Real vs Fake Images Detection with ELA(Error Level Analysis) and CNN

In [1]:
```
#import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
np.random.seed(2)
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from keras.utils.np_utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPool2D, Dropout
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import EarlyStopping

```

In [2]:
```
from PIL import Image, ImageChops, ImageEnhance
import os
import itertools
```

In [3]:
```
def convert_to_ela_image(path, quality):
    temp_filename = 'temp_file_name.jpg'
    ela_filename = 'temp_ela.png'

    image = Image.open(path).convert('RGB')
    image.save(temp_filename, 'JPEG', quality = quality)
    temp_image = Image.open(temp_filename)

    ela_image = ImageChops.difference(image, temp_image)

    extrema = ela_image.getextrema()
    max_diff = max([ex[1] for ex in extrema])
    if max_diff == 0:
        max_diff = 1
    scale = 255.0 / max_diff

    ela_image = ImageEnhance.Brightness(ela_image).enhance(scale)

    return ela_image
```

Open a real image

In [5]:
```
real_image_path = 'CASIA2/Au/Au_ani_00001.jpg'
Image.open(real_image_path)
```

Out[5]:

After converting to ELA image

In [6]:
```
convert_to_ela_image(real_image_path, 90)
```

Out[6]:

Open a fake image

In [7]:
```
fake_image_path = 'CASIA2/Tp/Tp_D_NRN_S_N_ani10171_ani00001_12458.jpg'
Image.open(fake_image_path)
```

Out[7]:

In [8]:
```
convert_to_ela_image(fake_image_path, 90)
```

Out[8]:

Dataset Preparation

In [9]:
```
image_size = (128, 128)
```

In [10]:
```
def prepare_image(image_path):
    return np.array(convert_to_ela_image(image_path, 90).resize(image_size
```

In [11]:
```
X = [] # ELA converted images
Y = [] # 0 for fake, 1 for real
```

Au => Total Images 7354, Take 2100 random images from the list

Tp => Total Images 2064

In [12]:
```
import random
path = 'CASIA2/Au/'
for dirname, _, filenames in os.walk(path):
    for filename in filenames:
        if filename.endswith('jpg') or filename.endswith('png'):
            full_path = os.path.join(dirname, filename)
            X.append(prepare_image(full_path))
            Y.append(1)
            if len(Y) % 500 == 0:
                print(f'Processing {len(Y)} images')

random.shuffle(X)
X = X[:2100]
Y = Y[:2100]
print(len(X), len(Y))
```

```
Processing 500 images
Processing 1000 images
Processing 1500 images
Processing 2000 images
Processing 2500 images
Processing 3000 images
Processing 3500 images
Processing 4000 images
Processing 4500 images
Processing 5000 images
Processing 5500 images
Processing 6000 images
Processing 6500 images
Processing 7000 images
2100 2100
```

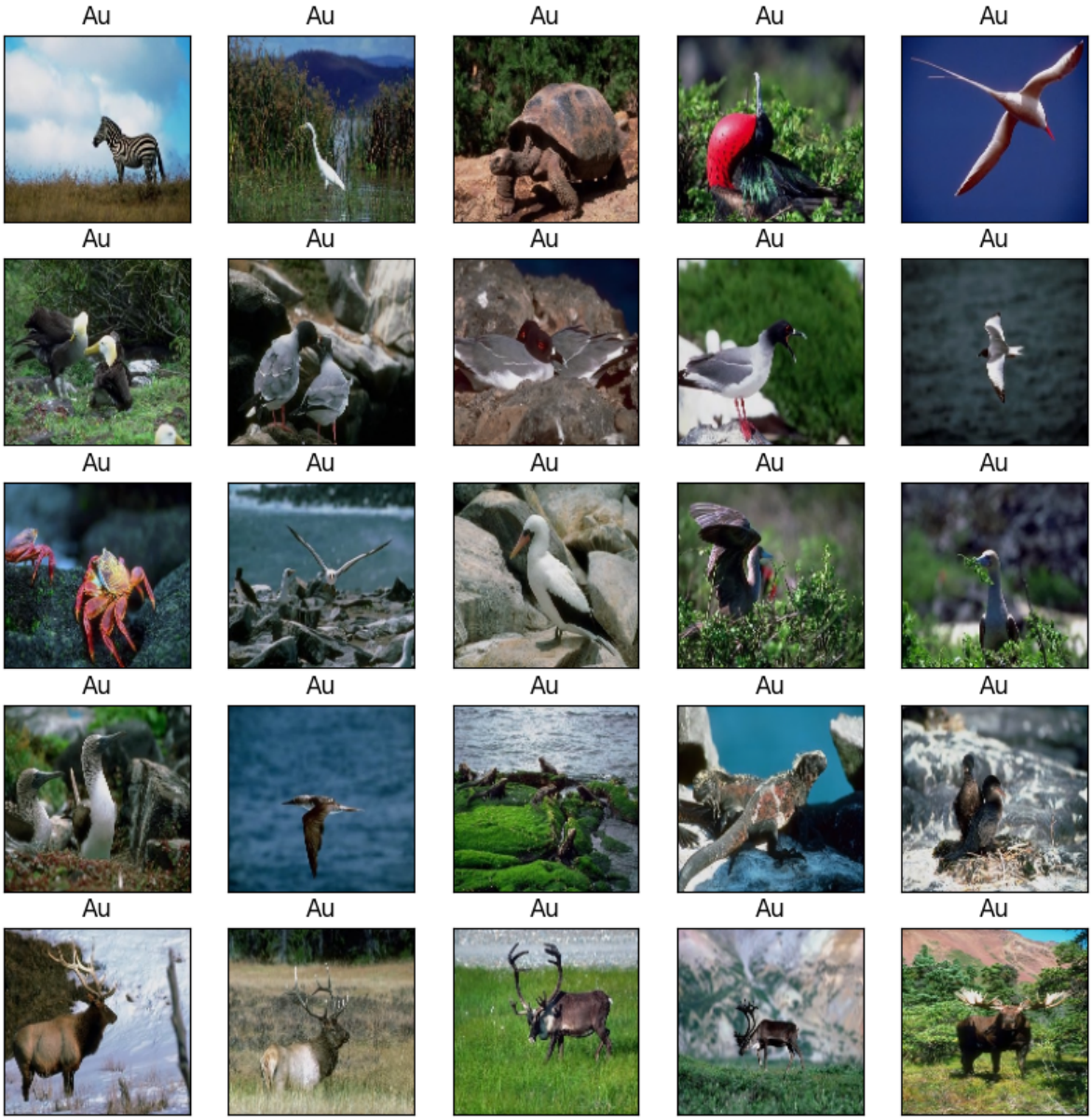
In [13]:

```
path = 'CASIA2/Tp/'
for dirname, _, filenames in os.walk(path):
    for filename in filenames:
        if filename.endswith('jpg') or filename.endswith('png'):
            full_path = os.path.join(dirname, filename)
            X.append(prepare_image(full_path))
            Y.append(0)
            if len(Y) % 500 == 0:
                print(f'Processing {len(Y)} images')

print(len(X), len(Y))
```

```
Processing 2500 images
Processing 3000 images
Processing 3500 images
Processing 4000 images
4164 4164
```

