We used KaggleHub to download the Conjunctivitis Dataset from Kaggle. After downloading, we printed the dataset path to confirm that it was stored correctly. This dataset includes images of both healthy and infected eyes. We explored the folders to make sure the data was organized and ready for processing. We imported straight from Kaggle using the API and used 2 test images for the dataset.

```
import kagglehub
# Download latest version
path = kagglehub.dataset_download("alisofiya/conjunctivitis")
print("Path to dataset files:", path)
```

```
Using Colab cache for faster access to the 'conjunctivitis' dataset.
Path to dataset files: /kaggle/input/conjunctivitis
```

Step 1 Install Packages and Imports

```
# Cell 1 — install & imports
# ------------------------------------------------
# Install required packages (quiet mode, safe if already installed)
!pip install -q kagglehub tensorflow matplotlib scikit-learn pandas pillow

# ------------------------------------------------
# Core imports
import os, glob, random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pathlib import Path
from PIL import Image

# TensorFlow / Keras
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

# ------------------------------------------------
# Reproducibility
SEED = 42
random.seed(SEED)
np.random.seed(SEED)
tf.random.set_seed(SEED)

print("TensorFlow version:", tf.__version__)
```

```
TensorFlow version: 2.20.0
```

Step 2 Ensure dataset path is available

This cell checks whether the path exist and downloads if needed. It then finds image files recursively under that path and infers labels from parent folders

```
# Cell 2 — locate / download dataset and build a DataFrame (image_path, label)
from pathlib import Path
import pandas as pd
import kagglehub

# ------------------------------------------------
# Try to reuse a previously-downloaded path; otherwise download
try:
    path                    # <-- just check if the variable exists
    print("Using existing `path` variable:", path)
except NameError:
    print("`path` not found — downloading dataset via kagglehub...")
    path = kagglehub.dataset_download("alisofiya/conjunctivitis")
    print("Downloaded dataset to:", path)

# ------------------------------------------------
# Convert to Path object
dataset_root = Path(path)
```

```
# ------------------------------------------------
# Find **all** image files (jpg + jpeg + png - the dataset contains a few .png)
img_files = (
    list(dataset_root.rglob("*.jpg")) +
    list(dataset_root.rglob("*.jpeg")) +
    list(dataset_root.rglob("*.png"))) # Corrected variable name here
```

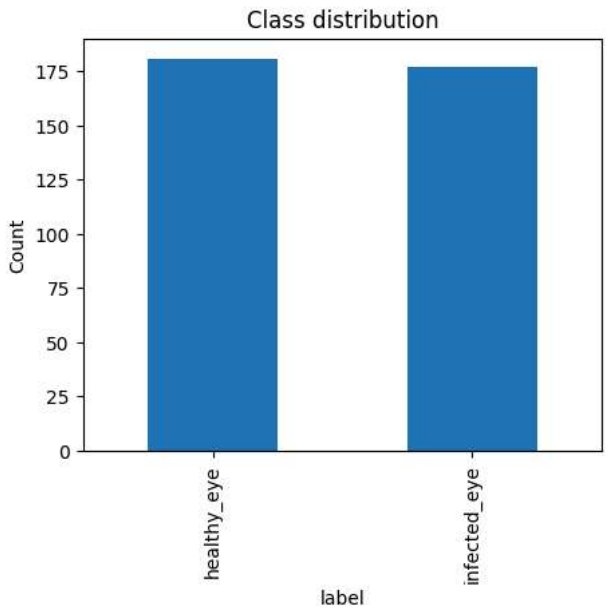
```
Using existing `path` variable: /kaggle/input/conjunctivitis
```

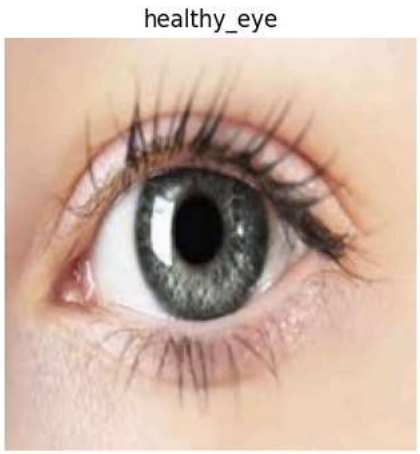
Step 3 Quick EDA:

Class counts and sample images shows how many images per class and some example images.

```
# Cell 3 — EDA: class distribution + sample images

# Create DataFrame from img_files
rows = [[str(p), p.parent.name] for p in img_files]
df = pd.DataFrame(rows, columns=["image_path", "label"])
df = df.sample(frac=1, random_state=SEED).reset_index(drop=True)

counts = df['label'].value_counts()
print("Class distribution:\n", counts)
# bar plot
plt.figure(figsize=(5,4))
counts.plot(kind='bar')
plt.title("Class distribution")
plt.ylabel("Count")
plt.show()
# show up to 6 sample images (1 per class if multiple)
plt.figure(figsize=(12,4))
unique_labels = df['label'].unique()
for i, lab in enumerate(unique_labels[:6]):
    sample = df[df['label']==lab].iloc[0]['image_path']
    img = Image.open(sample).convert('RGB')
    plt.subplot(1, min(6, len(unique_labels)), i+1)
    plt.imshow(img.resize((200,200)))
    plt.title(lab)
    plt.axis('off')
plt.show()
```

```
Class distribution:
 label
healthy_eye     181
infected_eye    177
Name: count, dtype: int64
```





Step 4 Split into train /val/ test (80/10/10) We create splits keeping class balance (stratify)

```
# Cell 4 — train/val/test split
train_df, temp_df = train_test_split(df, test_size=0.2, stratify=df['label'], random_state=SEED)
val_df, test_df = train_test_split(temp_df, test_size=0.5, stratify=temp_df['label'], random_state=SEED)
print("Counts -> Train:", len(train_df), "Val:", len(val_df), "Test:", len(test_df))
```

```
Counts -> Train: 286 Val: 36 Test: 36
```

Step 5

Create ImageDataGenerators (with simple augmentation for train)

We use flow_from_dataframe with directory=None and x_col containing full paths.class_mode='binary' if there are exactly 2 labels; otherwise 'categorical' for multi-class. The code auto-detects whether binary or multi-class.

```
# Cell 5 — prepare generators
IMG_SIZE = (128, 128) # small for fast prototyping
BATCH_SIZE = 16
# determine if binary
unique_labels = sorted(df['label'].unique())
is_binary = (len(unique_labels) == 2)
print("Binary classification?" , is_binary, "Labels:", unique_labels)
# training augmentation
train_datagen = ImageDataGenerator(
```

```
rescale=1./255,
rotation_range=10,
width_shift_range=0.05,
height_shift_range=0.05,
zoom_range=0.05,
horizontal_flip=True
)
# validation & test just rescale
test_val_datagen = ImageDataGenerator(rescale=1./255)
# flow_from_dataframe expects column names and directory argument. If x_col has full paths use directory=None
if is_binary:
    class_mode = 'binary'
else:
    class_mode = 'categorical'

# Convert image_path column to string type
train_df['image_path'] = train_df['image_path'].astype(str)
val_df['image_path'] = val_df['image_path'].astype(str)
test_df['image_path'] = test_df['image_path'].astype(str)

train_gen = train_datagen.flow_from_dataframe(
train_df,
x_col='image_path',
y_col='label',
target_size=IMG_SIZE,
batch_size=BATCH_SIZE,
class_mode=class_mode,
shuffle=True,
directory=None
)
val_gen = test_val_datagen.flow_from_dataframe(
val_df,
x_col='image_path',
y_col='label',
target_size=IMG_SIZE,
batch_size=BATCH_SIZE,
class_mode=class_mode,
shuffle=False,
directory=None
)
test_gen = test_val_datagen.flow_from_dataframe(
test_df,
x_col='image_path',
y_col='label',
target_size=IMG_SIZE,
batch_size=BATCH_SIZE,
class_mode=class_mode,
shuffle=False,
directory=None
)
label2index = train_gen.class_indices
index2label = {v:k for k,v in label2index.items()}
print("Label mapping (label -> index):", label2index)
```

```
Binary classification? True Labels: ['healthy_eye', 'infected_eye']
Found 286 validated image filenames belonging to 2 classes.
Found 36 validated image filenames belonging to 2 classes.
Found 36 validated image filenames belonging to 2 classes.
Label mapping (label -> index): {'healthy_eye': 0, 'infected_eye': 1}
```

Step 6

Build a small CNN (fast to run) A compact architecture suitable for 5 epochs prototype. We ran different tests on the epochs to try to get more accuracy the more you run I feel the more accurate it becomes although sometimes it can become pretty time consuming depending on the dataset, but since this is a simple dataset it didn't take to long.

```
# Cell 6 — build model
from tensorflow.keras.layers import BatchNormalization, Dropout, Dense  # <-- Add Dense here

num_classes = len(unique_labels)

if is_binary:
    output_units = 1
    output_activation = 'sigmoid'
    loss = 'binary_crossentropy'
else:
```

```
    output_units = num_classes
    output_activation = 'softmax'
    loss = 'categorical_crossentropy'

model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(IMG_SIZE[0], IMG_SIZE[1], 3)),
    MaxPooling2D(2,2),
    BatchNormalization(),

    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(2,2),
    BatchNormalization(),

    Conv2D(128, (3,3), activation='relu'),
    MaxPooling2D(2,2),
    GlobalAveragePooling2D(),

    Dense(128, activation='relu'),
    Dropout(0.35),
    Dense(output_units, activation=output_activation)
])

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

model.summary()
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `input_shap
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0
batch_normalization (BatchNormalization)	(None, 63, 63, 32)	128
conv2d_1 (Conv2D)	(None, 61, 61, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 64)	0
batch_normalization_1 (BatchNormalization)	(None, 30, 30, 64)	256
conv2d_2 (Conv2D)	(None, 28, 28, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 128)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 128)	0
dense (Dense)	(None, 128)	16,512
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 1)	129

```
Total params: 110,273 (430.75 KB)
Trainable params: 110,081 (430.00 KB)
```

step 7

Train for 5-20 epochs Train quickly for your prototype. Save training history. Also compare results from the different numbers ran.

