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

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

```
print("\n🔧 Installing required packages...")
!pip install imbalanced-learn scikit-learn opencv-python-headless --quiet
```

```
🔧 Installing required packages...
```

```
import os
import shutil
import numpy as np
import cv2
import matplotlib.pyplot as plt
from pathlib import Path
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint
from tensorflow.keras.applications import MobileNetV2
import seaborn as sns
from collections import Counter
import pickle
import warnings
warnings.filterwarnings('ignore')

print(f"\n✅ TensorFlow version: {tf.__version__}")
print(f"✅ GPU Available: {len(tf.config.list_physical_devices('GPU')) > 0}")
```

```
✅ TensorFlow version: 2.19.0
✅ GPU Available: True
```

```
# ==================== PART 2: CONFIGURATION ====================
class Config:
    # Paths - YOUR ACTUAL GOOGLE DRIVE STRUCTURE
    BASE_PATH = '/content/drive/MyDrive/acne-dataset-fin'  # ✅ YOUR FOLDER
    PROCESSED_PATH = '/content/Acne_Processed'
    MODEL_PATH = '/content/models'
    DRIVE_MODEL_PATH = '/content/drive/MyDrive/acne_models_final'

    # Image parameters
    IMG_SIZE = (224, 224)
    BATCH_SIZE = 8  # Optimized for 7 classes

    # Training parameters
    EPOCHS = 80  # Sufficient for 7 classes
    LEARNING_RATE = 0.0001
    VALIDATION_SPLIT = 0.15  # More training data

    # Augmentation parameters
    TARGET_SAMPLES = 400  # Target samples per class after augmentation

    # Classes - YOUR 7 ACNE TYPES + NORMAL SKIN
    CLASSES = ['blackheads', 'cysts', 'darkspots', 'normal', 'papules', 'pustules', 'whiteheads']

config = Config()

# Create necessary directories
os.makedirs(config.PROCESSED_PATH, exist_ok=True)
os.makedirs(config.MODEL_PATH, exist_ok=True)
os.makedirs(config.DRIVE_MODEL_PATH, exist_ok=True)

print("\n" + "="*70)
print("🎯 ACNE DETECTION MODEL - TRAINING CONFIGURATION")
print("="*70)
print(f"📁 Dataset Path: {config.BASE_PATH}")
print(f"📊 Classes: {len(config.CLASSES)} types")
for i, cls in enumerate(config.CLASSES, 1):
    print(f"   {i}. {cls}")
print(f"\n⚙️  Target samples per class: {config.TARGET_SAMPLES}")
print(f"⚙️  Image size: {config.IMG_SIZE}")
```

```
print(f"⚙️  Batch size: {config.BATCH_SIZE}")
print(f"⚙️  Training epochs: {config.EPOCHS}")
print(f"⚙️  Validation split: {config.VALIDATION_SPLIT*100}%")
```

```
======================================================================
🎯 ACNE DETECTION MODEL - TRAINING CONFIGURATION
======================================================================
📁 Dataset Path: /content/drive/MyDrive/acne-dataset-fin
📊 Classes: 7 types
   1. blackheads
   2. cysts
   3. darkspots
   4. normal
   5. papules
   6. pustules
   7. whiteheads

⚙️  Target samples per class: 400
⚙️  Image size: (224, 224)
⚙️  Batch size: 8
⚙️  Training epochs: 80
⚙️  Validation split: 15.0%
```

```
# ==================== PART 3: PRE-FLIGHT VERIFICATION ====================
print("\n" + "="*70)
print("🔍 PRE-FLIGHT VERIFICATION")
print("="*70)

# Check if base path exists
if not os.path.exists(config.BASE_PATH):
    print(f"❌ ERROR: Dataset path does NOT exist!")
    print(f"   Looking for: {config.BASE_PATH}")
    print("\n🔍 Available folders in MyDrive:")
    mydrive_path = '/content/drive/MyDrive'
    if os.path.exists(mydrive_path):
        folders = [f for f in os.listdir(mydrive_path) if os.path.isdir(os.path.join(mydrive_path, f))]
        for folder in folders[:15]:
            print(f"   📁 {folder}")
    print("\n⚠️  Please update config.BASE_PATH with the correct folder name!")
    raise FileNotFoundError(f"Dataset path not found: {config.BASE_PATH}")
else:
    print(f"✅ Base path verified: {config.BASE_PATH}")

# Check class folders and count images
print("\n📁 Checking class folders:")
total_images = 0
missing_classes = []
low_count_classes = []

for class_name in config.CLASSES:
    class_path = os.path.join(config.BASE_PATH, class_name)
    if os.path.exists(class_path):
        files = [f for f in os.listdir(class_path) if f.lower().endswith(('.jpg', '.jpeg', '.png', '.bmp'))]
        file_count = len(files)
        total_images += file_count

        status = "✅"
        warning = ""
        if file_count < 30:
            status = "⚠️ "
            warning = " (LOW - May affect accuracy!)"
            low_count_classes.append((class_name, file_count))
        elif file_count < 50:
            warning = " (Borderline - Will be augmented)"

        print(f"   {status} {class_name:12s}: {file_count:3d} images{warning}")
    else:
        print(f"   ❌ {class_name:12s}: FOLDER NOT FOUND!")
        missing_classes.append(class_name)

if missing_classes:
    print(f"\n❌ CRITICAL ERROR: Missing class folders: {missing_classes}")
    print("🔍 Available folders in dataset:")
    if os.path.exists(config.BASE_PATH):
        folders = [f for f in os.listdir(config.BASE_PATH) if os.path.isdir(os.path.join(config.BASE_PATH, f))]
        for folder in folders:
            print(f"   📁 {folder}")
    print("\n⚠️  Please update config.CLASSES with exact folder names (case-sensitive)!")
    raise FileNotFoundError(f"Missing class folders: {missing_classes}")

if low_count_classes:
```

```
    print(f"\n⚠️  WARNING: {len(low_count_classes)} class(es) have <30 images:")
    for cls, count in low_count_classes:
        print(f"   - {cls}: {count} images")
    print("   Recommendation: Add more images or expect lower accuracy for these classes")

print(f"\n✅ Total images found: {total_images}")
print(f"✅ Average per class: {total_images//len(config.CLASSES)}")
print("✅ All pre-flight checks passed! Ready to proceed.")
print("="*70)
```

```
======================================================================
🔍 PRE-FLIGHT VERIFICATION
======================================================================
✅ Base path verified: /content/drive/MyDrive/acne-dataset-fin

📁 Checking class folders:
   ✅ blackheads  : 250 images
   ✅ cysts       :  46 images (Borderline - Will be augmented)
   ✅ darkspots   : 250 images
   ✅ normal      : 807 images
   ✅ papules     :  52 images
   ✅ pustules    :  51 images
   ✅ whiteheads  :  65 images

✅ Total images found: 1521
✅ Average per class: 217
✅ All pre-flight checks passed! Ready to proceed.
======================================================================
```

```
# ==================== PART 4: DATA ORGANIZATION & RENAMING ====================
def organize_and_rename_files():
    """Organize dataset and rename files with consistent naming"""
    print("\n" + "="*70)
    print("📝 STEP 1: Organizing and Renaming Files")
    print("="*70)

    class_counts = {}
    renamed_count = 0

    for class_name in config.CLASSES:
        source_dir = os.path.join(config.BASE_PATH, class_name)
        dest_dir = os.path.join(config.PROCESSED_PATH, class_name)
        os.makedirs(dest_dir, exist_ok=True)

        if not os.path.exists(source_dir):
            print(f"⚠️  Warning: {source_dir} not found! Skipping...")
