Double-click (or enter) to edit

1. File Integrity

- No corrupted or unreadable image files.
- All images open correctly and are in supported formats (.jpg, .png, etc.).

```
Start coding or generate with AI.
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
from google.colab import drive
drive.mount('/content/drive')
import os
from PIL import Image

def check_image_status(directory):
    if not os.path.isdir(directory):
        print(f"Error: Directory not found at {directory}")
        return

    print(f"Checking images in directory: {directory}")
    for filename in os.listdir(directory):
        filepath = os.path.join(directory, filename)
        if os.path.isfile(filepath):
            try:
                img = Image.open(filepath)
                img.verify()  # Verify that the file is an image
                print(f"{filename}: NOT corrupted")
            except (IOError, SyntaxError) as e:
                print(f"{filename}: IS corrupted - {e}")
            except Exception as e:
                print(f"{filename}: Could not process - {e}")
        else:
            print(f"{filename}: IS NOT a file (skipping)")
```

```
address='/content/drive/My Drive/PROJECT1/data/data_set(15,15,70)'
class_es=['test','train','valid']
sub_class_es=['Immature','Mature','Normal']
```

```
for cls in class_es:
  for sub_cls in sub_class_es:
    print(address+'/'+cls+'/'+sub_cls)
    check_image_status(address+'/'+cls+'/'+sub_cls)
```

```
Streaming output truncated to the last 5000 lines.
Immature (492).jpg: NOT corrupted
Immature (493).jpg: NOT corrupted
Immature (494).jpg: NOT corrupted
Immature (495).jpg: NOT corrupted
Immature (496).jpg: NOT corrupted
Immature (497).jpg: NOT corrupted
Immature (498).jpg: NOT corrupted
Immature (499).jpg: NOT corrupted
Immature (500).jpg: NOT corrupted
Immature (501).jpg: NOT corrupted
Immature (502).jpg: NOT corrupted
Immature (503).jpg: NOT corrupted
Immature (504).jpg: NOT corrupted
Immature (505).jpg: NOT corrupted
Immature (506).jpg: NOT corrupted
Immature (507).jpg: NOT corrupted
Immature (508).jpg: NOT corrupted
Immature (509).jpg: NOT corrupted
Immature (510).jpg: NOT corrupted
Immature (511).jpg: NOT corrupted
Immature (512).jpg: NOT corrupted
Immature (513).jpg: NOT corrupted
Immature (514).jpg: NOT corrupted
```

```
Immature (515).jpg: NOT corrupted
Immature (516).jpg: NOT corrupted
Immature (517).jpg: NOT corrupted
Immature (518).jpg: NOT corrupted
Immature (519).jpg: NOT corrupted
Immature (520).jpg: NOT corrupted
Immature (521).jpg: NOT corrupted
Immature (522).jpg: NOT corrupted
Immature (523).jpg: NOT corrupted
Immature (524).jpg: NOT corrupted
Immature (525).jpg: NOT corrupted
Immature (526).jpg: NOT corrupted
Immature (527).jpg: NOT corrupted
Immature (528).jpg: NOT corrupted
Immature (529).jpg: NOT corrupted
Immature (530).jpg: NOT corrupted
Immature (531).jpg: NOT corrupted
Immature (532).jpg: NOT corrupted
Immature (533).jpg: NOT corrupted
Immature (534).jpg: NOT corrupted
Immature (535).jpg: NOT corrupted
Immature (536).jpg: NOT corrupted
Immature (537).jpg: NOT corrupted
Immature (538).jpg: NOT corrupted
Immature (539).jpg: NOT corrupted
Immature (540).jpg: NOT corrupted
Immature (541).jpg: NOT corrupted
Immature (542).jpg: NOT corrupted
Immature (543).jpg: NOT corrupted
Immature (544).jpg: NOT corrupted
Immature (545).jpg: NOT corrupted
Immature (546).jpg: NOT corrupted
Immature (547).jpg: NOT corrupted
Immature (548).jpg: NOT corrupted
```

2. Consistent Structure

- Folder names match class labels exactly (immature, mature, normal).
- No extra or misplaced files (e.g., .txt, .DS_Store, etc.).
- Each folder contains only images of that class.

```
import os

def check_folder_names(base_directory, expected_folders):
    """
    Checks if the folders in a base directory match the expected list of folder names.

    Args:
        base_directory (str): The path to the base directory.
        expected_folders (list): A list of expected folder names.
    """
    print(f"Checking folder names in base directory: {base_directory}")

    if not os.path.isdir(base_directory):
        print(f"Error: Base directory not found at {base_directory}")
        return

    found_folders = set([item for item in os.listdir(base_directory) if os.path.isdir(os.path.join(base_directory, item))])
    expected_folders_set = set(expected_folders)

    missing_folders = expected_folders_set - found_folders
    if missing_folders:
        print("Warning: Expected folders not found:")
        for folder in missing_folders:
            print(f"- {folder}")

    unexpected_folders = found_folders - expected_folders_set
    if unexpected_folders:
        print("Warning: Unexpected folders found:")
        for folder in unexpected_folders:
            print(f"- {folder}")

    if not missing_folders and not unexpected_folders:
        print("All expected folders found and no unexpected folders.")
```

```
def check_image_files_only(directory):
    """
    Checks if a directory contains only image files (based on common extensions).

    Args:
        directory (str): The path to the directory to check.
    """
    print(f"Checking files in directory: {directory}")

    if not os.path.isdir(directory):
        print(f"Error: Directory not found at {directory}")
        return

    non_image_files_found = False
    for filename in os.listdir(directory):
        filepath = os.path.join(directory, filename)
        if os.path.isfile(filepath):
            # Check if the file is an image (basic check based on extension)
            if not filename.lower().endswith(('.png', '.jpg', '.jpeg')):
                print(f"Warning: Non-image file found: {filename}")
                non_image_files_found = True
        else:
            print(f"Info: Non-file item found (skipping): {filename}")

    if not non_image_files_found:
        print("All files appear to be images.")

# Example usage (using variables from your notebook):
# address='/content/drive/My Drive/PROJECT1/data/data_set(15,15,70)'
# class_es=['test','train','valid']
# sub_class_es=['Immature','Mature','Normal']

# Check the main class folders:
#

# Check the sub-class folders within each class folder:
#

# Check the files within each sub-class folder:
#
```

```
check_folder_names(address, class_es)
```

```
Checking folder names in base directory: /content/drive/My Drive/PROJECT1/data/data_set(15,15,70)
All expected folders found and no unexpected folders.
```

```
for cls in class_es:
     check_folder_names(os.path.join(address, cls), sub_class_es)
```

```
Checking folder names in base directory: /content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/test
All expected folders found and no unexpected folders.
Checking folder names in base directory: /content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/train
All expected folders found and no unexpected folders.
Checking folder names in base directory: /content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/valid
All expected folders found and no unexpected folders.
```

```
for cls in class_es:
     for sub_cls in sub_class_es:
         check_image_files_only(os.path.join(address, cls, sub_cls))
```

```
Checking files in directory: /content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/test/Immature
All files appear to be images.
Checking files in directory: /content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/test/Mature
All files appear to be images.
Checking files in directory: /content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/test/Normal
All files appear to be images.
Checking files in directory: /content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/train/Immature
All files appear to be images.
Checking files in directory: /content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/train/Mature
All files appear to be images.
Checking files in directory: /content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/train/Normal
All files appear to be images.
Checking files in directory: /content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/valid/Immature
All files appear to be images.
Checking files in directory: /content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/valid/Mature
All files appear to be images.
Checking files in directory: /content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/valid/Normal
All files appear to be images.
```

```
import os

def check_filename_label_accuracy(base_directory, class_es, sub_class_es):
       print(f"Checking filename label accuracy in base directory: {base_directory}")
    if not os.path.isdir(base_directory):
        print(f"Error: Base directory not found at {base_directory}")
        return
    for cls in class_es:
        class_folder_path = os.path.join(base_directory, cls)
        if not os.path.isdir(class_folder_path):
            print(f"Warning: Class folder not found (skipping): {class_folder_path}")
            continue
        for sub_cls in sub_class_es:
            sub_class_folder_path = os.path.join(class_folder_path, sub_cls)
            print(f"\nChecking folder: {sub_class_folder_path}")
            if not os.path.isdir(sub_class_folder_path):
                print(f"Warning: Sub-class folder not found (skipping): {sub_class_folder_path}")
                continue
            for filename in os.listdir(sub_class_folder_path):
                filepath = os.path.join(sub_class_folder_path, filename)
                if os.path.isfile(filepath):
                    # Get the folder name (sub_cls) and check if it's in the filename (case-insensitive)
                    if sub_cls.lower() not in filename.lower():
                        print(f"Warning: Filename '{filename}' in folder '{sub_cls}' does not contain the folder name.")
                    else:
                        print(f"Filename '{filename}' in folder '{sub_cls}' contains the folder name.")
                else:
                    print(f"Info: Non-file item found (skipping): {filename}")
# Example usage (using variables from your notebook):
# address='/content/drive/My Drive/PROJECT1/data/data_set(15,15,70)'
# class_es=['test','train','valid']
# sub_class_es=['Immature','Mature','Normal']
```

```
check_filename_label_accuracy(address, class_es, sub_class_es)
```

```
Streaming output truncated to the last 5000 lines.
Filename 'Immature (492).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (493).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (494).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (495).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (496).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (497).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (498).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (499).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (500).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (501).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (502).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (503).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (504).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (505).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (506).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (507).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (508).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (509).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (510).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (511).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (512).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (513).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (514).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (515).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (516).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (517).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (518).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (519).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (520).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (521).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (522).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (523).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (524).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (525).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (526).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (527).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (528).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (529).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (530).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (531).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (532).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (533).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (534).jpg' in folder 'Immature' contains the folder name.
```

```
Filename 'Immature (535).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (536).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (537).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (538).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (539).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (540).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (541).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (542).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (543).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (544).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (545).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (546).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (547).jpg' in folder 'Immature' contains the folder name.
Filename 'Immature (548).jpg' in folder 'Immature' contains the folder name.
```

```
def check_image_size_consistency(directory, expected_size):
    print(f"Checking image sizes in directory: {directory}")

    if not os.path.isdir(directory):
        print(f"Error: Directory not found at {directory}")
        return

    size_mismatch_found = False
    for filename in os.listdir(directory):
        filepath = os.path.join(directory, filename)
        if os.path.isfile(filepath):
            try:
                img = Image.open(filepath)
                if img.size != expected_size:
                    print(f"Warning: Image '{filename}' has size {img.size}, expected {expected_size}.")
                    size_mismatch_found = True
            except IOError:
                print(f"Error: Could not open or process image file: {filename}")
                size_mismatch_found = True # Consider inability to open as a mismatch for the success message
        else:
            print(f"Info: Non-file item found (skipping): {filename}")


    if not size_mismatch_found:
        print(f"All images in directory '{directory}' are of expected size {expected_size}.")


# Checking for pixel value normalization (e.g., divided by 255) is more complex
# and typically done during the data loading and preprocessing steps for model training.
# It's not something easily checked by just looking at the image files themselves
# without knowing the original pixel range.
```

```
expected_image_size = (224, 224)
```

```
for cls in class_es:
  for sub_cls in sub_class_es:
    check_image_size_consistency(os.path.join(address, cls, sub_cls), expected_image_size)
```

```
Checking image sizes in directory: /content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/test/Immature
All images in directory '/content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/test/Immature' are of expected size (224, 224).
Checking image sizes in directory: /content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/test/Mature
All images in directory '/content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/test/Mature' are of expected size (224, 224).
Checking image sizes in directory: /content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/test/Normal
All images in directory '/content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/test/Normal' are of expected size (224, 224).
Checking image sizes in directory: /content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/train/Immature
All images in directory '/content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/train/Immature' are of expected size (224, 224
Checking image sizes in directory: /content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/train/Mature
All images in directory '/content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/train/Mature' are of expected size (224, 224).
Checking image sizes in directory: /content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/train/Normal
All images in directory '/content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/train/Normal' are of expected size (224, 224).
Checking image sizes in directory: /content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/valid/Immature
All images in directory '/content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/valid/Immature' are of expected size (224, 224
Checking image sizes in directory: /content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/valid/Mature
All images in directory '/content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/valid/Mature' are of expected size (224, 224).
Checking image sizes in directory: /content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/valid/Normal
All images in directory '/content/drive/My Drive/PROJECT1/data/data_set(15,15,70)/valid/Normal' are of expected size (224, 224).
```

```
import os
import pandas as pd
```

```
def check_class_distribution(base_directory, class_es, sub_class_es):
    print(f"Checking class distribution in base directory: {base_directory}")
    if not os.path.isdir(base_directory):
        print(f"Error: Base directory not found at {base_directory}")
        return
    distribution_data = {}
    total_images = 0
    for cls in class_es:
        class_folder_path = os.path.join(base_directory, cls)
        if not os.path.isdir(class_folder_path):
            print(f"Warning: Class folder not found (skipping): {class_folder_path}")
            continue
        distribution_data[cls] = {}
        class_total = 0
        for sub_cls in sub_class_es:
            sub_class_folder_path = os.path.join(class_folder_path, sub_cls)
            if not os.path.isdir(sub_class_folder_path):
                print(f"Warning: Sub-class folder not found (skipping): {sub_class_folder_path}")
                distribution_data[cls][sub_cls] = 0
                continue
            image_count = 0
            for filename in os.listdir(sub_class_folder_path):
                if os.path.isfile(os.path.join(sub_class_folder_path, filename)) and filename.lower().endswith(('.png', '.jpg',
                    image_count += 1
            distribution_data[cls][sub_cls] = image_count
            class_total += image_count
        distribution_data[cls]['Total'] = class_total
        total_images += class_total
    df_distribution = pd.DataFrame(distribution_data)
    df_distribution['Sub-class Total'] = df_distribution.loc[sub_class_es].sum(axis=1)
    print("\nImage Distribution per Sub-class and Class:")
    display(df_distribution)
    print(f"\nTotal number of images found: {total_images}")
```

```
check_class_distribution(address, class_es, sub_class_es)
```

```
Checking class distribution in base directory: /content/drive/My Drive/PROJECT1/data/data_set(15,15,70)

Image Distribution per Sub-class and Class:
```

	test	train	valid	Sub-class Total
Immature	281	1317	185	1783.0
Mature	320	1499	321	2140.0
Normal	324	1510	332	2166.0
Total	925	4326	838	NaN

```
Total number of images found: 6089
```

