```
# -*- coding: utf-8 -*-
"""
CIFake Image Classification - Restructured Version

Maintains identical functionality to original script but uses modular functions.
"""

import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import kagglehub
import tensorflow as tf
from tensorflow.keras import layers, models, callbacks
from tensorflow.keras.preprocessing import image_dataset_from_directory
from sklearn.metrics import confusion_matrix

# Configuration constants
CONFIG = {
    "dataset_slug": "birdy654/cifake-real-and-ai-generated-synthetic-images",
    "image_size": (32, 32),
    "batch_size": 32,
    "validation_split": 0.2,
    "random_seed": 123,
    "autotune": tf.data.AUTOTUNE,
    "training_epochs": 10,
    "early_stopping_patience": 5,
    "model_save_path": "best_model.keras"
}

def handle_dataset_download():
    """Download and validate dataset structure"""
    try:
        base_path = kagglehub.dataset_download(CONFIG["dataset_slug"])
        train_path = os.path.join(base_path, 'train')
        test_path = os.path.join(base_path, 'test')

        if not all(os.path.exists(p) for p in [train_path, test_path]):
            raise FileNotFoundError("Dataset directories missing")

        return train_path, test_path

    except Exception as e:
        print(f"Dataset error: {str(e)}")
        exit()

def create_datasets(train_dir, test_dir):
    """Create optimized TensorFlow datasets"""
    try:
        train_ds = image_dataset_from_directory(
            train_dir,
            validation_split=CONFIG["validation_split"],
            subset="training",
            seed=CONFIG["random_seed"],
            image_size=CONFIG["image_size"],
            batch_size=CONFIG["batch_size"],
            label_mode='binary'
        )

        val_ds = image_dataset_from_directory(
            train_dir,
            validation_split=CONFIG["validation_split"],
            subset="validation",
            seed=CONFIG["random_seed"],
            image_size=CONFIG["image_size"],
            batch_size=CONFIG["batch_size"]
        )

        test_ds = image_dataset_from_directory(
            test_dir,
            image_size=CONFIG["image_size"],
            batch_size=CONFIG["batch_size"],
            shuffle=False
        )

        return (
```

```
        return (
            train_ds.prefetch(CONFIG["autotune"]),
            val_ds.prefetch(CONFIG["autotune"]),
            test_ds.prefetch(CONFIG["autotune"]),
            train_ds.class_names
        )

    except Exception as e:
        print(f"Data loading error: {str(e)}")
        exit()

def build_cnn_model():
    """Construct the CNN architecture"""
    input_shape = CONFIG["image_size"] + (3,)

    model = models.Sequential([
        layers.Input(shape=input_shape),
        layers.Rescaling(1./255),
        layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2)),
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(1, activation='sigmoid')
    ])

    model.compile(
        optimizer='adam',
        loss='binary_crossentropy',
        metrics=['accuracy']
    )
    return model

def train_model(model, train_data, val_data):
    """Handle model training with callbacks"""
    # Changed variable name to avoid conflict with callbacks module
    callback_list = [
        callbacks.EarlyStopping(
