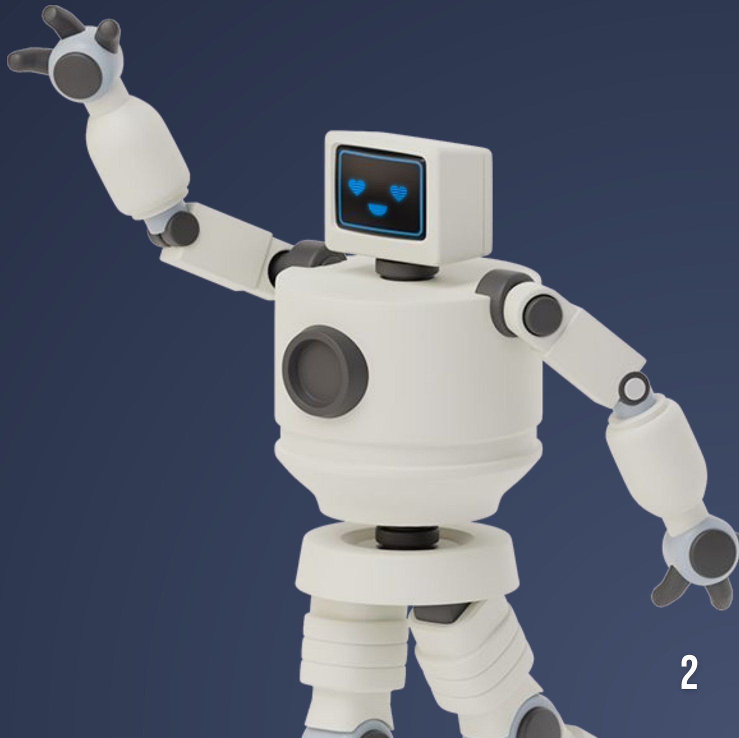
DRONE-ASSISTED WILDLIFE SURVEILLANCE USING DEEP LEARNING-BASED IMAGE ANALYSIS.

CSE4060: INTELLIGENT ROBOTS & DRONE TECHNOLOGY

JCOMPONENT PROJECT REVIEW

- Pranshu Choubey (20BAI1069)



ABSTRACT

The idea is to implement an automated approach for object detection by combining the power of drones and deep learning and integrating drones with advanced image analysis capabilities.

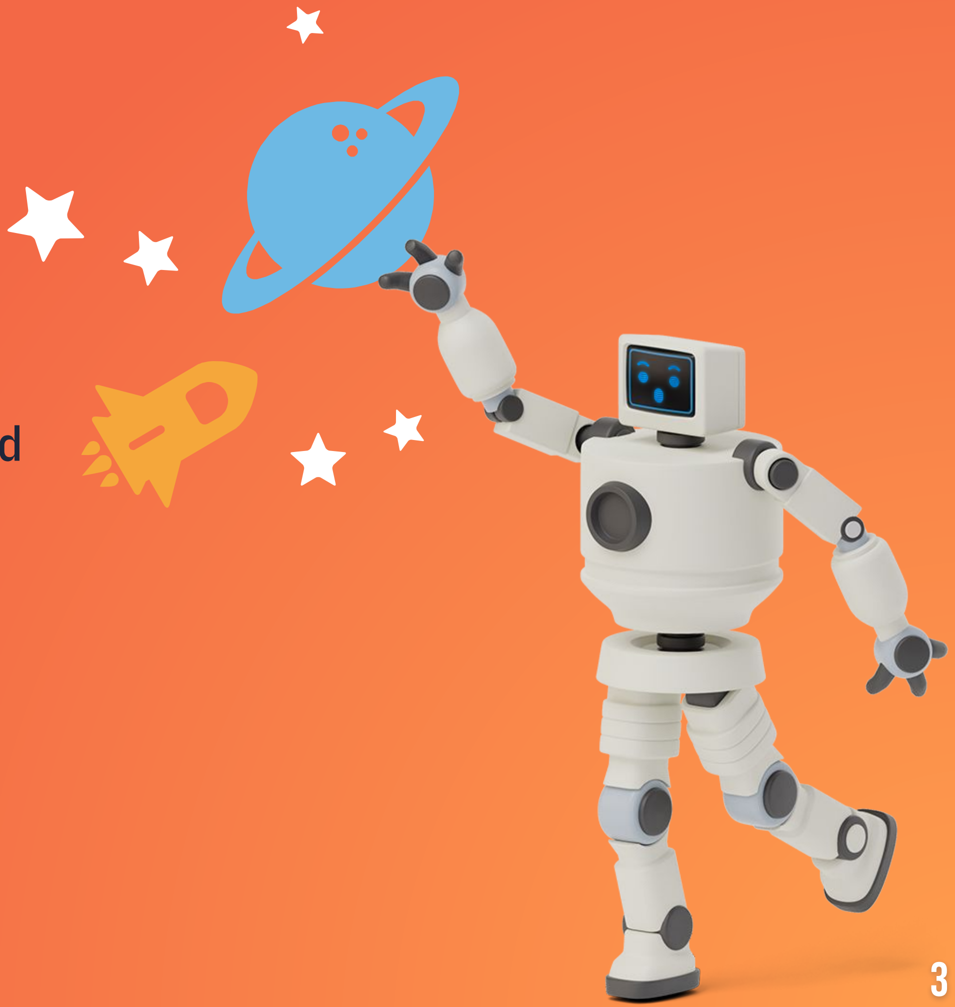
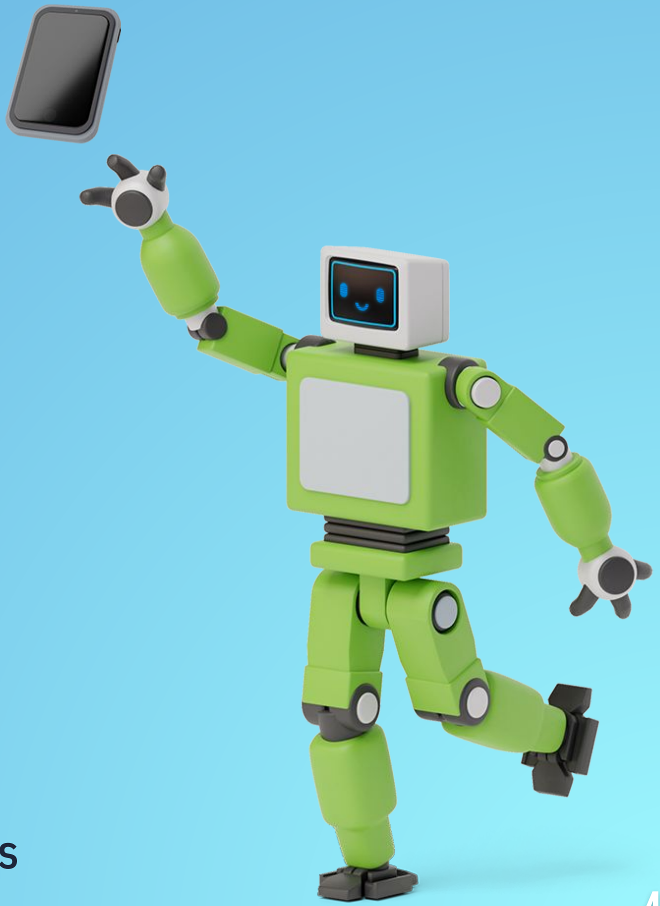


TABLE OF CONTENTS:



INTRODUCTION: AI & DRONES

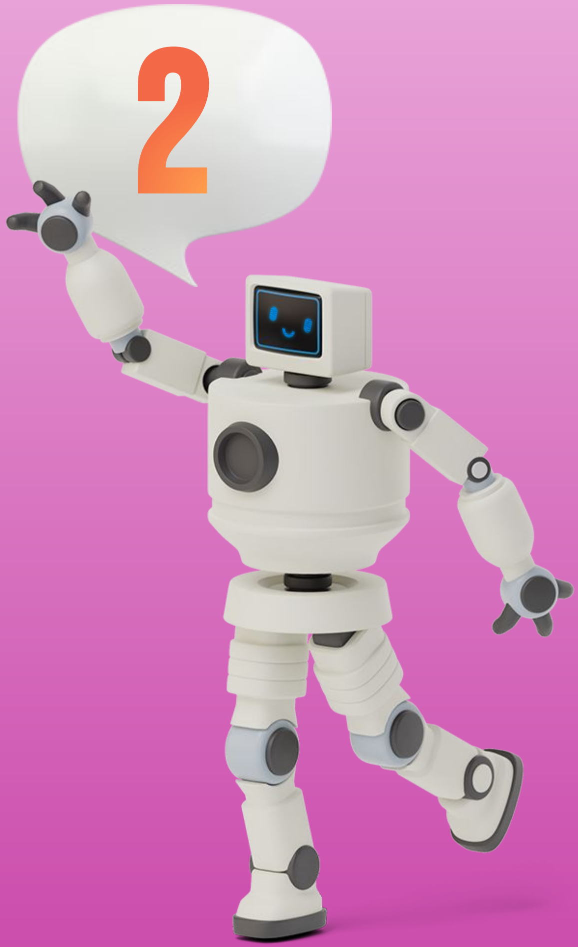
AI and drones are a powerful combination that is rapidly transforming many industries, including wildlife monitoring and conservation, agriculture, and public safety.

Drones equipped with advanced sensors, cameras, and AI-powered algorithms can autonomously collect and analyze large amounts of data in real-time, providing valuable insights and enhancing decision-making capabilities.

With AI, drones can be trained to perform tasks such as object detection and classification, mapping, and even predictive maintenance.

The use of AI in drones is driving innovation and efficiency across various sectors and is expected to continue to revolutionize the way we work, live, and interact with the world around us.

NEED FOR AI POWERED DRONES



The need for AI-powered drones stems from the growing demand for more efficient, accurate, and cost-effective ways to collect and analyze data in various industries. Drones equipped with AI algorithms can autonomously perform complex tasks, such as object detection and classification, precision mapping, and predictive maintenance, with greater accuracy and speed than traditional methods.

In wildlife conservation, for example, AI-powered drones can monitor and track endangered species, helping conservationists to protect and manage their habitats. In agriculture, drones can be used to identify and treat crops with precision, reducing waste and increasing yields. In public safety, drones equipped with AI can help law enforcement agencies to monitor crowds and identify potential threats, improving situational awareness and response times.

BENEFITS OF USING AI POWERED DRONE FOR WILDLIFE MONITORING

Improved accuracy

A deep learning model can accurately classify different types of animals based on their features, improving the accuracy of wildlife monitoring, compared to manual observation.

Wide coverage

A drone can cover a large area quickly and efficiently, allowing for more comprehensive monitoring of wildlife populations, useful in remote or inaccessible areas.

Enhanced Safety

Drones can be used to monitor wildlife in hazardous or hard-to-reach areas, reducing the risk to human personnel.

Reduced costs

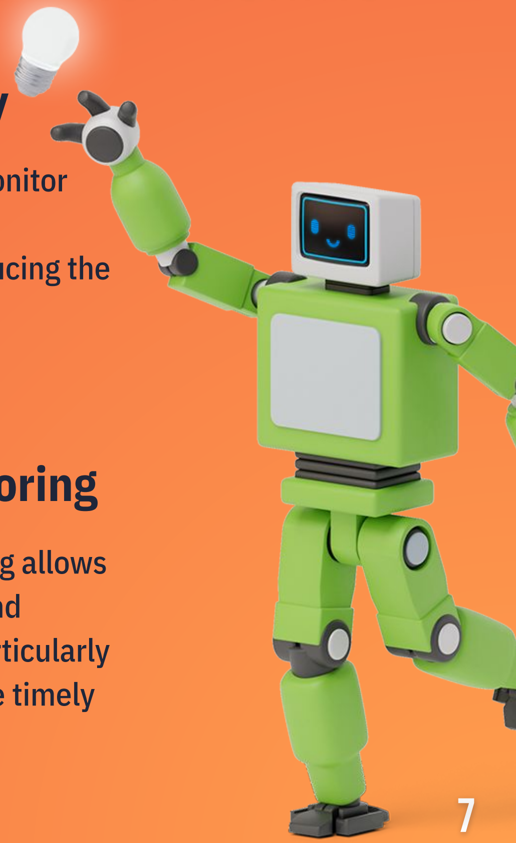
It’s more cost-effective as a drone can eliminate the need for expensive helicopter surveys, and the data can be analyzed more efficiently using deep learning algorithms.

Non-invasive

Contrary to traditional methods, drones equipped with DL models can capture images without disturbing the animals, minimizing the impact on their natural behavior and habitat.

Real-time monitoring

Real-time image capturing allows for immediate analysis and response. This can be particularly useful in situations where timely action is required.



ROADMAP FOR MODEL DEPLOYMENT

1. Choose a suitable drone compatible with the DL model, consider the size, weight, battery life, and computing power, it should also have a high-quality camera to capture clear images for classification.

2. Prepare the drone for deployment by connecting it to a computer and installing any necessary software and drivers. Calibrate and configure the drone, ensure that the camera is working correctly.

3. Convert the model to a format suitable for the drone i.e. to a format compatible with the drone's hardware. A conversion tool provided by the DL framework, such as TensorFlow Lite or ONNX, could be used.

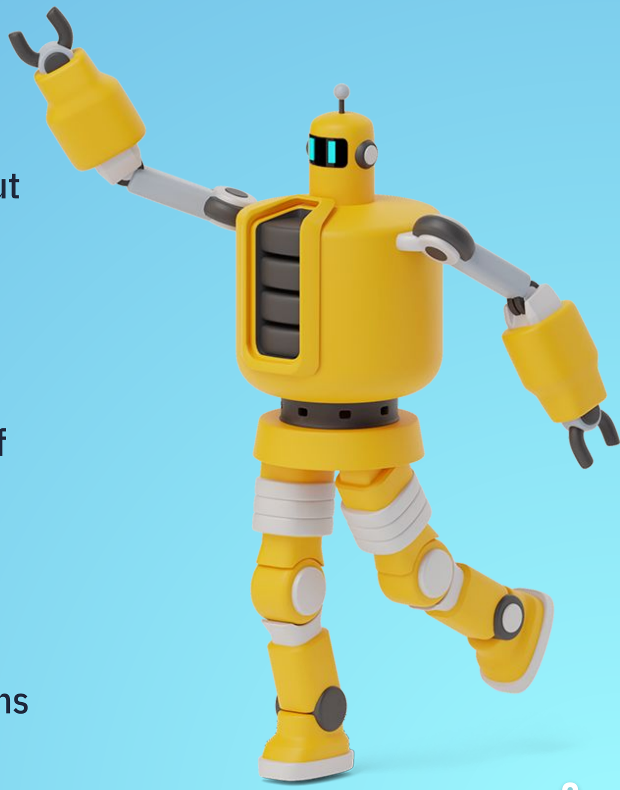
4. Transfer the model to the drone after conversion, a USB cable, SD card, or wireless connection could be used. Ensure that the model is saved in a location that is easily accessible by the software running on the drone.

5. Integrate the model with the drone's software programming language such as Python to write code that interfaces with the drone's camera and controls the flow of data to and from the deep learning model.

6. Test and optimize the model, collect real-world data from the drone's camera and use it to evaluate the accuracy of the model. Fine-tune the model's parameters to optimize its performance and test the model under different conditions to ensure its robustness.

THE ANN MODEL FOR IMAGE CLASSIFICATION

- ANN is a type of deep learning model used for image classification. It consists of interconnected artificial neurons that receive input signals and perform mathematical computations to produce output signals.
- The input signals are the pixel values of the input image, and the output signals are the class labels for the image.
- The weights and biases of the artificial neurons are learned through a training process using large datasets of labeled images.
- The training process involves adjusting the weights and biases to minimize the error between the predicted output and the true labels of the training data.
- Once the ANN model has been trained, it can be used to classify new, unseen images with high accuracy.
- The performance of the ANN model depends on the number of hidden layers, the number of neurons in each layer, and the activation functions used in the neurons.



jupyter WildlifeClassification Last Checkpoint: 8 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Help

Trusted | tfenv

Run | Code | Commit

This line of code is creating an instance of a PyTorch model and moving it to a specified device, so that it can be trained and used for prediction.

In [26]:
```
model = to_device(Model(input_size, output_size), device)
```

In [27]:
```
history = [evaluate(model, val_loader)]
history
```

Out[27]:
```
[{'val_loss': 1.0997869968414307, 'val_acc': 0.34062498807907104}]
```

In [28]:
```
history += fit(3, 0.05, model, train_loader, val_loader)
```

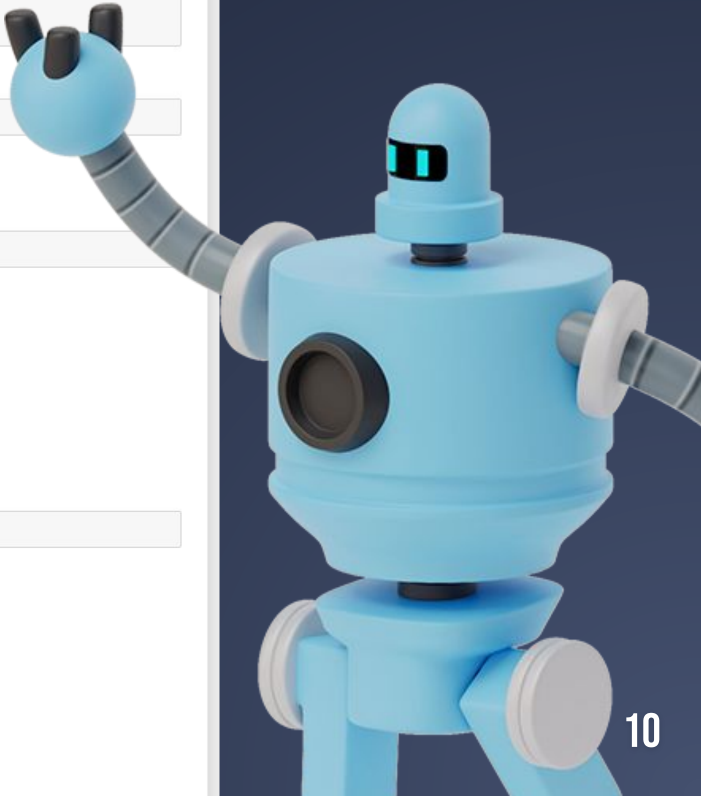
```
Epoch [0], val_loss: 1.0998, val_acc: 0.3423
Epoch [1], val_loss: 1.0947, val_acc: 0.3259
Epoch [2], val_loss: 1.0968, val_acc: 0.3861
```

In [29]:
```
history += fit(10, 0.001, model, train_loader, val_loader)
```

```
Epoch [0], val_loss: 1.0699, val_acc: 0.4176
Epoch [1], val_loss: 1.0651, val_acc: 0.4290
Epoch [2], val_loss: 1.0619, val_acc: 0.4352
Epoch [3], val_loss: 1.0604, val_acc: 0.4384
Epoch [4], val_loss: 1.0588, val_acc: 0.4443
Epoch [5], val_loss: 1.0585, val_acc: 0.4446
Epoch [6], val_loss: 1.0582, val_acc: 0.4446
Epoch [7], val_loss: 1.0584, val_acc: 0.4415
Epoch [8], val_loss: 1.0583, val_acc: 0.4415
Epoch [9], val_loss: 1.0579, val_acc: 0.4460
```

In [30]:
```
history += fit(10, 0.02, model, train_loader, val_loader)
```

```
Epoch [0], val_loss: 1.0728, val_acc: 0.4389
Epoch [1], val_loss: 1.0931, val_acc: 0.3960
Epoch [2], val_loss: 1.0574, val_acc: 0.4423
Epoch [3], val_loss: 1.1080, val_acc: 0.4119
Epoch [4], val_loss: 1.0474, val_acc: 0.4886
Epoch [5], val_loss: 1.0494, val_acc: 0.4702
Epoch [6], val_loss: 1.0406, val_acc: 0.4858
Epoch [7], val_loss: 1.0444, val_acc: 0.4702
Epoch [8], val_loss: 1.0366, val_acc: 0.4608
Epoch [9], val_loss: 1.0438, val_acc: 0.4861
```



jupyter WildlifeClassification Last Checkpoint: 6 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Help Trusted tfenv

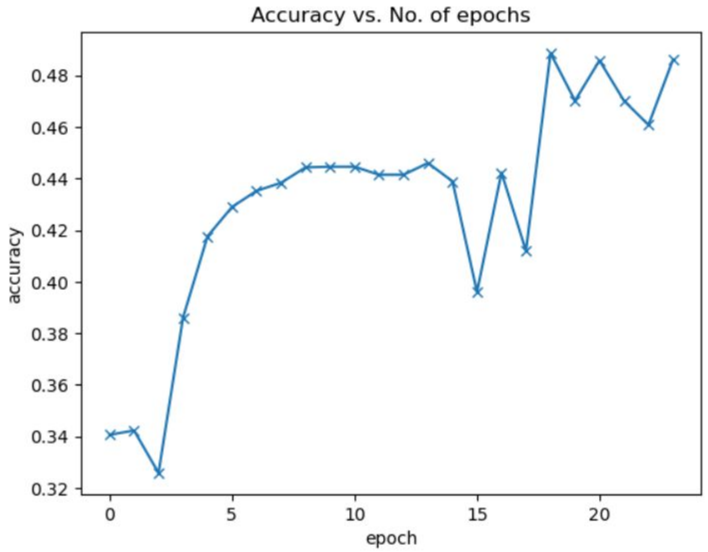
Run Code Commit

```
Epoch [6], val_loss: 1.0406, val_acc: 0.4858
Epoch [7], val_loss: 1.0444, val_acc: 0.4702
Epoch [8], val_loss: 1.0366, val_acc: 0.4608
Epoch [9], val_loss: 1.0438, val_acc: 0.4861
```

Analyze the Model

In [31]:
```
def plot_accuracies(history):
    accuracies = [x['val_acc'] for x in history]
    plt.plot(accuracies, '-x')
    plt.xlabel('epoch')
    plt.ylabel('accuracy')
    plt.title('Accuracy vs. No. of epochs');
```

In [32]:
```
plot_accuracies(history)
```



jupyter WildlifeClassification Last Checkpoint: 4 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Help

Trusted | tfenv

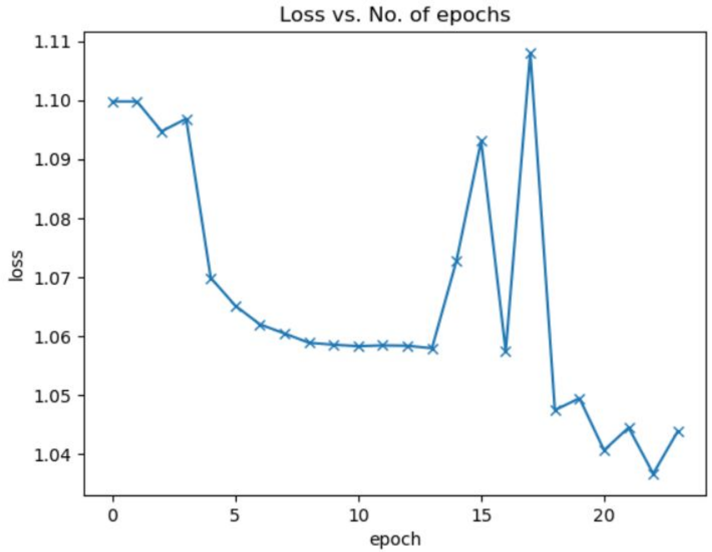
Run | Code | Commit

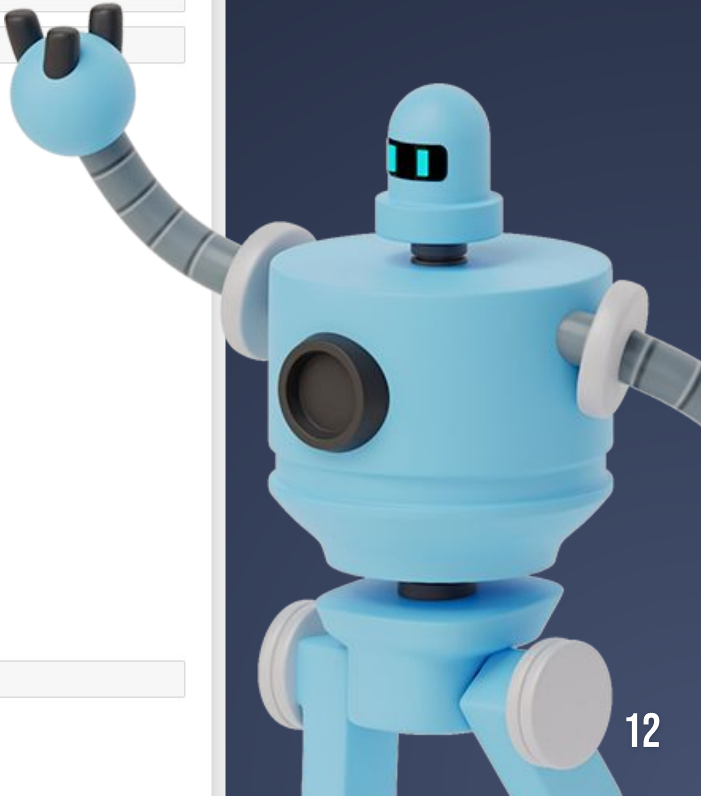
In [33]:
```
def plot_losses(history):
    losses = [x['val_loss'] for x in history]
    plt.plot(losses, '-x')
    plt.xlabel('epoch')
    plt.ylabel('loss')
    plt.title('Loss vs. No. of epochs');
```

Refer the complete notebook from the GitHub link below

https://github.com/pranshu911/WildlifeClassification

In [34]:
```
plot_losses(history)
```





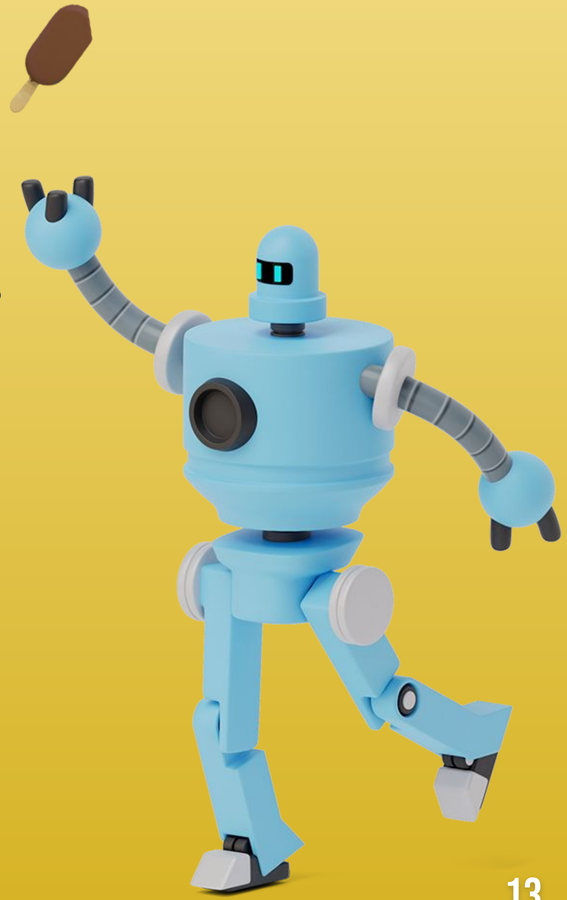
In [35]:
```
evaluate(model, test_loader)
```

Out[35]: `{'val_loss': 0.9714632034301758, 'val_acc': 0.5352272987365723}`

The above model has an accuracy of 53.52%

THE CNN MODEL FOR IMAGE CLASSIFICATION

- CNN is a type of deep learning model used for image classification and analysis. CNN is designed to recognize patterns in images, by learning features from local regions of the image using convolutional layers.
- A CNN consists of multiple layers, including convolutional layers, pooling layers, and fully connected layers.
    - Convolutional layers apply a set of filters to the input image, generating a feature map that highlights important regions of the image.
    - Pooling layers downsample the feature maps to reduce the size and computational complexity of the model, while preserving important features.
    - Fully connected layers apply weights and biases to the output of the previous layers, generating a probability distribution over the possible class labels for the image.



CNN MODEL CONTD.

- The weights and biases of the CNN model are learned through a training process using large datasets of labeled images.
- The training process involves adjusting the weights and biases to minimize the error between the predicted output and the true labels of the training data.
- Once the CNN model has been trained, it can be used to classify new, unseen images with high accuracy.
- The performance of the CNN model depends on the architecture of the model, the number of layers, and the hyperparameters used during training

jupyter WildlifeClassification Last Checkpoint: 16 minutes ago (autosaved)

Logout

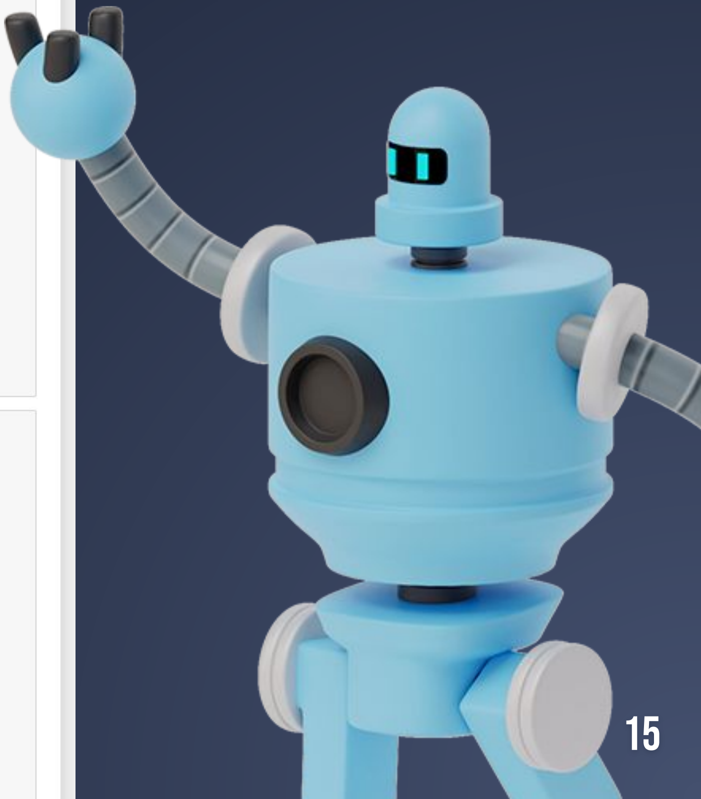
File Edit View Insert Cell Kernel Help

Trusted tfenv

Run Code Commit

In [40]:
```
def accuracy(outputs, labels):
    _, preds = torch.max(outputs, dim=1)
    return torch.tensor(torch.sum(preds == labels).item() / len(preds))

class ImageClassificationBase(nn.Module):
    def training_step(self, batch):
        images, labels = batch
        out = self(images)                  # Generate predictions
        loss = F.cross_entropy(out, labels) # Calculate loss
        return loss

    def validation_step(self, batch):
        images, labels = batch
        out = self(images)                    # Generate predictions
        loss = F.cross_entropy(out, labels)   # Calculate loss
        acc = accuracy(out, labels)           # Calculate accuracy
        return {'val_loss': loss.detach(), 'val_acc': acc}

    def validation_epoch_end(self, outputs):
        batch_losses = [x['val_loss'] for x in outputs]
        epoch_loss = torch.stack(batch_losses).mean()   # Combine losses
        batch_accs = [x['val_acc'] for x in outputs]
        epoch_acc = torch.stack(batch_accs).mean()      # Combine accuracies
        return {'val_loss': epoch_loss.item(), 'val_acc': epoch_acc.item()}

    def epoch_end(self, epoch, result):
        print("Epoch [{}], train_loss: {:.4f}, val_loss: {:.4f}, val_acc: {:.4f}".format(
            epoch, result['train_loss'], result['val_loss'], result['val_acc']))
```

In [41]:
```
class CnnModel(ImageClassificationBase):
    def __init__(self):
        super().__init__()
        self.network = nn.Sequential(
            nn.Conv2d((3,400,400), 100, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(100, 150, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2), # output: 150 x 16 x 16

            nn.Conv2d(150, 200, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.Conv2d(200, 200, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2), # output: 200 x 8 x 8

            nn.Conv2d(200, 250, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.Conv2d(250, 250, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
```



jupyter WildlifeClassification Last Checkpoint: 12 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Help

Trusted | tfenv

Run | Code | Commit

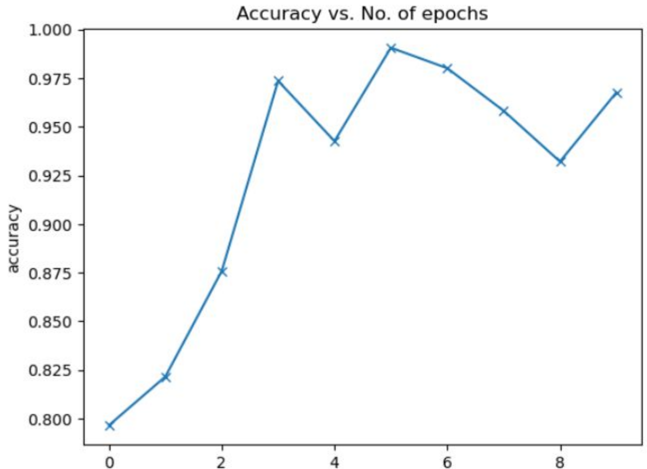
In [52]:
```
history = fit(num_epochs, lr, model, train_dl, val_dl, opt_func)
```

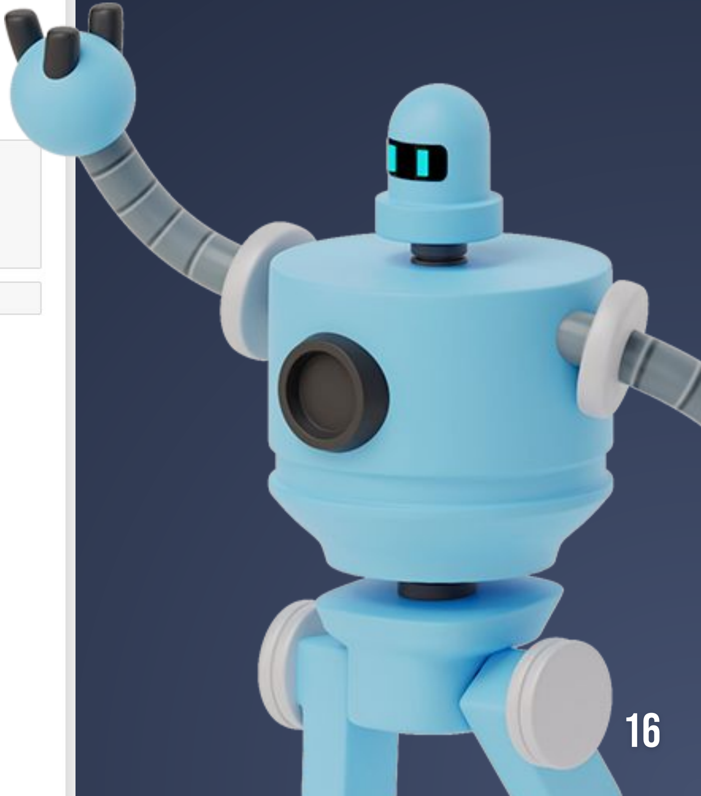
```
Epoch [0], train_loss: 0.6143, val_loss: 0.7334, val_acc: 0.7966
Epoch [1], train_loss: 0.5886, val_loss: 0.7235, val_acc: 0.8216
Epoch [2], train_loss: 0.5764, val_loss: 0.6599, val_acc: 0.8759
Epoch [3], train_loss: 0.5668, val_loss: 0.5823, val_acc: 0.9736
Epoch [4], train_loss: 0.5639, val_loss: 0.6075, val_acc: 0.9426
Epoch [5], train_loss: 0.5632, val_loss: 0.5650, val_acc: 0.9906
Epoch [6], train_loss: 0.5561, val_loss: 0.5726, val_acc: 0.9801
Epoch [7], train_loss: 0.5549, val_loss: 0.5862, val_acc: 0.9582
Epoch [8], train_loss: 0.5608, val_loss: 0.6182, val_acc: 0.9321
Epoch [9], train_loss: 0.5656, val_loss: 0.5849, val_acc: 0.9676
```

Analyze the Model

In [53]:
```
def plot_accuracies(history):
    accuracies = [x['val_acc'] for x in history]
    plt.plot(accuracies, '-x')
    plt.xlabel('epoch')
    plt.ylabel('accuracy')
    plt.title('Accuracy vs. No. of epochs');
```

In [54]:
```
plot_accuracies(history)
```





jupyter WildlifeClassification Last Checkpoint: 14 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Help

Trusted | tfenv

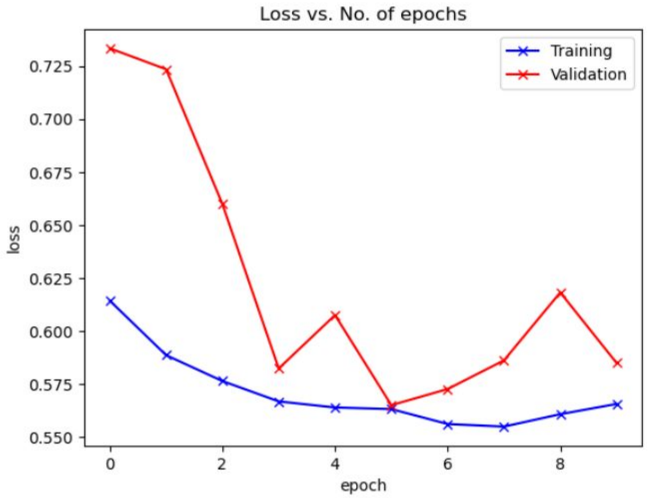
Run | Code | Commit

In [55]:
```
def plot_losses(history):
    train_losses = [x.get('train_loss') for x in history]
    val_losses = [x['val_loss'] for x in history]
    plt.plot(train_losses, '-bx')
    plt.plot(val_losses, '-rx')
    plt.xlabel('epoch')
    plt.ylabel('loss')
    plt.legend(['Training', 'Validation'])
    plt.title('Loss vs. No. of epochs');
```

Refer the complete notebook from the GitHub link below

https://github.com/pranshu911/WildlifeClassification

In [56]:
```
plot_losses(history)
```

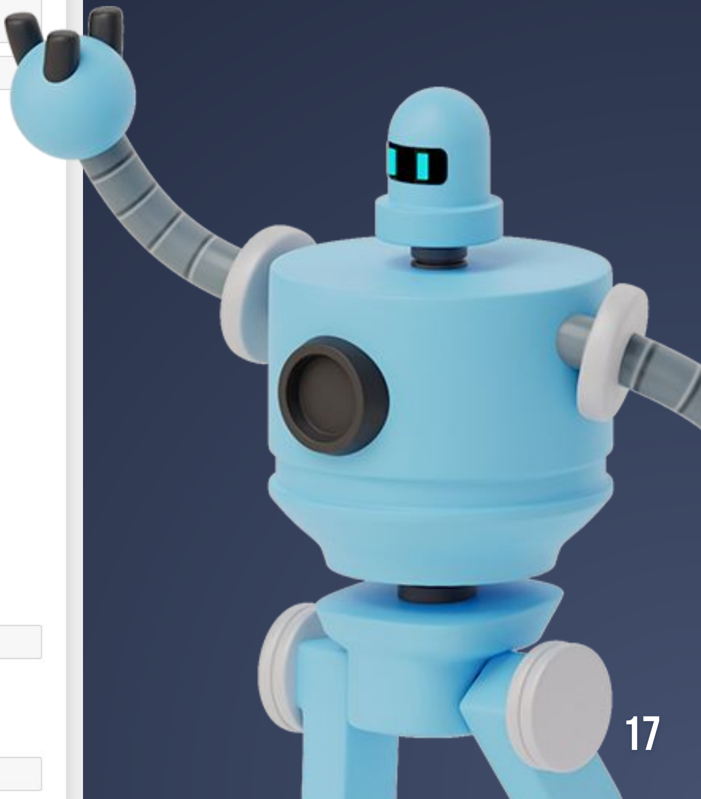


In [57]:
```
evaluate(model, test_loader)
```

Out[57]:
```
{'val_loss': 0.5840606689453125, 'val_acc': 0.9659091234207153}
```

The CNN model has an accuracy of 96.59% as compared to the previous one which had only 53.52%

In [2]:
```
!pip install jovian --upgrade --quiet
```



MODEL COMPARISON: ANN VS CNN

ANN

- Accuracy = 53.52%
- Train time for 10 Epochs = 15 minutes
- Feature Extraction inefficient due to fully connected layers
- Cannot capture spatial relationships among features
- Cannot handle image translations and distortions
- General purpose, non-image classification tasks

CNN

- Accuracy = 96.59%
- Train time for 10 Epochs = 2 hours 30 minutes
- Feature Extraction efficient due to convolutional and pooling layers
- Can capture spatial relationships among features
- Can handle image translations and distortions
- Image classification, object detection, and segmentation

ANNs are well-suited for general-purpose tasks, CNNs are specifically designed for image-related tasks such as classification, object detection, and segmentation.
This is due to their ability to capture spatial relationships among features, learn hierarchical representations of input data, and handle image translations and distortions.

WHY CNN IS BETTER THAN ANN FOR IMAGE CLASSIFICATION TASKS

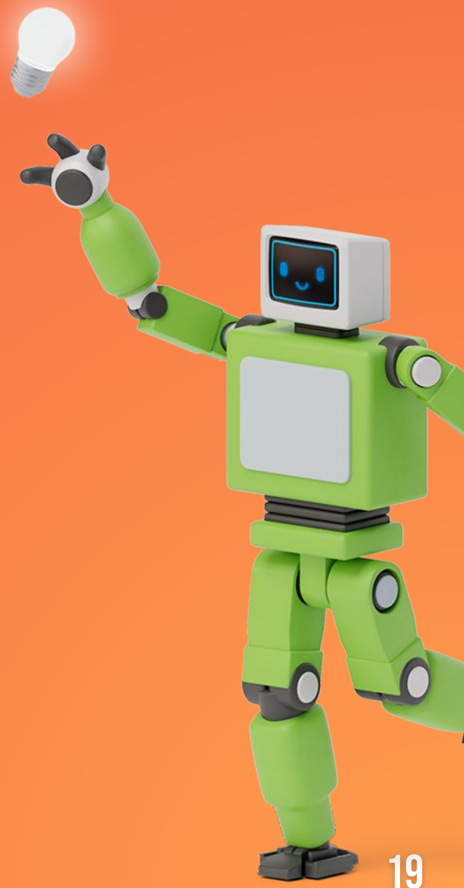
Handling of spatial data: CNNs are specifically designed to handle spatial data such as images, whereas ANNs treat input data as a flat vector, ignoring the spatial relationships among pixels in the image.

Parameter sharing: CNNs use parameter sharing, where a single set of parameters is applied to multiple regions of the input image. This reduces the number of parameters in the model and improves its ability to generalize to new images.

Local connectivity: In a CNN, each neuron is connected only to a small, local region of the input image, allowing the model to focus on local patterns and features.

Hierarchical feature learning: CNNs are able to learn hierarchical representations of the input image, where low-level features such as edges and corners are learned in lower layers, and higher-level features such as shapes and textures are learned in higher layers.

Translation invariance: CNNs are able to recognize the same pattern in different regions of the input image, thanks to the use of pooling layers that downsample the feature maps, making the model translation invariant.



STEPS TO INTEGRATE A TRAINED CNN MODEL WITH THE DRONE'S SOFTWARE

Establish a communication link between the drone and a computer: This can be achieved through a USB cable or a wireless connection such as Wi-Fi or Bluetooth.

Capture images and videos: Use the drone's remote controller or a software interface to control the drone camera and capture images or start recording videos of the wildlife being monitored.

Send the captured data to the computer: Once the data is captured, send it to a computer for analysis in real-time or after the data collection is complete.

Analyze the data using the trained CNN model: Process the data to prepare it for analysis, such as resizing images or normalizing pixel values, and use the trained CNN model to classify the images or videos based on learned features.

Send commands to the drone based on the analysis results: Based on the analysis results, send commands back to the drone to adjust its flight path or behavior, such as tracking an animal or changing its altitude to get a better view.

PX4

BUILDING THE DRONE

DRONE COMPONENTS

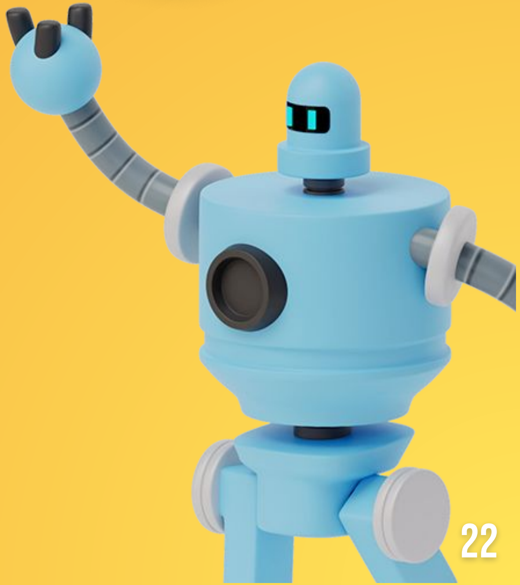
The drone features a **lightweight frame**, selected to ensure durability and stability during flight.

It is powered by **brushless motors** and **propellers** that are designed for optimal lift and thrust.

The controller is paired with **electronic speed controllers** (ESCs) that regulate the speed of the motors and ensure that the drone operates efficiently.

For remote control, we have included a r**adio transmitter** and **receiver** that allows us to control the drone's movements, direction, and altitude with precision.

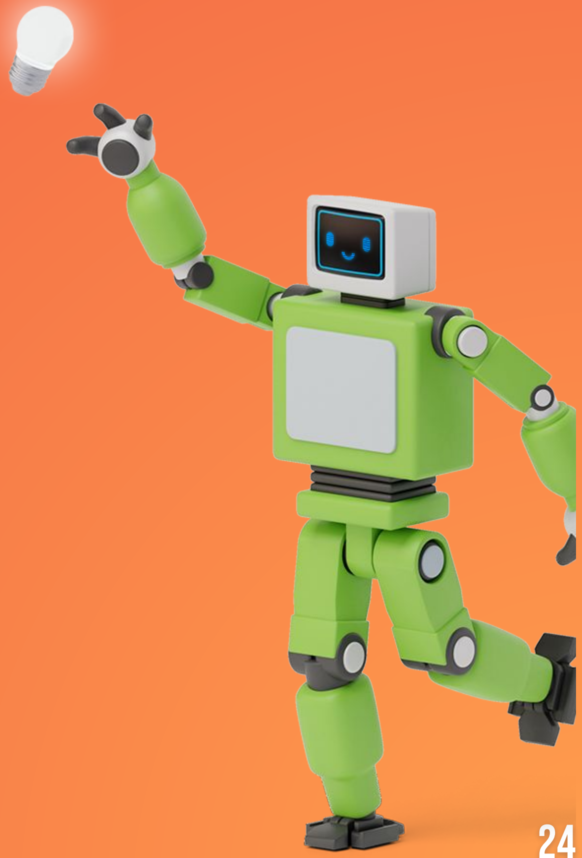
Installed with a high resolution **camera**, capable of capturing clear and detailed footage in real-time, and viewed on an LCD **display**.



SPECIFICATIONS

- We have chosen the **A2212/13T 1000KV motor** for our drone because of its high efficiency and reliability. This motor is designed to deliver powerful performance while consuming less power, making it ideal for extended flight times.
- To power our motor, we have selected the **GF 10x5R propeller**. This propeller has a high thrust-to-weight ratio, making it perfect for our drone's needs. It is also durable and efficient, ensuring stable flight even in adverse weather conditions.
- For controlling our drone, we have chosen the **FS-R6B controller**. This controller provides precise and accurate control over the drone's movement and features multiple channels for added flexibility. Its compact design also makes it easy to install and use.

- To display live video footage from our drone, we will be using the **Waveshare 5-inch HDMI Display**. This high-resolution display provides clear and vivid images and can be easily connected to our drone's camera system.
- For capturing high-quality video footage, we have chosen the **Zinq USB 1080p camera**. This camera is compact and lightweight, making it easy to mount on our drone. It also features advanced image stabilization technology, ensuring smooth and steady footage even during fast movements.
- Finally, to power our drone, we have selected the **RANGE 18650 Li-ion battery pack**. This battery pack is medium weight and compact, making it ideal for our drone's needs. It also has a high capacity of 4400mAh, providing us with extended flight times.
- Additionally, we have opted for a **Raspberry Pi 4 with Ubuntu** (64-bit + 8GB RAM) for our onboard computer system. This will allow us to run complex programs and algorithms that will enable our drone to perform advanced functions.




ESC
FC
Motor
Propeller
Camera
Landing Gear

```python
from djitellopy import Tello
import cv2
import numpy as np

def initializeTello():
    myDrone=Tello()
    myDrone.connect()
    myDrone.for_back_velocity=0
    myDrone.left_right_velocity=0
    myDrone.up_down_velocity=0
    myDrone.streamon()
    return myDrone


def telloGetFrame(myDrone,w=360,h=240)
    myFrame=mydrone.get_frame_read()
    myFrame=myFrame.frame
    img=cv2.resize(myFrame,(w,h))
    return img

def findFace(img):
    faceCascade=cv2.CascadeClassifier('haarcascade_frontalface_default')
    imgGray=cv2.cvtColor(img,cv.Color_BGR2GRAY)
    faces= faceCascade.detectMultiScale(imgGray,1,2,4)

    myFaceListC=[]
    myFaceListArea=[]
```

```python
from utilis import *
import cv2

w,h= 480,360
pid=[0.5,0.5,0]
pError=0
startCounter=0 ##  testing for no flight put this as 1, and for flight put this as 0

myDrone = initializeTello()

while True:


    ## Flight
    if startCounter==0:
        myDrone.takeoff()
        startCounter = 1
    ## step 1
    img = telloGetFrame(myDrone,w,h)
    ## step 2
    img,info = findFace(img)
    ## step 3
    pError= trackFace(myDrone,info,w,pis,pError)
    print(info[0] [0])
    cv2.imshow('image',img)
    if cv2.waitkey(1) & 0xFF== ord('q'):
        myDrone.land()
        break
```

THANK YOU