In [4]:
```
import tensorflow as tf
import random
import os
import numpy as np
from sklearn.metrics import roc_curve, auc, precision_recall_curve, average_precision_score, confusion_matrix
import pandas as pd
import matplotlib.pyplot as plt
```

Path system

In [5]:
```
# you need the current working directory NB: works both windows and linux
current_working_directory = os.getcwd()
current_working_directory = os.path.dirname(current_working_directory)

# get the directory where I want to download the dataset
path_of_download = os.path.join(*['..', current_working_directory,'CVSF', 'Datasets', 'cookies_vs_chihuahua'])
print(f"[DIR] The directory of the current dataset is {path_of_download}")
```

```
[DIR] The directory of the current dataset is c:\CVSF\Datasets\cookies_vs_chihuahua
```

Dataset function

In [6]:
```
# here let s do some functions that we can re-use also for other assignment
def load_the_data_and_the_labels(data_set_path: str, target_size: tuple or None = None):
    try:
        dataset, labels, name_of_the_labels = list(), list(), list()
        # let s loop here and we try to discover how many class we have
        for class_number, class_name in enumerate(os.listdir(data_set_path)):
            full_path_the_data = os.path.join(data_set_path, class_name)
            print(f"[WALK] I am walking into {full_path_the_data}")

            # add the list to nam _list
            name_of_the_labels.append(class_name)

            for single_image in os.listdir(f"{full_path_the_data}"):
                full_path_to_image = os.path.join(*[full_path_the_data, single_image])

                # add the class number
                labels.append(class_number)

                if target_size is None:
                    # let s load the image
                    image = tf.keras.utils.load_img(full_path_to_image)
                else:
                    image = tf.keras.utils.load_img(full_path_to_image, target_size=target_size)

                # transform PIL object in image
```

```
                image = tf.keras.utils.img_to_array(image)

                # add the image to the ds list
                dataset.append(image)

        return np.array(dataset, dtype='uint8'), np.array(labels, dtype='int'), name_of_the_labels
    except Exception as ex:
        print(f"[EXCEPTION] load the data and the labels throws exceptions {ex}")
```

load train set

In [ ]:
```
train_dataset, train_labels, train_labels_names = load_the_data_and_the_labels(
    "C:/CVSF/Datasets/cookies_vs_chihuahua/train", target_size=(224, 224))
```

```
[WALK] I am walking into C:/CVSF/Datasets/cookies_vs_chihuahua/train\chihuahua
[WALK] I am walking into C:/CVSF/Datasets/cookies_vs_chihuahua/train\muffin
```

In [8]:
```
train_dataset.shape
```

Out[8]:
```
(4733, 224, 224, 3)
```

load test set

In [ ]:
```
test_dataset, test_labels, test_labels_name = load_the_data_and_the_labels(
    "C:/CVSF/Datasets/cookies_vs_chihuahua/test", target_size=(224, 224))
```

```
[WALK] I am walking into C:/CVSF/Datasets/cookies_vs_chihuahua/test\chihuahua
[WALK] I am walking into C:/CVSF/Datasets/cookies_vs_chihuahua/test\muffin
```

In [10]:
```
test_dataset[0].shape, test_dataset[1].shape, test_dataset[2]
```

```
Out[10]: ((224, 224, 3),
          (224, 224, 3),
          array([[[255, 255, 255],
                  [255, 255, 255],
                  [255, 255, 255],
                  ...,
                  [255, 255, 255],
                  [255, 255, 255],
                  [255, 255, 255]],

                 [[255, 255, 255],
                  [255, 255, 255],
                  [255, 255, 255],
                  ...,
                  [255, 255, 255],
                  [255, 255, 255],
                  [255, 255, 255]],

                 [[255, 255, 255],
                  [255, 255, 255],
                  [255, 255, 255],
                  ...,
                  [255, 255, 255],
                  [255, 255, 255],
                  [255, 255, 255]],

                 ...,

                 [[255, 255, 255],
                  [255, 255, 255],
                  [255, 255, 255],
                  ...,
                  [255, 255, 255],
                  [255, 255, 255],
                  [255, 255, 255]],

                 [[255, 255, 255],
                  [255, 255, 255],
                  [255, 255, 255],
                  ...,
                  [255, 255, 255],
                  [255, 255, 255],
                  [255, 255, 255]],

                 [[255, 255, 255],
                  [255, 255, 255],
                  [255, 255, 255],
                  ...,
                  [255, 255, 255],
```

```
        [255, 255, 255],
        [255, 255, 255]]], dtype=uint8))
```

normalize the data

In [11]:
```
from sklearn.model_selection import train_test_split

# Splitting data into training and testing sets with 30% test size
x_train, x_val, y_train, y_val = train_test_split(train_dataset, train_labels, test_size=0.3, random_state=42)
```

In [12]:
```
x_train_normalized = x_train / 255.0
x_val_normalized = x_val / 255.0
x_test_normalized = test_dataset / 255.0
```

create a cnn with the following characteristics:

a. Input layer b. Data augmentation, with random flip and random rotation. c. Two hidden layers each composed with the following characteristics: 16 conv2d units, max pooling 2d and batch normalization, the second one should have 24 conv2d units max pooling 2d and batch normalization. d. After this, add a flatten layer and a dense layer with 8 units e. Add the final classifier (a dense layer) with the correct number of output and activation

And Compiling the model using Adam and Binary cross entropy

In [13]:
```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, BatchNormalization, RandomFlip, RandomRotation
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import BinaryCrossentropy
# Create the CNN model
model = Sequential()

# Input Layer (224, 224, 3)
model.add(tf.keras.layers.InputLayer(input_shape=(224, 224, 3)))

# Data Augmentation Layer
model.add(RandomFlip("horizontal"))
model.add(RandomRotation(0.2))

# First convolutional layer: 16 filters, max pooling, and batch normalization
model.add(Conv2D(16, (3,3), activation='relu', padding='same'))
model.add(MaxPooling2D((2,2)))
model.add(BatchNormalization())

# Second convolutional layer: 24 filters, max pooling, and batch normalization
model.add(Conv2D(24, (3,3), activation='relu', padding='same'))
model.add(MaxPooling2D((2,2)))
model.add(BatchNormalization())
```

```
# Flattening layer
model.add(Flatten())

# Dense layer with 8 units
model.add(Dense(8, activation='relu'))

# Output layer with correct number of classes and activation function
num_classes = len(np.unique(train_labels))  # Assuming binary classification
output_activation = 'sigmoid' if num_classes == 2 else 'softmax'
model.add(Dense(1 if num_classes == 2 else num_classes, activation=output_activation))

# Compile the model
model.compile(optimizer=Adam(), loss=BinaryCrossentropy(), metrics=['accuracy'])

# Print the model summary
model.summary()
```

```
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 random_flip (RandomFlip)    (None, 224, 224, 3)       0

 random_rotation (RandomRota  (None, 224, 224, 3)      0
 tion)

 conv2d (Conv2D)             (None, 224, 224, 16)      448

 max_pooling2d (MaxPooling2D  (None, 112, 112, 16)     0
 )

 batch_normalization (BatchN  (None, 112, 112, 16)     64
 ormalization)

 conv2d_1 (Conv2D)           (None, 112, 112, 24)      3480

 max_pooling2d_1 (MaxPooling  (None, 56, 56, 24)       0
 2D)

 batch_normalization_1 (Batc  (None, 56, 56, 24)       96
 hNormalization)

 flatten (Flatten)           (None, 75264)             0

 dense (Dense)               (None, 8)                 602120

 dense_1 (Dense)             (None, 1)                 9

=================================================================
Total params: 606,217
Trainable params: 606,137
Non-trainable params: 80
_________________________________________________________________
```

Training the model with a btach size of 64 and 30 epochs all together.

In [14]:
```
history = model.fit(
    x_train_normalized, y_train,
    validation_data=(x_val, y_val),
    batch_size=64,
    epochs=30,
```

```
    verbose=1
)
```

```
Epoch 1/30
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
52/52 [==============================] - 52s 943ms/step - loss: 0.9950 - accuracy: 0.5804 - val_loss: 125.0984 - val_accuracy: 0.7049
Epoch 2/30
52/52 [==============================] - 47s 909ms/step - loss: 0.6260 - accuracy: 0.6342 - val_loss: 902.9070 - val_accuracy: 0.5704
Epoch 3/30
52/52 [==============================] - 46s 879ms/step - loss: 0.6184 - accuracy: 0.6849 - val_loss: 1059.3723 - val_accuracy: 0.4930
Epoch 4/30
52/52 [==============================] - 47s 910ms/step - loss: 0.6431 - accuracy: 0.6079 - val_loss: 579.6101 - val_accuracy: 0.5268
Epoch 5/30
52/52 [==============================] - 55s 1s/step - loss: 0.6380 - accuracy: 0.6565 - val_loss: 970.4413 - val_accuracy: 0.5204
Epoch 6/30
52/52 [==============================] - 55s 1s/step - loss: 0.6123 - accuracy: 0.6589 - val_loss: 270.0031 - val_accuracy: 0.6648
Epoch 7/30
52/52 [==============================] - 57s 1s/step - loss: 0.5942 - accuracy: 0.6731 - val_loss: 1460.0958 - val_accuracy: 0.5634
Epoch 8/30
52/52 [==============================] - 55s 1s/step - loss: 0.6031 - accuracy: 0.6650 - val_loss: 590.7426 - val_accuracy: 0.5197
Epoch 9/30
52/52 [==============================] - 55s 1s/step - loss: 0.5989 - accuracy: 0.7024 - val_loss: 738.2090 - val_accuracy: 0.5380
Epoch 10/30
52/52 [==============================] - 54s 1s/step - loss: 0.5699 - accuracy: 0.6894 - val_loss: 264.5912 - val_accuracy: 0.6549
Epoch 11/30
52/52 [==============================] - 51s 976ms/step - loss: 0.5286 - accuracy: 0.7452 - val_loss: 1153.1038 - val_accuracy: 0.5324
Epoch 12/30
52/52 [==============================] - 72s 1s/step - loss: 0.5387 - accuracy: 0.7386 - val_loss: 6769.4937 - val_accuracy: 0.5324
Epoch 13/30
52/52 [==============================] - 77s 1s/step - loss: 0.5344 - accuracy: 0.7365 - val_loss: 1681.3934 - val_accuracy: 0.5711
Epoch 14/30
52/52 [==============================] - 79s 2s/step - loss: 0.5184 - accuracy: 0.7492 - val_loss: 1816.5609 - val_accuracy: 0.6204
Epoch 15/30
52/52 [==============================] - 76s 1s/step - loss: 0.5272 - accuracy: 0.7452 - val_loss: 884.9794 - val_accuracy: 0.7021
Epoch 16/30
52/52 [==============================] - 76s 1s/step - loss: 0.5168 - accuracy: 0.7700 - val_loss: 4363.8374 - val_accuracy: 0.5542
Epoch 17/30
52/52 [==============================] - 76s 1s/step - loss: 0.4945 - accuracy: 0.7730 - val_loss: 3614.0029 - val_accuracy: 0.5683
Epoch 18/30
52/52 [==============================] - 77s 1s/step - loss: 0.4900 - accuracy: 0.7827 - val_loss: 10316.8584 - val_accuracy: 0.5423
Epoch 19/30
52/52 [==============================] - 76s 1s/step - loss: 0.4694 - accuracy: 0.7917 - val_loss: 435.3255 - val_accuracy: 0.7662
Epoch 20/30
52/52 [==============================] - 76s 1s/step - loss: 0.4455 - accuracy: 0.8074 - val_loss: 799.9057 - val_accuracy: 0.7472
```

```
Epoch 21/30
52/52 [==============================] - 77s 1s/step - loss: 0.4213 - accuracy: 0.8295 - val_loss: 615.1035 - val_accuracy: 0.7563
Epoch 22/30
52/52 [==============================] - 78s 1s/step - loss: 0.4084 - accuracy: 0.8394 - val_loss: 8692.3291 - val_accuracy: 0.5444
Epoch 23/30
52/52 [==============================] - 77s 1s/step - loss: 0.4694 - accuracy: 0.8430 - val_loss: 101.5703 - val_accuracy: 0.6880
Epoch 24/30
52/52 [==============================] - 76s 1s/step - loss: 0.3933 - accuracy: 0.8512 - val_loss: 146.3092 - val_accuracy: 0.7070
Epoch 25/30
52/52 [==============================] - 80s 2s/step - loss: 0.3739 - accuracy: 0.8624 - val_loss: 1779.2628 - val_accuracy: 0.5810
Epoch 26/30
52/52 [==============================] - 76s 1s/step - loss: 0.3732 - accuracy: 0.8696 - val_loss: 435.5845 - val_accuracy: 0.6451
Epoch 27/30
52/52 [==============================] - 76s 1s/step - loss: 0.3613 - accuracy: 0.8690 - val_loss: 950.4363 - val_accuracy: 0.5866
Epoch 28/30
52/52 [==============================] - 78s 1s/step - loss: 0.3518 - accuracy: 0.8738 - val_loss: 904.6357 - val_accuracy: 0.6176
Epoch 29/30
52/52 [==============================] - 76s 1s/step - loss: 0.3402 - accuracy: 0.8841 - val_loss: 1481.1353 - val_accuracy: 0.5556
Epoch 30/30
52/52 [==============================] - 76s 1s/step - loss: 0.3339 - accuracy: 0.8856 - val_loss: 721.7999 - val_accuracy: 0.7070
```

Evaluate the model and report the accuracy.

In [15]:
```
loss, accuracy = model.evaluate(x_test_normalized, test_labels, verbose=0)
print(f"Test Loss: {loss:.4f}, Test Accuracy: {accuracy:.4f}")
```

```
Test Loss: 0.4230, Test Accuracy: 0.8226
```

In [16]:
```
model.save('cnn_model.h5')
```

Make prediction with the test set and use a threshold of 0.5 as boundaries decision between the classes.

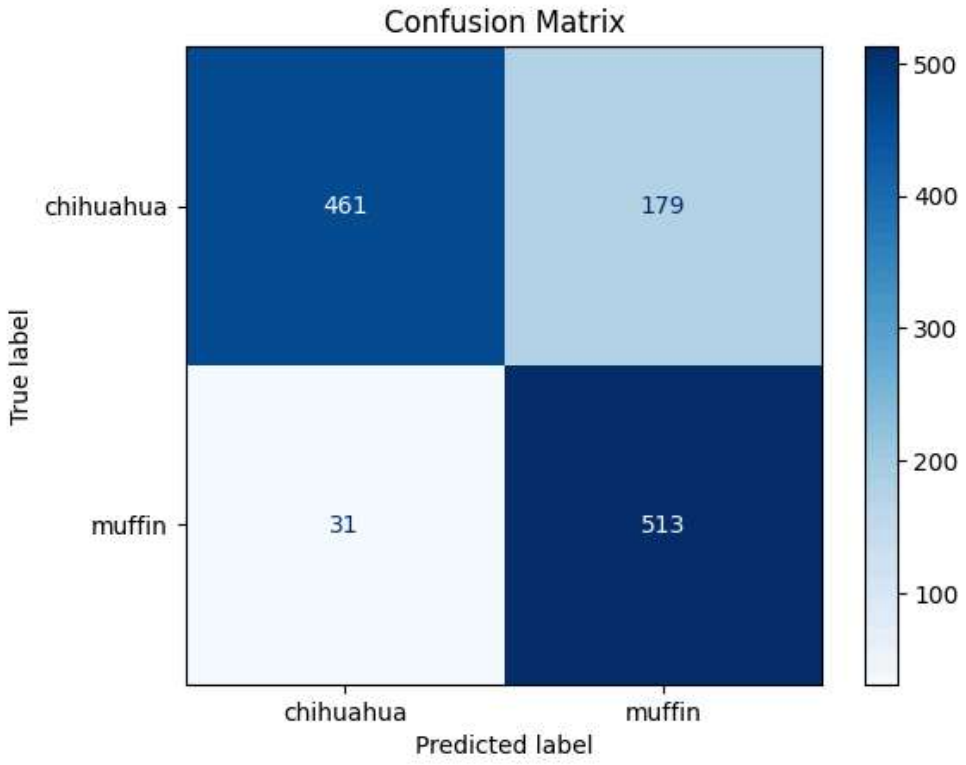
In [59]:
```
# Predict on test set
y_pred_probs = model.predict(x_test_normalized)
y_pred = (y_pred_probs > 0.5).astype(int)  # Threshold of 0.5


for i in range(10):

    print(f"Predicted: {test_labels_name[y_pred[i][0]]}, Actual: {test_labels_name[test_labels[i]]}")
    print(f"Predicted: {y_pred[i][0]}, Actual:{ test_labels[i]}")
```

```
37/37 [==============================] - 3s 86ms/step
Predicted: chihuahua, Actual: chihuahua
Predicted: 0, Actual:0
Predicted: muffin, Actual: chihuahua
Predicted: 1, Actual:0
Predicted: chihuahua, Actual: chihuahua
Predicted: 0, Actual:0
Predicted: chihuahua, Actual: chihuahua
Predicted: 0, Actual:0
Predicted: muffin, Actual: chihuahua
Predicted: 1, Actual:0
Predicted: muffin, Actual: chihuahua
Predicted: 1, Actual:0
Predicted: chihuahua, Actual: chihuahua
Predicted: 0, Actual:0
Predicted: muffin, Actual: chihuahua
Predicted: 1, Actual:0
Predicted: chihuahua, Actual: chihuahua
Predicted: 0, Actual:0
Predicted: chihuahua, Actual: chihuahua
Predicted: 0, Actual:0
```

Thus in our case Class 0 is Cihuahua and 1 is Muffin

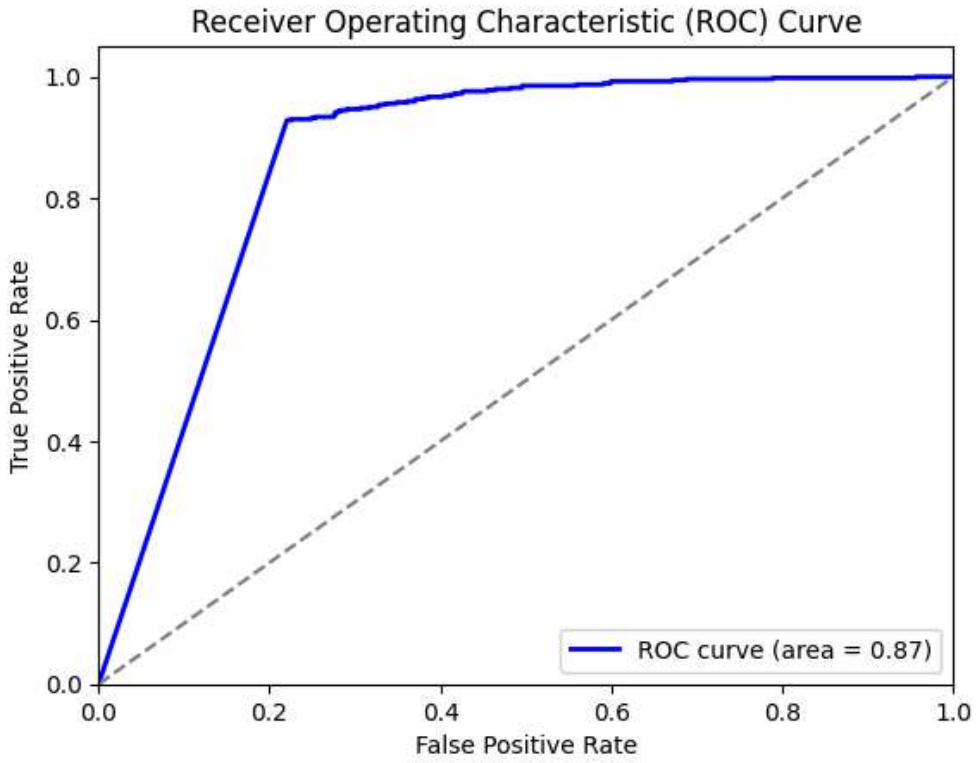
plot confusion matrix and ROC curve

In [61]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import seaborn as sns



cm = confusion_matrix(test_labels, y_pred)

disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=["chihuahua", "muffin"])
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.show()
```



In [62]:
```
from sklearn.metrics import roc_curve, auc

# Calculate ROC curve
fpr, tpr, list_threshold= roc_curve(test_labels, y_pred_probs)
print(fpr, tpr, list_threshold)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

```
[0.        0.2203125 0.225     0.225     0.2484375 0.2484375 0.253125
 0.253125  0.275     0.275     0.2765625 0.2765625 0.2796875 0.2796875
 0.2828125 0.2828125 0.290625  0.290625  0.30625   0.30625   0.3171875
 0.3171875 0.3265625 0.3265625 0.328125  0.328125  0.3359375 0.3359375
 0.3484375 0.3484375 0.3609375 0.3609375 0.3703125 0.3703125 0.3796875
 0.3796875 0.3859375 0.3859375 0.403125  0.403125  0.4125    0.4125
 0.4171875 0.4171875 0.425     0.425     0.4265625 0.4265625 0.4546875
 0.4546875 0.4640625 0.4640625 0.48125   0.48125   0.4953125 0.4953125
 0.559375  0.559375  0.590625  0.590625  0.5984375 0.5984375 0.6
 0.6       0.671875  0.671875  0.6875    0.6875    0.7890625 0.7890625
 0.95625   0.95625   1.       ] [0.         0.92830882 0.92830882 0.93014706 0.93014706 0.93198529
 0.93198529 0.93382353 0.93382353 0.93566176 0.93566176 0.94117647
 0.94117647 0.94301471 0.94301471 0.94485294 0.94485294 0.94669118
 0.94669118 0.94852941 0.94852941 0.95036765 0.95036765 0.95220588
 0.95220588 0.95404412 0.95404412 0.95588235 0.95588235 0.95772059
 0.95772059 0.95955882 0.95955882 0.96323529 0.96323529 0.96507353
 0.96507353 0.96691176 0.96691176 0.96875    0.96875    0.97058824
 0.97058824 0.97242647 0.97242647 0.97426471 0.97426471 0.97610294
 0.97610294 0.97794118 0.97794118 0.97977941 0.97977941 0.98161765
 0.98161765 0.98529412 0.98529412 0.98713235 0.98713235 0.98897059
 0.98897059 0.99080882 0.99080882 0.99264706 0.99264706 0.99448529
 0.99448529 0.99632353 0.99632353 0.99816176 0.99816176 1.
 1.        ] [1.7343204e+00 7.3432040e-01 7.2771347e-01 7.1356606e-01 6.2301606e-01
 6.1558563e-01 6.0080057e-01 5.9484208e-01 5.4018241e-01 5.3137380e-01
 5.3110039e-01 5.1703179e-01 5.0402260e-01 5.0217694e-01 4.9510401e-01
 4.9288541e-01 4.7057965e-01 4.6409452e-01 4.3650413e-01 4.3579152e-01
 4.1315240e-01 4.0717241e-01 3.7793639e-01 3.7703890e-01 3.7106749e-01
 3.7097323e-01 3.4656426e-01 3.4471366e-01 3.2027188e-01 3.1944516e-01
 2.9521218e-01 2.9392335e-01 2.8479016e-01 2.7402425e-01 2.6565316e-01
 2.6431543e-01 2.5596410e-01 2.5534198e-01 2.2379991e-01 2.2330360e-01
 2.0436491e-01 2.0360850e-01 1.9680656e-01 1.9679254e-01 1.7671336e-01
 1.7436109e-01 1.7246734e-01 1.7102411e-01 1.4048387e-01 1.3849849e-01
 1.3015401e-01 1.3001284e-01 1.1505736e-01 1.1476111e-01 1.0649972e-01
 1.0189703e-01 7.1009696e-02 7.0967473e-02 5.5388864e-02 5.4313753e-02
 5.2674770e-02 4.9887486e-02 4.9444538e-02 4.9322881e-02 3.0518809e-02
 3.0186489e-02 2.6847867e-02 2.6637170e-02 1.1234911e-02 1.0721028e-02
 7.8791595e-04 7.8279775e-04 5.7396932e-08]
```



Calcualte best threshold

In [63]:
```
best_threshold_index = np.argmax(tpr - fpr)
best_threshold = list_threshold[best_threshold_index]
print(f"Best threshold: {best_threshold:.2f}")
```

```
Best threshold: 0.73
```

Now that we have got the Best Threshold value as 0.73 and we use this value as threshold and try to plot the confusion matrix again to find the values of TP,FP,TN,FN.

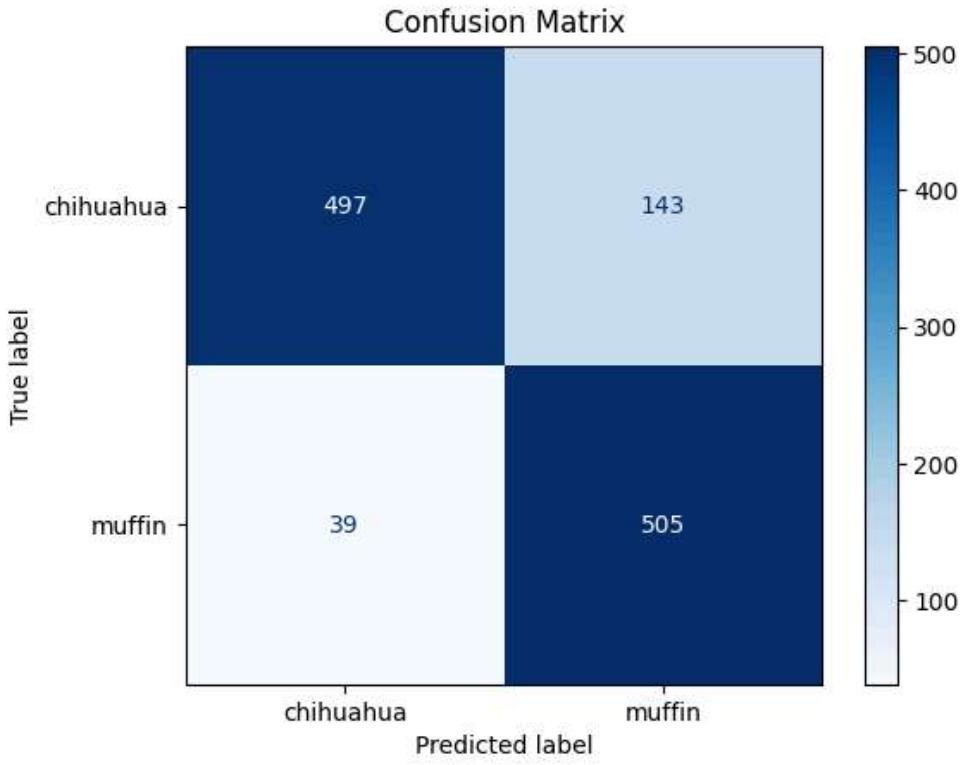
Plot confusion matrix

In [64]:
```
# Predict on test set
y_pred_probs = model.predict(x_test_normalized)
y_pred = (y_pred_probs > 0.73).astype(int)  # Threshold of 0.5


for i in range(10):
```

```
    print(f"Predicted: {test_labels_name[y_pred[i][0]]}, Actual: {test_labels_name[test_labels[i]]}")
    print(f"Predicted: {y_pred[i][0]}, Actual:{ test_labels[i]}")
```

```
37/37 [==============================] - 3s 89ms/step
Predicted: chihuahua, Actual: chihuahua
Predicted: 0, Actual:0
Predicted: muffin, Actual: chihuahua
Predicted: 1, Actual:0
Predicted: chihuahua, Actual: chihuahua
Predicted: 0, Actual:0
Predicted: chihuahua, Actual: chihuahua
Predicted: 0, Actual:0
Predicted: muffin, Actual: chihuahua
Predicted: 1, Actual:0
Predicted: chihuahua, Actual: chihuahua
Predicted: 0, Actual:0
Predicted: chihuahua, Actual: chihuahua
Predicted: 0, Actual:0
Predicted: chihuahua, Actual: chihuahua
Predicted: 0, Actual:0
Predicted: chihuahua, Actual: chihuahua
Predicted: 0, Actual:0
Predicted: chihuahua, Actual: chihuahua
Predicted: 0, Actual:0
```

In [66]:
```
cm = confusion_matrix(test_labels, y_pred)

disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=["chihuahua", "muffin"])
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.show()
```



Performance with the Optimal Threshold

- **Chihuahua Detection**: The model correctly identified **36 more** chihuahuas, an improvement of about **8%.**
- **Muffin Detection**: There was a slight decrease of **8** correctly identified muffins (approximately **1.5%** fewer than before).
- **Overall Impact**: This threshold significantly improves chihuahua detection, with only a minimal trade-off in muffin detection.