



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.

```
In [37]: 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 & imports

This installs required packages (safe if already installed) and imports modules.

```
In [38]: # Cell 1 — install & imports
!pip install -q kagglehub tensorflow matplotlib scikit-learn pandas pillow

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

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img,
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

# for reproducibility
SEED = 42
random.seed(SEED)
np.random.seed(SEED)
tf.random.set_seed(SEED)

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

TensorFlow version: 2.19.0

Step 2 — Ensure dataset path is available (use path if it exists, otherwise download)

This cell checks whether path exists (from your earlier `kagglehub.dataset_download(...)`) and downloads if needed. It then finds image files recursively under that path and infers labels from parent folders (e.g., `.../healthy/...jpg -> label healthy`).

```
In [39]: # Cell 2 – locate / download dataset and build a DataFrame (image_path, label)
try:
    path # use existing variable if present
    print("Using existing `path` variable:", path)
except NameError:
    import kagglehub
    print("`path` not found – downloading dataset via kagglehub...")
    path = kagglehub.dataset_download("alisofiya/conjunctivitis")
    print("Downloaded dataset to:", path)

# convert to Path
dataset_root = Path(path)

# find all image files (common extensions)
img_files = list(dataset_root.rglob("*.jpg")) + list(dataset_root.rglob("*.jpeg"))
print(f"Found {len(img_files)} image files under {dataset_root}")

# Build dataframe: label = immediate parent directory name
rows = []
for p in img_files:
    label = p.parent.name # parent folder name
    rows.append([str(p.resolve()), label])

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

Using existing `path` variable: /kaggle/input/conjunctivitis  
Found 358 image files under /kaggle/input/conjunctivitis  
Sample rows:

	image_path	label
0	/kaggle/input/conjunctivitis/Dataset/infected_...	infected_eye
1	/kaggle/input/conjunctivitis/Dataset/healthy_e...	healthy_eye
2	/kaggle/input/conjunctivitis/Dataset/infected_...	infected_eye
3	/kaggle/input/conjunctivitis/Dataset/infected_...	infected_eye
4	/kaggle/input/conjunctivitis/Dataset/healthy_e...	healthy_eye

Unique labels found: ['infected\_eye' 'healthy\_eye']

Step 3 — (Optional) Quick EDA: class counts and sample images

Shows how many images per class and some example images.

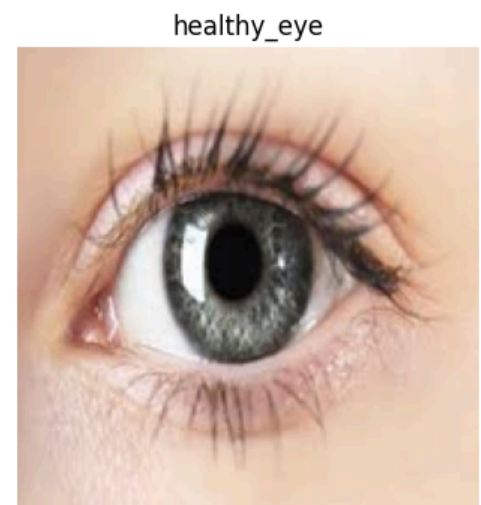
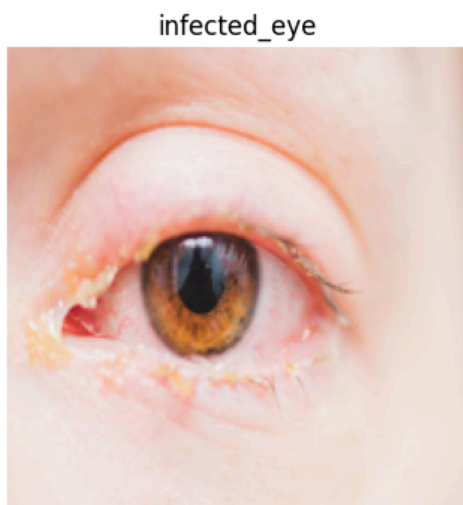
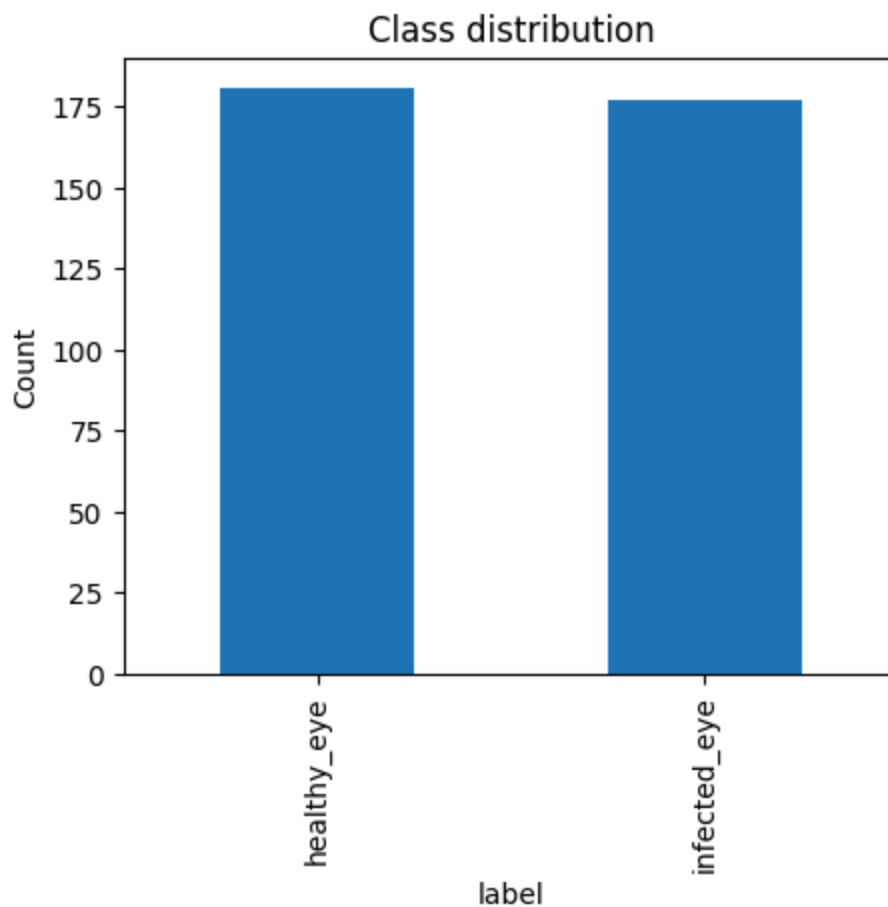
```
In [40]: # Cell 3 – EDA: class distribution + sample images
counts = df['label'].value_counts()
print("Class distribution:\n", counts)

