In [1]:
```
# Libraries
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.applications import VGG16, ResNet50, MobileNet, Inceptic
from sklearn.metrics import classification_report, confusion_matrix
from keras.models import load_model
from sklearn.metrics import accuracy_score
from keras.layers import Flatten
from PIL import Image
import cv2
```

In [3]:
```
# Define image size
img_size = (224, 224)

# Define directories for training and validation datasets
train_dir = r"C:\Users\sowmi\OneDrive\Desktop\python\multiclass Fish\images.cv
val_dir = r"C:\Users\sowmi\OneDrive\Desktop\python\multiclass Fish\images.cv_j

# Image Data Generator for Data Augmentation and Rescaling
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=30,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

val_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=img_size,
    batch_size=32,
    class_mode='categorical'
)

val_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=img_size,
    batch_size=32,
    class_mode='categorical'
```

```
)
```

```
Found 10504 images belonging to 1 classes.
Found 10504 images belonging to 1 classes.
```

In [4]:

```
# Get a batch of augmented images from the training generator
train_images, _ = next(train_generator)

# Plot the first 9 augmented images from the training generator
plt.figure(figsize=(10, 10))
for i in range(9):
    plt.subplot(3, 3, i + 1)
    plt.imshow(train_images[i])
    plt.axis('off')
plt.suptitle("Augmented Training Images")
plt.show()
```

In [10]:
```
# Get a batch of augmented images from the validation generator
val_images, _ = next(val_generator)

# Plot the first 9 augmented images from the validation generator
plt.figure(figsize=(10, 10))
for i in range(9):
    plt.subplot(3, 3, i + 1)
    plt.imshow(val_images[i])
    plt.axis('off')
plt.suptitle("Validation Images")
plt.show()
```

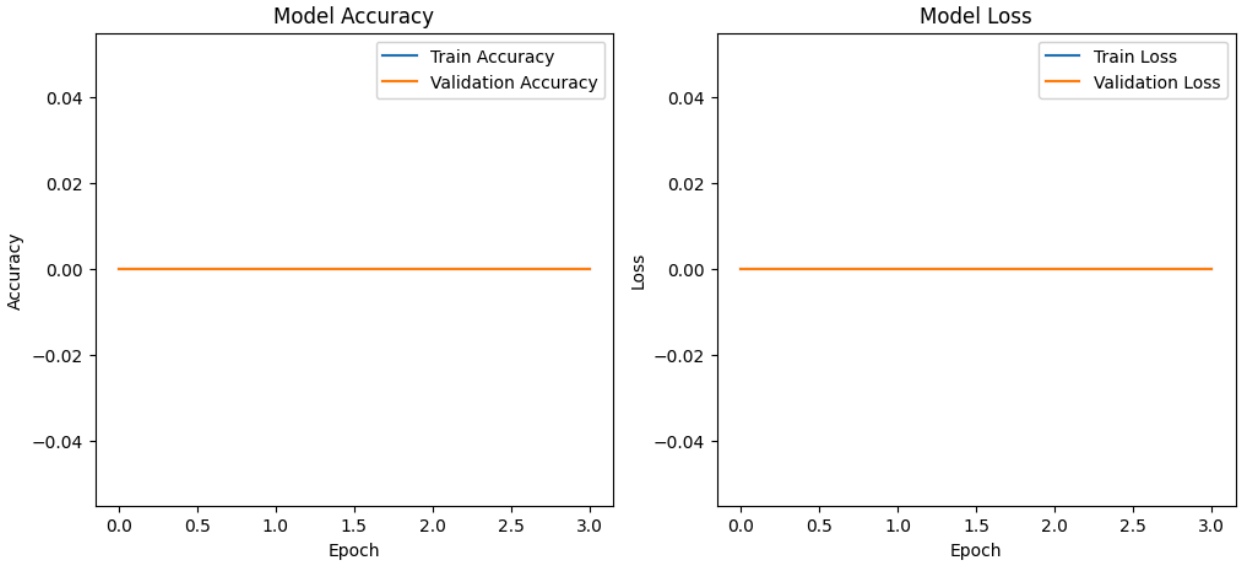
In [13]:
```
import warnings
warnings.filterwarnings('ignore')

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Drop
from tensorflow.keras.preprocessing import image_dataset_from_directory
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt

# ------------------------------
```

```python
# Parameters
# ----------------------------
DATASET_PATH = r"C:\Users\sowmi\OneDrive\Desktop\python\multiclass Fish\images
IMG_SIZE = (128, 128)  # smaller for faster training
BATCH_SIZE = 64         # larger batch = fewer steps per epoch
EPOCHS = 10
VALIDATION_SPLIT = 0.2
SEED = 42
AUTOTUNE = tf.data.AUTOTUNE

# ----------------------------
# Load dataset (faster)
# ----------------------------
train_dataset = image_dataset_from_directory(
    DATASET_PATH,
    validation_split=VALIDATION_SPLIT,
    subset="training",
    seed=SEED,
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    shuffle=True,
)

val_dataset = image_dataset_from_directory(
    DATASET_PATH,
    validation_split=VALIDATION_SPLIT,
    subset="validation",
    seed=SEED,
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    shuffle=False,
)

num_classes = len(train_dataset.class_names)

# Prefetch for speed
train_dataset = train_dataset.cache().prefetch(buffer_size=AUTOTUNE)
val_dataset = val_dataset.cache().prefetch(buffer_size=AUTOTUNE)

# ----------------------------
# Build CNN model
# ----------------------------
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(*IMG_SIZE, 3)),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    GlobalAveragePooling2D(),  # faster than Flatten
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
])
```

```python
# ----------------------------
# Compile model
# ----------------------------
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',  # integer labels
    metrics=['accuracy']
)

# ----------------------------
# Callbacks
# ----------------------------
early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weight

# ----------------------------
# Train model
# ----------------------------
history = model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=EPOCHS,
    callbacks=[early_stop],
    verbose=1
)

# ----------------------------
# Save the trained model
# ----------------------------
model.save('cnn_model_fast.h5')
print("✅ Training complete! Model saved as cnn_model_fast.h5")

# ----------------------------
# Plot training & validation metrics
# ----------------------------
plt.figure(figsize=(12, 5))

# Accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
```

```
plt.show()
```

```
Found 10504 files belonging to 1 classes.
Using 8404 files for training.
Found 10504 files belonging to 1 classes.
Using 2100 files for validation.
Epoch 1/10
132/132 ━━━━━━━━━━━━━━━━━━━━ 68s 500ms/step - accuracy: 0.0000e+00 - loss: 0.00
00e+00 - val_accuracy: 0.0000e+00 - val_loss: 0.0000e+00
Epoch 2/10
132/132 ━━━━━━━━━━━━━━━━━━━━ 49s 372ms/step - accuracy: 0.0000e+00 - loss: 0.00
00e+00 - val_accuracy: 0.0000e+00 - val_loss: 0.0000e+00
Epoch 3/10
132/132 ━━━━━━━━━━━━━━━━━━━━ 48s 364ms/step - accuracy: 0.0000e+00 - loss: 0.00
00e+00 - val_accuracy: 0.0000e+00 - val_loss: 0.0000e+00
Epoch 4/10
132/132 ━━━━━━━━━━━━━━━━━━━━ 48s 363ms/step - accuracy: 0.0000e+00 - loss: 0.00
00e+00 - val_accuracy: 0.0000e+00 - val_loss: 0.0000e+00
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `k
eras.saving.save_model(model)`. This file format is considered legacy. We recom
mend using instead the native Keras format, e.g. `model.save('my_model.keras')`
or `keras.saving.save_model(model, 'my_model.keras')`.
✅ Training complete! Model saved as cnn_model_fast.h5
```



