In [1]:
```
import tensorflow as tf
from tensorflow.keras import models,layers
import matplotlib.pyplot as plt
```

In [2]:
```
## loading the data directory --
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "..\\Data\\animals\\",
    batch_size=32,
    image_size=(224,224),
    shuffle=True,
    color_mode="rgb"
)
```

```
Found 5400 files belonging to 90 classes.
```

In [3]:
```
## class_names
class_names = dataset.class_names
class_names
```

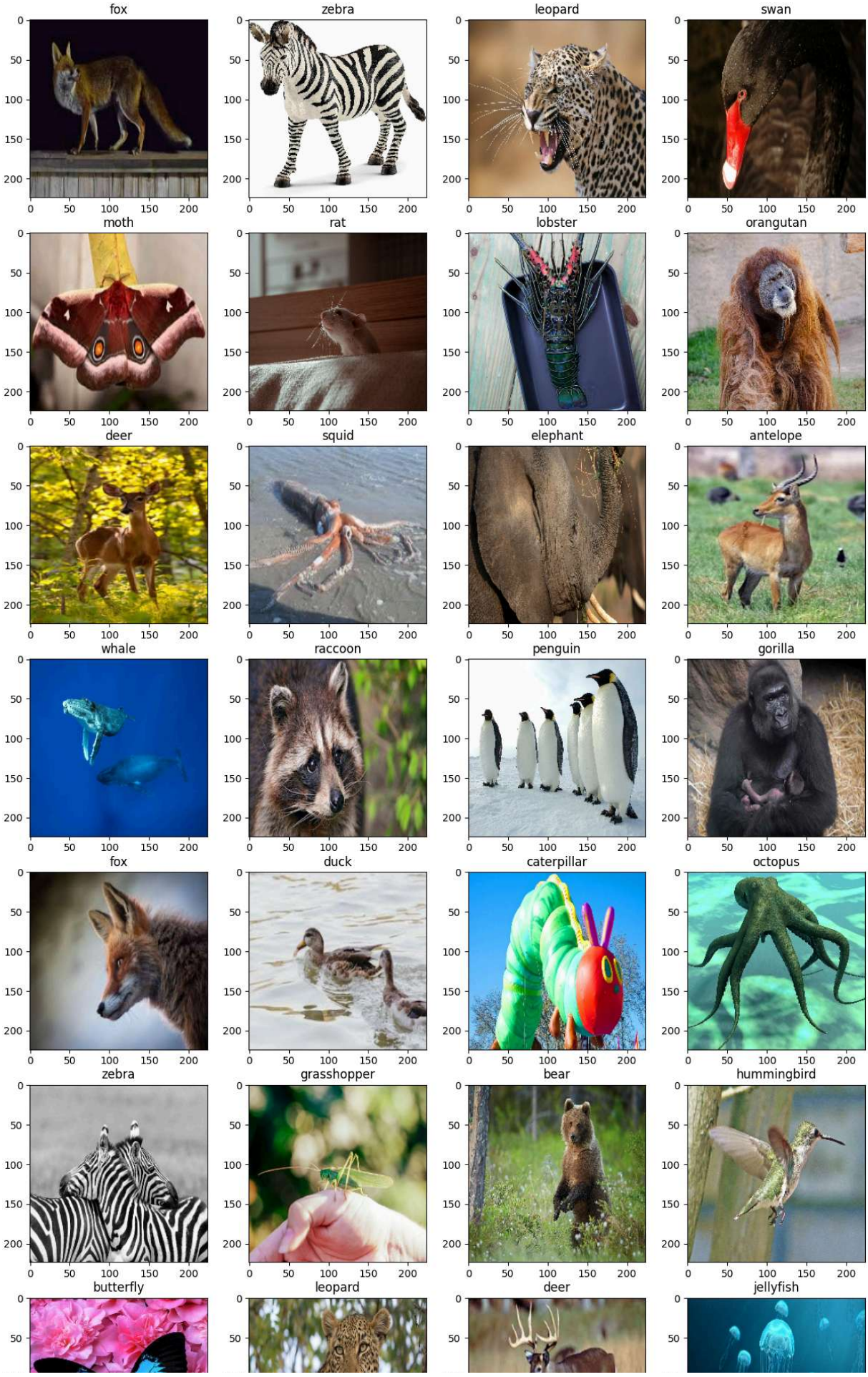
Out[3]:
```
['antelope',
 'badger',
 'bat',
 'bear',
 'bee',
 'beetle',
 'bison',
 'boar',
 'butterfly',
 'cat',
 'caterpillar',
 'chimpanzee',
 'cockroach',
 'cow',
 'coyote',
 'crab',
 'crow',
 'deer',
 'dog',
 'dolphin',
 'donkey',
 'dragonfly',
 'duck',
 'eagle',
 'elephant',
 'flamingo',
 'fly',
 'fox',
 'goat',
 'goldfish',
 'goose',
 'gorilla',
 'grasshopper',
 'hamster',
 'hare',
 'hedgehog',
 'hippopotamus',
 'hornbill',
 'horse',
 'hummingbird',
 'hyena',
 'jellyfish',
 'kangaroo',
 'koala',
 'ladybugs',
 'leopard',
 'lion',
 'lizard',
 'lobster',
 'mosquito',
 'moth',
 'mouse',
 'octopus',
 'okapi',
 'orangutan',
 'otter',
```