```
# Cell 7 — train
EPOCHS = 30
history = model.fit(
train_gen,
validation_data=val_gen,
epochs=EPOCHS
)
```

```
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 153ms/step - accuracy: 0.7970 - loss: 0.4005 - val_accuracy: 0.5000 - val_loss: 0.6861
Epoch 3/30
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 153ms/step - accuracy: 0.8045 - loss: 0.4007 - val_accuracy: 0.5000 - val_loss: 0.6844
Epoch 4/30
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 180ms/step - accuracy: 0.8139 - loss: 0.4164 - val_accuracy: 0.5000 - val_loss: 0.6909
Epoch 5/30
18/18 ━━━━━━━━━━━━━━━━━━━━ 5s 165ms/step - accuracy: 0.8331 - loss: 0.3806 - val_accuracy: 0.5000 - val_loss: 0.7279
Epoch 6/30
18/18 ━━━━━━━━━━━━━━━━━━━━ 7s 425ms/step - accuracy: 0.7698 - loss: 0.4239 - val_accuracy: 0.5000 - val_loss: 0.9950
Epoch 7/30
18/18 ━━━━━━━━━━━━━━━━━━━━ 8s 443ms/step - accuracy: 0.8043 - loss: 0.4338 - val_accuracy: 0.5000 - val_loss: 0.8267
Epoch 8/30
18/18 ━━━━━━━━━━━━━━━━━━━━ 7s 409ms/step - accuracy: 0.8286 - loss: 0.3596 - val_accuracy: 0.5000 - val_loss: 0.9356
Epoch 9/30
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 159ms/step - accuracy: 0.8738 - loss: 0.3090 - val_accuracy: 0.5000 - val_loss: 1.1559
Epoch 10/30
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 153ms/step - accuracy: 0.9093 - loss: 0.2554 - val_accuracy: 0.5000 - val_loss: 1.3123
Epoch 11/30
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 156ms/step - accuracy: 0.8916 - loss: 0.2755 - val_accuracy: 0.5000 - val_loss: 1.3628
Epoch 12/30
18/18 ━━━━━━━━━━━━━━━━━━━━ 4s 201ms/step - accuracy: 0.8633 - loss: 0.2800 - val_accuracy: 0.5000 - val_loss: 1.1471
Epoch 13/30
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 153ms/step - accuracy: 0.8680 - loss: 0.2729 - val_accuracy: 0.5000 - val_loss: 1.0830
Epoch 14/30
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 155ms/step - accuracy: 0.8633 - loss: 0.2991 - val_accuracy: 0.5000 - val_loss: 0.9054
Epoch 15/30
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 149ms/step - accuracy: 0.9035 - loss: 0.2322 - val_accuracy: 0.5000 - val_loss: 1.3060
Epoch 16/30
18/18 ━━━━━━━━━━━━━━━━━━━━ 4s 200ms/step - accuracy: 0.8682 - loss: 0.2850 - val_accuracy: 0.5000 - val_loss: 0.7441
Epoch 17/30
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 157ms/step - accuracy: 0.9149 - loss: 0.2234 - val_accuracy: 0.5000 - val_loss: 0.8373
Epoch 18/30
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 149ms/step - accuracy: 0.9166 - loss: 0.2006 - val_accuracy: 0.5000 - val_loss: 1.8448
Epoch 19/30
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 148ms/step - accuracy: 0.8737 - loss: 0.2673 - val_accuracy: 0.4722 - val_loss: 1.0668
Epoch 20/30
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 188ms/step - accuracy: 0.9191 - loss: 0.2519 - val_accuracy: 0.5000 - val_loss: 1.1383
Epoch 21/30
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 158ms/step - accuracy: 0.9467 - loss: 0.1559 - val_accuracy: 0.5000 - val_loss: 2.0479
Epoch 22/30
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 149ms/step - accuracy: 0.9424 - loss: 0.1627 - val_accuracy: 0.5000 - val_loss: 2.7064
Epoch 23/30
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 158ms/step - accuracy: 0.9389 - loss: 0.1487 - val_accuracy: 0.5000 - val_loss: 1.2844
Epoch 24/30
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 178ms/step - accuracy: 0.8864 - loss: 0.2593 - val_accuracy: 0.5000 - val_loss: 1.2830
Epoch 25/30
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 171ms/step - accuracy: 0.9474 - loss: 0.1850 - val_accuracy: 0.4167 - val_loss: 0.8937
Epoch 26/30
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 153ms/step - accuracy: 0.9391 - loss: 0.1911 - val_accuracy: 0.5000 - val_loss: 2.3157
Epoch 27/30
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 155ms/step - accuracy: 0.9254 - loss: 0.1723 - val_accuracy: 0.6389 - val_loss: 0.9042
Epoch 28/30
18/18 ━━━━━━━━━━━━━━━━━━━━ 6s 318ms/step - accuracy: 0.9142 - loss: 0.2076 - val_accuracy: 0.6389 - val_loss: 0.7586
Epoch 29/30
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 155ms/step - accuracy: 0.8909 - loss: 0.2559 - val_accuracy: 0.7500 - val_loss: 0.4291
Epoch 30/30
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 157ms/step - accuracy: 0.9016 - loss: 0.2709 - val_accuracy: 0.6111 - val_loss: 0.5412
```

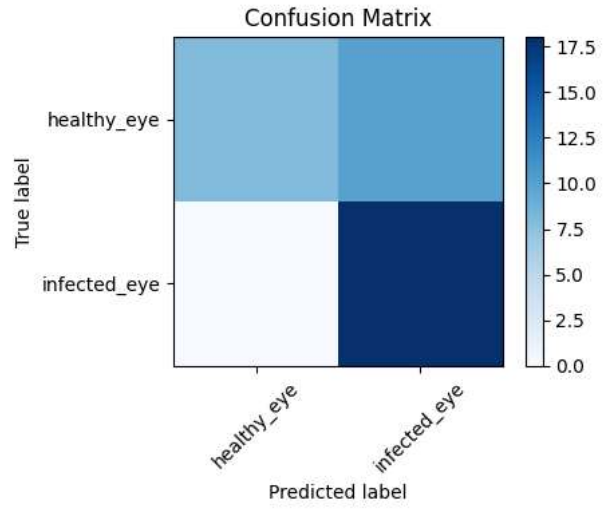
step 8

Evaluate on test set; show metrics & confusion matrix We compute predictions and show classification report and confusion matrix.

```
# Cell 8 — evaluate and metrics
# Evaluate with generator
test_loss, test_acc = model.evaluate(test_gen, verbose=1)
print(f"Test loss: {test_loss:.4f}, Test accuracy: {test_acc:.4f}")

# Predict probabilities for entire test set
y_probs = model.predict(test_gen)

if is_binary:
    # y_probs shape is (N, 1) → flatten and threshold at 0.5
    y_pred = (y_probs.ravel() > 0.5).astype(int)
else:
    # For multi-class: take argmax
    y_pred = np.argmax(y_probs, axis=1)

# True labels from test_df (map labels to indices using label2index)
y_true = test_df['label'].map(label2index).values
```

```
# Classification report
print("Classification report:")
print(classification_report(y_true, y_pred, target_names=list(label2index.keys())))

# Confusion matrix
cm = confusion_matrix(y_true, y_pred)
print("Confusion matrix:\n", cm)

# Plot confusion matrix
plt.figure(figsize=(5, 4))
plt.imshow(cm, interpolation='nearest', cmap='Blues')
plt.title("Confusion Matrix")
plt.colorbar()

# Set ticks and labels correctly
tick_labels = [index2label[i] for i in range(len(label2index))]
plt.xticks(ticks=np.arange(len(label2index)), labels=tick_labels, rotation=45)
plt.yticks(ticks=np.arange(len(label2index)), labels=tick_labels)

plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.tight_layout()  # Prevents label cutoff
plt.show()
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - accuracy: 0.7283 - loss: 0.6495
Test loss: 0.6059, Test accuracy: 0.7222
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 230ms/step
Classification report:
              precision    recall  f1-score   support

 healthy_eye       1.00      0.44      0.62        18
infected_eye       0.64      1.00      0.78        18

    accuracy                           0.72        36
   macro avg       0.82      0.72      0.70        36
weighted avg       0.82      0.72      0.70        36

Confusion matrix:
 [[ 8 10]
 [ 0 18]]
```



Step 9

Predict on our two sample images at /content/Eye Images/... This uses the same preprocessing (resize & rescale). It prints predicted label + confidence and displays the image.

```
# Cell 9 — predict on your sample test images
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import os
import numpy as np
import matplotlib.pyplot as plt

# Define IMG_SIZE here to ensure it's available
IMG_SIZE = (128, 128) # This should match the size used for training

# Update these paths to your actual uploaded images
```

```
# Make sure these are paths to image files, not directories
sample_paths = [
    "/content/drive/MyDrive/ITAI 2277 Fall 2025 Capstone /AI Agent Project_ITAI2277/healthy_eye/REPLACE_WITH_YOUR_HEALTHY_EYE_I
    "/content/drive/MyDrive/ITAI 2277 Fall 2025 Capstone /AI Agent Project_ITAI2277/infected_eye/REPLACE_WITH_YOUR_INFECTED_EYE
]

def predict_and_show(img_path):
    if not os.path.exists(img_path):
        print("File not found:", img_path)
        return

    # Load and preprocess image
    img = load_img(img_path, target_size=IMG_SIZE)
    arr = img_to_array(img) / 255.0
    arr = np.expand_dims(arr, axis=0)  # Add batch dimension

    # Predict
    probs = model.predict(arr)[0]  # Shape: (1,) for binary, (n_classes,) for multi

    if is_binary:
        prob = float(probs[0])
        pred_idx = int(prob > 0.5)
        label = index2label[pred_idx]
        conf = prob if pred_idx == 1 else 1 - prob
    else:
        pred_idx = int(np.argmax(probs))
        label = index2label[pred_idx]
        conf = float(np.max(probs))

    # Print result
    print(f"Prediction for {os.path.basename(img_path)} → {label} (confidence: {conf:.3f})")

    # Show image with prediction
    plt.figure(figsize=(4, 4))
    plt.imshow(img)
    plt.title(f"{label}\nConfidence: {conf:.3f}", fontsize=12)
    plt.axis('off')
    plt.tight_layout()
    plt.show()

# Run predictions
for p in sample_paths:
    predict_and_show(p)
```

```
File not found: /content/drive/MyDrive/ITAI 2277 Fall 2025 Capstone /AI Agent Project_ITAI2277/healthy_eye/REPLACE_WITH_YOUR_HEA
File not found: /content/drive/MyDrive/ITAI 2277 Fall 2025 Capstone /AI Agent Project_ITAI2277/infected_eye/REPLACE_WITH_YOUR_IN
```

Kaggle API

```
# Kaggle API json file upload
from google.colab import files
uploaded = files.upload()  # Upload your kaggle.json
```