            class_counts[class_name] = 0
            continue

        # Get all image files
        valid_extensions = ('.jpg', '.jpeg', '.png', '.bmp')
        files = [f for f in os.listdir(source_dir)
                 if f.lower().endswith(valid_extensions)]

        print(f"\n📁 Processing {class_name}:")
        print(f"   Found: {len(files)} images")

        # Copy and rename files with verification
        success_count = 0
        for idx, filename in enumerate(files, 1):
            src_path = os.path.join(source_dir, filename)
            ext = os.path.splitext(filename)[1].lower()
            new_filename = f"{class_name}_{idx:04d}{ext}"
            dest_path = os.path.join(dest_dir, new_filename)

            try:
                shutil.copy2(src_path, dest_path)
                if os.path.exists(dest_path):
                    success_count += 1
                    renamed_count += 1
                    if idx % 50 == 0:
                        print(f"   ✓ Renamed {idx}/{len(files)} files...", end='\r')
            except Exception as e:
                print(f"   ✗ Failed to copy {filename}: {e}")

        class_counts[class_name] = success_count
        print(f"   ✅ Successfully renamed: {success_count}/{len(files)} files")

    print("\n" + "="*70)
    print("📊 Renaming Summary:")
    print("="*70)
```

```
    for class_name, count in class_counts.items():
        print(f"   {class_name:12s}: {count:3d} files renamed")
    print(f"\n✅ Total files renamed: {renamed_count}")

    return class_counts

class_counts = organize_and_rename_files()
```

```
======================================================================
📝 STEP 1: Organizing and Renaming Files
======================================================================

📁 Processing blackheads:
   Found: 250 images
   ✅ Successfully renamed: 250/250 files

📁 Processing cysts:
   Found: 46 images
   ✅ Successfully renamed: 46/46 files

📁 Processing darkspots:
   Found: 250 images
   ✅ Successfully renamed: 250/250 files

📁 Processing normal:
   Found: 807 images
   ✅ Successfully renamed: 807/807 files

📁 Processing papules:
   Found: 52 images
   ✅ Successfully renamed: 52/52 files

📁 Processing pustules:
   Found: 51 images
   ✅ Successfully renamed: 51/51 files

📁 Processing whiteheads:
   Found: 65 images
   ✅ Successfully renamed: 65/65 files

======================================================================
📊 Renaming Summary:
======================================================================
   blackheads  : 250 files renamed
   cysts       :  46 files renamed
   darkspots   : 250 files renamed
   normal      : 807 files renamed
   papules     :  52 files renamed
   pustules    :  51 files renamed
   whiteheads  :  65 files renamed

✅ Total files renamed: 1521
```

```
# ==================== PART 5: ENHANCED DATA AUGMENTATION ====================
def augment_minority_classes():
    """Augment underrepresented classes with enhanced transformations"""
    print("\n" + "="*70)
    print("🔄 STEP 2: Data Augmentation (Enhanced)")
    print("="*70)
    print("Augmentation includes: rotation, flipping (horizontal & vertical),")
    print("zoom, shift, shear, brightness, and color variations")

    # ENHANCED augmentation generator with more aggressive transformations
    aug_generator = ImageDataGenerator(
        rotation_range=40,          # Rotate up to 40 degrees
        width_shift_range=0.25,     # Shift horizontally
        height_shift_range=0.25,    # Shift vertically
        shear_range=0.25,           # Shear transformation
        zoom_range=0.3,             # Zoom in/out
        horizontal_flip=True,       # ✅ Horizontal inverse/mirror
        vertical_flip=True,         # ✅ Vertical inverse/mirror
        brightness_range=[0.7, 1.3], # Brightness variations
        channel_shift_range=20,     # Color channel shifts
        fill_mode='nearest'
    )

    augmentation_summary = []

    for class_name in config.CLASSES:
        class_dir = os.path.join(config.PROCESSED_PATH, class_name)

        if not os.path.exists(class_dir):
            print(f"\n⚠️  Skipping {class_name}: folder doesn't exist")
```

```
                continue

            current_count = len([f for f in os.listdir(class_dir)
                                 if f.lower().endswith(('.jpg', '.jpeg', '.png'))])

            if current_count == 0:
                print(f"\n⚠️  Skipping {class_name}: no images found")
                continue

            if current_count < config.TARGET_SAMPLES:
                augment_count = config.TARGET_SAMPLES - current_count
                print(f"\n📸 Augmenting {class_name}:")
                print(f"   Current: {current_count} images")
                print(f"   Target:  {config.TARGET_SAMPLES} images")
                print(f"   To generate: {augment_count} augmented images")

                # Get existing images
                existing_files = [f for f in os.listdir(class_dir)
                                  if f.lower().endswith(('.jpg', '.jpeg', '.png'))]

                aug_idx = current_count + 1
                augmented = 0

                while augmented < augment_count:
                    # Randomly select an image
                    img_file = np.random.choice(existing_files)
                    img_path = os.path.join(class_dir, img_file)

                    try:
                        # Read and preprocess image
                        img = cv2.imread(img_path)
                        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                        img = cv2.resize(img, config.IMG_SIZE)
                        img = img.reshape((1,) + img.shape)

                        # Generate augmented image
                        for batch in aug_generator.flow(img, batch_size=1):
                            aug_img = batch[0].astype(np.uint8)

                            # Save augmented image
                            new_filename = f"{class_name}_aug_{aug_idx:04d}.jpg"
                            save_path = os.path.join(class_dir, new_filename)
                            cv2.imwrite(save_path, cv2.cvtColor(aug_img, cv2.COLOR_RGB2BGR))

                            aug_idx += 1
                            augmented += 1

                            if augmented % 30 == 0:
                                progress = (augmented / augment_count) * 100
                                print(f"   Progress: {augmented}/{augment_count} ({progress:.1f}%)", end='\r')

                            break
                    except Exception as e:
                        print(f"\n    ⚠️  Error augmenting {img_file}: {e}")

                final_count = len([f for f in os.listdir(class_dir)
                                   if f.lower().endswith(('.jpg', '.jpeg', '.png'))])
                print(f"\n    ✅ Completed: {augmented} new images generated")
                print(f"    📊 Final count: {final_count} images")
                augmentation_summary.append((class_name, current_count, final_count, augmented))
            else:
                print(f"\n✅ {class_name}: Already has {current_count} images (no augmentation needed)")
                augmentation_summary.append((class_name, current_count, current_count, 0))

        # Print summary
        print("\n" + "="*70)
        print("📊 Augmentation Summary:")
        print("="*70)
        print(f"{'Class':<15} {'Original':<10} {'Final':<10} {'Generated':<10}")
        print("-"*70)
        for cls, orig, final, gen in augmentation_summary:
            print(f"{cls:<15} {orig:<10} {final:<10} {gen:<10}")

        total_generated = sum(item[3] for item in augmentation_summary)
        print("-"*70)
        print(f"{'TOTAL':<15} {'':<10} {'':<10} {total_generated:<10}")
        print("="*70)

    augment_minority_classes()

    # ==================== PART 6: ADVANCED PREPROCESSING ====================
    def preprocess_image(img_path):
```

```
    """Advanced preprocessing pipeline with CLAHE"""
    img = cv2.imread(img_path)
    if img is None:
        raise ValueError(f"Failed to load image: {img_path}")

    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, config.IMG_SIZE)

    # Apply CLAHE for better contrast
    lab = cv2.cvtColor(img, cv2.COLOR_RGB2LAB)
    l, a, b = cv2.split(lab)
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    l = clahe.apply(l)
    img = cv2.cvtColor(cv2.merge([l,a,b]), cv2.COLOR_LAB2RGB)

    # Normalize to [0, 1]
    img = img.astype(np.float32) / 255.0

    return img
```

```
📸 Augmenting cysts:
   Current: 46 images
   Target:  400 images
   To generate: 354 augmented images

   ✅ Completed: 354 new images generated
   📊 Final count: 400 images

📸 Augmenting darkspots:
   Current: 250 images
   Target:  400 images
   To generate: 150 augmented images
   Progress: 150/150 (100.0%)
   ✅ Completed: 150 new images generated
   📊 Final count: 400 images

✅ normal: Already has 807 images (no augmentation needed)

📸 Augmenting papules:
   Current: 52 images
   Target:  400 images
   To generate: 348 augmented images

   ✅ Completed: 348 new images generated
   📊 Final count: 400 images

📸 Augmenting pustules:
   Current: 51 images
   Target:  400 images
   To generate: 349 augmented images

   ✅ Completed: 349 new images generated
   📊 Final count: 400 images

📸 Augmenting whiteheads:
   Current: 65 images
   Target:  400 images
   To generate: 335 augmented images
   Progress: 330/335 (98.5%)
   ✅ Completed: 335 new images generated
   📊 Final count: 400 images

======================================================================
📊 Augmentation Summary:
======================================================================
Class           Original   Final      Generated
----------------------------------------------------------------------
blackheads      250        400        150
cysts           46         400        354
darkspots       250        400        150
normal          807        807        0
papules         52         400        348
pustules        51         400        349
whiteheads      65         400        335
----------------------------------------------------------------------
TOTAL                                 1686
======================================================================
```

```
# ==================== PART 7: DATA LOADING ====================
def load_data():
    """Load and split data with proper shuffling"""
    print("\n" + "="*70)
    print("📥 STEP 3: Loading and Splitting Data")
    print("="*70)

    X, y = [], []
```

```
    class_to_idx = {class_name: idx for idx, class_name in enumerate(config.CLASSES)}

    for class_name in config.CLASSES:
        class_dir = os.path.join(config.PROCESSED_PATH, class_name)

        if not os.path.exists(class_dir):
            print(f"⚠️  Warning: {class_name} folder not found, skipping...")
