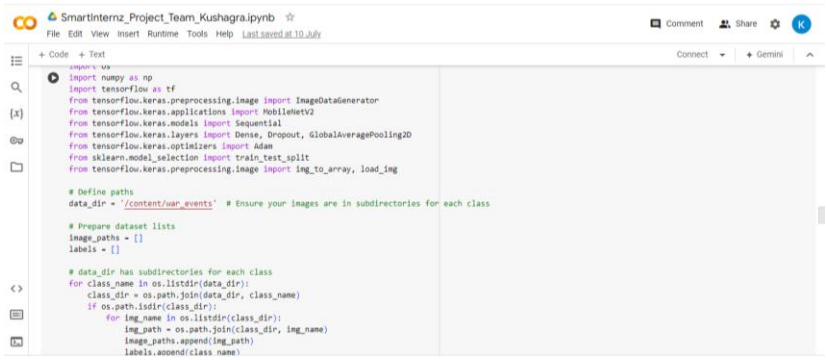
Data Collection and Preprocessing Phase

Date	11 July 2024
Team ID	SWTID1720012105
Project Title	WarLens: Transfer Learning for Event Classification in Conflict Zones
Maximum Marks	6 Marks

Preprocessing Template

The images will be preprocessed by resizing, normalizing, augmenting, denoising, adjusting contrast, detecting edges, converting color space, cropping, batch normalizing, and whitening data. These steps will enhance data quality, promote model generalization, and improve convergence during neural network training, ensuring robust and efficient performance across various computer vision tasks.

Section	**Description**
Data Overview	We are using a Kaggle dataset name war events with over 84,151,000 images which are having various types like fire,combat, DestroyedBuildings ,humanatarian ai and military vehicles and weapons.
Resizing	Resize images to a specified target size.
Normalization	Normalize pixel values to a specific range.
Data Augmentation	Apply augmentation techniques such as flipping, rotation, shifting, zooming, or shearing.
Denoising	Apply denoising filters to reduce noise in the images.
Edge Detection	Apply edge detection algorithms to highlight prominent edges in the images.

Color Space Conversion	Convert images from one color space to another.
Image Cropping	Crop images to focus on the regions containing objects of interest.
Batch Normalization	Apply batch normalization to the input of each layer in the neural network.
Data Preprocessing Code Screenshots	
Loading Data	
Resizing	
Normalization	

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import img_to_array, load_img

# Define paths
data_dir = '/content/war_events'  # Ensure your images are in subdirectories for each class

# Prepare dataset lists
image_paths = []
labels = []

# data_dir has subdirectories for each class
for class_name in os.listdir(data_dir):
    class_dir = os.path.join(data_dir, class_name)
    if os.path.isdir(class_dir):
        for img_name in os.listdir(class_dir):
            img_path = os.path.join(class_dir, img_name)
            image_paths.append(img_path)
            labels.append(class_name)
```

```
# Prepare dataset lists
image_paths = []
labels = []

# data_dir has subdirectories for each class
for class_name in os.listdir(data_dir):
    class_dir = os.path.join(data_dir, class_name)
    if os.path.isdir(class_dir):
        for img_name in os.listdir(class_dir):
            img_path = os.path.join(class_dir, img_name)
            image_paths.append(img_path)
            labels.append(class_name)

# Convert labels to numerical values
label_to_index = {label: idx for idx, label in enumerate(set(labels))}
labels = [label_to_index[label] for label in labels]

# Split the dataset
train_paths, val_paths, train_labels, val_labels = train_test_split(image_paths, labels, test_size=0.2, stratify=labels)

# Data augmentation and preprocessing
def preprocess_image(img_path):
    img = load_img(img_path, target_size=(224, 224))
    img_array = img_to_array(img) / 255.0
    return img_array
```

```
# Prepare dataset lists
image_paths = []
labels = []

# data_dir has subdirectories for each class
for class_name in os.listdir(data_dir):
    class_dir = os.path.join(data_dir, class_name)
    if os.path.isdir(class_dir):
        for img_name in os.listdir(class_dir):
            img_path = os.path.join(class_dir, img_name)
            image_paths.append(img_path)
            labels.append(class_name)

# Convert labels to numerical values
label_to_index = {label: idx for idx, label in enumerate(set(labels))}
labels = [label_to_index[label] for label in labels]

# Split the dataset
train_paths, val_paths, train_labels, val_labels = train_test_split(image_paths, labels, test_size=0.2, stratify=labels)