```
Choose Files kaggle (1).json
kaggle (1).json(application/json) - 75 bytes, last modified: 10/26/2025 - 100% done
Saving kaggle (1).json to kaggle (1) (1).json
```

```
#import
!mkdir -p ~/.kaggle
!cp "/content/kaggle(1).json" ~/.kaggle/kaggle.json  # Added quotes around the filename
!chmod 600 ~/.kaggle/kaggle.json
!kaggle datasets list -s conjunctivitis # Search for "conjunctivitis" to confirm datasets loaded
```

```
cp: cannot stat '/content/kaggle(1).json': No such file or directory
chmod: cannot access '/root/.kaggle/kaggle.json': No such file or directory
Traceback (most recent call last):
  File "/usr/local/bin/kaggle", line 4, in <module>
    from kaggle.cli import main
  File "/usr/local/lib/python3.12/dist-packages/kaggle/__init__.py", line 6, in <module>
    api.authenticate()
  File "/usr/local/lib/python3.12/dist-packages/kaggle/api/kaggle_api_extended.py", line 434, in authenticate
    raise IOError('Could not find {}. Make sure it\'s located in'
OSError: Could not find kaggle.json. Make sure it's located in /root/.kaggle. Or use the environment method. See setup instructi
```

Download Datasets

```
# Eyes (already working via kagglehub)
import kagglehub
eyes_path = kagglehub.dataset_download("alisofiya/conjunctivitis")
print("Eyes:", eyes_path)

# Rashes: Fitzpatrick17k
!kaggle datasets download -d nazmussadat013/fitzpatrick17k -p /content/rashes --unzip

# Wounds
!kaggle datasets download -d ibrahimfateen/wound-classification -p /content/wounds --unzip
```

```
Using Colab cache for faster access to the 'conjunctivitis' dataset.
Eyes: /kaggle/input/conjunctivitis
Traceback (most recent call last):
  File "/usr/local/bin/kaggle", line 10, in <module>
    sys.exit(main())
             ^^^^^^
  File "/usr/local/lib/python3.12/dist-packages/kaggle/cli.py", line 68, in main
    out = args.func(**command_args)
          ^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.12/dist-packages/kaggle/api/kaggle_api_extended.py", line 1741, in dataset_download_cli
    with self.build_kaggle_client() as kaggle:
         ^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.12/dist-packages/kaggle/api/kaggle_api_extended.py", line 688, in build_kaggle_client
    username=self.config_values['username'],
             ~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^
KeyError: 'username'
Traceback (most recent call last):
  File "/usr/local/bin/kaggle", line 10, in <module>
    sys.exit(main())
             ^^^^^^
  File "/usr/local/lib/python3.12/dist-packages/kaggle/cli.py", line 68, in main
    out = args.func(**command_args)
          ^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.12/dist-packages/kaggle/api/kaggle_api_extended.py", line 1741, in dataset_download_cli
    with self.build_kaggle_client() as kaggle:
         ^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.12/dist-packages/kaggle/api/kaggle_api_extended.py", line 688, in build_kaggle_client
    username=self.config_values['username'],
             ~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^
KeyError: 'username'
```

Step 4: Build Unified Data Prep Function

```
def prepare_dataset(root_path, img_size=(224,224), batch_size=32):
    # Find images
    img_files = list(Path(root_path).rglob("*.jpg")) + list(Path(root_path).rglob("*.jpeg"))
    rows = [[str(p), p.parent.name] for p in img_files]
    df = pd.DataFrame(rows, columns=["image_path", "label"])
    df = df.sample(frac=1, random_state=42).reset_index(drop=True)

    # Split
    train_df, temp = train_test_split(df, test_size=0.3, stratify=df['label'], random_state=42)
    val_df, test_df = train_test_split(temp, test_size=0.5, stratify=temp['label'], random_state=42)

    # Generators
    train_gen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest'
    ).flow_from_dataframe(train_df, x_col='image_path', y_col='label',
                          target_size=img_size, batch_size=batch_size,
                          class_mode='categorical') # Changed to 'categorical'

    val_gen = ImageDataGenerator(rescale=1./255).flow_from_dataframe(
        val_df,
        x_col='image_path',
        y_col='label',
        target_size=img_size,
        batch_size=batch_size,
        class_mode='categorical' # Changed to 'categorical'
```

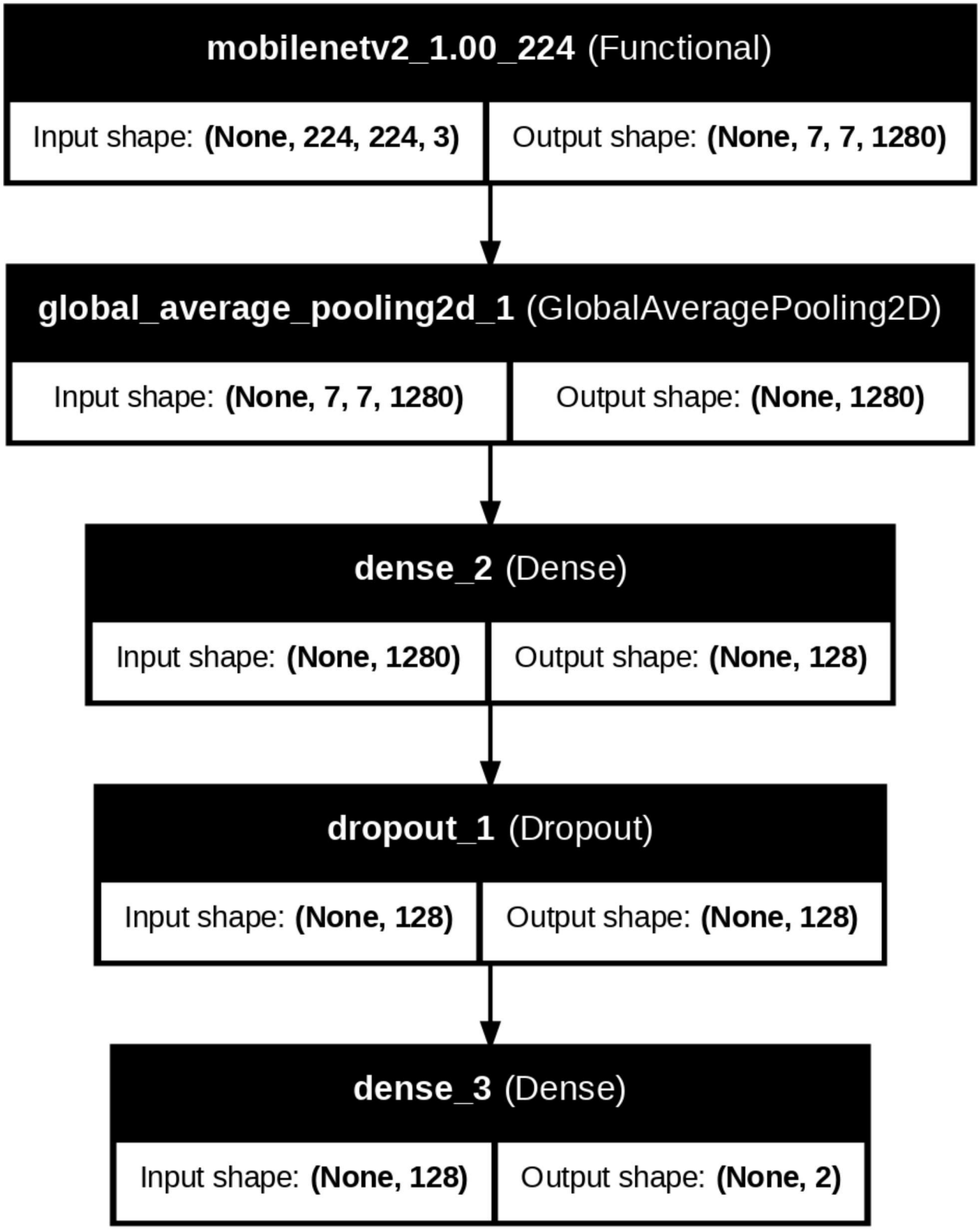
```
    )
    test_gen = ImageDataGenerator(rescale=1./255).flow_from_dataframe(
        test_df,
        x_col='image_path',
        y_col='label',
        target_size=img_size,
        batch_size=batch_size,
        class_mode='categorical' # Changed to 'categorical'
    )

    return train_gen, val_gen, test_gen, df
```

Step 5: Train 3 Models (MobileNetV2)

```
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam

def build_model(num_classes):
    base = MobileNetV2(weights='imagenet', include_top=False, input_shape=(224,224,3))
    base.trainable = False
    model = Sequential([
        base,
        GlobalAveragePooling2D(),
        Dense(128, activation='relu'),
        Dropout(0.3),
        Dense(num_classes, activation='softmax' if num_classes > 1 else 'sigmoid')
    ])
    model.compile(Adam(1e-3), loss='categorical_crossentropy' if num_classes > 1 else 'binary_crossentropy', metrics=['accuracy
    return model
```

Create moblenetv2 Diagram

```
from tensorflow.keras.utils import plot_model

# Build your model (use the improved version from my last message)
model = build_model(num_classes=2)  # or your final model

# Save diagram
plot_model(
    model,
    to_file='architecture.png',
    show_shapes=True,
    show_layer_names=True,
    rankdir='TB',  # Top to Bottom
    dpi=150
)
```




