Skill4

2100030910

Sec-23

Model.py

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
class Modelsc:
    def adam(self):
        model2 = tf.keras.Sequential([
            layers.Conv2D(64, (3, 3), padding="valid", input_shape=(224,
224, 3), activation='relu'),
            layers.MaxPooling2D(pool_size=(2, 2)),

            layers.Conv2D(16, (3, 3), padding="valid", activation='relu'),
            layers.MaxPooling2D(pool_size=(2, 2)),

            layers.Conv2D(32, (4, 4), padding="valid", activation='relu'),
            layers.MaxPooling2D(pool_size=(2, 2)),

            layers.Conv2D(128, (4, 4), padding="valid", activation='relu'),
            layers.MaxPooling2D(pool_size=(2, 2)),

            layers.Flatten(),
            layers.Dense(64, activation='relu'),
            layers.Dropout(0.15),
            layers.Dense(1, activation='sigmoid')
        ])


model2.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.01),
                       loss=tf.keras.losses.BinaryCrossentropy(),
                       metrics=[tf.keras.metrics.BinaryAccuracy()])
        return model2

    def sgd(self):
        model3 = tf.keras.Sequential([
            layers.Conv2D(64, (3, 3), padding="valid", input_shape=(224,
224, 3), activation='relu'),
            layers.MaxPooling2D(pool_size=(2, 2)),

            layers.Conv2D(16, (3, 3), padding="valid", activation='relu'),
            layers.MaxPooling2D(pool_size=(2, 2)),

            layers.Conv2D(32, (4, 4), padding="valid", activation='relu'),
            layers.MaxPooling2D(pool_size=(2, 2)),

            layers.Conv2D(128, (4, 4), padding="valid", activation='relu'),
            layers.MaxPooling2D(pool_size=(2, 2)),

            layers.Flatten(),
            layers.Dense(64, activation='relu'),
```

```
            layers.Dropout(0.15),
            layers.Dense(1, activation='sigmoid')
        ])


model3.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.01),
                       loss=tf.keras.losses.BinaryCrossentropy(),
                       metrics=[tf.keras.metrics.BinaryAccuracy()])
        return model3

    def rmsprop(self):
        model4 = tf.keras.Sequential([
            layers.Conv2D(64, (3, 3), padding="valid", input_shape=(224,
224, 3), activation='relu'),
            layers.MaxPooling2D(pool_size=(2, 2)),

            layers.Conv2D(16, (3, 3), padding="valid", activation='relu'),
            layers.MaxPooling2D(pool_size=(2, 2)),

            layers.Conv2D(32, (4, 4), padding="valid", activation='relu'),
            layers.MaxPooling2D(pool_size=(2, 2)),

            layers.Conv2D(128, (4, 4), padding="valid", activation='relu'),
            layers.MaxPooling2D(pool_size=(2, 2)),

            layers.Flatten(),
            layers.Dense(64, activation='relu'),
            layers.Dropout(0.15),
            layers.Dense(1, activation='sigmoid')
        ])


