Wildfire Detection through Image Analysis
using CNN and MobileNetV2

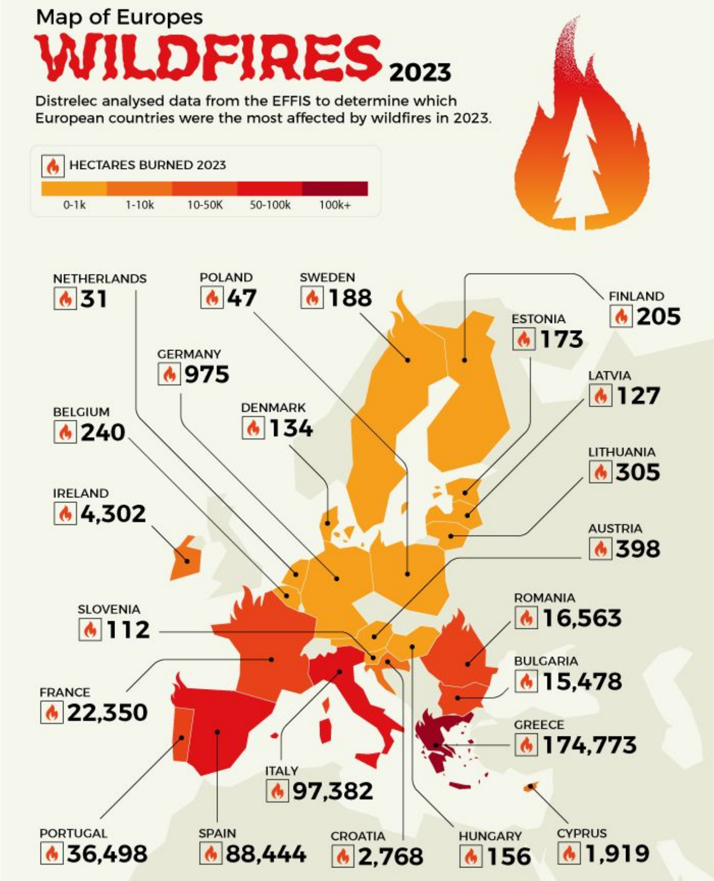
Artificial Intelligence & Machine Learning
Alpen-Adria-University Klagenfurt

Evangelia Panteliadou

Introduction

Introduction

"Welcome to my presentation on wildfire detection using image analysis, where I'll be discussing how machine learning techniques can be leveraged to enhance early detection and response to wildfires."