# Data augmentation and preprocessing
def preprocess_image(img_path):
    img = load_img(img_path, target_size=(224, 224))
    img_array = img_to_array(img) / 255.0
    return img_array
```

Data Augmentation	SmartInternz_Project_Team_Kushagra.ipynb
	File Edit View Insert Runtime Tools Help Last saved at 10 July
	+ Code + Text
	Comment Share Connect Gemini
	``python
	# Data augmentation and preprocessing
	def preprocess_image(img_path):
	img = load_img(img_path, target_size=(224, 224))
	img_array = img_to_array(img) / 255.0
	return img_array
	
	def data_generator(paths, labels, batch_size, is_train):
	data_gen = ImageDataGenerator(
	rotation_range=20,
	width_shift_range=0.2,
	height_shift_range=0.2,
	shear_range=0.2,
	zoom_range=0.2,
	horizontal_flip=True,
	fill_mode='nearest'
	) if is_train else ImageDataGenerator(rescale=1./255)
	
	while True:
	for start in range(0, len(paths), batch_size):
	end = min(start + batch_size, len(paths))
	batch_paths = paths[start:end]
	batch_labels = labels[start:end]
	
	batch_images = np.array([preprocess_image(img_path) for img_path in batch_paths])
	``
	Last saved at 10 July

Denoising	SmartInternz_Project_Team_Kushagra.ipynb
	File Edit View Insert Runtime Tools Help Last saved at 10 July
	+ Code + Text
	Comment Share Connect Gemini
	``python
	train_gen = data_generator(train_paths, train_labels, batch_size, is_train=True)
	val_gen = data_generator(val_paths, val_labels, batch_size, is_train=False)
	
	# Load pre-trained MobileNetV2 model + higher level layers
	mobilenet_model = MobileNetV2(input_shape=(224, 224, 3), include_top=False, weights='imagenet')
	
	# Freeze the pretrained layers
	mobilenet_model.trainable = False
	
	# Create a new model on top
	model_new1 = Sequential([
	mobilenet_model,
	GlobalAveragePooling2D(),
	Dense(128, activation='relu'),
	Dropout(0.5),
	Dense(len(label_to_index), activation='softmax') # Adjust the number of classes dynamically
	])
	
	# Compile the model
	model_new1.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
	
	# Train the model
	history_new1 = model_new1.fit(
	train_gen,
	``
	Last saved at 10 July

Edge Detection	SmartInternz_Project_Team_Kushagra.ipynb
	File Edit View Insert Runtime Tools Help Last saved at 10 July
	+ Code + Text
	Comment Share Connect Gemini
	``python
	train_gen = data_generator(train_paths, train_labels, batch_size, is_train=True)
	val_gen = data_generator(val_paths, val_labels, batch_size, is_train=False)
	
	# Load pre-trained MobileNetV2 model + higher level layers
	mobilenet_model = MobileNetV2(input_shape=(224, 224, 3), include_top=False, weights='imagenet')
	
	# Freeze the pretrained layers
	mobilenet_model.trainable = False
	
	# Create a new model on top
	model_new1 = Sequential([
	mobilenet_model,
	GlobalAveragePooling2D(),
	Dense(128, activation='relu'),
	Dropout(0.5),
	Dense(len(label_to_index), activation='softmax') # Adjust the number of classes dynamically
	])
	
	# Compile the model
	model_new1.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
	
	# Train the model
	history_new1 = model_new1.fit(
	train_gen,
	``
	Last saved at 10 July

Color Space Conversion	SmartInternz_Project_Team_Kushagra.ipynb
	File Edit View Insert Runtime Tools Help Last saved at 10 July
	+ Code + Text
	Comment Share Connect Gemini
	``python
	train_gen = data_generator(train_paths, train_labels, batch_size, is_train=True)
	val_gen = data_generator(val_paths, val_labels, batch_size, is_train=False)
	
	# Load pre-trained MobileNetV2 model + higher level layers
	mobilenet_model = MobileNetV2(input_shape=(224, 224, 3), include_top=False, weights='imagenet')
	
	# Freeze the pretrained layers
	mobilenet_model.trainable = False
	
	# Create a new model on top
	model_new1 = Sequential([
	mobilenet_model,
	GlobalAveragePooling2D(),
	Dense(128, activation='relu'),
	Dropout(0.5),
	Dense(len(label_to_index), activation='softmax') # Adjust the number of classes dynamically
	])
	
	# Compile the model
	model_new1.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
	
	# Train the model
	history_new1 = model_new1.fit(
	train_gen,
	``
	Last saved at 10 July

Image Cropping	SmartInternz_Project_Team_Kushagra.ipynb
	File Edit View Insert Runtime Tools Help Last saved at 10 July
	+ Code + Text
	Comment Share Connect Gemini
	train_gen = data_generator(train_paths, train_labels, batch_size, is_train=True)
	val_gen = data_generator(val_paths, val_labels, batch_size, is_train=False)
	
	# Load pre-trained MobileNetV2 model + higher level layers
	mobilenet_model = MobileNetV2(input_shape=(224, 224, 3), include_top=False, weights='imagenet')
	
	# Freeze the pretrained layers
	mobilenet_model.trainable = False
	
	# Create a new model on top
	model_new1 = Sequential([
	mobilenet_model,
	GlobalAveragePooling2D(),
	Dense(128, activation='relu'),
	Dropout(0.5),
	Dense(len(label_to_index), activation='softmax') # Adjust the number of classes dynamically
	])
	
	# Compile the model
	model_new1.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
	
	# Train the model
	history_new1 = model_new1.fit(
	train_gen,
	.

Batch Normalization	SmartInternz_Project_Team_Kushagra.ipynb
	File Edit View Insert Runtime Tools Help Last saved at 10 July
	+ Code + Text
	Comment Share Connect Gemini
	# Freeze the pretrained layers
	mobilenet_model.trainable = False
	
	# Create a new model on top
	model_new1 = Sequential([
	mobilenet_model,
	GlobalAveragePooling2D(),
	Dense(128, activation='relu'),
	Dropout(0.5),
	Dense(len(label_to_index), activation='softmax') # Adjust the number of classes dynamically
	])
	
	# Compile the model
	model_new1.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
	
	# Train the model
	history_new1 = model_new1.fit(
	train_gen,
	steps_per_epoch=len(train_paths) // batch_size,
	validation_data=val_gen,
	validation_steps=len(val_paths) // batch_size,
	epochs=20
	)
	
	# Save the model