Loading the Dataset

```
# FIXED SOLUTION: Exclude nested aptos-augmented-images folder

import os
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.applications import ResNet50, ResNet152, DenseNet121
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import classification_report, confusion_matrix, precision_score
from sklearn.utils.class_weight import compute_class_weight
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
import shutil
import kagglehub

# Set random seeds
np.random.seed(42)
tf.random.set_seed(42)

# Configuration
IMG_HEIGHT, IMG_WIDTH = 224, 224
BATCH_SIZE = 32
NUM_CLASSES = 5
EPOCHS = 30

# Dataset path - go directly to the inner directory with class folders
path = kagglehub.dataset_download("aitude/aptos-augmented-images")
actual_path = os.path.join(path, "aptos-augmented-images")

print("STEP 1: Fixing dataset structure...")
print(f"Using directory: {actual_path}")

# Create a clean directory structure excluding the nested folder
clean_path = "/tmp/clean_aptos"
if os.path.exists(clean_path):
    shutil.rmtree(clean_path)
os.makedirs(clean_path, exist_ok=True)

print("\nSTEP 2: Creating clean class structure...")
# Copy only the 5 diabetic retinopathy class folders (0, 1, 2, 3, 4)
class_names = ['0', '1', '2', '3', '4']
total_images = 0

for class_name in class_names:
    src_class_path = os.path.join(actual_path, class_name)
    dst_class_path = os.path.join(clean_path, class_name)

    if os.path.exists(src_class_path) and os.path.isdir(src_class_path):
        # Create destination class directory
        os.makedirs(dst_class_path, exist_ok=True)

        # Copy all images from source to destination
        images = [f for f in os.listdir(src_class_path)
                  if f.lower().endswith(('.png', '.jpg', '.jpeg'))]

        for img in images:
            src_img = os.path.join(src_class_path, img)
            dst_img = os.path.join(dst_class_path, img)
            shutil.copy2(src_img, dst_img)

        print(f"  Class {class_name}: {len(images)} images copied")
        total_images += len(images)
    else:
        print(f" Class {class_name}: folder not found")

print(f"\nTotal images processed: {total_images}")
print("Excluded: aptos-augmented-images nested folder")

# Verify clean structure
print("\nSTEP 3: Verifying clean structure...")
for class_name in class_names:
    class_path = os.path.join(clean_path, class_name)
    if os.path.exists(class_path):
        count = len(os.listdir(class_path))
        print(f"  Class {class_name}: {count} images")

# Create data generators using the clean structure
print("\nSTEP 4: Creating data generators...")
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    zoom_range=0.2,
    shear_range=0.2,
    fill_mode='nearest',
    validation_split=0.2
)

train_generator = train_datagen.flow_from_directory(
    clean_path,  # Use clean path without nested folder
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='training',
    shuffle=True
)

validation_generator = train_datagen.flow_from_directory(
    clean_path,  # Use clean path without nested folder
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='validation',
    shuffle=False
)

# Verify the fix
print("\nSTEP 5: Verification Results...")
print(f"Train: {train_generator.samples} images, {train_generator.num_classes} classes")
print(f"Validation: {validation_generator.samples} images, {validation_generator.num_classes} classes")
print(f"Class indices: {train_generator.class_indices}")

# Test batch shapes
x_batch, y_batch = next(train_generator)
print(f"\nBatch shapes:")
print(f"Input: {x_batch.shape}")
print(f"Labels: {y_batch.shape}")

if y_batch.shape[1] == NUM_CLASSES and train_generator.num_classes == NUM_CLASSES:
    print("\n SUCCESS: Perfect! 5 classes with correct label shape!")
    print(" Ready for categorical_crossentropy loss function")
    use_categorical = True
else:
    print("\n Still have issues with class count")
    use_categorical = False

# Calculate class weights for balanced training
class_labels = train_generator.classes
unique_classes = np.unique(class_labels)
class_weights = compute_class_weight('balanced', classes=unique_classes, y=class_labels)
class_weight_dict = dict(zip(unique_classes, class_weights))

print(f"\nClass distribution in training:")
for class_idx in range(NUM_CLASSES):
    count = np.sum(class_labels == class_idx)
    print(f"  Class {class_idx}: {count} samples")

print(f"\nClass weights: {class_weight_dict}")

# Setup callbacks
early_stopping = EarlyStopping(patience=10, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(factor=0.2, patience=5, min_lr=0.0001)

print("\n🎯 DATASET READY FOR MODEL TRAINING!")
```

```
Downloading from https://www.kaggle.com/api/v1/datasets/download/aitude/aptos-augmented-images?dataset_version_number=1...
100%|██████████| 2.77G/2.77G [00:31<00:00, 92.9MB/s]Extracting files...

STEP 1: Fixing dataset structure...
Using directory: /root/.cache/kagglehub/datasets/aitude/aptos-augmented-images/versions/1/aptos-augmented-images

STEP 2: Creating clean class structure...
  Class 0: 2000 images copied
  Class 1: 2000 images copied
  Class 2: 2000 images copied
  Class 3: 2000 images copied
  Class 4: 2000 images copied

Total images processed: 10000
Excluded: aptos-augmented-images nested folder

STEP 3: Verifying clean structure...
  Class 0: 2000 images
  Class 1: 2000 images
  Class 2: 2000 images
  Class 3: 2000 images
  Class 4: 2000 images

STEP 4: Creating data generators...
Found 8000 images belonging to 5 classes.
Found 2000 images belonging to 5 classes.

STEP 5: Verification Results...
Train: 8000 images, 5 classes
Validation: 2000 images, 5 classes
Class indices: {'0': 0, '1': 1, '2': 2, '3': 3, '4': 4}

Batch shapes:
Input: (32, 224, 224, 3)
Labels: (32, 5)

 SUCCESS: Perfect! 5 classes with correct label shape!
 Ready for categorical_crossentropy loss function

Class distribution in training:
  Class 0: 1600 samples
  Class 1: 1600 samples
  Class 2: 1600 samples
  Class 3: 1600 samples
  Class 4: 1600 samples

Class weights: {np.int32(0): np.float64(1.0), np.int32(1): np.float64(1.0), np.int32(2): np.float64(1.0), np.int32(3): np.float

🎯 DATASET READY FOR MODEL TRAINING!
```