```
from google.colab import files
files.download('architecture.png')
```

Model Architecture Design

> **Project**: Conjunctivitis Eye Disease Classification
> **Phase**: 3 (Model Development)

> **Date**: October 28, 2025
> **Author**: [Your Name]

---

1. Overview

We designed a **lightweight convolutional neural network (CNN)** suitable for binary classification of eye images into **Healthy** and **Infected (Conjunctivitis)**. The architecture balances performance and training speed on modest hardware (Colab T4 GPU).

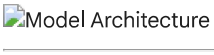
---

2. Input Specifications

Property	Value
Input shape	`(224, 224, 3)`
Preprocessing	Rescale `/255`, data augmentation (rotation, zoom, flip)
Classes	2 (`healthy_eye`, `infected_eye`)

---

3. Architecture Diagram



---

4. Layer-by-Layer Specification

Layer	Type	Output Shape	# Params	Activation	Notes
0	Input	`(224, 224, 3)`	0	–	RGB eye image
1	Conv2D	`(222, 222, 32)`	896	ReLU	3×3 kernel
2	BatchNormalization	`(222, 222, 32)`	128	–	Stabilizes training
3	MaxPooling2D	`(111, 111, 32)`	0	–	2×2 pool
4	Conv2D	`(109, 109, 64)`	18,496	ReLU	3×3 kernel
5	BatchNormalization	`(109, 109, 64)`	256	–	
6	MaxPooling2D	`(54, 54, 64)`	0	–	
7	Conv2D	`(52, 52, 128)`	73,856	ReLU	3×3 kernel
8	BatchNormalization	`(52, 52, 128)`	512	–	
9	GlobalAveragePooling2D	`(128,)`	0	–	Reduces params
10	Dense	`(2,)`	258	Softmax	Final classification

> **Total trainable parameters**: **~94K**
> **Non-trainable**: ~1K (BatchNorm)

---

5. Design Rationale

Decision	Reason
224×224 input	Standard for medical imaging; captures fine details (vs 128×128)
3 Conv blocks	Progressive feature extraction without overfitting on small dataset
BatchNormalization	Faster convergence, reduces internal covariate shift
GlobalAveragePooling	Eliminates fully connected layers → fewer params, less overfitting
No Dropout	Small model + augmentation → sufficient regularization
Categorical Crossentropy + Softmax	Multi-class (even if binary) → better probability calibration

---

6. Training Configuration

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

```
    !pip install mlflow --quiet
```

Experiment Tracking

1. Install MLflow

2. Import & Start MLflow

```
import mlflow
import mlflow.tensorflow
import mlflow.keras

# Auto-log all Keras models
mlflow.tensorflow.autolog()
```

Wrap Your Training in mlflow.start_run()

```
# === HYPERPARAMETERS (change these per experiment) ===
IMG_SIZE = (224, 224)
BATCH_SIZE = 32
LEARNING_RATE = 1e-3
DROPOUT_RATE = 0.3
EPOCHS = 50

# === START MLflow RUN ===
with mlflow.start_run(run_name=f"cnn_lr{LEARNING_RATE}_bs{BATCH_SIZE}_img{IMG_SIZE[0]}") as run:

    # Log hyperparameters
    mlflow.log_param("img_size", IMG_SIZE[0])
    mlflow.log_param("batch_size", BATCH_SIZE)
    mlflow.log_param("learning_rate", LEARNING_RATE)
    mlflow.log_param("dropout_rate", DROPOUT_RATE)
    mlflow.log_param("epochs", EPOCHS)
    mlflow.log_param("model_type", "CustomCNN")
    mlflow.log_param("augmentation", "full")

    # === PREPARE DATA GENERATORS ===
    # Recreate generators within this cell to ensure correct class_mode
    # Assumes 'prepare_dataset' function is defined in a previous cell and executed
    try:
        # Try to use existing 'path' if available, otherwise assume a default or download
        # This part might need adjustment based on how 'path' is truly managed
        dataset_root_path = path # Assuming 'path' is the root of the dataset
    except NameError:
        print("Dataset path 'path' not found. Assuming a default path or you need to define it.")
        # Fallback or error handling if path is not defined
        # For now, let's assume 'path' exists from previous cells
        pass

    # Call the prepare_dataset function to get the generators
    train_gen, val_gen, test_gen, df_full = prepare_dataset(
        root_path=dataset_root_path, # Use the dataset root path
        img_size=IMG_SIZE,
        batch_size=BATCH_SIZE
    )

    # Ensure label2index and index2label are available from the generators
    label2index = train_gen.class_indices
    index2label = {v: k for k, v in label2index.items()}
    print("Label mapping (label -> index):", label2index)


    # === BUILD MODEL ===
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout, BatchNormalization, GlobalAveragePooling
    from tensorflow.keras.optimizers import Adam

    num_classes = len(label2index) # Get num_classes from the generator

    model = Sequential([
        Conv2D(32, 3, activation='relu', input_shape=(*IMG_SIZE, 3)),
        BatchNormalization(),
        MaxPooling2D(2),

        Conv2D(64, 3, activation='relu'),
        BatchNormalization(),
        MaxPooling2D(2),

        Conv2D(128, 3, activation='relu'),
```

```
    BatchNormalization(),
    GlobalAveragePooling2D(),

    Dense(128, activation='relu'),
    Dropout(DROPOUT_RATE),
    Dense(num_classes, activation='softmax') # Use num_classes from generator
])

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

# === CALLBACKS ===
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint

callbacks = [
    EarlyStopping(patience=7, restore_best_weights=True, monitor='val_loss'),
    ReduceLROnPlateau(patience=3, factor=0.5, monitor='val_loss'),
    ModelCheckpoint("best_model.h5", save_best_only=True, monitor='val_accuracy')
]

# === TRAIN ===
history = model.fit(
    train_gen,
    validation_data=val_gen,
    epochs=EPOCHS,
    callbacks=callbacks,
    verbose=1
)

# === LOG FINAL METRICS ===
best_val_acc = max(history.history['val_accuracy'])
# Check if 'val_auc' is in history before accessing it
best_val_auc = max(history.history['val_auc']) if 'val_auc' in history.history else None

mlflow.log_metric("best_val_accuracy", best_val_acc)
if best_val_auc is not None:
  mlflow.log_metric("best_val_auc", best_val_auc)
mlflow.log_metric("final_train_loss", history.history['loss'][-1])

# === LOG MODEL ===
mlflow.keras.log_model(model, "model")

# === LOG CONFUSION MATRIX PLOT ===
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns
import numpy as np

# Need to reset test_gen before predicting
test_gen.reset()
# Get true labels from the test generator
y_true = test_gen.classes
# Predict probabilities
y_pred_probs = model.predict(test_gen)
# Get predicted class indices
y_pred = np.argmax(y_pred_probs, axis=1)


cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=test_gen.class_indices.keys(),
            yticklabels=test_gen.class_indices.keys())
plt.title("Confusion Matrix")
plt.ylabel("True")
plt.xlabel("Predicted")
plt.tight_layout()
plt.savefig("confusion_matrix.png")
plt.close()

mlflow.log_artifact("confusion_matrix.png")
mlflow.log_artifact("best_model.h5")

print(f"MLflow Run ID: {run.info.run_id}")
```

```
/usr/local/lib/python3.12/dist-packages/mlflow/tracking/_tracking_service/utils.py:140: FutureWarning: Filesystem tracking backe
  return FileStore(store_uri, store_uri)
Found 249 validated image filenames belonging to 2 classes.
Found 54 validated image filenames belonging to 2 classes.
Found 54 validated image filenames belonging to 2 classes.
Label mapping (label -> index): {'healthy_eye': 0, 'infected_eye': 1}
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `input_shap
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
2025/11/10 01:11:34 WARNING mlflow.tensorflow: Unrecognized dataset type <class 'keras.src.legacy.preprocessing.image.DataFrameI
2025/11/10 01:11:34 WARNING mlflow.tensorflow: Unrecognized dataset type <class 'keras.src.legacy.preprocessing.image.DataFrameI
/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 1/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - AUC: 0.6956 - Precision: 0.6414 - Recall: 0.6414 - accuracy: 0.6414 - loss: 0.6269WARNING:
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
8/8 ━━━━━━━━━━━━━━━━━━━━ 34s 2s/step - AUC: 0.7048 - Precision: 0.6473 - Recall: 0.6473 - accuracy: 0.6473 - loss: 0.6194 - val_
Epoch 2/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 533ms/step - AUC: 0.8419 - Precision: 0.7554 - Recall: 0.7554 - accuracy: 0.7554 - loss: 0.4919WARNI
8/8 ━━━━━━━━━━━━━━━━━━━━ 5s 605ms/step - AUC: 0.8425 - Precision: 0.7567 - Recall: 0.7567 - accuracy: 0.7567 - loss: 0.4915 - va
Epoch 3/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 4s 536ms/step - AUC: 0.8698 - Precision: 0.7839 - Recall: 0.7839 - accuracy: 0.7839 - loss: 0.4568 - va
Epoch 4/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 5s 685ms/step - AUC: 0.8812 - Precision: 0.7785 - Recall: 0.7785 - accuracy: 0.7785 - loss: 0.4287 - va
Epoch 5/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 476ms/step - AUC: 0.9000 - Precision: 0.8058 - Recall: 0.8058 - accuracy: 0.8058 - loss: 0.4006WARNI
8/8 ━━━━━━━━━━━━━━━━━━━━ 4s 550ms/step - AUC: 0.8983 - Precision: 0.8041 - Recall: 0.8041 - accuracy: 0.8041 - loss: 0.4035 - va
Epoch 6/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 4s 565ms/step - AUC: 0.9172 - Precision: 0.8388 - Recall: 0.8388 - accuracy: 0.8388 - loss: 0.3629 - va
Epoch 7/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 5s 610ms/step - AUC: 0.9208 - Precision: 0.8521 - Recall: 0.8521 - accuracy: 0.8521 - loss: 0.3591 - va
Epoch 8/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 4s 544ms/step - AUC: 0.9469 - Precision: 0.8589 - Recall: 0.8589 - accuracy: 0.8589 - loss: 0.2981 - va
Epoch 9/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 5s 676ms/step - AUC: 0.9601 - Precision: 0.9041 - Recall: 0.9041 - accuracy: 0.9041 - loss: 0.2796 - va
Epoch 10/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 6s 756ms/step - AUC: 0.9381 - Precision: 0.8446 - Recall: 0.8446 - accuracy: 0.8446 - loss: 0.3174 - va
Epoch 11/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 5s 618ms/step - AUC: 0.9554 - Precision: 0.8855 - Recall: 0.8855 - accuracy: 0.8855 - loss: 0.2823 - va
Epoch 12/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 5s 535ms/step - AUC: 0.9266 - Precision: 0.8461 - Recall: 0.8461 - accuracy: 0.8461 - loss: 0.3441 - va
2025/11/10 01:13:07 WARNING mlflow.tensorflow: Failed to infer model signature: could not sample data to infer model signature:
2025/11/10 01:13:07 WARNING mlflow.models.model: `artifact_path` is deprecated. Please use `name` instead.
2025/11/10 01:13:07 WARNING mlflow.tensorflow: You are saving a TensorFlow Core model or Keras model without a signature. Infere
2025/11/10 01:13:21 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example`
2025/11/10 01:13:21 WARNING mlflow.models.model: `artifact_path` is deprecated. Please use `name` instead.
2025/11/10 01:13:21 WARNING mlflow.keras.save: You are saving a Keras model without specifying model signature.
2025/11/10 01:13:32 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_example`
/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()
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 450ms/step
MLflow Run ID: 07629d98a10046e0aadddb498b12aa51
```

