

```
In [1]: import os
import tensorflow as tf
import numpy as np
from tensorflow.keras import layers, models
from efficientnet.tfkeras import EfficientNetB4
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=(224, 224),
    batch_size=32,
    class_mode='binary',
    shuffle=True)

valid_generator = valid_datagen.flow_from_directory(
    valid_path,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    shuffle=True)

test_generator = test_datagen.flow_from_directory(
    test_path,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    shuffle=False)
```

```
Found 30250 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]: class Patches(layers.Layer):
    def __init__(self, patch_size):
        super(Patches, self).__init__()
        self.patch_size = patch_size

    def call(self, images):
        batch_size = tf.shape(images)[0]
        patches = tf.image.extract_patches(
            images=images,
            sizes=[1, self.patch_size, self.patch_size, 1],
            strides=[1, self.patch_size, self.patch_size, 1],
            rates=[1, 1, 1, 1],
            padding='VALID',
        )
        patch_dims = patches.shape[-1]
        patches = tf.reshape(patches, [batch_size, -1, patch_dims])
        return patches

class PatchEncoder(layers.Layer):
    def __init__(self, num_patches, projection_dim):
        super(PatchEncoder, self).__init__()
        self.num_patches = num_patches
        self.projection = layers.Dense(units=projection_dim)
        self.position_embedding = layers.Embedding(
            input_dim=num_patches, output_dim=projection_dim
        )

    def call(self, patch):
        positions = tf.range(start=0, limit=self.num_patches, delta=1)
        encoded = self.projection(patch) + self.position_embedding(positions)
        return encoded

def create_vit_binary_classifier():
    input_shape = (224, 224, 3)
    patch_size = 16
    num_patches = (input_shape[0] // patch_size) * (input_shape[1] // patch_size)
    projection_dim = 64
    num_heads = 4
    transformer_units = [
        projection_dim * 2,
        projection_dim,
    ]
    transformer_layers = 8
    mlp_head_units = [2048, 1024]

    inputs = layers.Input(shape=input_shape)
    patches = Patches(patch_size)(inputs)
    encoded_patches = PatchEncoder(num_patches, projection_dim)(patches)

    for _ in range(transformer_layers):
        x1 = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
        attention_output = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=projection_dim, dropout=0.1
        )(x1, x1)
        x2 = layers.Add()([attention_output, encoded_patches])
        x3 = layers.LayerNormalization(epsilon=1e-6)(x2)
        x3 = layers.Dense(transformer_units[0], activation=tf.nn.gelu)(x3)
        x3 = layers.Dense(transformer_units[1])(x3)
        encoded_patches = layers.Add()([x3, x2])

    representation = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
    representation = layers.Flatten()(representation)
    representation = layers.Dropout(0.5)(representation)
    features = layers.Dense(mlp_head_units[0], activation=tf.nn.gelu)(representation)
    features = layers.Dropout(0.5)(features)
```

```

features = layers.Dense(mlp_head_units[1], activation=tf.nn.gelu)(features)
logits = layers.Dense(1, activation='sigmoid')(features) # Adjusted for binary classification
model = models.Model(inputs=inputs, outputs=logits)
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.name = 'ViT'
return model

```

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

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

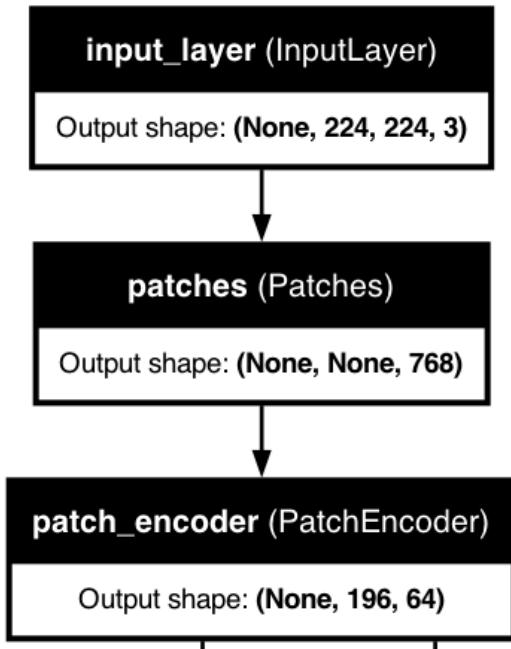
<code>layer_normalization_1 (LayerNormalization)</code>	(None, 196, 64)	128	<code>add_15[0][0]</code>
<code>flatten (Flatten)</code>	(None, 12544)	0	<code>layer_normalization_1[0][0]</code>
<code>dropout_8 (Dropout)</code>	(None, 12544)	0	<code>flatten[0][0]</code>
<code>dense_17 (Dense)</code>	(None, 2048)	25,692,160	<code>dropout_8[0][0]</code>
<code>dropout_9 (Dropout)</code>	(None, 2048)	0	<code>dense_17[0][0]</code>
<code>dense_18 (Dense)</code>	(None, 1024)	2,098,176	<code>dropout_9[0][0]</code>
<code>dense_19 (Dense)</code>	(None, 1)	1,025	<code>dense_18[0][0]</code>

Total params: 28,518,849 (108.79 MB)

Trainable params: 28,518,849 (108.79 MB)

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

Out[12]:



In [13]:

```
%time
checkpoint = 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=[checkpoint, early_stopping]
)

Epoch 1/10
946/946 0s 770ms/step - accuracy: 0.7867 - loss: 1.1663
Epoch 1: val_loss improved from inf to 0.20529, saving model to VIT.weights.h5
946/946 800s 834ms/step - accuracy: 0.7867 - loss: 1.1656 - val_accuracy: 0.9205 - val_loss: 0.2053
Epoch 2/10
946/946 0s 765ms/step - accuracy: 0.9103 - loss: 0.2368
Epoch 2: val_loss improved from 0.20529 to 0.17595, saving model to VIT.weights.h5
946/946 783s 827ms/step - accuracy: 0.9103 - loss: 0.2368 - val_accuracy: 0.9276 - val_loss: 0.1759
Epoch 3/10
946/946 0s 775ms/step - accuracy: 0.9297 - loss: 0.1904
Epoch 3: val_loss improved from 0.17595 to 0.15869, saving model to VIT.weights.h5
946/946 791s 835ms/step - accuracy: 0.9297 - loss: 0.1904 - val_accuracy: 0.9349 - val_loss: 0.1587
Epoch 4/10
946/946 0s 798ms/step - accuracy: 0.9362 - loss: 0.1654
Epoch 4: val_loss did not improve from 0.15869
946/946 813s 859ms/step - accuracy: 0.9362 - loss: 0.1654 - val_accuracy: 0.9459 - val_loss: 0.1710
Epoch 5/10
946/946 0s 814ms/step - accuracy: 0.9444 - loss: 0.1515
Epoch 5: val_loss did not improve from 0.15869
946/946 830s 877ms/step - accuracy: 0.9444 - loss: 0.1515 - val_accuracy: 0.9349 - val_loss: 0.2656
Epoch 6/10
946/946 0s 808ms/step - accuracy: 0.9479 - loss: 0.1424
Epoch 6: val_loss improved from 0.15869 to 0.13920, saving model to VIT.weights.h5
946/946 823s 870ms/step - accuracy: 0.9479 - loss: 0.1424 - val_accuracy: 0.9535 - val_loss: 0.1392
Epoch 7/10
946/946 0s 784ms/step - accuracy: 0.9548 - loss: 0.1267
Epoch 7: val_loss did not improve from 0.13920
946/946 800s 846ms/step - accuracy: 0.9548 - loss: 0.1267 - val_accuracy: 0.8856 - val_loss: 0.2230
Epoch 8/10
946/946 0s 828ms/step - accuracy: 0.9521 - loss: 0.1269
Epoch 8: val_loss did not improve from 0.13920
946/946 856s 904ms/step - accuracy: 0.9521 - loss: 0.1269 - val_accuracy: 0.9046 - val_loss: 0.2100
Epoch 9/10
946/946 0s 4s/step - accuracy: 0.9567 - loss: 0.1167
Epoch 9: val_loss improved from 0.13920 to 0.13178, saving model to VIT.weights.h5
946/946 3504s 4s/step - accuracy: 0.9567 - loss: 0.1167 - val_accuracy: 0.9470 - val_loss: 0.1318
Epoch 10/10
946/946 0s 1s/step - accuracy: 0.9623 - loss: 0.1025
Epoch 10: val_loss did not improve from 0.13178
946/946 1024s 1s/step - accuracy: 0.9623 - loss: 0.1025 - val_accuracy: 0.9552 - val_loss: 0.1472
CPU times: user 5h 50min, sys: 1h 57min 35s, total: 7h 47min 35s
Wall time: 3h 3min 43s
```

```
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 ━━━━━━ 1s 1s/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()

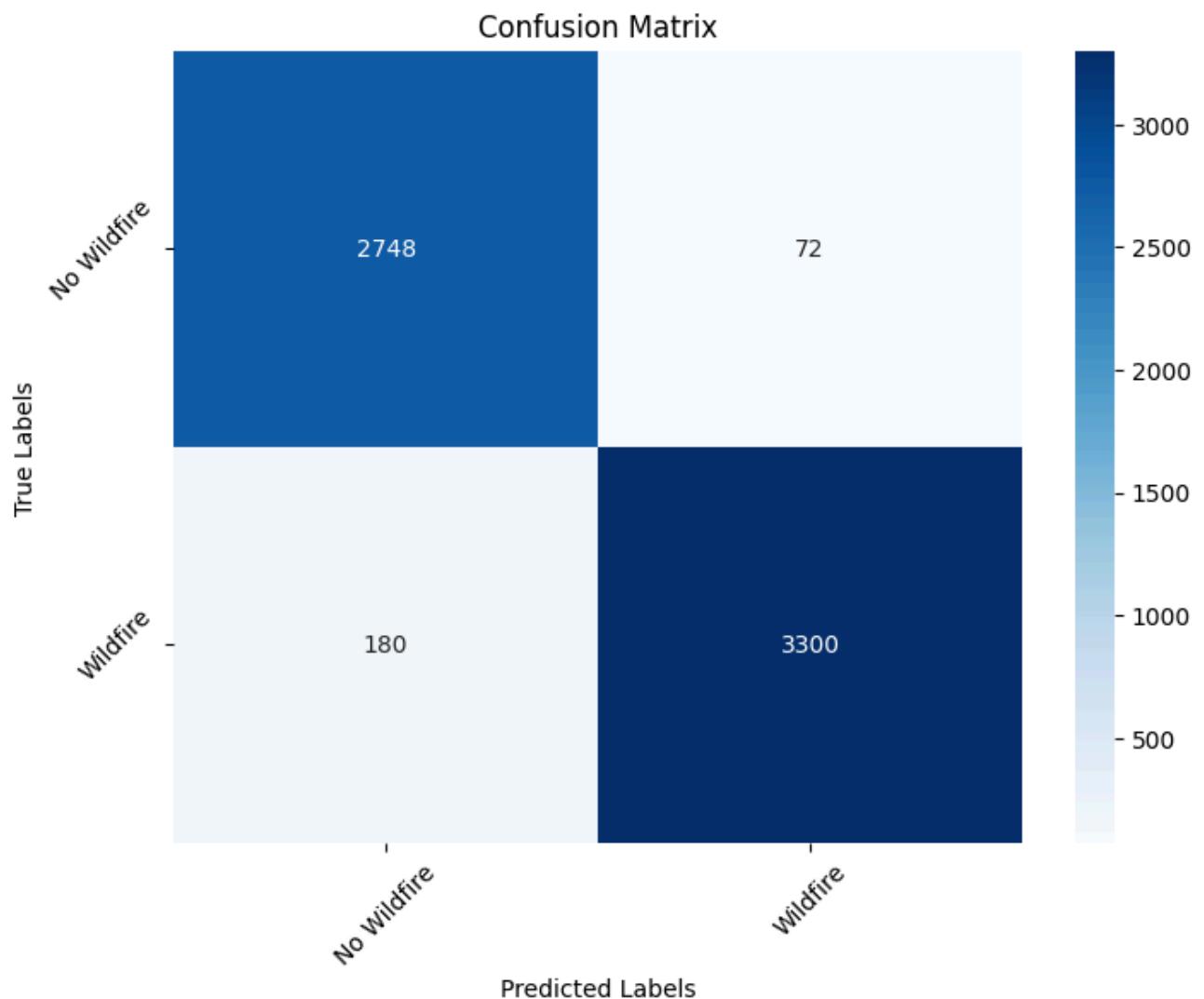
    # Classification Report in text
    report_text = classification_report(y_true, y_pred, target_names=label_names)
    print(report_text)

    # 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 ————— 70s 349ms/step



	precision	recall	f1-score	support
No Wildfire	0.94	0.97	0.96	2820
Wildfire	0.98	0.95	0.96	3480
accuracy			0.96	6300
macro avg	0.96	0.96	0.96	6300
weighted avg	0.96	0.96	0.96	6300

F1 Score: 0.96

Accuracy: 96.00%

```
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)
```