            monitor='val_loss',
            patience=CONFIG["early_stopping_patience"],
            restore_best_weights=True,
            verbose=1
        ),
        callbacks.ModelCheckpoint(
            CONFIG["model_save_path"],
            monitor='val_accuracy',
            save_best_only=True,
            verbose=1
        )
    ]

    return model.fit(
        train_data,
        validation_data=val_data,
        epochs=CONFIG["training_epochs"],
        callbacks=callback_list  # Changed variable name here
    )

def visualize_results(history, class_names, test_data, model):
    """Generate performance visualizations"""
    # Training history plots
    history_df = pd.DataFrame(history.history)
    fig, ax = plt.subplots(1, 2, figsize=(14, 5))

    ax[0].plot(history_df['accuracy'], label='Train')
    ax[0].plot(history_df['val_accuracy'], label='Validation')
    ax[0].set_title('Accuracy Metrics')
    ax[0].set_ylabel('Accuracy')
    ax[0].legend()

    ax[1].plot(history_df['loss'], label='Train')
    ax[1].plot(history_df['val_loss'], label='Validation')
    ax[1].set_title('Loss Metrics')
    ax[1].set_ylabel('Loss')
    ax[1].legend()

    plt.tight_layout()
```

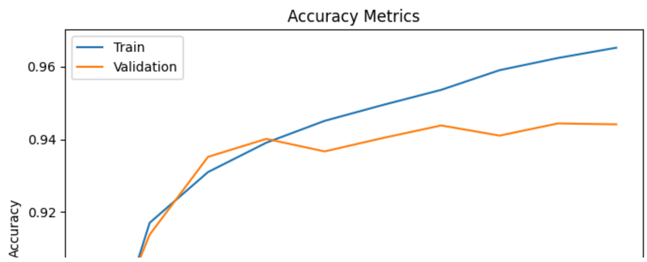
```
    plt.tight_layout()
    plt.show()

    # Confusion matrix
    y_pred = (model.predict(test_data) > 0.5).astype(int).flatten()
    y_true = np.concatenate([y.numpy() for _, y in test_data], axis=0)

    plt.figure(figsize=(8, 6))
    sns.heatmap(
        confusion_matrix(y_true, y_pred),
        annot=True,
        fmt='d',
        xticklabels=class_names,
        yticklabels=class_names
    )
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Confusion Matrix')
    plt.show()

def main():
    """Main execution flow"""
    # Dataset setup
    global test_ds # Define test_ds as a global variable
    train_path, test_path = handle_dataset_download()
    train_ds, val_ds, test_ds, classes = create_datasets(train_path, test_path)

    # Model lifecycle
    model = build_cnn_model()
    model.summary()

    history = train_model(model, train_ds, val_ds)

    # Final evaluation
    test_results = model.evaluate(test_ds, verbose=0)
    print(f"\nTest Accuracy: {test_results[1]:.2%}")
    print(f"Test Loss: {test_results[0]:.4f}")

    # Visualizations
    visualize_results(history, classes, test_ds, model)

if __name__ == "__main__":
    main()
```

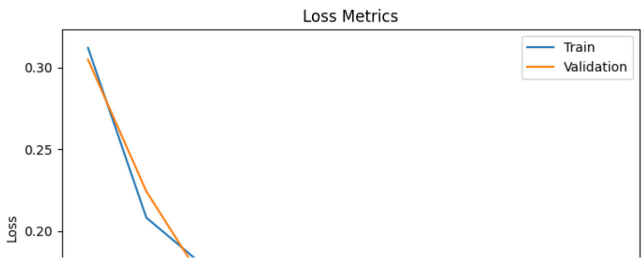
```
Warning: Looks like you're using an outdated `kagglehub` version (installed: 0.3.10), please consider upgrading to the l
Found 100000 files belonging to 2 classes.
Using 80000 files for training.
Found 100000 files belonging to 2 classes.
Using 20000 files for validation.
Found 20000 files belonging to 2 classes.
```

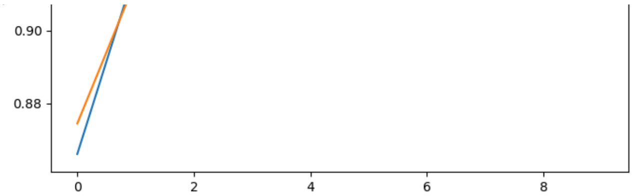
Model: "sequential_3"

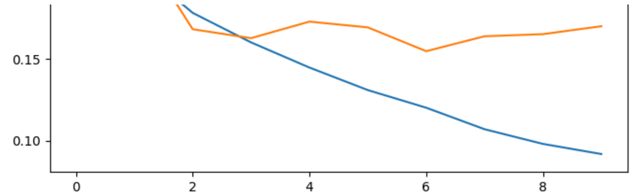
Layer (type)	Output Shape	Param #
rescaling_1 (Rescaling)	(None, 32, 32, 3)	0
conv2d_2 (Conv2D)	(None, 32, 32, 32)	896
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_3 (Conv2D)	(None, 16, 16, 64)	18,496
max_pooling2d_3 (MaxPooling2D)	(None, 8, 8, 64)	0
flatten_1 (Flatten)	(None, 4096)	0
dense_6 (Dense)	(None, 128)	524,416
dropout_3 (Dropout)	(None, 128)	0
dense_7 (Dense)	(None, 1)	129

```
