

```
In [1]: import os
import tensorflow as tf
import numpy as np
from keras.applications import ResNet50
from efficientnet.tfkeras import EfficientNetB7
from keras.models import Sequential, Model
from keras.layers import Input, Conv2D, MaxPooling2D, Dropout, Flatten, Dense, BatchNorm
from keras import regularizers
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.utils import plot_model
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, f1_score, classification_report, accuracy_score
import seaborn as sns
from IPython.display import Image
import warnings
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
warnings.filterwarnings('ignore', category=UserWarning)
```

```
In [2]: from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True
```

```
In [3]: train_path = "./wildfire-dataset/train"
valid_path = "./wildfire-dataset/valid"
test_path = "./wildfire-dataset/test"
```

```
In [4]: train_datagen = ImageDataGenerator(rescale=1./255)
valid_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_path,
    target_size=(350, 350),
    batch_size=32,
    class_mode='binary',
    shuffle=True)

valid_generator = valid_datagen.flow_from_directory(
    valid_path,
    target_size=(350, 350),
    batch_size=32,
    class_mode='binary',
    shuffle=True)

test_generator = test_datagen.flow_from_directory(
    test_path,
    target_size=(350, 350),
    batch_size=32,
    class_mode='binary',
    shuffle=False)
```

```
Found 30251 images belonging to 2 classes.
Found 6300 images belonging to 2 classes.
Found 6300 images belonging to 2 classes.
```

```
In [5]: def display_images_from_generator(generator, num_images=5):
    x_batch, y_batch = next(generator)

    plt.figure(figsize=(num_images * 3, 3))
    for i in range(num_images):
        plt.subplot(1, num_images, i+1)
        plt.imshow(x_batch[i])
        label = "1: wildfire" if y_batch[i] > 0.5 else "0: nowildfire"
        plt.title(label, fontsize=12)
        plt.axis('off')
    plt.tight_layout()
    plt.show()
```

```
In [6]: display_images_from_generator(train_generator)
```



```
In [7]: display_images_from_generator(valid_generator)
```



```
In [8]: display_images_from_generator(test_generator)
```



```
In [9]: def efficientnet_b7(input_shape=(350, 350, 3), num_classes=1):
    base_model = EfficientNetB7(weights='imagenet', include_top=False, input_shape=input_shape)

    for layer in base_model.layers[:-4]:
        layer.trainable = False

    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(1024, activation='relu')(x)
    predictions = Dense(num_classes, activation='sigmoid')(x)

    model = Model(inputs=base_model.input, outputs=predictions)
    model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])
    model.name = 'EfficientNetB7'

    return model
```

```
In [10]: model=efficientnet_b7()
```

```
Downloading data from https://github.com/Callidior/keras-applications/releases/download/efficientnet/efficientnet-b7_weights_tf_dim_ordering_tf_kernels_autoaugment_notop.h5 (https://github.com/Callidior/keras-applications/releases/download/efficientnet/efficientnet-b7_weights_tf_dim_ordering_tf_kernels_autoaugment_notop.h5)
258434480/258434480 ━━━━━━━━━━━━ 55s 0us/step
```

```
In [11]: model.summary()
```

	2560)		
top_bn (BatchNormalizatio...	(None, 11, 11, 2560)	10,240	top_conv[0][0]
top_activation (Activation)	(None, 11, 11, 2560)	0	top_bn[0][0]
global_average_poo... (GlobalAveragePool...	(None, 2560)	0	top_activation[0...
dense (Dense)	(None, 1024)	2,622,464	global_average_p...
dense_1 (Dense)	(None, 1)	1,025	dense[0][0]

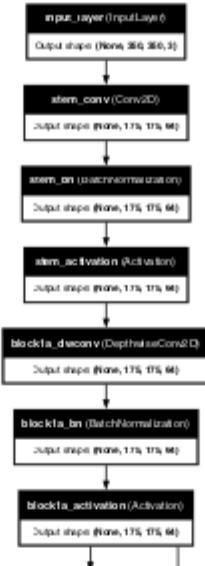
Total params: 66,721,169 (254.52 MB)