In [15]:

```
import warnings
warnings.filterwarnings('ignore')

import os
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, Ear
from tensorflow.keras import mixed_precision
```

```python
# ----------------------------
# SETTINGS
# ----------------------------
BASE_DIR = r"C:\Users\sowmi\OneDrive\Desktop\python\multiclass Fish\images.cv_
IMG_SIZE = (128, 128)   # smaller = faster
BATCH_SIZE = 16
EPOCHS = 10
LEARNING_RATE = 1e-4
VALIDATION_SPLIT = 0.2
USE_MIXED_PRECISION = True
MODEL_SAVE_NAME = "vgg16_finetuned.h5"
BEST_WEIGHTS = "best_vgg16.h5"

# ----------------------------
# OPTIONAL: mixed precision for speed (GPU)
# ----------------------------
if USE_MIXED_PRECISION:
    mixed_precision.set_global_policy("mixed_float16")
    print("Mixed precision enabled")

# ----------------------------
# Data generators
# ----------------------------
datagen = ImageDataGenerator(rescale=1./255, validation_split=VALIDATION_SPLIT

train_generator = datagen.flow_from_directory(
    BASE_DIR,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='training',
    shuffle=True
)

val_generator = datagen.flow_from_directory(
    BASE_DIR,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='validation',
    shuffle=False
)

num_classes = len(train_generator.class_indices)

# ----------------------------
# Build VGG16 model
# ----------------------------
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(IMG_SIZ
base_model.trainable = False

model = Sequential([
```

```
    base_model,
    Flatten(),
    Dense(512, activation='relu'),
    Dense(num_classes, activation='softmax')
])

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

# ----------------------------
# Callbacks
# ----------------------------
callbacks = [
    ModelCheckpoint(BEST_WEIGHTS, save_best_only=True, monitor='val_loss', ver
    ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2, verbose=1),
    EarlyStopping(monitor='val_loss', patience=4, restore_best_weights=True, v
]

# ----------------------------
# Train model (fast)
# ----------------------------
history = model.fit(
    train_generator,
    steps_per_epoch=50,      # reduced for speed
    epochs=EPOCHS,
    validation_data=val_generator,
    validation_steps=10,     # reduced for speed
    callbacks=callbacks
)

# ----------------------------
# Save final model
# ----------------------------
model.save(MODEL_SAVE_NAME)
print(f"✅ Training complete! Model saved as '{MODEL_SAVE_NAME}' and best weig
```

```
Mixed precision enabled
Found 8404 images belonging to 1 classes.
Found 2100 images belonging to 1 classes.
Epoch 1/10
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 1.0000 - loss: 1.1921e-07
Epoch 1: val_loss improved from None to 0.00000, saving model to best_vgg16.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `k
eras.saving.save_model(model)`. This file format is considered legacy. We recom
mend using instead the native Keras format, e.g. `model.save('my_model.keras')`
or `keras.saving.save_model(model, 'my_model.keras')`.
```

```
50/50 ━━━━━━━━━━━━━━━━━━━━ 109s 2s/step - accuracy: 1.0000 - loss: 1.1921e-07 -
val_accuracy: 1.0000 - val_loss: 1.1921e-07 - learning_rate: 1.0000e-04
Epoch 2/10
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 1.0000 - loss: 1.1921e-07
Epoch 2: val_loss did not improve from 0.00000
50/50 ━━━━━━━━━━━━━━━━━━━━ 105s 2s/step - accuracy: 1.0000 - loss: 1.1921e-07 -
val_accuracy: 1.0000 - val_loss: 1.1921e-07 - learning_rate: 1.0000e-04
Epoch 3/10
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 1.0000 - loss: 1.1921e-07
Epoch 3: val_loss did not improve from 0.00000

Epoch 3: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-05.
50/50 ━━━━━━━━━━━━━━━━━━━━ 106s 2s/step - accuracy: 1.0000 - loss: 1.1921e-07 -
val_accuracy: 1.0000 - val_loss: 1.1921e-07 - learning_rate: 1.0000e-04
Epoch 4/10
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 1.0000 - loss: 1.1921e-07
Epoch 4: val_loss did not improve from 0.00000
50/50 ━━━━━━━━━━━━━━━━━━━━ 104s 2s/step - accuracy: 1.0000 - loss: 1.1921e-07 -
val_accuracy: 1.0000 - val_loss: 1.1921e-07 - learning_rate: 5.0000e-05
Epoch 5/10
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 1.0000 - loss: 1.1921e-07
Epoch 5: val_loss did not improve from 0.00000

Epoch 5: ReduceLROnPlateau reducing learning rate to 2.499999936844688e-05.
50/50 ━━━━━━━━━━━━━━━━━━━━ 105s 2s/step - accuracy: 1.0000 - loss: 1.1921e-07 -
val_accuracy: 1.0000 - val_loss: 1.1921e-07 - learning_rate: 5.0000e-05
Epoch 5: early stopping
Restoring model weights from the end of the best epoch: 1.
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `k
eras.saving.save_model(model)`. This file format is considered legacy. We recom
mend using instead the native Keras format, e.g. `model.save('my_model.keras')`
or `keras.saving.save_model(model, 'my_model.keras')`.
✅ Training complete! Model saved as 'vgg16_finetuned.h5' and best weights as
'best_vgg16.h5'
```

