


 max_pooling2d_2 (MaxPooling  (None, 31, 31, 32)       0
 2D)
 flatten_2 (Flatten)         (None, 30752)             0

 dense_5 (Dense)             (None, 128)               3936384

 dense_6 (Dense)             (None, 64)                8256

 dense_7 (Dense)             (None, 0)                 0

=================================================================
Total params: 3,945,536
Trainable params: 3,945,536
Non-trainable params: 0
_________________________________________________________________
```

[21]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
data = pd.read_csv("House Price India.csv")

# Splitting features and Price variable
X = data.drop('Price', axis=1).values
y = data['Price'].values

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Creating the ANN model
model = Sequential()

# Adding the input layer and the first hidden layer
model.add(Dense(units=64, activation='relu', input_dim=X_train.shape[1]))

# Adding additional hidden layers
model.add(Dense(units=64, activation='relu'))
model.add(Dense(units=32, activation='relu'))

# Adding the output layer
```

```
model.add(Dense(units=1, activation='sigmoid'))

# Compiling the model
model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

# Training the model
model.fit(X_train, y_train, epochs=10, batch_size=32)

# Evaluating the model on the testing set
loss, accuracy = model.evaluate(X_test, y_test)
print("Test loss:", loss)
print("Test accuracy:", accuracy)
```

```
Epoch 1/10
366/366 [==============================] - 1s 2ms/step - loss: -8855980032.0000
- accuracy: 0.0000e+00
Epoch 2/10
366/366 [==============================] - 1s 2ms/step - loss:
-453630164992.0000 - accuracy: 0.0000e+00
Epoch 3/10
366/366 [==============================] - 0s 1ms/step - loss:
-2993069752320.0000 - accuracy: 0.0000e+00
Epoch 4/10
366/366 [==============================] - 0s 969us/step - loss:
-10290401902592.0000 - accuracy: 0.0000e+00
Epoch 5/10
366/366 [==============================] - 0s 963us/step - loss:
-25622485139456.0000 - accuracy: 0.0000e+00
Epoch 6/10
366/366 [==============================] - 0s 1ms/step - loss:
-51752011300864.0000 - accuracy: 0.0000e+00
Epoch 7/10
366/366 [==============================] - 0s 931us/step - loss:
-91405779206144.0000 - accuracy: 0.0000e+00
Epoch 8/10
366/366 [==============================] - 0s 881us/step - loss:
-147669649457152.0000 - accuracy: 0.0000e+00
Epoch 9/10
366/366 [==============================] - 0s 897us/step - loss:
-223086121058304.0000 - accuracy: 0.0000e+00
Epoch 10/10
366/366 [==============================] - 0s 905us/step - loss:
-320986427686912.0000 - accuracy: 0.0000e+00
92/92 [==============================] - 0s 893us/step - loss:
-400258504851456.0000 - accuracy: 0.0000e+00
Test loss: -400258504851456.0
Test accuracy: 0.0
```

[ ]:

```
```