            continue

        files = [f for f in os.listdir(class_dir)
                 if f.lower().endswith(('.jpg', '.jpeg', '.png'))]

        print(f"📂 Loading {class_name}: {len(files)} images")

        loaded = 0
        for img_file in files:
            img_path = os.path.join(class_dir, img_file)
            try:
                img = preprocess_image(img_path)
                X.append(img)
                y.append(class_to_idx[class_name])
                loaded += 1
            except Exception as e:
                print(f"   ⚠️  Failed to load {img_file}: {e}")

        print(f"   ✅ Loaded: {loaded}/{len(files)} images")

    X = np.array(X)
    y = np.array(y)

    print(f"\n📊 Dataset shape: {X.shape}")
    print(f"📊 Labels shape: {y.shape}")

    # Shuffle data
    indices = np.random.permutation(len(X))
    X = X[indices]
    y = y[indices]

    # Split data
    split_idx = int(len(X) * (1 - config.VALIDATION_SPLIT))
    X_train, X_val = X[:split_idx], X[split_idx:]
    y_train, y_val = y[:split_idx], y[split_idx:]

    print(f"\n✅ Data split completed:")
    print(f"   Training set:   {len(X_train)} samples ({(1-config.VALIDATION_SPLIT)*100:.0f}%)")
    print(f"   Validation set: {len(X_val)} samples ({config.VALIDATION_SPLIT*100:.0f}%)")

    print(f"\n📊 Training set class distribution:")
    for class_name, idx in class_to_idx.items():
        count = np.sum(y_train == idx)
        percentage = (count / len(y_train)) * 100
        print(f"   {class_name:12s}: {count:3d} samples ({percentage:5.1f}%)")

    print(f"\n📊 Validation set class distribution:")
    for class_name, idx in class_to_idx.items():
        count = np.sum(y_val == idx)
        percentage = (count / len(y_val)) * 100
        print(f"   {class_name:12s}: {count:3d} samples ({percentage:5.1f}%)")

    return X_train, X_val, y_train, y_val

X_train, X_val, y_train, y_val = load_data()
```

```
======================================================================
📥 STEP 3: Loading and Splitting Data
======================================================================
📂 Loading blackheads: 400 images
   ✅ Loaded: 400/400 images
📂 Loading cysts: 400 images
   ✅ Loaded: 400/400 images
📂 Loading darkspots: 400 images
   ✅ Loaded: 400/400 images
📂 Loading normal: 807 images
   ✅ Loaded: 807/807 images
📂 Loading papules: 400 images
   ✅ Loaded: 400/400 images
📂 Loading pustules: 400 images
   ✅ Loaded: 400/400 images
📂 Loading whiteheads: 400 images
   ✅ Loaded: 400/400 images

📊 Dataset shape: (3207, 224, 224, 3)
```

```
📊 Labels shape: (3207,)

✅ Data split completed:
   Training set:   2725 samples (85%)
   Validation set: 482 samples (15%)

📊 Training set class distribution:
   blackheads  : 334 samples ( 12.3%)
   cysts       : 342 samples ( 12.6%)
   darkspots   : 349 samples ( 12.8%)
   normal      : 681 samples ( 25.0%)
   papules     : 333 samples ( 12.2%)
   pustules    : 344 samples ( 12.6%)
   whiteheads  : 342 samples ( 12.6%)

📊 Validation set class distribution:
   blackheads  :  66 samples ( 13.7%)
   cysts       :  58 samples ( 12.0%)
   darkspots   :  51 samples ( 10.6%)
   normal      : 126 samples ( 26.1%)
   papules     :  67 samples ( 13.9%)
   pustules    :  56 samples ( 11.6%)
   whiteheads  :  58 samples ( 12.0%)
```

```python
# ==================== PART 8: CNN FEATURE EXTRACTOR ====================
def build_cnn_feature_extractor():
    """Build CNN with MobileNetV2 backbone for feature extraction"""
    print("\n" + "="*70)
    print("🏗️  STEP 4: Building CNN Feature Extractor")
    print("="*70)

    # Use MobileNetV2 as base (pre-trained on ImageNet)
    base_model = MobileNetV2(
        input_shape=(*config.IMG_SIZE, 3),
        include_top=False,
        weights='imagenet'
    )

    # Freeze base model initially
    base_model.trainable = False
    print("✅ MobileNetV2 base loaded (ImageNet weights)")

    # Build custom top with increased capacity for 7 classes
    model = models.Sequential([
        base_model,
        layers.GlobalAveragePooling2D(),
        layers.BatchNormalization(),
        layers.Dropout(0.5),
        layers.Dense(512, activation='relu', kernel_regularizer=keras.regularizers.l2(0.001)),
        layers.BatchNormalization(),
        layers.Dropout(0.4),
        layers.Dense(256, activation='relu', kernel_regularizer=keras.regularizers.l2(0.001)),
        layers.BatchNormalization(),
        layers.Dropout(0.3),
        layers.Dense(128, activation='relu', name='feature_layer'),  # Feature extraction layer
        layers.BatchNormalization(),
        layers.Dropout(0.2),
        layers.Dense(len(config.CLASSES), activation='softmax')
    ])

    model.compile(
        optimizer=keras.optimizers.Adam(learning_rate=config.LEARNING_RATE),
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )

    print("✅ CNN model architecture built")
    print(f"   Total parameters: {model.count_params():,}")
    print(f"   Trainable parameters: {sum([tf.size(w).numpy() for w in model.trainable_weights]):,}")

    return model, base_model

cnn_model, base_model = build_cnn_feature_extractor()
```

```
======================================================================
🏗️  STEP 4: Building CNN Feature Extractor
======================================================================
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_orde
9406464/9406464 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
✅ MobileNetV2 base loaded (ImageNet weights)
✅ CNN model architecture built
   Total parameters: 3,087,687
```

```
Trainable parameters: 825,351
```

```
# ==================== PART 9: TRAIN CNN ====================
def train_cnn():
    """Train CNN with two-stage approach: initial + fine-tuning"""
    print("\n" + "="*70)
    print("🚀 STEP 5: Training CNN (Two-Stage Training)")
    print("="*70)

    # Enhanced data augmentation for training
    train_datagen = ImageDataGenerator(
        rotation_range=20,
        width_shift_range=0.15,
        height_shift_range=0.15,
        zoom_range=0.15,
        horizontal_flip=True,
        fill_mode='nearest'
    )

    # Callbacks
    callbacks = [
        EarlyStopping(
            monitor='val_loss',
            patience=15,
            restore_best_weights=True,
            verbose=1
        ),
        ReduceLROnPlateau(
            monitor='val_loss',
            factor=0.5,
            patience=7,
            min_lr=1e-7,
            verbose=1
        ),
        ModelCheckpoint(
            os.path.join(config.MODEL_PATH, 'best_cnn_model.h5'),
            monitor='val_accuracy',
            save_best_only=True,
            verbose=1
        )
    ]

    # STAGE 1: Initial training with frozen base