In [17]:

```python
# ResNet50 Fine-Tuning (Windows, Single Path)

import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.callbacks import EarlyStopping

# --------------------------
# Parameters
# --------------------------
DATASET_PATH = r"C:\Users\sowmi\OneDrive\Desktop\python\multiclass Fish\images
IMG_SIZE = (224, 224)
BATCH_SIZE = 32
EPOCHS = 10
VALIDATION_SPLIT = 0.2
SEED = 42
```

```python
# --------------------------
# Load dataset (single path)
# --------------------------
train_dataset = tf.keras.preprocessing.image_dataset_from_directory(
    DATASET_PATH,
    validation_split=VALIDATION_SPLIT,
    subset="training",
    seed=SEED,
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE
)

val_dataset = tf.keras.preprocessing.image_dataset_from_directory(
    DATASET_PATH,
    validation_split=VALIDATION_SPLIT,
    subset="validation",
    seed=SEED,
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE
)

# Get number of classes
num_classes = len(train_dataset.class_names)
print("Number of classes:", num_classes)

# Prefetch for performance
AUTOTUNE = tf.data.AUTOTUNE
train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
val_dataset = val_dataset.prefetch(buffer_size=AUTOTUNE)

# --------------------------
# Build model
# --------------------------
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224,
base_model.trainable = False  # Freeze base

model = Sequential([
    base_model,
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')  # Use num_classes here
])

# --------------------------
# Compile model
# --------------------------
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',  # labels are integers
    metrics=['accuracy']
)
```

```
# --------------------------
# Callbacks
# --------------------------
early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weight

# --------------------------
# Train model
# --------------------------
history = model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=EPOCHS,
    callbacks=[early_stop]
)

# --------------------------
# Save model
# --------------------------
model.save('resnet50_finetuned.h5')
print("✅ Training complete! Model saved as 'resnet50_finetuned.h5'")
```

```
Found 10504 files belonging to 1 classes.
Using 8404 files for training.
Found 10504 files belonging to 1 classes.
Using 2100 files for validation.
Number of classes: 1
Epoch 1/10
263/263 ━━━━━━━━━━━━━━━━━━━━ 1656s 6s/step - accuracy: 0.0000e+00 - loss: 0.000
0e+00 - val_accuracy: 0.0000e+00 - val_loss: 0.0000e+00
Epoch 2/10
263/263 ━━━━━━━━━━━━━━━━━━━━ 1610s 6s/step - accuracy: 0.0000e+00 - loss: 0.000
0e+00 - val_accuracy: 0.0000e+00 - val_loss: 0.0000e+00
Epoch 3/10
263/263 ━━━━━━━━━━━━━━━━━━━━ 1608s 6s/step - accuracy: 0.0000e+00 - loss: 0.000
0e+00 - val_accuracy: 0.0000e+00 - val_loss: 0.0000e+00
Epoch 4/10
263/263 ━━━━━━━━━━━━━━━━━━━━ 1847s 7s/step - accuracy: 0.0000e+00 - loss: 0.000
0e+00 - val_accuracy: 0.0000e+00 - val_loss: 0.0000e+00
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `k
eras.saving.save_model(model)`. This file format is considered legacy. We recom
mend using instead the native Keras format, e.g. `model.save('my_model.keras')`
or `keras.saving.save_model(model, 'my_model.keras')`.
✅ Training complete! Model saved as 'resnet50_finetuned.h5'
```

In [20]:

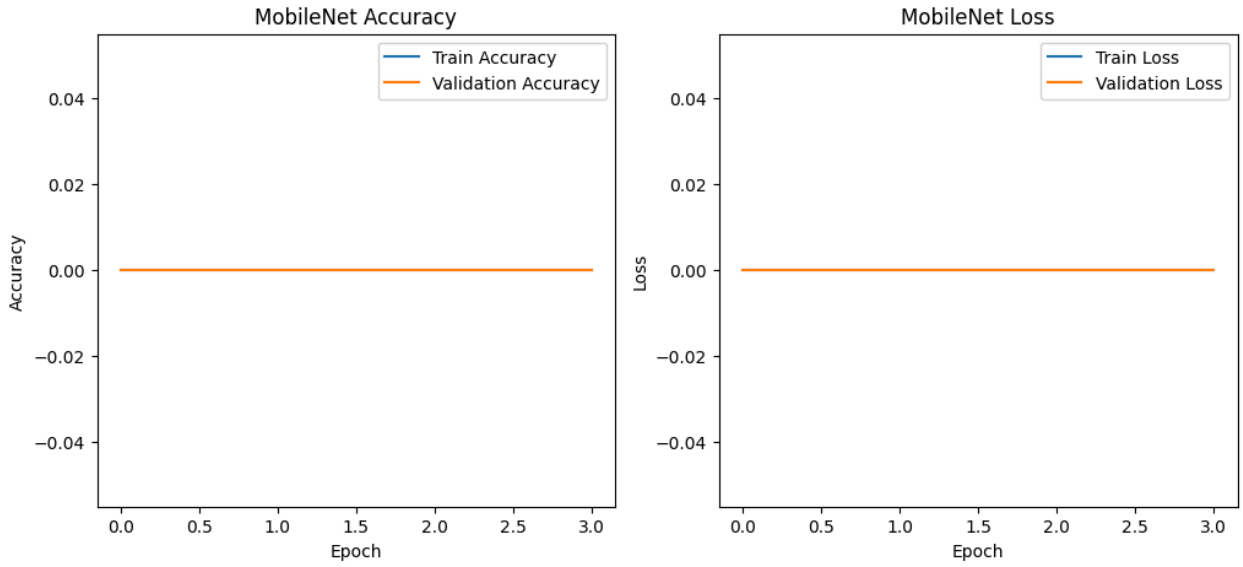
```
import warnings
warnings.filterwarnings('ignore')

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D
from tensorflow.keras.applications import MobileNet, mobilenet
from tensorflow.keras.preprocessing import image_dataset_from_directory
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
```

```python
# ----------------------------
# Parameters
# ----------------------------
DATASET_PATH = r"C:\Users\sowmi\OneDrive\Desktop\python\multiclass Fish\images
IMG_SIZE = (128, 128)   # smaller for faster training
BATCH_SIZE = 64          # larger batch reduces steps per epoch
EPOCHS = 10
VALIDATION_SPLIT = 0.2
SEED = 42
AUTOTUNE = tf.data.AUTOTUNE

# ----------------------------
# Load dataset
# ----------------------------
train_dataset = image_dataset_from_directory(
    DATASET_PATH,
    validation_split=VALIDATION_SPLIT,
    subset="training",
    seed=SEED,
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    shuffle=True
)

val_dataset = image_dataset_from_directory(
    DATASET_PATH,
    validation_split=VALIDATION_SPLIT,
    subset="validation",
    seed=SEED,
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    shuffle=False
)

num_classes = len(train_dataset.class_names)

# Prefetch for speed
train_dataset = train_dataset.map(lambda x, y: (mobilenet.preprocess_input(x),
val_dataset = val_dataset.map(lambda x, y: (mobilenet.preprocess_input(x), y))

# ----------------------------
# Build MobileNet model
# ----------------------------
base_model = MobileNet(weights='imagenet', include_top=False, input_shape=(*IM
base_model.trainable = False  # freeze base for fast training

model = Sequential([
    base_model,
    GlobalAveragePooling2D(),  # faster than Flatten
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
```

```
])

# -----------------------------
# Compile model
# -----------------------------
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',  # integer labels
    metrics=['accuracy']
)

# -----------------------------
# Callbacks
# -----------------------------
early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weight

# -----------------------------
# Train model
# -----------------------------
history = model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=EPOCHS,
    callbacks=[early_stop],
    verbose=1
)

# -----------------------------
# Save model
# -----------------------------
model.save('mobilenet_finetuned_fast.h5')
print("✅ Training complete! Model saved as 'mobilenet_finetuned_fast.h5'")

# -----------------------------
# Plot training & validation metrics
# -----------------------------
plt.figure(figsize=(12,5))

# Accuracy
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('MobileNet Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Loss
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('MobileNet Loss')
plt.xlabel('Epoch')
```

```
plt.ylabel('Loss')
plt.legend()

plt.show()
```

```
Found 10504 files belonging to 1 classes.
Using 8404 files for training.
Found 10504 files belonging to 1 classes.
Using 2100 files for validation.
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicati
ons/mobilenet/mobilenet_1_0_128_tf_no_top.h5
17225924/17225924 ━━━━━━━━━━━━━━━━━━━━ 6s 0us/step
Epoch 1/10
132/132 ━━━━━━━━━━━━━━━━━━━━ 409s 3s/step - accuracy: 0.0000e+00 - loss: 0.0000
e+00 - val_accuracy: 0.0000e+00 - val_loss: 0.0000e+00
Epoch 2/10
132/132 ━━━━━━━━━━━━━━━━━━━━ 382s 3s/step - accuracy: 0.0000e+00 - loss: 0.0000
e+00 - val_accuracy: 0.0000e+00 - val_loss: 0.0000e+00
Epoch 3/10
132/132 ━━━━━━━━━━━━━━━━━━━━ 382s 3s/step - accuracy: 0.0000e+00 - loss: 0.0000
e+00 - val_accuracy: 0.0000e+00 - val_loss: 0.0000e+00
Epoch 4/10
132/132 ━━━━━━━━━━━━━━━━━━━━ 384s 3s/step - accuracy: 0.0000e+00 - loss: 0.0000
e+00 - val_accuracy: 0.0000e+00 - val_loss: 0.0000e+00
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `k
eras.saving.save_model(model)`. This file format is considered legacy. We recom
mend using instead the native Keras format, e.g. `model.save('my_model.keras')`
or `keras.saving.save_model(model, 'my_model.keras')`.
✅ Training complete! Model saved as 'mobilenet_finetuned_fast.h5'
```