MOUNT DRIVE

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

IMPORT LIBRARIES

```
import tensorflow as tf
from tensorflow.keras.preprocessing import image
import numpy as np
import pandas as pd
from tensorflow.keras.applications.densenet import preprocess_input as preprocess_densenet
from tensorflow.keras.applications.efficientnet import preprocess_input as preprocess_efficientnet
from tensorflow.keras.applications.resnet50 import preprocess_input as preprocess_resnet
from tensorflow.keras.applications.vgg19 import preprocess_input as preprocess_vgg19
```

DEFINE CLASS NAMES

```
# CELL 3: DEFINE CLASS NAMES
CLASS_NAMES = ['Immature', 'Mature', 'Normal']
```

DEFINE MODEL PATHS AND IMAGE PATH

```
model_paths = {
    'densenet121': '/content/drive/MyDrive/PROJECT/PROJECT-1/trained models/H5 files/densenet121(99_03).h5',
    'efficientnetb3': '/content/drive/MyDrive/PROJECT/PROJECT-1/trained models/H5 files/efficientnetb3(99_84).h5',
    'resnet': '/content/drive/MyDrive/PROJECT/PROJECT-1/trained models/H5 files/resnet(99_91).h5',
    'vgg19': '/content/drive/MyDrive/PROJECT/PROJECT-1/trained models/H5 files/vgg19(99_89).h5'
}
confidence_dict = {}
predictions = {}
acc_dic = {'densenet121': 99.03, 'efficientnetb3': 99.84, 'resnet': 99.91, 'vgg19': 99.89}

img_path = input("image path:")
```

```
image path:/content/drive/MyDrive/PROJECT/PROJECT-1/Input Images/testcase2(n).jpg
```

MODEL:DENSENET121 PREDICTION

```
print("Running DenseNet121...")
IMG_SIZE = (224, 224)
model = tf.keras.models.load_model(model_paths['densenet121'], compile=False)
img = image.load_img(img_path, target_size=IMG_SIZE)
x = image.img_to_array(img)
x = preprocess_densenet(x)
x = np.expand_dims(x, axis=0)
preds = model.predict(x)[0]
predictions['densenet121'] = preds
idx = np.argmax(preds)
label = CLASS_NAMES[idx]
confidence = float(preds[idx])
confidence_dict['densenet121'] = confidence
print(f"DenseNet121 Prediction: {label} (Confidence: {confidence:.2%})")
```

```
Running DenseNet121...
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
DenseNet121 Prediction: Normal (Confidence: 99.90%)
```

MODEL:EFFICIENTNETB3 PREDICTION

```
print("Running EfficientNetB3...")
IMG_SIZE = (224, 224)
try:
    model = tf.keras.models.load_model(model_paths['efficientnetb3'], compile=False)
    img = image.load_img(img_path, target_size=IMG_SIZE)
    x = image.img_to_array(img)
    x = preprocess_efficientnet(x)
    x = np.expand_dims(x, axis=0)
    preds = model.predict(x)[0]
    predictions['efficientnetb3'] = preds
    idx = np.argmax(preds)
    label = CLASS_NAMES[idx]
    confidence = float(preds[idx])
    confidence_dict['efficientnetb3'] = confidence
    print(f"EfficientNetB3 Prediction: {label} (Confidence: {confidence:.2%})")
except Exception as e:
    print("Error loading or running EfficientNetB3:", e)
```

```
Running EfficientNetB3...
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
EfficientNetB3 Prediction: Normal (Confidence: 100.00%)
```

MODEL:RESNET PREDICTION

```
print("Running ResNet...")
IMG_SIZE = (224, 224)
model = tf.keras.models.load_model(model_paths['resnet'], compile=False)
img = image.load_img(img_path, target_size=IMG_SIZE)
x = image.img_to_array(img)
x = preprocess_resnet(x)
x = np.expand_dims(x, axis=0)
preds = model.predict(x)[0]
predictions['resnet'] = preds
idx = np.argmax(preds)
label = CLASS_NAMES[idx]
confidence = float(preds[idx])
confidence_dict['resnet'] = confidence
print(f"ResNet Prediction: {label} (Confidence: {confidence:.2%})")
```

```
Running ResNet...
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
ResNet Prediction: Normal (Confidence: 100.00%)
```

MODEL:VGG19 PREDICTION

```
print("Running VGG19...")
IMG_SIZE = (224, 224)
model = tf.keras.models.load_model(model_paths['vgg19'], compile=False)
img = image.load_img(img_path, target_size=IMG_SIZE)
x = image.img_to_array(img)
x = preprocess_vgg19(x)
x = np.expand_dims(x, axis=0)
preds = model.predict(x)[0]
predictions['vgg19'] = preds
idx = np.argmax(preds)
label = CLASS_NAMES[idx]
confidence = float(preds[idx])
confidence_dict['vgg19'] = confidence
print(f"VGG19 Prediction: {label} (Confidence: {confidence:.2%})")
```

```
Running VGG19...
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 874ms/step
VGG19 Prediction: Normal (Confidence: 100.00%)
```

DISPLAY INDIVIDUAL MODEL PREDICTIONS

```
print("\n" + "="*70)
print("SUMMARY: Individual Model Predictions")
print("="*70)
for model_name, conf in confidence_dict.items():
    pred_class = CLASS_NAMES[np.argmax(predictions[model_name])]
    print(f"{model_name:20s}: {pred_class:10s} - {conf:.4f} ({conf:.2%})")
print("="*70)
print(f"\nTotal models run: {len(confidence_dict)}")
```

```
======================================================================
SUMMARY: Individual Model Predictions
======================================================================
densenet121         : Normal     - 0.9990 (99.90%)
efficientnetb3      : Normal     - 1.0000 (100.00%)
resnet              : Normal     - 1.0000 (100.00%)
vgg19               : Normal     - 1.0000 (100.00%)
======================================================================

Total models run: 4
```

CONFIDENCE MATRIX AND FINAL ENSEMBLE PREDICTION

```
print("\n\n" + "="*70)
print("CONFIDENCE MATRIX (Class × Model)")
print("="*70)
confidence_matrix = []
for class_name in CLASS_NAMES:
    row = []
    for model_name in model_paths.keys():
        class_idx = CLASS_NAMES.index(class_name)
        confidence = predictions[model_name][class_idx]
        row.append(confidence)
    confidence_matrix.append(row)
conf_df = pd.DataFrame(confidence_matrix, index=CLASS_NAMES, columns=list(model_paths.keys()))
print(conf_df.round(4))
print("="*70)
total_accuracy = sum(acc_dic.values())
weighted_confidences = {}
for class_name in CLASS_NAMES:
    class_idx = CLASS_NAMES.index(class_name)
    weighted_sum = 0
    for model_name in model_paths.keys():
        model_confidence = predictions[model_name][class_idx]
        model_weight = acc_dic[model_name] / total_accuracy
        weighted_sum += model_confidence * model_weight
    weighted_confidences[class_name] = weighted_sum
print("\n" + "="*70)
print("WEIGHTED CONFIDENCE FOR EACH CLASS")
print("-"*70)
for class_name, conf in weighted_confidences.items():
    print(f"{class_name:12s}: {conf:.4f} ({conf:.2%})")
print("-"*70)
final_class = max(weighted_confidences, key=weighted_confidences.get)
final_confidence = weighted_confidences[final_class]
print("\n" + "-"*70)
print("FINAL ENSEMBLE PREDICTION")
print("="*70)
print(f"Predicted Class: {final_class}")
print(f"Ensemble Confidence: {final_confidence:.4f} ({final_confidence:.2%})")
```

```
======================================================================
CONFIDENCE MATRIX (Class × Model)
======================================================================
          densenet121  efficientnetb3  resnet  vgg19
Immature        0.001             0.0     0.0    0.0
Mature          0.000             0.0     0.0    0.0
Normal          0.999             1.0     1.0    1.0
======================================================================

======================================================================
WEIGHTED CONFIDENCE FOR EACH CLASS
----------------------------------------------------------------------
Immature    : 0.0002 (0.02%)
```

```
Mature      : 0.0000 (0.00%)
Normal      : 0.9998 (99.98%)
----------------------------------------------------------------------

----------------------------------------------------------------------
FINAL ENSEMBLE PREDICTION
======================================================================
Predicted Class: Normal
Ensemble Confidence: 0.9998 (99.98%)
```

CHECK FOR MODEL DISAGREEMENTS

```
individual_preds = {name: CLASS_NAMES[np.argmax(predictions[name])] for name in model_paths.keys()}
if len(set(individual_preds.values())) > 1:
    print(f"MODEL DISAGREEMENT DETECTED:")
    for model, pred in individual_preds.items():
        print(f"  {model:20s}: {pred}")
    print("Resolved using weighted confidence matrix approach")
else:
    print(f"All models agree on: {final_class}")
print("="*70)
```

```
All models agree on: Normal
======================================================================
```

Mount Google Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

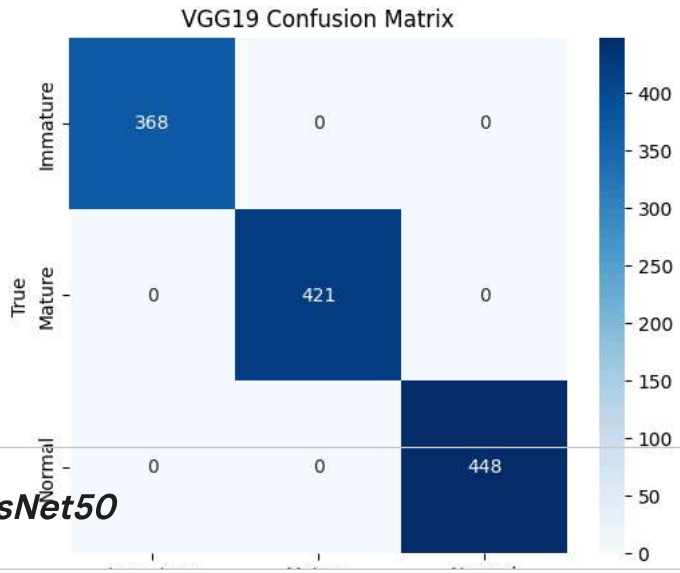
IMPORT

```
# Constants
TEST_DATA_DIR = "/content/drive/My Drive/PROJECT1/data/data_set_20_20_60/test"
IMG_SIZE = (224, 224)

# Core imports
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Preprocessing imports from different architectures
from tensorflow.keras.applications.efficientnet import preprocess_input as efficientnet_preprocess
from tensorflow.keras.applications.mobilenet_v3 import preprocess_input as mobilenetv3_preprocess
from tensorflow.keras.applications.resnet50 import preprocess_input as resnet50_preprocess
from tensorflow.keras.applications.densenet import preprocess_input as densenet_preprocess
from tensorflow.keras.applications.inception_resnet_v2 import preprocess_input as inception_resnet_preprocess
from tensorflow.keras.applications.vgg19 import preprocess_input as vgg19_preprocess
```

VGG19

```
test_datagen = ImageDataGenerator(preprocessing_function=vgg19_preprocess)
test_data = test_datagen.flow_from_directory(
    TEST_DATA_DIR,
    target_size=IMG_SIZE,
    batch_size=32,
    class_mode="categorical",
    shuffle=False
)

model_path = "/content/drive/My Drive/PROJECT1/trained models/H5/vgg19(99_89).h5"
model = tf.keras.models.load_model(model_path)

y_true = test_data.classes
y_pred = np.argmax(model.predict(test_data), axis=1)
label_names = list(test_data.class_indices.keys())

print("\nClassification Report:")
print(classification_report(y_true, y_pred, target_names=label_names))

cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=label_names, yticklabels=label_names)
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("VGG19 Confusion Matrix")
plt.show()
```

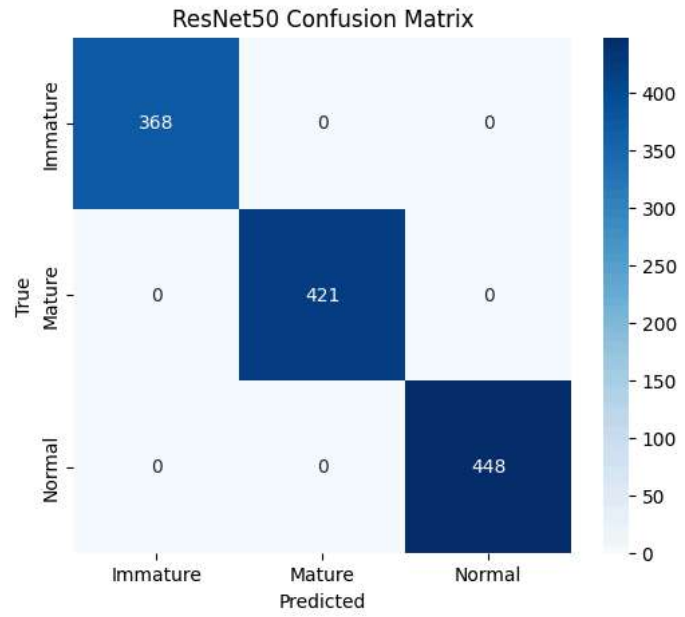
```
Found 1237 images belonging to 3 classes.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty unt
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset
  self._warn_if_super_not_called()
39/39 ━━━━━━━━━━━━━━━━━━━━ 9s 209ms/step

Classification Report:
              precision    recall  f1-score   support

    Immature       1.00      1.00      1.00       368
      Mature       1.00      1.00      1.00       421
      Normal       1.00      1.00      1.00       448

    accuracy                           1.00      1237
   macro avg       1.00      1.00      1.00      1237
weighted avg       1.00      1.00      1.00      1237
```