    print("\n" + "-"*70)
    print("🎯 STAGE 1: Initial Training (Base Frozen)")
    print("-"*70)

    history = cnn_model.fit(
        train_datagen.flow(X_train, y_train, batch_size=config.BATCH_SIZE),
        validation_data=(X_val, y_val),
        epochs=config.EPOCHS,
        callbacks=callbacks,
        verbose=1
    )

    # Evaluate after initial training
    val_loss, val_acc = cnn_model.evaluate(X_val, y_val, verbose=0)
    print(f"\n✅ Stage 1 completed - Validation accuracy: {val_acc*100:.2f}%")

    # STAGE 2: Fine-tuning with unfrozen layers
    print("\n" + "-"*70)
    print("🔄 STAGE 2: Fine-Tuning (Unfreezing Last Layers)")
    print("-"*70)

    base_model.trainable = True
    # Freeze all layers except last 30
    for layer in base_model.layers[:-30]:
        layer.trainable = False

    trainable_count = sum([1 for layer in base_model.layers if layer.trainable])
    print(f"✅ Unfrozen {trainable_count} layers in base model")

    # Recompile with lower learning rate
    cnn_model.compile(
        optimizer=keras.optimizers.Adam(learning_rate=config.LEARNING_RATE/10),
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )

    history_finetune = cnn_model.fit(
        train_datagen.flow(X_train, y_train, batch_size=config.BATCH_SIZE),
```

```
        validation_data=(X_val, y_val),
        epochs=25,
        callbacks=callbacks,
        verbose=1
    )

    # Final evaluation
    val_loss, val_acc = cnn_model.evaluate(X_val, y_val, verbose=0)
    print(f"\n✅ Stage 2 completed - Final validation accuracy: {val_acc*100:.2f}%")

    return history, history_finetune

history, history_finetune = train_cnn()
```

```
Epoch 7/25
341/341 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step - accuracy: 0.7422 - loss: 1.4282
Epoch 7: val_accuracy did not improve from 0.91286
341/341 ━━━━━━━━━━━━━━━━━━━━ 32s 95ms/step - accuracy: 0.7422 - loss: 1.4282 - val_accuracy: 0.9066 - val_loss: 0.9883 - learni
Epoch 8/25
341/341 ━━━━━━━━━━━━━━━━━━━━ 0s 98ms/step - accuracy: 0.7792 - loss: 1.3542
Epoch 8: val_accuracy did not improve from 0.91286
341/341 ━━━━━━━━━━━━━━━━━━━━ 34s 99ms/step - accuracy: 0.7791 - loss: 1.3542 - val_accuracy: 0.9025 - val_loss: 0.9980 - learni
Epoch 9/25
341/341 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step - accuracy: 0.7539 - loss: 1.3890
Epoch 9: val_accuracy did not improve from 0.91286
341/341 ━━━━━━━━━━━━━━━━━━━━ 32s 95ms/step - accuracy: 0.7539 - loss: 1.3890 - val_accuracy: 0.9025 - val_loss: 1.0019 - learni
Epoch 10/25
341/341 ━━━━━━━━━━━━━━━━━━━━ 0s 98ms/step - accuracy: 0.7726 - loss: 1.3424
Epoch 10: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-06.

Epoch 10: val_accuracy did not improve from 0.91286
341/341 ━━━━━━━━━━━━━━━━━━━━ 34s 99ms/step - accuracy: 0.7726 - loss: 1.3424 - val_accuracy: 0.9025 - val_loss: 0.9915 - learni
Epoch 11/25
341/341 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step - accuracy: 0.7865 - loss: 1.2914
Epoch 11: val_accuracy did not improve from 0.91286
341/341 ━━━━━━━━━━━━━━━━━━━━ 33s 95ms/step - accuracy: 0.7865 - loss: 1.2913 - val_accuracy: 0.9046 - val_loss: 0.9932 - learni
Epoch 12/25
341/341 ━━━━━━━━━━━━━━━━━━━━ 0s 99ms/step - accuracy: 0.7810 - loss: 1.3265
Epoch 12: val_accuracy did not improve from 0.91286
341/341 ━━━━━━━━━━━━━━━━━━━━ 34s 101ms/step - accuracy: 0.7809 - loss: 1.3265 - val_accuracy: 0.9025 - val_loss: 0.9938 - learr
Epoch 13/25
341/341 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step - accuracy: 0.7507 - loss: 1.4133
Epoch 13: val_accuracy did not improve from 0.91286
341/341 ━━━━━━━━━━━━━━━━━━━━ 32s 95ms/step - accuracy: 0.7508 - loss: 1.4131 - val_accuracy: 0.8942 - val_loss: 0.9959 - learni
Epoch 14/25
341/341 ━━━━━━━━━━━━━━━━━━━━ 0s 98ms/step - accuracy: 0.7840 - loss: 1.3347
Epoch 14: val_accuracy did not improve from 0.91286
341/341 ━━━━━━━━━━━━━━━━━━━━ 34s 100ms/step - accuracy: 0.7840 - loss: 1.3347 - val_accuracy: 0.9004 - val_loss: 0.9968 - learr
Epoch 15/25
341/341 ━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step - accuracy: 0.7755 - loss: 1.3211
Epoch 15: val_accuracy did not improve from 0.91286
341/341 ━━━━━━━━━━━━━━━━━━━━ 32s 95ms/step - accuracy: 0.7756 - loss: 1.3211 - val_accuracy: 0.9066 - val_loss: 0.9917 - learni
Epoch 16/25
341/341 ━━━━━━━━━━━━━━━━━━━━ 0s 98ms/step - accuracy: 0.7826 - loss: 1.2935
Epoch 16: val_accuracy did not improve from 0.91286
341/341 ━━━━━━━━━━━━━━━━━━━━ 34s 99ms/step - accuracy: 0.7825 - loss: 1.2935 - val_accuracy: 0.9025 - val_loss: 0.9886 - learni
Epoch 17/25
341/341 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step - accuracy: 0.7806 - loss: 1.3139
Epoch 17: ReduceLROnPlateau reducing learning rate to 2.499999936844688e-06.

Epoch 17: val_accuracy improved from 0.91286 to 0.91494, saving model to /content/models/best_cnn_model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
341/341 ━━━━━━━━━━━━━━━━━━━━ 33s 96ms/step - accuracy: 0.7806 - loss: 1.3138 - val_accuracy: 0.9149 - val_loss: 0.9847 - learni
Epoch 18/25
341/341 ━━━━━━━━━━━━━━━━━━━━ 0s 97ms/step - accuracy: 0.7858 - loss: 1.2971
Epoch 18: val_accuracy did not improve from 0.91494
341/341 ━━━━━━━━━━━━━━━━━━━━ 34s 99ms/step - accuracy: 0.7858 - loss: 1.2971 - val_accuracy: 0.9108 - val_loss: 0.9815 - learni
Epoch 18: early stopping
Restoring model weights from the end of the best epoch: 3.

✅ Stage 2 completed - Final validation accuracy: 90.87%
```

```
# ==================== PART 10: FEATURE EXTRACTION FOR SVM ====================
print("\n" + "="*70)
print("🔧 STEP 6: Extracting Features for SVM")
print("="*70)

# Initialize model with dummy input
dummy_input = np.zeros((1, *config.IMG_SIZE, 3), dtype=np.float32)
_ = cnn_model(dummy_input)
print("✅ Model initialized")

def extract_features_batch(images, model, batch_size=16):
    """Extract features from the feature_layer in batches"""
    all_features = []
```

```
    total_batches = (len(images) + batch_size - 1) // batch_size

    print(f"Extracting features from {len(images)} images in {total_batches} batches...")

    for i in range(0, len(images), batch_size):
        batch = images[i:i+batch_size]

        # Forward pass through model until feature_layer
        x = batch
        for layer in model.layers:
            x = layer(x, training=False)
            if layer.name == 'feature_layer':
                all_features.append(x.numpy())
                break

        if (i // batch_size + 1) % 10 == 0:
            progress = ((i + len(batch)) / len(images)) * 100
            print(f"  Progress: {i + len(batch)}/{len(images)} ({progress:.1f}%)")

    return np.vstack(all_features)

# Extract training features
print("\n📊 Extracting training features...")