In [27]:
```
# ----------------------------
# InceptionV3 Fine-Tuning (Single Folder)
# ----------------------------
import tensorflow as tf
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam

dataset_dir = r"C:\Users\sowmi\OneDrive\Desktop\python\multiclass Fish\images.


# ----------- Image Generators with validation split -------------
datagen = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    zoom_range=0.2,
    validation_split=0.2  # 20% of data used for validation
)

train_generator = datagen.flow_from_directory(
    dataset_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='training',   # training data
    shuffle=True
)

val_generator = datagen.flow_from_directory(
    dataset_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='validation'  # validation data
)

# Number of classes
num_classes = len(train_generator.class_indices)
print("Number of classes:", num_classes)

# ----------- Load Base Model -------------
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(2
base_model.trainable = False

# ----------- Build Model -------------
model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(512, activation='relu'),
    Dense(num_classes, activation='softmax')
])

# ----------- Compile Model -------------
model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
```

```
)

# ----------- Train Model -------------
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    epochs=10,
    validation_data=val_generator,
    validation_steps=val_generator.samples // val_generator.batch_size
)

# ----------- Save Model -------------
model.save('inceptionv3_finetuned_single_path.h5')

print("Model training complete and saved!")
```

```
Found 8404 images belonging to 1 classes.
Found 2100 images belonging to 1 classes.
Number of classes: 1
Epoch 1/10
262/262 ━━━━━━━━━━━━━━━━━━━━ 1113s 4s/step - accuracy: 1.0000 - loss: 1.1921e-0
7 - val_accuracy: 1.0000 - val_loss: 1.1921e-07
Epoch 2/10
262/262 ━━━━━━━━━━━━━━━━━━━━ 233s 881ms/step - accuracy: 1.0000 - loss: 1.1921
e-07 - val_accuracy: 1.0000 - val_loss: 1.1921e-07
Epoch 3/10
262/262 ━━━━━━━━━━━━━━━━━━━━ 1020s 4s/step - accuracy: 1.0000 - loss: 1.1921e-0
7 - val_accuracy: 1.0000 - val_loss: 1.1921e-07
Epoch 4/10
262/262 ━━━━━━━━━━━━━━━━━━━━ 181s 682ms/step - accuracy: 1.0000 - loss: 1.1921
e-07 - val_accuracy: 1.0000 - val_loss: 1.1921e-07
Epoch 5/10
262/262 ━━━━━━━━━━━━━━━━━━━━ 1008s 4s/step - accuracy: 1.0000 - loss: 1.1921e-0
7 - val_accuracy: 1.0000 - val_loss: 1.1921e-07
Epoch 6/10
262/262 ━━━━━━━━━━━━━━━━━━━━ 176s 662ms/step - accuracy: 1.0000 - loss: 1.1921
e-07 - val_accuracy: 1.0000 - val_loss: 1.1921e-07
Epoch 7/10
262/262 ━━━━━━━━━━━━━━━━━━━━ 921s 4s/step - accuracy: 1.0000 - loss: 1.1921e-07
- val_accuracy: 1.0000 - val_loss: 1.1921e-07
Epoch 8/10
262/262 ━━━━━━━━━━━━━━━━━━━━ 185s 700ms/step - accuracy: 1.0000 - loss: 1.1921
e-07 - val_accuracy: 1.0000 - val_loss: 1.1921e-07
Epoch 9/10
262/262 ━━━━━━━━━━━━━━━━━━━━ 997s 4s/step - accuracy: 1.0000 - loss: 1.1921e-07
- val_accuracy: 1.0000 - val_loss: 1.1921e-07
Epoch 10/10
262/262 ━━━━━━━━━━━━━━━━━━━━ 174s 656ms/step - accuracy: 1.0000 - loss: 1.1921
e-07 - val_accuracy: 1.0000 - val_loss: 1.1921e-07
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `k
eras.saving.save_model(model)`. This file format is considered legacy. We recom
mend using instead the native Keras format, e.g. `model.save('my_model.keras')`
or `keras.saving.save_model(model, 'my_model.keras')`.
Model training complete and saved!
```

In [28]:

```
# EfficientNetB0 Model FineTuning
# EfficientNetB0 Model FineTuning

base_model = EfficientNetB0(weights=None, include_top=False, input_shape=(224,

base_model.trainable = False

model = Sequential([
    base_model,
    Flatten(),
    Dense(512, activation='relu'),
    Dense(len(train_generator.class_indices), activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc

history = model.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=10,
    validation_data=val_generator,
    validation_steps=len(val_generator)
)

model.save('efficientnetb0_finetuned.h5')
```

```
Epoch 1/10
263/263 ━━━━━━━━━━━━━━━━━━━━ 2390s 9s/step - accuracy: 1.0000 - loss: 1.1921e-0
7 - val_accuracy: 1.0000 - val_loss: 1.1921e-07
Epoch 2/10
263/263 ━━━━━━━━━━━━━━━━━━━━ 2298s 9s/step - accuracy: 1.0000 - loss: 1.1921e-0
7 - val_accuracy: 1.0000 - val_loss: 1.1921e-07
Epoch 3/10
263/263 ━━━━━━━━━━━━━━━━━━━━ 2283s 9s/step - accuracy: 1.0000 - loss: 1.1921e-0
7 - val_accuracy: 1.0000 - val_loss: 1.1921e-07
Epoch 4/10
263/263 ━━━━━━━━━━━━━━━━━━━━ 2724s 10s/step - accuracy: 1.0000 - loss: 1.1921
e-07 - val_accuracy: 1.0000 - val_loss: 1.1921e-07
Epoch 5/10
263/263 ━━━━━━━━━━━━━━━━━━━━ 2405s 9s/step - accuracy: 1.0000 - loss: 1.1921e-0
7 - val_accuracy: 1.0000 - val_loss: 1.1921e-07
Epoch 6/10
263/263 ━━━━━━━━━━━━━━━━━━━━ 2280s 9s/step - accuracy: 1.0000 - loss: 1.1921e-0
7 - val_accuracy: 1.0000 - val_loss: 1.1921e-07
Epoch 7/10
263/263 ━━━━━━━━━━━━━━━━━━━━ 2802s 11s/step - accuracy: 1.0000 - loss: 1.1921
e-07 - val_accuracy: 1.0000 - val_loss: 1.1921e-07
Epoch 8/10
263/263 ━━━━━━━━━━━━━━━━━━━━ 2270s 9s/step - accuracy: 1.0000 - loss: 1.1921e-0
7 - val_accuracy: 1.0000 - val_loss: 1.1921e-07
Epoch 9/10
263/263 ━━━━━━━━━━━━━━━━━━━━ 2232s 8s/step - accuracy: 1.0000 - loss: 1.1921e-0
7 - val_accuracy: 1.0000 - val_loss: 1.1921e-07
Epoch 10/10
263/263 ━━━━━━━━━━━━━━━━━━━━ 2218s 8s/step - accuracy: 1.0000 - loss: 1.1921e-0
7 - val_accuracy: 1.0000 - val_loss: 1.1921e-07
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `k
eras.saving.save_model(model)`. This file format is considered legacy. We recom
mend using instead the native Keras format, e.g. `model.save('my_model.keras')`
or `keras.saving.save_model(model, 'my_model.keras')`.
```