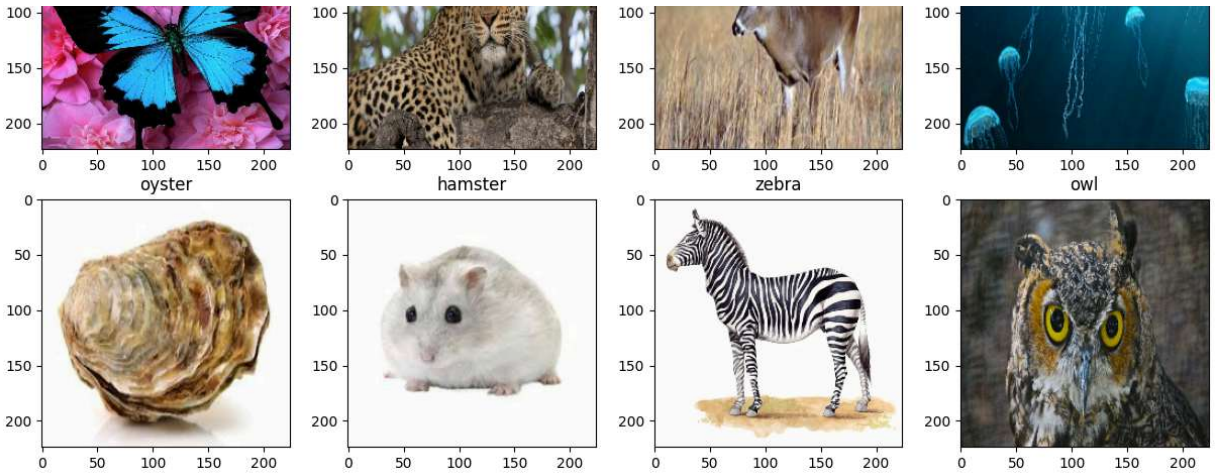
```
 'owl',
 'ox',
 'oyster',
 'panda',
 'parrot',
 'pelecaniformes',
 'penguin',
 'pig',
 'pigeon',
 'porcupine',
 'possum',
 'raccoon',
 'rat',
 'reindeer',
 'rhinoceros',
 'sandpiper',
 'seahorse',
 'seal',
 'shark',
 'sheep',
 'snake',
 'sparrow',
 'squid',
 'squirrel',
 'starfish',
 'swan',
 'tiger',
 'turkey',
 'turtle',
 'whale',
 'wolf',
 'wombat',
 'woodpecker',
 'zebra']
```

In [4]:
```
len(dataset) ## 32 images per batch ==> 169*32=5408
```

Out[4]: 169

In [5]:
```
plt.figure(figsize=(15,30))
for image_batch,label_batch in dataset.take(1):
    for i in range(32):
        plt.subplot(8,4,i+1)
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.title(class_names[label_batch[i]])
```


fox
zebra
leopard
swan
moth
rat
lobster
orangutan
deer
squid
elephant
antelope
whale
raccoon
penguin
gorilla
fox
duck
caterpillar
octopus
zebra
grasshopper
bear
hummingbird
butterfly
leopard
deer
jellyfish