In [51]:

```
import os
import random
import matplotlib.pyplot as plt

# ... [Your existing code for importing and defining prepare_image]

def plot_images(image_list, title):
    plt.figure(figsize=(10,10))
    for i in range(25):
        plt.subplot(5,5,i+1)
        plt.xticks([])
        plt.yticks([])
        plt.imshow(image_list[i])
        plt.title(title)
    plt.show()

path = 'CASIA2/Au/'
au_images = []
for dirname, _, filenames in os.walk(path):
    for filename in filenames:
        if filename.endswith('jpg') or filename.endswith('png'):
            full_path = os.path.join(dirname, filename)
            au_images.append(prepare_image(full_path))
            if len(au_images) == 25:
                break
    if len(au_images) == 25:
        break

path = 'CASIA2/Tp/'
tp_images = []
for dirname, _, filenames in os.walk(path):
    for filename in filenames:
        if filename.endswith('jpg') or filename.endswith('png'):
            full_path = os.path.join(dirname, filename)
            tp_images.append(prepare_image(full_path))
            if len(tp_images) == 25:
                break
    if len(tp_images) == 25:
        break

# Plotting the images
plot_images(au_images, "Au")
plot_images(tp_images, "Tp")

```


Au
Au
Au
Au
Au
Au
Au
Au
Au
Au
Au
Au
Au
Au
Au
Au
Au
Au
Au
Au
Au
Au
Au
Au
Au

One Hot Encoding to Categorical

In [14]:
```
X = np.array(X)
Y = to_categorical(Y, 2)
X = X.reshape(-1, 128, 128, 3)
```

Train Test split with 80:20 ratio

In [15]:
```
X_train, X_val, Y_train, Y_val = train_test_split(X, Y, test_size = 0.2, r
X = X.reshape(-1,1,1,1)
print(len(X_train), len(Y_train))
print(len(X_val), len(Y_val))
```

```
3331 3331
833 833
```

Custom CNN Model

In [16]:
```
def build_model():
    model = Sequential()
    model.add(Conv2D(filters = 32, kernel_size = (5, 5), padding = 'valid'
    model.add(Conv2D(filters = 32, kernel_size = (5, 5), padding = 'valid'
    model.add(MaxPool2D(pool_size = (2, 2)))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(256, activation = 'relu'))
    model.add(Dropout(0.5))
    model.add(Dense(2, activation = 'softmax'))
    return model
```

In [17]:
```
model = build_model()
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 124, 124, 32)      2432

 conv2d_1 (Conv2D)           (None, 120, 120, 32)      25632

 max_pooling2d (MaxPooling2D  (None, 60, 60, 32)       0
 )

 dropout (Dropout)           (None, 60, 60, 32)        0

 flatten (Flatten)           (None, 115200)            0

 dense (Dense)               (None, 256)               29491456

 dropout_1 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 2)                 514

=================================================================
Total params: 29,520,034
Trainable params: 29,520,034
Non-trainable params: 0
_________________________________________________________________
```

In [18]:
```
epochs = 30
batch_size = 32

```

In [19]:
```
init_lr = 1e-4
optimizer = Adam(lr = init_lr, decay = init_lr/epochs)
```

```
C:\Users\Nawaz\anaconda3\envs\media\lib\site-packages\keras\optimizers\optimizer_v2\adam.py:114: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
```

In [20]:
```
model.compile(optimizer = optimizer, loss = 'binary_crossentropy', metrics
```

In [21]:
```
early_stopping = EarlyStopping(monitor = 'val_acc',
                               min_delta = 0,
                               patience = 2,
                               verbose = 0,
                               mode = 'auto')
```

In [22]:

```
hist = model.fit(X_train,
                 Y_train,
                 batch_size = batch_size,
                 epochs = epochs,
                validation_data = (X_val, Y_val),
                callbacks = [early_stopping])
```

```
Epoch 1/30
105/105 [==============================] - ETA: 0s - loss: 0.5010 - accuracy:
0.7544WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 8s 19ms/step - loss: 0.5010 - accu
racy: 0.7544 - val_loss: 0.3261 - val_accuracy: 0.8896
Epoch 2/30
105/105 [==============================] - ETA: 0s - loss: 0.3036 - accuracy:
0.8973WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.3036 - accu
racy: 0.8973 - val_loss: 0.2523 - val_accuracy: 0.9160
Epoch 3/30
104/105 [============================>.] - ETA: 0s - loss: 0.2477 - accuracy:
0.9186WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.2485 - accu
racy: 0.9183 - val_loss: 0.2277 - val_accuracy: 0.9196
Epoch 4/30
101/105 [===========================>..] - ETA: 0s - loss: 0.2276 - accuracy:
0.9177WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.2330 - accu
racy: 0.9168 - val_loss: 0.2180 - val_accuracy: 0.9232
Epoch 5/30
102/105 [============================>.] - ETA: 0s - loss: 0.2001 - accuracy:
0.9265WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.2005 - accu
racy: 0.9264 - val_loss: 0.2219 - val_accuracy: 0.9160
Epoch 6/30
103/105 [============================>.] - ETA: 0s - loss: 0.1930 - accuracy:
0.9272WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.1921 - accu
racy: 0.9273 - val_loss: 0.2445 - val_accuracy: 0.9064
Epoch 7/30
101/105 [===========================>..] - ETA: 0s - loss: 0.1764 - accuracy:
0.9369WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.1751 - accu
racy: 0.9373 - val_loss: 0.2024 - val_accuracy: 0.9244
Epoch 8/30
102/105 [============================>.] - ETA: 0s - loss: 0.1571 - accuracy:
0.9406WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.1570 - accu
racy: 0.9409 - val_loss: 0.2412 - val_accuracy: 0.9064
Epoch 9/30
101/105 [===========================>..] - ETA: 0s - loss: 0.1372 - accuracy:
0.9505WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.1379 - accu
racy: 0.9508 - val_loss: 0.2319 - val_accuracy: 0.9076
Epoch 10/30
102/105 [============================>.] - ETA: 0s - loss: 0.1298 - accuracy:
0.9525WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
```

```
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.1299 - accu
racy: 0.9529 - val_loss: 0.1881 - val_accuracy: 0.9304
Epoch 11/30
104/105 [============================>.] - ETA: 0s - loss: 0.1102 - accuracy:
0.9633WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.1101 - accu
racy: 0.9634 - val_loss: 0.1766 - val_accuracy: 0.9388
Epoch 12/30
104/105 [============================>.] - ETA: 0s - loss: 0.1028 - accuracy:
0.9615WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.1028 - accu
racy: 0.9616 - val_loss: 0.1765 - val_accuracy: 0.9376
Epoch 13/30
104/105 [============================>.] - ETA: 0s - loss: 0.0910 - accuracy:
0.9697WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.0910 - accu
racy: 0.9697 - val_loss: 0.2175 - val_accuracy: 0.9280
Epoch 14/30
103/105 [============================>.] - ETA: 0s - loss: 0.0767 - accuracy:
0.9760WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.0764 - accu
racy: 0.9763 - val_loss: 0.1710 - val_accuracy: 0.9388
Epoch 15/30
102/105 [============================>.] - ETA: 0s - loss: 0.0744 - accuracy:
0.9776WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.0757 - accu
racy: 0.9769 - val_loss: 0.1932 - val_accuracy: 0.9328
Epoch 16/30
104/105 [============================>.] - ETA: 0s - loss: 0.0700 - accuracy:
0.9790WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.0699 - accu
racy: 0.9790 - val_loss: 0.1750 - val_accuracy: 0.9376
Epoch 17/30
105/105 [==============================] - ETA: 0s - loss: 0.0567 - accuracy:
0.9811WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.0567 - accu
racy: 0.9811 - val_loss: 0.2171 - val_accuracy: 0.9352
Epoch 18/30
104/105 [============================>.] - ETA: 0s - loss: 0.0511 - accuracy:
0.9868WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.0511 - accu
racy: 0.9868 - val_loss: 0.1886 - val_accuracy: 0.9436
Epoch 19/30
105/105 [==============================] - ETA: 0s - loss: 0.0437 - accuracy:
0.9898WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.0437 - accu
racy: 0.9898 - val_loss: 0.1894 - val_accuracy: 0.9484
```

```
Epoch 20/30
104/105 [============================>.] - ETA: 0s - loss: 0.0430 - accuracy:
0.9871WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.0430 - accu
racy: 0.9871 - val_loss: 0.1773 - val_accuracy: 0.9436
Epoch 21/30
102/105 [============================>.] - ETA: 0s - loss: 0.0439 - accuracy:
0.9859WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.0437 - accu
racy: 0.9859 - val_loss: 0.1860 - val_accuracy: 0.9412
Epoch 22/30
105/105 [==============================] - ETA: 0s - loss: 0.0338 - accuracy:
0.9895WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.0338 - accu
racy: 0.9895 - val_loss: 0.1976 - val_accuracy: 0.9424
```

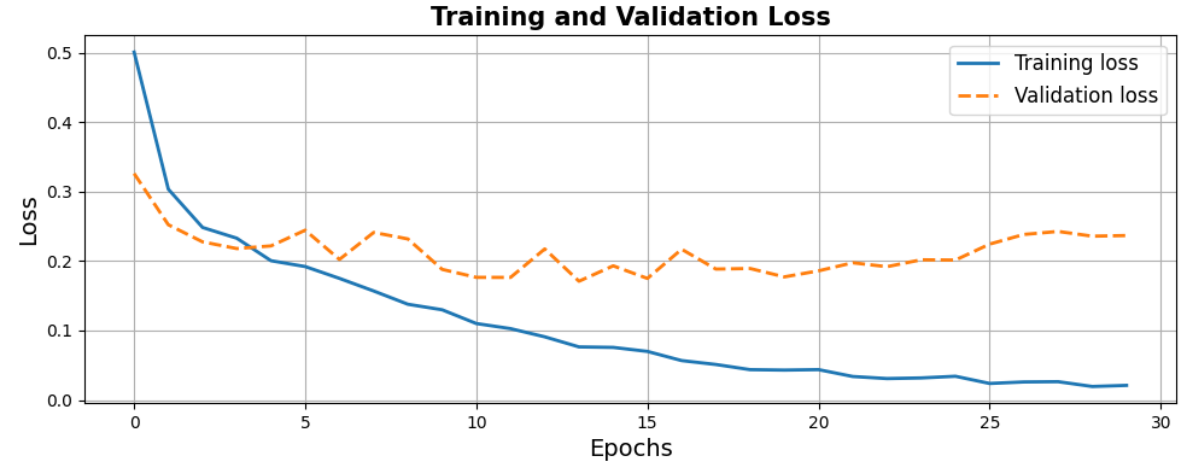
```
Epoch 23/30
103/105 [============================>.] - ETA: 0s - loss: 0.0307 - accuracy:
0.9903WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.0308 - accu
racy: 0.9904 - val_loss: 0.1919 - val_accuracy: 0.9412
Epoch 24/30
104/105 [============================>.] - ETA: 0s - loss: 0.0315 - accuracy:
0.9907WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.0317 - accu
racy: 0.9904 - val_loss: 0.2019 - val_accuracy: 0.9400
Epoch 25/30
102/105 [============================>.] - ETA: 0s - loss: 0.0336 - accuracy:
0.9920WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.0341 - accu
racy: 0.9916 - val_loss: 0.2015 - val_accuracy: 0.9424
Epoch 26/30
101/105 [============================>..] - ETA: 0s - loss: 0.0237 - accuracy:
0.9938WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.0238 - accu
racy: 0.9937 - val_loss: 0.2245 - val_accuracy: 0.9412
Epoch 27/30
104/105 [============================>.] - ETA: 0s - loss: 0.0259 - accuracy:
0.9928WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.0260 - accu
racy: 0.9928 - val_loss: 0.2384 - val_accuracy: 0.9304
Epoch 28/30
102/105 [============================>.] - ETA: 0s - loss: 0.0263 - accuracy:
0.9917WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.0263 - accu
racy: 0.9919 - val_loss: 0.2427 - val_accuracy: 0.9328
Epoch 29/30
102/105 [============================>.] - ETA: 0s - loss: 0.0196 - accuracy:
0.9954WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.0194 - accu
racy: 0.9955 - val_loss: 0.2359 - val_accuracy: 0.9388
Epoch 30/30
102/105 [============================>.] - ETA: 0s - loss: 0.0212 - accuracy:
0.9936WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which
is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
105/105 [==============================] - 2s 16ms/step - loss: 0.0210 - accu
racy: 0.9937 - val_loss: 0.2368 - val_accuracy: 0.9412
```

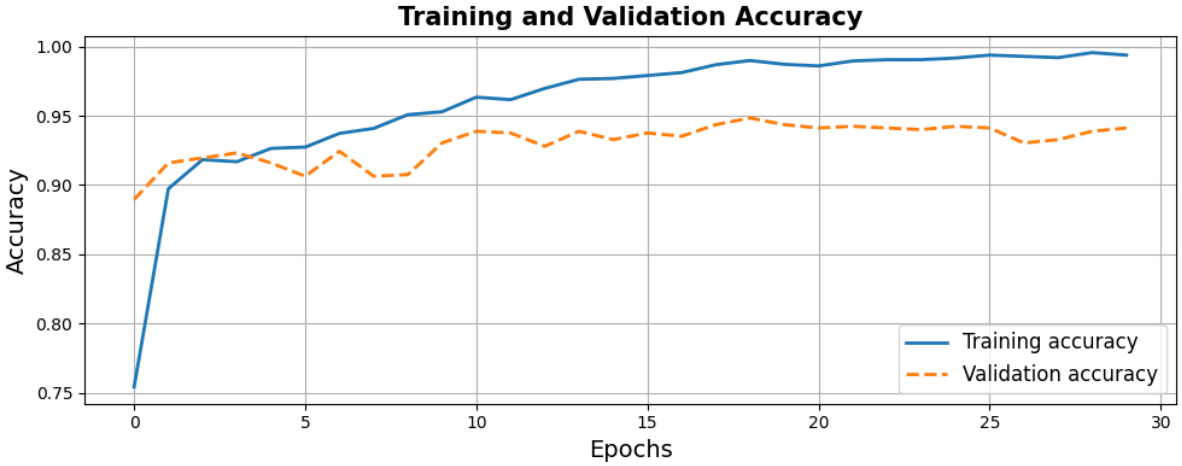
In [23]:

```
1 model.save('model_casia_run_hamad.h5')
```

In [24]:

```
import matplotlib.pyplot as plt

# Set a consistent color palette
colors = {
    'train': '#1f77b4',  # blue
    'val': '#ff7f0e'     # orange
}

fig, ax = plt.subplots(2, 1, figsize=(10, 8))  # Adjusted the figure size

# Training and validation loss
ax[0].plot(hist.history['loss'], color=colors['train'], label="Training lo
ax[0].plot(hist.history['val_loss'], color=colors['val'], label="Validatio
ax[0].set_title('Training and Validation Loss', fontweight='bold', fontsiz
ax[0].set_xlabel('Epochs', fontsize=14)
ax[0].set_ylabel('Loss', fontsize=14)
ax[0].legend(loc='upper right', fontsize=12)
ax[0].grid(True)

# Training and validation accuracy
ax[1].plot(hist.history['accuracy'], color=colors['train'], label="Trainin
ax[1].plot(hist.history['val_accuracy'], color=colors['val'], label="Valid
ax[1].set_title('Training and Validation Accuracy', fontweight='bold', fon
ax[1].set_xlabel('Epochs', fontsize=14)
ax[1].set_ylabel('Accuracy', fontsize=14)
ax[1].legend(loc='lower right', fontsize=12)
ax[1].grid(True)

plt.tight_layout()  # Adjust the spacing between plots
plt.show()

```


Training and Validation Loss
Training loss
Validation loss
Loss
Epochs
0.0
0.1
0.2
0.3
0.4
0.5
0
5
10
15
20
25
30

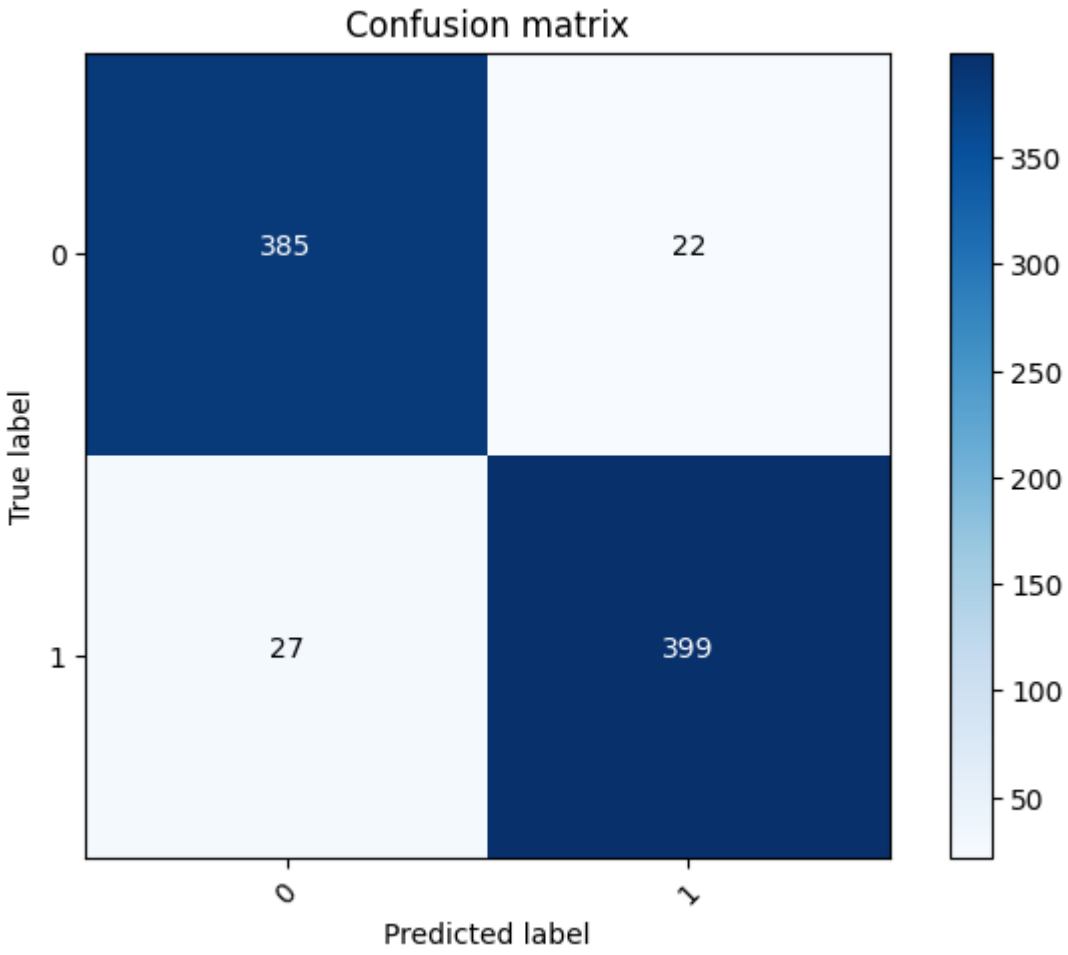

Training and Validation Accuracy
Training accuracy
Validation accuracy
Accuracy
Epochs
0.75
0.80
0.85
0.90
0.95
1.00
0
5
10
15
20
25
30

In [26]:

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [27]:
```
# Predict the values from the validation dataset
Y_pred = model.predict(X_val)
# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_pred,axis = 1)
# Convert validation observations to one hot vectors
Y_true = np.argmax(Y_val,axis = 1)
# compute the confusion matrix
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
# plot the confusion matrix
plot_confusion_matrix(confusion_mtx, classes = range(2))
```

```
27/27 [==============================] - 1s 16ms/step
```



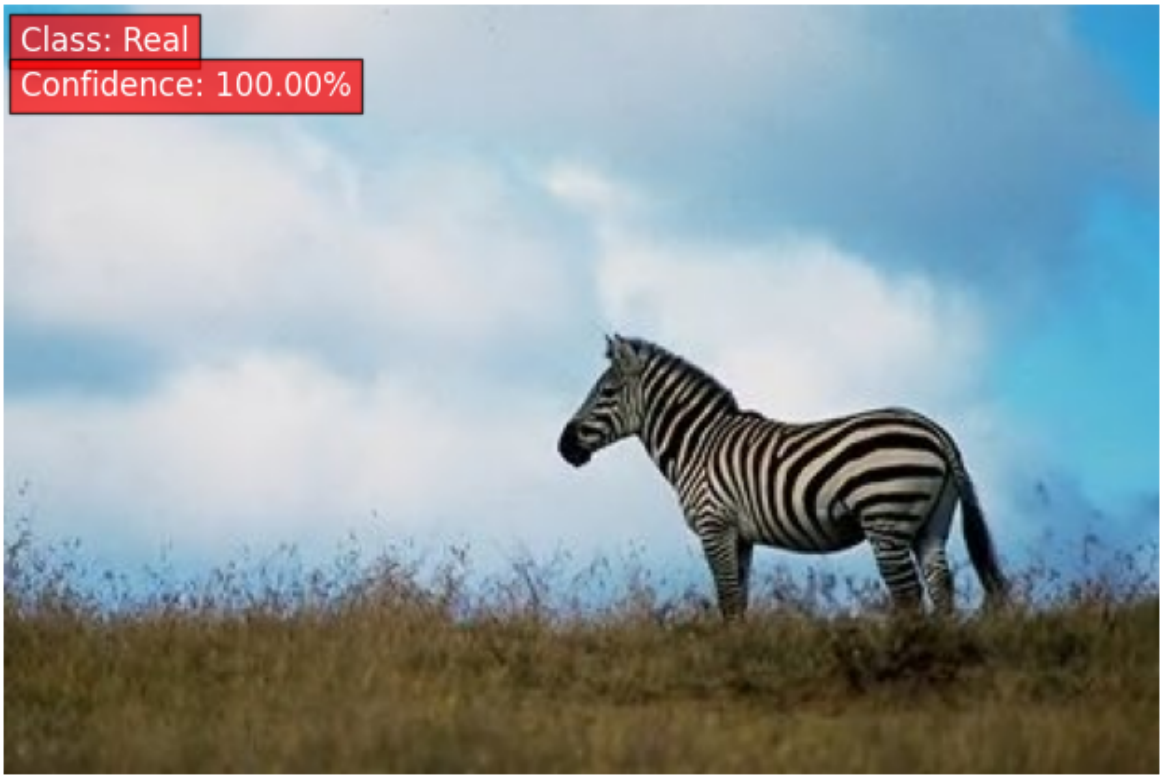
Testing Prediction

In [37]:
```
class_names = ['Forgery', 'Real']
```

In [38]:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# Assuming the prepare_image function loads and preprocesses the image
def prepare_image(image_path):
    img = mpimg.imread(image_path)
    # ... any other preprocessing steps you might have ...
    return img

real_image_path = 'CASIA2/Au/Au_ani_00001.jpg'
image = prepare_image(real_image_path)
image_for_model = image.reshape(-1, 128, 128, 3)  # reshaping for the mode

y_pred = model.predict(image_for_model)
y_pred_class = np.argmax(y_pred, axis = 1)[0]
print('Image Shape',image.shape)

# Display the image using matplotlib
plt.figure(figsize=(8, 6))  # adjust the figure size
plt.imshow(image)
plt.axis('off')  # turn off axis numbers and ticks

# Overlay the predicted class and confidence level on the image
confidence = np.amax(y_pred) * 100
plt.text(5, 15, f"Class: {class_names[y_pred_class]}", color='white', font
plt.text(5, 30, f"Confidence: {confidence:0.2f}%", color='white', fontsize

plt.show()

```

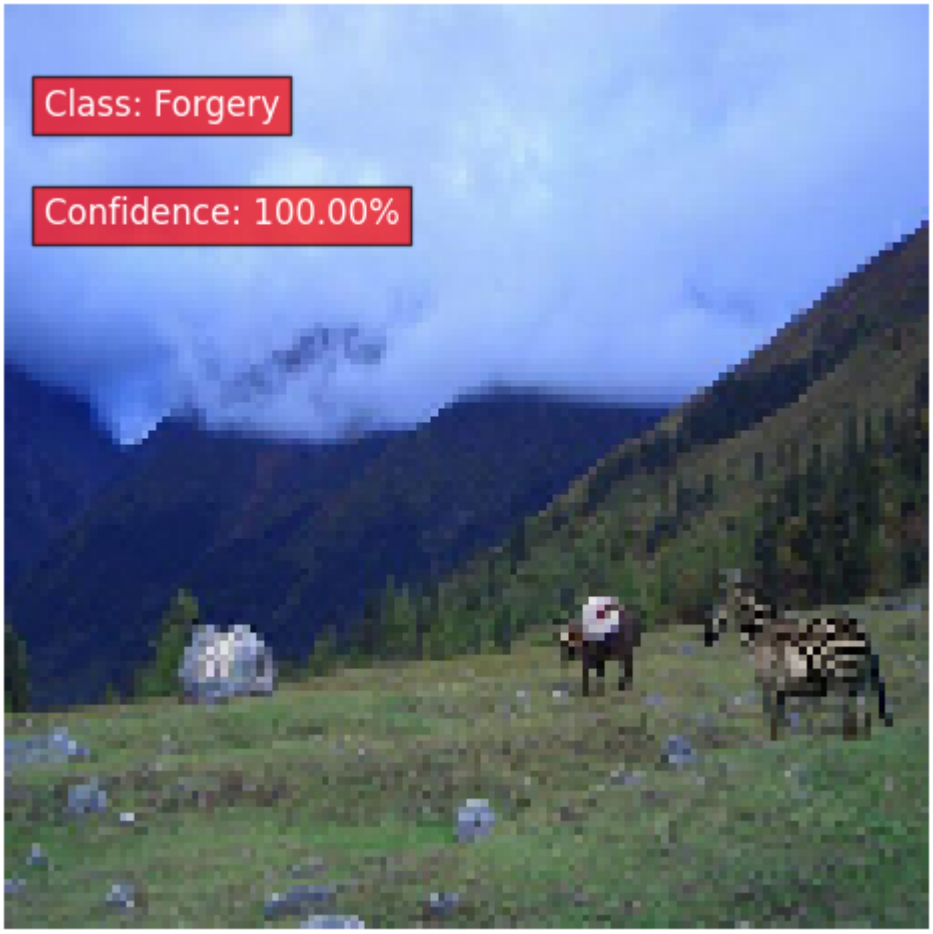
```
1/1 [==============================] - 0s 19ms/step
Image Shape (256, 384, 3)
```


Class: Real
Confidence: 100.00%

In [43]:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def prepare_image(image_path):
    img = cv2.imread(image_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert from BGR to RGB
    img = cv2.resize(img, (128, 128))  # Resize the image
    img = img / 255.0  # Normalize
    return img

fake_image_path = 'CASIA2/Tp/Tp_D_NRN_S_N_ani10171_ani00001_12458.jpg'
image = prepare_image(fake_image_path)
image_for_model = image.reshape(-1, 128, 128, 3)  # reshaping for the mode

y_pred = model.predict(image_for_model)
y_pred_class = np.argmax(y_pred, axis=1)[0]
confidence = np.amax(y_pred) * 100

# If confidence is below 40%, label it as "Forgery"
if confidence < 40:
    label = "Forgery"
    confidence = 100 - confidence  # Flip the confidence to represent the
else:
    label = class_names[y_pred_class]

# Display the image using matplotlib
plt.figure(figsize=(8, 6))  # adjust the figure size
plt.imshow(image)
plt.axis('off')  # turn off axis numbers and ticks

# Overlay the predicted class and confidence level on the image
plt.text(5, 15, f"Class: {label}", color='white', fontsize=12, bbox=dict(f
plt.text(5, 30, f"Confidence: {confidence:0.2f}%", color='white', fontsize

plt.show()

```

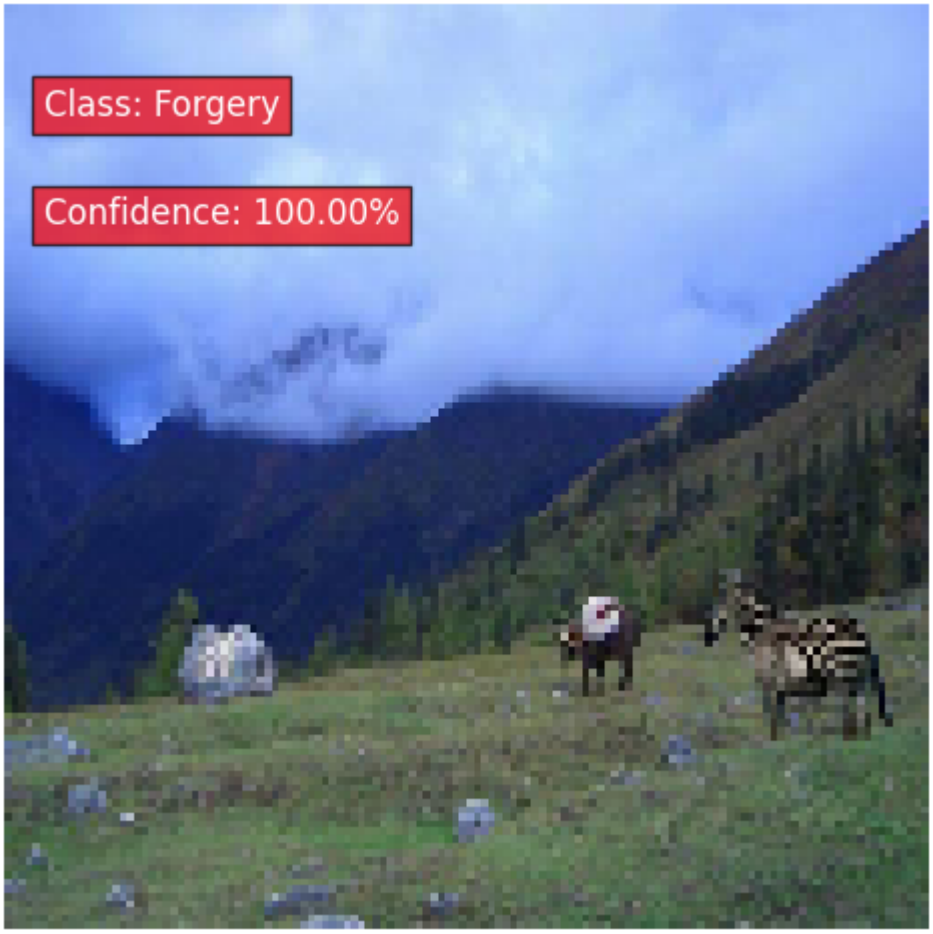
```
1/1 [==============================] - 0s 16ms/step
```


Class: Forgery
Confidence: 100.00%

In [57]:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def prepare_image(image_path):
    img = cv2.imread(image_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert from BGR to RGB
    img = cv2.resize(img, (128, 128))  # Resize the image
    img = img / 255.0  # Normalize
    return img

fake_image_path = 'CASIA2/Tp/Tp_D_NRN_S_N_ani10171_ani00001_12458.jpg'
image = prepare_image(fake_image_path)
image_for_model = image.reshape(-1, 128, 128, 3)  # reshaping for the mode

y_pred = model.predict(image_for_model)
y_pred_class = np.argmax(y_pred, axis=1)[0]

# Display the image using matplotlib
plt.figure(figsize=(8, 6))  # adjust the figure size
plt.imshow(image)
plt.axis('off')  # turn off axis numbers and ticks

# Overlay the predicted class and confidence level on the image
confidence = np.amax(y_pred) * 100
plt.text(5, 15, f"Class: {class_names[y_pred_class]}", color='white', font
plt.text(5, 30, f"Confidence: {confidence:0.2f}%", color='white', fontsize

plt.show()

```

```
1/1 [==============================] - 0s 20ms/step
```



In [58]:
```
# fake_image = os.listdir('CASIA2/Tp/')
# correct = 0
# total = 0
# for file_name in fake_image:
#     if file_name.endswith('jpg') or filename.endswith('png'):
#         fake_image_path = os.path.join('CASIA2/Tp/', file_name)
#         image = prepare_image(fake_image_path)
#         image = image.reshape(-1, 128, 128, 3)
#         y_pred = model.predict(image)
#         y_pred_class = np.argmax(y_pred, axis = 1)[0]
#         total += 1
#         if y_pred_class == 0:
#             correct += 1
#             print(f'Class: {class_names[y_pred_class]} Confidence: {np.a
```

In [55]:
```
# print(f'Total: {total}, Correct: {correct}, Acc: {correct / total * 100.
```

In [52]:
```
# real_image = os.listdir('CASIA2/Au/')
# correct_r = 0
# total_r = 0
# for file_name in real_image:
#     if file_name.endswith('jpg') or filename.endswith('png'):
#         real_image_path = os.path.join('CASIA2/Au/', file_name)
#         image = prepare_image(real_image_path)
#         image = image.reshape(-1, 128, 128, 3)
#         y_pred = model.predict(image)
#         y_pred_class = np.argmax(y_pred, axis = 1)[0]
#         total_r += 1
#         if y_pred_class == 1:
#             correct_r += 1
#             print(f'Class: {class_names[y_pred_class]} Confidence: {np.a
```

In [54]:
```
# correct += correct_r
# total += total_r
# print(f'Total: {total_r}, Correct: {correct_r}, Acc: {correct_r / total_
# print(f'Total: {total}, Correct: {correct}, Acc: {correct / total * 100.
```