model4.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.01),
                       loss=tf.keras.losses.BinaryCrossentropy(),
                       metrics=[tf.keras.metrics.BinaryAccuracy()])
        return model4
```

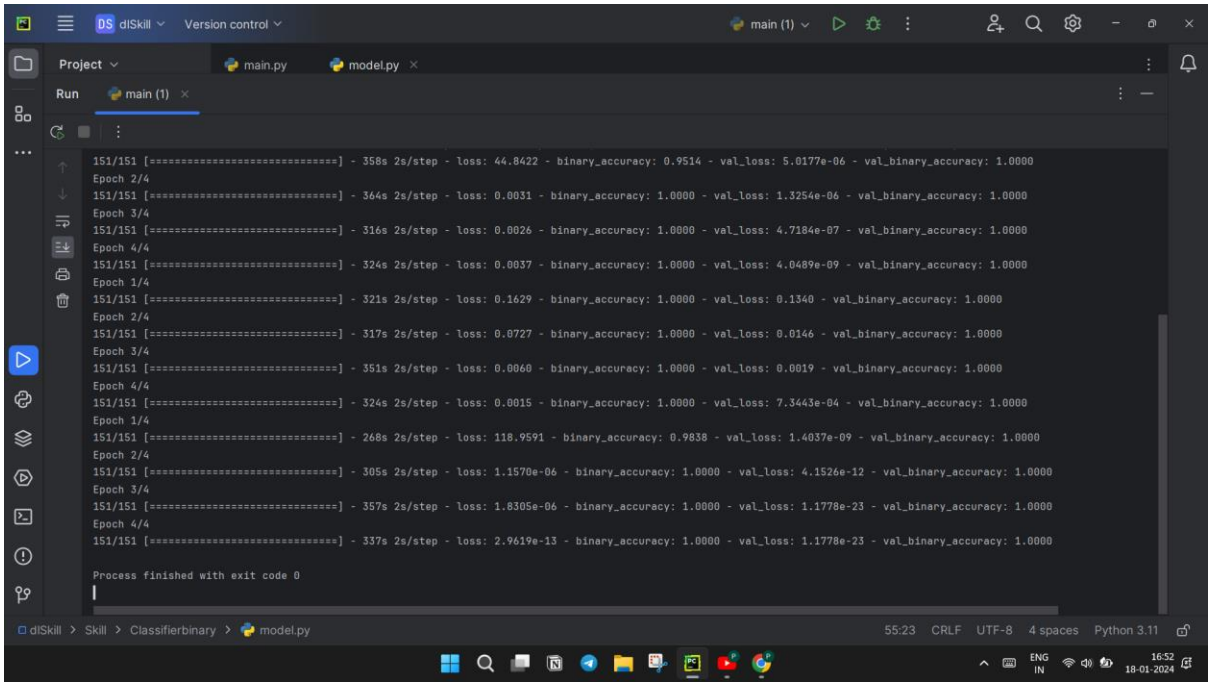
```
import numpy as np
import pandas as pd
import os
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator,
load_img, img_to_array
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense,
GlobalAveragePooling2D, Dropout, Flatten, BatchNormalization
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow import keras
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.applications.efficientnet import preprocess_input
from PIL import Image
import shutil
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import EarlyStopping
import model as mC

train_directory =
r'C:\Users\dell\PycharmProjects\dlSkill\Skill\genData2\train'
```

```
valid_directory =
r'C:\Users\dell\PycharmProjects\dlSkill\Skill\genData2\valid'

cloud_directory =
r'C:\Users\dell\PycharmProjects\dlSkill\Skill\data\cloudy'

cloud_train_directory =
r'C:\Users\dell\PycharmProjects\dlSkill\Skill\genData2\train\cloudy'
cloud_valid_directory =
r'C:\Users\dell\PycharmProjects\dlSkill\Skill\genData2\valid\cloudy'

water_directory = r'C:\Users\dell\PycharmProjects\dlSkill\Skill\data\water'
green_directory =
r'C:\Users\dell\PycharmProjects\dlSkill\Skill\data\green_area'
desert_directory =
r'C:\Users\dell\PycharmProjects\dlSkill\Skill\data\desert'

non_cloud_train_directory =
r'C:\Users\dell\PycharmProjects\dlSkill\Skill\genData2\train\noncloudy'
non_cloud_valid_directory =
r'C:\Users\dell\PycharmProjects\dlSkill\Skill\genData2\valid\noncloudy'


cloud_image_files = [f for f in os.listdir(cloud_directory) if
                     f.lower().endswith(('.jpg', '.jpeg'))]
# Specify a new name format
# You can customize the format according to your needs


water_image_files = [f for f in os.listdir(water_directory) if
                     f.lower().endswith(('.jpg', '.jpeg'))]
# Specify a new name format
# You can customize the format according to your needs


desert_image_files = [f for f in os.listdir(desert_directory) if
                      f.lower().endswith(('.jpg', '.jpeg'))]
# Specify a new name format
# You can customize the format according to your needs
# In this example, we're using a base name and appending an incremental
number


green_image_files = [f for f in os.listdir(green_directory) if
                     f.lower().endswith(('.jpg', '.jpeg'))]