ResNet50

```
test_datagen = ImageDataGenerator(preprocessing_function=resnet50_preprocess)
test_data = test_datagen.flow_from_directory(
    TEST_DATA_DIR,
    target_size=IMG_SIZE,
    batch_size=32,
    class_mode="categorical",
    shuffle=False
)

model_path = "/content/drive/My Drive/PROJECT1/trained models/H5/resnet(99_91).h5"
model = tf.keras.models.load_model(model_path)

y_true = test_data.classes
y_pred = np.argmax(model.predict(test_data), axis=1)
label_names = list(test_data.class_indices.keys())

print("\nClassification Report:")
print(classification_report(y_true, y_pred, target_names=label_names))

cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=label_names, yticklabels=label_names)
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("ResNet50 Confusion Matrix")
plt.show()
```

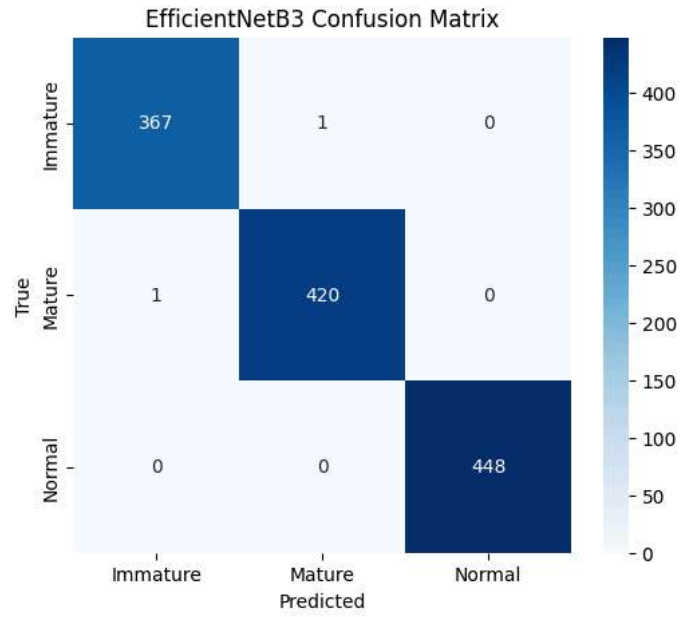
```
Found 1237 images belonging to 3 classes.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty unt
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset
  self._warn_if_super_not_called()
39/39 ━━━━━━━━━━━━━━━━━━━━ 14s 240ms/step

Classification Report:
              precision    recall  f1-score   support

    Immature       1.00      1.00      1.00       368
      Mature       1.00      1.00      1.00       421
      Normal       1.00      1.00      1.00       448

    accuracy                           1.00      1237
   macro avg       1.00      1.00      1.00      1237
weighted avg       1.00      1.00      1.00      1237
```



EfficientNetB3

```
test_datagen = ImageDataGenerator(preprocessing_function=efficientnet_preprocess)
test_data = test_datagen.flow_from_directory(
    TEST_DATA_DIR,
    target_size=IMG_SIZE,
    batch_size=32,
    class_mode="categorical",
    shuffle=False
)

model_path = "/content/drive/My Drive/PROJECT1/trained models/H5/efficientnetb3(99_84).h5"
model = tf.keras.models.load_model(model_path)

y_true = test_data.classes
y_pred = np.argmax(model.predict(test_data), axis=1)
label_names = list(test_data.class_indices.keys())

print("\nClassification Report:")
print(classification_report(y_true, y_pred, target_names=label_names))

cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=label_names, yticklabels=label_names)
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("EfficientNetB3 Confusion Matrix")
plt.show()
```

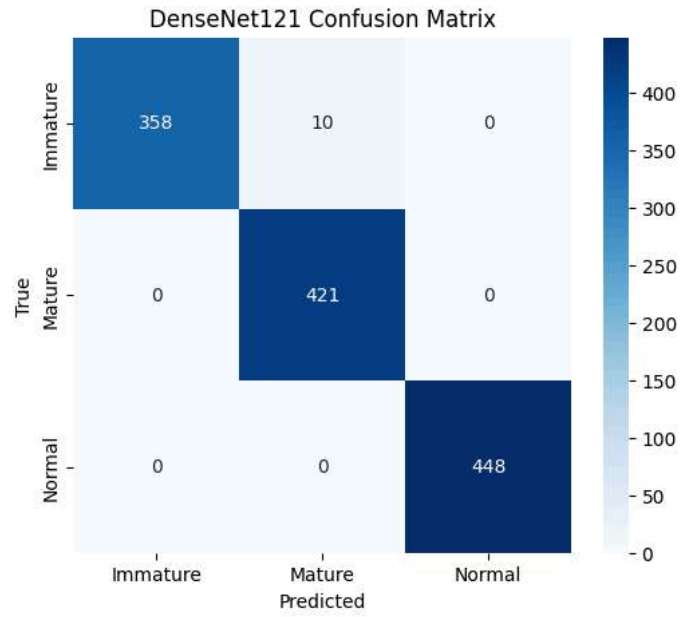
```
Found 1237 images belonging to 3 classes.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty unt
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset
  self._warn_if_super_not_called()
39/39 ━━━━━━━━━━━━━━━━━━━━ 24s 369ms/step

Classification Report:
              precision    recall  f1-score   support

    Immature       1.00      1.00      1.00       368
      Mature       1.00      1.00      1.00       421
      Normal       1.00      1.00      1.00       448

    accuracy                           1.00      1237
   macro avg       1.00      1.00      1.00      1237
weighted avg       1.00      1.00      1.00      1237
```



DenseNet121

```
test_datagen = ImageDataGenerator(preprocessing_function=densenet_preprocess)
test_data = test_datagen.flow_from_directory(
    TEST_DATA_DIR,
    target_size=IMG_SIZE,
    batch_size=32,
    class_mode="categorical",
    shuffle=False
)

model_path = "/content/drive/My Drive/PROJECT1/trained models/H5/densenet121(99_19).h5"
model = tf.keras.models.load_model(model_path)

y_true = test_data.classes
y_pred = np.argmax(model.predict(test_data), axis=1)
label_names = list(test_data.class_indices.keys())

print("\nClassification Report:")
print(classification_report(y_true, y_pred, target_names=label_names))

cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=label_names, yticklabels=label_names)
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("DenseNet121 Confusion Matrix")
plt.show()
```

```
Found 1237 images belonging to 3 classes.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty unt
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset
  self._warn_if_super_not_called()
39/39 ━━━━━━━━━━━━━━━━━━━━ 28s 426ms/step

Classification Report:
              precision    recall  f1-score   support

    Immature       1.00      0.97      0.99       368
      Mature       0.98      1.00      0.99       421
      Normal       1.00      1.00      1.00       448

    accuracy                           0.99      1237
   macro avg       0.99      0.99      0.99      1237
weighted avg       0.99      0.99      0.99      1237
```



1. Imports and Drive Mount

```
import os
import numpy as np
import tensorflow as tf
from sklearn.utils import class_weight
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
import matplotlib.pyplot as plt
```

Mounted Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

2. Paths and Hyperparameters

```
DATA_DIR = "/content/drive/My Drive/PROJECT1/data/data_set_20_20_60"
IMG_SIZE = (224, 224)
BATCH_SIZE = 32
EPOCHS_HEAD = 15
EPOCHS_FINE = 25
CLASS_NAMES = ["Immature", "Mature", "Normal"]

SAVE_DIR_H5 = "/content/drive/My Drive/PROJECT1/trained models/H5"
SAVE_DIR_KERAS = "/content/drive/My Drive/PROJECT1/trained models/keras"
os.makedirs(SAVE_DIR_H5, exist_ok=True)
os.makedirs(SAVE_DIR_KERAS, exist_ok=True)
```

3. Data Generators and Class Mapping

```
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.applications.densenet import preprocess_input

train_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    horizontal_flip=True,
    rotation_range=15,
    zoom_range=0.1,
    width_shift_range=0.1,
    height_shift_range=0.1
)
val_test_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)

train_data = train_datagen.flow_from_directory(
    os.path.join(DATA_DIR, "train"),
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical"
)
valid_data = val_test_datagen.flow_from_directory(
    os.path.join(DATA_DIR, "valid"),
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=False
)
test_data = val_test_datagen.flow_from_directory(
    os.path.join(DATA_DIR, "test"),
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical",
```

```
    shuffle=False
)

print("Class indices:", train_data.class_indices)
```

```
Found 3712 images belonging to 3 classes.
Found 1237 images belonging to 3 classes.
Found 1237 images belonging to 3 classes.
Class indices: {'Immature': 0, 'Mature': 1, 'Normal': 2}
```

4. Compute Class Weights

```
class_weights = class_weight.compute_class_weight(
    class_weight='balanced',
    classes=np.unique(train_data.classes),
    y=train_data.classes
)
class_weights = dict(enumerate(class_weights))
```

5. Model Creation and Compilation

```
base_model = DenseNet121(weights='imagenet', include_top=False, input_shape=(*IMG_SIZE, 3))
base_model.trainable = False

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.3)(x)
predictions = Dense(len(CLASS_NAMES), activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)

model.compile(optimizer=Adam(learning_rate=1e-3), loss="categorical_crossentropy", metrics=["accuracy"])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet121_weights_tf_dim_ordering_
29084464/29084464 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
```

6. Callbacks Setup

```
es = EarlyStopping(monitor="val_loss", patience=6, restore_best_weights=True)
rlr = ReduceLROnPlateau(monitor="val_loss", factor=0.2, patience=3, min_lr=1e-6)
```

7. Head Training

```
history_head = model.fit(
    train_data,
    validation_data=valid_data,
    epochs=EPOCHS_HEAD,
    class_weight=class_weights,
    callbacks=[es, rlr]
)
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset
  self._warn_if_super_not_called()
Epoch 1/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 2664s 23s/step - accuracy: 0.4365 - loss: 1.2791 - val_accuracy: 0.7494 - val_loss: 0.5963 - learni
Epoch 2/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 51s 439ms/step - accuracy: 0.6912 - loss: 0.7146 - val_accuracy: 0.8027 - val_loss: 0.4618 - learni
Epoch 3/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 48s 417ms/step - accuracy: 0.7546 - loss: 0.5799 - val_accuracy: 0.8715 - val_loss: 0.3908 - learni
Epoch 4/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 49s 418ms/step - accuracy: 0.7870 - loss: 0.5134 - val_accuracy: 0.8286 - val_loss: 0.3798 - learni
Epoch 5/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 50s 427ms/step - accuracy: 0.8024 - loss: 0.4746 - val_accuracy: 0.8230 - val_loss: 0.3675 - learni
Epoch 6/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 49s 423ms/step - accuracy: 0.7966 - loss: 0.4695 - val_accuracy: 0.8909 - val_loss: 0.3101 - learni
Epoch 7/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 48s 409ms/step - accuracy: 0.8137 - loss: 0.4386 - val_accuracy: 0.8957 - val_loss: 0.2903 - learni
Epoch 8/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 408ms/step - accuracy: 0.8403 - loss: 0.3928 - val_accuracy: 0.9281 - val_loss: 0.2634 - learni
```

```
Epoch 9/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 48s 415ms/step - accuracy: 0.8349 - loss: 0.4030 - val_accuracy: 0.9175 - val_loss: 0.2657 - learni
Epoch 10/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 48s 413ms/step - accuracy: 0.8294 - loss: 0.4010 - val_accuracy: 0.9289 - val_loss: 0.2484 - learni
Epoch 11/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 48s 416ms/step - accuracy: 0.8494 - loss: 0.3601 - val_accuracy: 0.9200 - val_loss: 0.2504 - learni
Epoch 12/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 48s 414ms/step - accuracy: 0.8215 - loss: 0.4083 - val_accuracy: 0.9434 - val_loss: 0.2275 - learni
Epoch 13/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 49s 421ms/step - accuracy: 0.8336 - loss: 0.4038 - val_accuracy: 0.9386 - val_loss: 0.2260 - learni
Epoch 14/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 49s 420ms/step - accuracy: 0.8548 - loss: 0.3608 - val_accuracy: 0.9184 - val_loss: 0.2355 - learni
Epoch 15/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 48s 413ms/step - accuracy: 0.8560 - loss: 0.3580 - val_accuracy: 0.9378 - val_loss: 0.2207 - learni
```

