This script implements **DeepFish classification using VGG16, ResNet50, and EfficientNetB0** with and without data augmentation. Here's what it does step by step:

**1. Import Required Libraries**

* Uses **TensorFlow and Keras** for deep learning models.
* Uses **Matplotlib, Seaborn, and Scikit-learn** for evaluation metrics.

**2. Load and Preprocess the Dataset**

**A computer screen with white text

AI-generated content may be incorrect.**

* Defines the paths for **training, validation, and test datasets**.
* Sets image parameters (**224x224 pixels, batch size 32**).
* Uses ImageDataGenerator to **normalize pixel values (rescale=1./255)**.

**3. Define a Function to Create Models**

* Loads **pre-trained models** (VGG16, ResNet50, EfficientNetB0).
* Adds:
  + **Flatten layer**
  + **Dense layer (256 neurons, ReLU activation)**
  + **Dropout layer (0.5)**
  + **Output layer (Softmax activation for multi-class classification)**.
* Compiles the model with **Adam optimizer, categorical cross-entropy loss, and evaluation metrics**:
  + Accuracy
  + Precision
  + Recall
  + AUC (Area Under Curve)
  + Top-K Categorical Accuracy

**4. Train Models Without Augmentation**

* Loops through each model (VGG16, ResNet50, EfficientNetB0) and **trains them on the DeepFish dataset**.

**5. Apply Data Augmentation**

* Uses ImageDataGenerator to **enhance dataset variability**:
  + Rotation
  + Width/Height shift
  + Shear transformation
  + Zooming
  + Horizontal flip
* Trains models again **with augmentation**.

**6. Evaluate Models on Test Data**

* Loops through each model to evaluate **test accuracy**.

**7. Generate Confusion Matrix & Classification Report**

* Computes predictions on the test set.
* Displays a **confusion matrix** using Seaborn.
* Prints a **classification report** with precision, recall, and F1-score.

**🔹 Key Features**

Uses **VGG16, ResNet50, EfficientNetB0** (pre-trained on ImageNet).  
Implements **data augmentation** to improve generalization.  
Tracks **multiple evaluation metrics** beyond accuracy.  
Provides a **confusion matrix & classification report** for better analysis.

**Purpose of Epochs in Training**

An **epoch** is one complete pass through the entire training dataset. In your code, epochs=10 means that the model will go through the dataset **10 times** to update its weights and learn patterns.

**Why Are Multiple Epochs Used?**

1. **Better Learning** – The model refines its weights multiple times to improve accuracy.
2. **Convergence** – With each epoch, the loss decreases, leading to better predictions.
3. **Avoid Underfitting** – A single pass (epoch) might not be enough for the model to learn useful features.

**What Happens During Each Epoch?**

* The model **processes all training data** in batches (size = 32 in your case).
* It **computes loss** and **updates weights** using **backpropagation**.
* It evaluates performance on the **validation set** to check generalization.

**How Many Epochs to Use?**

* Too few → **Underfitting** (model doesn't learn enough).
* Too many → **Overfitting** (model memorizes training data but fails on new data).
* Use **Early Stopping** (stop training when validation loss stops improving).

Would you like to adjust the number of epochs or add early stopping? 🚀

**Checking for Vanishing Gradient Issue**

Vanishing gradients occur when the gradients (used to update the weights in backpropagation) become too small, slowing or stopping the learning process. This is more common in deep networks.

**How to Check for Vanishing Gradients in Your Model?**

**1. Monitor Training Loss and Validation Loss**

* If the loss **stagnates near zero** but accuracy does not improve, gradients may be too small.
* Solution: Use **gradient clipping** or a different activation function (like LeakyReLU instead of ReLU).

**2. Check Gradients Directly**

You can print gradient values during training:

import tensorflow as tf

@tf.function

def get\_gradients(model, data):

with tf.GradientTape() as tape:

predictions = model(data, training=True)

loss = tf.keras.losses.categorical\_crossentropy(data[1], predictions)

gradients = tape.gradient(loss, model.trainable\_variables)

return gradients

sample\_batch = next(iter(train\_generator))

gradients = get\_gradients(model, sample\_batch)

# Print the average gradient magnitude for each layer

for i, grad in enumerate(gradients):

if grad is not None:

print(f"Layer {i}: {tf.reduce\_mean(tf.abs(grad))}")

**If gradients are close to zero** for most layers, the vanishing gradient problem exists.

**How to Fix Vanishing Gradients?**

✅ **Use Batch Normalization**  
BatchNorm helps stabilize gradients and speeds up training.