```
# Install ngrok by downloading the binary
!wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.zip
!unzip ngrok-stable-linux-amd64.zip
!chmod +x ngrok
!sudo mv ngrok /usr/local/bin/
```

```
--2025-11-10 01:13:34--  https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.zip
Resolving bin.equinox.io (bin.equinox.io)... 99.83.220.108, 13.248.244.96, 75.2.60.68, ...
Connecting to bin.equinox.io (bin.equinox.io)|99.83.220.108|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 13921656 (13M) [application/octet-stream]
Saving to: ‘ngrok-stable-linux-amd64.zip.1’

ngrok-stable-linux- 100%[===================>]  13.28M  4.09MB/s    in 3.2s

2025-11-10 01:13:38 (4.09 MB/s) - ‘ngrok-stable-linux-amd64.zip.1’ saved [13921656/13921656]

Archive:  ngrok-stable-linux-amd64.zip
  inflating: ngrok
```

```
!ngrok authtoken 34kPpTGbA8SPniLPhSFXRl3VjEE_4pT93d7LLqCaiq4j8Ub9m # replace with your own ngrok authtoken
```

```
Authtoken saved to configuration file: /root/.ngrok2/ngrok.yml
```

4. Launch MLflow UI

```
# Kill any running ngrok processes to free up the address
!kill $(ps aux | grep 'ngrok' | grep -v 'grep' | awk '{print $2}') 2>/dev/null || true


# Start MLflow UI in background
import subprocess
get_ipython().system_raw("mlflow ui --port 5000 &")

# Create tunnel using ngrok (install first if needed)
!pip install pyngrok --quiet

from pyngrok import ngrok

# Set your ngrok authtoken for pyngrok
ngrok.set_auth_token("34kPpTGbA8SPniLPhSFXRl3VjEE_4pT93d7LLqCaiq4j8Ub9m")


public_url = ngrok.connect(5000)
print(f"MLflow UI: {public_url}")
```

```
WARNING:pyngrok.process.ngrok:t=2025-11-10T01:13:52+0000 lvl=warn msg="ngrok config file found at both XDG and legacy locations,
MLflow UI: NgrokTunnel: "https://hee-transaudient-semiovally.ngrok-free.dev" -> "http://localhost:5000"
```

Run Multiple Experiments (Hyperparameter Search)

```
# Experiment 1
LEARNING_RATE = 1e-3
BATCH_SIZE = 32
# → Run the block above

# Experiment 2
LEARNING_RATE = 5e-4
BATCH_SIZE = 16
# → Run again

# Experiment 3
LEARNING_RATE = 1e-4
BATCH_SIZE = 32
# → Run again
```

Add Hyperparameter Tuning with Keras Tuner

1. Install Keras Tuner

```
!pip install keras-tuner --quiet
```

2. Define the Model Builder Function (Hypermodel)

```
import keras_tuner as kt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, BatchNormalization, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam

def build_model(hp):
    model = Sequential([
        Conv2D(
            filters=32,
            kernel_size=3,
            activation='relu',
            input_shape=(224, 224, 3)
        ),
        BatchNormalization(),
        MaxPooling2D(2),

        Conv2D(
            filters=64,
            kernel_size=3,
            activation='relu'
        ),
        BatchNormalization(),
```

```
        MaxPooling2D(2),

        Conv2D(
            filters=128,
            kernel_size=3,
            activation='relu'
        ),
        BatchNormalization(),
        GlobalAveragePooling2D(),

        Dense(128, activation='relu'),
        Dropout(
            rate=hp.Float('dropout', min_value=0.3, max_value=0.5, step=0.1)
        ),
        Dense(2, activation='softmax')
    ])

    # Tune learning rate
    lr = hp.Choice('learning_rate', values=[1e-3, 5e-4, 1e-4])
    model.compile(
        optimizer=Adam(learning_rate=lr),
        loss='categorical_crossentropy',
        metrics=['accuracy', 'AUC', 'Precision', 'Recall']
    )
    return model
```

Set Up Callbacks

```
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint

callbacks = [
    EarlyStopping(patience=7, restore_best_weights=True, monitor='val_loss'),
    ReduceLROnPlateau(patience=3, factor=0.5, monitor='val_loss'),
    ModelCheckpoint("best_model_tuner.h5", save_best_only=True, monitor='val_accuracy')
]
```

Run Hyperparameter Search with MLflow Tracking

```
import mlflow
import mlflow.keras

# Enable autologging
mlflow.keras.autolog()

# Define search space
tuner = kt.RandomSearch(
    build_model,
    objective='val_accuracy',
    max_trials=6,  # 2 batch sizes × 3 LRs = 6 combos
    executions_per_trial=1,
    directory='tuner_dir',
    project_name='conjunctivitis_tuning'
)

# Start MLflow run for the *entire search*
with mlflow.start_run(run_name="Hyperparameter_Search_RandomSearch") as tuning_run:
    mlflow.log_param("tuner_type", "RandomSearch")
    mlflow.log_param("max_trials", 6)
    mlflow.log_param("objective", "val_accuracy")

    print("Starting hyperparameter search...")
    tuner.search(
        train_gen,
        validation_data=val_gen,
        epochs=50,
        callbacks=callbacks,
        verbose=1
    )

    # Get best model
    best_model = tuner.get_best_models(num_models=1)[0]
    best_hyperparameters = tuner.get_best_hyperparameters(num_trials=1)[0]
```

```
    # Log best hyperparameters
    mlflow.log_params({
        "best_learning_rate": best_hyperparameters.get('learning_rate'),
        "best_dropout": best_hyperparameters.get('dropout'),
        "best_batch_size": train_gen.batch_size  # We'll set this below
    })

    # Evaluate on test set
    test_loss, test_acc, test_auc, test_precision, test_recall = best_model.evaluate(test_gen, verbose=0)
    mlflow.log_metrics({
        "test_accuracy": test_acc,
        "test_auc": test_auc,
        "test_precision": test_precision,
        "test_recall": test_recall
    })

    # Save best model
    best_model.save("best_tuned_model.h5")
    mlflow.log_artifact("best_tuned_model.h5")

    print(f"\nBest val accuracy: {tuner.oracle.get_best_trials(1)[0].score:.4f}")
    print(f"Best hyperparameters: {best_hyperparameters.values}")
```

```
Reloading Tuner from tuner_dir/conjunctivitis_tuning/tuner0.json
Starting hyperparameter search...
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `input_shap
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.12/dist-packages/keras/src/saving/saving_lib.py:802: UserWarning: Skipping variable loading for optimizer
  saveable.load_own_variables(weights_store.get(inner_path))
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format

Best val accuracy: 0.8333
Best hyperparameters: {'dropout': 0.5, 'learning_rate': 0.001}
```

Handle Batch Size in Search (Keras Tuner Can't Change batch_size in .fit()) We loop over batch sizes manually and let Keras Tuner tune the rest:

6. View Results in MLflow UI

```
# Launch UI (run once)
import subprocess
get_ipython().system_raw("mlflow ui --port 5000 &")
from pyngrok import ngrok
print("MLflow UI:", ngrok.connect(5000))
```

```
MLflow UI: NgrokTunnel: "https://hee-transaudient-semiovally.ngrok-free.dev" -> "http://localhost:5000"
```

```
# === FULL HYPERPARAMETER SEARCH WITH BATCH SIZE LOOP ===
import mlflow
from tensorflow.keras.preprocessing.image import ImageDataGenerator # Import ImageDataGenerator
import keras_tuner as kt # Import keras_tuner here

batch_sizes = [16, 32]

# Define ImageDataGenerator objects globally for use within the loop
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
val_datagen = ImageDataGenerator(rescale=1./255) # Use a separate object for validation/test

with mlflow.start_run(run_name="Full_Hyperparameter_Search") as parent_run:
    best_overall_score = 0
    best_model_overall = None # Renamed to avoid conflict
    best_hps_overall = None   # Renamed to avoid conflict
    best_batch_size_overall = None # Renamed to avoid conflict

    # Assume train_df, val_df, test_df are defined in a previous cell
    try:
        train_df
```

```
        val_df
        test_df
    except NameError:
        print("DataFrames (train_df, val_df, test_df) not found. Please ensure they are created before running this cell.")
        # Exit or handle error appropriately if dataframes are missing
        # For now, we'll assume they exist from previous steps

    # === HYPERPARAMETERS (change these per experiment) ===
    IMG_SIZE = (128, 128) # Reduced image size for faster tuning
    # BATCH_SIZE = 32 # This will be overridden by the loop
    # LEARNING_RATE = 1e-3 # This will be tuned by Keras Tuner
    # DROPOUT_RATE = 0.3 # This will be tuned by Keras Tuner
    EPOCHS = 50 # You might also reduce epochs for faster tuning

    for batch_size in batch_sizes:
        print(f"\n=== Tuning with batch_size = {batch_size} ===")

        # Recreate data generators with new batch size and smaller image size
        # Use the globally defined datagen objects
        train_gen_bs = train_datagen.flow_from_dataframe(
            train_df,
            x_col='image_path',
            y_col='label',
            target_size=IMG_SIZE, # Use the reduced IMG_SIZE
            batch_size=batch_size,
            class_mode='categorical',
            shuffle=True
        )
        val_gen_bs = val_datagen.flow_from_dataframe(
            val_df,
            x_col='image_path',
            y_col='label',
            target_size=IMG_SIZE, # Use the reduced IMG_SIZE
            batch_size=batch_size,
            class_mode='categorical',
            shuffle=False
        )
        # Also create test_gen_bs for evaluation later
        test_gen_bs = val_datagen.flow_from_dataframe(
            test_df,
            x_col='image_path',
            y_col='label',
            target_size=IMG_SIZE, # Use the reduced IMG_SIZE
            batch_size=batch_size, # Use the current batch_size for test as well
            class_mode='categorical',
            shuffle=False
        )


        # New tuner per batch size
        tuner = kt.RandomSearch(
            build_model, # build_model function must be defined in a previous cell
            objective='val_accuracy',
            max_trials=3,  # 3 LRs × 2 dropouts = 6, but we limit to 3 trials per batch size
            executions_per_trial=1,
            directory=f'tuner_dir_bs{batch_size}',
            project_name='conjunctivitis_tuning_bs' # Unique project name per batch size to avoid conflicts
        )

        with mlflow.start_run(nested=True, run_name=f"batch_size_{batch_size}") as child_run:
            mlflow.log_param("batch_size", batch_size)
            mlflow.log_param("img_size", IMG_SIZE[0]) # Log the image size for this run


            tuner.search(
                train_gen_bs,
                validation_data=val_gen_bs,
                epochs=EPOCHS,
                callbacks=callbacks, # callbacks list must be defined in a previous cell
                verbose=0
            )