Trainable params: 4,267,009 (16.28 MB)

```
In [12]: plot_model(model, to_file=f'{model.name}_model.png', show_shapes=True, show_layer_name  
Image(filename=f'{model.name}_model.png')
```

dot: graph is too large for cairo-renderer bitmaps. Scaling by 0.256423 to fit

Out[12]:



```
In [13]: %%time
checkpointer = ModelCheckpoint(f'{model.name}.weights.h5', verbose=1, save_best_only=True)
early_stopping = EarlyStopping(monitor='val_loss', patience=10)

history = model.fit(
    train_generator,
    epochs=10,
    verbose=1,
    validation_data=valid_generator,
    callbacks=[checkpointer, early_stopping]
)
```

Epoch 1/10
946/946 0s 5s/step - accuracy: 0.9254 - loss: 0.2118
Epoch 1: val_loss improved from inf to 0.08041, saving model to EfficientNetB7.weights.h5
946/946 5360s 6s/step - accuracy: 0.9254 - loss: 0.2117 - val_accuracy: 0.9722 - val_loss: 0.0804
Epoch 2/10
946/946 0s 5s/step - accuracy: 0.9565 - loss: 0.1181
Epoch 2: val_loss improved from 0.08041 to 0.06582, saving model to EfficientNetB7.weights.h5
946/946 5505s 6s/step - accuracy: 0.9565 - loss: 0.1181 - val_accuracy: 0.9789 - val_loss: 0.0658
Epoch 3/10
946/946 0s 5s/step - accuracy: 0.9611 - loss: 0.1000
Epoch 3: val_loss improved from 0.06582 to 0.05713, saving model to EfficientNetB7.weights.h5
946/946 5586s 6s/step - accuracy: 0.9611 - loss: 0.1000 - val_accuracy: 0.9816 - val_loss: 0.0571
Epoch 4/10
946/946 0s 5s/step - accuracy: 0.9665 - loss: 0.0902
Epoch 4: val_loss improved from 0.05713 to 0.05349, saving model to EfficientNetB7.weights.h5
946/946 5202s 5s/step - accuracy: 0.9665 - loss: 0.0902 - val_accuracy: 0.9833 - val_loss: 0.0535
Epoch 5/10
946/946 0s 5s/step - accuracy: 0.9697 - loss: 0.0815
Epoch 5: val_loss did not improve from 0.05349
946/946 5154s 5s/step - accuracy: 0.9697 - loss: 0.0815 - val_accuracy: 0.9805 - val_loss: 0.0631
Epoch 6/10
946/946 0s 5s/step - accuracy: 0.9716 - loss: 0.0788
Epoch 6: val_loss improved from 0.05349 to 0.05237, saving model to EfficientNetB7.weights.h5
946/946 5258s 6s/step - accuracy: 0.9716 - loss: 0.0788 - val_accuracy: 0.9814 - val_loss: 0.0524
Epoch 7/10
946/946 0s 5s/step - accuracy: 0.9749 - loss: 0.0714
Epoch 7: val_loss improved from 0.05237 to 0.04960, saving model to EfficientNetB7.weights.h5
946/946 5297s 6s/step - accuracy: 0.9749 - loss: 0.0714 - val_accuracy: 0.9838 - val_loss: 0.0496
Epoch 8/10
946/946 0s 5s/step - accuracy: 0.9753 - loss: 0.0679
Epoch 8: val_loss improved from 0.04960 to 0.04748, saving model to EfficientNetB7.weights.h5
946/946 5270s 6s/step - accuracy: 0.9753 - loss: 0.0679 - val_accuracy: 0.9830 - val_loss: 0.0475
Epoch 9/10
946/946 0s 4s/step - accuracy: 0.9774 - loss: 0.0632
Epoch 9: val_loss improved from 0.04748 to 0.04611, saving model to EfficientNetB7.weights.h5
946/946 4813s 5s/step - accuracy: 0.9774 - loss: 0.0632 - val_accuracy: 0.9857 - val_loss: 0.0461
Epoch 10/10
946/946 0s 5s/step - accuracy: 0.9797 - loss: 0.0549
Epoch 10: val_loss did not improve from 0.04611
946/946 5271s 6s/step - accuracy: 0.9797 - loss: 0.0549 - val_accuracy: 0.9829 - val_loss: 0.0505
CPU times: user 49min 57s, sys: 3h 44min 4s, total: 4h 34min 2s
Wall time: 14h 38min 36s

```
In [14]: def display_images_with_predictions_from_generator(generator, model, num_wildfire=2, num_nowildfire=2):
    wildfire_pool = []
    nowildfire_pool = []

    while len(wildfire_pool) < num_wildfire or len(nowildfire_pool) < num_nowildfire:
        x_batch, y_batch = next(generator)
        for i in range(len(y_batch)):
            if y_batch[i] > 0.5: # Wildfire
                wildfire_pool.append((x_batch[i], y_batch[i]))
            else: # No Wildfire
                nowildfire_pool.append((x_batch[i], y_batch[i]))

    np.random.seed(10)
    selected_wildfire = np.random.choice(range(len(wildfire_pool)), size=num_wildfire)
    selected_nowildfire = np.random.choice(range(len(nowildfire_pool)), size=num_nowildfire)

    selected_images = [wildfire_pool[i][0] for i in selected_wildfire] + [nowildfire_pool[i][0] for i in selected_nowildfire]
    selected_labels = [wildfire_pool[i][1] for i in selected_wildfire] + [nowildfire_pool[i][1] for i in selected_nowildfire]

    predictions = model.predict(np.array(selected_images)).flatten()

    plt.figure(figsize=(len(selected_images) * 3, 3))
    for i in range(len(selected_images)):
        plt.subplot(1, len(selected_images), i + 1)
        plt.imshow(selected_images[i])
        true_label = "Wildfire" if selected_labels[i] > 0.5 else "No Wildfire"
        predicted_label = "Wildfire" if predictions[i] > 0.5 else "No Wildfire"
        plt.title(f"True: {true_label}\nPred: {predicted_label}")
        plt.axis('off')
    plt.tight_layout()
    plt.show()
```

```
In [15]: display_images_with_predictions_from_generator(test_generator, model)
```

1/1 ————— 11s 11s/step



```
In [16]: def plot_confusion_matrix_and_metrics(generator, model):
    label_names = ['No Wildfire', 'Wildfire']
    # Step 1: Calculate metrics and prepare the data
    y_true = generator.classes
    y_pred = model.predict(generator)
    y_pred = (y_pred > 0.5).astype(int)

    # Confusion matrix visualization
    conf_mat = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues')
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.xticks(ticks=np.arange(len(label_names)) + 0.5, labels=label_names, rotation=45)
    plt.yticks(ticks=np.arange(len(label_names)) + 0.5, labels=label_names, rotation=45)
    plt.show()

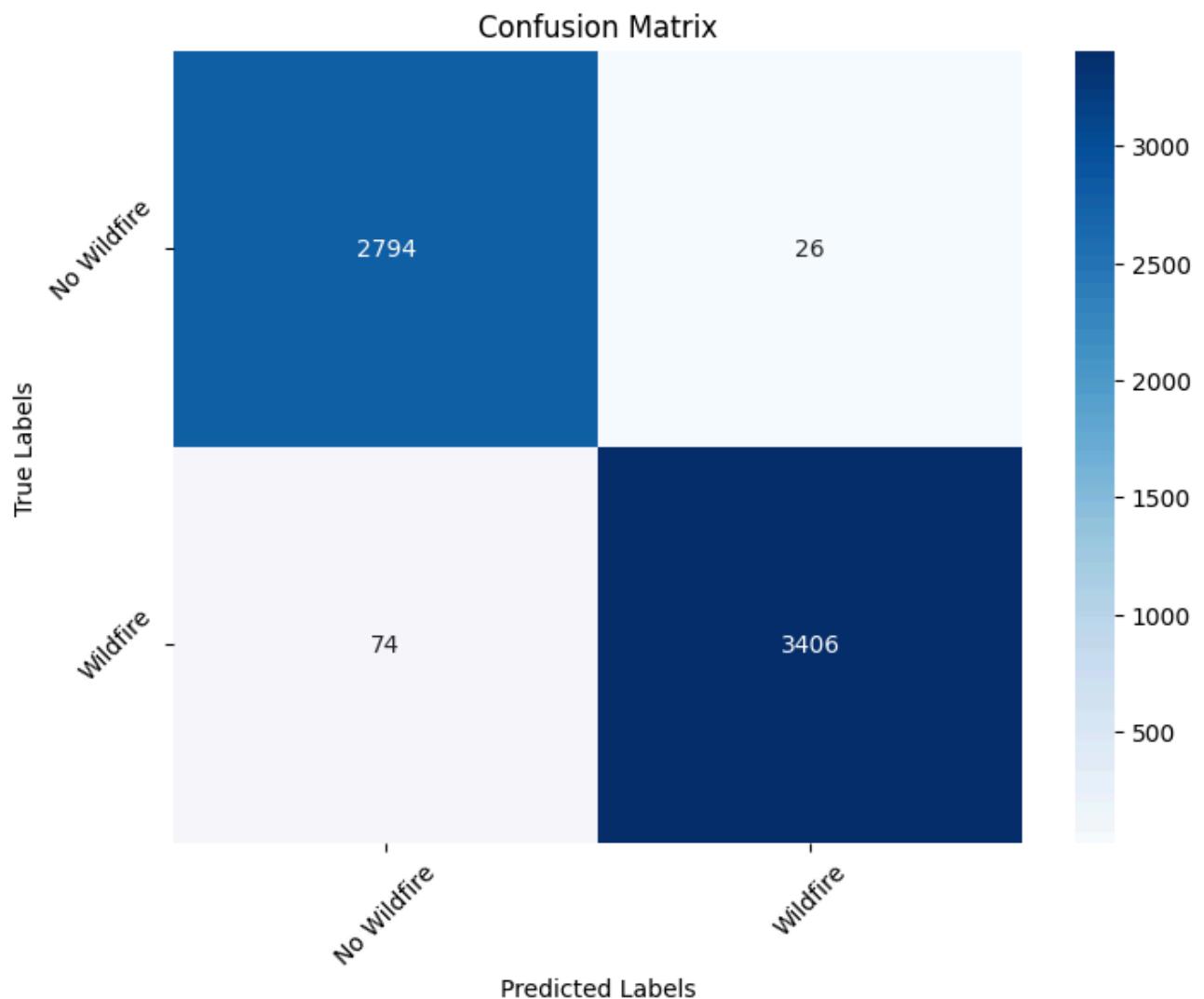
    # Classification Report in text
    report_text = classification_report(y_true, y_pred, target_names=label_names)
    print(report_text)

    # Print F1 score
    f1 = f1_score(y_true, y_pred)
    print(f"F1 Score: {f1:.2f}")

    # Print accuracy in percentage
    accuracy = accuracy_score(y_true, y_pred)
    print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
In [17]: plot_confusion_matrix_and_metrics(test_generator, model)
```

197/197 ━━━━━━ 709s 4s/step



	precision	recall	f1-score	support
No Wildfire	0.97	0.99	0.98	2820
Wildfire	0.99	0.98	0.99	3480
accuracy			0.98	6300
macro avg	0.98	0.98	0.98	6300
weighted avg	0.98	0.98	0.98	6300

F1 Score: 0.99

Accuracy: 98.41%

```
In [18]: def plot_training_history(history):
    plt.figure(figsize=(16, 6))

    epochs = range(1, len(history.history['accuracy']) + 1)

    plt.subplot(1, 2, 1)
    plt.plot(epochs, history.history['accuracy'], label='Train Accuracy', marker='o')
    plt.plot(epochs, history.history['val_accuracy'], label='Validation Accuracy', marker='o')
    plt.title('Model Training and Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.xticks(epochs)
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(epochs, history.history['loss'], label='Train Loss', marker='o')
    plt.plot(epochs, history.history['val_loss'], label='Validation Loss', marker='o')
    plt.title('Model Training and Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.xticks(epochs)
    plt.legend()

    plt.tight_layout()
    plt.savefig(f'{model.name}_training_history.png', dpi=300)
    plt.show()
```

```
In [19]: plot_training_history(history)
```