Splitting the Dataset into Train(80%),Test(10%) and Validation(10%)

In [6]:
```
def get_dataset_partitions(ds,train_split=0.8,val_split=0.1,test_split=0.1,shuffle=

    ds_size = len(dataset)

    if shuffle:
        ds = ds.shuffle(shuffle_size,seed=12)
    train_size = int(train_split*ds_size)
    val_size = int(val_split*ds_size)

    train_ds = ds.take(train_size)
    validation_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)

    return train_ds,validation_ds,test_ds
```

In [7]:
```
train_ds,val_ds,test_ds = get_dataset_partitions(dataset)
```

In [8]:
```
len(train_ds),len(test_ds),len(val_ds)
```

Out[8]: (135, 18, 16)

Resizing and Rescaling

In [9]:
```
from tensorflow.keras.applications.vgg16 import preprocess_input
```

In [10]:
```
## Applying VGG16 preprocessing

def process(image,label):
    image = tf.cast(image,tf.float32)
    image = preprocess_input(image)
    return image,label
```

In [11]:
```
## Applying process function to the dataset
train_ds = train_ds.map(process).cache().shuffle(1000).prefetch(buffer_size=tf.data
val_ds = val_ds.map(process).cache().prefetch(buffer_size=tf.data.AUTOTUNE)
test_ds = test_ds.map(process).cache().prefetch(buffer_size=tf.data.AUTOTUNE)
```

In [12]:
```
resize_and_rescale = tf.keras.Sequential([
    layers.Resizing(224,224),
])
```

In [13]:
```
data_augmentation = tf.keras.Sequential([
    layers.RandomFlip("horizontal_and_vertical"),
    layers.RandomRotation(0.2),
    layers.RandomZoom(height_factor=(0.2, 0.3)),
    layers.RandomZoom(height_factor=(-0.3,-0.2)),
])
```

Transfer Learing (Pretrained model)

In [14]:
```
## Loading VGG16 model

conv_base = tf.keras.applications.VGG16(
    include_top=False,
    weights="imagenet",
    input_shape=(224,224,3)
)
```

Fine Tuning

In [15]:
```
## Freezing the Convolution layers and Training the Dense Layers and Some Convoluti
conv_base.trainable = True

set_trainable = False

for layer in conv_base.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else: layer.trainable = False
for layer in conv_base.layers:
    print(layer.name,layer.trainable)
```

```
input_layer False
block1_conv1 False
block1_conv2 False
block1_pool False
block2_conv1 False
block2_conv2 False
block2_pool False
block3_conv1 False
block3_conv2 False
block3_conv3 False
block3_pool False
block4_conv1 False
block4_conv2 False
block4_conv3 False
block4_pool False
block5_conv1 True
block5_conv2 True
block5_conv3 True
block5_pool True
```

In [16]:
```
model = models.Sequential([
    resize_and_rescale,
    data_augmentation,
    conv_base,
    layers.GlobalAveragePooling2D(),
    layers.Dense(256,activation='relu'), ## Relu activation function
    layers.Dropout(0.3),  ## Using Dropout to avoid overfitting
    layers.Dense(len(class_names),activation='softmax') ## As the dataset is a mult
])
```

In [17]:
```
model.build(input_shape=(None, 224, 224, 3)) #force building the model
```

In [18]:
```
model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	
sequential (Sequential)	(None, 224, 224, 3)	
sequential_1 (Sequential)	(None, 224, 224, 3)	
vgg16 (Functional)	(None, 7, 7, 512)	1
global_average_pooling2d (GlobalAveragePooling2D)	(None, 512)	
dense (Dense)	(None, 256)	
dropout (Dropout)	(None, 256)	
dense_1 (Dense)	(None, 90)	

Total params: 14,869,146 (56.72 MB)

Trainable params: 7,233,882 (27.60 MB)

Non-trainable params: 7,635,264 (29.13 MB)

In [19]:
```
model.compile(
    optimizer=tf.optimizers.RMSprop(learning_rate=1e-4),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
## compiling the model at 0.0001 learning rate
```

In [20]:
```
from tensorflow.keras.callbacks import EarlyStopping
```

In [21]:
```
early_stopping_callbacks = EarlyStopping(monitor='val_loss',patience=10,restore_bes
```

In [22]:
```
history = model.fit(train_ds,validation_data=val_ds,epochs=25,callbacks=[early_stop
```