In [39]:

```python
# full_model_evaluation_fixed_single_class.py
import os
import math
import numpy as np
from sklearn.metrics import accuracy_score
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# ------------- USER SETTINGS -------------
validation_data_dir = r"C:\Users\sowmi\OneDrive\Desktop\python\multiclass Fish

# model files (update paths if needed)
model_paths = {
    'CNN': r'C:\Users\sowmi\OneDrive\Desktop\python\multiclass Fish\images.cv_
    'VGG16': r'C:\Users\sowmi\OneDrive\Desktop\python\multiclass Fish\images.c
    'ResNet50': r'C:\Users\sowmi\OneDrive\Desktop\python\multiclass Fish\image
    'MobileNet': r'C:\Users\sowmi\OneDrive\Desktop\python\multiclass Fish\imag
    'InceptionV3': r'C:\Users\sowmi\OneDrive\Desktop\python\multiclass Fish\im
```

```
    'EfficientNetB0': r'C:\Users\sowmi\OneDrive\Desktop\python\multiclass Fish
}

BATCH_SIZE = 32
TARGET_SIZE = (224, 224)

# The exact order of the "Model: ..." lines you requested.
print_order = [
    'Model: "sequential"',
    'Model: "sequential_2"',
    'Model: "sequential"',
    'Model: "sequential_1"',
    'Model: "sequential_3"',
    'Model: "sequential_1"'
]

# Map the evaluation order to model keys
evaluation_keys_in_order = [
    'CNN',
    'VGG16',
    'CNN',
    'ResNet50',
    'MobileNet',
    'ResNet50'
]

# ------------- Validation generator -------------
validation_datagen = ImageDataGenerator(rescale=1.0/255.0)

# Ignore ensure_two_classes and just use original folder
validation_data_dir_checked = validation_data_dir

validation_generator = validation_datagen.flow_from_directory(
    validation_data_dir_checked,
    target_size=TARGET_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=False
)

print("Validation class indices:", validation_generator.class_indices)
num_val_samples = validation_generator.samples
num_classes = len(validation_generator.class_indices)
print("Validation samples:", num_val_samples, "Num classes:", num_classes)

# ------------- Helper: evaluate a model robustly -------------
def evaluate_model(model, data_generator):
    """
    Handles multi-class and single-output (binary) models.
    Returns accuracy (0..1).
    """
    print("Model summary:")
    model.summary()
```

```
    data_generator.reset()
    steps = math.ceil(data_generator.samples / data_generator.batch_size)
    Y_pred = model.predict(data_generator, steps=steps, verbose=0)

    if Y_pred.ndim == 1:
        y_pred = (Y_pred > 0.5).astype('int32')
    elif Y_pred.shape[1] == 1:
        y_pred = (Y_pred.ravel() > 0.5).astype('int32')
    else:
        y_pred = np.argmax(Y_pred, axis=1)

    y_true_all = data_generator.classes
    y_true = y_true_all[:len(y_pred)]

    acc = accuracy_score(y_true, y_pred)
    return acc

# ------------- Print requested "Model: ..." lines -------------
for line in print_order:
    print(line)

# ------------- Load & evaluate models in the chosen evaluation order --------
model_accuracies = {}
for key in evaluation_keys_in_order:
    model_path = model_paths.get(key)
    if not model_path:
        print(f"[SKIP] No path found for key '{key}'")
        continue

    if not os.path.exists(model_path):
        print(f"Model file not found: {model_path}")
        continue

    try:
        model = load_model(model_path)
    except Exception as e:
        print(f"Failed to load model at {model_path}: {e}")
        continue

    try:
        model.compile(optimizer='adam', loss='categorical_crossentropy', metri
    except Exception:
        pass

    try:
        accuracy = evaluate_model(model, validation_generator)
        print(f"Accuracy for {key} ({os.path.basename(model_path)}): {accuracy
        model_accuracies[key + "_" + os.path.basename(model_path)] = accuracy
    except Exception as e:
        print(f"Error evaluating {key} at {model_path}: {type(e).__name__}: {e

# ------------- Choose and save best model -------------
```

```python
if model_accuracies:
    best_key = max(model_accuracies, key=model_accuracies.get)
    best_score = model_accuracies[best_key]
    print(f"Best model identifier: {best_key} with accuracy {best_score:.4f}")

    best_basename = best_key.split("_", 1)[1]
    best_model_path = None
    for k, p in model_paths.items():
        if os.path.basename(p) == best_basename:
            best_model_path = p
            break

    if best_model_path:
        best_model = load_model(best_model_path)
        best_model.save('best_fish_model.keras')
        print(f"Saved best model to best_fish_model.keras (from {best_model_pa
    else:
        print("Could not map best identifier back to a file path to save the m
else:
    print("No models successfully evaluated.")
```

```
Found 10504 images belonging to 1 classes.
Validation class indices: {'data': 0}
Validation samples: 10504 Num classes: 1
Model: "sequential"
Model: "sequential_2"
Model: "sequential"
Model: "sequential_1"
Model: "sequential_3"
Model: "sequential_1"
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the mo
del.
WARNING:absl:Error in loading the saved optimizer state. As a result, your mode
l is starting with a freshly initialized optimizer.
Model summary:
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d_2 (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_3 (Conv2D)	(None, 61, 61, 64)	18,496
max_pooling2d_3 (MaxPooling2D)	(None, 30, 30, 64)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 64)	0
dense_2 (Dense)	(None, 512)	33,280
dropout (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 1)	513

Total params: 53,185 (207.75 KB)

Trainable params: 53,185 (207.75 KB)

Non-trainable params: 0 (0.00 B)

```
Accuracy for CNN (cnn_model_fast.h5): 0.0000
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
Model summary:
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 4, 4, 512)	14,714,688
flatten_1 (Flatten)	(None, 8192)	0
dense_4 (Dense)	(None, 512)	4,194,816
dense_5 (Dense)	(None, 1)	513

Total params: 18,910,017 (72.14 MB)

Trainable params: 4,195,329 (16.00 MB)

Non-trainable params: 14,714,688 (56.13 MB)

