



```
from google.colab import files
files.upload() # Upload kaggle.json when prompted
```

 Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.  
Saving kaggle.json to kaggle.json  
{'kaggle.json': b'{"username": "tanuiakunapareddi", "key": "a6dd128a47bac00709f8b99405764e8b"}'}

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

```
!kaggle datasets download -d chandrashekarnatesh/poultry-diseases
```

 Dataset URL: <https://www.kaggle.com/datasets/chandrashekarnatesh/poultry-diseases>  
License(s): MIT  
^C

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("chandrashekarnatesh/poultry-diseases")

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

 Path to dataset files: /kaggle/input/poultry-diseases

```
train_path = "/kaggle/input/poultry-diseases/data/data/train"
val_path = "/kaggle/input/poultry-diseases/data/data/val"
test_path = "/kaggle/input/poultry-diseases/data/data/test"
```

```
train_path[5]
```


 'l'

```
# Basic Python
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Image processing
import cv2
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
```


```
import os
```

```
# List subfolders (i.e., class names)
class_folders = os.listdir(train_path)
print("Classes found in train_path:")
print(class_folders)
```

 Classes found in train\_path:  
['Coccidiosis', 'Healthy', 'Salmonella', 'New Castle Disease']

```
sample_class = os.path.join(train_path, class_folders[0]) # or manually replace with a class name
image_files = os.listdir(sample_class)
```

```
print(f"Sample images in class '{class_folders[0]}':")
print(image_files[:5])
```

 Sample images in class 'Coccidiosis':  
['cocci.741.jpg\_aug28.JPG', 'cocci.2045.jpg\_aug34.JPG', 'cocci.975.jpg\_aug50.JPG', 'cocci.104.jpg', 'cocci.610.jpg\_aug47.JPG']

```
def read_data(folder, image_size=(224, 224)):
    data = []
    labels = []
    paths = []
    class_names = sorted(os.listdir(folder))
```

```

for class_name in class_names:
    class_path = os.path.join(folder, class_name)
    image_files = os.listdir(class_path)[:500]

    for image_name in image_files:
        img_path = os.path.join(class_path, image_name)
        img = cv2.imread(img_path)
        if img is not None:
            img = cv2.resize(img, image_size)
            data.append(img)
            labels.append(class_name)
            paths.append(img_path)

return data, labels, paths, class_names

X_train, y_train, path_train, class_names = read_data(train_path)
X_val, y_val, path_val, _ = read_data(val_path)
X_test, y_test, path_test, _ = read_data(test_path)

import pandas as pd

train_df = pd.DataFrame({
    "image": X_train,
    "label": y_train,
    "path": path_train
})

train_df = pd.DataFrame({
    "image": X_train,
    "label": y_train,
    "path": path_train
})

val_df = pd.DataFrame({
    "image": X_val,
    "label": y_val,
    "path": path_val
})

test_df = pd.DataFrame({
    "image": X_test,
    "label": y_test,
    "path": path_test
})

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Initialize ImageDataGenerator
gen = ImageDataGenerator()

# Train generator
train_gen = gen.flow_from_dataframe(
    train_df,
    x_col='path',
    y_col='label',
    target_size=(224, 224),
    seed=123,
    class_mode='categorical',
    color_mode='rgb',
    shuffle=True,
    batch_size=32
)

# Validation generator
val_gen = gen.flow_from_dataframe(
    val_df,
    x_col='path',
    y_col='label',
    target_size=(224, 224),
    seed=123,
    class_mode='categorical',
    color_mode='rgb',
    shuffle=True,
    batch_size=32
)


# Test generator
test_gen = gen.flow_from_dataframe(

```

```

test_df,
x_col='path',
y_col='label',
target_size=(224, 224),
seed=123,
class_mode='categorical',
color_mode='rgb',
shuffle=False, # Keep False for evaluation
batch_size=32
)

```

 Found 2000 validated image filenames belonging to 4 classes.  
 Found 2000 validated image filenames belonging to 4 classes.  
 Found 2000 validated image filenames belonging to 4 classes.

```

from tensorflow.keras.layers import Dense, Flatten, Dropout, BatchNormalization, GlobalAveragePooling2D
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model

```

```

# Load VGG16 base model without top layers
vgg = VGG16(input_shape=(224, 224, 3), weights='imagenet', include_top=False)

```

```

# Freeze all pretrained layers
for layer in vgg.layers:
    layer.trainable = False

```

```

# Add custom classification layers
x = vgg.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(512, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
predictions = Dense(4, activation='softmax')(x) # change 4 if your classes differ

```

```

# Final model
model = Model(inputs=vgg.input, outputs=predictions)

```

```

# Compile
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

```

```

# Callbacks
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

```

```

early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)

```

```

reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.2,
    patience=3,
    min_lr=1e-5
)

```

 Downloading data from [https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16\\_weights\\_tf\\_dim\\_ordering\\_tf\\_kernels\\_n\\_58889256/58889256](https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_n_58889256/58889256)  1s 0us/step

```

model.summary()