```
Epoch 1/25
135/135 ━━━━━━━━━━━━━━━━━━━━ 408s 3s/step - accuracy: 0.0232 - loss: 4.5402 - val_ac
curacy: 0.0273 - val_loss: 4.4064
Epoch 2/25
135/135 ━━━━━━━━━━━━━━━━━━━━ 388s 3s/step - accuracy: 0.0427 - loss: 4.3200 - val_ac
curacy: 0.0781 - val_loss: 4.0476
Epoch 3/25
135/135 ━━━━━━━━━━━━━━━━━━━━ 384s 3s/step - accuracy: 0.1078 - loss: 4.0219 - val_ac
curacy: 0.1816 - val_loss: 3.5956
Epoch 4/25
135/135 ━━━━━━━━━━━━━━━━━━━━ 386s 3s/step - accuracy: 0.1695 - loss: 3.6665 - val_ac
curacy: 0.2520 - val_loss: 3.2220
Epoch 5/25
135/135 ━━━━━━━━━━━━━━━━━━━━ 385s 3s/step - accuracy: 0.2303 - loss: 3.3142 - val_ac
curacy: 0.3242 - val_loss: 2.7943
Epoch 6/25
135/135 ━━━━━━━━━━━━━━━━━━━━ 378s 3s/step - accuracy: 0.3008 - loss: 2.9517 - val_ac
curacy: 0.4121 - val_loss: 2.3398
Epoch 7/25
135/135 ━━━━━━━━━━━━━━━━━━━━ 395s 3s/step - accuracy: 0.3525 - loss: 2.6145 - val_ac
curacy: 0.5391 - val_loss: 1.9623
Epoch 8/25
135/135 ━━━━━━━━━━━━━━━━━━━━ 364s 3s/step - accuracy: 0.4265 - loss: 2.2971 - val_ac
curacy: 0.5723 - val_loss: 1.7092
Epoch 9/25
135/135 ━━━━━━━━━━━━━━━━━━━━ 359s 3s/step - accuracy: 0.4879 - loss: 2.0093 - val_ac
curacy: 0.6738 - val_loss: 1.3312
Epoch 10/25
135/135 ━━━━━━━━━━━━━━━━━━━━ 357s 3s/step - accuracy: 0.5397 - loss: 1.8096 - val_ac
curacy: 0.6738 - val_loss: 1.3774
Epoch 11/25
135/135 ━━━━━━━━━━━━━━━━━━━━ 357s 3s/step - accuracy: 0.5795 - loss: 1.6112 - val_ac
curacy: 0.7070 - val_loss: 1.0894
Epoch 12/25
135/135 ━━━━━━━━━━━━━━━━━━━━ 357s 3s/step - accuracy: 0.6125 - loss: 1.4664 - val_ac
curacy: 0.7598 - val_loss: 0.9862
Epoch 13/25
135/135 ━━━━━━━━━━━━━━━━━━━━ 357s 3s/step - accuracy: 0.6473 - loss: 1.3269 - val_ac
curacy: 0.7305 - val_loss: 1.0365
Epoch 14/25
135/135 ━━━━━━━━━━━━━━━━━━━━ 355s 3s/step - accuracy: 0.6737 - loss: 1.1944 - val_ac
curacy: 0.7422 - val_loss: 1.0222
Epoch 15/25
135/135 ━━━━━━━━━━━━━━━━━━━━ 356s 3s/step - accuracy: 0.7027 - loss: 1.0919 - val_ac
curacy: 0.7910 - val_loss: 0.8672
Epoch 16/25
135/135 ━━━━━━━━━━━━━━━━━━━━ 356s 3s/step - accuracy: 0.7210 - loss: 1.0268 - val_ac
curacy: 0.8105 - val_loss: 0.8362
Epoch 17/25
135/135 ━━━━━━━━━━━━━━━━━━━━ 356s 3s/step - accuracy: 0.7393 - loss: 0.9682 - val_ac
curacy: 0.8105 - val_loss: 0.9132
Epoch 18/25
135/135 ━━━━━━━━━━━━━━━━━━━━ 357s 3s/step - accuracy: 0.7491 - loss: 0.9067 - val_ac
curacy: 0.8340 - val_loss: 0.7585
Epoch 19/25
135/135 ━━━━━━━━━━━━━━━━━━━━ 353s 3s/step - accuracy: 0.7658 - loss: 0.8738 - val_ac
```