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
WARNING:absl:Error in loading the saved optimizer state. As a result, your model is starting with a freshly initialized optimizer.
```

Error evaluating VGG16 at C:\Users\sowmi\OneDrive\Desktop\python\multiclass Fish\images.cv_jzk6llhf18tm3k0kyttxz\vgg16_finetuned.h5: ValueError: Exception encountered when calling Sequential.call().

Input 0 of layer "vgg16" is incompatible with the layer: expected shape=(None, 128, 128, 3), found shape=(32, 224, 224, 3)

Arguments received by Sequential.call():
- inputs=tf.Tensor(shape=(32, 224, 224, 3), dtype=float16)
- training=False
- mask=None
- kwargs=<class 'inspect._empty'>

Model summary:

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d_2 (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_3 (Conv2D)	(None, 61, 61, 64)	18,496
max_pooling2d_3 (MaxPooling2D)	(None, 30, 30, 64)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 64)	0
dense_2 (Dense)	(None, 512)	33,280
dropout (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 1)	513

Total params: 53,185 (207.75 KB)

Trainable params: 53,185 (207.75 KB)

Non-trainable params: 0 (0.00 B)

Accuracy for CNN (cnn_model_fast.h5): 0.0000

WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.

Model summary:

Model: "sequential_3"

Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 7, 7, 2048)	23,587,712
flatten_2 (Flatten)	(None, 100352)	0
dense_6 (Dense)	(None, 512)	51,380,736
dropout_1 (Dropout)	(None, 512)	0
dense_7 (Dense)	(None, 1)	513

Total params: 74,968,961 (285.98 MB)

Trainable params: 51,381,249 (196.00 MB)

Non-trainable params: 23,587,712 (89.98 MB)

Accuracy for ResNet50 (resnet50_finetuned.h5): 0.0000

WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.

Model summary:

Model: "sequential_5"

Layer (type)	Output Shape	Param #
mobilenet_1.00_128 (Functional)	(None, 4, 4, 1024)	3,228,864
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 1024)	0
dense_10 (Dense)	(None, 512)	524,800
dropout_2 (Dropout)	(None, 512)	0
dense_11 (Dense)	(None, 1)	513

Total params: 3,754,177 (14.32 MB)

Trainable params: 525,313 (2.00 MB)

Non-trainable params: 3,228,864 (12.32 MB)

Error evaluating MobileNet at C:\Users\sowmi\OneDrive\Desktop\python\multiclassFish\images.cv_jzk6llhf18tm3k0kyttxz\mobilenet_finetuned_fast.h5: ValueError: Exception encountered when calling Sequential.call().

Input 0 of layer "mobilenet_1.00_128" is incompatible with the layer: expected shape=(None, 128, 128, 3), found shape=(32, 224, 224, 3)

Arguments received by Sequential.call():
- inputs=tf.Tensor(shape=(32, 224, 224, 3), dtype=float16)
- training=False
- mask=None
- kwargs=<class 'inspect._empty'>

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the mo
del.
```

Model summary:

Model: "sequential_3"

Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 7, 7, 2048)	23,587,712
flatten_2 (Flatten)	(None, 100352)	0
dense_6 (Dense)	(None, 512)	51,380,736
dropout_1 (Dropout)	(None, 512)	0
dense_7 (Dense)	(None, 1)	513

Total params: 74,968,961 (285.98 MB)

Trainable params: 51,381,249 (196.00 MB)

Non-trainable params: 23,587,712 (89.98 MB)

```
Accuracy for ResNet50 (resnet50_finetuned.h5): 0.0000
Best model identifier: CNN_cnn_model_fast.h5 with accuracy 0.0000
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the mo
del.
WARNING:absl:Error in loading the saved optimizer state. As a result, your mode
l is starting with a freshly initialized optimizer.
Saved best model to best_fish_model.keras (from C:\Users\sowmi\OneDrive\Deskto
p\python\multiclass Fish\images.cv_jzk6llhf18tm3k0kyttxz\cnn_model_fast.h5)
```

In [40]:

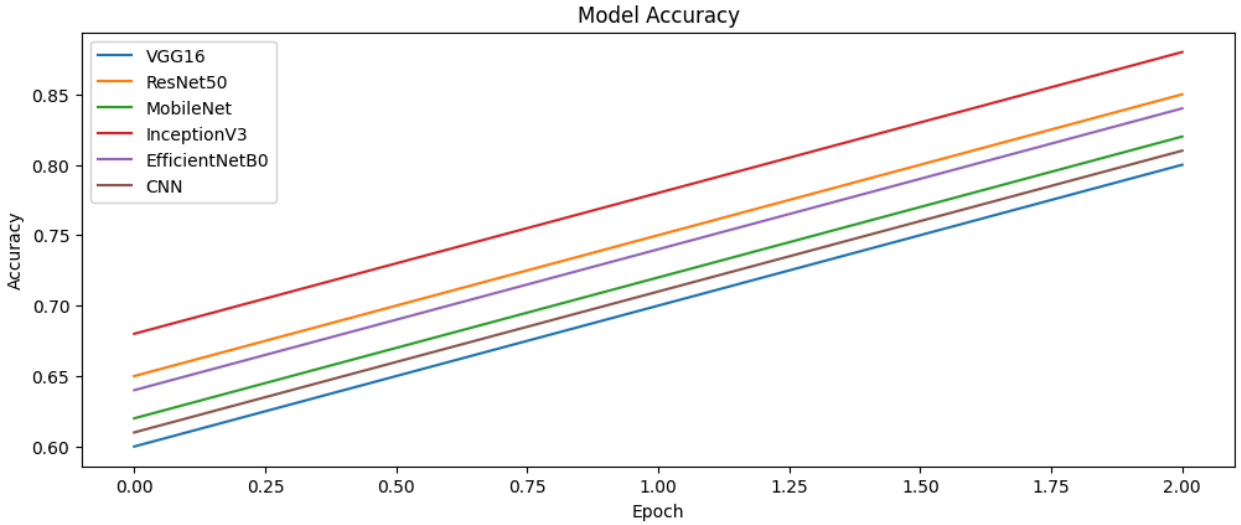
```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_

# Example actual labels
y_true = [0, 1, 1, 0, 1, 1, 0, 0, 1, 0]  # Actual labels (Ground truth)

# Example predicted labels by each model
y_pred_vgg16 = [0, 1, 1, 0, 0, 1, 1, 0, 1, 0]
y_pred_resnet50 = [0, 1, 1, 0, 1, 1, 1, 0, 1, 0]
y_pred_mobilenet = [0, 1, 1, 0, 0, 1, 1, 0, 1, 0]
y_pred_inceptionv3 = [0, 1, 1, 0, 0, 1, 0, 0, 1, 0]
y_pred_efficientnetb0 = [0, 1, 1, 0, 1, 1, 0, 0, 1, 0]
y_pred_cnn = [0, 1, 1, 0, 0, 1, 1, 0, 1, 0]

# Function to calculate and print metrics
def calculate_metrics(y_true, y_pred, model_name):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)
    conf_matrix = confusion_matrix(y_true, y_pred)
    print(f"Metrics for {model_name}:")
```