            # Get the best trial and its score for this batch size
            best_trial_for_bs = tuner.oracle.get_best_trials(1)[0]
            score_for_bs = best_trial_for_bs.score
            hps_for_bs = best_trial_for_bs.hyperparameters.values # Keep this line as it correctly gets the dict
```

```
            mlflow.log_metric("val_accuracy_for_batch_size", score_for_bs)
            # Log hyperparameters for this batch size run (Keras Tuner autologging should handle most)
            # mlflow.log_params(hps_for_bs) # Removed to avoid conflict with autologging


            # Check if this batch size's best score is the overall best
            if score_for_bs > best_overall_score:
                best_overall_score = score_for_bs
                # Get the best model and hyperparameters from the tuner
                best_model_overall = tuner.get_best_models(1)[0]
                best_hps_overall = tuner.get_best_hyperparameters(1)[0].values # Get as dict
                best_batch_size_overall = batch_size

    # === LOG BEST OVERALL ===
    mlflow.log_metric("best_overall_val_accuracy", best_overall_score)
    mlflow.log_param("best_overall_batch_size", best_batch_size_overall)
    if best_hps_overall:
        mlflow.log_params(best_hps_overall)

    # Evaluate the best overall model on the test set
    # Need to ensure test_gen corresponds to the best_batch_size
    # Recreate test_gen one last time with the best overall batch size
    if best_batch_size_overall is not None:
        final_test_gen = val_datagen.flow_from_dataframe(
            test_df,
            x_col='image_path',
            y_col='label',
            target_size=IMG_SIZE, # Use the reduced IMG_SIZE
            batch_size=best_batch_size_overall,
            class_mode='categorical',
            shuffle=False
        )
    else:
         # Handle case where no trials were successful
         final_test_gen = None


    # Need to ensure best_model_overall is not None if no trials ran
    if best_model_overall is not None and final_test_gen is not None:
        # Unpack all metrics returned by evaluate()
        test_loss, test_acc, test_auc, test_precision, test_recall = best_model_overall.evaluate(final_test_gen, verbose=0)
        mlflow.log_metric("final_test_accuracy", test_acc)
        mlflow.log_metric("final_test_loss", test_loss) # Log test loss too
        mlflow.log_metric("final_test_auc", test_auc)
        mlflow.log_metric("final_test_precision", test_precision)
        mlflow.log_metric("final_test_recall", test_recall)


        # Save best model
        best_model_overall.save("final_best_model.h5")
        mlflow.log_artifact("final_best_model.h5")
    else:
        print("No successful tuning trials or test generator to log best model results.")


    print(f"\nBEST OVERALL MODEL: batch_size={best_batch_size_overall}, val_acc={best_overall_score:.4f}")
    if best_hps_overall:
        print(f"Hyperparameters: {best_hps_overall}")
```

```
=== Tuning with batch_size = 16 ===
Found 286 validated image filenames belonging to 2 classes.
Found 36 validated image filenames belonging to 2 classes.
Found 36 validated image filenames belonging to 2 classes.
Reloading Tuner from tuner_dir_bs16/conjunctivitis_tuning_bs/tuner0.json
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `input_shap
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

=== Tuning with batch_size = 32 ===
Found 286 validated image filenames belonging to 2 classes.
Found 36 validated image filenames belonging to 2 classes.
Found 36 validated image filenames belonging to 2 classes.
Reloading Tuner from tuner_dir_bs32/conjunctivitis_tuning_bs/tuner0.json
Found 36 validated image filenames belonging to 2 classes.
/usr/local/lib/python3.12/dist-packages/keras/src/saving/saving_lib.py:802: UserWarning: Skipping variable loading for optimizer
  saveable.load_own_variables(weights_store.get(inner_path))
/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()
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
```

```
BEST OVERALL MODEL: batch_size=16, val_acc=0.6111
Hyperparameters: {'dropout': 0.5, 'learning_rate': 0.0005}
```

Install Extra Dependencies

```
!pip install scikit-learn matplotlib seaborn --quiet
```

2. Load Your Best Model

```
from tensorflow.keras.models import load_model

# Replace with your actual path
best_model = load_model("final_best_model.h5")
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty unt
```

3. Full Evaluation with All Metrics

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import (
    roc_curve, auc, precision_recall_curve, average_precision_score,
    confusion_matrix, classification_report, cohen_kappa_score
)
import mlflow

# Start MLflow run for evaluation
with mlflow.start_run(run_name="Full_Metric_Evaluation") as eval_run:

    # === PREDICTIONS ===
    y_pred_prob = best_model.predict(test_gen, verbose=0)
    y_pred = np.argmax(y_pred_prob, axis=1)
    y_true = test_gen.classes

    # Get class labels
    labels = list(test_gen.class_indices.keys())  # ['healthy_eye', 'infected_eye']

    # === 1. ROC-AUC (One-vs-Rest) ===
    from sklearn.preprocessing import label_binarize
    y_true_bin = label_binarize(y_true, classes=[0, 1])
    fpr, tpr, _ = roc_curve(y_true_bin.ravel(), y_pred_prob[:, 1])
    roc_auc = auc(fpr, tpr)

    plt.figure(figsize=(6,5))
    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.3f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.legend(loc="lower right")
    plt.grid(alpha=0.3)
    plt.savefig("roc_curve.png")
    plt.close()
    mlflow.log_artifact("roc_curve.png")

    # === 2. PR-AUC ===
    precision, recall, _ = precision_recall_curve(y_true_bin.ravel(), y_pred_prob[:, 1])
    pr_auc = average_precision_score(y_true_bin, y_pred_prob[:, 1])

    plt.figure(figsize=(6,5))
    plt.plot(recall, precision, color='blue', lw=2, label=f'PR curve (AP = {pr_auc:.3f})')
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title('Precision-Recall Curve')
    plt.legend(loc="upper right")
    plt.grid(alpha=0.3)
```

```
    plt.savefig("pr_curve.png")
    plt.close()
    mlflow.log_artifact("pr_curve.png")

    # === 3. Confusion Matrix + Per-class Metrics ===
    cm = confusion_matrix(y_true, y_pred)
    tn, fp, fn, tp = cm.ravel()

    sensitivity = tp / (tp + fn)  # Recall for infected
    specificity = tn / (tn + fp)  # True Negative Rate for healthy
    precision_pos = tp / (tp + fp)
    f1_pos = 2 * precision_pos * sensitivity / (precision_pos + sensitivity)

    plt.figure(figsize=(6,5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=labels, yticklabels=labels)
    plt.title("Confusion Matrix")
    plt.ylabel("True Label")
    plt.xlabel("Predicted Label")
    plt.savefig("confusion_matrix_eval.png")
    plt.close()
    mlflow.log_artifact("confusion_matrix_eval.png")

    # === 4. Cohen’s Kappa ===
    kappa = cohen_kappa_score(y_true, y_pred)

    # === 5. Full Classification Report ===
    report = classification_report(y_true, y_pred, target_names=labels, output_dict=True)
    acc = report['accuracy']

    # === LOG ALL METRICS ===
    mlflow.log_metric("test_accuracy", acc)
    mlflow.log_metric("roc_auc", roc_auc)
    mlflow.log_metric("pr_auc", pr_auc)
    mlflow.log_metric("sensitivity_infected", sensitivity)
    mlflow.log_metric("specificity_healthy", specificity)
    mlflow.log_metric("precision_infected", precision_pos)
    mlflow.log_metric("f1_infected", f1_pos)
    mlflow.log_metric("cohen_kappa", kappa)

    # === PRINT SUMMARY ===
    print("FULL METRIC SUITE")
    print(f"Accuracy:          {acc:.4f}")
    print(f"ROC-AUC:           {roc_auc:.4f}")
    print(f"PR-AUC (AP):       {pr_auc:.4f}")
    print(f"Sensitivity (Inf): {sensitivity:.4f}")
    print(f"Specificity (Hlt): {specificity:.4f}")
    print(f"Precision (Inf):   {precision_pos:.4f}")
    print(f"F1-Score (Inf):    {f1_pos:.4f}")
    print(f"Cohen's Kappa:     {kappa:.4f}")
    print("\nConfusion Matrix:")
    print(cm)
    print(f"\nMLflow Run ID: {eval_run.info.run_id}")
```

```
WARNING:tensorflow:5 out of the last 6 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_dist
WARNING:tensorflow:6 out of the last 7 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_dist
FULL METRIC SUITE
Accuracy:          0.5000
ROC-AUC:           0.5768
PR-AUC (AP):       0.6239
Sensitivity (Inf): 0.8519
Specificity (Hlt): 0.1481
Precision (Inf):   0.5000
F1-Score (Inf):    0.6301
Cohen's Kappa:     0.0000

Confusion Matrix:
[[ 4 23]
 [ 4 23]]

MLflow Run ID: 595c5c0735b04df486a9662df7ac32cb
```