```
curacy: 0.8281 - val_loss: 0.6951
Epoch 20/25
135/135 ━━━━━━━━━━━━━━━━━━━━ 356s 3s/step - accuracy: 0.7806 - loss: 0.7809 - val_ac
curacy: 0.8086 - val_loss: 0.8104
Epoch 21/25
135/135 ━━━━━━━━━━━━━━━━━━━━ 354s 3s/step - accuracy: 0.7910 - loss: 0.7476 - val_ac
curacy: 0.8203 - val_loss: 0.9920
Epoch 22/25
135/135 ━━━━━━━━━━━━━━━━━━━━ 356s 3s/step - accuracy: 0.8087 - loss: 0.7026 - val_ac
curacy: 0.8262 - val_loss: 0.9028
Epoch 23/25
135/135 ━━━━━━━━━━━━━━━━━━━━ 381s 3s/step - accuracy: 0.8170 - loss: 0.6678 - val_ac
curacy: 0.8027 - val_loss: 0.8082
Epoch 24/25
135/135 ━━━━━━━━━━━━━━━━━━━━ 365s 3s/step - accuracy: 0.8237 - loss: 0.6542 - val_ac
curacy: 0.8594 - val_loss: 0.7876
Epoch 25/25
135/135 ━━━━━━━━━━━━━━━━━━━━ 362s 3s/step - accuracy: 0.8356 - loss: 0.5752 - val_ac
curacy: 0.8496 - val_loss: 0.8100
```

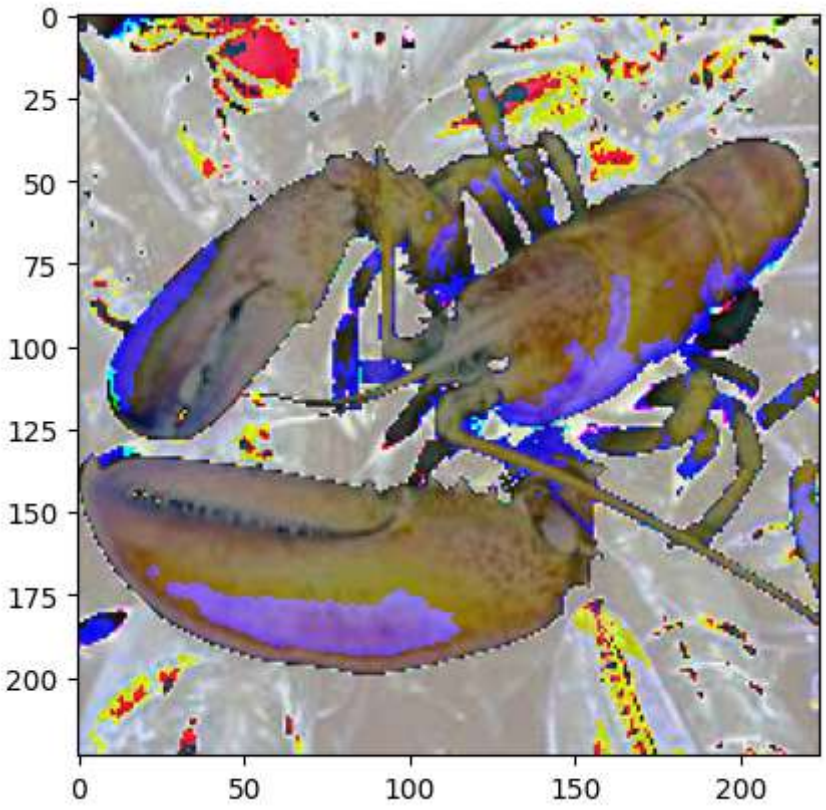
In [23]:
```
scores = model.evaluate(test_ds) ## Evaluating the model on test_data
```