Motivation

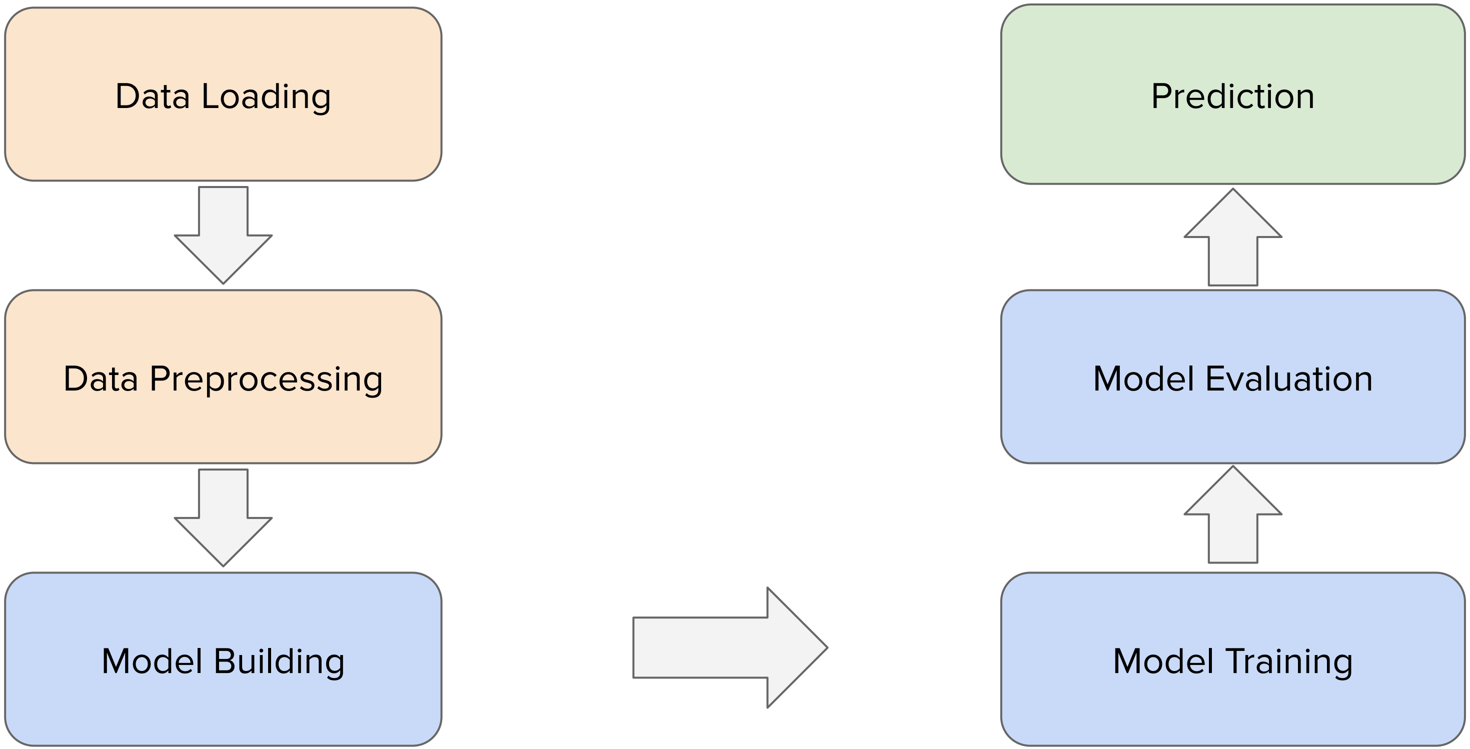
- Personal Experience
- Environmental Impact
- Technological Potential

Evia Island, Greece - Wildfire 2022 - Photo taken by Konstantinos Tsakalidis

Goals

- Primary Objective - an ML model that can accurately detect signs of wildfire
- Key Benefits
    - Timely Alerts - early warnings to local authorities
    - Damage Mitigation - improving preparedness and resource allocation
    - Technological Advancement - solving real-world environmental challenges with the use of AI

Workflow


Data Loading
Data Preprocessing
Model Building
Model Training
Model Evaluation
Prediction

Example of inputs - Fire and smoke class

Example of inputs - No fire and nosmoke class












Learning Task

CNN Model

- Classification of images
  - **fire and/or smoke**
  - **no fire/no smoke**
- Train data
- Validate data
- Test data

Tools and Libraries

- Data Preprocessing
  - NumPy
  - Pillow
- Development Environment
  - Google Colab
- Model Design
  - TensorFlow
  - Keras
  - MobileNetV2
- Data Analysis
  - Pandas
  - Matplotlib
  - Seaborn
- Metrics
  - scikit-learn

Model Training

```
# Model Training
checkpoint = ModelCheckpoint('models/best_model.keras', monitor='val_loss', save_best_only=True, verbose=1)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2, min_lr=0.00001)
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

history = model.fit(
    train_generator,
    steps_per_epoch=steps_per_epoch,
    validation_data=validate_generator,
    validation_steps=validation_steps,
    epochs=5,
    callbacks=[checkpoint, reduce_lr, early_stopping],
    class_weight=class_weights
)
```

