In [1]:
```
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.regularizers import l2
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras import models,layers
import numpy as np
import math
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import  BatchNormalization
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.offsetbox as offsetbox
from PIL import Image
import os
```

In [2]:
```
df = tf.keras.preprocessing.image_dataset_from_directory(r"C:\Users\soura\OneDrive\Des
                                                          shuffle=True ,
                                                          image_size = (256,256),
                                                          batch_size = 32)
```

```
Found 52623 files belonging to 2 classes.
```

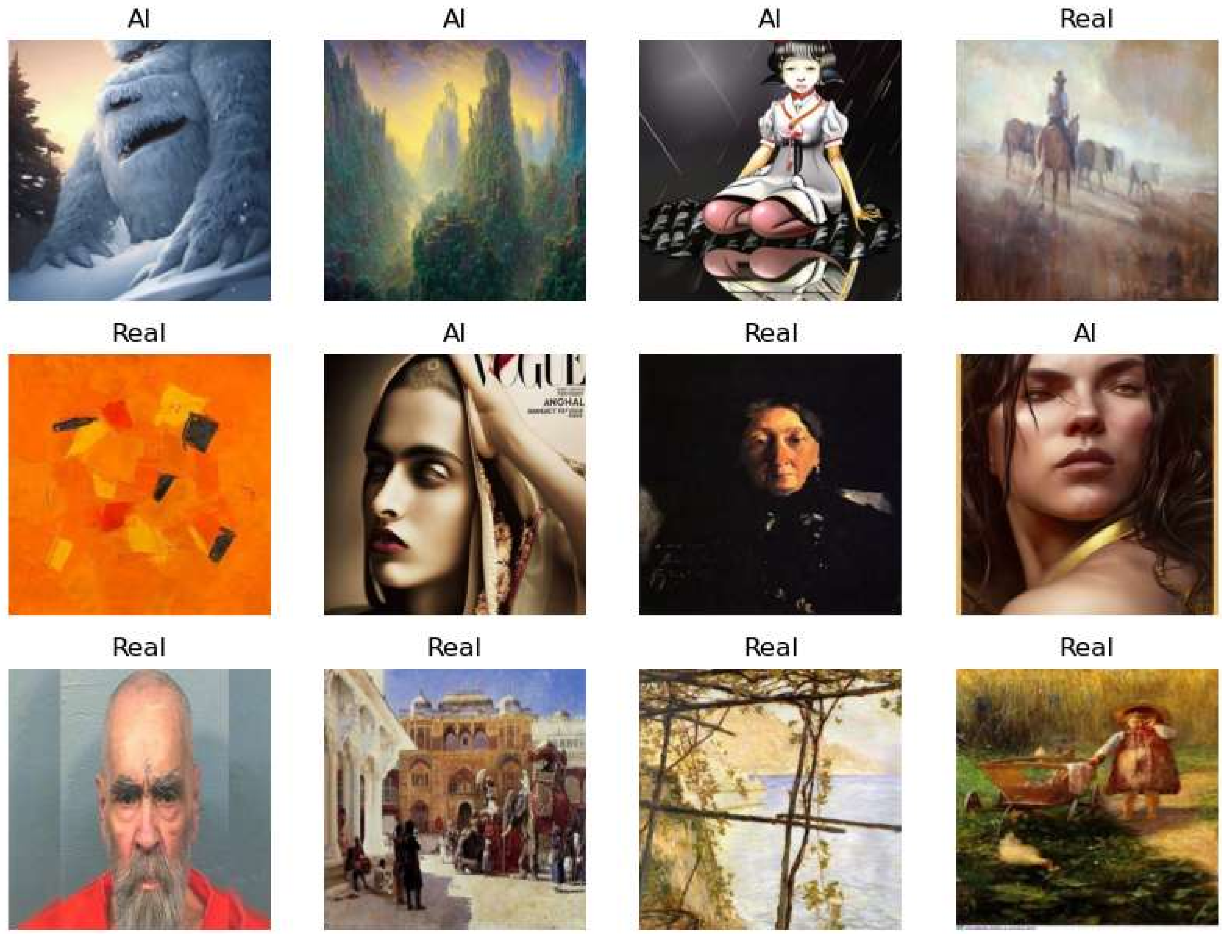
In [3]:
```
classes =df.class_names
classes
```

Out[3]:
```
['AI', 'Real']
```

In [4]:
```
for image_batch,label_batch in df.take(1):
    print(image_batch.shape)
    print(label_batch.numpy())
```

```
(32, 256, 256, 3)
[0 1 0 1 0 0 1 1 1 0 1 0 1 1 0 1 0 1 0 1 1 0 0 1 1 1 1 0 0 1 0 0]
```