# 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).

```
In [41]: # Cell 4 – train/val/test split
train_df, temp_df = train_test_split(df, test_size=0.2, stratify=df['label'],
val_df, test_df = train_test_split(temp_df, test_size=0.5, stratify=temp_df['l
print("Counts -> Train:", len(train_df), "Val:", len(val_df), "Test:", len(test
```

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.

```
In [42]: # 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
if is_binary:
    class_mode = 'binary'
else:
    class_mode = 'categorical'

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,

```

In [43]: # Cell 6 – build model
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],
    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\_shape`/`input\_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

```
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

**Model: "sequential\_3"**

Layer (type)	Output Shape	Param #
conv2d_9 (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d_9 (MaxPooling2D)	(None, 63, 63, 32)	0
batch_normalization_6 (BatchNormalization)	(None, 63, 63, 32)	128
conv2d_10 (Conv2D)	(None, 61, 61, 64)	18,496
max_pooling2d_10 (MaxPooling2D)	(None, 30, 30, 64)	0
batch_normalization_7 (BatchNormalization)	(None, 30, 30, 64)	256
conv2d_11 (Conv2D)	(None, 28, 28, 128)	73,856
max_pooling2d_11 (MaxPooling2D)	(None, 14, 14, 128)	0
global_average_pooling2d_3 (GlobalAveragePooling2D)	(None, 128)	0
dense_6 (Dense)	(None, 128)	16,512
dropout_3 (Dropout)	(None, 128)	0
dense_7 (Dense)	(None, 1)	129

**Total params:** 110,273 (430.75 KB)

**Trainable params:** 110,081 (430.00 KB)


**Non-trainable params:** 192 (768.00 B)


## Step 7 — Train for 5-20 epochs


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