8. Fine-Tuning

```
for layer in base_model.layers[-50:]:
    if not isinstance(layer, tf.keras.layers.BatchNormalization):
        layer.trainable = True

model.compile(optimizer=Adam(learning_rate=1e-5), loss="categorical_crossentropy", metrics=["accuracy"])

history_fine = model.fit(
    train_data,
    validation_data=valid_data,
    epochs=EPOCHS_FINE,
    class_weight=class_weights,
    callbacks=[es, rlr]
)
```

```
Epoch 1/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 91s 572ms/step - accuracy: 0.8647 - loss: 0.3433 - val_accuracy: 0.9386 - val_loss: 0.2049 - learni
Epoch 2/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 407ms/step - accuracy: 0.8740 - loss: 0.3117 - val_accuracy: 0.9434 - val_loss: 0.1857 - learni
Epoch 3/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 406ms/step - accuracy: 0.8830 - loss: 0.2939 - val_accuracy: 0.9563 - val_loss: 0.1642 - learni
Epoch 4/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 48s 410ms/step - accuracy: 0.8889 - loss: 0.2713 - val_accuracy: 0.9531 - val_loss: 0.1550 - learni
Epoch 5/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 48s 412ms/step - accuracy: 0.8930 - loss: 0.2543 - val_accuracy: 0.9555 - val_loss: 0.1451 - learni
Epoch 6/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 48s 411ms/step - accuracy: 0.9059 - loss: 0.2490 - val_accuracy: 0.9580 - val_loss: 0.1354 - learni
Epoch 7/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 409ms/step - accuracy: 0.9299 - loss: 0.2004 - val_accuracy: 0.9620 - val_loss: 0.1243 - learni
Epoch 8/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 48s 410ms/step - accuracy: 0.9291 - loss: 0.1949 - val_accuracy: 0.9628 - val_loss: 0.1197 - learni
Epoch 9/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 404ms/step - accuracy: 0.9274 - loss: 0.1900 - val_accuracy: 0.9717 - val_loss: 0.1001 - learni
Epoch 10/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 46s 398ms/step - accuracy: 0.9432 - loss: 0.1698 - val_accuracy: 0.9774 - val_loss: 0.0899 - learni
Epoch 11/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 405ms/step - accuracy: 0.9344 - loss: 0.1665 - val_accuracy: 0.9717 - val_loss: 0.0937 - learni
Epoch 12/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 48s 409ms/step - accuracy: 0.9527 - loss: 0.1376 - val_accuracy: 0.9806 - val_loss: 0.0755 - learni
Epoch 13/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 406ms/step - accuracy: 0.9527 - loss: 0.1396 - val_accuracy: 0.9822 - val_loss: 0.0679 - learni
Epoch 14/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 401ms/step - accuracy: 0.9565 - loss: 0.1235 - val_accuracy: 0.9838 - val_loss: 0.0683 - learni
Epoch 15/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 407ms/step - accuracy: 0.9498 - loss: 0.1262 - val_accuracy: 0.9846 - val_loss: 0.0666 - learni
Epoch 16/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 48s 411ms/step - accuracy: 0.9657 - loss: 0.1090 - val_accuracy: 0.9846 - val_loss: 0.0624 - learni
Epoch 17/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 48s 410ms/step - accuracy: 0.9684 - loss: 0.0988 - val_accuracy: 0.9871 - val_loss: 0.0545 - learni
Epoch 18/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 408ms/step - accuracy: 0.9683 - loss: 0.0915 - val_accuracy: 0.9887 - val_loss: 0.0510 - learni
Epoch 19/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 404ms/step - accuracy: 0.9670 - loss: 0.0938 - val_accuracy: 0.9911 - val_loss: 0.0457 - learni
Epoch 20/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 408ms/step - accuracy: 0.9738 - loss: 0.0828 - val_accuracy: 0.9903 - val_loss: 0.0436 - learni
Epoch 21/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 46s 400ms/step - accuracy: 0.9751 - loss: 0.0828 - val_accuracy: 0.9935 - val_loss: 0.0408 - learni
Epoch 22/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 48s 411ms/step - accuracy: 0.9776 - loss: 0.0723 - val_accuracy: 0.9879 - val_loss: 0.0495 - learni
Epoch 23/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 407ms/step - accuracy: 0.9758 - loss: 0.0736 - val_accuracy: 0.9911 - val_loss: 0.0448 - learni
Epoch 24/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 405ms/step - accuracy: 0.9829 - loss: 0.0656 - val_accuracy: 0.9935 - val_loss: 0.0350 - learni
```

```
Epoch 25/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 406ms/step - accuracy: 0.9797 - loss: 0.0633 - val_accuracy: 0.9935 - val_loss: 0.0321 - learni
```

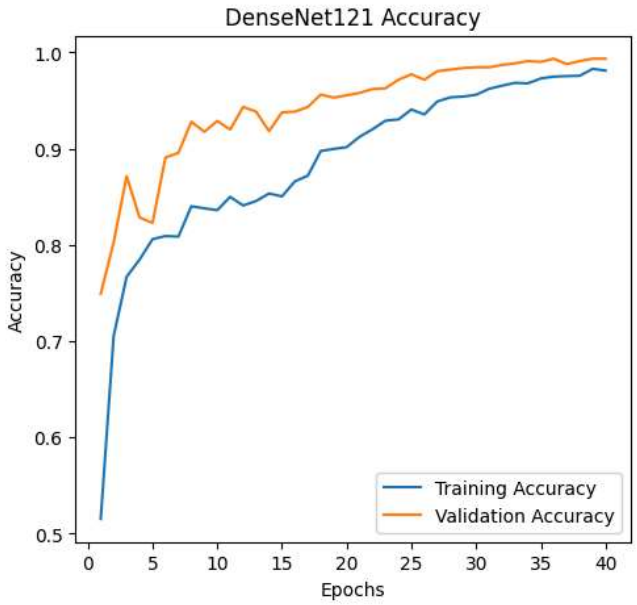
9. Evaluation and Save Model

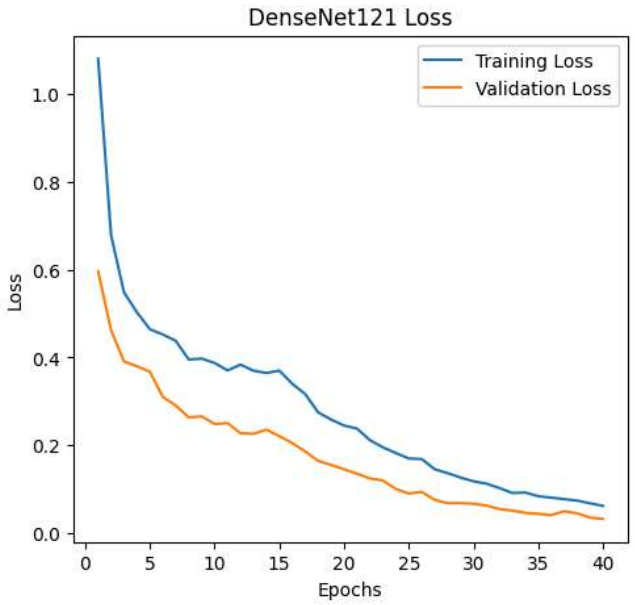
```
loss, acc = model.evaluate(test_data)
accuracy_str = f"{acc*100:.2f}".replace(".", "_")
model_name_h5 = f"densenet121({accuracy_str}).h5"
model_name_keras = f"densenet121({accuracy_str}).keras"

model.save(os.path.join(SAVE_DIR_H5, model_name_h5))
model.save(os.path.join(SAVE_DIR_KERAS, model_name_keras))

print(f"Test accuracy: {acc*100:.2f}%")
print(f"Models saved as:\n{model_name_h5}\n{model_name_keras}")
```

```
39/39 ━━━━━━━━━━━━━━━━━━━━ 828s 22s/step - accuracy: 0.9843 - loss: 0.0563
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
Test accuracy: 99.19%
Models saved as:
densenet121(99_19).h5
densenet121(99_19).keras
```

10. Plot Training History

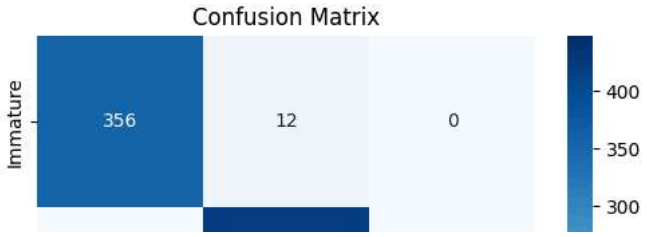
```
def plot_training(history_head, history_fine, model_name="DenseNet121"):
    acc = history_head.history['accuracy'] + history_fine.history['accuracy']
    val_acc = history_head.history['val_accuracy'] + history_fine.history['val_accuracy']
    loss = history_head.history['loss'] + history_fine.history['loss']
    val_loss = history_head.history['val_loss'] + history_fine.history['val_loss']
    epochs = range(1, len(acc) + 1)

    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(epochs, acc, label='Training Accuracy')
    plt.plot(epochs, val_acc, label='Validation Accuracy')
    plt.title(f'{model_name} Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(epochs, loss, label='Training Loss')
    plt.plot(epochs, val_loss, label='Validation Loss')
    plt.title(f'{model_name} Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()

plot_training(history_head, history_fine, model_name="DenseNet121")
```






```
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Get ground-truth labels and predictions for test set
test_steps = test_data.samples // test_data.batch_size + int(test_data.samples % test_data.batch_size != 0)
y_true = test_data.classes
y_pred_probs = model.predict(test_data, steps=test_steps, verbose=1)
y_pred = np.argmax(y_pred_probs, axis=1)

# Confusion matrix
cm = confusion_matrix(y_true, y_pred)
label_names = list(test_data.class_indices.keys())

plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=label_names, yticklabels=label_names)
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix')
plt.show()

# Print classification report for precision, recall, f1-score
print(classification_report(y_true, y_pred, target_names=label_names))
```

```
39/39 ━━━━━━━━━━━━━━━━━━━━ 25s 380ms/step
```



IMPORT

```
import os
import numpy as np
import tensorflow as tf
from sklearn.utils import class_weight
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
import matplotlib.pyplot as plt
from tensorflow.keras.applications import EfficientNetB3
from tensorflow.keras.applications.efficientnet import preprocess_input
```

MOUNT DRIVE

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

PATH

```
# Paths and parameters
DATA_DIR = "/content/drive/My Drive/PROJECT1/data/data_set_20_20_60"
IMG_SIZE = (224, 224)
BATCH_SIZE = 32
EPOCHS_HEAD = 15
EPOCHS_FINE = 25
CLASS_NAMES = ["Immature", "Mature", "Normal"]

SAVE_DIR_H5 = "/content/drive/My Drive/PROJECT1/trained models/H5"
SAVE_DIR_KERAS = "/content/drive/My Drive/PROJECT1/trained models/keras"
os.makedirs(SAVE_DIR_H5, exist_ok=True)
os.makedirs(SAVE_DIR_KERAS, exist_ok=True)
```

Data generators with augmentation

```
train_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    horizontal_flip=True,
    rotation_range=15,
    zoom_range=0.1,
    width_shift_range=0.1,
    height_shift_range=0.1
)
val_test_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)

train_data = train_datagen.flow_from_directory(
    os.path.join(DATA_DIR, "train"),
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical"
)

valid_data = val_test_datagen.flow_from_directory(
    os.path.join(DATA_DIR, "valid"),
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=False
```

```
)

test_data = val_test_datagen.flow_from_directory(
    os.path.join(DATA_DIR, "test"),
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=False
)
print("Class indices:", train_data.class_indices)
```

```
Found 3712 images belonging to 3 classes.
Found 1237 images belonging to 3 classes.
Found 1237 images belonging to 3 classes.
Class indices: {'Immature': 0, 'Mature': 1, 'Normal': 2}
```

Compute class weights to handle class imbalance

```
class_weights = class_weight.compute_class_weight(
    class_weight='balanced',
    classes=np.unique(train_data.classes),
    y=train_data.classes
)
class_weights = dict(enumerate(class_weights))
```

Create EfficientNet-B3 model with custom head

```
base_model = EfficientNetB3(weights='imagenet', include_top=False, input_shape=(*IMG_SIZE, 3))
base_model.trainable = False

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.3)(x)
predictions = Dense(len(CLASS_NAMES), activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)
```

Compile model for initial training

```
model.compile(optimizer=Adam(learning_rate=1e-3), loss="categorical_crossentropy", metrics=["accuracy"])

es = EarlyStopping(monitor="val_loss", patience=6, restore_best_weights=True)
rlr = ReduceLROnPlateau(monitor="val_loss", factor=0.2, patience=3, min_lr=1e-6)

print("Training EfficientNet-B3 classification head...")
history_head = model.fit(
    train_data,
    validation_data=valid_data,
    epochs=EPOCHS_HEAD,
    class_weight=class_weights,
    callbacks=[es, rlr]
)
```

```
Training EfficientNet-B3 classification head...
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset
  self._warn_if_super_not_called()
Epoch 1/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 1829s 15s/step - accuracy: 0.4718 - loss: 1.0295 - val_accuracy: 0.7615 - val_loss: 0.5957 - learni
Epoch 2/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 407ms/step - accuracy: 0.7532 - loss: 0.6235 - val_accuracy: 0.8003 - val_loss: 0.5092 - learni
Epoch 3/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 48s 416ms/step - accuracy: 0.8211 - loss: 0.5016 - val_accuracy: 0.8610 - val_loss: 0.4188 - learni
Epoch 4/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 46s 398ms/step - accuracy: 0.8289 - loss: 0.4561 - val_accuracy: 0.8480 - val_loss: 0.3994 - learni
Epoch 5/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 46s 400ms/step - accuracy: 0.8359 - loss: 0.4268 - val_accuracy: 0.8601 - val_loss: 0.3813 - learni
Epoch 6/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 405ms/step - accuracy: 0.8451 - loss: 0.4095 - val_accuracy: 0.8892 - val_loss: 0.3255 - learni
Epoch 7/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 48s 410ms/step - accuracy: 0.8660 - loss: 0.3551 - val_accuracy: 0.9030 - val_loss: 0.3056 - learni
Epoch 8/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 404ms/step - accuracy: 0.8682 - loss: 0.3541 - val_accuracy: 0.9078 - val_loss: 0.2800 - learni
```

```
Epoch 9/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 403ms/step - accuracy: 0.8853 - loss: 0.3209 - val_accuracy: 0.8698 - val_loss: 0.3194 - learni
Epoch 10/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 403ms/step - accuracy: 0.8645 - loss: 0.3391 - val_accuracy: 0.9175 - val_loss: 0.2601 - learni
Epoch 11/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 402ms/step - accuracy: 0.8909 - loss: 0.3164 - val_accuracy: 0.9175 - val_loss: 0.2545 - learni
Epoch 12/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 46s 400ms/step - accuracy: 0.8721 - loss: 0.3317 - val_accuracy: 0.9378 - val_loss: 0.2279 - learni
Epoch 13/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 46s 398ms/step - accuracy: 0.9020 - loss: 0.2900 - val_accuracy: 0.9232 - val_loss: 0.2375 - learni
Epoch 14/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 404ms/step - accuracy: 0.8895 - loss: 0.3001 - val_accuracy: 0.9111 - val_loss: 0.2471 - learni
Epoch 15/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 408ms/step - accuracy: 0.8902 - loss: 0.2960 - val_accuracy: 0.9240 - val_loss: 0.2215 - learni
```