X_train_features = extract_features_batch(X_train, cnn_model, batch_size=config.BATCH_SIZE)
print(f"✅ Training features shape: {X_train_features.shape}")

# Extract validation features
print("\n📊 Extracting validation features...")
X_val_features = extract_features_batch(X_val, cnn_model, batch_size=config.BATCH_SIZE)
print(f"✅ Validation features shape: {X_val_features.shape}")

# Scale features
print("\n🔄 Scaling features...")
scaler = StandardScaler()
X_train_features = scaler.fit_transform(X_train_features)
X_val_features = scaler.transform(X_val_features)
print("✅ Features scaled successfully")

# ==================== PART 11: TRAIN SVM ====================
print("\n" + "="*70)
print("🎯 STEP 7: Training SVM Classifier")
print("="*70)

svm_classifier = SVC(
    kernel='rbf',
    C=100,  # Increased for better separation
    gamma='scale',
    probability=True,
    class_weight='balanced',
    random_state=42,
    cache_size=1000,
    verbose=True
)

print("🚀 Training SVM on extracted features...")
svm_classifier.fit(X_train_features, y_train)
print("✅ SVM training completed")
```

```
✅ Model initialized

📊 Extracting training features...
Extracting features from 2725 images in 341 batches...
  Progress: 80/2725 (2.9%)
  Progress: 160/2725 (5.9%)
  Progress: 240/2725 (8.8%)
  Progress: 320/2725 (11.7%)
  Progress: 400/2725 (14.7%)
  Progress: 480/2725 (17.6%)
  Progress: 560/2725 (20.6%)
  Progress: 640/2725 (23.5%)
  Progress: 720/2725 (26.4%)
  Progress: 800/2725 (29.4%)
  Progress: 880/2725 (32.3%)
  Progress: 960/2725 (35.2%)
  Progress: 1040/2725 (38.2%)
  Progress: 1120/2725 (41.1%)
  Progress: 1200/2725 (44.0%)
  Progress: 1280/2725 (47.0%)
  Progress: 1360/2725 (49.9%)
```

```
  Progress: 1760/2725 (64.6%)
  Progress: 1840/2725 (67.5%)
  Progress: 1920/2725 (70.5%)
  Progress: 2000/2725 (73.4%)
  Progress: 2080/2725 (76.3%)
  Progress: 2160/2725 (79.3%)
  Progress: 2240/2725 (82.2%)
  Progress: 2320/2725 (85.1%)
  Progress: 2400/2725 (88.1%)
  Progress: 2480/2725 (91.0%)
  Progress: 2560/2725 (93.9%)
  Progress: 2640/2725 (96.9%)
  Progress: 2720/2725 (99.8%)
✅ Training features shape: (2725, 128)

📊 Extracting validation features...
Extracting features from 482 images in 61 batches...
  Progress: 80/482 (16.6%)
  Progress: 160/482 (33.2%)
  Progress: 240/482 (49.8%)
  Progress: 320/482 (66.4%)
  Progress: 400/482 (83.0%)
  Progress: 480/482 (99.6%)
✅ Validation features shape: (482, 128)

🔄 Scaling features...
✅ Features scaled successfully

======================================================================
🎯 STEP 7: Training SVM Classifier
======================================================================
🚀 Training SVM on extracted features...
[LibSVM]✅ SVM training completed
```

```
# ==================== PART 12: COMPREHENSIVE EVALUATION ====================
print("\n" + "="*70)
print("📊 STEP 8: Model Evaluation")
print("="*70)

# CNN-only predictions
print("\n🔍 Evaluating CNN-only model...")
y_pred_cnn = np.argmax(cnn_model.predict(X_val, verbose=0), axis=1)
cnn_accuracy = accuracy_score(y_val, y_pred_cnn)

# Hybrid (CNN+SVM) predictions
print("🔍 Evaluating CNN+SVM hybrid model...")
y_pred_hybrid = svm_classifier.predict(X_val_features)
hybrid_accuracy = accuracy_score(y_val, y_pred_hybrid)

# Display results
print("\n" + "="*70)
print("🎯 FINAL RESULTS")
print("="*70)
print(f"CNN-only Accuracy:     {cnn_accuracy*100:.2f}%")
print(f"CNN+SVM Hybrid Accuracy: {hybrid_accuracy*100:.2f}% {'✅' if hybrid_accuracy >= 0.90 else '⚠️'}")

if hybrid_accuracy >= 0.90:
    print("\n🎉 SUCCESS! Achieved 90%+ accuracy target!")
else:
    print(f"\n⚠️  Current accuracy: {hybrid_accuracy*100:.2f}% (Target: 90%+)")
    print("   Consider: More training data, longer training, or hyperparameter tuning")

# Detailed classification report
print("\n" + "="*70)
print("📋 Detailed Classification Report (Hybrid Model)")
print("="*70)
print(classification_report(y_val, y_pred_hybrid, target_names=config.CLASSES, digits=3))

# Per-class accuracy
print("\n📊 Per-Class Accuracy:")
for idx, class_name in enumerate(config.CLASSES):
    class_mask = y_val == idx
    if np.sum(class_mask) > 0:
        class_acc = accuracy_score(y_val[class_mask], y_pred_hybrid[class_mask])
        print(f"   {class_name:12s}: {class_acc*100:.2f}%")

# Confusion Matrix Visualization
cm = confusion_matrix(y_val, y_pred_hybrid)

plt.figure(figsize=(14, 12))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=config.CLASSES, yticklabels=config.CLASSES,
            cbar_kws={'label': 'Count'}, annot_kws={'size': 11})
plt.title(f'Confusion Matrix - CNN+SVM Hybrid Model\nOverall Accuracy: {hybrid_accuracy*100:.2f}%',
          fontsize=14, fontweight='bold')
plt.ylabel('True Label', fontsize=12)
```

```
plt.xlabel('Predicted Label', fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.savefig(os.path.join(config.DRIVE_MODEL_PATH, 'confusion_matrix.png'), dpi=300, bbox_inches='tight')
print("\n✅ Confusion matrix saved")
plt.show()

# Training History Plots
print("\n📈 Generating training history plots...")