✅ **Use a Lower Learning Rate**  
Too high a learning rate can cause weights to get stuck in low-gradient regions.

✅ **Use Gradient Clipping**  
Prevents gradients from becoming too small:

optimizer = tf.keras.optimizers.Adam(learning\_rate=0.0001, clipnorm=1.0)

✅ **Use Activation Functions That Avoid Vanishing Gradients**

* Instead of ReLU, try **LeakyReLU** or **ELU**:
* from tensorflow.keras.layers import LeakyReLU
* top\_model = Dense(256)(top\_model)
* top\_model = LeakyReLU()(top\_model)

Would you like me to modify your code to check gradients and apply fixes? 🚀

Questions:

the dataset contains train, test and valid folders each contains images and labels folders in them. these images need to be used for identifying identical fish and group them this datasets includes photos from deepfish expecting to identify the number of similar fishes unfortunately i can not train it the lable of those fishes but there are images which doesn't contain any fish in it, so how am i going to proceed forward with VGG16, resnet50 and EfficientNet

Since your dataset consists of images categorized into folders but does **not have explicit labels for individual fish species**, you need to modify your approach. Instead of classification, you can use **clustering (unsupervised learning)** or **feature extraction + similarity matching** to group similar fish. Here’s how you can proceed:

**1️⃣ Modify the Approach: Feature Extraction for Similarity**

Instead of training a classifier, use **VGG16, ResNet50, and EfficientNet** as **feature extractors**. The extracted features can then be used to:

* **Cluster similar images** (e.g., K-Means clustering)
* **Find the most similar images** to a given input (e.g., cosine similarity)

**2️⃣ Update the Code for Feature Extraction**

Modify your existing code to **extract features instead of classifying images**.

**🔹 Load Pretrained Models for Feature Extraction**

Replace the create\_model function with this:

from tensorflow.keras.models import Model

def get\_feature\_extractor(base\_model):

base\_model.trainable = False # Freeze convolutional layers

feature\_extractor = Model(inputs=base\_model.input, outputs=base\_model.output)

return feature\_extractor

**3️⃣ Extract Features from Images**

Use the model to **convert images into feature vectors**.

import numpy as np

from tensorflow.keras.preprocessing.image import img\_to\_array, load\_img

def extract\_features(image\_path, model):

img = load\_img(image\_path, target\_size=(224, 224)) # Resize image

img\_array = img\_to\_array(img)

img\_array = np.expand\_dims(img\_array, axis=0) # Add batch dimension

img\_array = img\_array / 255.0 # Normalize

features = model.predict(img\_array)

return features.flatten() # Convert to 1D vector

**4️⃣ Cluster Similar Images Using K-Means**

To group similar fish images, apply **K-Means clustering** to the extracted feature vectors.

from sklearn.cluster import KMeans

import os

# Load feature extractor

model\_vgg = get\_feature\_extractor(VGG16(weights='imagenet', include\_top=False, input\_shape=(224, 224, 3)))

# Extract features for all images

image\_paths = [os.path.join(test\_dir, fname) for fname in os.listdir(test\_dir)]

feature\_list = np.array([extract\_features(img, model\_vgg) for img in image\_paths])

# Apply K-Means clustering

num\_clusters = 5 # Adjust based on dataset size

kmeans = KMeans(n\_clusters=num\_clusters, random\_state=42)

clusters = kmeans.fit\_predict(feature\_list)

# Print clustering results

for i, img\_path in enumerate(image\_paths):

print(f"Image {img\_path} belongs to cluster {clusters[i]}")

**5️⃣ Find the Most Similar Images**

To find the **top N most similar fish images**, use **cosine similarity**.

from sklearn.metrics.pairwise import cosine\_similarity

# Compute similarity between all images

similarity\_matrix = cosine\_similarity(feature\_list)

# Function to get top N similar images

def get\_similar\_images(image\_index, top\_n=5):

similar\_indices = np.argsort(-similarity\_matrix[image\_index])[:top\_n]

return [image\_paths[i] for i in similar\_indices]

# Example usage

image\_index = 0 # Change this index to test different images

print(f"Top similar images for {image\_paths[image\_index]}:")

print(get\_similar\_images(image\_index))

**6️⃣ Handling Images Without Fish**

* For images with **no fish**, the feature vectors should have **low similarity scores** with fish-containing images.
* You can set a **similarity threshold** to filter them out:
* threshold = 0.2 # Adjust based on observations
* if max(similarity\_matrix[image\_index]) < threshold:
* print("This image likely contains no fish.")