Unfreeze last 50 layers (except BatchNorm) for fine-tuning

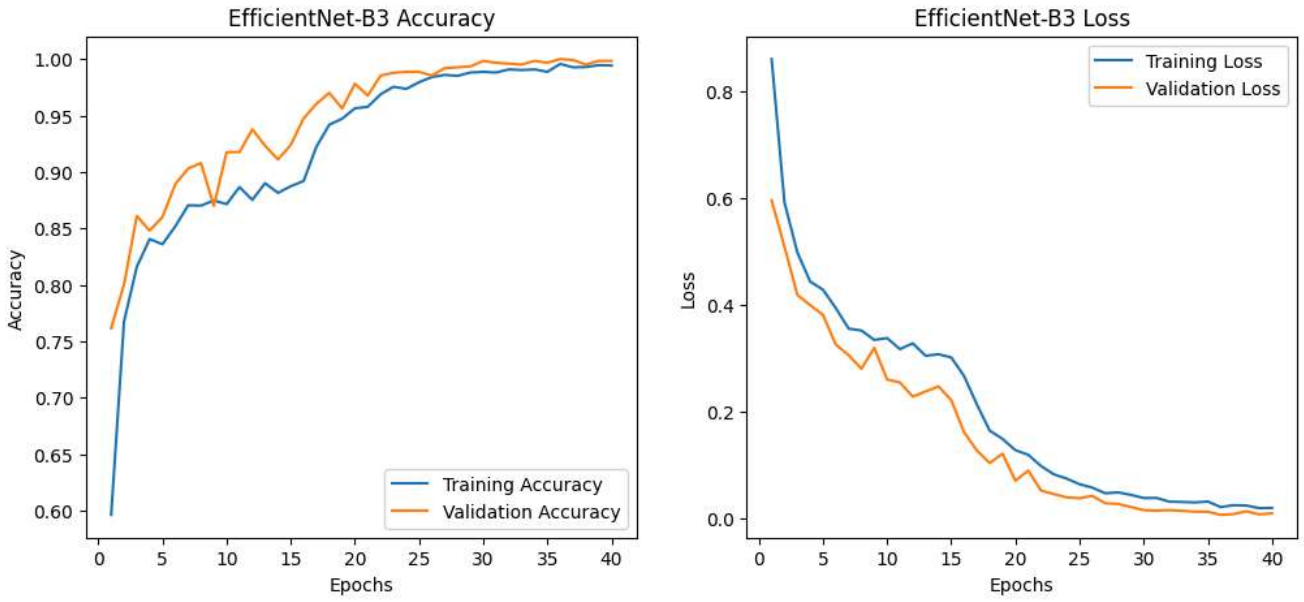
```
for layer in base_model.layers[-50:]:
    if not isinstance(layer, tf.keras.layers.BatchNormalization):
        layer.trainable = True

model.compile(optimizer=Adam(learning_rate=1e-5), loss="categorical_crossentropy", metrics=["accuracy"])

print("Fine-tuning EfficientNet-B3 last layers...")
history_fine = model.fit(
    train_data,
    validation_data=valid_data,
    epochs=EPOCHS_FINE,
    class_weight=class_weights,
    callbacks=[es, rlr]
)
```

```
Fine-tuning EfficientNet-B3 last layers...
Epoch 1/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 99s 538ms/step - accuracy: 0.8909 - loss: 0.2706 - val_accuracy: 0.9475 - val_loss: 0.1612 - learni
Epoch 2/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 407ms/step - accuracy: 0.9184 - loss: 0.2188 - val_accuracy: 0.9604 - val_loss: 0.1269 - learni
Epoch 3/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 403ms/step - accuracy: 0.9333 - loss: 0.1738 - val_accuracy: 0.9701 - val_loss: 0.1033 - learni
Epoch 4/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 408ms/step - accuracy: 0.9460 - loss: 0.1543 - val_accuracy: 0.9563 - val_loss: 0.1209 - learni
Epoch 5/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 400ms/step - accuracy: 0.9565 - loss: 0.1285 - val_accuracy: 0.9782 - val_loss: 0.0702 - learni
Epoch 6/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 46s 398ms/step - accuracy: 0.9597 - loss: 0.1209 - val_accuracy: 0.9677 - val_loss: 0.0893 - learni
Epoch 7/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 46s 400ms/step - accuracy: 0.9640 - loss: 0.1052 - val_accuracy: 0.9854 - val_loss: 0.0519 - learni
Epoch 8/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 401ms/step - accuracy: 0.9724 - loss: 0.0850 - val_accuracy: 0.9879 - val_loss: 0.0452 - learni
Epoch 9/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 46s 397ms/step - accuracy: 0.9691 - loss: 0.0782 - val_accuracy: 0.9887 - val_loss: 0.0391 - learni
Epoch 10/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 401ms/step - accuracy: 0.9800 - loss: 0.0654 - val_accuracy: 0.9887 - val_loss: 0.0374 - learni
Epoch 11/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 46s 399ms/step - accuracy: 0.9822 - loss: 0.0591 - val_accuracy: 0.9854 - val_loss: 0.0420 - learni
Epoch 12/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 401ms/step - accuracy: 0.9877 - loss: 0.0459 - val_accuracy: 0.9919 - val_loss: 0.0279 - learni
Epoch 13/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 405ms/step - accuracy: 0.9835 - loss: 0.0492 - val_accuracy: 0.9927 - val_loss: 0.0267 - learni
Epoch 14/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 403ms/step - accuracy: 0.9896 - loss: 0.0410 - val_accuracy: 0.9935 - val_loss: 0.0210 - learni
Epoch 15/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 403ms/step - accuracy: 0.9876 - loss: 0.0406 - val_accuracy: 0.9984 - val_loss: 0.0151 - learni
Epoch 16/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 46s 397ms/step - accuracy: 0.9899 - loss: 0.0364 - val_accuracy: 0.9968 - val_loss: 0.0142 - learni
Epoch 17/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 400ms/step - accuracy: 0.9926 - loss: 0.0278 - val_accuracy: 0.9960 - val_loss: 0.0151 - learni
Epoch 18/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 404ms/step - accuracy: 0.9920 - loss: 0.0294 - val_accuracy: 0.9951 - val_loss: 0.0137 - learni
Epoch 19/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 402ms/step - accuracy: 0.9895 - loss: 0.0301 - val_accuracy: 0.9984 - val_loss: 0.0123 - learni
Epoch 20/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 409ms/step - accuracy: 0.9909 - loss: 0.0256 - val_accuracy: 0.9968 - val_loss: 0.0121 - learni
Epoch 21/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 404ms/step - accuracy: 0.9948 - loss: 0.0241 - val_accuracy: 1.0000 - val_loss: 0.0063 - learni
Epoch 22/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 401ms/step - accuracy: 0.9955 - loss: 0.0219 - val_accuracy: 0.9992 - val_loss: 0.0078 - learni
Epoch 23/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 46s 397ms/step - accuracy: 0.9934 - loss: 0.0202 - val_accuracy: 0.9951 - val_loss: 0.0129 - learni
```

```
Epoch 24/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 402ms/step - accuracy: 0.9948 - loss: 0.0184 - val_accuracy: 0.9984 - val_loss: 0.0073 - learni
Epoch 25/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 47s 402ms/step - accuracy: 0.9946 - loss: 0.0172 - val_accuracy: 0.9984 - val_loss: 0.0091 - learni
```

Plot training and validation accuracy/loss graphs

```
def plot_training(history_head, history_fine, model_name="EfficientNet-B3"):
    acc = history_head.history['accuracy'] + history_fine.history['accuracy']
    val_acc = history_head.history['val_accuracy'] + history_fine.history['val_accuracy']
    loss = history_head.history['loss'] + history_fine.history['loss']
    val_loss = history_head.history['val_loss'] + history_fine.history['val_loss']

    epochs = range(1, len(acc) + 1)

    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    plt.plot(epochs, acc, label='Training Accuracy')
    plt.plot(epochs, val_acc, label='Validation Accuracy')
    plt.title(f'{model_name} Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(epochs, loss, label='Training Loss')
    plt.plot(epochs, val_loss, label='Validation Loss')
    plt.title(f'{model_name} Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    plt.show()

plot_training(history_head, history_fine, model_name="EfficientNet-B3")
```




```
print("Evaluating EfficientNet-B3 on test set...")
loss, acc = model.evaluate(test_data)
print(f"Test accuracy: {acc*100:.2f}%")

accuracy_str = f"{acc*100:.2f}".replace(".", "_")
model_name_h5 = f"efficientnetb3({accuracy_str}).h5"
model_name_keras = f"efficientnetb3({accuracy_str}).keras"

model.save(os.path.join(SAVE_DIR_H5, model_name_h5))
model.save(os.path.join(SAVE_DIR_KERAS, model_name_keras))
```

```
print(f"Models saved as:\n- {os.path.join(SAVE_DIR_H5, model_name_h5)}\n- {os.path.join(SAVE_DIR_KERAS, model_name_keras)}")
```

```
Evaluating EfficientNet-B3 on test set...
39/39 ━━━━━━━━━━━━━━━━━━━━ 758s 20s/step - accuracy: 0.9972 - loss: 0.0110
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
Test accuracy: 99.84%
Models saved as:
- /content/drive/My Drive/PROJECT1/trained models/H5/efficientnetb3(99_84).h5
- /content/drive/My Drive/PROJECT1/trained models/keras/efficientnetb3(99_84).keras
```

IMPORTS

```
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.utils import class_weight
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras.preprocessing import image
```

MOUNT DRIVE

```
from google.colab import drive
drive.mount("/content/drive")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

CONFIGURATION

```
DATA_DIR = "/content/drive/My Drive/PROJECT1/data/data_set_20_20_60"
IMG_SIZE = (224, 224)
BATCH_SIZE = 32
EPOCHS = 40
CLASS_NAMES = ["Immature", "Mature", "Normal"]
```

DATA GENERATORS

```
train_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    horizontal_flip=True
)

val_test_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)

train_data = train_datagen.flow_from_directory(
    os.path.join(DATA_DIR, "train"),
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical"
)
valid_data = val_test_datagen.flow_from_directory(
    os.path.join(DATA_DIR, "valid"),
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=False
)
test_data = val_test_datagen.flow_from_directory(
    os.path.join(DATA_DIR, "test"),
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode="categorical",
    shuffle=False
)
```

```
print("Class indices:", train_data.class_indices)
```

```
Found 3712 images belonging to 3 classes.
Found 1237 images belonging to 3 classes.
Found 1237 images belonging to 3 classes.
Class indices: {'Immature': 0, 'Mature': 1, 'Normal': 2}
```

COMPUTE CLASS WEIGHTS

```
class_weights = class_weight.compute_class_weight(
    class_weight="balanced",
    classes=np.unique(train_data.classes),
    y=train_data.classes
)
class_weights = dict(enumerate(class_weights))
print("Class Weights:", class_weights)
```

```
Class Weights: {0: np.float64(1.0998518518518519), 1: np.float64(0.9644063393089114), 2: np.float64(0.9488752556237219)}
```

BUILD MODEL (ResNet50)

```
base_model = ResNet50(weights="imagenet", include_top=False, input_shape=IMG_SIZE + (3,))
base_model.trainable = False  # freeze first

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.3)(x)
preds = Dense(len(CLASS_NAMES), activation="softmax")(x)

model = Model(inputs=base_model.input, outputs=preds)
```

TRAINING

```
model.compile(optimizer=Adam(1e-3), loss="categorical_crossentropy", metrics=["accuracy"])

es = EarlyStopping(monitor="val_loss", patience=6, restore_best_weights=True)
rlr = ReduceLROnPlateau(monitor="val_loss", factor=0.2, patience=3, min_lr=1e-6)

print("\n=== Stage 1: Training top layers ===")
history1 = model.fit(
    train_data,
    validation_data=valid_data,
    epochs=15,
    class_weight=class_weights,
    callbacks=[es, rlr]
)
```

```
=== Stage 1: Training top layers ===
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset
  self._warn_if_super_not_called()
Epoch 1/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 1696s 15s/step - accuracy: 0.5888 - loss: 0.9605 - val_accuracy: 0.8610 - val_loss: 0.3412 - learni
Epoch 2/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 16s 140ms/step - accuracy: 0.8390 - loss: 0.3681 - val_accuracy: 0.9224 - val_loss: 0.2176 - learni
Epoch 3/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 16s 136ms/step - accuracy: 0.9084 - loss: 0.2388 - val_accuracy: 0.9426 - val_loss: 0.1631 - learni
Epoch 4/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 16s 137ms/step - accuracy: 0.9183 - loss: 0.2142 - val_accuracy: 0.9507 - val_loss: 0.1428 - learni
Epoch 5/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 16s 138ms/step - accuracy: 0.9304 - loss: 0.1785 - val_accuracy: 0.9693 - val_loss: 0.1099 - learni
Epoch 6/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 17s 144ms/step - accuracy: 0.9456 - loss: 0.1574 - val_accuracy: 0.9806 - val_loss: 0.0905 - learni
Epoch 7/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 16s 138ms/step - accuracy: 0.9548 - loss: 0.1300 - val_accuracy: 0.9895 - val_loss: 0.0696 - learni
Epoch 8/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 15s 132ms/step - accuracy: 0.9716 - loss: 0.1021 - val_accuracy: 0.9919 - val_loss: 0.0627 - learni
Epoch 9/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 16s 136ms/step - accuracy: 0.9712 - loss: 0.0991 - val_accuracy: 0.9879 - val_loss: 0.0582 - learni
```

```
Epoch 10/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 16s 135ms/step - accuracy: 0.9713 - loss: 0.0931 - val_accuracy: 0.9766 - val_loss: 0.0758 - learni
Epoch 11/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 16s 135ms/step - accuracy: 0.9709 - loss: 0.0920 - val_accuracy: 0.9757 - val_loss: 0.0740 - learni
Epoch 12/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 16s 133ms/step - accuracy: 0.9633 - loss: 0.0981 - val_accuracy: 0.9887 - val_loss: 0.0597 - learni
Epoch 13/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 16s 133ms/step - accuracy: 0.9759 - loss: 0.0823 - val_accuracy: 0.9895 - val_loss: 0.0444 - learni
Epoch 14/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 16s 139ms/step - accuracy: 0.9738 - loss: 0.0737 - val_accuracy: 0.9919 - val_loss: 0.0473 - learni
Epoch 15/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 16s 137ms/step - accuracy: 0.9779 - loss: 0.0751 - val_accuracy: 0.9911 - val_loss: 0.0440 - learni
```

Fine-tune deeper layers

```
for layer in base_model.layers[-50:]:
    if not isinstance(layer, tf.keras.layers.BatchNormalization):
        layer.trainable = True

model.compile(optimizer=Adam(1e-5), loss="categorical_crossentropy", metrics=["accuracy"])

print("\n=== Stage 2: Fine-tuning ===")
history2 = model.fit(
    train_data,
    validation_data=valid_data,
    epochs=EPOCHS,
    class_weight=class_weights,
    callbacks=[es, rlr]
)
```

```
=== Stage 2: Fine-tuning ===
Epoch 1/40
116/116 ━━━━━━━━━━━━━━━━━━━━ 42s 217ms/step - accuracy: 0.9770 - loss: 0.0657 - val_accuracy: 0.9984 - val_loss: 0.0127 - learni
Epoch 2/40
116/116 ━━━━━━━━━━━━━━━━━━━━ 20s 168ms/step - accuracy: 0.9940 - loss: 0.0190 - val_accuracy: 0.9992 - val_loss: 0.0059 - learni
Epoch 3/40
116/116 ━━━━━━━━━━━━━━━━━━━━ 19s 164ms/step - accuracy: 0.9978 - loss: 0.0071 - val_accuracy: 1.0000 - val_loss: 0.0016 - learni
Epoch 4/40
116/116 ━━━━━━━━━━━━━━━━━━━━ 19s 166ms/step - accuracy: 0.9990 - loss: 0.0043 - val_accuracy: 1.0000 - val_loss: 0.0013 - learni
Epoch 5/40
116/116 ━━━━━━━━━━━━━━━━━━━━ 19s 164ms/step - accuracy: 0.9983 - loss: 0.0039 - val_accuracy: 0.9992 - val_loss: 0.0016 - learni
Epoch 6/40
116/116 ━━━━━━━━━━━━━━━━━━━━ 19s 164ms/step - accuracy: 0.9998 - loss: 0.0018 - val_accuracy: 0.9992 - val_loss: 0.0027 - learni
Epoch 7/40
116/116 ━━━━━━━━━━━━━━━━━━━━ 19s 167ms/step - accuracy: 0.9996 - loss: 0.0016 - val_accuracy: 1.0000 - val_loss: 0.0011 - learni
Epoch 8/40
116/116 ━━━━━━━━━━━━━━━━━━━━ 19s 166ms/step - accuracy: 0.9981 - loss: 0.0064 - val_accuracy: 1.0000 - val_loss: 5.8520e-04 - le
Epoch 9/40
116/116 ━━━━━━━━━━━━━━━━━━━━ 19s 166ms/step - accuracy: 1.0000 - loss: 4.3453e-04 - val_accuracy: 1.0000 - val_loss: 4.4524e-04
Epoch 10/40
116/116 ━━━━━━━━━━━━━━━━━━━━ 19s 164ms/step - accuracy: 1.0000 - loss: 3.8521e-04 - val_accuracy: 1.0000 - val_loss: 3.0113e-04
Epoch 11/40
116/116 ━━━━━━━━━━━━━━━━━━━━ 20s 168ms/step - accuracy: 1.0000 - loss: 1.7812e-04 - val_accuracy: 1.0000 - val_loss: 2.1980e-04
Epoch 12/40
116/116 ━━━━━━━━━━━━━━━━━━━━ 19s 167ms/step - accuracy: 1.0000 - loss: 8.5134e-05 - val_accuracy: 1.0000 - val_loss: 2.1052e-04
Epoch 13/40
116/116 ━━━━━━━━━━━━━━━━━━━━ 19s 167ms/step - accuracy: 1.0000 - loss: 1.2951e-04 - val_accuracy: 1.0000 - val_loss: 1.3938e-04
Epoch 14/40
116/116 ━━━━━━━━━━━━━━━━━━━━ 19s 167ms/step - accuracy: 1.0000 - loss: 8.8093e-05 - val_accuracy: 1.0000 - val_loss: 1.6783e-04
Epoch 15/40
116/116 ━━━━━━━━━━━━━━━━━━━━ 19s 167ms/step - accuracy: 1.0000 - loss: 2.2866e-04 - val_accuracy: 1.0000 - val_loss: 1.1104e-04
Epoch 16/40
116/116 ━━━━━━━━━━━━━━━━━━━━ 19s 167ms/step - accuracy: 1.0000 - loss: 3.6013e-05 - val_accuracy: 1.0000 - val_loss: 9.9985e-05
Epoch 17/40
116/116 ━━━━━━━━━━━━━━━━━━━━ 19s 165ms/step - accuracy: 1.0000 - loss: 3.1717e-05 - val_accuracy: 1.0000 - val_loss: 9.1776e-05
Epoch 18/40
116/116 ━━━━━━━━━━━━━━━━━━━━ 19s 164ms/step - accuracy: 1.0000 - loss: 5.8651e-05 - val_accuracy: 1.0000 - val_loss: 8.7151e-05
Epoch 19/40
116/116 ━━━━━━━━━━━━━━━━━━━━ 19s 164ms/step - accuracy: 1.0000 - loss: 6.5629e-05 - val_accuracy: 1.0000 - val_loss: 3.9000e-04
Epoch 20/40
116/116 ━━━━━━━━━━━━━━━━━━━━ 19s 165ms/step - accuracy: 1.0000 - loss: 6.9724e-05 - val_accuracy: 1.0000 - val_loss: 3.1208e-04
Epoch 21/40
116/116 ━━━━━━━━━━━━━━━━━━━━ 19s 165ms/step - accuracy: 1.0000 - loss: 6.5551e-05 - val_accuracy: 1.0000 - val_loss: 2.3299e-04
Epoch 22/40
116/116 ━━━━━━━━━━━━━━━━━━━━ 19s 163ms/step - accuracy: 1.0000 - loss: 4.1113e-05 - val_accuracy: 1.0000 - val_loss: 1.5085e-04
Epoch 23/40
116/116 ━━━━━━━━━━━━━━━━━━━━ 19s 164ms/step - accuracy: 1.0000 - loss: 4.4856e-05 - val_accuracy: 1.0000 - val_loss: 1.5439e-04
Epoch 24/40
```

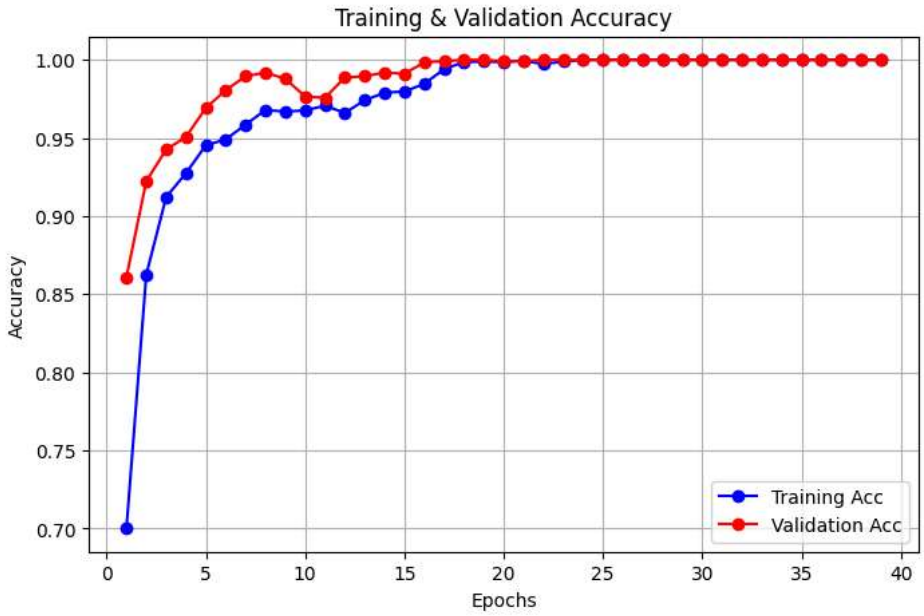
```
116/116 ━━━━━━━━━━━━━━━━━━━━ 19s 161ms/step - accuracy: 1.0000 - loss: 6.8415e-05 - val_accuracy: 1.0000 - val_loss: 1.7100e-04
```

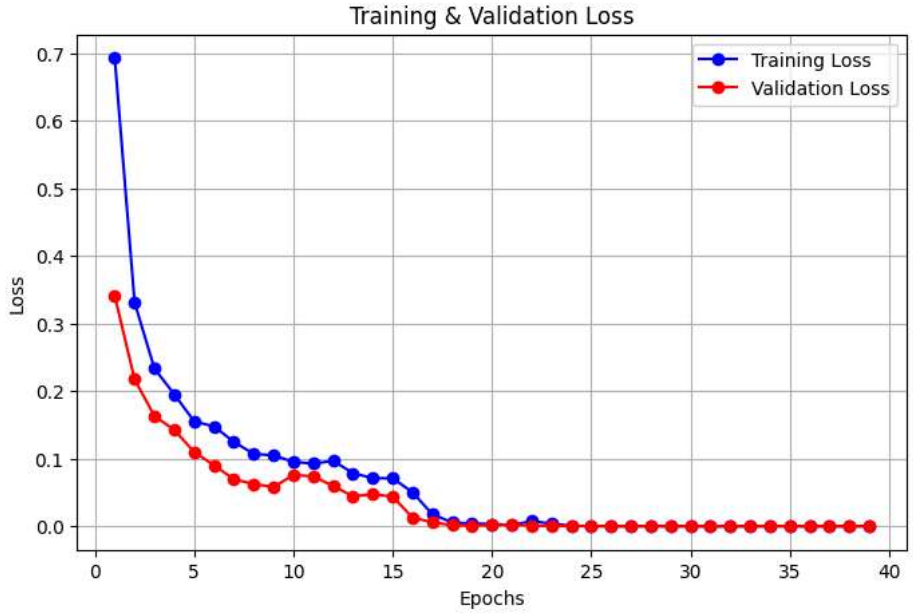
MERGE HISTORIES

```
def combine_histories(h1, h2):
    history = {}
    for k in h1.history.keys():
        history[k] = h1.history[k] + h2.history[k]
    return history

full_history = combine_histories(history1, history2)
```

PLOT TRAINING GRAPHS

```
def plot_training(history):
    acc = history["accuracy"]
    val_acc = history["val_accuracy"]
    loss = history["loss"]
    val_loss = history["val_loss"]
    epochs = range(1, len(acc) + 1)

    plt.figure(figsize=(8,5))
    plt.plot(epochs, acc, "bo-", label="Training Acc")
    plt.plot(epochs, val_acc, "ro-", label="Validation Acc")
    plt.title("Training & Validation Accuracy")
    plt.xlabel("Epochs"); plt.ylabel("Accuracy"); plt.legend(); plt.grid(True)
    plt.show()

    plt.figure(figsize=(8,5))
    plt.plot(epochs, loss, "bo-", label="Training Loss")
    plt.plot(epochs, val_loss, "ro-", label="Validation Loss")
    plt.title("Training & Validation Loss")
    plt.xlabel("Epochs"); plt.ylabel("Loss"); plt.legend(); plt.grid(True)
    plt.show()

plot_training(full_history)
```





EVALUATE ON TEST SET

```
print("\n=== Evaluating on test set ===")
loss, acc = model.evaluate(test_data)
print(f"Test Accuracy: {acc:.2%}")
```

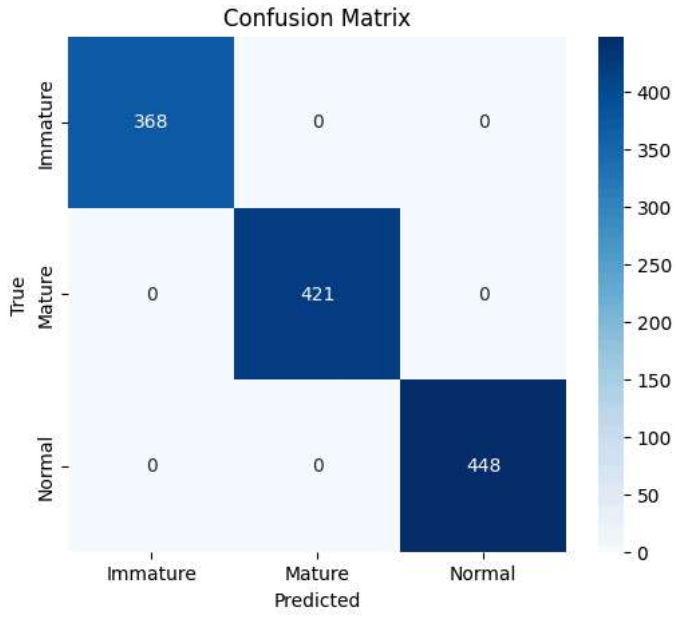
```
=== Evaluating on test set ===
39/39 ━━━━━━━━━━━━━━━━━━━━ 919s 24s/step - accuracy: 1.0000 - loss: 1.0607e-04
Test Accuracy: 100.00%
```

CONFUSION MATRIX & REPORT

```
y_true = test_data.classes
y_pred = np.argmax(model.predict(test_data), axis=1)

print("\nClassification Report:")
print(classification_report(y_true, y_pred, target_names=CLASS_NAMES))

cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(6,5))
```

```
sns.heatmap(cm, annot=True, fmt="d", xticklabels=CLASS_NAMES, yticklabels=CLASS_NAMES, cmap="Blues")
plt.xlabel("Predicted"); plt.ylabel("True"); plt.title("Confusion Matrix")
plt.show()
```

```
39/39 ━━━━━━━━━━━━━━━━━━━━ 11s 189ms/step

Classification Report:
              precision    recall  f1-score   support

    Immature       1.00      1.00      1.00       368
      Mature       1.00      1.00      1.00       421
      Normal       1.00      1.00      1.00       448

    accuracy                           1.00      1237
   macro avg       1.00      1.00      1.00      1237
weighted avg       1.00      1.00      1.00      1237
```



SAVE MODEL WITH ACCURACY IN NAME

```
accuracy = acc * 100
model_name_h5 = f"1602020resnet_{accuracy:.2f}.h5"
model_name_keras = f"1602020resnet_{accuracy:.2f}.keras"

save_dir = "/content/drive/MyDrive/PROJECT1/trained models/"
os.makedirs(save_dir, exist_ok=True)

model.save(os.path.join(save_dir, model_name_h5))
model.save(os.path.join(save_dir, model_name_keras))

print(f"Models saved as: {model_name_h5}, {model_name_keras}")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
Models saved as: 602020resnet_100.00.h5, 602020resnet_100.00.keras
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
Models saved as: 1602020resnet_100.00.h5, 1602020resnet_100.00.keras
```

PREDICTION FUNCTION

```
def predict_image(model_path, img_path):
    model = tf.keras.models.load_model(model_path, compile=False)
    img = image.load_img(img_path, target_size=IMG_SIZE)
    x = image.img_to_array(img)
    x = preprocess_input(x)
    x = np.expand_dims(x, axis=0)
    preds = model.predict(x)[0]
    idx = np.argmax(preds)
    return CLASS_NAMES[idx], float(preds[idx]), preds
```

```
label, conf, probs = predict_image(os.path.join(save_dir, model_name_h5), "/content/drive/My Drive/PROJECT1/data/data_set_20_20
print(f"Predicted: {label} (confidence {conf:.2%})")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
Predicted: Mature (confidence 100.00%)
```

```
label, conf, probs = predict_image(os.path.join(save_dir, model_name_h5), "/content/drive/My Drive/PROJECT1/data/data_set_20_20
print(f"Predicted: {label} (confidence {conf:.2%})")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
Predicted: Immature (confidence 100.00%)
```

```
label, conf, probs = predict_image(os.path.join(save_dir, model_name_h5), "/content/drive/My Drive/PROJECT1/data/data_set_20_20
print(f"Predicted: {label} (confidence {conf:.2%})")
```

```
WARNING:tensorflow:5 out of the last 42 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_dis
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
Predicted: Normal (confidence 100.00%)
```

```
print(model_name_h5)
```

IMPORT

```
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.utils import class_weight
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG19
from tensorflow.keras.applications.vgg19 import preprocess_input
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
```

MOUNT DRIVE

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

PATH

```
# Paths and parameters
DATA_DIR = "/content/drive/My Drive/PROJECT1/data/data_set_20_20_60"
IMG_SIZE = (224, 224)
BATCH_SIZE = 32
EPOCHS_HEAD = 15   # epochs for initial head training
EPOCHS_FINE = 25   # epochs for fine-tuning
CLASS_NAMES = ["Immature", "Mature", "Normal"]
SAVE_DIR = "/content/drive/MyDrive/PROJECT1/trained models/"
os.makedirs(SAVE_DIR, exist_ok=True)
```

DATA GENERATORS

```
train_datagen = ImageDataGenerator(preprocessing_function=preprocess_input,
                                   horizontal_flip=True,
                                   rotation_range=15,
                                   zoom_range=0.1,
                                   width_shift_range=0.1,
                                   height_shift_range=0.1)
```

```
val_test_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)
```

```
train_data = train_datagen.flow_from_directory(
    os.path.join(DATA_DIR, "train"),
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical'
)

valid_data = val_test_datagen.flow_from_directory(
    os.path.join(DATA_DIR, "valid"),
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=False
)
```

```
test_data = val_test_datagen.flow_from_directory(
    os.path.join(DATA_DIR, "test"),
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=False
)

print("Class indices:", train_data.class_indices)
```

```
Found 3712 images belonging to 3 classes.
Found 1237 images belonging to 3 classes.
Found 1237 images belonging to 3 classes.
Class indices: {'Immature': 0, 'Mature': 1, 'Normal': 2}
```

COMPUTE CLASS WEIGHTS

```
class_weights = class_weight.compute_class_weight(
    class_weight='balanced',
    classes=np.unique(train_data.classes),
    y=train_data.classes
)
class_weights = dict(enumerate(class_weights))
print("Class weights:", class_weights)
```

```
Class weights: {0: np.float64(1.0998518518518519), 1: np.float64(0.9644063393089114), 2: np.float64(0.9488752556237219)}
```

LOADING MODEL WITHOUT TOP LAYERS

```
# Load VGG19 base model without top layers
base_model = VGG19(weights='imagenet', include_top=False, input_shape=(*IMG_SIZE, 3))
base_model.trainable = False  # Freeze all layers initially
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernel
80134624/80134624 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

ADD CLASSIFICATION HEADEER

```
x = base_model.output
x = Flatten()(x)
x = Dropout(0.3)(x)
predictions = Dense(len(CLASS_NAMES), activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)
```

COMPILE FOR HEAD TRAINING

```
model.compile(optimizer=Adam(learning_rate=1e-3),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

CALLBACKS FOR TRAINING

```
es = EarlyStopping(monitor='val_loss', patience=6, restore_best_weights=True)
rlr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3, min_lr=1e-6)
checkpoint_path = os.path.join(SAVE_DIR, "vgg19_best_model.h5")
mc = ModelCheckpoint(checkpoint_path, monitor='val_loss', save_best_only=True, verbose=1)
```

Stage 1: Training classification head

```
history1 = model.fit(
    train_data,
    validation_data=valid_data,
    epochs=EPOCHS_HEAD,
    class_weight=class_weights,
    callbacks=[es, rlr, mc]
)
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset
  self._warn_if_super_not_called()
Epoch 1/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 0s 6s/step - accuracy: 0.6214 - loss: 11.4494
Epoch 1: val_loss improved from inf to 1.19653, saving model to /content/drive/MyDrive/PROJECT1/trained models/vgg19_best_mode
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
116/116 ━━━━━━━━━━━━━━━━━━━━ 993s 8s/step - accuracy: 0.6225 - loss: 11.4061 - val_accuracy: 0.9345 - val_loss: 1.1965 - learn
Epoch 2/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 0s 368ms/step - accuracy: 0.8961 - loss: 2.0194
Epoch 2: val_loss did not improve from 1.19653
116/116 ━━━━━━━━━━━━━━━━━━━━ 52s 444ms/step - accuracy: 0.8961 - loss: 2.0201 - val_accuracy: 0.9402 - val_loss: 1.2794 - learn
Epoch 3/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 0s 364ms/step - accuracy: 0.9256 - loss: 1.8885
Epoch 3: val_loss improved from 1.19653 to 0.24909, saving model to /content/drive/MyDrive/PROJECT1/trained models/vgg19_best_r
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
116/116 ━━━━━━━━━━━━━━━━━━━━ 53s 457ms/step - accuracy: 0.9256 - loss: 1.8874 - val_accuracy: 0.9895 - val_loss: 0.2491 - learn
Epoch 4/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 0s 377ms/step - accuracy: 0.9468 - loss: 1.2358
Epoch 4: val_loss improved from 0.24909 to 0.12830, saving model to /content/drive/MyDrive/PROJECT1/trained models/vgg19_best_r
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
116/116 ━━━━━━━━━━━━━━━━━━━━ 54s 465ms/step - accuracy: 0.9468 - loss: 1.2368 - val_accuracy: 0.9919 - val_loss: 0.1283 - learn
Epoch 5/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 0s 376ms/step - accuracy: 0.9504 - loss: 1.3392
Epoch 5: val_loss did not improve from 0.12830
116/116 ━━━━━━━━━━━━━━━━━━━━ 53s 452ms/step - accuracy: 0.9504 - loss: 1.3387 - val_accuracy: 0.9863 - val_loss: 0.1797 - learn
Epoch 6/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 0s 368ms/step - accuracy: 0.9484 - loss: 1.7455
Epoch 6: val_loss did not improve from 0.12830
116/116 ━━━━━━━━━━━━━━━━━━━━ 52s 444ms/step - accuracy: 0.9484 - loss: 1.7456 - val_accuracy: 0.9887 - val_loss: 0.1397 - learn
Epoch 7/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 0s 369ms/step - accuracy: 0.9433 - loss: 1.9982
Epoch 7: val_loss improved from 0.12830 to 0.12438, saving model to /content/drive/MyDrive/PROJECT1/trained models/vgg19_best_r
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
116/116 ━━━━━━━━━━━━━━━━━━━━ 53s 459ms/step - accuracy: 0.9434 - loss: 1.9940 - val_accuracy: 0.9943 - val_loss: 0.1244 - learn
Epoch 8/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 0s 368ms/step - accuracy: 0.9735 - loss: 0.9249
Epoch 8: val_loss improved from 0.12438 to 0.08358, saving model to /content/drive/MyDrive/PROJECT1/trained models/vgg19_best_r
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
116/116 ━━━━━━━━━━━━━━━━━━━━ 53s 458ms/step - accuracy: 0.9735 - loss: 0.9261 - val_accuracy: 0.9935 - val_loss: 0.0836 - learn
Epoch 9/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 0s 377ms/step - accuracy: 0.9707 - loss: 0.7558
Epoch 9: val_loss improved from 0.08358 to 0.03231, saving model to /content/drive/MyDrive/PROJECT1/trained models/vgg19_best_r
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
116/116 ━━━━━━━━━━━━━━━━━━━━ 54s 467ms/step - accuracy: 0.9707 - loss: 0.7572 - val_accuracy: 0.9984 - val_loss: 0.0323 - learn
Epoch 10/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 0s 379ms/step - accuracy: 0.9698 - loss: 1.1214
Epoch 10: val_loss did not improve from 0.03231
116/116 ━━━━━━━━━━━━━━━━━━━━ 53s 455ms/step - accuracy: 0.9698 - loss: 1.1214 - val_accuracy: 0.9968 - val_loss: 0.0656 - learn
Epoch 11/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 0s 368ms/step - accuracy: 0.9659 - loss: 1.2381
Epoch 11: val_loss did not improve from 0.03231
116/116 ━━━━━━━━━━━━━━━━━━━━ 52s 444ms/step - accuracy: 0.9659 - loss: 1.2390 - val_accuracy: 0.9903 - val_loss: 0.2407 - learn
Epoch 12/15
116/116 ━━━━━━━━━━━━━━━━━━━━ 0s 372ms/step - accuracy: 0.9781 - loss: 0.8754
Epoch 12: val_loss did not improve from 0.03231
116/116 ━━━━━━━━━━━━━━━━━━━━ 52s 448ms/step - accuracy: 0.9780 - loss: 0.8761 - val_accuracy: 0.9968 - val_loss: 0.1130 - learn
Epoch 13/15
```

Unfreeze last convolutional block of base_model for fine-tuning

```
for layer in base_model.layers:
    layer.trainable = layer.name.startswith("block5")
```

Recompile with lower learning rate for fine-tuning

```
model.compile(optimizer=Adam(learning_rate=1e-5),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Stage 2: Fine-tuning last layers

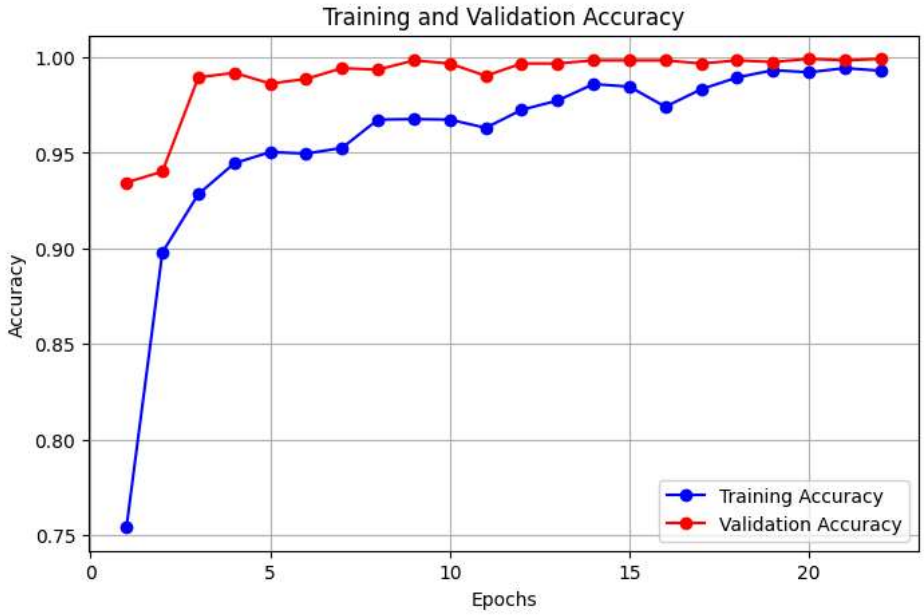
```
history2 = model.fit(
    train_data,
    validation_data=valid_data,
    epochs=EPOCHS_FINE,
    class_weight=class_weights,
    callbacks=[es, rlr, mc]
)
```