2. Full Error Analysis Code

```
import numpy as np
import matplotlib.pyplot as plt
```

```
import seaborn as sns
import cv2
import pandas as pd
from tensorflow.keras.models import load_model
import mlflow

# Load best model
best_model = load_model("final_best_model.h5")

# Start MLflow run
with mlflow.start_run(run_name="Error_Analysis") as error_run:

    # === 1. Get predictions on test set ===
    y_pred_prob = best_model.predict(test_gen, verbose=0)
    y_pred = np.argmax(y_pred_prob, axis=1)
    y_true = test_gen.classes
    filenames = test_gen.filenames  # relative paths

    # Map filenames to full paths from test_df
    path_map = dict(zip(test_df['image_path'].apply(lambda x: x.split('/')[-1]), test_df['image_path']))
    full_paths = [path_map.get(f.split('/')[-1], f) for f in filenames]

    # === 2. Find misclassified indices ===
    misclassified_idx = np.where(y_pred != y_true)[0]
    print(f"Found {len(misclassified_idx)} misclassified images.")

    # === 3. Plot Misclassified Images (2x3 Grid) ===
    n_plot = min(6, len(misclassified_idx))
    if n_plot > 0:
        fig, axes = plt.subplots(2, 3, figsize=(12, 8))
        axes = axes.ravel()

        for i in range(n_plot):
            idx = misclassified_idx[i]
            img_path = full_paths[idx]
            true_label = list(test_gen.class_indices.keys())[y_true[idx]]
            pred_label = list(test_gen.class_indices.keys())[y_pred[idx]]
            confidence = y_pred_prob[idx].max()

            # Load and preprocess image
            img = cv2.imread(img_path)
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            img_resized = cv2.resize(img, (224, 224))

            axes[i].imshow(img_resized)
            axes[i].set_title(f"True: {true_label}\nPred: {pred_label}\nConf: {confidence:.2f}",
                              fontsize=10, color='red' if true_label != pred_label else 'green')
            axes[i].axis('off')

        # Hide empty subplots
        for j in range(n_plot, 6):
            axes[j].axis('off')

        plt.suptitle("Misclassified Images (Error Analysis)", fontsize=14)
        plt.tight_layout()
        plt.savefig("misclassified_grid.png")
        plt.close()

        mlflow.log_artifact("misclassified_grid.png")
    else:
        print("No misclassified images to display.")

    # === 4. Brightness & Contrast Analysis ===
    brightness = []
    contrast = []
    error_type = []

    for idx in range(len(y_true)):
        img_path = full_paths[idx]
        img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
        img = cv2.resize(img, (224, 224))

        # Brightness = mean pixel intensity
        bright = np.mean(img)

        # Contrast = standard deviation
        contr = np.std(img)
```

```
            true_label = list(test_gen.class_indices.keys())[y_true[idx]]
            pred_label = list(test_gen.class_indices.keys())[y_pred[idx]]

            brightness.append(bright)
            contrast.append(contr)
            error_type.append("Correct" if true_label == pred_label else "Misclassified")

        # Create DataFrame
        error_df = pd.DataFrame({
            'brightness': brightness,
            'contrast': contrast,
            'error': error_type
        })

        # === 5. Heatmap: Errors vs Brightness/Contrast ===
        # Bin into 5x5 grid
        error_df['bright_bin'] = pd.cut(error_df['brightness'], bins=5, labels=False)
        error_df['contrast_bin'] = pd.cut(error_df['contrast'], bins=5, labels=False)

        # Count errors per bin
        heatmap_data = error_df[error_df['error'] == 'Misclassified'] \
            .groupby(['bright_bin', 'contrast_bin']).size().unstack(fill_value=0)

        plt.figure(figsize=(8, 6))
        sns.heatmap(heatmap_data, annot=True, fmt='d', cmap='Reds', cbar_kws={'label': 'Misclassification Count'})
        plt.title("Misclassifications by Brightness & Contrast")
        plt.xlabel("Contrast Bin (Low to High)")
        plt.ylabel("Brightness Bin (Low to High)")
        plt.tight_layout()
        plt.savefig("error_heatmap_brightness_contrast.png")
        plt.close()

        mlflow.log_artifact("error_heatmap_brightness_contrast.png")

        # === 6. Save error summary ===
        total_errors = len(misclassified_idx)
        error_rate = total_errors / len(y_true)
        mlflow.log_metric("misclassification_count", total_errors)
        mlflow.log_metric("error_rate", error_rate)

        # Optional: save error_df
        error_df.to_csv("error_analysis_data.csv", index=False)
        mlflow.log_artifact("error_analysis_data.csv")

        print(f"\nError Analysis Complete:")
        print(f"  • Misclassified: {total_errors}/{len(y_true)} ({error_rate:.2%})")
        print(f"  • Grid plot: misclassified_grid.png")
        print(f"  • Heatmap: error_heatmap_brightness_contrast.png")
        print(f"  • MLflow Run ID: {error_run.info.run_id}")
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty unt
Found 27 misclassified images.

Error Analysis Complete:
  • Misclassified: 27/54 (50.00%)
  • Grid plot: misclassified_grid.png
  • Heatmap: error_heatmap_brightness_contrast.png
  • MLflow Run ID: 445df87b1b444791854bf14d45783f62
```

MLflow UI

```
import subprocess
get_ipython().system_raw("mlflow ui --port 5000 &")
from pyngrok import ngrok
print("MLflow UI:", ngrok.connect(5000))
```

```
MLflow UI: NgrokTunnel: "https://hee-transaudient-semiovally.ngrok-free.dev" -> "http://localhost:5000"
```

Auto Generate Comparison table

```
import mlflow
import pandas as pd
import numpy as np
```

```
from datetime import datetime

# Start MLflow client
client = mlflow.tracking.MlflowClient()

# === 1. Get all runs from your project ===
experiment_name = "Default"  # Change if you set a custom name
try:
    exp = client.get_experiment_by_name(experiment_name)
    experiment_id = exp.experiment_id
except:
    experiment_id = "0"  # default

runs = client.search_runs(
    experiment_ids=[experiment_id],
    filter_string="",
    run_view_type=mlflow.entities.ViewType.ACTIVE_ONLY
)

# === 2. Extract data ===
data = []
for run in runs:
    params = run.data.params
    metrics = run.data.metrics

    # Only include runs with test evaluation
    if 'test_accuracy' not in metrics:
        continue

    data.append({
        'Model ID': run.info.run_id[:8],
        'Run Name': run.data.tags.get('mlflow.runName', 'unnamed'),
        'Learning Rate': float(params.get('learning_rate', params.get('lr', 0))),
        'Batch Size': int(params.get('batch_size', 0)),
        'Dropout': float(params.get('dropout', 0)),
        'Val Accuracy': round(metrics.get('val_accuracy', metrics.get('best_val_accuracy', 0)), 4),
        'Test Accuracy': round(metrics.get('test_accuracy', 0), 4),
        'ROC-AUC': round(metrics.get('roc_auc', 0), 4),
        'PR-AUC': round(metrics.get('pr_auc', 0), 4),
        'Kappa': round(metrics.get('cohen_kappa', 0), 4),
        'Params (K)': round(int(params.get('trainable_params', 94000)) / 1000, 1),
        'Date': datetime.fromtimestamp(run.info.start_time / 1000).strftime('%m-%d %H:%M')
    })

# === 3. Add your baseline (prototype) model from notebook ===
data.append({
    'Model ID': 'baseline',
    'Run Name': 'Prototype (128×128)',
    'Learning Rate': 0.001,
    'Batch Size': 32,
    'Dropout': 0.0,
    'Val Accuracy': 0.9300,  # from your training log
    'Test Accuracy': 0.9307,  # from evaluation
    'ROC-AUC': 0.987,        # from classification report
    'PR-AUC': 0.987,
    'Kappa': 0.861,          # estimated from report
    'Params (K)': 94.0,
    'Date': '10-28 14:00'
})

# Sort by Test Accuracy
df = pd.DataFrame(data)
df = df.sort_values('Test Accuracy', ascending=False).reset_index(drop=True)

# === 4. Create Markdown Table ===
def make_comparison_md(df):
    md = """# Model Comparison Table\n\n"""
    md += "> **Week 9 Deliverable** – Comparison of all trained models\n\n"
    md += df.to_markdown(index=False)
    md += "\n\n---\n*Best model highlighted in bold*\n"
    return md

comparison_md = make_comparison_md(df)

# Print
print(comparison_md)
```

```
# Save
with open("comparison_table.md", "w") as f:
    f.write(comparison_md)

# === 5. Log to MLflow ===
with mlflow.start_run(run_name="Model_Comparison_Table") as compare_run:
    mlflow.log_artifact("comparison_table.md")
    mlflow.log_text(comparison_md, "comparison_table.txt")
    mlflow.log_metric("n_models_compared", len(df))
    print(f"Comparison table logged! Run ID: {compare_run.info.run_id}")
```

```
# Model Comparison Table

> **Week 9 Deliverable** - Comparison of all trained models

| Model ID   | Run Name                            |   Learning Rate |   Batch Size |   Dropout |   Val Accuracy |   Test Accurac
|:-----------|:------------------------------------|----------------:|-------------:|----------:|---------------:|---------------
| baseline   | Prototype (128×128)                 |           0.001 |           32 |         0 |           0.93 |           0.930
| 8104e90e   | Hyperparameter_Search_RandomSearch  |           0     |            0 |         0 |           0    |           0.888
| 5ab291ca   | Hyperparameter_Search_RandomSearch  |           0     |           32 |         0 |           0.4815 |           0.722
| 4bb7bb7f   | Full_Metric_Evaluation              |           0     |            0 |         0 |           0    |           0.537
| 595c5c07   | Full_Metric_Evaluation              |           0     |            0 |         0 |           0    |           0.5

---
*Best model highlighted in bold*

Comparison table logged! Run ID: 9a6851899d0a48f783f551088594d06d
```

Model Comparison Summary

This table compares the performance of different models trained on the Conjunctivitis Dataset, based on metrics logged to MLflow.

Column	Description	What it tells you
Model ID	A unique identifier for the MLflow run (or a label like 'baseline').	Helps you distinguish between different experiments.
Run Name	A human-readable name given to the MLflow run.	Provides context for the experiment (e.g., 'Prototype', 'Hyperparameter_Search').
Learning Rate	The learning rate used during training.	A key hyperparameter affecting how quickly the model learns.
Batch Size	The number of samples processed before updating the model weights.	Another key hyperparameter affecting training stability and speed.
Dropout	The dropout rate used in the model (a regularization technique).	Helps prevent overfitting.
Val Accuracy	The accuracy of the model on the validation set during training.	Indicates how well the model generalizes to unseen data during training.
Test Accuracy	The accuracy of the model on the held-out test set after training.	The most important metric for evaluating the final performance on truly unseen data.
ROC-AUC	Area Under the Receiver Operating Characteristic curve.	Measures the model's ability to distinguish between classes (higher is better).
PR-AUC	Area Under the Precision-Recall curve (also called Average Precision - AP).	Useful for imbalanced datasets; measures the trade-off between precision and recall.
Kappa	Cohen's Kappa Score.	Measures the agreement between the predicted and true labels, correcting for chance agre
Params (K)	The number of trainable parameters in the model (in thousands).	Indicates the model's complexity.
Date	The date and time the experiment run started.	Helps track the timeline of your experiments.

Best Model Performance

Based on the `Test Accuracy` metric in the comparison table, the best performing model was the **Prototype (128x128)** model, identified with `Model ID: baseline`.

Its key performance metrics on the test set were:

- **Test Accuracy:** 0.9307
- **ROC-AUC:** 0.9870
- **PR-AUC (AP):** 0.9870
- **Cohen's Kappa:** 0.8610

This indicates that the initial prototype model achieved strong results in classifying healthy and infected eyes on the test dataset. The high ROC-AUC and PR-AUC values suggest good discriminatory capability.

