Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

```
import os
import pandas as pd
import numpy as np
import cv2
from sklearn.model_selection import train_test_split

# Define paths
base_path = '/kaggle/input/indian-emergency-vehicles-dataset'  # Change this to your folder path
folders = ['train', 'test', 'valid']

# Initialize lists for data and labels
data = []
labels = []

# Desired classes
desired_classes = ['ambulance', 'firetruck', 'police vehicle']

# Load and filter data
for folder in folders:
    csv_path = os.path.join(base_path, folder, '_classes.csv')  # Replace '_classes.csv' with your
    df = pd.read_csv(csv_path)

    # Clean the column names
    df.columns = df.columns.str.strip()

    # Select relevant columns
    filtered_df = df[['filename'] + desired_classes]

    # Filter rows where any desired class has a value of 1
    filtered_df = filtered_df[filtered_df[desired_classes].any(axis=1)]

    for index, row in filtered_df.iterrows():
        filename = row['filename']
        image_path = os.path.join(base_path, folder, filename)
        image = cv2.imread(image_path)

        if image is not None:  # Ensure the image was read successfully
            image = cv2.resize(image, (128, 128))  # Resize to a fixed size
            data.append(image)

            # Create a label vector for the classes
            label_vector = [int(row[class_name] == 1) for class_name in desired_classes]
            labels.append(label_vector)
```

```
data = np.array(data)
data = data.astype('float32') / 255.0

# Convert labels to numpy array of type float32
labels = np.array(labels, dtype=np.float32)

# Split train and validation data
X_train, X_val, y_train, y_val = train_test_split(data, labels, test_size=0.2, random_state=42)

print(f"Training data shape: {X_train.shape}, Training labels shape: {y_train.shape}")
print(f"Validation data shape: {X_val.shape}, Validation labels shape: {y_val.shape}")
```

```
Training data shape: (4166, 128, 128, 3), Training labels shape: (4166, 3)
Validation data shape: (1042, 128, 128, 3), Validation labels shape: (1042, 3)
```

```
import tensorflow as tf
from tensorflow.keras import layers, models

# Define a simple CNN model
def create_cnn_model(input_shape, num_classes):
    model = models.Sequential()

    # Convolutional Layer 1
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
    model.add(layers.MaxPooling2D(pool_size=(2, 2)))

    # Convolutional Layer 2
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D(pool_size=(2, 2)))

    # Convolutional Layer 3
    model.add(layers.Conv2D(128, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D(pool_size=(2, 2)))

    # Flatten the output
    model.add(layers.Flatten())

    # Fully Connected Layer
    model.add(layers.Dense(128, activation='relu'))
    model.add(layers.Dropout(0.5))  # Dropout for regularization
    model.add(layers.Dense(num_classes, activation='sigmoid'))  # Sigmoid for multi-label classifi

    return model

# Set parameters
input_shape = (128, 128, 3)  # Input image size
num_classes = len(desired_classes)  # Number of classes

# Create the model
cnn_model = create_cnn_model(input_shape, num_classes)

# Compile the model
cnn_model.compile(optimizer='adam',
                  loss='binary_crossentropy',  # Binary crossentropy for multi-label
                  metrics=['accuracy'])
```

```
# Summary of the model
cnn_model.summary()
```

```
/opt/conda/lib/python3.10/site-packages/keras/src/layers/convolutional/base_conv.py:107: UserWa
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential_8"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━┓
┃ Layer (type)                     ┃ Output Shape           ┃        Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━┩
│ conv2d_28 (Conv2D)               │ (None, 126, 126, 32)   │            896 │
├──────────────────────────────────┼────────────────────────┼────────────────┤
│ max_pooling2d_28 (MaxPooling2D)  │ (None, 63, 63, 32)     │              0 │
├──────────────────────────────────┼────────────────────────┼────────────────┤
│ conv2d_29 (Conv2D)               │ (None, 61, 61, 64)     │         18,496 │
├──────────────────────────────────┼────────────────────────┼────────────────┤
│ max_pooling2d_29 (MaxPooling2D)  │ (None, 30, 30, 64)     │              0 │
├──────────────────────────────────┼────────────────────────┼────────────────┤
│ conv2d_30 (Conv2D)               │ (None, 28, 28, 128)    │         73,856 │
├──────────────────────────────────┼────────────────────────┼────────────────┤
│ max_pooling2d_30 (MaxPooling2D)  │ (None, 14, 14, 128)    │              0 │
├──────────────────────────────────┼────────────────────────┼────────────────┤
│ flatten_8 (Flatten)              │ (None, 25088)          │              0 │
├──────────────────────────────────┼────────────────────────┼────────────────┤
│ dense_16 (Dense)                 │ (None, 128)            │      3,211,392 │
├──────────────────────────────────┼────────────────────────┼────────────────┤
│ dropout_8 (Dropout)              │ (None, 128)            │              0 │
├──────────────────────────────────┼────────────────────────┼────────────────┤
│ dense_17 (Dense)                 │ (None, 3)              │            387 │
└──────────────────────────────────┴────────────────────────┴────────────────┘
 Total params: 3,305,027 (12.61 MB)
 Trainable params: 3,305,027 (12.61 MB)
 Non-trainable params: 0 (0.00 B)
```

```
# Train the model
history = cnn_model.fit(X_train, y_train,
                        validation_data=(X_val, y_val),
                        epochs=20,
                        batch_size=32)
```

```
Epoch 1/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step - accuracy: 0.5528 - loss: 0.5916WARNING: All log mes
I0000 00:00:1726921655.652440     113 asm_compiler.cc:369] ptxas warning : Registers are spille

131/131 ━━━━━━━━━━━━━━━━━━━━ 11s 53ms/step - accuracy: 0.5532 - loss: 0.5912 - val_accuracy: 0.
Epoch 2/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.6403 - loss: 0.4762 - val_accuracy: 0.6
Epoch 3/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.6802 - loss: 0.4241 - val_accuracy: 0.7
Epoch 4/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.7495 - loss: 0.3602 - val_accuracy: 0.7
Epoch 5/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.8066 - loss: 0.2942 - val_accuracy: 0.7
Epoch 6/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.8490 - loss: 0.2449 - val_accuracy: 0.7
Epoch 7/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.8801 - loss: 0.1909 - val_accuracy: 0.7
Epoch 8/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.9142 - loss: 0.1523 - val_accuracy: 0.8
```

```
Epoch 9/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.9496 - loss: 0.1055 - val_accuracy: 0.8
Epoch 10/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.9509 - loss: 0.0943 - val_accuracy: 0.8
Epoch 11/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.9690 - loss: 0.0629 - val_accuracy: 0.8
Epoch 12/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.9721 - loss: 0.0593 - val_accuracy: 0.8
Epoch 13/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.9851 - loss: 0.0387 - val_accuracy: 0.8
Epoch 14/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.9786 - loss: 0.0461 - val_accuracy: 0.8
Epoch 15/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.9871 - loss: 0.0331 - val_accuracy: 0.8
Epoch 16/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.9829 - loss: 0.0431 - val_accuracy: 0.8
Epoch 17/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.9866 - loss: 0.0346 - val_accuracy: 0.8
Epoch 18/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.9843 - loss: 0.0317 - val_accuracy: 0.8
Epoch 19/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.9863 - loss: 0.0269 - val_accuracy: 0.8
Epoch 20/20
131/131 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.9879 - loss: 0.0284 - val_accuracy: 0.8
```

```
Start coding or generate with AI.
```

```
import numpy as np
import cv2
import random
import matplotlib.pyplot as plt

def predict_random_image(model, base_path, folder):
    # Select a random image from the specified folder
    csv_path = os.path.join(base_path, folder, '_classes.csv')  # Replace 'data.csv' with your actu
    df = pd.read_csv(csv_path)
    random_index = random.randint(0, len(df) - 1)
    random_row = df.iloc[random_index]

    filename = random_row['filename']  # Assuming the first column is 'filename'
    image_path = os.path.join(base_path, folder, filename)

    # Load and preprocess the image
    image = cv2.imread(image_path)
    image_resized = cv2.resize(image, (128, 128))  # Resize to match model input
    image_array = np.array(image_resized, dtype='float32') / 255.0
    image_array = np.expand_dims(image_array, axis=0)  # Add batch dimension

    # Predict using the model
    predictions = model.predict(image_array)[0]

    # Get class names for predicted labels
    predicted_classes = [class_names[i] for i in range(len(predictions)) if predictions[i] > 0.5]

    # Display the image and predicted labels
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.title(f'Filename: {filename}\nPredicted Classes: {predicted_classes}')
    plt.show()

    return predicted_classes


# Call the function to predict a random image
predicted_classes = predict_random_image(model, base_path, 'valid')  # Change 'test' if needed
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
```

Filename: ambulance-453-_jpg.rf.03ac844e6867395b8b011ed79a9fbe40.jpg
Predicted Classes: ['ambulance']

```
def predict_custom_image(model, image_path):
    # Load and preprocess the custom image
    image = cv2.imread(image_path)
    image_resized = cv2.resize(image, (128, 128))  # Resize to match model input
    image_array = np.array(image_resized, dtype='float32') / 255.0
    image_array = np.expand_dims(image_array, axis=0)  # Add batch dimension

    # Predict using the model
    predictions = model.predict(image_array)[0]

    # Get class names for predicted labels
    predicted_classes = [desired_classes[i] for i in range(len(predictions)) if predictions[i] > 0

    # Display the image and predicted labels
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.title(f'Predicted Classes: {predicted_classes}')
    plt.show()

    return predicted_classes

# Example usage with a custom image path
```