**✅ Summary of Changes**

1. **Use VGG16, ResNet50, and EfficientNet for feature extraction** instead of classification.
2. **Convert images into feature vectors**.
3. **Apply K-Means clustering** to group similar fish images.
4. **Use cosine similarity** to find identical fish images.
5. **Filter out empty images** by setting a similarity threshold.

Would you like to implement this step-by-step in your existing code? 🚀A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.import os

import matplotlib.pyplot as plt

import cv2

# Define dataset paths

base\_dir = r"C:\Users\w10185657\DeepFish\datasets"

train\_img\_dir = os.path.join(base\_dir, "train", "images")

train\_label\_dir = os.path.join(base\_dir, "train", "labels")

valid\_img\_dir = os.path.join(base\_dir, "valid", "images")

valid\_label\_dir = os.path.join(base\_dir, "valid", "labels")

test\_img\_dir = os.path.join(base\_dir, "test", "images")

test\_label\_dir = os.path.join(base\_dir, "test", "labels")

'''

# List some files to verify

print("Train Images:", os.listdir(train\_img\_dir)[:5])

print("Train Labels:", os.listdir(train\_label\_dir)[:5])

# Select a sample image from training set

sample\_image = os.listdir(train\_img\_dir)[0]

image\_path = os.path.join(train\_img\_dir, sample\_image)

# Load and display the image

image = cv2.imread(image\_path)

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

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

plt.imshow(image)

plt.axis("off")

plt.title("Sample Image from Train Set")

plt.show()

sample\_label = os.listdir(train\_label\_dir)[0]  # Get a sample label file

label\_path = os.path.join(train\_label\_dir, sample\_label)

# Read label file

with open(label\_path, "r") as file:

    label\_data = file.readlines()

print("Sample Label Data:", label\_data)

'''

A screenshot of a computer

AI-generated content may be incorrect.# Pick a sample label file

'''import os

from code\_train1 import train\_label\_dir

sample\_label\_file = os.listdir(train\_label\_dir)[0]  # Get first label file

label\_path = os.path.join(train\_label\_dir, sample\_label\_file)

# Read and print label file contents

try:

    with open(label\_path, "r", encoding="utf-8") as file:

        label\_data = file.readlines()

    if not label\_data:

        print(f"❌ Label file {sample\_label\_file} is empty!")

    else:

        print(f"✅ Label File: {sample\_label\_file}")

        for line in label\_data[:10]:  # Print first 10 lines

            print(line.strip())

except Exception as e:

    print(f"❌ Error reading label file: {e}")

'''

import cv2

import os

import matplotlib.pyplot as plt

# Set your directories

train\_img\_dir = r"C:\Users\w10185657\DeepFish\datasets\train\images"

train\_label\_dir = r"C:\Users\w10185657\DeepFish\datasets\train\labels"

# Pick a sample image

sample\_image = "7117\_Caranx\_sexfasciatus\_juvenile\_f000000.jpg"

label\_file = "7117\_Caranx\_sexfasciatus\_juvenile\_f000000.txt"

# Load image

image\_path = os.path.join(train\_img\_dir, sample\_image)

image = cv2.imread(image\_path)

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)  # Convert BGR to RGB

# Load labels

label\_path = os.path.join(train\_label\_dir, label\_file)

with open(label\_path, "r") as file:

    label\_data = file.readlines()

# Get image dimensions

img\_h, img\_w, \_ = image.shape

# Draw bounding boxes

for line in label\_data:

    data = line.strip().split()

    class\_id, x\_center, y\_center, width, height = map(float, data)

    # Convert normalized values to pixel values

    x1 = int((x\_center - width / 2) \* img\_w)

    y1 = int((y\_center - height / 2) \* img\_h)

    x2 = int((x\_center + width / 2) \* img\_w)

    y2 = int((y\_center + height / 2) \* img\_h)

    # Draw rectangle

    cv2.rectangle(image, (x1, y1), (x2, y2), (255, 0, 0), 2)

# Show image with bounding boxes

plt.imshow(image)

plt.axis("off")

plt.show()

A computer screen with a picture of a fish

AI-generated content may be incorrect.pip install torch torchvision

pip install yolov5 # For YOLOv5git clone https://github.com/ultralytics/yolov5.git

cd yolov5

pip install -r requirements.txt

Great! Now that the bounding boxes are working correctly, let's move on to the next steps for training your model on the DeepSea Fish Detection task. I'll guide you through setting up your **model** and **training pipeline**.

**📚 Next Steps:**

**1. Organize Data for Training**

We need to make sure the data is structured properly for training. Typically, for object detection tasks, you will have:

* **Images**: Containing the fish in various conditions.
* **Labels**: Containing bounding box coordinates for each image.

**Folder structure should look like:**

datasets/

├── train/

│ ├── images/

│ ├── labels/

├── val/

│ ├── images/

│ ├── labels/

└── test/

├── images/

└── labels/

Make sure you have all your training, validation, and test sets organized this way.

**2. Install Necessary Libraries**

For training, you’ll need object detection libraries. The most popular is **YOLO**, but other options include **Faster R-CNN** and **SSD**. I'll show you an example using **YOLOv5** (one of the most efficient models).

To install the necessary libraries, use the following commands:

pip install torch torchvision

pip install yolov5 # For YOLOv5

**3. Train Your Model Using YOLOv5**

Here's how you can train a **YOLOv5** model on your dataset:

**Step 1: Set Up YOLOv5 Training Script**

First, ensure your dataset is in the **correct format**:

* The label files should follow **YOLO format**: class\_id x\_center y\_center width height (all normalized to [0,1]).

**Step 2: Download YOLOv5 Repository and Set Up**

Clone the **YOLOv5** repository if you don't have it yet:

git clone https://github.com/ultralytics/yolov5.git

cd yolov5

pip install -r requirements.txt

**Step 3: Modify Data Config**

Create a **data.yaml** configuration file to specify the paths and number of classes in your dataset.

Example data.yaml:

train: C:/Users/w10185657/DeepFish/datasets/train/images

val: C:/Users/w10185657/DeepFish/datasets/val/images

nc: 1 # Number of classes (e.g., fish species)

names: ['Fish'] # Class names (update this list with your fish species)

**Step 4: Train Your Model**

Now, train the model with the following command:

python train.py --img 640 --batch 16 --epochs 50 --data data.yaml --weights yolov5s.pt --cache

* **img**: Size of the input images (usually 640x640 or 416x416).
* **batch**: Batch size (you can adjust based on available memory).
* **epochs**: Number of epochs (you can start with 50, adjust as necessary).
* **weights**: Pre-trained weights for faster convergence (yolov5s.pt is a small, fast model).
* **data**: Path to your data config file.

**Step 5: Monitor Training Progress**

During training, YOLOv5 will display training metrics and save the trained weights in the runs/train/exp/weights directory. You can evaluate the model using:

python val.py --weights runs/train/exp/weights/best.pt --data data.yaml --img 640

**Step 6: Inference with Trained Model**

After training, you can test the trained model on new images using:

python detect.py --weights runs/train/exp/weights/best.pt --img 640 --source path\_to\_images\_or\_video

This will output images with bounding boxes drawn on detected fish.

**🧑‍💻 Further Exploration**

* **Data Augmentation**: To enhance model robustness, apply transformations like rotation, flipping, etc.
* **Hyperparameter Tuning**: Fine-tune the learning rate, batch size, and number of epochs.
* **Model Optimization**: Use techniques like pruning or quantization to optimize model performance.
* **Advanced Models**: Explore Faster R-CNN or SSD models if you want more accuracy but at the cost of speed.

Let me know how it goes or if you need further help with any of the steps! 🚀

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer screen

AI-generated content may be incorrect.python train.py --img 640 --batch 16 --epochs 50 --data data.yaml --weights yolov5s.pt --cacheA screenshot of a computer

AI-generated content may be incorrect.A computer screen shot of a program

AI-generated content may be incorrect.A computer screen shot of a black screen

AI-generated content may be incorrect.

Model summary: 214 layers, 7022326 parameters, 7022326 gradients, 15.9 GFLOPs

Transferred 343/349 items from yolov5s.pt

optimizer: SGD(lr=0.01) with parameter groups 57 weight(decay=0.0), 60 weight(decay=0.0005), 60 bias

train: Scanning C:\Users\w10185657\DeepFish\datasets\train\labels... 3596 images, 0 backgrounds, 0 cor

train: New cache created: C:\Users\w10185657\DeepFish\datasets\train\labels.cache

train: Caching images (2.3GB ram): 100%|██████████| 3596/3596 [00:24<00:00, 147.19it/s]

A computer screen with white text

AI-generated content may be incorrect.

python val.py --weights runs/train/exp/weights/best.pt --data data.yaml --img 640

python detect.py --weights runs/train/exp/weights/best.pt --img 640 --source path\_to\_images\_or\_video