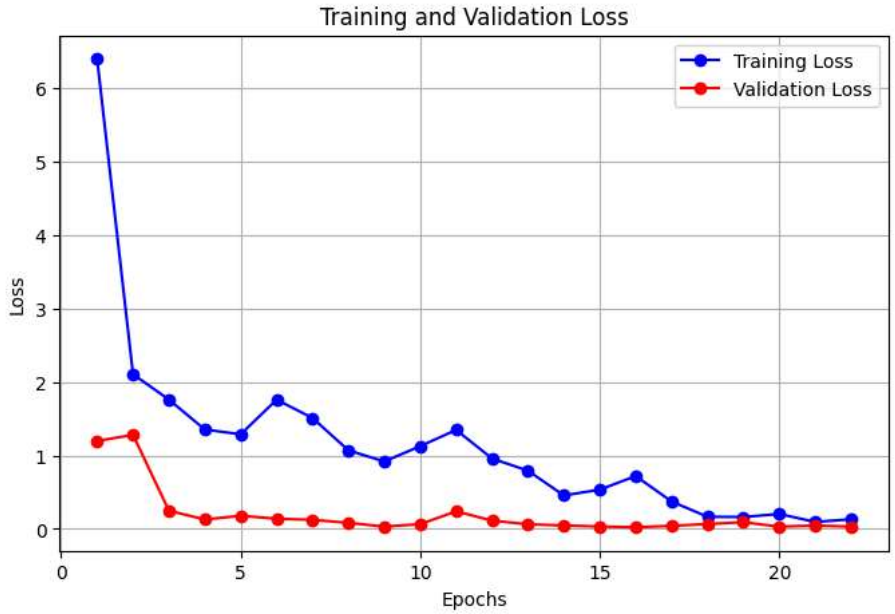
```
Epoch 1/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 0s 373ms/step - accuracy: 0.9710 - loss: 0.8243
Epoch 1: val_loss improved from 0.03231 to 0.02317, saving model to /content/drive/MyDrive/PROJECT1/trained models/vgg19_best_mo
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
116/116 ━━━━━━━━━━━━━━━━━━━━ 62s 477ms/step - accuracy: 0.9710 - loss: 0.8235 - val_accuracy: 0.9984 - val_loss: 0.0232 - learni
Epoch 2/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 0s 412ms/step - accuracy: 0.9876 - loss: 0.2876
Epoch 2: val_loss did not improve from 0.02317
116/116 ━━━━━━━━━━━━━━━━━━━━ 57s 488ms/step - accuracy: 0.9876 - loss: 0.2884 - val_accuracy: 0.9968 - val_loss: 0.0425 - learni
Epoch 3/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 0s 372ms/step - accuracy: 0.9885 - loss: 0.1756
Epoch 3: val_loss did not improve from 0.02317
116/116 ━━━━━━━━━━━━━━━━━━━━ 52s 448ms/step - accuracy: 0.9885 - loss: 0.1755 - val_accuracy: 0.9984 - val_loss: 0.0670 - learni
Epoch 4/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 0s 369ms/step - accuracy: 0.9935 - loss: 0.1334
Epoch 4: val_loss did not improve from 0.02317
116/116 ━━━━━━━━━━━━━━━━━━━━ 52s 445ms/step - accuracy: 0.9935 - loss: 0.1337 - val_accuracy: 0.9976 - val_loss: 0.0924 - learni
Epoch 5/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 0s 373ms/step - accuracy: 0.9897 - loss: 0.3050
Epoch 5: val_loss did not improve from 0.02317
116/116 ━━━━━━━━━━━━━━━━━━━━ 52s 449ms/step - accuracy: 0.9897 - loss: 0.3041 - val_accuracy: 0.9992 - val_loss: 0.0316 - learni
Epoch 6/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 0s 370ms/step - accuracy: 0.9930 - loss: 0.1151
Epoch 6: val_loss did not improve from 0.02317
116/116 ━━━━━━━━━━━━━━━━━━━━ 52s 446ms/step - accuracy: 0.9931 - loss: 0.1149 - val_accuracy: 0.9984 - val_loss: 0.0463 - learni
Epoch 7/25
116/116 ━━━━━━━━━━━━━━━━━━━━ 0s 371ms/step - accuracy: 0.9930 - loss: 0.1101
Epoch 7: val_loss did not improve from 0.02317
116/116 ━━━━━━━━━━━━━━━━━━━━ 52s 447ms/step - accuracy: 0.9930 - loss: 0.1103 - val_accuracy: 0.9992 - val_loss: 0.0321 - learni
```

```
# Combine histories
def combine_histories(h1, h2):
    history = {}
    for k in h1.history.keys():
        history[k] = h1.history[k] + h2.history[k]
    return history

full_history = combine_histories(history1, history2)
```

```
# Plot training and validation accuracy/loss
def plot_training_history(history):
    acc = history['accuracy']
    val_acc = history['val_accuracy']
    loss = history['loss']
    val_loss = history['val_loss']
    epochs_range = range(1, len(acc) + 1)

    plt.figure(figsize=(8, 5))
    plt.plot(epochs_range, acc, 'bo-', label='Training Accuracy')
    plt.plot(epochs_range, val_acc, 'ro-', label='Validation Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
```

```
    plt.grid(True)
    plt.show()

    plt.figure(figsize=(8, 5))
    plt.plot(epochs_range, loss, 'bo-', label='Training Loss')
    plt.plot(epochs_range, val_loss, 'ro-', label='Validation Loss')
    plt.title('Training and Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)
    plt.show()

plot_training_history(full_history)
```



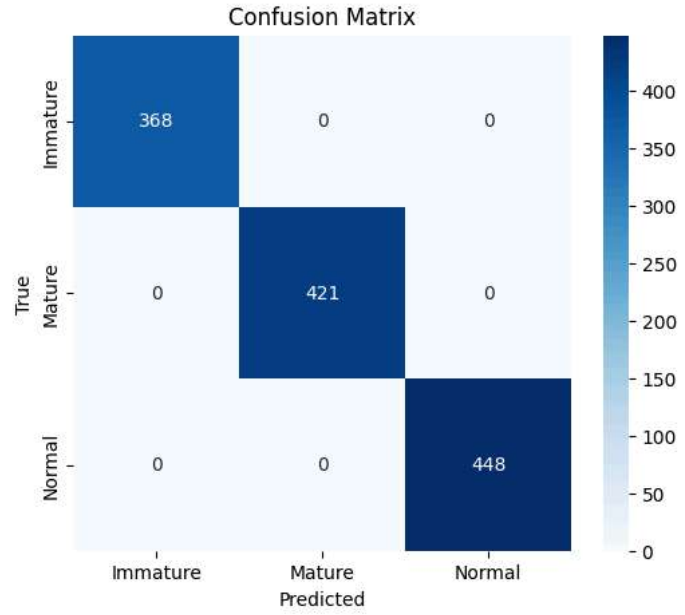



```
# Final evaluation
print("Evaluating on test set")
test_loss, test_acc = model.evaluate(test_data)
print(f"Test Accuracy: {test_acc:.2f}")
```

```
Evaluating on test set
39/39 ━━━━━━━━━━━━━━━━━━━━ 308s 8s/step - accuracy: 1.0000 - loss: 1.1484e-06
Test Accuracy: 1.00
```

```
# Classification report and confusion matrix
y_true = test_data.classes
y_pred = np.argmax(model.predict(test_data), axis=1)

print("Classification Report:")
print(classification_report(y_true, y_pred, target_names=CLASS_NAMES))

cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', xticklabels=CLASS_NAMES, yticklabels=CLASS_NAMES, cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

```
39/39 ━━━━━━━━━━━━━━━━━━━━ 9s 224ms/step
Classification Report:
              precision    recall  f1-score   support

    Immature       1.00      1.00      1.00       368
      Mature       1.00      1.00      1.00       421
      Normal       1.00      1.00      1.00       448

    accuracy                           1.00      1237
   macro avg       1.00      1.00      1.00      1237
weighted avg       1.00      1.00      1.00      1237
```




```
accuracy = test_acc * 100
accuracy_str = f"{accuracy:.2f}".replace('.', '_')
model_name_h5 = f"vgg19_{accuracy_str}.h5"
model_name_keras = f"vgg19_{accuracy_str}.keras"

model.save(os.path.join(SAVE_DIR, model_name_h5))
model.save(os.path.join(SAVE_DIR, model_name_keras))

print(f"Models saved as: {model_name_h5}, {model_name_keras}")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
Models saved as: vgg19_100_00.h5, vgg19_100_00.keras
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
!pip install -q streamlit pyngrok
```

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.2/10.2 MB 74.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.9/6.9 MB 67.9 MB/s eta 0:00:00
```

```
%%writefile app.py
import streamlit as st
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.densenet import preprocess_input as preprocess_densenet
from tensorflow.keras.applications.efficientnet import preprocess_input as preprocess_efficientnet
from tensorflow.keras.applications.resnet50 import preprocess_input as preprocess_resnet
from tensorflow.keras.applications.vgg19 import preprocess_input as preprocess_vgg19
from PIL import Image
import numpy as np

st.set_page_config(
    page_title="Cataract Detection Ensemble",
    page_icon="👁",
    layout="centered"
)

# Your actual Google Drive model paths:
model_info = {
    'densenet121': {'path': '/content/drive/MyDrive/PROJECT/PROJECT-1/trained models/H5 files/densenet121(99_03).h5',       'pre
    'efficientnetb3': {'path': '/content/drive/MyDrive/PROJECT/PROJECT-1/trained models/H5 files/efficientnetb3(99_84).h5', 'pre
    'resnet': {'path': '/content/drive/MyDrive/PROJECT/PROJECT-1/trained models/H5 files/resnet(99_91).h5',                 'pre
    'vgg19': {'path': '/content/drive/MyDrive/PROJECT/PROJECT-1/trained models/H5 files/vgg19(99_89).h5',                   'pre
}
CLASS_LABELS = ["Immature Cataract", "Mature Cataract", "Normal"]
IMG_SIZE = (224, 224)

@st.cache_resource
def load_models():
    models = {}
    for model_name, item in model_info.items():
        models[model_name] = tf.keras.models.load_model(item['path'], compile=False)
    return models

def preprocess_img(img, pre_func):
    img = img.resize(IMG_SIZE)
    arr = image.img_to_array(img)
    arr = np.expand_dims(arr, axis=0)
    arr = pre_func(arr)
    return arr

def ensemble_predict(image_pil, models):
    all_probs = []
    for model_name, model in models.items():
        pre_func = model_info[model_name]['preprocess']
        processed = preprocess_img(image_pil, pre_func)
        pred = model.predict(processed, verbose=0)[0]
        all_probs.append(pred)
    all_probs = np.array(all_probs)
    avg_probs = all_probs.mean(axis=0)
    best_idx = np.argmax(avg_probs)
    return CLASS_LABELS[best_idx], avg_probs[best_idx]*100, avg_probs

st.title("👁 Cataract Detection System — Ensemble")
st.markdown("Upload an eye image to detect and classify cataracts using 4 models (average confidence shown).")

with st.sidebar:
    st.info("""
        This system uses DenseNet121, EfficientNetB3, ResNet, and VGG19.
        The predicted class is based on **average confidence** across all four models.
        Make sure your models are accessible at the paths above!
    """)
```

```
uploaded_file = st.file_uploader(
    "Choose an eye image...", type=['jpg', 'jpeg', 'png']
)
if uploaded_file is not None:
    image_pil = Image.open(uploaded_file).convert('RGB')
    st.image(image_pil, caption='Uploaded Image', use_container_width=True)
    if st.button(' Analyze Image'):
        with st.spinner("Analyzing with all models..."):
            models = load_models()
            result, confidence, avg_probs = ensemble_predict(image_pil, models)
        st.markdown(f"### Prediction: <span style='color:green'>{result}</span>", unsafe_allow_html=True)
        st.markdown(f"### Average Confidence: <span style='color:white'>{confidence:.2f}%</span>", unsafe_allow_html=True)
        st.write("**Average class probabilities from all models:**")
        for label, prob in zip(CLASS_LABELS, avg_probs):
            st.write(f"- **{label}**: {prob*100:.2f}%")
else:
    st.info("Please upload an eye image to begin analysis")
```

```
Writing app.py
```

```
from pyngrok import ngrok
import subprocess
import time

ngrok.set_auth_token("358HfFSjXSmlojBiorl4mNse6xs_MJS8wgAkf3qAxFhZUVnY")
!pkill -f streamlit
subprocess.Popen(["streamlit", "run", "app.py", "--server.port", "8501", "--server.headless", "true"])
time.sleep(7)
public_url = ngrok.connect(8501)
print("\n" + "="*60)
print(f"🎉 Your Cataract Detection site is LIVE!\n🌐 Public URL: {public_url}\n")
print("Keep this Colab tab open while using your public site.")
print("="*60 + "\n")
```

```
============================================================
🎉 Your Cataract Detection site is LIVE!
🌐 Public URL: NgrokTunnel: "https://von-operatable-verbatim.ngrok-free.dev" -> "http://localhost:8501"

Keep this Colab tab open while using your public site.
============================================================
```

```
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
# Model file paths
model_paths = {
    "inceptionresnetv2": "/content/drive/MyDrive/PROJECT/PROJECT-1/trained models/inceptionresnetv2(100_00).h5",
    "resnet": "/content/drive/MyDrive/PROJECT/PROJECT-1/trained models/resnet(99_91).h5",
    "efficientnetb3": "/content/drive/MyDrive/PROJECT/PROJECT-1/trained models/efficientnetb3(99_84).h5",
    "vgg19": "/content/drive/MyDrive/PROJECT/PROJECT-1/trained models/vgg19(99_89).h5",
    "mobilenetv3": "/content/drive/MyDrive/PROJECT/PROJECT-1/trained models/mobilenetv3(99_92).h5",
    "densenet121": "/content/drive/MyDrive/PROJECT/PROJECT-1/trained models/densenet121(99_19).h5"
}
```

```
from tensorflow.keras.models import load_model
```