ResNet50 Model

```python
# CELL 1: ResNet50 Direct Implementation
print("="*60)
print("RESNET50 MODEL ON APTOS DATASET")
print("="*60)

# Load ResNet50 pretrained model
resnet50_base = ResNet50(
    weights='imagenet',
    include_top=False,  # Remove final classification layer
    input_shape=(IMG_HEIGHT, IMG_WIDTH, 3)
)

# Freeze the base model
resnet50_base.trainable = False

# Add custom classification head
resnet50_model = keras.Sequential([
    resnet50_base,
    keras.layers.GlobalAveragePooling2D(),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(NUM_CLASSES, activation='softmax')
], name='ResNet50_APTOS')

# Compile model
resnet50_model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy', 'precision', 'recall']
)

print(f"ResNet50 Model: {resnet50_model.count_params():,} parameters")
print("Base model frozen, training classification head only")

# Train ResNet50
print("\nTraining ResNet50...")
resnet50_history = resnet50_model.fit(
    train_generator,
    epochs=EPOCHS,
    validation_data=validation_generator,
    class_weight=class_weight_dict,
    callbacks=[early_stopping, reduce_lr],
    verbose=1
)

# Fine-tune ResNet50
print("\nFine-tuning ResNet50 (unfreezing base model)...")
resnet50_base.trainable = True
resnet50_model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.0001),
    loss='categorical_crossentropy',
    metrics=['accuracy', 'precision', 'recall']
)

resnet50_history_fine = resnet50_model.fit(
    train_generator,
    epochs=20,
    validation_data=validation_generator,
    class_weight=class_weight_dict,
    callbacks=[early_stopping, reduce_lr],
    verbose=1
)

# Get predictions and precision matrix
resnet50_predictions = resnet50_model.predict(validation_generator)
resnet50_pred_classes = np.argmax(resnet50_predictions, axis=1)

# Get true classes
resnet50_true_classes = []
validation_generator.reset()
for i in range(len(validation_generator)):
    _, y_batch = validation_generator[i]
    resnet50_true_classes.extend(np.argmax(y_batch, axis=1))

# Precision matrix (confusion matrix)
resnet50_cm = confusion_matrix(resnet50_true_classes, resnet50_pred_classes)
resnet50_precision = precision_score(resnet50_true_classes, resnet50_pred_classes, average=None, zero_division=0)

print("\nResNet50 Results:")
print("Precision Matrix (Confusion Matrix):")
print(resnet50_cm)
print(f"Class-wise Precision: {resnet50_precision}")
print(f"Mean Precision: {np.mean(resnet50_precision):.4f}")
```

Show hidden output

```
from sklearn.metrics import accuracy_score

resnet50_accuracy = accuracy_score(resnet50_true_classes, resnet50_pred_classes)
print(f"Accuracy Score: {resnet50_accuracy:.4f}")
```

```
Accuracy Score: 0.7820
```

```
# Accuracy from confusion matrix
correct_predictions = np.trace(resnet50_cm)   # sum of diagonal values
total_predictions = np.sum(resnet50_cm)
resnet50_accuracy = correct_predictions / total_predictions

print(f"Accuracy from Confusion Matrix: {resnet50_accuracy:.4f}")
```

```
Accuracy from Confusion Matrix: 0.7820
```

