This notebook is inspired by the famous cartoon "Phineas and Ferb" and how Dr. Heinz Doofenshmirtz fails to identify Perry the Platypus (his arch nemesis) until he wears his hat. This model is called the "Distinguishinator" to simulate what Dr. Doofenshmirtz must be thinking XD

Import libraries

In [1]:
```
import numpy as np

import tensorflow as tf
import tensorflow.keras.applications.mobilenet
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import layers

import keras
from keras.utils import np_utils
from keras.layers import Activation, Dropout, Convolution2D, GlobalAveragePooling2D
from keras.models import Sequential

import os

import matplotlib.pyplot as plt

import random
```

In [2]:
```
IMG_SAVE_PATH = r'/kaggle/input/platypus-or-perry-the-platypus/train'

#Str_to_Int is to one-hot encode the labels
Str_to_Int = {
    'perry the platypus':0,
    'platypus':1
}

NUM_CLASSES = 2

def str_to_Int_mapper(val):
    return Str_to_Int[val]
```

Converting image dataset to 3D arrays

In [3]:
```
import PIL
import cv2

dataset = []
for directory in os.listdir(IMG_SAVE_PATH):
    path = os.path.join(IMG_SAVE_PATH, directory)
    for image in os.listdir(path):
        new_path = os.path.join(path, image)
        try:
            imgpath=PIL.Image.open(new_path)
            imgpath=imgpath.convert('RGB')
            img = np.asarray(imgpath)
            img = cv2.resize(img, (240,240))
```

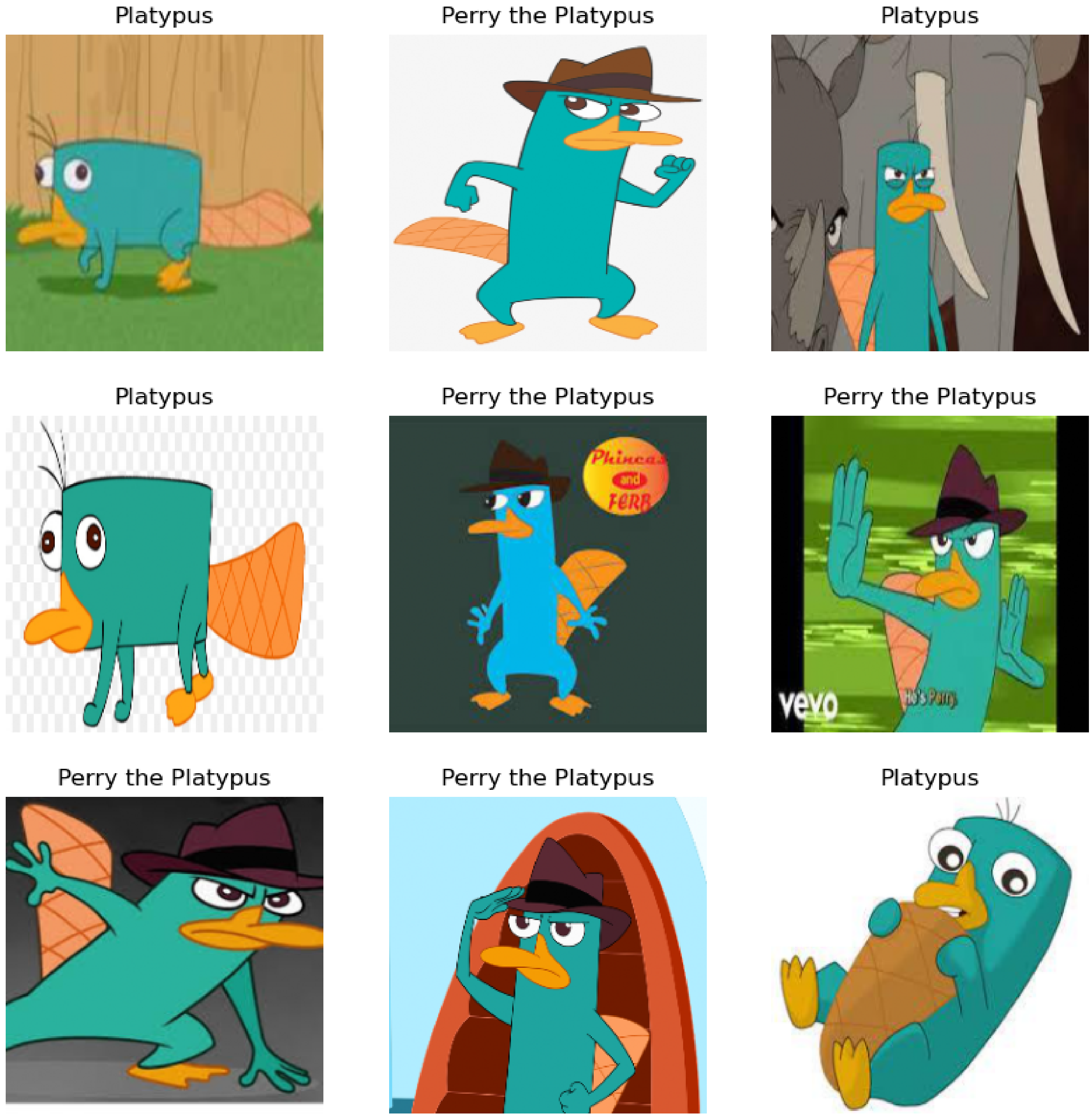
```
            img=img/255.
            dataset.append([img, directory])
        except PIL.UnidentifiedImageError:
            print("Skiping image")
data, labels = zip(*dataset)
temp = list(map(str_to_Int_mapper, labels))
labels = keras.utils.to_categorical(temp)
```

```
Skiping image
```

Plotting train dataset

In [4]:

```
# Select num_images random images from the testing_data array
indices = random.sample(range(len(data)), 9)
images = np.array(data)[indices]
label = np.array(labels)[indices]

# Create a grid of subplots with the specified dimensions
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(10, 10))

# Plot each image with its corresponding label as the title
for i, ax in enumerate(axes.flat):
    ax.imshow(images[i])
    if label[i][0]==1:
        ax.set_title("Perry the Platypus")
    else:
        ax.set_title("Platypus")
    ax.axis('off')

# Display the plot
plt.show()
```



Data Augmentation

In [5]:

```
data_augmentation = tf.keras.Sequential([
  layers.RandomFlip("horizontal_and_vertical"),
  layers.RandomRotation(0.2),
])
```

Example of augmentation

In [6]:

```
image = tf.cast(tf.expand_dims(data,0), tf.float32)
image = tf.squeeze(image, axis=0)
```

In [7]:

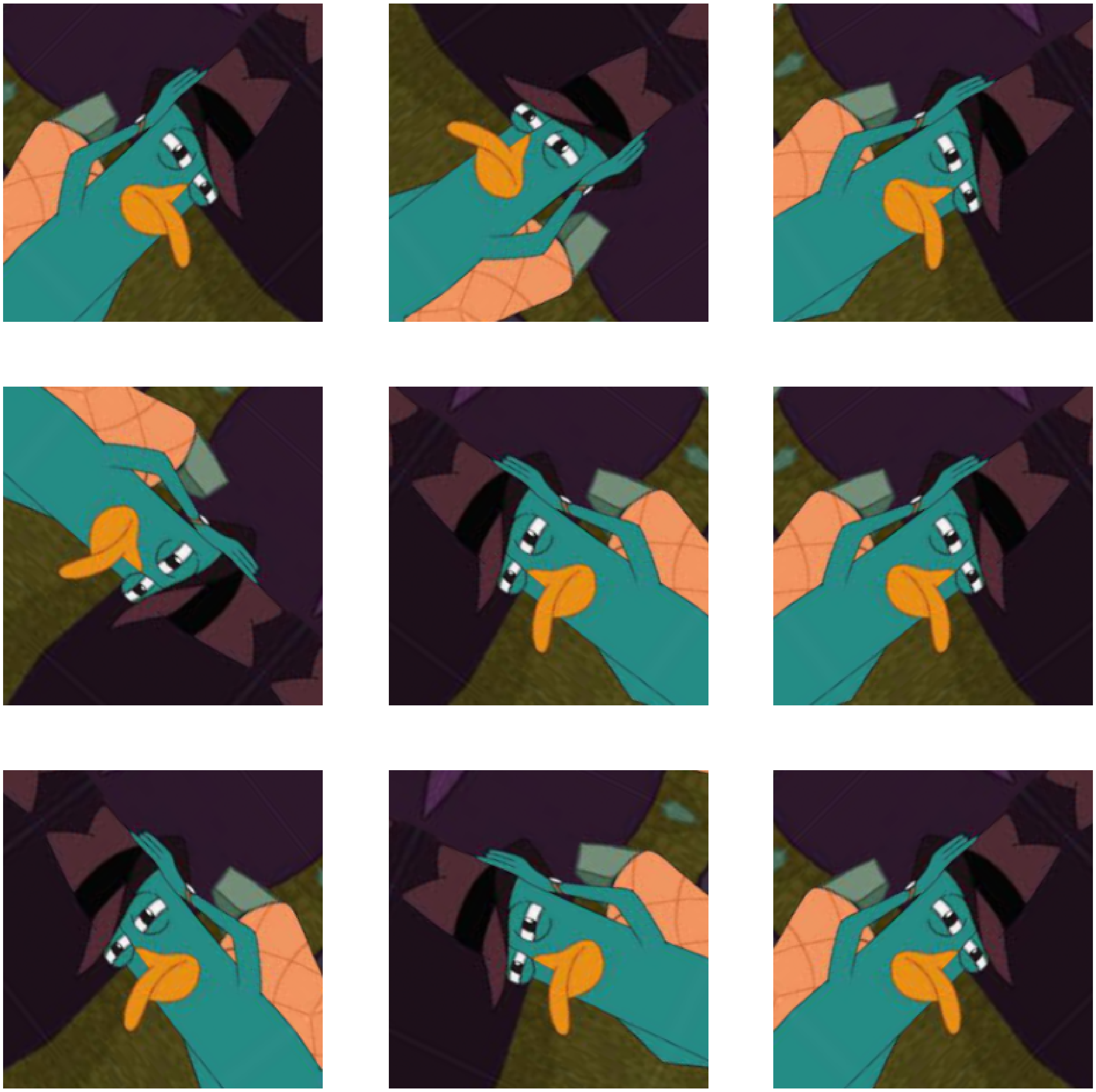
```
data_augmentation(image[1])
```

Out[7]:

```
<tf.Tensor: shape=(240, 240, 3), dtype=float32, numpy=
array([[[0.43137255, 0.49803922, 0.42745098],
        [0.43137258, 0.49803925, 0.427451  ],
        [0.43137255, 0.49803922, 0.42745095],
        ...,
        [0.43137255, 0.49803925, 0.42745098],
        [0.43137255, 0.4980392 , 0.427451  ],
        [0.43137258, 0.49803922, 0.42745098]],

       [[0.43137258, 0.4980392 , 0.42745095],
        [0.43137255, 0.49803925, 0.42745098],
        [0.43137258, 0.49803925, 0.42745098],
        ...,
        [0.43137258, 0.49803925, 0.42745098],
        [0.43137252, 0.49803925, 0.42745098],
        [0.43137255, 0.4980392 , 0.427451  ]],

       [[0.43137255, 0.49803925, 0.42745098],
        [0.43137252, 0.49803922, 0.42745098],
        [0.43137258, 0.4980392 , 0.42745095],
        ...,
        [0.43137252, 0.49803925, 0.42745095],
        [0.43137255, 0.49803922, 0.42745095],
        [0.43137252, 0.49803925, 0.42745095]],

       ...,

       [[0.42594498, 0.49315917, 0.42257094],
        [0.42791182, 0.49457848, 0.42399025],
        [0.42745095, 0.49411768, 0.4235294 ],
        ...,
        [0.42972696, 0.49639362, 0.4258054 ],
        [0.42745098, 0.49411762, 0.42352942],
        [0.42745098, 0.49411762, 0.42596447]],

       [[0.4333166 , 0.5011313 , 0.43054307],
        [0.43178308, 0.49844974, 0.42786154],
        [0.42745098, 0.49411768, 0.42352942],
        ...,
        [0.42987692, 0.4965436 , 0.4259554 ],
        [0.427451  , 0.49411762, 0.42352942],
        [0.427451  , 0.49411762, 0.42650914]],

       [[0.4232806 , 0.4892996 , 0.4187113 ],
        [0.427451  , 0.49411765, 0.42352942],
        [0.42745098, 0.49411762, 0.42352942],
        ...,
        [0.43057448, 0.49724114, 0.42665297],
        [0.42919677, 0.49586344, 0.4252752 ],
        [0.4293468 , 0.49601343, 0.42680946]]], dtype=float32)>
```

In [8]:

```
plt.figure(figsize=(10, 10))
for i in range(9):
    augmented_image = data_augmentation(image)
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_image[0])
    plt.axis("off")
```



Model Training

In [9]:

```
from keras.applications import DenseNet121
#from keras.callbacks import Callback, ModelCheckpoint
from tensorflow.keras import layers
from keras.layers import Dense, Flatten

densenet = DenseNet121(
    weights='imagenet',
    include_top=False,
    input_shape=(240,240,3)
)

def build_densenet():
    model = Sequential()
    model.add(densenet)
    model.add(data_augmentation)
    model.add(Dense(64, activation='relu'))
```

```
    model.add(Dense(32, activation='relu'))
    model.add(Flatten())
    model.add(Dense(16, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Flatten())
    #model.add(layers.GlobalAveragePooling2D())
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(2, activation='sigmoid'))

    model.compile(
        loss='binary_crossentropy',
        optimizer=Adam(learning_rate=0.0001),
        metrics=['accuracy']
    )

    return model
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/dense
net/densenet121_weights_tf_dim_ordering_tf_kernels_notop.h5
29084464/29084464 [==============================] - 2s 0us/step
```

In [10]:

```
model = build_densenet()
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 densenet121 (Functional)    (None, 7, 7, 1024)        7037504   
                                                                 
 sequential (Sequential)     (240, 240, 3)             0         
                                                                 
 dense (Dense)               (None, 7, 7, 64)          65600     
                                                                 
 dense_1 (Dense)             (None, 7, 7, 32)          2080      
                                                                 
 flatten (Flatten)           (None, 1568)              0         
                                                                 
 dense_2 (Dense)             (None, 16)                25104     
                                                                 
 dense_3 (Dense)             (None, 8)                 136       
                                                                 
 flatten_1 (Flatten)         (None, 8)                 0         
                                                                 
 dropout (Dropout)           (None, 8)                 0         
                                                                 
 dense_4 (Dense)             (None, 2)                 18        
                                                                 
=================================================================
Total params: 7,130,442
Trainable params: 7,046,794
Non-trainable params: 83,648
_________________________________________________________________
```

In [11]:

```
history=model.fit(np.array(data), np.array(labels), epochs = 50, shuffle = True, valida
```

```
Epoch 1/50
2/2 [==============================] - 65s 6s/step - loss: 0.6685 - accuracy: 0.6486 - v
```

```
al_loss: 0.8655 - val_accuracy: 0.2000
Epoch 2/50
2/2 [==============================] - 19s 3s/step - loss: 0.7620 - accuracy: 0.5405 - v
al_loss: 0.8595 - val_accuracy: 0.2000
Epoch 3/50
2/2 [==============================] - 19s 3s/step - loss: 0.5538 - accuracy: 0.6757 - v
al_loss: 0.8868 - val_accuracy: 0.2000
Epoch 4/50
2/2 [==============================] - 19s 4s/step - loss: 0.5310 - accuracy: 0.7568 - v
al_loss: 0.9060 - val_accuracy: 0.2000
Epoch 5/50
2/2 [==============================] - 18s 3s/step - loss: 0.5556 - accuracy: 0.7297 - v
al_loss: 0.9110 - val_accuracy: 0.1000
Epoch 6/50
2/2 [==============================] - 19s 3s/step - loss: 0.4505 - accuracy: 0.8378 - v
al_loss: 0.9194 - val_accuracy: 0.1000
Epoch 7/50
2/2 [==============================] - 19s 4s/step - loss: 0.5250 - accuracy: 0.7297 - v
al_loss: 0.9348 - val_accuracy: 0.0000e+00
Epoch 8/50
2/2 [==============================] - 19s 4s/step - loss: 0.5878 - accuracy: 0.6757 - v
al_loss: 0.9429 - val_accuracy: 0.0000e+00
Epoch 9/50
2/2 [==============================] - 19s 3s/step - loss: 0.4753 - accuracy: 0.7838 - v
al_loss: 0.9588 - val_accuracy: 0.0000e+00
Epoch 10/50
2/2 [==============================] - 18s 3s/step - loss: 0.4938 - accuracy: 0.8108 - v
al_loss: 0.9753 - val_accuracy: 0.0000e+00
Epoch 11/50
2/2 [==============================] - 19s 4s/step - loss: 0.4745 - accuracy: 0.8649 - v
al_loss: 0.9821 - val_accuracy: 0.0000e+00
Epoch 12/50
2/2 [==============================] - 19s 4s/step - loss: 0.5011 - accuracy: 0.7838 - v
al_loss: 0.9985 - val_accuracy: 0.1000
Epoch 13/50
2/2 [==============================] - 18s 3s/step - loss: 0.3637 - accuracy: 0.8378 - v
al_loss: 1.0193 - val_accuracy: 0.1000
Epoch 14/50
2/2 [==============================] - 19s 4s/step - loss: 0.4591 - accuracy: 0.7568 - v
al_loss: 1.0331 - val_accuracy: 0.2000
Epoch 15/50
2/2 [==============================] - 19s 4s/step - loss: 0.3809 - accuracy: 0.8649 - v
al_loss: 1.0531 - val_accuracy: 0.2000
Epoch 16/50
2/2 [==============================] - 19s 4s/step - loss: 0.3045 - accuracy: 0.8919 - v
al_loss: 1.0928 - val_accuracy: 0.2000
Epoch 17/50
2/2 [==============================] - 19s 4s/step - loss: 0.4259 - accuracy: 0.8649 - v
al_loss: 1.1440 - val_accuracy: 0.2000
Epoch 18/50
2/2 [==============================] - 19s 4s/step - loss: 0.3266 - accuracy: 0.8378 - v
al_loss: 1.2025 - val_accuracy: 0.1000
Epoch 19/50
2/2 [==============================] - 19s 4s/step - loss: 0.3014 - accuracy: 0.8649 - v
al_loss: 1.2640 - val_accuracy: 0.1000
Epoch 20/50
2/2 [==============================] - 19s 4s/step - loss: 0.3321 - accuracy: 0.8649 - v
al_loss: 1.3385 - val_accuracy: 0.1000
Epoch 21/50
2/2 [==============================] - 19s 4s/step - loss: 0.2926 - accuracy: 0.8919 - v
```

```
al_loss: 1.3961 - val_accuracy: 0.1000
Epoch 22/50
2/2 [==============================] - 18s 3s/step - loss: 0.3376 - accuracy: 0.8378 - v
al_loss: 1.4194 - val_accuracy: 0.1000
Epoch 23/50
2/2 [==============================] - 18s 3s/step - loss: 0.2407 - accuracy: 0.9189 - v
al_loss: 1.4256 - val_accuracy: 0.1000
Epoch 24/50
2/2 [==============================] - 19s 4s/step - loss: 0.2242 - accuracy: 0.9730 - v
al_loss: 1.4464 - val_accuracy: 0.1000
Epoch 25/50
2/2 [==============================] - 19s 4s/step - loss: 0.3260 - accuracy: 0.8649 - v
al_loss: 1.4679 - val_accuracy: 0.1000
Epoch 26/50
2/2 [==============================] - 19s 3s/step - loss: 0.2007 - accuracy: 0.8919 - v
al_loss: 1.4947 - val_accuracy: 0.1000
Epoch 27/50
2/2 [==============================] - 19s 4s/step - loss: 0.2835 - accuracy: 0.8919 - v
al_loss: 1.4979 - val_accuracy: 0.1000
Epoch 28/50
2/2 [==============================] - 19s 3s/step - loss: 0.3451 - accuracy: 0.8649 - v
al_loss: 1.4888 - val_accuracy: 0.1000
Epoch 29/50
2/2 [==============================] - 18s 4s/step - loss: 0.1915 - accuracy: 0.9459 - v
al_loss: 1.4914 - val_accuracy: 0.1000
Epoch 30/50
2/2 [==============================] - 18s 3s/step - loss: 0.2464 - accuracy: 0.8919 - v
al_loss: 1.4904 - val_accuracy: 0.1000
Epoch 31/50
2/2 [==============================] - 19s 3s/step - loss: 0.2274 - accuracy: 0.8649 - v
al_loss: 1.4808 - val_accuracy: 0.1000
Epoch 32/50
2/2 [==============================] - 19s 4s/step - loss: 0.2303 - accuracy: 0.9189 - v
al_loss: 1.4654 - val_accuracy: 0.1000
Epoch 33/50
2/2 [==============================] - 19s 3s/step - loss: 0.1950 - accuracy: 0.9189 - v
al_loss: 1.4652 - val_accuracy: 0.1000
Epoch 34/50
2/2 [==============================] - 19s 4s/step - loss: 0.1979 - accuracy: 0.8919 - v
al_loss: 1.4690 - val_accuracy: 0.2000
Epoch 35/50
2/2 [==============================] - 18s 3s/step - loss: 0.2121 - accuracy: 0.8919 - v
al_loss: 1.4848 - val_accuracy: 0.1000
Epoch 36/50
2/2 [==============================] - 19s 3s/step - loss: 0.1638 - accuracy: 0.9189 - v
al_loss: 1.4985 - val_accuracy: 0.1000
Epoch 37/50
2/2 [==============================] - 18s 3s/step - loss: 0.2502 - accuracy: 0.8378 - v
al_loss: 1.5175 - val_accuracy: 0.1000
Epoch 38/50
2/2 [==============================] - 19s 3s/step - loss: 0.1789 - accuracy: 0.9189 - v
al_loss: 1.5447 - val_accuracy: 0.2000
Epoch 39/50
2/2 [==============================] - 19s 4s/step - loss: 0.2440 - accuracy: 0.8919 - v
al_loss: 1.5733 - val_accuracy: 0.2000
Epoch 40/50
2/2 [==============================] - 19s 3s/step - loss: 0.1659 - accuracy: 0.9459 - v
al_loss: 1.5966 - val_accuracy: 0.2000
Epoch 41/50
2/2 [==============================] - 19s 4s/step - loss: 0.2209 - accuracy: 0.8919 - v
```

```
al_loss: 1.6175 - val_accuracy: 0.2000
Epoch 42/50
2/2 [==============================] - 19s 4s/step - loss: 0.1880 - accuracy: 0.9189 - v
al_loss: 1.6214 - val_accuracy: 0.3000
Epoch 43/50
2/2 [==============================] - 19s 3s/step - loss: 0.1701 - accuracy: 0.8919 - v
al_loss: 1.6057 - val_accuracy: 0.3000
Epoch 44/50
2/2 [==============================] - 18s 3s/step - loss: 0.0939 - accuracy: 0.9459 - v
al_loss: 1.5822 - val_accuracy: 0.3000
Epoch 45/50
2/2 [==============================] - 19s 4s/step - loss: 0.1314 - accuracy: 0.8919 - v
al_loss: 1.5599 - val_accuracy: 0.3000
Epoch 46/50
2/2 [==============================] - 19s 4s/step - loss: 0.2007 - accuracy: 0.8919 - v
al_loss: 1.5377 - val_accuracy: 0.4000
Epoch 47/50
2/2 [==============================] - 18s 3s/step - loss: 0.1523 - accuracy: 0.9459 - v
al_loss: 1.5240 - val_accuracy: 0.4000
Epoch 48/50
2/2 [==============================] - 19s 4s/step - loss: 0.1601 - accuracy: 0.9459 - v
al_loss: 1.4919 - val_accuracy: 0.4000
Epoch 49/50
2/2 [==============================] - 18s 3s/step - loss: 0.1526 - accuracy: 0.9189 - v
al_loss: 1.4530 - val_accuracy: 0.4000
Epoch 50/50
2/2 [==============================] - 19s 3s/step - loss: 0.1478 - accuracy: 0.9189 - v
al_loss: 1.4278 - val_accuracy: 0.4000
```

In [12]:

```python
import seaborn as sns
from matplotlib import pyplot

def plot_acc(history):
    sns.set()

    fig = pyplot.figure(0, (12, 4))

    ax = pyplot.subplot(1, 2, 1)
    sns.lineplot(x=history.epoch, y=history.history['accuracy'], label='train')
    sns.lineplot(x=history.epoch, y=history.history['val_accuracy'], label='valid')
    pyplot.title('Accuracy')
    pyplot.tight_layout()

    ax = pyplot.subplot(1, 2, 2)
    sns.lineplot(x=history.epoch, y=history.history['loss'], label='train')
    sns.lineplot(x=history.epoch, y=history.history['val_loss'], label='valid')
    pyplot.title('Loss')
    pyplot.tight_layout()

    pyplot.show()
```

In [13]:

```python
plot_acc(history)
```

In [14]:
```
# save the model for later use
model.save("perry.h5")
```

Testing

In [15]:
```
IMG_SAVE_PATH_TESTING = r'/kaggle/input/platypus-or-perry-the-platypus/test'
```

In [16]:
```
dataset_testing = []
for directory in os.listdir(IMG_SAVE_PATH_TESTING):
    path = os.path.join(IMG_SAVE_PATH_TESTING, directory)
    for image in os.listdir(path):
        new_path = os.path.join(path, image)
        imgpath=PIL.Image.open(new_path)
        imgpath=imgpath.convert('RGB')
        img = np.asarray(imgpath)
        img = cv2.resize(img, (240, 240))
        img=img/255.
        dataset_testing.append([img, directory])
```

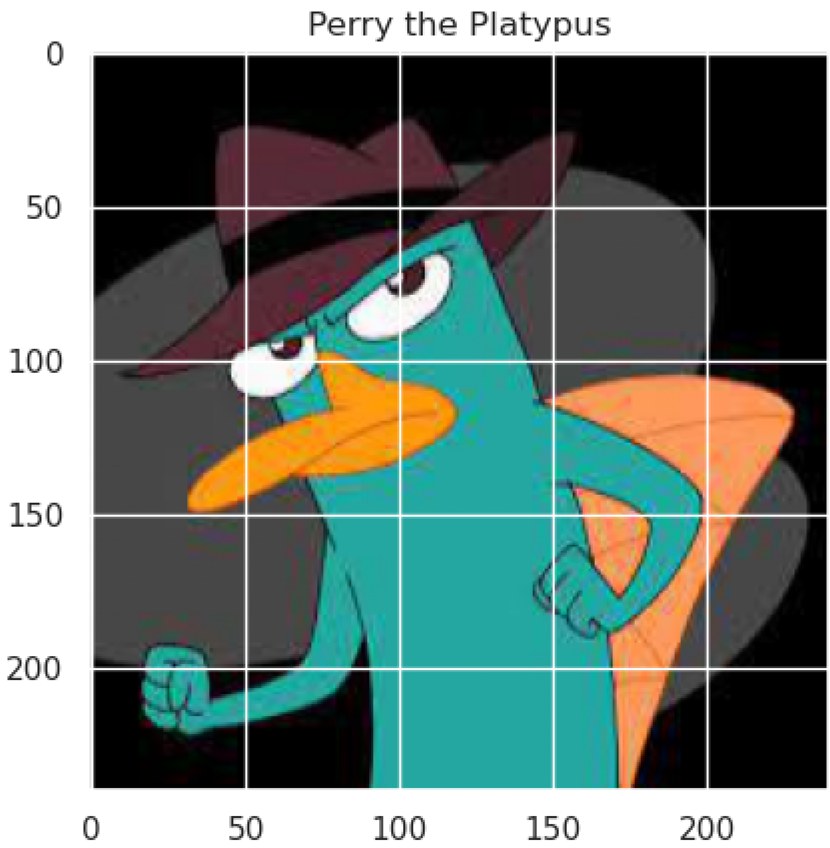
In [17]:
```
testing_data, testing_labels = zip(*dataset_testing)
testing_temp = list(map(str_to_Int_mapper, testing_labels))
testing_labels = keras.utils.to_categorical(testing_temp)
```

In [18]:
```
model.evaluate(np.array(testing_data), np.array(testing_labels))
```

```
1/1 [==============================] - 2s 2s/step - loss: 0.6625 - accuracy: 0.7222
```

Out[18]:
```
[0.6625138521194458, 0.7222222089767456]
```

In [19]:
```
plt.imshow(testing_data[1])
if testing_labels[1][0]==1:
    plt.title("Perry the Platypus")
else:
    plt.title("Platypus")
```



In [20]:

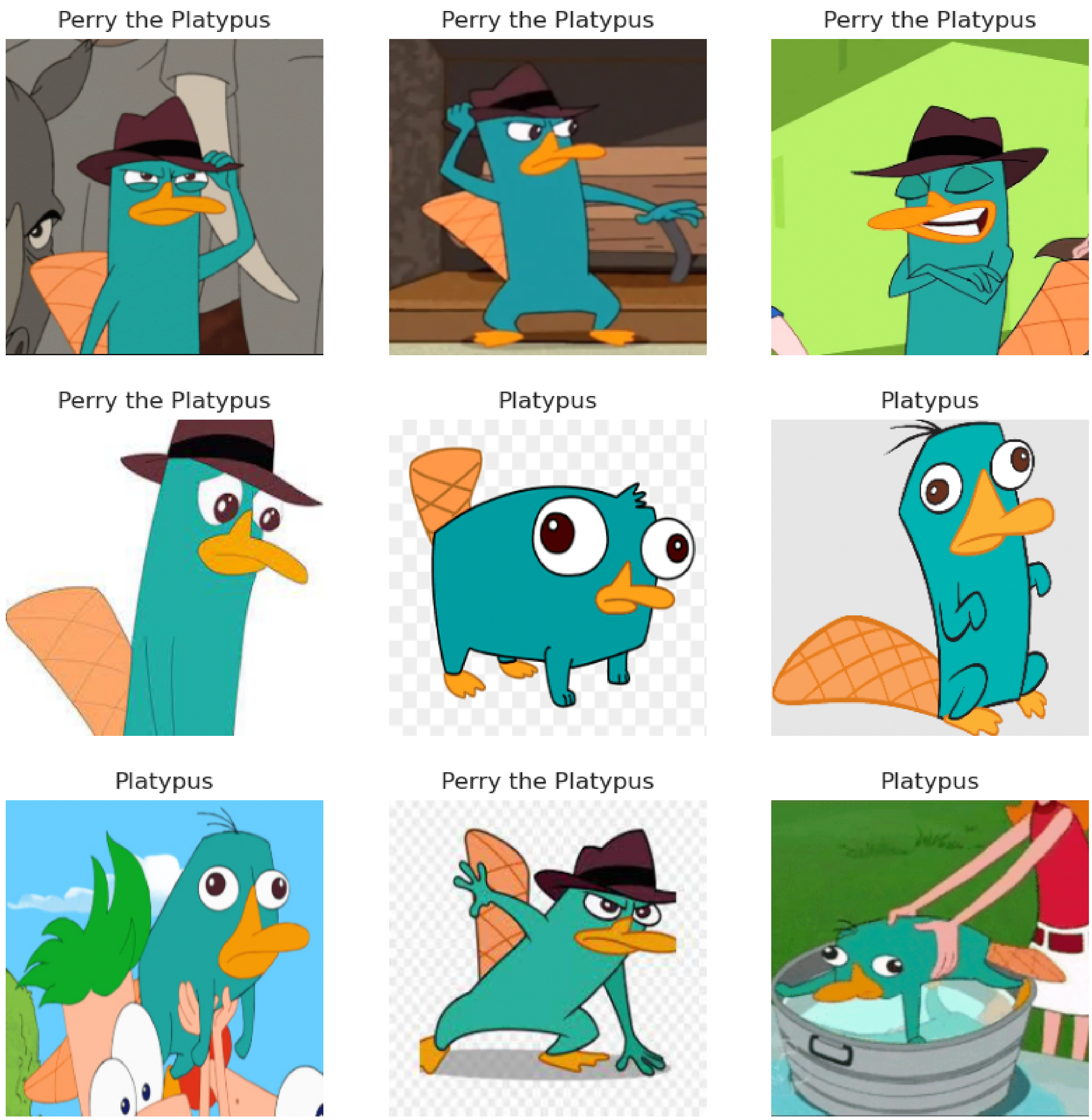
```
len(testing_data)
```

Out[20]: 18

Plotting test data

In [21]:

```
# Select num_images random images from the testing_data array
indices = random.sample(range(len(testing_data)), 9)
images = np.array(testing_data)[indices]
label = np.array(testing_labels)[indices]

# Create a grid of subplots with the specified dimensions
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(10, 10))

# Plot each image with its corresponding label as the title
for i, ax in enumerate(axes.flat):
    ax.imshow(images[i])
    if label[i][0]==1:
        ax.set_title("Perry the Platypus")
    else:
        ax.set_title("Platypus")
    ax.axis('off')

# Display the plot
plt.show()
```



References:

1. https://www.tensorflow.org/tutorials/images/data_augmentation