# Loop through the image files and rename them







os.makedirs(train_directory, exist_ok=True)
os.makedirs(valid_directory, exist_ok=True)



cloud_train_files, cloud_valid_files = train_test_split(cloud_image_files,
test_size=0.2, random_state=42)
```

```
for file in cloud_train_files:
    source_file_path = os.path.join(cloud_directory, file)
    destination_file_path = os.path.join(cloud_train_directory, file)
    shutil.copy(source_file_path, destination_file_path)

for file in cloud_valid_files:
    source_file_path = os.path.join(cloud_directory, file)
    destination_file_path = os.path.join(cloud_valid_directory, file)
    shutil.copy(source_file_path, destination_file_path)

water_train_files, water_valid_files = train_test_split(water_image_files,
test_size=0.2, random_state=42)
for file in water_train_files:
    source_file_path = os.path.join(water_directory, file)
    destination_file_path = os.path.join(non_cloud_train_directory, file)
    shutil.copy(source_file_path, destination_file_path)

for file in water_valid_files:
    source_file_path = os.path.join(water_directory, file)
    destination_file_path = os.path.join(non_cloud_valid_directory, file)
    shutil.copy(source_file_path, destination_file_path)

desert_train_files, desert_valid_files =
train_test_split(desert_image_files, test_size=0.2, random_state=42)
for file in desert_train_files:
    source_file_path = os.path.join(desert_directory, file)
    destination_file_path = os.path.join(non_cloud_train_directory, file)
    shutil.copy(source_file_path, destination_file_path)

for file in desert_valid_files:
    source_file_path = os.path.join(desert_directory, file)
    destination_file_path = os.path.join(non_cloud_valid_directory, file)
    shutil.copy(source_file_path, destination_file_path)

green_train_files, green_valid_files = train_test_split(green_image_files,
test_size=0.2, random_state=42)
for file in green_train_files:
    source_file_path = os.path.join(green_directory, file)
    destination_file_path = os.path.join(non_cloud_train_directory, file)
    shutil.copy(source_file_path, destination_file_path)

for file in green_valid_files:
    source_file_path = os.path.join(green_directory, file)
    destination_file_path = os.path.join(non_cloud_valid_directory, file)
    shutil.copy(source_file_path, destination_file_path)

# Creating training image data generator
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_imagenerator = ImageDataGenerator(

    rescale=1.0 / 255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='wrap'

)
```

```
train_generator = train_imagenerator.flow_from_directory(
    train_directory,
    target_size=(224,224),
    batch_size=20,
    class_mode='binary',
    shuffle=True
)

val_imagenerator = ImageDataGenerator(rescale=1.0/255)

validation_generator = val_imagenerator.flow_from_directory(
    valid_directory,
    target_size=(224,224),
    batch_size=20,
    class_mode='binary',
    shuffle=True
)

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
batch_size = 30
image_size = (224, 224)

# Create ImageDataGenerator

# Create a tf.data.Dataset using image_dataset_from_directory
train_data = tf.keras.preprocessing.image_dataset_from_directory(
    train_directory,
    seed=45,
    shuffle=True,
    image_size=image_size,
    batch_size=batch_size
)

val_data = tf.keras.preprocessing.image_dataset_from_directory(
    valid_directory,
    seed=45,
    shuffle=False,
    image_size=image_size,
    batch_size=batch_size
)

class_names = train_data.class_names
print(class_names)

class_names = train_data.class_names

plt.figure(figsize=(12, 8))
for images, labels in train_data.take(1):
    for i in range(4):
        ax = plt.subplot(2, 2, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
        plt.axis("off")

augmented_images, labels = train_generator.next()

plt.figure(figsize=(12, 8))
for i in range(min(6, augmented_images.shape[0])):
    ax = plt.subplot(2, 3, i + 1)
```