```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 512)	0
dense (Dense)	(None, 1024)	525,312
batch_normalization (BatchNormalization)	(None, 1024)	4,096
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 512)	524,800
batch_normalization_1 (BatchNormalization)	(None, 512)	2,048
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 4)	2,052

```
from tensorflow.keras.optimizers.legacy import Adam
```

```
# Compile the model
model.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)
```

```
# Train the model
r = model.fit(
    train_gen,
    validation_data=val_gen,
    epochs=1,
    callbacks=[early_stopping, reduce_lr]
)
```

63/63 ————— 2997s 48s/step - accuracy: 0.4106 - loss: 1.9048 - val\_accuracy: 0.5660 - val\_loss: 1.5207 - learning\_rate

```
from tensorflow.keras.applications import VGG19
from tensorflow.keras.layers import Flatten, Dense
from tensorflow.keras.models import Model
```

```
# Load VGG19 without top layers
```

```

vgg19 = VGG19(input_shape=(224, 224, 3), weights='imagenet', include_top=False)

# Freeze pretrained layers
for layer in vgg19.layers:
    layer.trainable = False

# Add custom layers
x = Flatten()(vgg19.output)
prediction = Dense(4, activation='softmax')(x) # change 4 if your class count differs

# Define model
model1 = Model(inputs=vgg19.input, outputs=prediction)

# Compile
model1.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)

```

```

from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import Flatten, Dense
from tensorflow.keras.models import Model

```

```

# Load ResNet50 without top layers
res = ResNet50(input_shape=(224, 224, 3), weights='imagenet', include_top=False)

```

```

# Freeze pretrained layers
for layer in res.layers:
    layer.trainable = False

```

```

# Add custom layers
x = Flatten()(res.output)
prediction = Dense(4, activation='softmax')(x) # adjust '4' to number of classes

```

```


# Build model
model2 = Model(inputs=res.input, outputs=prediction)

```

```

# Compile model
model2.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)

```

 Downloading data from [https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50\\_weights\\_tf\\_dim\\_ordering\\_tf\\_kernels/94765736/94765736](https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels/94765736/94765736) 8s 0us/step



Start coding or [generate](#) with AI.

```

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

```

```

def predictor(model, test_gen, sample_size=200):
    # Get class labels
    classes = list(test_gen.class_indices.keys())
    class_count = len(classes)

    # Predict on a limited number of test samples
    X = []
    y_true = []

    # Get sample_size images from generator
    for i in range(sample_size // test_gen.batch_size + 1):
        batch_imgs, batch_labels = next(test_gen)
        X.append(batch_imgs)
        y_true.append(np.argmax(batch_labels, axis=1))
        if len(np.concatenate(y_true)) >= sample_size:
            break

    # Trim to exact sample_size
    X = np.concatenate(X)[:sample_size]
    y_true = np.concatenate(y_true)[:sample_size]

    # Predict
    preds = model.predict(X)

```

```
y_pred = np.argmax(preds, axis=1)

# Accuracy
correct = np.sum(y_pred == y_true)
accuracy = (correct / sample_size) * 100
print(f"\n✅ Accuracy on {sample_size} test samples: {accuracy:.2f}%")

# Confusion matrix
cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=classes, yticklabels=classes)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Classification report
clr = classification_report(y_true, y_pred, target_names=classes, digits=4)
print("Classification Report:\n", clr)
```

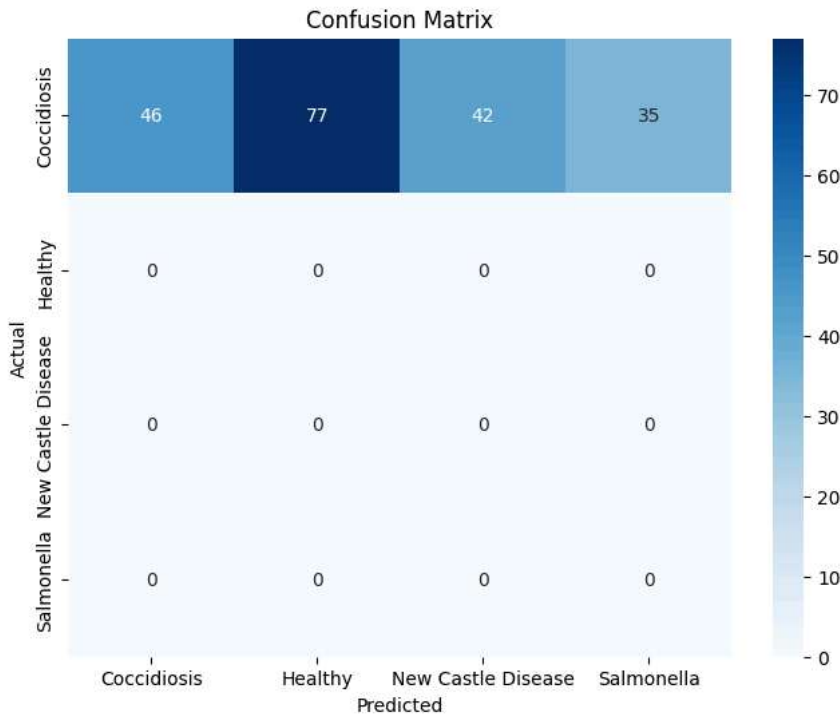
```
predictor(model2, test_gen, sample_size=200) # You can change sample_size
```