ResNet152 Model

```
# CELL 2: ResNet152 Direct Implementation
print("\n" + "="*60)
print("RESNET152 MODEL ON APTOS DATASET")
print("="*60)

# Load ResNet152 pretrained model
resnet152_base = ResNet152(
    weights='imagenet',
    include_top=False,
    input_shape=(IMG_HEIGHT, IMG_WIDTH, 3)
)

# Freeze the base model
```

```
resnet152_base.trainable = False

# Add classification head
resnet152_model = keras.Sequential([
    resnet152_base,
    keras.layers.GlobalAveragePooling2D(),
    keras.layers.Dropout(0.3),
    keras.layers.Dense(256, activation='relu'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(NUM_CLASSES, activation='softmax')
], name='ResNet152_APTOS')

# Compile and train
resnet152_model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy', 'precision', 'recall']
)

print(f"ResNet152 Model: {resnet152_model.count_params():,} parameters")

# Train ResNet152
resnet152_history = resnet152_model.fit(
    train_generator,
    epochs=70,
    validation_data=validation_generator,
    class_weight=class_weight_dict,
    callbacks=[early_stopping, reduce_lr],
    verbose=1
)

# Fine-tune ResNet152
resnet152_base.trainable = True
resnet152_model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.00005),
    loss='categorical_crossentropy',
    metrics=['accuracy', 'precision', 'recall']
)

resnet152_history_fine = resnet152_model.fit(
    train_generator,
    epochs=15,
    validation_data=validation_generator,
    class_weight=class_weight_dict,
    callbacks=[early_stopping, reduce_lr],
    verbose=1
)

# Calculate precision matrix
```

Show hidden output

```
from sklearn.metrics import accuracy_score
val_loss, val_acc, val_precision, val_recall = resnet152_model.evaluate(validation_generator, verbose=1)
print(f"Validation Accuracy (Keras evaluate): {val_acc:.4f}")
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 24s 381ms/step - accuracy: 0.8840 - loss: 0.3535 - precision: 0.8913 - recall: 0.8740
Validation Accuracy (Keras evaluate): 0.8215
```

DenseNet Model

```
# CELL 3: DenseNet121 Direct Implementation
print("\n" + "="*60)
print("DENSENET121 MODEL ON APTOS DATASET")
print("="*60)

# Load DenseNet121 pretrained model
densenet121_base = DenseNet121(
    weights='imagenet',
    include_top=False,
    input_shape=(IMG_HEIGHT, IMG_WIDTH, 3)
)

# Freeze the base model
densenet121_base.trainable = False

# Add classification head
```

```
densenet121_model = keras.Sequential([
    densenet121_base,
    keras.layers.GlobalAveragePooling2D(),
    keras.layers.Dropout(0.3),
    keras.layers.Dense(512, activation='relu'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(NUM_CLASSES, activation='softmax')
], name='DenseNet121_APTOS')

# Compile and train
densenet121_model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy', 'precision', 'recall']
)

print(f"DenseNet121 Model: {densenet121_model.count_params():,} parameters")

# Train DenseNet121
densenet121_history = densenet121_model.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator,
    class_weight=class_weight_dict,
    callbacks=[early_stopping, reduce_lr],
    verbose=1
)

# Fine-tune DenseNet121
densenet121_base.trainable = True
densenet121_model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.0001),
    loss='categorical_crossentropy',
    metrics=['accuracy', 'precision', 'recall']
)

densenet121_history_fine = densenet121_model.fit(
    train_generator,
    epochs=20,
    validation_data=validation_generator,
    class_weight=class_weight_dict,
    callbacks=[early_stopping, reduce_lr],
    verbose=1
)

# Calculate precision matrix
# densenet121_predictions = densenet121_model.predict(validation_generator)
# densenet121_pred_classes = np.argmax(densenet121_predictions, axis=1)
# densenet121_cm = confusion_matrix(resnet50_true_classes, densenet121_pred_classes)
# densenet121_precision = precision_score(resnet50_true_classes, densenet121_pred_classes, average=None, zero_division=0)

# print("\nDenseNet121 Results:")
# print("Precision Matrix (Confusion Matrix):")
# print(densenet121_cm)
# print(f"Class-wise Precision: {densenet121_precision}")
# print(f"Mean Precision: {np.mean(densenet121_precision):.4f}")

# Save all models
# resnet50_model.save('resnet50_aptos_model.h5')
# resnet152_model.save('resnet152_aptos_model.h5')


print("\n✅ All models trained and saved successfully!")
```

```
densenet121_model.save('densenet121_aptos_model.h5')

import numpy as np
# correct_predictions = np.trace(densenet121_cm)   # diagonal sum
# total_predictions = np.sum(densenet121_cm)
# densenet121_accuracy_cm = correct_predictions / total_predictions
# print(f"Accuracy from Confusion Matrix: {densenet121_accuracy_cm:.4f}")

# ✅ Accuracy directly from Keras evaluate (alternative check)
val_loss, val_acc, val_precision, val_recall = densenet121_model.evaluate(validation_generator, verbose=1)
print(f"Validation Accuracy (Keras evaluate): {val_acc:.4f}")
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 369ms/step - accuracy: 0.8968 - loss: 0.3153 - precision: 0.9019 - recall: 0.8923
Validation Accuracy (Keras evaluate): 0.8590
```