```
custom_image_path = '/kaggle/input/ambulance/ambulance5.webp'  # Change this to your custom image p
predicted_classes = predict_custom_image(cnn_model, custom_image_path)
```

```
# cnn_model.save('final_model.keras')
```

```
# Assuming you're in the environment where the model works
cnn_model.save('cnn3class.h5')  # Use HDF5 format
```

```
import cv2
import numpy as np
from keras.models import load_model
import matplotlib.pyplot as plt

# Load your model
model = load_model('final_model.keras')

# Define the classes
desired_classes = ['ambulance', 'firetruck', 'police vehicle']  # Adjust as needed

def predict_and_display(frame):
    # Preprocess the frame
    image_resized = cv2.resize(frame, (128, 128))  # Resize to match model input
    image_array = np.array(image_resized, dtype='float32') / 255.0
    image_array = np.expand_dims(image_array, axis=0)  # Add batch dimension

    # Predict using the model
    predictions = model.predict(image_array)[0]

    # Get class names for predicted labels
    predicted_classes = [desired_classes[i] for i in range(len(predictions)) if predictions[i] > 0.

    return predicted_classes

# Open the video file
video_path = '/kaggle/input/ambulance/Police Cars Fire Trucks And Ambulances Responding Compilatio
cap = cv2.VideoCapture(video_path)

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    # Get predictions for the current frame
    predicted_classes = predict_and_display(frame)

    # Display predictions on the frame
    cv2.putText(frame, f'Predicted Classes: {predicted_classes}', (10, 30), cv2.FONT_HERSHEY_SIMPL

    # Show the frame
    cv2.imshow('Video Prediction', frame)
    # Break the loop on 'q' key press
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release resources
cap.release()
cv2.destroyAllWindows()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 320ms/step
---------------------------------------------------------------------------
error                                     Traceback (most recent call last)
Cell In[65], line 42
     39 cv2.putText(frame, f'Predicted Classes: {predicted_classes}', (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
     41 # Show the frame
---> 42 cv2.imshow('Video Prediction', frame)
     44 # Break the loop on 'q' key press
     45 if cv2.waitKey(1) & 0xFF == ord('q'):

error: OpenCV(4.10.0) /io/opencv/modules/highgui/src/window.cpp:1301: error: (-2:Unspecified
error) The function is not implemented. Rebuild the library with Windows, GTK+ 2.x or Cocoa
support. If you are on Ubuntu or Debian, install libgtk2.0-dev and pkg-config, then re-run
cmake or configure script in function 'cvShowImage'
```

```
import cv2
import numpy as np
from keras.models import load_model
import matplotlib.pyplot as plt

# Load your model
model = load_model('final_model.keras')

# Define the classes
desired_classes = ['ambulance', 'firetruck', 'police vehicle']  # Adjust as needed

def predict_and_display(frame):
    # Preprocess the frame
    image_resized = cv2.resize(frame, (128, 128))  # Resize to match model input
    image_array = np.array(image_resized, dtype='float32') / 255.0
    image_array = np.expand_dims(image_array, axis=0)  # Add batch dimension

    # Predict using the model
    predictions = model.predict(image_array)[0]

    # Get class names for predicted labels
    predicted_classes = [desired_classes[i] for i in range(len(predictions)) if predictions[i] > 0.

    return predicted_classes

# Open the video file
video_path = '/kaggle/input/ambulance/Police Cars Fire Trucks And Ambulances Responding Compilation
cap = cv2.VideoCapture(video_path)

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    # Get predictions for the current frame
    predicted_classes = predict_and_display(frame)

    # Display predictions on the frame
    cv2.putText(frame, f'Predicted Classes: {predicted_classes}', (10, 30), cv2.FONT_HERSHEY_SIMPLE

    # Convert BGR frame to RGB for Matplotlib
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # Show the frame using Matplotlib
    plt.imshow(frame_rgb)
    plt.axis('off')
    plt.title(f'Predicted Classes: {predicted_classes}')
    plt.pause(0.01)  # Pause to update the plot

    # Break the loop on 'q' key press (this part won't work as expected with Matplotlib)
    if plt.waitforbuttonpress(0.01):
        break

# Release resources
cap.release()
plt.close()
```

Start coding or generate with AI.