```
    plt.imshow(augmented_images[i])
    plt.title(int(np.argmax(labels[i])))  # Convert one-hot encoded label
to integer category
    plt.axis("off")

class_names = train_data.class_names
print(class_names)

clsmC=mC.Modelsc()

model2=clsmC.adam()

h2 = model2.fit(
    train_data,                    # Training data
    epochs=4,                  # Number of training epochs
    batch_size=batch_size,         # Batch size
    validation_data=val_data,    # Early stopping callback
)

model3=clsmC.sgd()

h3 = model3.fit(
    train_data,                    # Training data
    epochs=4,                  # Number of training epochs
    batch_size=batch_size,         # Batch size
    validation_data=val_data,    # Early stopping callback
)

model4=clsmC.rmsprop()

h4 = model4.fit(
    train_data,                    # Training data
    epochs=4,                  # Number of training epochs
    batch_size=batch_size,         # Batch size
    validation_data=val_data,    # Early stopping callback
)

plt.figure(figsize=(10,3))
plt.plot(h2.history['binary_accuracy'])
plt.plot(h3.history['binary_accuracy'])
plt.plot(h4.history['binary_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()
```


dlSkill
Version control
main (1)
Project
main.py
model.py
Run
main (1)
151/151 [==============================] - 358s 2s/step - loss: 44.8422 - binary_accuracy: 0.9514 - val_loss: 5.0177e-06 - val_binary_accuracy: 1.0000
Epoch 2/4
151/151 [==============================] - 364s 2s/step - loss: 0.0031 - binary_accuracy: 1.0000 - val_loss: 1.3254e-06 - val_binary_accuracy: 1.0000
Epoch 3/4
151/151 [==============================] - 316s 2s/step - loss: 0.0026 - binary_accuracy: 1.0000 - val_loss: 4.7184e-07 - val_binary_accuracy: 1.0000
Epoch 4/4
151/151 [==============================] - 324s 2s/step - loss: 0.0037 - binary_accuracy: 1.0000 - val_loss: 4.0489e-09 - val_binary_accuracy: 1.0000
Epoch 1/4
151/151 [==============================] - 321s 2s/step - loss: 0.1629 - binary_accuracy: 1.0000 - val_loss: 0.1340 - val_binary_accuracy: 1.0000
Epoch 2/4
151/151 [==============================] - 317s 2s/step - loss: 0.0727 - binary_accuracy: 1.0000 - val_loss: 0.0146 - val_binary_accuracy: 1.0000
Epoch 3/4
151/151 [==============================] - 351s 2s/step - loss: 0.0060 - binary_accuracy: 1.0000 - val_loss: 0.0019 - val_binary_accuracy: 1.0000
Epoch 4/4
151/151 [==============================] - 324s 2s/step - loss: 0.0015 - binary_accuracy: 1.0000 - val_loss: 7.3443e-04 - val_binary_accuracy: 1.0000
Epoch 1/4
151/151 [==============================] - 268s 2s/step - loss: 118.9591 - binary_accuracy: 0.9838 - val_loss: 1.4037e-09 - val_binary_accuracy: 1.0000
Epoch 2/4
151/151 [==============================] - 305s 2s/step - loss: 1.1570e-06 - binary_accuracy: 1.0000 - val_loss: 4.1526e-12 - val_binary_accuracy: 1.0000
Epoch 3/4
151/151 [==============================] - 357s 2s/step - loss: 1.8305e-06 - binary_accuracy: 1.0000 - val_loss: 1.1778e-23 - val_binary_accuracy: 1.0000
Epoch 4/4
151/151 [==============================] - 337s 2s/step - loss: 2.9619e-13 - binary_accuracy: 1.0000 - val_loss: 1.1778e-23 - val_binary_accuracy: 1.0000
Process finished with exit code 0
dlSkill > Skill > Classifierbinary > model.py
55:23
CRLF
UTF-8
4 spaces
Python 3.11
ENG
IN
16:52
18-01-2024

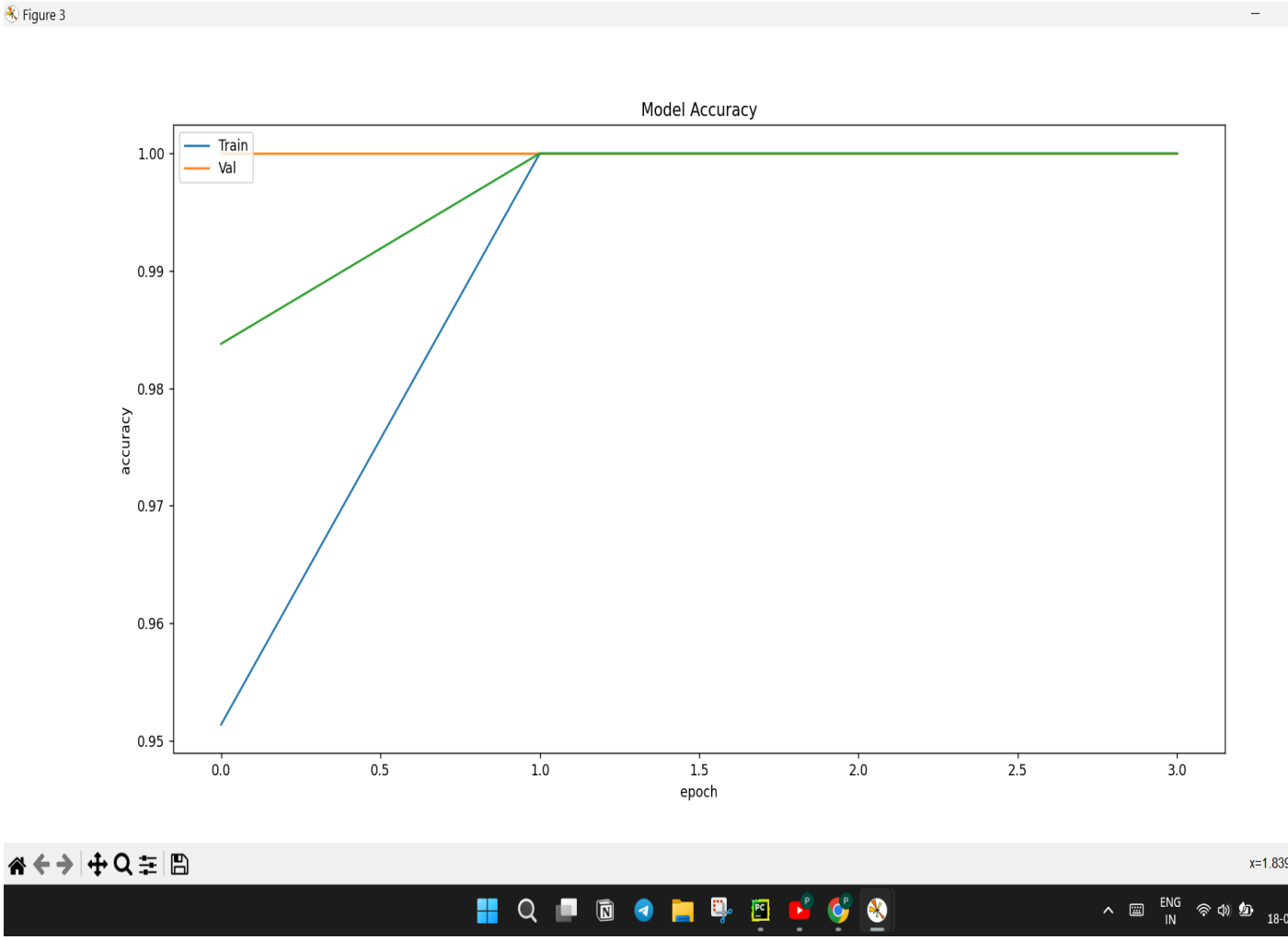


Figure 3
Model Accuracy
Train
Val
accuracy
epoch
1.00
0.99
0.98
0.97
0.96
0.95
0.0
0.5
1.0
1.5
2.0
2.5
3.0
x=1.839
ENG
IN