🔄

7/7

60s 8s/step

✅ Accuracy on 200 test samples: 23.00%



Classification Report:

	precision	recall	f1-score	support
Coccidiosis	1.0000	0.2300	0.3740	200
Healthy	0.0000	0.0000	0.0000	0
New Castle Disease	0.0000	0.0000	0.0000	0
Salmonella	0.0000	0.0000	0.0000	0
accuracy			0.2300	200
macro avg	0.2500	0.0575	0.0935	200
weighted avg	1.0000	0.2300	0.3740	200

```
from tensorflow.keras.layers import GlobalAveragePooling2D
```

```
pip install keras-tuner
```

🔄

Collecting keras-tuner

Downloading keras\_tuner-1.4.7-py3-none-any.whl.metadata (5.4 kB)

Requirement already satisfied: keras in /usr/local/lib/python3.11/dist-packages (from keras-tuner) (3.8.0)

Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from keras-tuner) (24.2)

Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from keras-tuner) (2.32.3)

Collecting kt-legacy (from keras-tuner)

Downloading kt\_legacy-1.0.5-py3-none-any.whl.metadata (221 bytes)

```

Requirement already satisfied: absl-py in /usr/local/lib/python3.11/dist-packages (from keras->keras-tuner) (1.4.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from keras->keras-tuner) (2.0.2)
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (from keras->keras-tuner) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages (from keras->keras-tuner) (0.1.0)
Requirement already satisfied: h5py in /usr/local/lib/python3.11/dist-packages (from keras->keras-tuner) (3.14.0)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages (from keras->keras-tuner) (0.16.0)
Requirement already satisfied: ml-dtypes in /usr/local/lib/python3.11/dist-packages (from keras->keras-tuner) (0.4.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->keras-tuner) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->keras-tuner) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->keras-tuner) (2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->keras-tuner) (2025.6.15)
Requirement already satisfied: typing-extensions>=4.6.0 in /usr/local/lib/python3.11/dist-packages (from optree->keras->keras-tuner) (4.12.2)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras->keras-tuner) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras->keras-tuner) (2.19.1)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich->keras->keras-tuner) (0.1.2)
Downloading keras_tuner-1.4.7-py3-none-any.whl (129 kB)
129.1/129.1 kB 4.1 MB/s eta 0:00:00
Downloading kt_legacy-1.0.5-py3-none-any.whl (9.6 kB)
Installing collected packages: kt-legacy, keras-tuner
Successfully installed keras-tuner-1.4.7 kt-legacy-1.0.5

```

```
import keras_tuner as kt
```

```

from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam

```

```

# Set predefined hyperparameters (you can change these)
UNITS = 512
LEARNING_RATE = 0.0005
NUM_CLASSES = len(train_gen.class_indices)

```

```

# Load base model
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
base_model.trainable = False # Freeze ResNet layers

```

```

# Add custom head
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(UNITS, activation='relu')(x)
x = Dropout(0.5)(x)
output = Dense(NUM_CLASSES, activation='softmax')(x)

```

```

# Build model
model = Model(inputs=base_model.input, outputs=output)

```

```

# Compile
model.compile(
    optimizer=Adam(learning_rate=LEARNING_RATE),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

```

```

# Train
history = model.fit(
    train_gen,
    validation_data=val_gen,
    epochs=3,
    steps_per_epoch=5,
    validation_steps=2
)

```

```

➡ Epoch 1/3
5/5 ————— 132s 18s/step - accuracy: 0.2173 - loss: 1.9695 - val_accuracy: 0.4688 - val_loss: 1.3223
Epoch 2/3
5/5 ————— 57s 12s/step - accuracy: 0.3875 - loss: 1.6056 - val_accuracy: 0.5625 - val_loss: 1.0506
Epoch 3/3
5/5 ————— 81s 18s/step - accuracy: 0.5188 - loss: 1.5028 - val_accuracy: 0.3750 - val_loss: 1.3015

```

```
train_gen.class_indices.keys() # This shows the class labels as dict_keys([...])
```

```
labels = ['Coccidiosis', 'Healthy', 'New Castle Disease', 'Salmonella']
```

```
from tensorflow.keras.preprocessing.image import load_img, img_to_array
```

```

def get_model_prediction(image_path):
    img = load_img(image_path, target_size=(224, 224))
    x = img_to_array(img)
    x = np.expand_dims(x, axis=0)

```

```
predictions = model.predict(x, verbose=0)
return labels[predictions.argmax()]
```

```
get_model_prediction('/kaggle/input/poultry-diseases/data/data/test/Salmonella/pcrsalmo.111.jpg_aug29.JPG')
```

```
↕ 'Salmonella'
```

```
model.save('poultry_model.h5') # Save the trained model
```

```
↕ WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
```

```
from google.colab import files
files.download('poultry_model.h5')
```

```
↕
```

Start coding or [generate](#) with AI.