 Total params: 543,937 (2.07 MB)
 Trainable params: 543,937 (2.07 MB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - accuracy: 0.8168 - loss: 0.3951
Epoch 1: val_accuracy improved from -inf to 0.87445, saving model to best_model.keras
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 168s 66ms/step - accuracy: 0.8168 - loss: 0.3951 - val_accuracy: 0.8745 - val_loss: 0.304
Epoch 2/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - accuracy: 0.9125 - loss: 0.2185
Epoch 2: val_accuracy improved from 0.87445 to 0.91375, saving model to best_model.keras
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 210s 69ms/step - accuracy: 0.9125 - loss: 0.2185 - val_accuracy: 0.9137 - val_loss: 0.224
Epoch 3/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 0.9297 - loss: 0.1829
Epoch 3: val_accuracy improved from 0.91375 to 0.93515, saving model to best_model.keras
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 172s 69ms/step - accuracy: 0.9297 - loss: 0.1829 - val_accuracy: 0.9352 - val_loss: 0.168
Epoch 4/10
2499/2500 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - accuracy: 0.9386 - loss: 0.1618
Epoch 4: val_accuracy improved from 0.93515 to 0.94010, saving model to best_model.keras
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 173s 69ms/step - accuracy: 0.9386 - loss: 0.1618 - val_accuracy: 0.9401 - val_loss: 0.162
Epoch 5/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - accuracy: 0.9447 - loss: 0.1455
Epoch 5: val_accuracy did not improve from 0.94010
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 195s 67ms/step - accuracy: 0.9447 - loss: 0.1455 - val_accuracy: 0.9366 - val_loss: 0.172
Epoch 6/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - accuracy: 0.9487 - loss: 0.1326
Epoch 6: val_accuracy improved from 0.94010 to 0.94035, saving model to best_model.keras
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 174s 70ms/step - accuracy: 0.9487 - loss: 0.1326 - val_accuracy: 0.9403 - val_loss: 0.169
Epoch 7/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - accuracy: 0.9540 - loss: 0.1197
Epoch 7: val_accuracy improved from 0.94035 to 0.94380, saving model to best_model.keras
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 201s 70ms/step - accuracy: 0.9540 - loss: 0.1197 - val_accuracy: 0.9438 - val_loss: 0.154
Epoch 8/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - accuracy: 0.9594 - loss: 0.1077
Epoch 8: val_accuracy did not improve from 0.94380
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 200s 69ms/step - accuracy: 0.9594 - loss: 0.1077 - val_accuracy: 0.9410 - val_loss: 0.163
Epoch 9/10
2499/2500 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - accuracy: 0.9621 - loss: 0.0997
Epoch 9: val_accuracy improved from 0.94380 to 0.94435, saving model to best_model.keras
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 196s 67ms/step - accuracy: 0.9621 - loss: 0.0997 - val_accuracy: 0.9444 - val_loss: 0.165
Epoch 10/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - accuracy: 0.9646 - loss: 0.0916
Epoch 10: val_accuracy did not improve from 0.94435
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 200s 66ms/step - accuracy: 0.9646 - loss: 0.0916 - val_accuracy: 0.9441 - val_loss: 0.170
Restoring model weights from the end of the best epoch: 7.