Model Training - Results

```
Epoch 1/5
93/93 [==============================] - ETA: 0s - loss: 0.1466 - accuracy: 0.9437
Epoch 1: val_loss improved from inf to 0.12397, saving model to models/best_model.keras
93/93 [==============================] - 708s 8s/step - loss: 0.1466 - accuracy: 0.9437 - val_loss: 0.1240 - val_accuracy: 0.9643 - lr: 1.0000e-04
Epoch 2/5
93/93 [==============================] - ETA: 0s - loss: 0.0881 - accuracy: 0.9700
Epoch 2: val_loss did not improve from 0.12397
93/93 [==============================] - 249s 3s/step - loss: 0.0881 - accuracy: 0.9700 - val_loss: 0.1310 - val_accuracy: 0.9583 - lr: 1.0000e-04
Epoch 3/5
93/93 [==============================] - ETA: 0s - loss: 0.0602 - accuracy: 0.9774
Epoch 3: val_loss improved from 0.12397 to 0.07729, saving model to models/best_model.keras
93/93 [==============================] - 260s 3s/step - loss: 0.0602 - accuracy: 0.9774 - val_loss: 0.0773 - val_accuracy: 0.9821 - lr: 1.0000e-04
Epoch 4/5
93/93 [==============================] - ETA: 0s - loss: 0.0521 - accuracy: 0.9838
Epoch 4: val_loss did not improve from 0.07729
93/93 [==============================] - 259s 3s/step - loss: 0.0521 - accuracy: 0.9838 - val_loss: 0.0804 - val_accuracy: 0.9807 - lr: 1.0000e-04
Epoch 5/5
93/93 [==============================] - ETA: 0s - loss: 0.0552 - accuracy: 0.9842
Epoch 5: val_loss improved from 0.07729 to 0.07381, saving model to models/best_model.keras
93/93 [==============================] - 248s 3s/step - loss: 0.0552 - accuracy: 0.9842 - val_loss: 0.0738 - val_accuracy: 0.9792 - lr: 1.0000e-04
```

Model Evaluation

```python
# Model Evaluation
best_model = tf.keras.models.load_model('models/best_model.keras')

# Ensure that the test set is fully covered
test_steps = np.ceil(test_generator.samples / batch_size).astype(int)

train_generator.reset()
train_loss, train_accuracy = best_model.evaluate(train_generator, steps=train_generator.samples // batch_size)
print(f'Train Accuracy: {train_accuracy * 100:.2f}%')

validate_generator.reset()
val_loss, val_accuracy = best_model.evaluate(validate_generator, steps=validate_generator.samples // batch_size)
print(f'Validation Accuracy: {val_accuracy * 100:.2f}%')

test_generator.reset()
test_loss, test_accuracy = best_model.evaluate(test_generator, steps=test_generator.samples // batch_size)
print(f'Test Accuracy: {test_accuracy * 100:.2f}%')
```

Model Evaluation - Output

```
93/93 [==============================] - 199s 2s/step - loss: 0.0246 - accuracy: 0.9926
Train Accuracy: 99.26%
21/21 [==============================] - 34s 2s/step - loss: 0.0732 - accuracy: 0.9792
Validation Accuracy: 97.92%
70/70 [==============================] - 530s 8s/step - loss: 0.3665 - accuracy: 0.8987
Test Accuracy: 89.87%
```

Preliminary Data Analysis

Data

- Data Sources
  - Combination of different Kaggle datasets
- Data Categories
  - images labeled
    - fire_smoke
    - nofire_nosmoke
- Data Characteristics
  - different times of day, weather conditions, and landscapes
- Visualization
  - initial examples of the data and distribution

Data Augmentation and Image Preprocessing

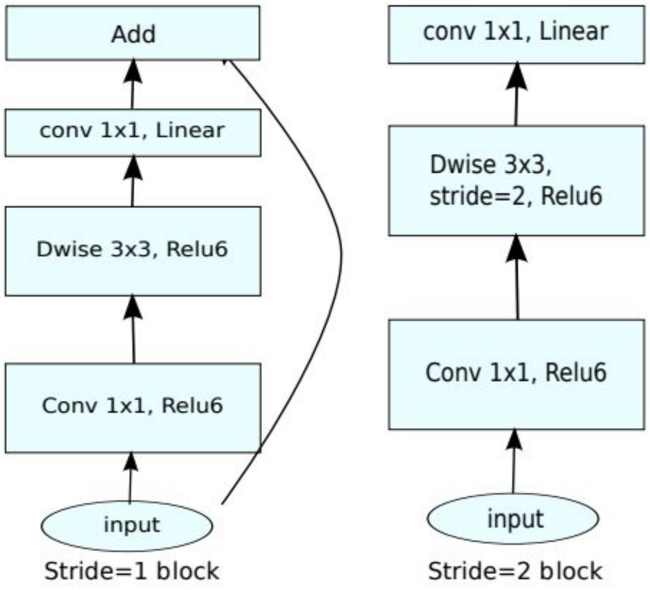
- setting the image height and width for compatibility with MobileNetV2
- using a batch size of 32 for efficient processing
- data augmentation like rescale, rotation, etc. for diversity

```
# Image dimensions
img_height, img_width = 224, 224
batch_size = 32  # Increased batch size for faster processing

# Data Augmentation and Generators
train_datagen = ImageDataGenerator(
    rescale=1./255,
    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'
)

val_test_datagen = ImageDataGenerator(rescale=1./255)
```

Implementation

MobileNetV2 Classification - Architecture

MobileNetV2 is a classification model developed by Google. It provides real-time classification capabilities under computing constraints in devices like smartphones. This implementation leverages transfer learning from ImageNet to a dataset.



(d) Mobilenet V2

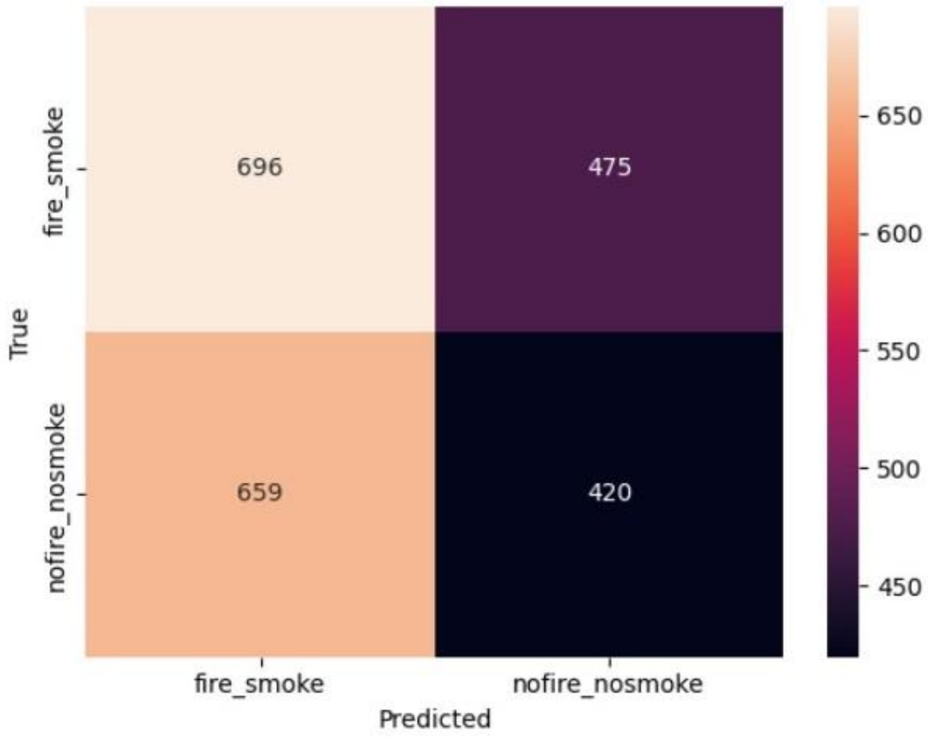
Custom Layers

```
[16] # Load pre-trained MobileNetV2 model + higher level layers
     base_model = tf.keras.applications.MobileNetV2(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

     # Add top layers
     x = base_model.output
     x = Flatten()(x)
     x = Dense(64, activation='relu')(x)
     x = Dropout(0.5)(x)
     predictions = Dense(1, activation='sigmoid')(x)

     # Create final model
     model = Model(inputs=base_model.input, outputs=predictions)

     # Freeze the base_model layers
     for layer in base_model.layers:
         layer.trainable = False

     # Compile the model
     optimizer = Adam(learning_rate=0.0001)  # Reduced learning rate
     model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
     model.summary()
```