fig, axes = plt.subplots(2, 2, figsize=(16, 12))

# Combine both training stages
all_acc = history.history['accuracy'] + history_finetune.history['accuracy']
all_val_acc = history.history['val_accuracy'] + history_finetune.history['val_accuracy']
all_loss = history.history['loss'] + history_finetune.history['loss']
all_val_loss = history.history['val_loss'] + history_finetune.history['val_loss']

# Accuracy plot
axes[0, 0].plot(all_acc, label='Training', linewidth=2)
axes[0, 0].plot(all_val_acc, label='Validation', linewidth=2)
axes[0, 0].axvline(x=len(history.history['accuracy']), color='red', linestyle='--',
                   label='Fine-tuning starts', alpha=0.7)
axes[0, 0].set_title('Model Accuracy', fontsize=12, fontweight='bold')
axes[0, 0].set_xlabel('Epoch')
axes[0, 0].set_ylabel('Accuracy')
axes[0, 0].legend()
axes[0, 0].grid(True, alpha=0.3)

# Loss plot
axes[0, 1].plot(all_loss, label='Training', linewidth=2)
axes[0, 1].plot(all_val_loss, label='Validation', linewidth=2)
axes[0, 1].axvline(x=len(history.history['loss']), color='red', linestyle='--',
                   label='Fine-tuning starts', alpha=0.7)
axes[0, 1].set_title('Model Loss', fontsize=12, fontweight='bold')
axes[0, 1].set_xlabel('Epoch')
axes[0, 1].set_ylabel('Loss')
axes[0, 1].legend()
axes[0, 1].grid(True, alpha=0.3)

# Per-class accuracy comparison
class_accuracies_cnn = []
class_accuracies_hybrid = []
for idx in range(len(config.CLASSES)):
    class_mask = y_val == idx
    if np.sum(class_mask) > 0:
        class_accuracies_cnn.append(accuracy_score(y_val[class_mask], y_pred_cnn[class_mask]))
        class_accuracies_hybrid.append(accuracy_score(y_val[class_mask], y_pred_hybrid[class_mask]))
    else:
        class_accuracies_cnn.append(0)
        class_accuracies_hybrid.append(0)

x = np.arange(len(config.CLASSES))
width = 0.35
axes[1, 0].bar(x - width/2, [acc*100 for acc in class_accuracies_cnn], width,
               label='CNN Only', alpha=0.8)
axes[1, 0].bar(x + width/2, [acc*100 for acc in class_accuracies_hybrid], width,
               label='CNN+SVM Hybrid', alpha=0.8)
axes[1, 0].set_xlabel('Class')
axes[1, 0].set_ylabel('Accuracy (%)')
axes[1, 0].set_title('Per-Class Accuracy Comparison', fontsize=12, fontweight='bold')
axes[1, 0].set_xticks(x)
axes[1, 0].set_xticklabels(config.CLASSES, rotation=45, ha='right')
axes[1, 0].legend()
axes[1, 0].grid(True, alpha=0.3, axis='y')
axes[1, 0].axhline(y=90, color='green', linestyle='--', alpha=0.5, label='90% Target')

# Model comparison summary
comparison_data = {
    'Metric': ['Accuracy', 'Precision (avg)', 'Recall (avg)', 'F1-Score (avg)'],
    'CNN Only': [
        cnn_accuracy * 100,
        0, 0, 0  # Will calculate below
    ],
    'CNN+SVM Hybrid': [
        hybrid_accuracy * 100,
        0, 0, 0  # Will calculate below
    ]
}

# Calculate additional metrics
from sklearn.metrics import precision_recall_fscore_support
prec_cnn, rec_cnn, f1_cnn,   = precision_recall_fscore_support(y_val, y_pred_cnn, average='weighted')
```

```
prec_hyb, rec_hyb, f1_hyb, _ = precision_recall_fscore_support(y_val, y_pred_hybrid, average='weighted')

comparison_data['CNN Only'][1:] = [prec_cnn*100, rec_cnn*100, f1_cnn*100]
comparison_data['CNN+SVM Hybrid'][1:] = [prec_hyb*100, rec_hyb*100, f1_hyb*100]

axes[1, 1].axis('tight')
axes[1, 1].axis('off')
table_data = [[metric, f"{cnn:.2f}%", f"{hyb:.2f}%"]
              for metric, cnn, hyb in zip(comparison_data['Metric'],
                                          comparison_data['CNN Only'],
                                          comparison_data['CNN+SVM Hybrid'])]
table = axes[1, 1].table(cellText=table_data,
                         colLabels=['Metric', 'CNN Only', 'CNN+SVM Hybrid'],
                         cellLoc='center', loc='center',
                         colWidths=[0.35, 0.25, 0.25])
table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1, 2)

# Style header
for i in range(3):
    table[(0, i)].set_facecolor('#4CAF50')
    table[(0, i)].set_text_props(weight='bold', color='white')

# Highlight best scores
for i in range(1, 5):
    cnn_val = comparison_data['CNN Only'][i-1]
    hyb_val = comparison_data['CNN+SVM Hybrid'][i-1]
    if hyb_val > cnn_val:
        table[(i, 2)].set_facecolor('#E8F5E9')
        table[(i, 2)].set_text_props(weight='bold')

axes[1, 1].set_title('Model Performance Comparison', fontsize=12, fontweight='bold', pad=20)

plt.tight_layout()
plt.savefig(os.path.join(config.DRIVE_MODEL_PATH, 'training_analysis.png'), dpi=300, bbox_inches='tight')
print("✅ Training analysis plots saved")
plt.show()
```

```
======================================================================
📊 STEP 8: Model Evaluation
======================================================================

🔍 Evaluating CNN-only model...
🔍 Evaluating CNN+SVM hybrid model...

======================================================================
🎯 FINAL RESULTS
======================================================================
CNN-only Accuracy:     90.87%
CNN+SVM Hybrid Accuracy: 90.66% ✅

🎉 SUCCESS! Achieved 90%+ accuracy target!

======================================================================
📋 Detailed Classification Report (Hybrid Model)
======================================================================
              precision    recall  f1-score   support

  blackheads      0.968     0.909     0.938        66
       cysts      0.846     0.948     0.894        58
   darkspots      0.830     0.863     0.846        51
      normal      0.976     0.984     0.980       126
     papules      0.855     0.791     0.822        67
    pustules      0.842     0.857     0.850        56
  whiteheads      0.946     0.914     0.930        58

    accuracy                          0.907       482
   macro avg      0.895     0.895     0.894       482
weighted avg      0.908     0.907     0.907       482


📊 Per-Class Accuracy:
   blackheads  : 90.91%
   cysts       : 94.83%
   darkspots   : 86.27%
```

```
# ==================== PART 13: SAVE ALL MODELS ====================
print("\n" + "="*70)
print("💾 STEP 9: Saving Models and Artifacts")
print("="*70)

# Save CNN model (fixed filenames for Keras 3.x)
print("\n📦 Saving CNN model...")
try:
    # Save full model in native Keras format
    cnn_model.save(os.path.join(config.MODEL_PATH, 'cnn_model_full.keras'))
    print("   ✅ Full model saved (local - .keras format)")
except Exception as e:
    print(f"   ⚠️ Full model save failed: {e}")

# Save weights (must end with .weights.h5 in Keras 3.x)
cnn_model.save_weights(os.path.join(config.MODEL_PATH, 'cnn_weights.weights.h5'))
print("   ✅ Model weights saved (local)")

# Save to Google Drive
cnn_model.save_weights(os.path.join(config.DRIVE_MODEL_PATH, 'cnn_weights.weights.h5'))
print("   ✅ Model weights saved (Google Drive) ✅✅✅")

# Also save full model to Drive
try:
    cnn_model.save(os.path.join(config.DRIVE_MODEL_PATH, 'cnn_model_full.keras'))
    print("   ✅ Full model saved (Google Drive) ✅✅✅")
except Exception as e:
    print(f"   ⚠️ Full model to Drive failed: {e}")

# Save model architecture as JSON
model_json = cnn_model.to_json()
with open(os.path.join(config.DRIVE_MODEL_PATH, 'model_architecture.json'), 'w') as f:
    f.write(model_json)
print("   ✅ Model architecture saved (Google Drive) ✅✅✅")

# Save SVM classifier
print("\n📦 Saving SVM classifier...")
with open(os.path.join(config.DRIVE_MODEL_PATH, 'svm_classifier.pkl'), 'wb') as f:
    pickle.dump(svm_classifier, f)
print("   ✅ SVM classifier saved (Google Drive) ✅✅✅")

# Save scaler
print("\n📦 Saving feature scaler...")
with open(os.path.join(config.DRIVE_MODEL_PATH, 'scaler.pkl'), 'wb') as f:
    pickle.dump(scaler, f)
print("   ✅ Scaler saved (Google Drive) ✅✅✅")
```

```
# Save class names
print("\n📦 Saving class names...")
with open(os.path.join(config.DRIVE_MODEL_PATH, 'class_names.pkl'), 'wb') as f:
    pickle.dump(config.CLASSES, f)
print("   ✅ Class names saved (Google Drive) ✅✅✅")

# Save comprehensive configuration
print("\n📦 Saving configuration...")
config_dict = {
    'classes': config.CLASSES,
    'num_classes': len(config.CLASSES),
    'img_size': config.IMG_SIZE,
    'batch_size': config.BATCH_SIZE,
    'epochs': config.EPOCHS,
    'learning_rate': config.LEARNING_RATE,
    'validation_split': config.VALIDATION_SPLIT,
    'target_samples': config.TARGET_SAMPLES,
    'cnn_accuracy': float(cnn_accuracy),
    'hybrid_accuracy': float(hybrid_accuracy),
    'training_samples': len(X_train),
    'validation_samples': len(X_val),
    'feature_dim': X_train_features.shape[1]
}
with open(os.path.join(config.DRIVE_MODEL_PATH, 'config.pkl'), 'wb') as f:
    pickle.dump(config_dict, f)
print("   ✅ Configuration saved (Google Drive) ✅✅✅")

# Save training history
print("\n📦 Saving training history...")