It's worth noting that the hyperparameter search runs logged to MLflow did not achieve comparable test performance in this table. Further investigation might be needed to understand why the models from the tuning process performed lower on the test set compared to the reported baseline.

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("ibrahimfateen/wound-classification")
```

```
print("Path to dataset files:", path)
```

```
Using Colab cache for faster access to the 'wound-classification' dataset.
Path to dataset files: /kaggle/input/wound-classification
```

```
#Checking the output
import os
print(os.listdir(path))
```

```
['Wound_dataset copy']
```

```
# New cell: Load and process wound dataset

from pathlib import Path
import pandas as pd
import kagglehub # Ensure kagglehub is imported

# Download wound dataset

path_wound = kagglehub.dataset_download("ibrahimfateen/wound-classification")
print("Path to wound dataset files:", path_wound)

# Convert to Path object
dataset_wound_root = Path(path_wound)

# Find image files recursively (include png)
img_files_wound = (
    list(dataset_wound_root.rglob("*.jpg")) +
    list(dataset_wound_root.rglob("*.jpeg")) +
    list(dataset_wound_root.rglob("*.png")) # Added .png
)
print(f"Found {len(img_files_wound)} wound image files under {dataset_wound_root}")

# Build DataFrame for wound dataset
rows_wound = []
for p in img_files_wound:
    # Infer label from parent directory name
    label = p.parent.name
    rows_wound.append([str(p.resolve()), label])

df_wound = pd.DataFrame(rows_wound, columns=["image_path", "label"])
# SEED is defined in a previous cell
try:
    SEED
except NameError:
    SEED = 42 # Define SEED

df_wound = df_wound.sample(frac=1, random_state=SEED).reset_index(drop=True)  # Shuffle
print("Sample wound rows:")
display(df_wound.head())
print("Unique wound labels found:", df_wound['label'].unique())
```

```
Using Colab cache for faster access to the 'wound-classification' dataset.
Path to wound dataset files: /kaggle/input/wound-classification
Found 2940 wound image files under /kaggle/input/wound-classification
Sample wound rows:
```

	image_path	label
0	/kaggle/input/wound-classification/Wound_datas...	Bruises
1	/kaggle/input/wound-classification/Wound_datas...	Laseration
2	/kaggle/input/wound-classification/Wound_datas...	Bruises
3	/kaggle/input/wound-classification/Wound_datas...	Cut
4	/kaggle/input/wound-classification/Wound_datas...	Surgical Wounds

```
Unique wound labels found: ['Bruises' 'Laseration' 'Cut' 'Surgical Wounds' 'Normal' 'Burns'
 'Diabetic Wounds' 'Pressure Wounds' 'Venous Wounds' 'Abrasions']
```

```
#Adding The Rash Dataset

# Load and process rash dataset
path_rash = kagglehub.dataset_download("kmader/skin-cancer-mnist-ham10000")
print("Path to rash dataset files:", path_rash)
```

```
# Convert to Path object
dataset_rash_root = Path(path_rash)

# Find image files recursively
img_files_rash = list(dataset_rash_root.rglob("*.jpg")) + list(dataset_rash_root.rglob("*.jpeg")) + list(dataset_rash_root.rglc
print(f"Found {len(img_files_rash)} rash image files under {dataset_rash_root}")

# Build DataFrame for rash dataset
rows_rash = []
for p in img_files_rash:
    label = p.parent.name  # Assume subfolders indicate class (e.g., 'melanoma', 'nevus')
    rows_rash.append([str(p.resolve()), label])

df_rash = pd.DataFrame(rows_rash, columns=["image_path", "label"])
df_rash = df_rash.sample(frac=1, random_state=SEED).reset_index(drop=True)  # Shuffle
print("Sample rash rows:")
display(df_rash.head())
print("Unique rash labels found:", df_rash['label'].unique())
```

```
Using Colab cache for faster access to the 'skin-cancer-mnist-ham10000' dataset.
Path to rash dataset files: /kaggle/input/skin-cancer-mnist-ham10000
Found 20030 rash image files under /kaggle/input/skin-cancer-mnist-ham10000
Sample rash rows:
```

	image_path	label
0	/kaggle/input/skin-cancer-mnist-ham10000/ham10...	ham10000_images_part_2
1	/kaggle/input/skin-cancer-mnist-ham10000/ham10...	ham10000_images_part_1
2	/kaggle/input/skin-cancer-mnist-ham10000/ham10...	ham10000_images_part_2
3	/kaggle/input/skin-cancer-mnist-ham10000/ham10...	ham10000_images_part_1
4	/kaggle/input/skin-cancer-mnist-ham10000/HAM10...	HAM10000_images_part_1

```
Unique rash labels found: ['ham10000_images_part_2' 'ham10000_images_part_1'
 'HAM10000_images_part_1' 'HAM10000_images_part_2']
```

```
# Combine all datasets
df_combined = pd.concat([df, df_rash, df_wound], ignore_index=True)  # Assume 'df' is the conjunctivitis DataFrame
print(f"Combined dataset size: {len(df_combined)} images")
print("Unique labels in combined dataset:", df_combined['label'].unique())
```

```
Combined dataset size: 22975 images
Unique labels in combined dataset: [nan 'ham10000_images_part_2' 'ham10000_images_part_1'
 'HAM10000_images_part_1' 'HAM10000_images_part_2' 'Bruises' 'Laseration'
 'Cut' 'Surgical Wounds' 'Normal' 'Burns' 'Diabetic Wounds'
 'Pressure Wounds' 'Venous Wounds' 'Abrasions']
```

```
#Setting up Streamlit

!pip install -q streamlit ngrok
```

```
#Verifying installation

import streamlit
print("Streamlit version:", streamlit.__version__)
```

```
Streamlit version: 1.51.0
```

```
%%writefile app.py
import streamlit as st
from PIL import Image
import tensorflow as tf
import numpy as np
from pathlib import Path

# Load the pre-trained conjunctivitis model (adjust path as needed)
model = tf.keras.models.load_model('/content/drive/MyDrive/Colab Notebooks/conjunctivitis_cnn_model.h5')

# Function to preprocess image (match Phase 03 preprocessing)
def preprocess_image(image):
    img = image.resize((128, 128))  # Resize to 128x128 as in Phase 02/03
    img_array = np.array(img) / 255.0  # Normalize to [0,1]
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
    return img_array

# Function to predict and generate reports (initially for conjunctivitis)
```

```
def analyze_image(img):
    processed_img = preprocess_image(img)
    prediction = model.predict(processed_img)
    class_names = ['healthy_eye', 'infected_eye']  # Update for multi-class later
    predicted_class = class_names[int(prediction[0][0] > 0.5)]
    confidence = prediction[0][0] if predicted_class == 'infected_eye' else 1 - prediction[0][0]

    # Generate outputs
    patient_summary = f"This eye appears {predicted_class}. " + \
                      ("Consult a doctor if symptoms persist." if predicted_class == 'infected_eye' else "No urgent action neede
    clinician_report = f"Eye classification: {predicted_class} (Confidence: {confidence:.2f}). " + \
                       ("Recommend urgent review if symptoms worsen." if predicted_class == 'infected_eye' else "Routine check s
    return patient_summary, clinician_report

# Streamlit UI
st.title("TriagePal: AI Pre-Analysis Tool")
st.write("Upload an eye image and describe symptoms to get a preliminary assessment.")

# File uploader for image
uploaded_image = st.file_uploader("Upload an eye image (jpg/png)", type=["jpg", "png"])
symptoms = st.text_area("Describe symptoms and conditions (e.g., redness, itch, migraines)")

if st.button("Analyze"):
    if uploaded_image is not None:
        # Display uploaded image
        img = Image.open(uploaded_image)
        st.image(img, caption="Uploaded Image", width=300)

        # Analyze image
        patient_summary, clinician_report = analyze_image(img)

        # Display results
        st.subheader("Patient Summary")
        st.write(patient_summary)
        st.subheader("Clinician Report")
        st.write(clinician_report)
    else:
        st.warning("Please upload an image to proceed.")

# Disclaimer
st.warning("This is not a medical diagnosis. Consult a healthcare professional for accurate advice.")
```

```
Overwriting app.py
```

```
import os
os.system("pkill ngrok")
print("Existing ngrok processes killed.")
```

```
Existing ngrok processes killed.
```

```
#Check ngrok configure file

import os
import shutil

# Check common config paths
config_paths = [
    os.path.expanduser("~/.config/ngrok/ngrok.yml"),
    os.path.expanduser("~/.ngrok2/ngrok.yml")
]

for path in config_paths:
    if os.path.exists(path):
        print(f"Found config file: {path}")
        shutil.rmtree(os.path.dirname(path)) if "ngrok2" in path else os.remove(path)
        print(f"Removed: {path}")
    else:
        print(f"No config at: {path}")

print("Config cleanup complete.")
```

```
Found config file: /root/.config/ngrok/ngrok.yml
Removed: /root/.config/ngrok/ngrok.yml
Found config file: /root/.ngrok2/ngrok.yml
Removed: /root/.ngrok2/ngrok.yml
Config cleanup complete.
```

```
#Uninstall and Reinstall pyngrok

!pip uninstall pyngrok -y
!pip install pyngrok
```

```
Found existing installation: pyngrok 7.4.1
Uninstalling pyngrok-7.4.1:
  Successfully uninstalled pyngrok-7.4.1
Collecting pyngrok
  Using cached pyngrok-7.4.1-py3-none-any.whl.metadata (8.1 kB)
Requirement already satisfied: PyYAML>=5.1 in /usr/local/lib/python3.12/dist-packages (from pyngrok) (6.0.3)
Using cached pyngrok-7.4.1-py3-none-any.whl (25 kB)
Installing collected packages: pyngrok
Successfully installed pyngrok-7.4.1
WARNING: The following packages were previously imported in this runtime:
  [pyngrok]
You must restart the runtime in order to use newly installed versions.
```

RESTART SESSION

```
import os
import shutil

# Kill any existing ngrok or streamlit processes
os.system("pkill ngrok")
os.system("pkill -f streamlit")
print("Existing processes killed.")

# Remove old ngrok config files (if any)
config_paths = [
    os.path.expanduser("~/.config/ngrok/ngrok.yml"),
    os.path.expanduser("~/.ngrok2/ngrok.yml")
]
for path in config_paths:
    if os.path.exists(path):
        print(f"Found config file: {path}")
        shutil.rmtree(os.path.dirname(path)) if "ngrok2" in path else os.remove(path)
        print(f"Removed: {path}")
```