```
In [44]: # Cell 7 – train
EPOCHS = 30
history = model.fit(
    train_gen,
    validation_data=val_gen,
    epochs=EPOCHS
)
```


```
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_data_adapter_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
```


Epoch 1/30  
**18/18**  **14s** 621ms/step - accuracy: 0.6789 - loss: 0.6007 - val\_accuracy: 0.5000 - val\_loss: 0.6905


Epoch 2/30  
**18/18**  **11s** 604ms/step - accuracy: 0.8117 - loss: 0.3955 - val\_accuracy: 0.5000 - val\_loss: 0.6877


Epoch 3/30  
**18/18**  **10s** 547ms/step - accuracy: 0.8282 - loss: 0.3537 - val\_accuracy: 0.5000 - val\_loss: 0.6908


Epoch 4/30  
**18/18**  **11s** 557ms/step - accuracy: 0.8067 - loss: 0.4218 - val\_accuracy: 0.5000 - val\_loss: 0.6938


Epoch 5/30  
**18/18**  **11s** 598ms/step - accuracy: 0.8273 - loss: 0.3547 - val\_accuracy: 0.5000 - val\_loss: 0.7083


Epoch 6/30  
**18/18**  **11s** 609ms/step - accuracy: 0.7637 - loss: 0.4799 - val\_accuracy: 0.5000 - val\_loss: 0.7703


Epoch 7/30  
**18/18**  **11s** 590ms/step - accuracy: 0.7928 - loss: 0.4384 - val\_accuracy: 0.5000 - val\_loss: 0.7790


Epoch 8/30  
**18/18**  **11s** 597ms/step - accuracy: 0.8324 - loss: 0.3659 - val\_accuracy: 0.5000 - val\_loss: 0.8986


Epoch 9/30  
**18/18**  **10s** 543ms/step - accuracy: 0.8493 - loss: 0.3423 - val\_accuracy: 0.5000 - val\_loss: 1.0070


Epoch 10/30  
**18/18**  **11s** 566ms/step - accuracy: 0.8848 - loss: 0.2640 - val\_accuracy: 0.5000 - val\_loss: 1.1598


Epoch 11/30  
**18/18**  **11s** 604ms/step - accuracy: 0.8560 - loss: 0.3115 - val\_accuracy: 0.5000 - val\_loss: 1.0424


Epoch 12/30  
**18/18**  **11s** 601ms/step - accuracy: 0.8379 - loss: 0.3051 - val\_accuracy: 0.5000 - val\_loss: 1.3722


Epoch 13/30  
**18/18**  **12s** 655ms/step - accuracy: 0.8536 - loss: 0.2886 - val\_accuracy: 0.5000 - val\_loss: 0.8961













Epoch 14/30  
**18/18**  **11s** 611ms/step - accuracy: 0.8853 - loss: 0.3071 - val\_accuracy: 0.5000 - val\_loss: 0.9579

Epoch 15/30  
**18/18**  **11s** 594ms/step - accuracy: 0.8928 - loss: 0.2440 - val\_accuracy: 0.5000 - val\_loss: 1.3621

Epoch 16/30  
**18/18**  **21s** 592ms/step - accuracy: 0.8533 - loss: 0.3085 - val\_accuracy: 0.5000 - val\_loss: 1.1853

Epoch 17/30  
**18/18**  **11s** 615ms/step - accuracy: 0.8606 - loss: 0.2739 - val\_accuracy: 0.5000 - val\_loss: 1.4214

Epoch 18/30  
**18/18**  **11s** 602ms/step - accuracy: 0.9356 - loss: 0.2042 - val\_accuracy: 0.5000 - val\_loss: 1.0606

Epoch 19/30  
**18/18**  **11s** 601ms/step - accuracy: 0.9154 - loss: 0.2435 - val\_accuracy: 0.5000 - val\_loss: 0.9956  
Epoch 20/30  
**18/18**  **11s** 611ms/step - accuracy: 0.8794 - loss: 0.2743 - val\_accuracy: 0.4722 - val\_loss: 0.8961  
Epoch 21/30  
**18/18**  **10s** 538ms/step - accuracy: 0.9313 - loss: 0.1674 - val\_accuracy: 0.5000 - val\_loss: 1.3437  
Epoch 22/30  
**18/18**  **11s** 560ms/step - accuracy: 0.9440 - loss: 0.1538 - val\_accuracy: 0.5000 - val\_loss: 2.2401  
Epoch 23/30  
**18/18**  **21s** 602ms/step - accuracy: 0.9510 - loss: 0.1544 - val\_accuracy: 0.5000 - val\_loss: 1.3507  
Epoch 24/30  
**18/18**  **20s** 597ms/step - accuracy: 0.9262 - loss: 0.2115 - val\_accuracy: 0.5000 - val\_loss: 1.5631  
Epoch 25/30  
**18/18**  **10s** 542ms/step - accuracy: 0.9257 - loss: 0.1713 - val\_accuracy: 0.4722 - val\_loss: 1.5957  
Epoch 26/30  
**18/18**  **11s** 562ms/step - accuracy: 0.9257 - loss: 0.2019 - val\_accuracy: 0.5000 - val\_loss: 2.1964  
Epoch 27/30  
**18/18**  **21s** 600ms/step - accuracy: 0.9168 - loss: 0.1775 - val\_accuracy: 0.5000 - val\_loss: 0.8494  
Epoch 28/30  
**18/18**  **11s** 617ms/step - accuracy: 0.9439 - loss: 0.1940 - val\_accuracy: 0.8056 - val\_loss: 0.4436  
Epoch 29/30  
**18/18**  **11s** 607ms/step - accuracy: 0.8890 - loss: 0.2207 - val\_accuracy: 0.7222 - val\_loss: 0.5475  
Epoch 30/30  
**18/18**  **10s** 546ms/step - accuracy: 0.8688 - loss: 0.3204 - val\_accuracy: 0.8611 - val\_loss: 0.3763

Step 8 — Evaluate on test set; show metrics & confusion matrix

We compute predictions and show classification report and confusion matrix.

```
In [45]: # 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)
    y_pred = (y_probs.ravel() > 0.5).astype(int)
else:
    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()
plt.xticks(ticks=list(range(len(label2index))), labels=[index2label[i] for i in range(len(label2index))])
plt.yticks(ticks=list(range(len(label2index))), labels=[index2label[i] for i in range(len(label2index))])
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()

```

3/3 ————— 1s 138ms/step - accuracy: 0.7917 - loss: 0.4108

Test loss: 0.3729, Test accuracy: 0.8333

3/3 ————— 1s 128ms/step

Classification report:

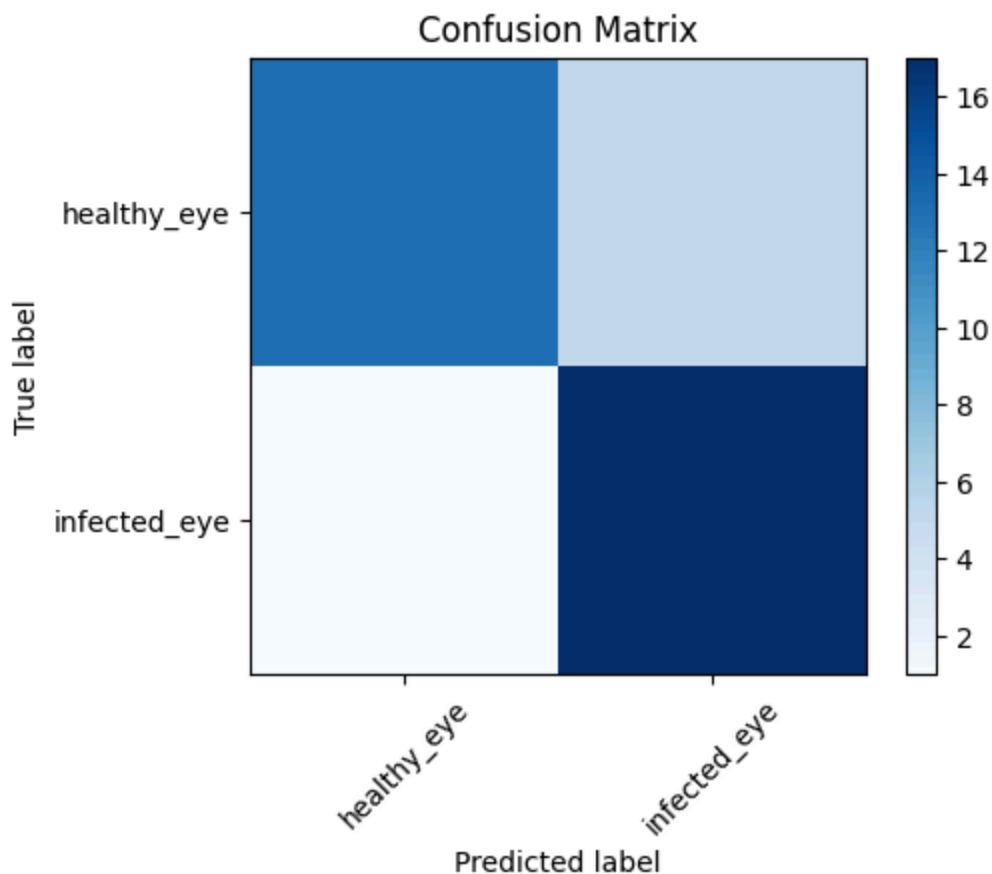
	precision	recall	f1-score	support
healthy_eye	0.93	0.72	0.81	18
infected_eye	0.77	0.94	0.85	18
accuracy			0.83	36
macro avg	0.85	0.83	0.83	36
weighted avg	0.85	0.83	0.83	36

Confusion matrix:

```

[[13  5]
 [ 1 17]]

```



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.

```
In [46]: # Cell 9 – predict on your sample test images
sample_paths = [
    "/content/Eye Images/eye1.jpg",
    "/content/Eye Images/eye2.jpg"
]

def predict_and_show(img_path):
    if not os.path.exists(img_path):
        print("File not found:", img_path)
        return
    img = load_img(img_path, target_size=IMG_SIZE)
    arr = img_to_array(img) / 255.0
    arr = np.expand_dims(arr, axis=0)
    probs = model.predict(arr)[0]
    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:
```

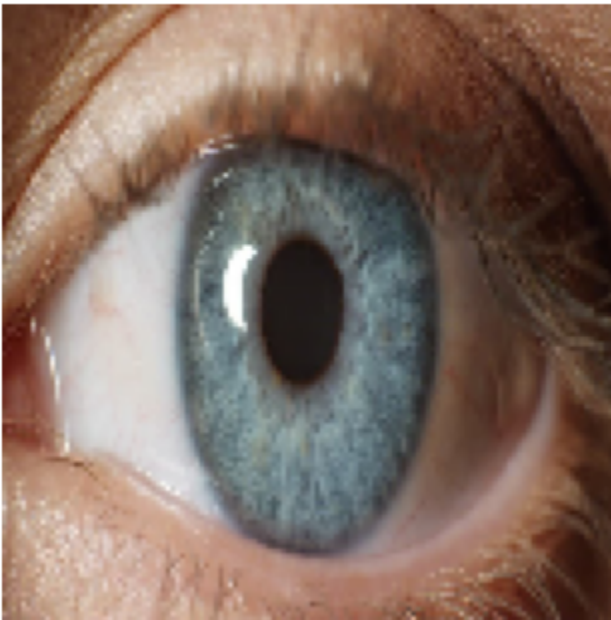
```
pred_idx = int(np.argmax(probs))
label = index2label[pred_idx]
conf = float(np.max(probs))
print(f"Prediction for {img_path} -> {label} (confidence {conf:.3f})")
plt.figure(figsize=(4,4))
plt.imshow(img)
plt.title(f"{label} ({conf:.2f})")
plt.axis('off')
plt.show()

for p in sample_paths:
    predict_and_show(p)
```

1/1 ————— 0s 135ms/step

Prediction for /content/Eye Images/eye1.jpg -> healthy\_eye (confidence 0.625)

healthy\_eye (0.63)



1/1 ————— 0s 44ms/step

Prediction for /content/Eye Images/eye2.jpg -> infected\_eye (confidence 0.994)

infected\_eye (0.99)