```
18/18 ━━━━━━━━━━━━━━━━━━━━ 46s 2s/step - accuracy: 0.8559 - loss: 0.6677
```

Predictions on Test Data

In [31]:
```
for image_batch,labels_batch in test_ds.take(1):
    first_image = image_batch[0].numpy().astype('uint8')
    first_label = labels_batch[0].numpy()

    print("first image prediction : ")
    plt.imshow(first_image)
    print("actual label:",class_names[first_label])

    batch_prediction = model.predict(image_batch)
    print("predicted label:",class_names[np.argmax(batch_prediction[0])])
```

```
first image prediction :
actual label: lobster
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
predicted label: lobster
```



In [26]:
```
def predict(model,img):
    img_array = tf.keras.preprocessing.image.img_to_array(images[i].numpy())
    img_array = tf.expand_dims(img_array,0)

    predicions = model.predict(img_array)

    predicted_class = class_names[np.argmax(predicions[0])]
    confidence = round(100*(np.max(predicions[0])),2)
    return predicted_class,confidence
```

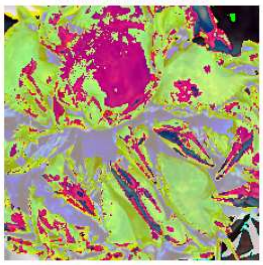
In [27]:
```
plt.figure(figsize=(15,35))
for images,labels in test_ds.take(1):
    for i in range(32):
        plt.subplot(8,4,i+1)
        plt.imshow(images[i].numpy().astype("uint8"))
        predicted_class,confidence = predict(model,images[i].numpy())
        actual_class = class_names[labels[i]]