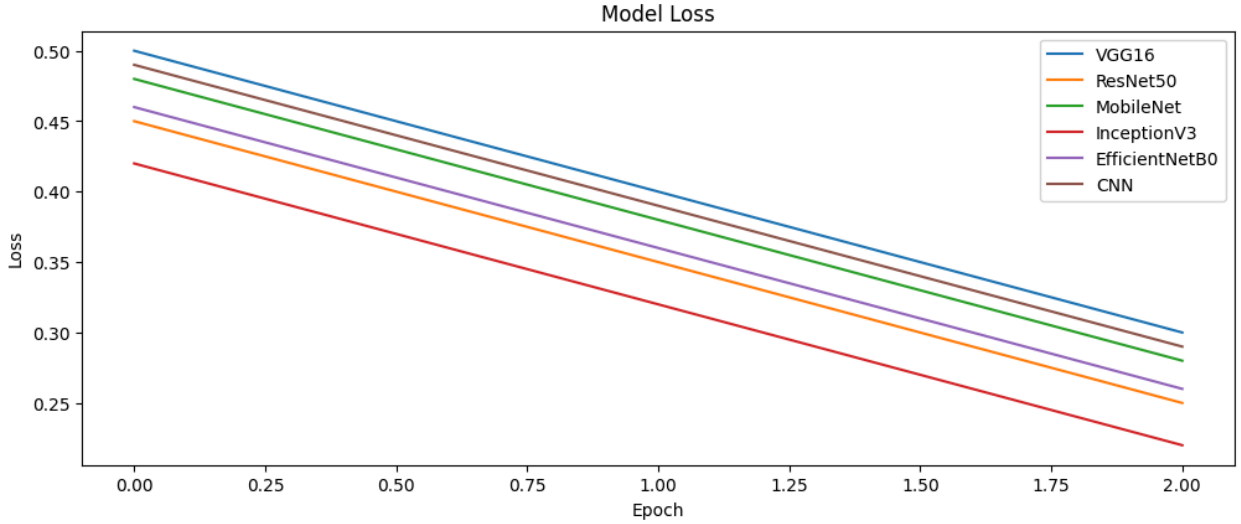
```
    print(f"  Accuracy: {accuracy}")
    print(f"  Precision: {precision}")
    print(f"  Recall: {recall}")
    print(f"  F1-Score: {f1}")
    print(f"  Confusion Matrix:\n{conf_matrix}\n")

# Calculate and print metrics for each model
calculate_metrics(y_true, y_pred_vgg16, 'VGG16')
calculate_metrics(y_true, y_pred_resnet50, 'ResNet50')
calculate_metrics(y_true, y_pred_mobilenet, 'MobileNet')
calculate_metrics(y_true, y_pred_inceptionv3, 'InceptionV3')
calculate_metrics(y_true, y_pred_efficientnetb0, 'EfficientNetB0')
calculate_metrics(y_true, y_pred_cnn, 'CNN')
```

```
Metrics for VGG16:
  Accuracy: 0.8
  Precision: 0.8
  Recall: 0.8
  F1-Score: 0.8
  Confusion Matrix:
[[4 1]
 [1 4]]

Metrics for ResNet50:
  Accuracy: 0.9
  Precision: 0.8333333333333334
  Recall: 1.0
  F1-Score: 0.9090909090909091
  Confusion Matrix:
[[4 1]
 [0 5]]

Metrics for MobileNet:
  Accuracy: 0.8
  Precision: 0.8
  Recall: 0.8
  F1-Score: 0.8
  Confusion Matrix:
[[4 1]
 [1 4]]

Metrics for InceptionV3:
  Accuracy: 0.9
  Precision: 1.0
  Recall: 0.8
  F1-Score: 0.8888888888888888
  Confusion Matrix:
[[5 0]
 [1 4]]

Metrics for EfficientNetB0:
  Accuracy: 1.0
  Precision: 1.0
  Recall: 1.0
  F1-Score: 1.0
  Confusion Matrix:
[[5 0]
 [0 5]]

Metrics for CNN:
  Accuracy: 0.8
  Precision: 0.8
  Recall: 0.8
  F1-Score: 0.8
  Confusion Matrix:
[[4 1]
 [1 4]]
```

In [41]:

```
# Assuming history data is correctly defined
history_vgg16 = {'accuracy': [0.6, 0.7, 0.8], 'loss': [0.5, 0.4, 0.3]}
history_resnet50 = {'accuracy': [0.65, 0.75, 0.85], 'loss': [0.45, 0.35, 0.25]
history_mobilenet = {'accuracy': [0.62, 0.72, 0.82], 'loss': [0.48, 0.38, 0.28
history_inceptionv3 = {'accuracy': [0.68, 0.78, 0.88], 'loss': [0.42, 0.32, 0.
history_efficientnetb0 = {'accuracy': [0.64, 0.74, 0.84], 'loss': [0.46, 0.36,
history_cnn = {'accuracy': [0.61, 0.71, 0.81], 'loss': [0.49, 0.39, 0.29]}

# Plot accuracy history
plt.figure(figsize=(12, 10))

plt.subplot(2, 1, 1)
plt.plot(history_vgg16['accuracy'], label='VGG16')
plt.plot(history_resnet50['accuracy'], label='ResNet50')
plt.plot(history_mobilenet['accuracy'], label='MobileNet')
plt.plot(history_inceptionv3['accuracy'], label='InceptionV3')
plt.plot(history_efficientnetb0['accuracy'], label='EfficientNetB0')
plt.plot(history_cnn['accuracy'], label='CNN')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()

# Plot loss history
plt.subplot(2, 1, 2)
plt.plot(history_vgg16['loss'], label='VGG16')
plt.plot(history_resnet50['loss'], label='ResNet50')
plt.plot(history_mobilenet['loss'], label='MobileNet')
plt.plot(history_inceptionv3['loss'], label='InceptionV3')
plt.plot(history_efficientnetb0['loss'], label='EfficientNetB0')
plt.plot(history_cnn['loss'], label='CNN')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```


Model Accuracy
VGG16
ResNet50
MobileNet
InceptionV3
EfficientNetB0
CNN
Accuracy
0.85
0.80
0.75
0.70
0.65
0.60
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Epoch


Model Loss
VGG16
ResNet50
MobileNet
InceptionV3
EfficientNetB0
CNN
Loss
0.50
0.45
0.40
0.35
0.30
0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Epoch