Test Accuracy: 94.46%
Test Loss: 0.1542
```

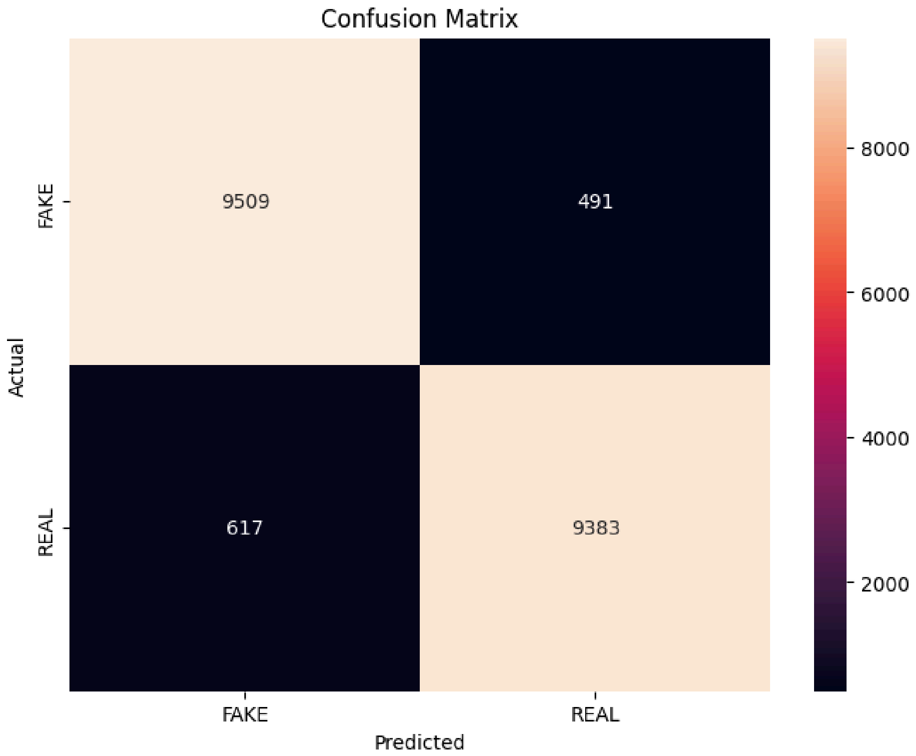









```
625/625 ━━━━━━━━━━━━━━━━━━━━ 11s 17ms/step
```



Start coding or generate with AI.

```
# -*- coding: utf-8 -*-
"""
CIFake Image Classification - Restructured Version

Maintains identical functionality to original script but uses modular functions.
"""

import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import kagglehub
import tensorflow as tf
from tensorflow.keras import layers, models, callbacks
from tensorflow.keras.preprocessing import image_dataset_from_directory
from sklearn.metrics import confusion_matrix

# Configuration constants
CONFIG = {
    "dataset_slug": "birdy654/cifake-real-and-ai-generated-synthetic-images",
    "image_size": (32, 32),
    "batch_size": 32,
    "validation_split": 0.2,
    "random_seed": 123,
    "autotune": tf.data.AUTOTUNE,
    "training_epochs": 10,
    "early_stopping_patience": 5,
    "model_save_path": "best_model.keras"
}

def handle_dataset_download():
    """Download and validate dataset structure"""
    try:
        base_path = kagglehub.dataset_download(CONFIG["dataset_slug"])
        train_path = os.path.join(base_path, 'train')
        test_path = os.path.join(base_path, 'test')

        if not all(os.path.exists(p) for p in [train_path, test_path]):
            raise FileNotFoundError("Dataset directories missing")

        return train_path, test_path

    except Exception as e:
        print(f"Dataset error: {str(e)}")
        exit()

def create_datasets(train_dir, test_dir):
    """Create optimized TensorFlow datasets"""
    try:
        train_ds = image_dataset_from_directory(
            train_dir,
            validation_split=CONFIG["validation_split"],
            subset="training",
            seed=CONFIG["random_seed"],
            image_size=CONFIG["image_size"],
            batch_size=CONFIG["batch_size"],
            label_mode='binary'
        )

        val_ds = image_dataset_from_directory(
            train_dir,
            validation_split=CONFIG["validation_split"],
            subset="validation",
            seed=CONFIG["random_seed"],
            image_size=CONFIG["image_size"],
            batch_size=CONFIG["batch_size"]
        )

        test_ds = image_dataset_from_directory(
            test_dir,
            image_size=CONFIG["image_size"],
            batch_size=CONFIG["batch_size"],
            shuffle=False
```

```
        )

        return (
            train_ds.prefetch(CONFIG["autotune"]),
            val_ds.prefetch(CONFIG["autotune"]),
            test_ds.prefetch(CONFIG["autotune"]),
            train_ds.class_names
        )

    except Exception as e:
        print(f"Data loading error: {str(e)}")
        exit()

def build_cnn_model():
    """Construct the CNN architecture"""
    input_shape = CONFIG["image_size"] + (3,)

    model = models.Sequential([
        layers.Input(shape=input_shape),
        layers.Rescaling(1./255),
        layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2)),
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(1, activation='sigmoid')
    ])

    model.compile(
        optimizer='adam',
        loss='binary_crossentropy',
        metrics=['accuracy']
    )
    return model

def train_model(model, train_data, val_data):