history_dict = {
    'initial_training': {
        'accuracy': history.history['accuracy'],
        'val_accuracy': history.history['val_accuracy'],
        'loss': history.history['loss'],
        'val_loss': history.history['val_loss']
    },
    'fine_tuning': {
        'accuracy': history_finetune.history['accuracy'],
        'val_accuracy': history_finetune.history['val_accuracy'],
        'loss': history_finetune.history['loss'],
        'val_loss': history_finetune.history['val_loss']
    }
}
with open(os.path.join(config.DRIVE_MODEL_PATH, 'training_history.pkl'), 'wb') as f:
    pickle.dump(history_dict, f)
print("   ✅ Training history saved (Google Drive) ✅✅✅")

print("\n" + "="*70)
print("🎉 ALL FILES SAVED TO GOOGLE DRIVE!")
print(f"📁 Location: {config.DRIVE_MODEL_PATH}")
print("="*70)

# ==================== PART 14: VERIFICATION & SUMMARY ====================
print("\n" + "="*70)
print("✅ VERIFICATION: Checking Saved Files")
print("="*70)

required_files = {
    'cnn_weights.weights.h5': 'CNN Model Weights',
    'cnn_model_full.keras': 'Full CNN Model',
    'model_architecture.json': 'Model Architecture',
    'svm_classifier.pkl': 'SVM Classifier',
    'scaler.pkl': 'Feature Scaler',
    'class_names.pkl': 'Class Names',
    'config.pkl': 'Configuration',
    'training_history.pkl': 'Training History',
    'confusion_matrix.png': 'Confusion Matrix',
    'training_analysis.png': 'Training Analysis'
}

all_files_present = True
total_size_mb = 0

for filename, description in required_files.items():
    filepath = os.path.join(config.DRIVE_MODEL_PATH, filename)
    if os.path.exists(filepath):
        size_mb = os.path.getsize(filepath) / (1024*1024)
        total_size_mb += size_mb
        print(f"   ✅ {description:25s} ({size_mb:.2f} MB)")
    else:
        print(f"   ❌ {description:25s} MISSING!")
```

```
        all_files_present = False

print(f"\n📊 Total size: {total_size_mb:.2f} MB")

# ==================== FINAL SUMMARY ====================
print("\n" + "="*70)
print("🎉 TRAINING COMPLETED SUCCESSFULLY!")
print("="*70)

print(f"\n📊 FINAL RESULTS:")
print(f"   Total Classes:          {len(config.CLASSES)}")
print(f"   Training Samples:       {len(X_train)}")
print(f"   Validation Samples:     {len(X_val)}")
print(f"   CNN Accuracy:           {cnn_accuracy*100:.2f}%")
print(f"   Hybrid Accuracy:        {hybrid_accuracy*100:.2f}%")
print(f"   Improvement:            +{(hybrid_accuracy-cnn_accuracy)*100:.2f}%")

if hybrid_accuracy >= 0.90:
    print(f"\n✅ SUCCESS! Achieved {hybrid_accuracy*100:.2f}% accuracy (Target: 90%+)")
else:
    print(f"\n⚠️  Current: {hybrid_accuracy*100:.2f}% (Target: 90%)")
    print("   Suggestions to improve:")
    print("   1. Add more training images (especially for low-performing classes)")
    print("   2. Increase TARGET_SAMPLES to 500-600")
    print("   3. Train for more epochs (100-120)")
    print("   4. Try different SVM parameters (C=200, gamma='auto')")

print(f"\n📁 MODEL FILES LOCATION:")
print(f"   Google Drive: {config.DRIVE_MODEL_PATH}")
print(f"   Local: {config.MODEL_PATH}")

if all_files_present:
    print("\n✅ All required files saved successfully!")
else:
    print("\n⚠️  Some files are missing. Check the errors above.")

print("\n📥 NEXT STEPS:")
print("   1. Go to Google Drive and navigate to:")
print(f"      {config.DRIVE_MODEL_PATH}")
print("   2. Download ALL files to your local VS Code project")
print("   3. Use these files in your inference script")
print("   4. Deploy with Flask/Streamlit/Gradio")

print("\n🚀 FILES TO DOWNLOAD:")
for i, (filename, description) in enumerate(required_files.items(), 1):
    print(f"   {i}. {filename:30s} - {description}")

print("\n" + "="*70)
print("💡 TIP: Right-click the folder in Google Drive → Download")
print("="*70)

print("\n🎯 Your acne detection model is ready for deployment!")
print("🔬 Model can classify: " + ", ".join(config.CLASSES))
print("\n✨ Happy coding!")
```

```
✅ All required files saved successfully!

📥 NEXT STEPS:
   1. Go to Google Drive and navigate to:
      /content/drive/MyDrive/acne_models_final
   2. Download ALL files to your local VS Code project
   3. Use these files in your inference script
   4. Deploy with Flask/Streamlit/Gradio

🚀 FILES TO DOWNLOAD:
   1. cnn_weights.weights.h5         - CNN Model Weights
   2. cnn_model_full.keras           - Full CNN Model
   3. model_architecture.json        - Model Architecture
   4. svm_classifier.pkl             - SVM Classifier
   5. scaler.pkl                     - Feature Scaler
   6. class_names.pkl                - Class Names
   7. config.pkl                     - Configuration
   8. training_history.pkl           - Training History
   9. confusion_matrix.png           - Confusion Matrix
   10. training_analysis.png          - Training Analysis

======================================================================
💡 TIP: Right-click the folder in Google Drive → Download
======================================================================

🎯 Your acne detection model is ready for deployment!
🔬 Model can classify: blackheads, cysts, darkspots, normal, papules, pustules, whiteheads

✨ Happy coding!
```

```
# ============================================================
# 🧴 AI Acne Detection & Analysis — Professional Skincare Edition (Bounding Box Version)
# Clean, modern, dermatologist-grade UI
# ============================================================

print("🔧 Installing dependencies...")
!pip install gradio opencv-python pillow tensorflow scikit-learn matplotlib --quiet

import gradio as gr
import numpy as np
import cv2
from PIL import Image
import pickle
import tensorflow as tf
from tensorflow import keras
import os

# ============================================================
# 🧠 Load Models
# ============================================================
MODEL_PATH = '/content/drive/MyDrive/acne_models_final'

print("\n📦 Loading trained models...")
try:
    cnn_model = keras.models.load_model(os.path.join(MODEL_PATH, 'cnn_model_full.keras'))
    print("✅ CNN model loaded successfully")
except Exception:
    from tensorflow.keras import layers, models
    from tensorflow.keras.applications import MobileNetV2
    base_model = MobileNetV2(input_shape=(224, 224, 3), include_top=False, weights='imagenet')
    base_model.trainable = False
    cnn_model = models.Sequential([
        base_model,
        layers.GlobalAveragePooling2D(),
        layers.Dense(128, activation='relu', name='feature_layer'),
        layers.Dense(7, activation='softmax')
    ])
    cnn_model.load_weights(os.path.join(MODEL_PATH, 'cnn_weights.weights.h5'))
    print("✅ CNN model weights loaded")

with open(os.path.join(MODEL_PATH, 'svm_classifier.pkl'), 'rb') as f:
    svm_classifier = pickle.load(f)
with open(os.path.join(MODEL_PATH, 'scaler.pkl'), 'rb') as f:
    scaler = pickle.load(f)
with open(os.path.join(MODEL_PATH, 'class_names.pkl'), 'rb') as f:
    class_names = pickle.load(f)

print("✅ SVM classifier, scaler, and class labels loaded")

# ============================================================
# 📦 Acne Info
# ============================================================
ACNE_INFO = {
    "normal": {"name": "Clear Skin", "severity": "None", "description": "No active acne detected."},
    "blackheads": {"name": "Blackheads", "severity": "Mild", "description": "Small open comedones."},
```

```
    "whiteheads": {"name": "Whiteheads", "severity": "Mild", "description": "Closed comedones with trapped sebum."},
    "papules": {"name": "Papules", "severity": "Moderate", "description": "Inflamed red bumps without pus."},
    "pustules": {"name": "Pustules", "severity": "Moderate", "description": "Inflamed lesions with visible pus."},
    "nodules": {"name": "Nodules", "severity": "Severe", "description": "Large, painful deep bumps."},
    "cysts": {"name": "Cystic Acne", "severity": "Very Severe", "description": "Large pus-filled lesions deep under skin."},
    "darkspots": {"name": "Dark Spots", "severity": "Post-Acne", "description": "Post-inflammatory hyperpigmentation."}
}

# ============================================================
# 🔬 Image Preprocessing
# ============================================================
def preprocess_image(img, img_size=(224, 224)):
    if isinstance(img, Image.Image):
        img = np.array(img)
    if img.shape[-1] == 4:
        img = cv2.cvtColor(img, cv2.COLOR_RGBA2RGB)
    img = cv2.resize(img, img_size)
    img = img.astype(np.float32) / 255.0
    return np.expand_dims(img, axis=0)

# ============================================================
# 🩺 Acne Prediction
# ============================================================
def predict_acne(image):
    if image is None:
        return None, "⚠️ Please upload an image first.", ""

    try:
        original = np.array(image)
        if len(original.shape) == 2:
            original = cv2.cvtColor(original, cv2.COLOR_GRAY2RGB)
        elif original.shape[-1] == 4:
            original = cv2.cvtColor(original, cv2.COLOR_RGBA2RGB)

        processed = preprocess_image(image)
        x = processed
        for layer in cnn_model.layers:
            x = layer(x, training=False)
            if layer.name == "feature_layer":
                features = x.numpy()
                break

        features = scaler.transform(features)
        prediction = svm_classifier.predict(features)[0]
        probabilities = svm_classifier.predict_proba(features)[0]
        predicted_class = class_names[prediction]
        confidence = probabilities[prediction] * 100

        annotated = create_bounding_box_overlay(original, predicted_class, confidence)

        analysis = f"""
### 🔍 Detected: **{ACNE_INFO[predicted_class]['name']}**
**Severity:** {ACNE_INFO[predicted_class]['severity']}
**Confidence:** {confidence:.1f}%

{ACNE_INFO[predicted_class]['description']}
"""

        recs = """
### 💊 Recommendations
- Maintain a gentle daily skincare routine
- Avoid picking or squeezing lesions
- Use non-comedogenic moisturizers
- Always apply SPF 30+ sunscreen
- Seek dermatologist care for persistent or painful acne

⚠️ **Note:** This AI analysis is for educational purposes only and not a medical diagnosis.
"""
        return annotated, analysis, recs

    except Exception as e:
        return None, f"❌ Error: {str(e)}", ""

# ============================================================
# 📦 Bounding Box Overlay
# ============================================================
def create_bounding_box_overlay(img, label, conf):
    img_bgr = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    orig = img_bgr.copy()

    # Detect reddish/acne-like regions
    hsv = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2HSV)
    lower_red1 = np.array([0, 48, 60])
```

```
        upper_red1 = np.array([15, 255, 255])
        lower_red2 = np.array([160, 48, 60])
        upper_red2 = np.array([179, 255, 255])
        mask1 = cv2.inRange(hsv, lower_red1, upper_red1)
        mask2 = cv2.inRange(hsv, lower_red2, upper_red2)
        mask = cv2.bitwise_or(mask1, mask2)

        # Denoise
        mask = cv2.medianBlur(mask, 7)
        mask = cv2.dilate(mask, None, iterations=2)
        mask = cv2.erode(mask, None, iterations=1)

        contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

        color_map = {
            "normal": (76, 175, 80),
            "blackheads": (0, 180, 255),
            "whiteheads": (255, 200, 80),
            "papules": (0, 102, 255),
            "pustules": (255, 0, 120),
            "nodules": (50, 0, 180),
            "cysts": (0, 0, 180),
            "darkspots": (100, 60, 50),
        }
        color = color_map.get(label, (0, 255, 0))

        min_area = 100
        for c in contours:
            if cv2.contourArea(c) > min_area:
                x, y, w, h = cv2.boundingRect(c)
                cv2.rectangle(orig, (x, y), (x + w, y + h), color, 2)

        cv2.rectangle(orig, (0, 0), (img_bgr.shape[1], 60), (0, 0, 0), -1)
        cv2.putText(orig, f"{ACNE_INFO[label]['name']} ({conf:.1f}%)", (15, 40),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 255, 255), 2)

        annotated = cv2.cvtColor(orig, cv2.COLOR_BGR2RGB)
        return annotated

    # ============================================================
    # 💅 CSS Styling
    # ============================================================
    custom_css = """
    body, .gradio-container {
        background: #faf9f7 !important;
        font-family: 'Inter', 'Poppins', sans-serif !important;
        color: #1a1a1a !important;
    }
    #title { text-align: center; font-size: 2.6em; font-weight: 700; color: #222; }
    #subtitle { text-align: center; color: #555; font-size: 1.1em; margin-bottom: 2.4em; }
    .input-image, .output-image {
        border-radius: 16px; background: #fff;
        box-shadow: 0 4px 14px rgba(0,0,0,0.08);
    }
    #analyze-btn {
        background: linear-gradient(90deg, #9be7e1 0%, #fbcfe8 100%);
        color: #1f1f1f; border: none; border-radius: 28px;
        padding: 14px 36px; font-weight: 600; font-size: 1.05em;
        transition: all 0.25s ease;
    }
    #analyze-btn:hover { transform: translateY(-2px);
        box-shadow: 0 6px 14px rgba(0,0,0,0.15);
    }
    .markdown-text {
        background: #fff; border-radius: 14px;
        padding: 22px 26px; box-shadow: 0 3px 10px rgba(0,0,0,0.07);
        font-size: 0.97em; color: #222;
    }
    #warning-box {
        background: #fff6e5; border-left: 5px solid #f5a623;
        padding: 20px 25px; border-radius: 10px;
        color: #4a3b21; font-size: 0.95em;
        box-shadow: 0 2px 10px rgba(0,0,0,0.05); margin-top: 40px;
    }
    #footer {
        text-align: center; font-size: 0.9em;
        color: #777; margin-top: 50px;
        padding: 30px 0; border-top: 1px solid #e2e2e2;
    }
    """

    # ============================================================
    # 🚀 Gradio Interface
```

```
# 🚀 Gradio Interface
# ============================================================
with gr.Blocks(css=custom_css, title="AI Skincare Diagnostic") as app:
    gr.HTML("""
        <h1 id='title'>AI Skincare Diagnostic</h1>
        <p id='subtitle'>Smart, dermatologist-inspired analysis for acne and skin conditions</p>
    """)

    with gr.Row(equal_height=True):
        with gr.Column(scale=1):
            gr.Markdown("#### Upload or Capture Image")
            input_image = gr.Image(type="pil", sources=["upload", "webcam", "clipboard"], height=400, elem_classes="input-image"
            analyze_btn = gr.Button("✨ Analyze Skin", elem_id="analyze-btn")
            gr.Markdown("<div id='tips'><b>Tips:</b> Use natural lighting, no makeup, and center affected area.</div>")

        with gr.Column(scale=1):
            gr.Markdown("#### Analysis Result")
            output_image = gr.Image(type="numpy", height=400, elem_classes="output-image")

    gr.HTML("<hr>")

    with gr.Row():
        with gr.Column(scale=1):
            gr.Markdown("#### Skin Condition Analysis")
            analysis_output = gr.Markdown(elem_classes="markdown-text")
        with gr.Column(scale=1):
            gr.Markdown("#### Care & Treatment Recommendations")
            recommendations_output = gr.Markdown(elem_classes="markdown-text")

    gr.HTML("""
        <div id='warning-box'>
            <b>⚠️ Important:</b> This tool is for educational insight only.
            Consult a board-certified dermatologist for professional evaluation.
        </div>
    """)

    gr.HTML("""
        <div id='footer'>
            © 2025 SkinAI — Developed with deep learning and dermatology references.<br>
            Always consult a licensed dermatologist for personalized care.
        </div>
    """)

    analyze_btn.click(fn=predict_acne, inputs=[input_image], outputs=[output_image, analysis_output, recommendations_output])

# ============================================================
# 🌎 Launch App
# ============================================================
print("\n🚀 Launching the professional skincare app...")
app.launch(share=True, server_name="0.0.0.0", server_port=7861)
```