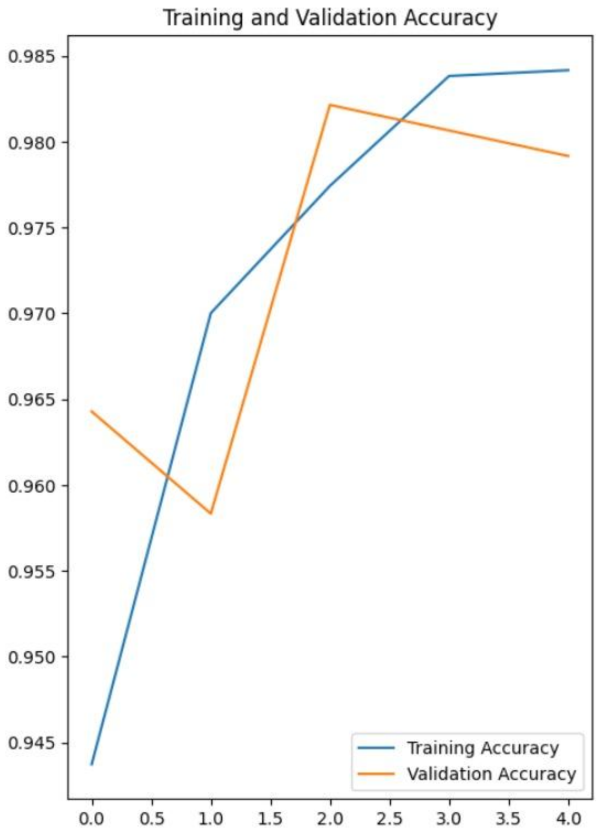
Obtained Results and Analysis

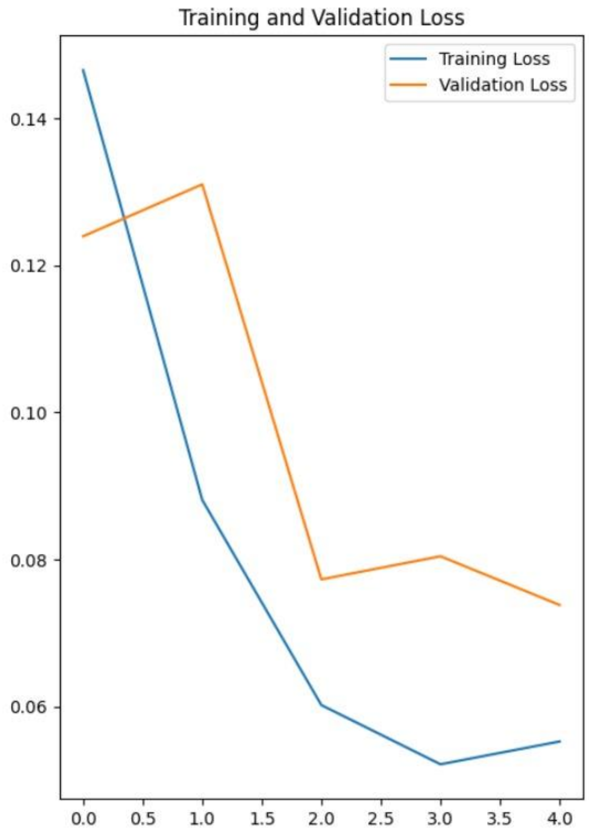
Model Evaluation Metrics

```
71/71 [==============================] - 118s 2s/step
Length of y_true: 2250
Length of y_pred_classes: 2250
Precision: 0.47
Recall: 0.39
F1 Score: 0.43
               precision    recall  f1-score   support

   fire_smoke       0.51      0.59      0.55      1171
nofire_nosmoke       0.47      0.39      0.43      1079

     accuracy                           0.50      2250
    macro avg       0.49      0.49      0.49      2250
 weighted avg       0.49      0.50      0.49      2250
```



Training and Validation Performance





Thank you for your attention!