    """Handle model training with callbacks"""
    # Changed variable name to avoid conflict with callbacks module
    callback_list = [
        callbacks.EarlyStopping(
            monitor='val_loss',
            patience=CONFIG["early_stopping_patience"],
            restore_best_weights=True,
            verbose=1
        ),
        callbacks.ModelCheckpoint(
            CONFIG["model_save_path"],
            monitor='val_accuracy',
            save_best_only=True,
            verbose=1
        )
    ]

    return model.fit(
        train_data,
        validation_data=val_data,
        epochs=CONFIG["training_epochs"],
        callbacks=callback_list  # Changed variable name here
    )

def visualize_results(history, class_names, test_data, model):
    """Generate performance visualizations"""
    # Training history plots
    history_df = pd.DataFrame(history.history)
    fig, ax = plt.subplots(1, 2, figsize=(14, 5))

    ax[0].plot(history_df['accuracy'], label='Train')
    ax[0].plot(history_df['val_accuracy'], label='Validation')
    ax[0].set_title('Accuracy Metrics')
    ax[0].set_ylabel('Accuracy')
    ax[0].legend()

    ax[1].plot(history_df['loss'], label='Train')
    ax[1].plot(history_df['val_loss'], label='Validation')
    ax[1].set_title('Loss Metrics')
    ax[1].set_ylabel('Loss')
```

```
    ax[1].legend()

    plt.tight_layout()
    plt.show()

    # Confusion matrix
    y_pred = (model.predict(test_data) > 0.5).astype(int).flatten()
    y_true = np.concatenate([y.numpy() for _, y in test_data], axis=0)

    plt.figure(figsize=(8, 6))
    sns.heatmap(
        confusion_matrix(y_true, y_pred),
        annot=True,
        fmt='d',
        xticklabels=class_names,
        yticklabels=class_names
    )
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Confusion Matrix')
    plt.show()

def generate_adversarial_examples(model, dataset, epsilon=0.01):
    """Generate adversarially perturbed images using FGSM"""
    adv_images = []
    adv_labels = []
    loss_object = tf.keras.losses.BinaryCrossentropy()

    for images, labels in dataset:
        images = tf.cast(images, tf.float32) / 255.0
        with tf.GradientTape() as tape:
            tape.watch(images)
            predictions = model(images)
            loss = loss_object(labels, predictions)

        gradients = tape.gradient(loss, images)
        perturbed_images = images + epsilon * tf.sign(gradients)
        perturbed_images = tf.clip_by_value(perturbed_images, 0, 1)

        adv_images.append(perturbed_images)
        adv_labels.append(labels)

    return tf.data.Dataset.from_tensor_slices((tf.concat(adv_images, axis=0), tf.concat(adv_labels, axis=0))).batch(CONFIG["batch

def main():
    """Main execution flow"""
    # Dataset setup
    global test_ds # Define test_ds as a global variable
    train_path, test_path = handle_dataset_download()
    train_ds, val_ds, test_ds, classes = create_datasets(train_path, test_path)

    # Model lifecycle
    model = build_cnn_model()
    model.summary()

    history = train_model(model, train_ds, val_ds)


    # Generate adversarial dataset
    adv_train_ds = generate_adversarial_examples(model, train_ds)


    # Train model again with adversarial dataset
    print("\nTraining model with adversarial examples...")
    history_adv = train_model(model, adv_train_ds, val_ds)

    # Final evaluation
    test_results = model.evaluate(test_ds, verbose=0)
    print(f"\nTest Accuracy: {test_results[1]:.2%}")
    print(f"Test Loss: {test_results[0]:.4f}")

    # Visualizations
    visualize_results(history, classes, test_ds, model)

if __name__ == "__main__":
    main()
```

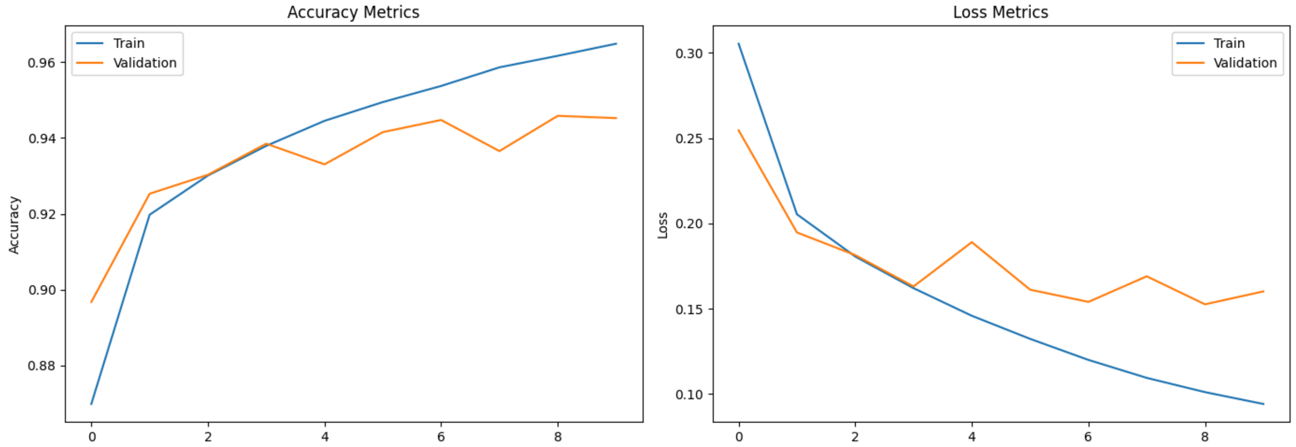
```
Warning: Looks like you're using an outdated `kagglehub` version (installed: 0.3.10), please consider upgrading to the l
Found 100000 files belonging to 2 classes.
Using 80000 files for training.
Found 100000 files belonging to 2 classes.
Using 20000 files for validation.
Found 20000 files belonging to 2 classes.
```

Model: "sequential_7"

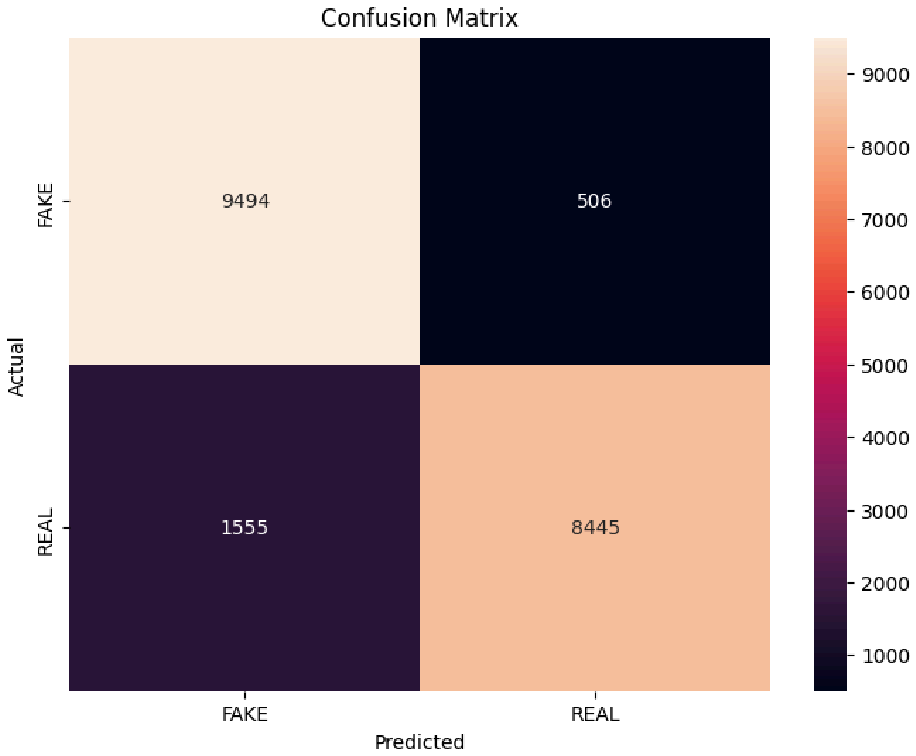
Layer (type)	Output Shape	Param #
rescaling_5 (Rescaling)	(None, 32, 32, 3)	0
conv2d_6 (Conv2D)	(None, 32, 32, 32)	896
max_pooling2d_6 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_7 (Conv2D)	(None, 16, 16, 64)	18,496
max_pooling2d_7 (MaxPooling2D)	(None, 8, 8, 64)	0
flatten_3 (Flatten)	(None, 4096)	0
dense_14 (Dense)	(None, 128)	524,416
dropout_7 (Dropout)	(None, 128)	0
dense_15 (Dense)	(None, 1)	129

```
 Total params: 543,937 (2.07 MB)
 Trainable params: 543,937 (2.07 MB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - accuracy: 0.8221 - loss: 0.3869
Epoch 1: val_accuracy improved from -inf to 0.89675, saving model to best_model.keras
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 178s 70ms/step - accuracy: 0.8221 - loss: 0.3869 - val_accuracy: 0.8967 - val_loss: 0.254
Epoch 2/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - accuracy: 0.9149 - loss: 0.2153
Epoch 2: val_accuracy improved from 0.89675 to 0.92530, saving model to best_model.keras
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 194s 67ms/step - accuracy: 0.9149 - loss: 0.2153 - val_accuracy: 0.9253 - val_loss: 0.194
Epoch 3/10
2499/2500 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - accuracy: 0.9286 - loss: 0.1839
Epoch 3: val_accuracy improved from 0.92530 to 0.93030, saving model to best_model.keras
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 176s 71ms/step - accuracy: 0.9286 - loss: 0.1839 - val_accuracy: 0.9303 - val_loss: 0.181
Epoch 4/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - accuracy: 0.9360 - loss: 0.1649
Epoch 4: val_accuracy improved from 0.93030 to 0.93850, saving model to best_model.keras
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 196s 68ms/step - accuracy: 0.9360 - loss: 0.1649 - val_accuracy: 0.9385 - val_loss: 0.163
Epoch 5/10
2499/2500 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - accuracy: 0.9437 - loss: 0.1488
Epoch 5: val_accuracy did not improve from 0.93850
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 170s 68ms/step - accuracy: 0.9438 - loss: 0.1488 - val_accuracy: 0.9330 - val_loss: 0.189
Epoch 6/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - accuracy: 0.9488 - loss: 0.1341
Epoch 6: val_accuracy improved from 0.93850 to 0.94155, saving model to best_model.keras
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 199s 67ms/step - accuracy: 0.9488 - loss: 0.1341 - val_accuracy: 0.9416 - val_loss: 0.161
Epoch 7/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - accuracy: 0.9532 - loss: 0.1206
Epoch 7: val_accuracy improved from 0.94155 to 0.94475, saving model to best_model.keras
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 177s 71ms/step - accuracy: 0.9532 - loss: 0.1206 - val_accuracy: 0.9448 - val_loss: 0.154
Epoch 8/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - accuracy: 0.9584 - loss: 0.1106
Epoch 8: val_accuracy did not improve from 0.94475
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 205s 72ms/step - accuracy: 0.9584 - loss: 0.1106 - val_accuracy: 0.9366 - val_loss: 0.169
Epoch 9/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - accuracy: 0.9606 - loss: 0.1043
Epoch 9: val_accuracy improved from 0.94475 to 0.94585, saving model to best_model.keras
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 201s 72ms/step - accuracy: 0.9606 - loss: 0.1043 - val_accuracy: 0.9459 - val_loss: 0.152
Epoch 10/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - accuracy: 0.9640 - loss: 0.0956
Epoch 10: val_accuracy did not improve from 0.94585
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 168s 67ms/step - accuracy: 0.9640 - loss: 0.0956 - val_accuracy: 0.9452 - val_loss: 0.166
Restoring model weights from the end of the best epoch: 9.

Training model with adversarial examples...
Epoch 1/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - accuracy: 0.4930 - loss: 0.9245
Epoch 1: val_accuracy improved from -inf to 0.89780, saving model to best_model.keras
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 160s 64ms/step - accuracy: 0.4930 - loss: 0.9244 - val_accuracy: 0.8978 - val_loss: 0.266
Epoch 2/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - accuracy: 0.4971 - loss: 0.6932
Epoch 2: val_accuracy did not improve from 0.89780
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 199s 62ms/step - accuracy: 0.4971 - loss: 0.6932 - val_accuracy: 0.8978 - val_loss: 0.266
Epoch 3/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - accuracy: 0.4966 - loss: 0.6932
Epoch 3: val_accuracy did not improve from 0.89780
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 165s 66ms/step - accuracy: 0.4966 - loss: 0.6932 - val_accuracy: 0.8978 - val_loss: 0.266
```

```
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 165s 66ms/step - accuracy: 0.4966 - loss: 0.6932 - val_accuracy: 0.8978 - val_loss: 0.26(
Epoch 4/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - accuracy: 0.4966 - loss: 0.6932
Epoch 4: val_accuracy did not improve from 0.89780
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 155s 62ms/step - accuracy: 0.4966 - loss: 0.6932 - val_accuracy: 0.8978 - val_loss: 0.26(
Epoch 5/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - accuracy: 0.4966 - loss: 0.6932
Epoch 5: val_accuracy did not improve from 0.89780
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 167s 67ms/step - accuracy: 0.4966 - loss: 0.6932 - val_accuracy: 0.8978 - val_loss: 0.26(
Epoch 6/10
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - accuracy: 0.4966 - loss: 0.6932
Epoch 6: val_accuracy did not improve from 0.89780
2500/2500 ━━━━━━━━━━━━━━━━━━━━ 157s 63ms/step - accuracy: 0.4966 - loss: 0.6932 - val_accuracy: 0.8978 - val_loss: 0.26(
Epoch 6: early stopping
Restoring model weights from the end of the best epoch: 1.

Test Accuracy: 89.70%
Test Loss: 0.2546
```




```
625/625 ━━━━━━━━━━━━━━━━━━━━ 11s 18ms/step
```



```
import zipfile
import os

zip_path = "/content/Test datasets.zip"  # Replace with your actual zip file path
extract_path = "/content"  # Change this to your desired output folder

with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

print("Unzipping Done!")
```

```
Unzipping Done!
```

```
import os
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import pandas as pd  # Added for CSV generation
from tensorflow.keras.preprocessing import image

# Load the trained model
model = tf.keras.models.load_model(CONFIG["model_save_path"])

# Paths to test datasets
testdataset1_path = "/content/Test datasets/Test_dataset_1"

# Function to load, preprocess, predict, and save results
def predict_images_from_folder(folder_path, output_csv="predictions_test1.csv"):
    """Loads images, preprocesses them, predicts labels, and saves results to CSV."""
    image_paths = sorted([os.path.join(folder_path, img) for img in os.listdir(folder_path) if img.endswith((".jpg", ".png"))])

    predictions_list = []

    for img_path in image_paths:
        # Load image and preprocess
        img = image.load_img(img_path, target_size=CONFIG["image_size"])
        img_array = image.img_to_array(img) / 255.0  # Normalize
        img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension

        # Get prediction
        prediction = model.predict(img_array)
        label = "AI-Generated" if prediction[0][0] > 0.5 else "Real"
        predictions_list.append((os.path.basename(img_path), label))

    # Save predictions to a CSV file
    df = pd.DataFrame(predictions_list, columns=["Image Name", "Predicted Label"])
    df.to_csv(output_csv, index=False)
    print(f"\nPredictions saved to {output_csv}")

    return predictions_list

# Run predictions on Test Dataset 1 and save results
predictions_test1 = predict_images_from_folder(testdataset1_path)

# Display results
print("\nPredicted Labels for Test Dataset 1:\n")
for img_name, label in predictions_test1:
    print(f"{img_name}: {label}")

# Optional: Plot images with predictions
plt.figure(figsize=(12, 6))
for i, (img_name, label) in enumerate(predictions_test1[:20]):  # Show first 20 images
    img_path = os.path.join(testdataset1_path, img_name)
    img = image.load_img(img_path, target_size=CONFIG["image_size"])
    plt.subplot(4, 5, i + 1)
    plt.imshow(img)
    plt.title(f"{img_name}\n{label}")
    plt.axis("off")

plt.tight_layout()
plt.show()
```