        plt.title(f"Actual: {actual_class},\n Predicted: {predicted_class},\nConfid
        plt.axis("off")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 712ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 211ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 282ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 330ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 276ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 223ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 236ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 213ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 258ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 197ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 291ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 232ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 196ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 211ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 226ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 239ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 216ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 191ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 172ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 215ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 166ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 280ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 187ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 242ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 169ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 246ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 157ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 218ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 156ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 226ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 161ms/step
```

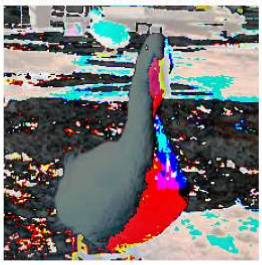
Actual: lobster,
Predicted: lobster,
Confidence: 99.97000122070312

Actual: flamingo,
Predicted: flamingo,
Confidence: 100.0

Actual: crab,
Predicted: crab,
Confidence: 100.0



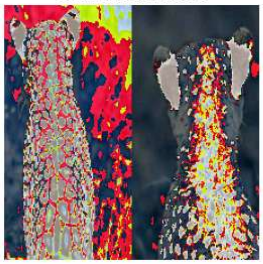
Actual: duck,
Predicted: duck,
Confidence: 99.98999786376953



Actual: ladybugs,
Predicted: ladybugs,
Confidence: 98.98999786376953

Actual: rhinoceros,
Predicted: hippopotamus,
Confidence: 69.20999908447266

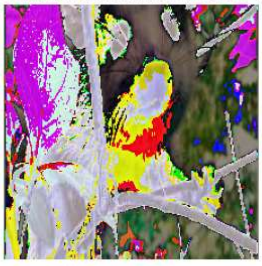