In [5]:
```
plt.figure(figsize=(10,10))
for image_batch,label_batch in df.take(1):
    for i in range(12):
     ax = plt.subplot(4,4,i+1)
     plt.title(classes[label_batch[i]])
     plt.imshow(image_batch[i].numpy().astype('uint8'))
     plt.axis('off')
```



In [6]:
```python
train_size = 0.7
len(df)*train_size
```
Out[6]:
```
1151.5
```

In [7]:
```python
train_ds = df.take(1151)
len(train_ds)
```
Out[7]:
```
1151
```

In [8]:
```python
test_ds = df.skip(1151)
len(test_ds)
```
Out[8]:
```
494
```

In [9]:
```python
val_size=0.2
len(df)*val_size
```
Out[9]:
```
329.0
```

In [10]:
```python
val_ds = test_ds.take(329)
len(val_ds)
```
Out[10]:
```
329
```

In [11]:
```python
test_ds = test_ds.skip(329)
len(test_ds)
```

Out[11]: 165

In [12]:
```
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
```

In [13]:
```
from tensorflow.keras import models,layers

resize_and_rescale = tf.keras.Sequential([
  layers.experimental.preprocessing.Resizing(224,224),
  layers.experimental.preprocessing.Rescaling(1./255),
])
```

In [14]:
```
@tf.function
def preprocess_data(x, y):
    # Random horizontal flip
    x = tf.image.random_flip_left_right(x)

    # Random vertical flip
    x = tf.image.random_flip_up_down(x)

    # Random rotation (0, 90, 180, or 270 degrees)
    k = tf.random.uniform(shape=[], minval=0, maxval=4, dtype=tf.int32)
    x = tf.image.rot90(x, k=k)

    # Random brightness adjustment
    x = tf.image.random_brightness(x, max_delta=0.1)

    # Random contrast adjustment
    x = tf.image.random_contrast(x, lower=0.8, upper=1.2)

    # Random hue shift
    x = tf.image.random_hue(x, max_delta=0.1)

    # Normalization
    x = tf.image.per_image_standardization(x)

    return x, y

batch_size = 32


preprocessed_train_ds = train_ds.map(preprocess_data).prefetch(buffer_size=tf.data.AUT
preprocessed_val_ds = val_ds.map(preprocess_data).prefetch(buffer_size=tf.data.AUTOTUN
```

In [15]:
```
num_classes = 2
input_shape = (256,256, 3)
initial_learning_rate = 0.001
decay_rate = 0.1
decay_steps = 10
```

In [16]:
```
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=input_shap
from tensorflow.keras.layers import BatchNormalization

# Add custom classification layers on top of the base model
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu', kernel_regularizer=l2(0.01))(x)
```

```
x = BatchNormalization()(x)
x = Dropout(0.5)(x)

x = Dense(512, activation='relu', kernel_regularizer=l2(0.01))(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)

x = Dense(256, activation='relu', kernel_regularizer=l2(0.01))(x)
x = BatchNormalization()(x)

# Add more Dense layers with regularization and BatchNormalization
x = Dense(128, activation='relu', kernel_regularizer=l2(0.01))(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)

x = Dense(64, activation='relu', kernel_regularizer=l2(0.01))(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)

predictions = Dense(num_classes, activation='softmax')(x)

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

# Freeze the layers of the base model
for layer in base_model.layers:
    layer.trainable = False
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [96,
128, 160, 192, 224]. Weights for input shape (224, 224) will be loaded as the defaul
t.
```

In [17]:

```
def exponential_decay(epoch, initial_lr=0.001, decay_rate=0.1, decay_steps=10):
    return initial_lr * math.pow(decay_rate, epoch / decay_steps)
lr_scheduler = tf.keras.callbacks.LearningRateScheduler(exponential_decay)
```

In [18]:

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),  # Set the init
             loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

In [19]:

```
import math
BATCH_SIZE = 32
history = model.fit(
    preprocessed_train_ds,
    batch_size=BATCH_SIZE,
    validation_data=preprocessed_val_ds,
    epochs=10,
    callbacks=[lr_scheduler],
    verbose=1
)
```

```
Epoch 1/10
1151/1151 [==============================] - 1646s 1s/step - loss: 6.2689 - accuracy:
0.8540 - val_loss: 0.6843 - val_accuracy: 0.8496 - lr: 0.0010
Epoch 2/10
1151/1151 [==============================] - 1073s 930ms/step - loss: 0.5594 - accura
cy: 0.8679 - val_loss: 0.5232 - val_accuracy: 0.8831 - lr: 7.9433e-04
Epoch 3/10
1151/1151 [==============================] - 1052s 912ms/step - loss: 0.4875 - accura
cy: 0.8745 - val_loss: 0.4453 - val_accuracy: 0.8858 - lr: 5.0119e-04
Epoch 4/10
1151/1151 [==============================] - 1049s 910ms/step - loss: 0.4117 - accura
cy: 0.8785 - val_loss: 0.3761 - val_accuracy: 0.8938 - lr: 2.5119e-04
Epoch 5/10
1151/1151 [==============================] - 1052s 913ms/step - loss: 0.3528 - accura
cy: 0.8895 - val_loss: 0.3015 - val_accuracy: 0.9036 - lr: 1.0000e-04
Epoch 6/10
1151/1151 [==============================] - 1060s 919ms/step - loss: 0.3125 - accura
cy: 0.8971 - val_loss: 0.2681 - val_accuracy: 0.9144 - lr: 3.1623e-05
Epoch 7/10
1151/1151 [==============================] - 1046s 908ms/step - loss: 0.2938 - accura
cy: 0.9013 - val_loss: 0.2544 - val_accuracy: 0.9169 - lr: 7.9433e-06
Epoch 8/10
1151/1151 [==============================] - 10276s 9s/step - loss: 0.2879 - accurac
y: 0.9049 - val_loss: 0.2539 - val_accuracy: 0.9173 - lr: 1.5849e-06
Epoch 9/10
1151/1151 [==============================] - 3410s 3s/step - loss: 0.2832 - accuracy:
0.9065 - val_loss: 0.2549 - val_accuracy: 0.9180 - lr: 2.5119e-07
Epoch 10/10
1151/1151 [==============================] - 1199s 1s/step - loss: 0.2877 - accuracy:
0.9043 - val_loss: 0.2512 - val_accuracy: 0.9198 - lr: 3.1623e-08
```

In [20]:

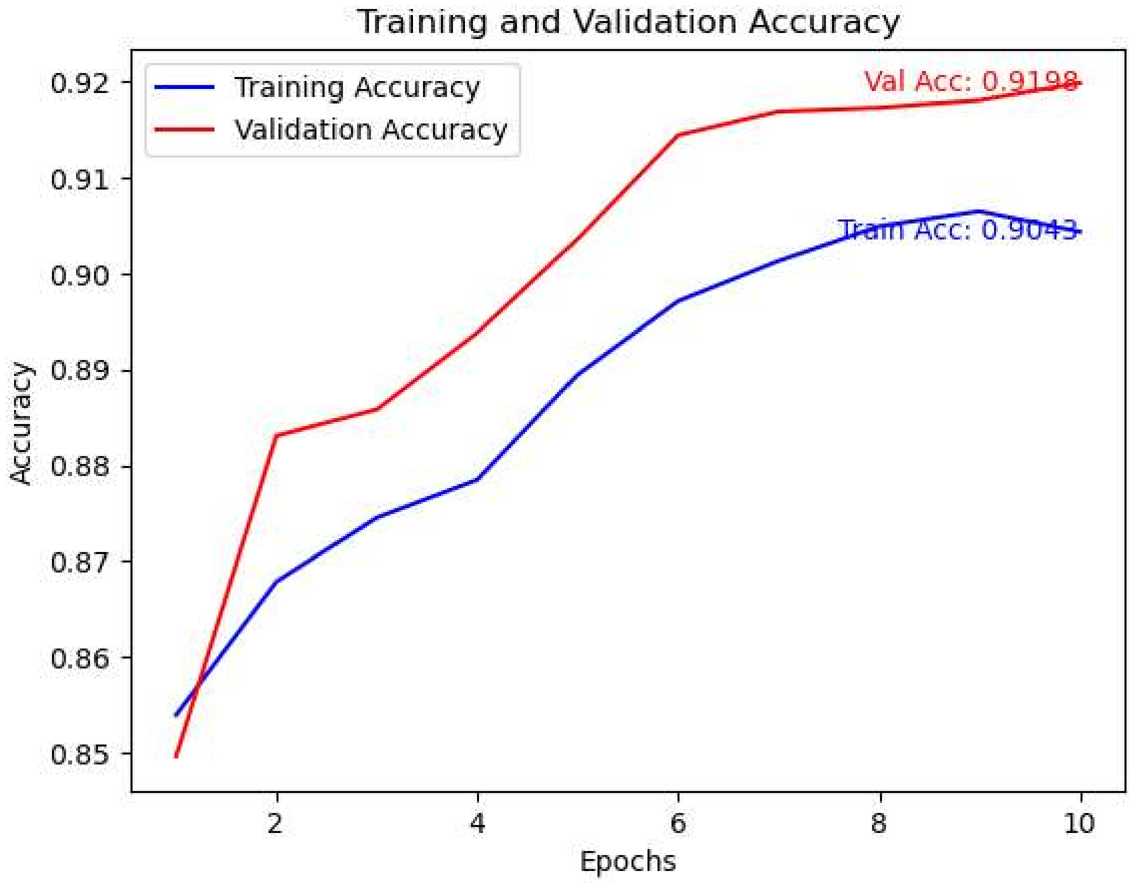
```python
import matplotlib.pyplot as plt
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

final_train_acc = train_acc[-1]
final_val_acc = val_acc[-1]

epochs = range(1, len(train_acc) + 1)
plt.plot(epochs, train_acc, 'b', label='Training Accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.text(epochs[-1], final_train_acc, f'Train Acc: {final_train_acc:.4f}', ha='right',
plt.text(epochs[-1], final_val_acc, f'Val Acc: {final_val_acc:.4f}', ha='right', va='c

plt.show()
```

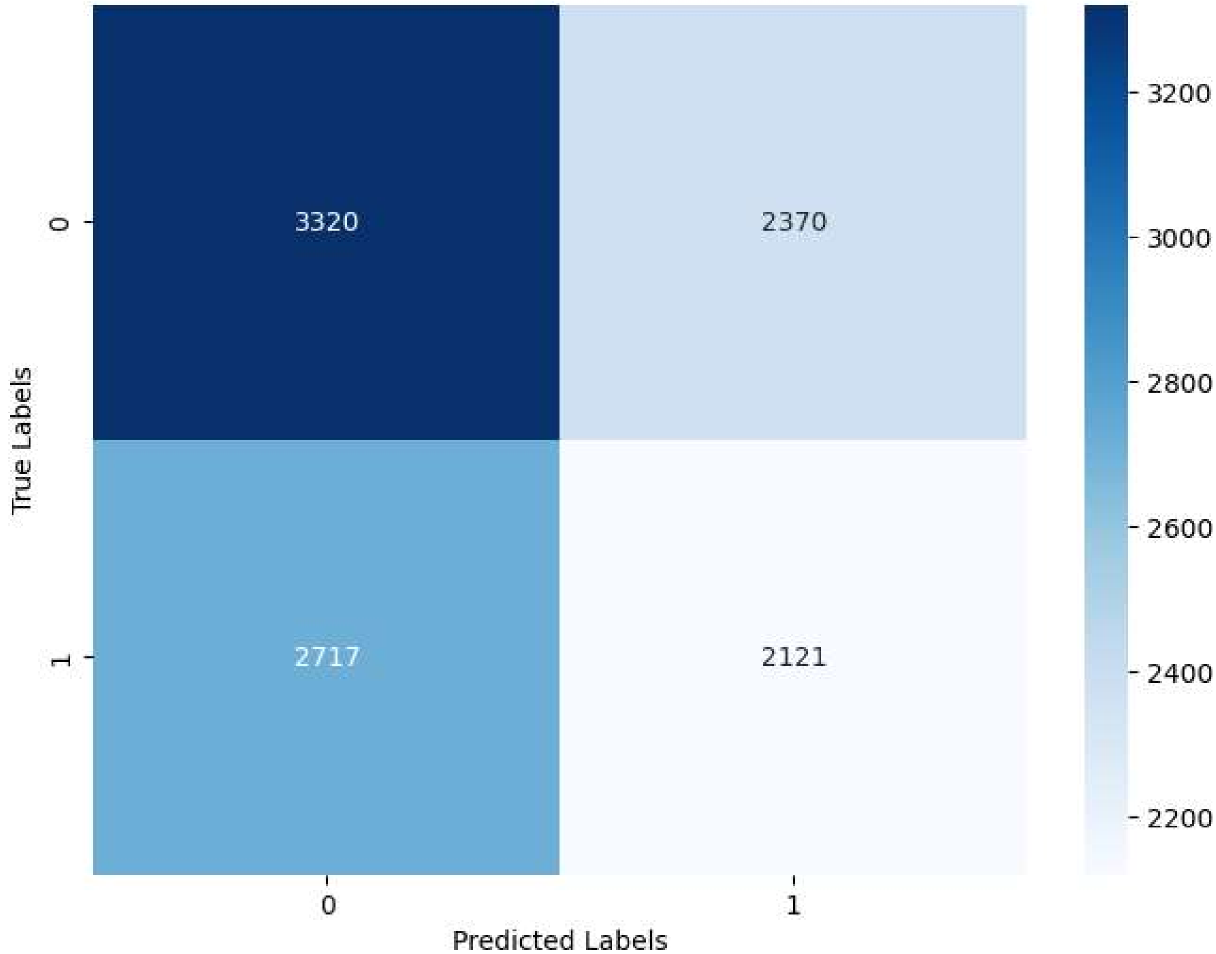


In [22]:
```
predictions = model.predict(preprocessed_val_ds)
predicted_labels = np.argmax(predictions, axis=1)
true_labels = np.concatenate([y for x, y in preprocessed_val_ds], axis=0)

cm = confusion_matrix(true_labels, predicted_labels)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

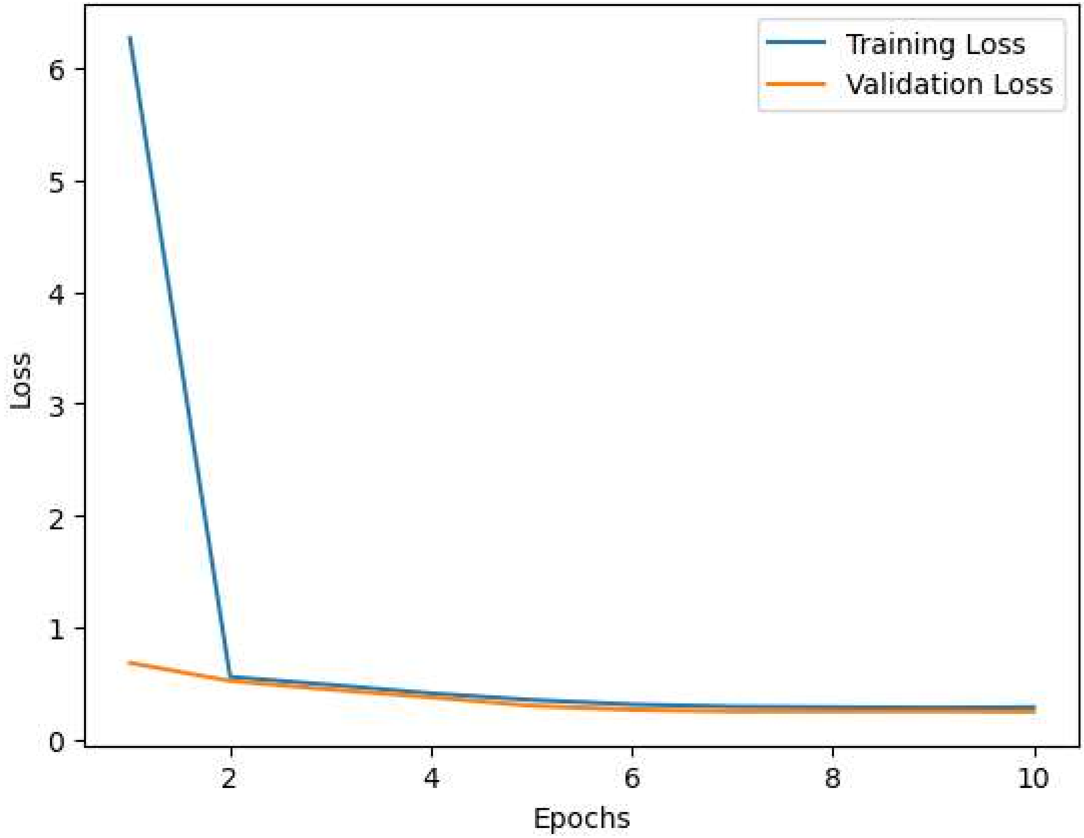
```
329/329 [==============================] - 240s 720ms/step
```



In [24]:
```
training_loss = history.history['loss']
validation_loss = history.history['val_loss']

# Create an array for the number of epochs
epochs = range(1, len(training_loss) + 1)

# Plot the training and validation loss over epochs
plt.plot(epochs, training_loss, label='Training Loss')
plt.plot(epochs, validation_loss, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [26]:
```
tf.keras.models.save_model(model,'my_final_model2.hdf5')
```

In [ ]: