:
:
:

2

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPool2D, Flatten, Dense,
GlobalAveragePooling2D, Dropout
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

train_dir="3_food_classes/train/"
test_dir="3_food_classes/test/"

# Create ImageDataGenerator for test (no augmentation, just rescaling and test validation split)

train_datagen = ImageDataGenerator(
  rescale=1./255,
  validation_split=0.2  # Use this to split the training data into train and validation sets
)



# Create ImageDataGenerator for test (no augmentation, just rescaling)
test_datagen = ImageDataGenerator(rescale=1./255)

# Training data generator
train_data_gen = train_datagen.flow_from_directory(
  train_dir,
  target_size=(64, 64),
  batch_size=32,
  class_mode='categorical',
  subset='training'  # Use the training subset
)

# Validation data generator
val_data_gen = train_datagen.flow_from_directory(
  train_dir,  # Note: still pointing to the training directory
  target_size=(64, 64),
  batch_size=32,
  class_mode='categorical',
  subset='validation'  # Use the validation subset
```

```
)


# Test data generator
test_data_gen = test_datagen.flow_from_directory(
  test_dir,
  target_size=(64, 64),
  batch_size=32,
  class_mode='categorical',
  shuffle=False  # Do not shuffle test data
)

model = Sequential()
model.add(Conv2D(4,(3,3),activation='relu',input_shape=(64,64,3)))
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Conv2D(16,(3,3),activation='relu'))
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Conv2D(32,(3,3),activation='relu'))
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Flatten())
model.add(Dense(12, activation='relu'))
model.add(Dense(3, activation='softmax'))

model.summary()

optimizer = SGD(learning_rate=0.01)

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

history = model.fit(train_data_gen, epochs=5)

val_loss, val_acc = model.evaluate(test_data_gen)
print("Validation loss: ", val_loss)
print("Validation accuracy: ", val_acc)
```

3

```
#--Apply Data Augmentation

train_datagen = ImageDataGenerator(
```

```
    rescale=1./255,
    rotation_range=30,
    height_shift_range=0.2,
    width_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.2  # Use this to split the training data into train and validation sets
)

# Create ImageDataGenerator for test (no augmentation, just rescaling)
test_datagen = ImageDataGenerator(rescale=1./255)

# Training data generator
train_data_gen = train_datagen.flow_from_directory(
    train_dir,
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical',
    subset='training'  # Use the training subset
)

# Test data generator
test_data_gen = test_datagen.flow_from_directory(
    test_dir,
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical',
    shuffle=False  # Do not shuffle test data
)

# Validation data generator
val_data_gen = train_datagen.flow_from_directory(
    train_dir,  # Note: still pointing to the training directory
    target_size=(64, 64),
    batch_size=32,
    class_mode='categorical',
    subset='validation'  # Use the validation subset
)

model_imp = Sequential()
model_imp.add(Conv2D(16, (3,3), activation='relu', input_shape=(64,64,3)))
```

```
model_imp.add(MaxPool2D(pool_size=(2,2)))

model_imp.add(Conv2D(32, (3,3), activation='relu'))
model_imp.add(MaxPool2D(pool_size=(2,2)))

model_imp.add(Conv2D(64, (3,3), activation='relu'))
model_imp.add(MaxPool2D(pool_size=(2,2)))

model_imp.add(Conv2D(256, (3,3), activation='relu'))
model_imp.add(MaxPool2D(pool_size=(2,2)))

model_imp.add(Flatten())
model_imp.add(Dropout(0.5))
model_imp.add(Dense(12, activation='relu'))
model_imp.add(Dense(3, activation='softmax'))

model_imp.summary()

optimizer = Adam(learning_rate=0.001)

model_imp.compile(
  optimizer=optimizer,
  loss='categorical_crossentropy',
  metrics=['accuracy']
)


callback_imp = ModelCheckpoint(
  filepath='best_model_weights.keras',
  save_best_only=True,
  monitor='val_loss',
  mode='max'
)

early_stopping = EarlyStopping(
  monitor='val_loss',
  patience=5,
  restore_best_weights=True
)

history_imp = model_imp.fit(
  train_data_gen,
  epochs=5,
```

```
  validation_data=val_data_gen,
  callbacks=[callback_imp, early_stopping]
)

val_loss, val_acc = model_imp.evaluate(test_data_gen)
print("Validation loss: ", val_loss)
print("Validation accuracy: ", val_acc)
```

4

```
train_dir="3_food_classes/train/"
test_dir="3_food_classes/test/"

import tensorflow as tf

# Load the pretrained base model from the SavedModel directory
Base_model = tf.keras.models.load_model("base_model")

# Check the summary of the loaded model to verify it has been loaded correctly
Base_model.summary()

model = Sequential()
model.add(Base_model)
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dense(3, activation ='softmax'))

model.summary()


datagen = ImageDataGenerator(
  rescale=1./255,
  validation_split=0.2,
  rotation_range=30,
  height_shift_range=0.2,
  width_shift_range=0.2,
  shear_range=0.2,
  zoom_range=0.2,
  horizontal_flip=True,
  fill_mode='nearest'
)
```

```
train_data_gen = datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    class_mode='categorical',
    subset='training'
)

val_data_gen = datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    class_mode='categorical',
    subset='validation'
)

test_data_gen = datagen.flow_from_directory(
    test_dir,
    target_size=(224,224),
    class_mode='categorical',

)

early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

checkpoint = ModelCheckpoint("best_model.keras", monitor='val_loss', save_best_only=True)

optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

history = model.fit(
    train_data_gen,
    epochs=5,
    validation_data=val_data_gen,
    batch_size=32,
    callbacks=[early_stopping, checkpoint]
)

val_loss, val_acc = model.evaluate(test_data_gen)
print(val_loss, val_acc)
```

```
# Define directories and constants
image_dir = 'C:/...../PES/Sem2/ESA/DL/.../Unet_Dataset/images'
mask_dir = 'C:/..../PES/Sem2/ESA/DL/.../Unet_Dataset/MASKS_BW'
IMG_SIZE = 150
IMG_HT = 128
IMG_WT = 128
IMG_CH = 3
MASK_CH = 1

# Initialize empty arrays for images and masks
img = np.zeros((IMG_SIZE, IMG_HT, IMG_WT, IMG_CH), dtype=np.float32)
mask = np.zeros((IMG_SIZE, IMG_HT, IMG_WT, MASK_CH), dtype=np.float32)

# Get sorted list of image and mask files
img_files = sorted(os.listdir(image_dir))
mask_files = sorted(os.listdir(mask_dir))


for i in range(IMG_SIZE):
   # Read and process image
   img_path = os.path.join(image_dir, img_files[i])
   image = cv2.imread(img_path)
   image_resized = cv2.resize(image, (IMG_HT, IMG_WT))
   img[i] = image_resized / 255.0

   # Read and process mask
   mask_path = os.path.join(mask_dir, mask_files[i])
   mask_image = cv2.imread(mask_path, cv2.IMREAD_GRAYSCALE)
   mask_resized = cv2.resize(mask_image, (IMG_HT, IMG_WT))
   mask[i] = np.expand_dims(mask_resized, axis=-1) / 255.0


xtrain, xtest, ytrain, ytest = train_test_split(img, mask, test_size=0.3)


pretrained_unet = Unet(
   backbone_name='resnet34',
   #encoder_weights='imagenet',
   input_shape=(IMG_HT, IMG_WT, IMG_CH),
   classes=MASK_CH,
```

```
    activation='sigmoid'
)

pretrained_unet.summary()


pretrained_unet.compile(
    optimizer='adam',
    loss=bce_jaccard_loss,
    metrics=[iou_score]
)


history = pretrained_unet.fit(
    xtrain,
    ytrain,
    batch_size=32,
    epochs=5,
    validation_data=(xtest, ytest)
)

val_loss, val_iou = pretrained_unet.evaluate(xtest, ytest)
(val_loss, val_iou)
```