Actual: leopard,
Predicted: leopard,
Confidence: 100.0



Actual: horse,
Predicted: horse,
Confidence: 87.9800033569336

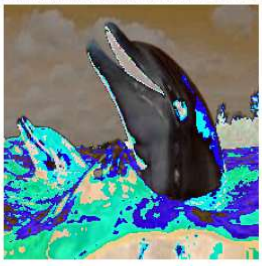
Actual: pelecaniformes,
Predicted: pelecaniformes,
Confidence: 100.0

Actual: possum,
Predicted: possum,
Confidence: 99.01000213623047

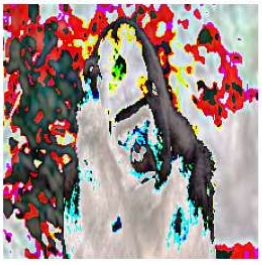


Actual: fox,
Predicted: fox,
Confidence: 100.0

Actual: dolphin,
Predicted: dolphin,
Confidence: 93.68000030517578

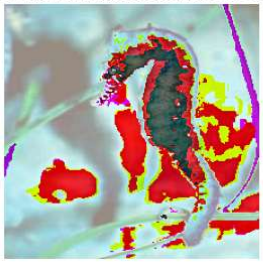


Actual: gorilla,
Predicted: gorilla,
Confidence: 90.44999694824219

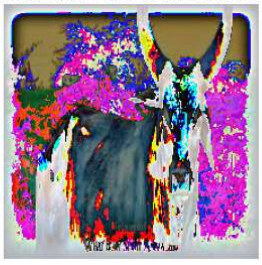


Actual: beetle,
Predicted: beetle,
Confidence: 99.98999786376953

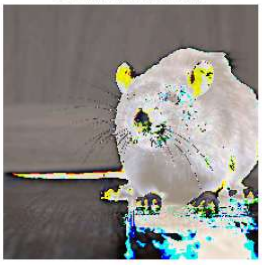
Actual: seahorse,
Predicted: seahorse,
Confidence: 91.30999755859375



Actual: ox,
Predicted: ox,
Confidence: 28.229999542236328



Actual: rat,
Predicted: rat,
Confidence: 100.0



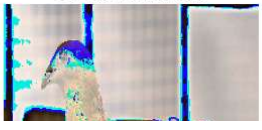
Actual: seal,
Predicted: seal,
Confidence: 86.48999786376953

Actual: chimpanzee,
Predicted: chimpanzee,
Confidence: 100.0

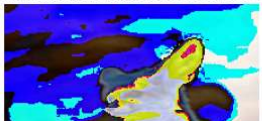
Actual: cat,
Predicted: cat,
Confidence: 99.55000305175781

Actual: lizard,
Predicted: lizard,
Confidence: 91.45999908447266

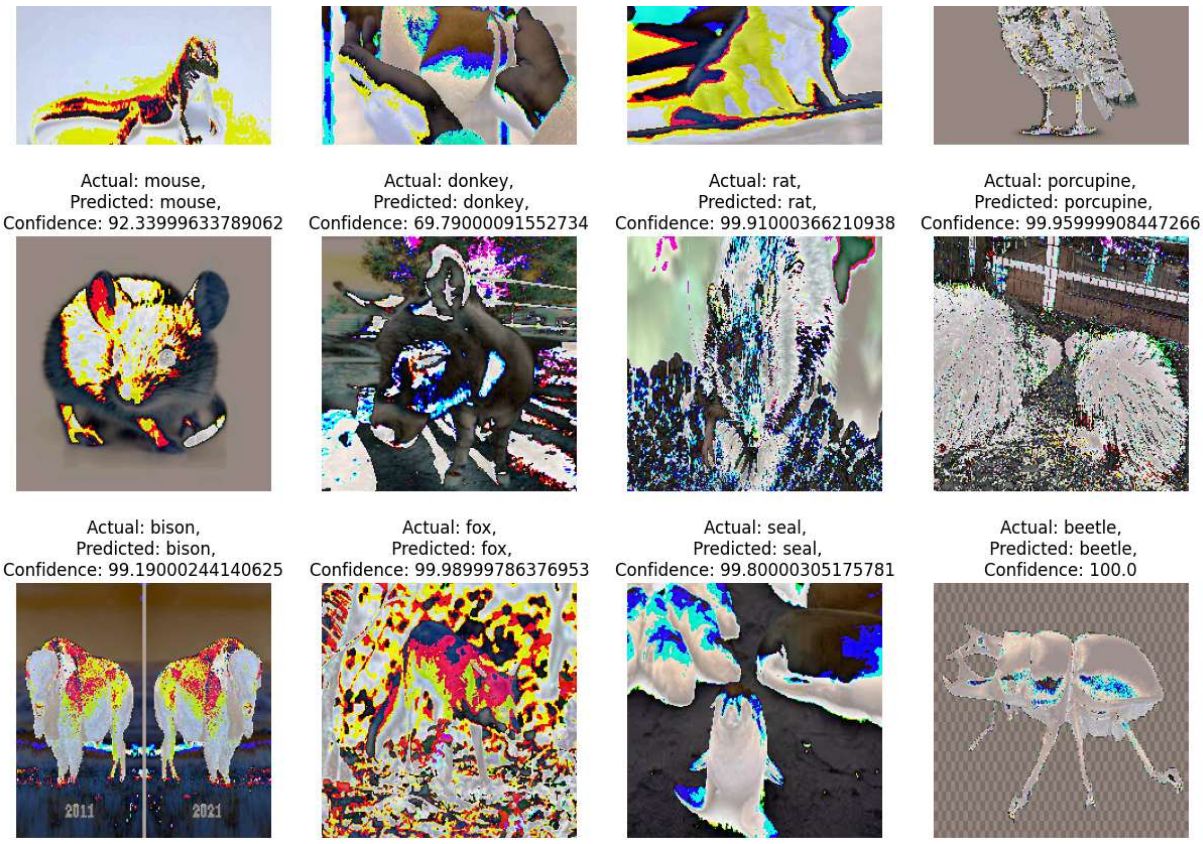
Actual: pigeon,
Predicted: pigeon,
Confidence: 100.0



Actual: fox,
Predicted: pig,
Confidence: 83.01000213623047



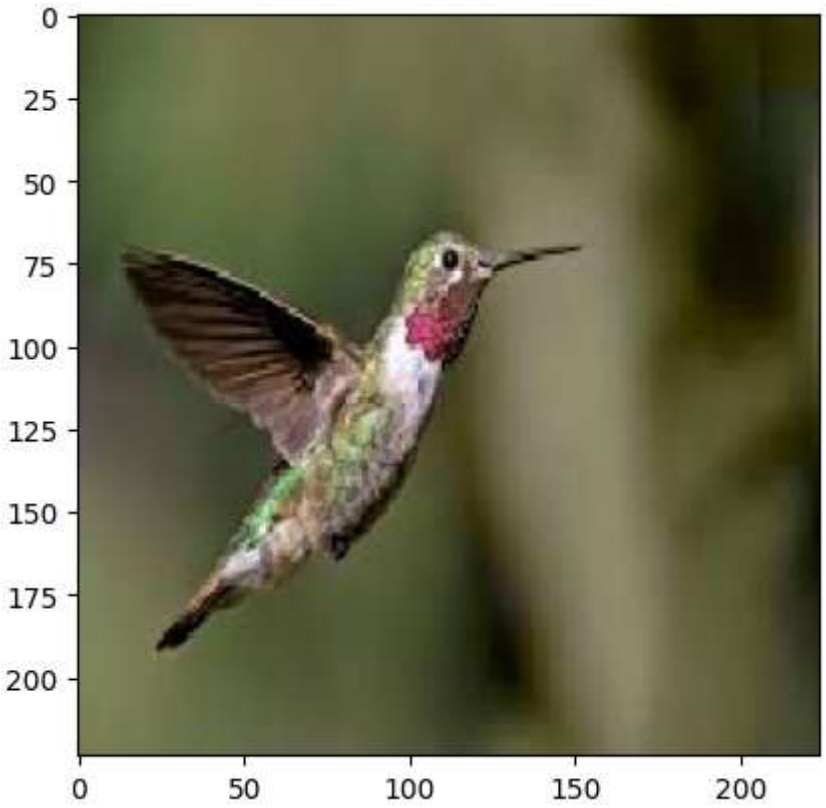
Actual: turkey,
Predicted: turkey,
Confidence: 100.0



Predicting on single image

In [28]:
```
from tensorflow.keras.preprocessing import image
import numpy as np

img_path = "D:/Artificial Intelligence/Deep Learning/Project2/Animal_Classifier/Dat

# Load and preprocess image
img = image.load_img(img_path, target_size=(224,224))
x = image.img_to_array(img)
plt.imshow(x.astype("uint8"))
x = np.expand_dims(x, axis=0)

# Applying VGG16 preprocessing
from tensorflow.keras.applications.vgg16 import preprocess_input
x = preprocess_input(x)

# Prediction
preds = model.predict(x)
pred_class = np.argmax(preds[0])
print("Predicted class:", class_names[pred_class])
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 231ms/step
Predicted class: hummingbird
```



In [29]:
```
# ## saving the model
# model_version = 2
# model.save(f"../models/classifier_{model_version}_acc_85.h5")
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: