Jungle Detectives: AI-Powered Image Classification of Wild Big Cats

Team-592617

Sushil Krishnan

Aaryan Bansal

Rounak Sonthalia

1. INTRODUCTION

Jungle Detectives is a transfer-learning based web app to identify the 10 big cats based on uploaded image. The app leverages transfer learning model to distinguish the majestic big cats. The big cats that can be identified are awe-inspiring Bengal tiger, the elusive snow leopard, the powerful African lion, the agile cheetah, the mysterious clouded leopard, the majestic jaguar, the enigmatic cougar, the endangered Amur leopard, the graceful serval, and the formidable leopard. By combining image recognition techniques with extensive training data, this state-of-the-art model provides conservationists, wildlife researchers, and enthusiasts with a powerful tool to safeguard and appreciate the beauty and diversity of these magnificent creatures, contributing to their preservation and understanding in their natural habitats. With the advancement of technology, many new techniques of Deep learning have contributed towards classification of images in a more efficient way such as ResNet, VGG16, Inception V3 etc.

1.1 Project Overview

The primary goal of this project is to develop a robust image classification system capable of accurately identifying and categorizing different types of big cats from large datasets of images. The system will leverage deep learning techniques to develop a high accuracy tool employing transfer learning approaches, that can accurately identify and classify ten big cat species from their images, despite the challenges of natural variation in appearance, pose, and lighting conditions.

Key Components:

1. Data Collection and Preprocessing
2. Model Development
3. Evaluation and Validation
4. Deployment
5. User Interface

1.2 Purpose

Big cats play an important role in their ecosystems as apex predators. They help to control populations of prey animals, which can prevent overgrazing and other ecological problems. However, big cats are facing a number of ecological problems, including habitat loss and fragmentation, poaching, prey depletion and climate change. These ecological problems are having a significant impact on big cat populations. The project can help to address these ecological problems by providing conservationists and wildlife researchers with a powerful tool to track and monitor big cat populations. The model can be used to identify these magnificent creatures.

2. LITERATURE SURVEY

2.1 Existing problem

While the "Big Cat Image Classification" project aims to address specific challenges related to automated big cat image classification, there are potential existing problems in this domain that the project seeks to overcome. Some of these challenges include:

- Scalability Issues:
  - Manual classification becomes impractical as the volume of data increases. Handling a large number of images manually is inefficient and may not be scalable with extensive datasets.
- Subjectivity and Inconsistency:
  - Human classifiers may introduce subjectivity and inconsistencies in categorizing. Different individuals may interpret and classify images differently, leading to variations in results.
- Limited Automation:
  - The lack of automated systems for classification limits the efficiency.
- Data Security Concerns:
  - Manual classification processes may involve sharing sensitive images with external parties, raising concerns about data security and privacy.
- Cost and Resource Intensiveness:

- Manually training individuals to classify big cat images is resource-intensive and may involve significant costs. An automated solution can provide a more cost-effective and efficient alternative.

- Lack of Real-Time Insights:
  - Traditional manual processes do not provide real-time insights into the classification of images. An automated system can deliver immediate results, enhancing decision-making processes.
- Difficulty in Handling Diverse Data:
  - An automated system with a well-trained model can better handle this diversity.
- Inability to Learn and Improve Over Time:
  - Manual classifiers may not adapt well to changing patterns and trends. An automated system with continuous learning through retraining can improve accuracy over time.
- Insufficient Feedback Loop:
  - Traditional methods lack a systematic feedback loop to continuously improve the classification model. An automated solution can incorporate user feedback for ongoing enhancements.

2.2 References

- Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition.
- Tan, M., & Le, Q. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks.
- Pan, S. J., & Yang, Q. (2010). A Survey on Transfer Learning.
- Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). How Transferable Are Features in Deep Neural Networks?
- LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553).
- Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the

Inception Architecture for Computer Vision.

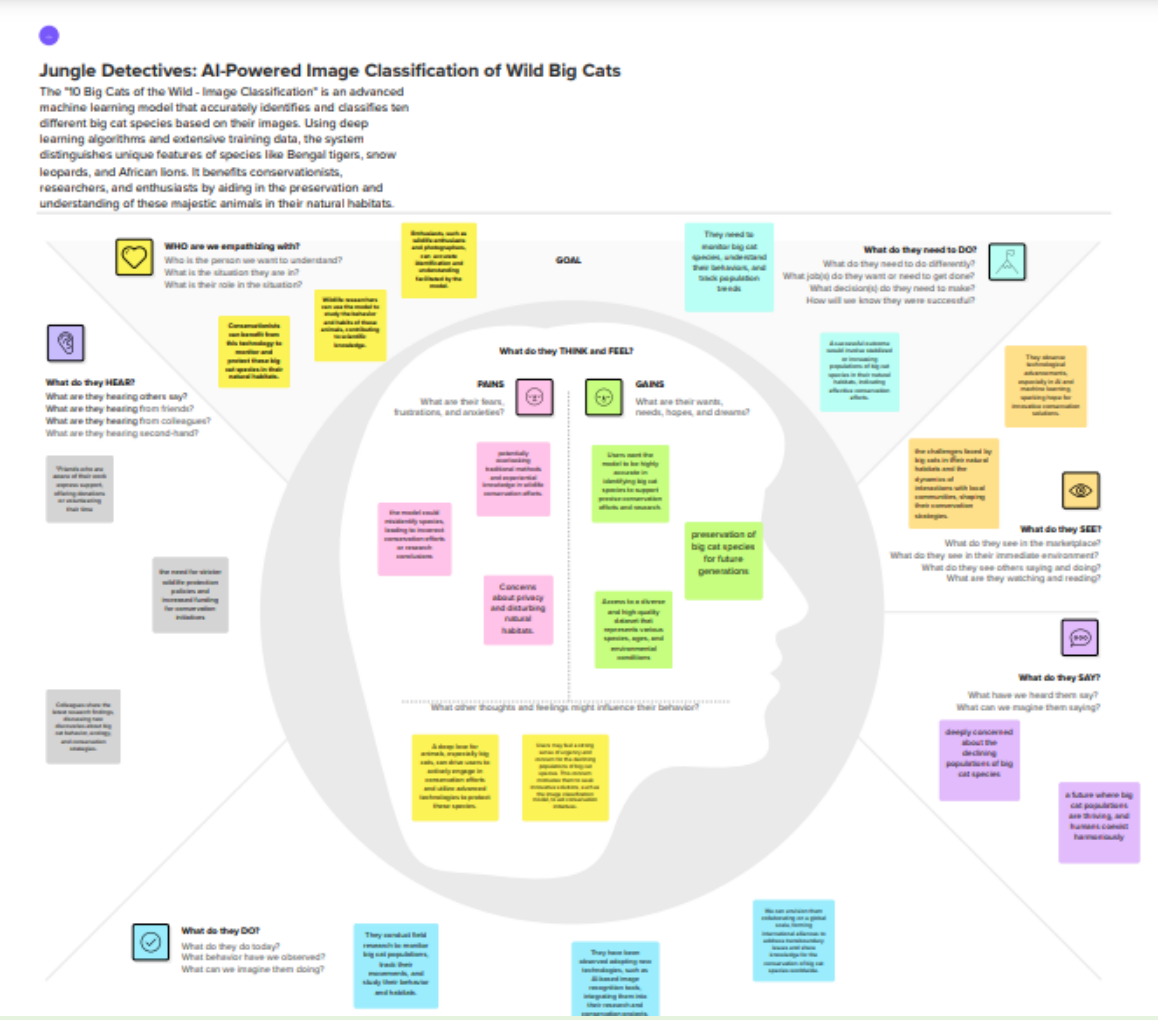
2.3 Problem Statement Definition

Jungle Detectives is a transfer-learning based web app to identify the 10 big cats based on uploaded image. The app leverages transfer learning model to distinguish the majestic big cats. The big cats that can be identified are awe-inspiring Bengal tiger, the elusive snow leopard, the powerful African lion, the agile cheetah, the mysterious clouded leopard, the majestic jaguar, the enigmatic cougar, the endangered Amur leopard, the graceful serval, and the formidable leopard. By combining image recognition techniques with extensive training data, this state-of-the-art model provides conservationists, wildlife researchers, and enthusiasts with a powerful tool to safeguard and appreciate the beauty and diversity of these magnificent creatures, contributing to their preservation and understanding in their natural habitats. With the advancement of technology, many new techniques of Deep learning have contributed towards classification of images in a more efficient way such as ResNet, VGG16, Inception V3 etc.

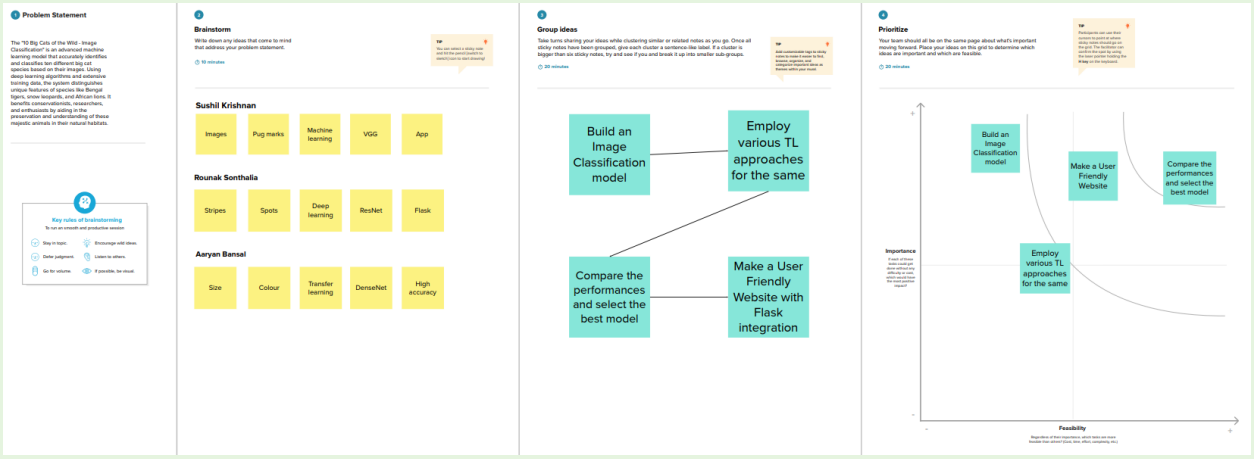
3. IDEATION & PROPOSED SOLUTION

Solution description	Developing a tool employing transfer learning approaches, that can accurately identify and classify ten big cat species from their images, despite the challenges of natural variation in appearance, pose, and lighting conditions.
Novelty	The novelty of the project lies in its use of transfer learning to develop an accurate and efficient model for classifying big cat species from their images. Transfer learning is a relatively new technique in machine learning. The project is also unique in its focus on big cats. Big cats are a challenging group of animals to classify, due to their natural variation in appearance, pose, and lighting conditions.

3.1 Empathy Map



3.2 Ideation & Brainstorming



4. REQUIREMENT ANALYSIS

4.1 Functional requirement

- Image Upload Feature: Users should be able to upload images of wild big cats through the system's interface.
- Big Cat Classification Model: The system must incorporate an advanced machine learning model capable of accurately identifying and classifying ten different big cat species.
- Deep Learning Algorithms: The model should utilize deep learning algorithms for image recognition and feature extraction.
- Extensive Training Data: The system must be trained on a diverse and extensive dataset containing images of various poses, lighting conditions, and backgrounds for each of the ten big cat species.
- User-Friendly Interface: The system should have an intuitive and user-friendly interface to facilitate easy interaction.
- Accurate Classification: The system must accurately classify the uploaded images into one of the ten big cat species.
- API Integration: The system should support integration with external applications or platforms through APIs, allowing data sharing and collaboration.

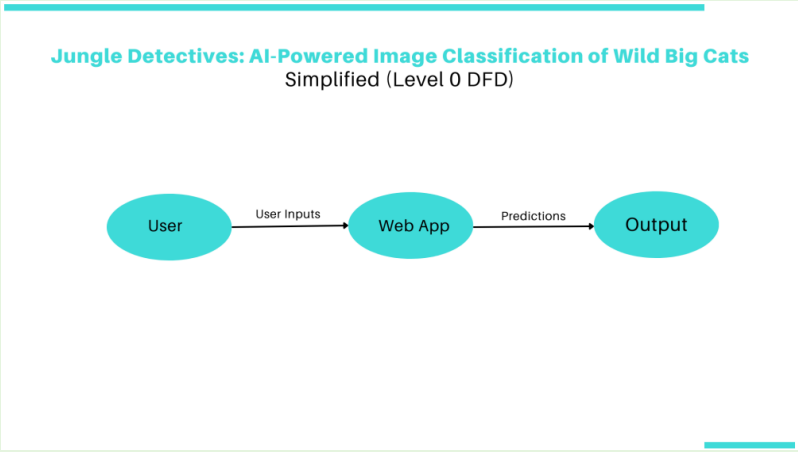
4.2 Non-Functional requirements

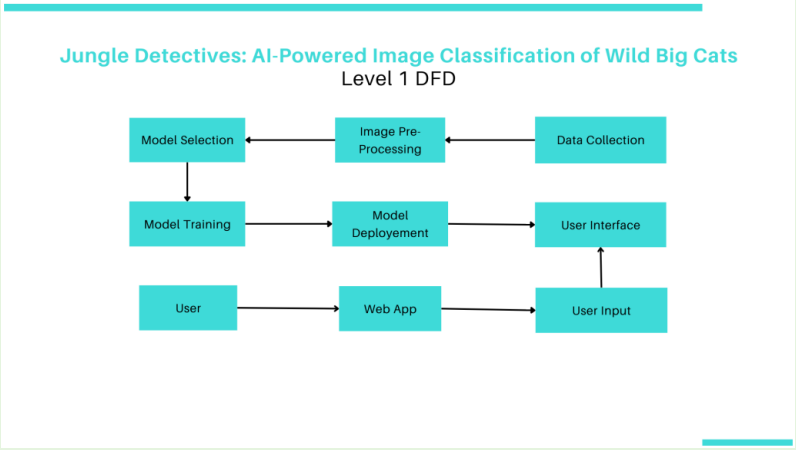
- Response Time: The system should provide classification results within a maximum response time of 5 seconds for uploaded images.
- Scalability: The system should be scalable to accommodate an increasing number of users and a growing dataset without significant degradation in performance.
- Availability: The system should aim for 99.9% availability, ensuring that users can access and utilize the service almost continuously.
- Cross-Browser Compatibility: The user interface should be compatible with major web browsers (Chrome, Firefox, Safari, etc.) to provide a consistent experience.
- Error Handling: Implement robust error handling mechanisms to gracefully

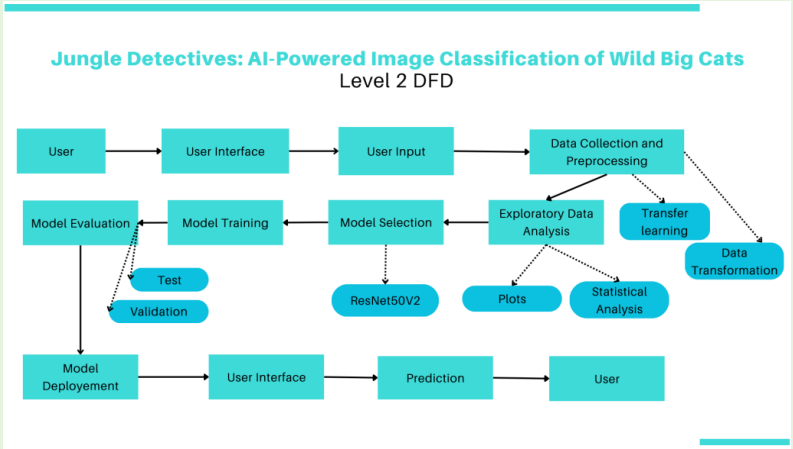
manage unexpected errors and provide informative error messages to users.

5. PROJECT DESIGN

5.1 Data Flow Diagrams & User Stories

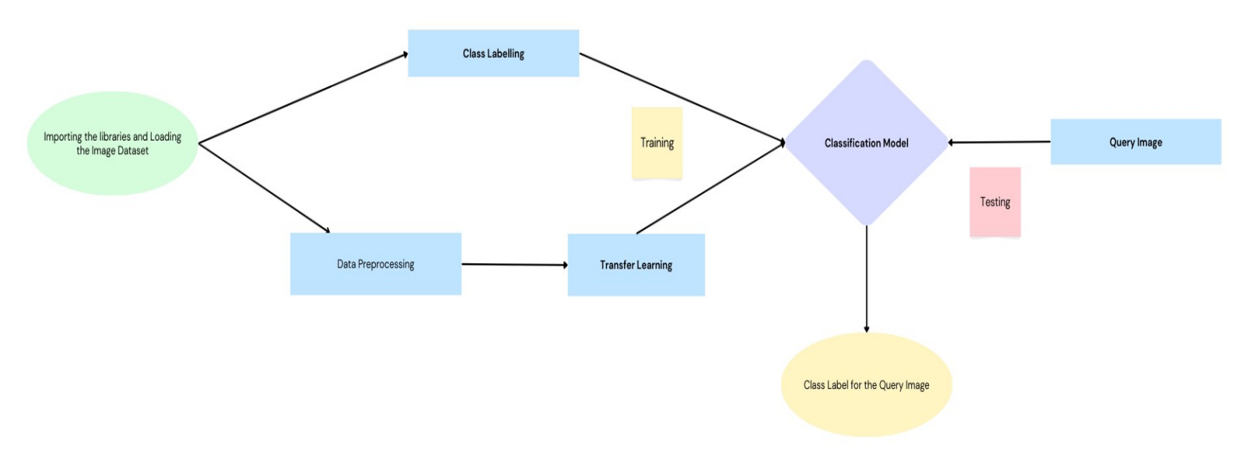






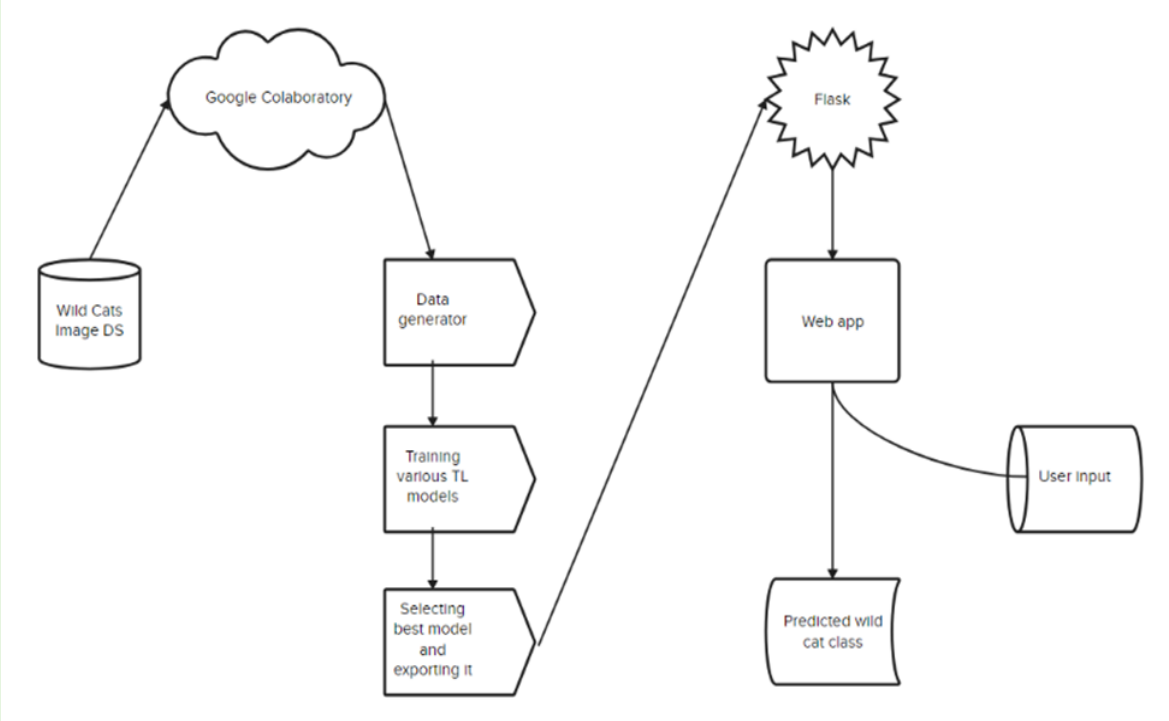
User Type	Functional Requirement (Epic)	User Story Number	User Story/Task	Acceptance Criteria	Priority	Release
Customer (Mobile user)	Prediction System	USN - 1	As a customer, I can utilize a professional predictive system to assess wild cats by inputting my key details.	I am presented with a user-friendly interface to input my details as image.	High	Sprint- 1
Customer (Web user)	Prediction System	USN - 2	As a customer, I want to have the ability to evaluate my capacity to obtain a clear image of wild cats.	I can obtain precise predictions	High	Sprint- 1
Customer (Web User)	Website	USN - 3	As a customer, I can enter the inputs in a very friendly website.	I can access the website	High	Sprint- 1

5.2 Solution Architecture



6. PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture



6.2 Sprint Planning & Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story/ Task	Story Points	Priority	Team Members
Sprint-1	Data Collection and Preparation	USN-1	Collecting big cats' images with appropriate class.	10	High	Aaryan, Rounak
Sprint-2	Model selection	USN-2	Training various TL models on the dataset	15	High	Sushil, Aaryan, Rounak
Sprint-3	Performance analysis	USN-3	Comparing the performance metrics of all the models used and selecting the best.	5	High	Sushil, Rounak
Sprint-4	Saving the model	USN-4	Saving the weights of best performing model to use in final app.	5	Medium	Sushil
Sprint-5	Web app	USN-5	Deploying the TL model using a Flask web app.	25	High	Sushil, Aaryan, Rounak

6.3 Sprint Delivery Schedule

Sprint	Total Story Points	Duration	Sprint Start Date	SprintEnd Date (Planned)	Story Points Completed (as on Planned	Sprint Release Date (Actual)

					End Date)	
Sprint-1	10	2 Days	25 Oct 2023	27 Oct 2023	10	27 Oct 2023
Sprint-2	25	3 Days	27 Oct 2023	30 Oct 2023	25	30 Oct 2023
Sprint-3	30	1 Days	30 Oct 2023	31 Oct 2023	30	31 Oct 2023
Sprint-4	35	1 Days	1 Nov 2023	2 Nov 2023	35	2 Nov 2023
Sprint-5	60	5 Days	2 Nov 2023	7 Nov 2023	60	7 Nov 2023

Velocity:

Team's averagevelocity (AV) per iteration unit (story points per day):

Total story points completed

= Sprint 1 + Sprint 2 + Sprint 3 + Sprint 4 + Sprint 5

= 60

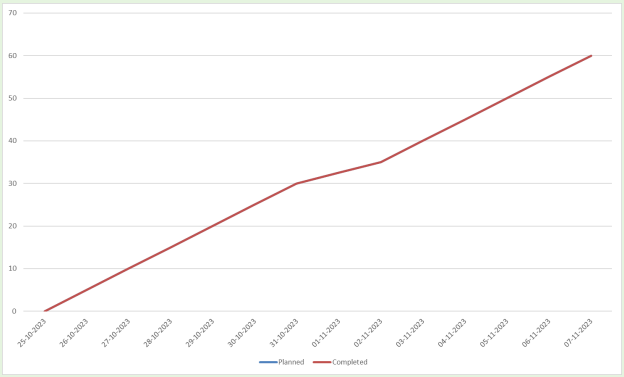
Total duration

= Sprint 1 + Sprint 2 + Sprint 3 + Sprint 4 + Sprint 5

= 12

$$AV = \frac{\text{Total Story Points Completed}}{\text{Total Duration}} = \frac{60}{12} = 5$$

Burndown Chart:



7. CODING & SOLUTIONING (Explain the features added in the project along with code)

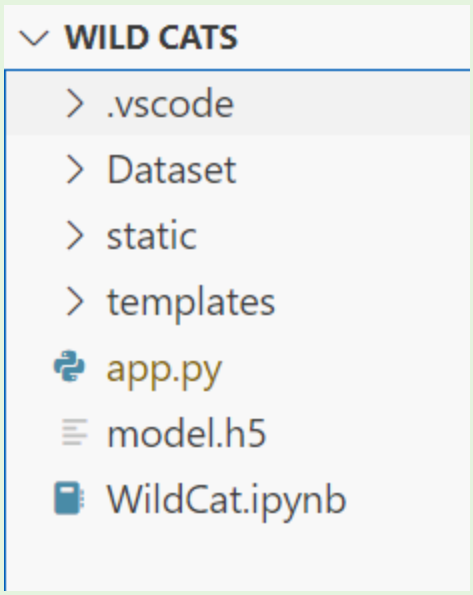
To accomplish objectives, the following tasks are required:

- Data Collection
  - Create Train and Test Folders
- Data Pre-processing
  - Import the ImageDataGenerator library
- Configure ImageDataGenerator class
  - Apply ImageDataGenerator functionality to Train Set, Test set and Valid set
- Model Building
  - Import the model building Libraries
  - Initializing the model
  - Adding Input Layer
  - Adding Hidden Layer
  - Adding Output Layer
  - Configure the Learning Process
- Training and testing the model
  - Comparing performances of various models and selecting the best model
- Save the Model
- Application Building using Flask
  - Create an HTML file
  - Build Python Code

Project Structure

Overall project folder will contain the files and folders are shown in the below image.

- The Dataset folder contains the training, testing and validation images for training our model.
- The python notebook WildCat.ipynb was used for training the models and exporting the best model. The best model's weight has been exported as model.h5.
- We are building a Flask Application that needs HTML pages stored in the templates folder and a python script app.py for server-side scripting.
- Static folder is for storing the images uploaded for prediction by users.



Milestone 1 - Collection of Data

The dataset contains different images of big wild cats. It is downloaded from Kaggle data repository. There are three folders for train, test and validation separately. Within each folder, there are 10 different folders representing 10 classes of big wild cats. The given dataset has 10 different types of big cats, they are following:

- Cheetah
- Lions
- Snow Leopard
- Caracal
- Tiger
- Clouded Leopard
- Puma
- Jaguar
- Ocelot

- African Leopard

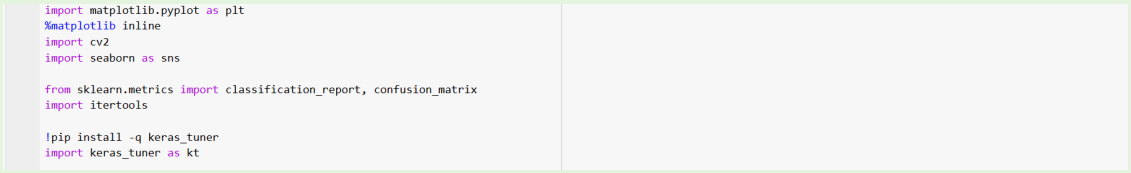
Link to download the dataset: 10 Big Cats of the Wild - Image Classification | Kaggle

Milestone 2 – Model building

The dataset is loaded and Data generator will be used. Some geometric transformations of images like rescaling, rotating, width shifting, height shifting, horizontal flipping is performed. The following actions are required:

1. Import the ImageDataGenerator library
    - Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset.
    - The Keras deep learning neural network library provides the capability to fit models using image data augmentation via the ImageDataGenerator class. Let us import the ImageDataGenerator class from TensorFlow Keras

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tqdm import tqdm
from PIL import Image
from sklearn.model_selection import train_test_split

from tensorflow import keras
from tensorflow.keras import layers
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import Callback, EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers.experimental import preprocessing

from tensorflow.keras.applications import ResNet50V2
from tensorflow.keras.applications import ResNet152V2
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.applications import Xception
from tensorflow.keras.applications import MobileNetV2

from pathlib import Path
import os.path
```



```
import matplotlib.pyplot as plt
%matplotlib inline
import cv2
import seaborn as sns

from sklearn.metrics import classification_report, confusion_matrix
import itertools

!pip install -q keras_tuner
import keras_tuner as kt
```

2. Configure ImageDataGenerator class
    - ImageDataGenerator class is instantiated and the configuration for the types of data augmentation. This repeated for train, test and validation.

```
[ ]  train_path = '/content/drive/MyDrive/Dataset/train'
     test_path = '/content/drive/MyDrive/Dataset/test'
     valid_path = '/content/drive/MyDrive/Dataset/valid'
```

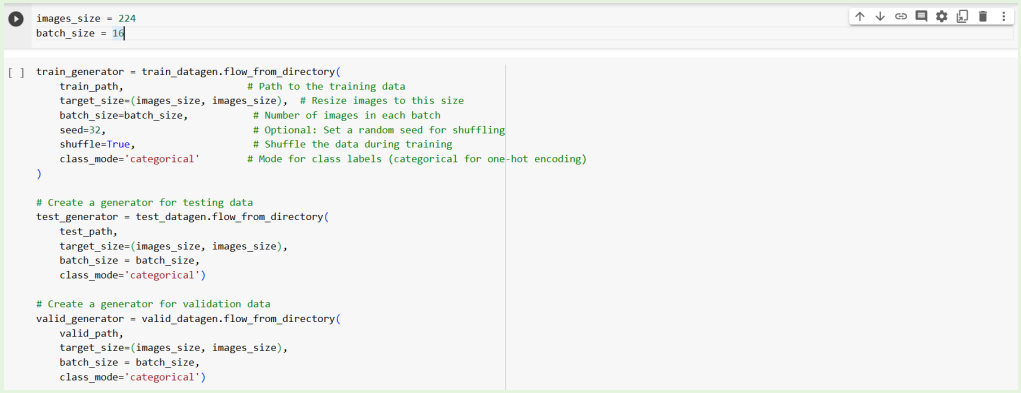
```
[ ]  # Data augmentation for training
     train_datagen = ImageDataGenerator(
         rescale=1.0/255,                # Rescale pixel values to [0, 1]
         rotation_range=20,              # Random rotation within 20 degrees
         width_shift_range=0.2,          # Random horizontal shift by 20% of image width
         height_shift_range=0.2,         # Random vertical shift by 20% of image height
         horizontal_flip=True,           # Random horizontal flipping
         fill_mode='nearest'             # Fill mode for new pixels after shifts/rotations
     )

     # data augmentation for testing
     test_datagen = ImageDataGenerator(rescale=1.0/255)  # Rescale pixel values to [0, 1]

     # data augmentation for validation
     valid_datagen = ImageDataGenerator(rescale=1.0/255)  # Rescale pixel values to [0, 1]
```

- Five main types of data augmentation techniques were used:
  - Image rescaling via rescale.
  - Image rotations via the rotation_range argument.
  - Image shifts via the width_shift_range and height_shift_range arguments.
  - The image flips via the horizontal_flip.
- An instance of the ImageDataGenerator class can be constructed for train, test and validation.

3. Apply ImageDataGenerator functionality to Train, Test and Validation set
    - Let us apply ImageDataGenerator functionality.
    - Arguments:

- directory: Directory where the data is located. If labels are "inferred", it should contain subdirectories, each containing images for a class. Otherwise, the directory structure is ignored.
- batch_size: Size of the batches of data which is 64.
- target_size: Size to resize images after they are read from disk.
- Shuffleing true or false.

- class_mode:
  - ‘int' means that the labels are encoded as integers (e.g., for sparse_categorical_crossentropy loss).
  - 'categorical' means that the labels are encoded as a categorical vector (e.g., for categorical_crossentropy loss).
  - 'binary' means that the labels (there can be only 2) are encoded as float32 scalars with values 0 or 1 (e.g., for binary_crossentropy).
  - None (no labels).



```
images_size = 224
batch_size = 16
```

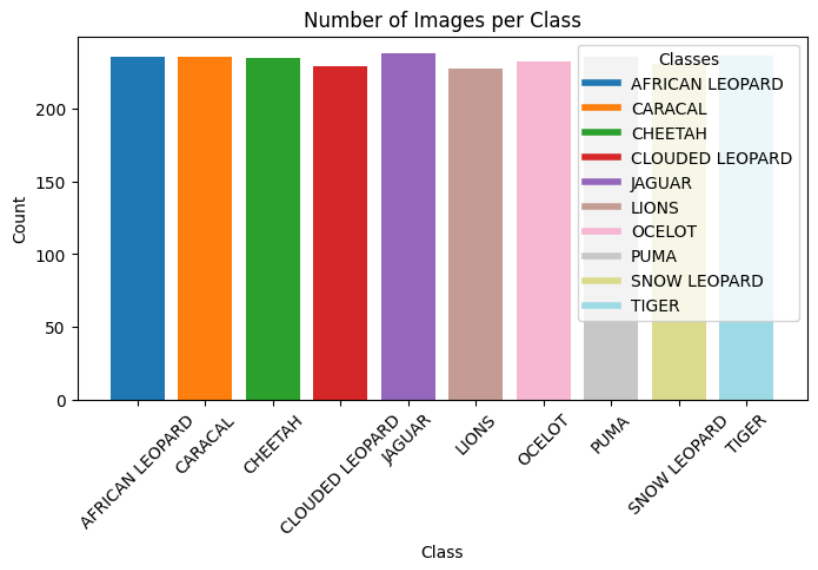
```
train_generator = train_datagen.flow_from_directory(
    train_path,                    # Path to the training data
    target_size=(images_size, images_size),  # Resize images to this size
    batch_size=batch_size,          # Number of images in each batch
    seed=32,                        # Optional: Set a random seed for shuffling
    shuffle=True,                   # Shuffle the data during training
    class_mode='categorical'        # Mode for class labels (categorical for one-hot encoding)
)

# Create a generator for testing data
test_generator = test_datagen.flow_from_directory(
    test_path,
    target_size=(images_size, images_size),
    batch_size = batch_size,
    class_mode='categorical')

# Create a generator for validation data
valid_generator = valid_datagen.flow_from_directory(
    valid_path,
    target_size=(images_size, images_size),
    batch_size = batch_size,
    class_mode='categorical')
```

4. Model Selection and training
   - A subset of 1000 images are taken to train 5 different transfer learning models to find which one is better in performance.
   - The following tasks are required:
     1. Importing the Model Building Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tqdm import tqdm
from PIL import Image
from sklearn.model_selection import train_test_split

from tensorflow import keras
from tensorflow.keras import layers
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import Callback, EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers.experimental import preprocessing

from tensorflow.keras.applications import ResNet50V2
from tensorflow.keras.applications import ResNet152V2
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.applications import Xception
from tensorflow.keras.applications import MobileNetV2

from pathlib import Path
import os.path
```

     2. Exploratory Data Analysis

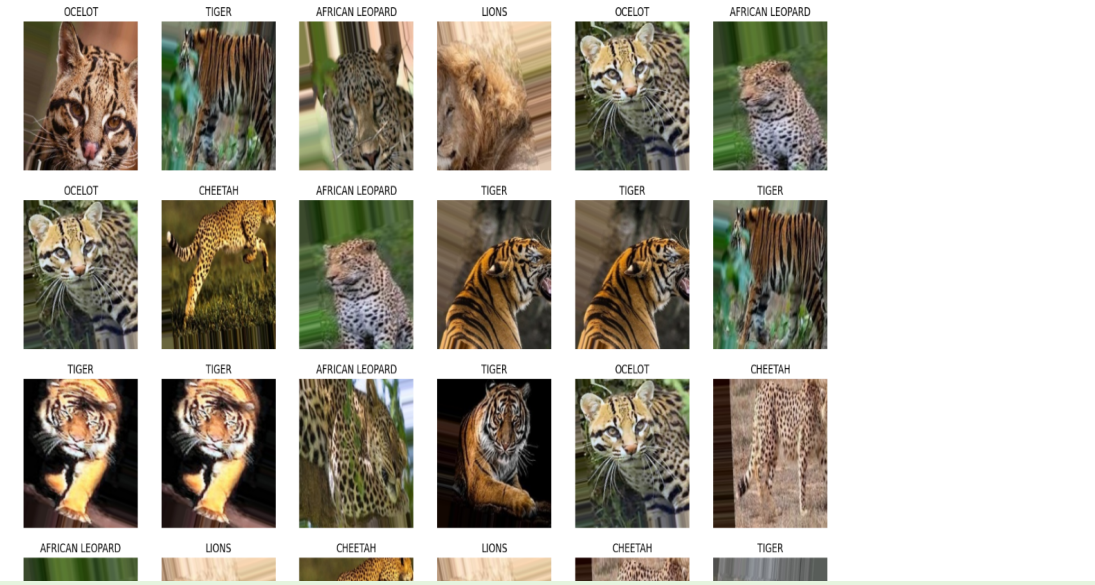
```python
# Get the class labels
class_labels = list(train_generator.class_indices.keys())
# Calculate the count of images per class
class_counts = {label: 0 for label in class_labels}

for i in range(len(train_generator)):
    batch_data, batch_labels = train_generator[i]
    for j in range(len(batch_data)):
        class_idx = int(batch_labels[j].argmax())
        class_label = class_labels[class_idx]
        class_counts[class_label] += 1

# Define unique colors for each class
class_colors = plt.cm.tab20(np.linspace(0, 1, len(class_labels)))
# Create a bar chart with different colors for each class
plt.figure(figsize=(8, 4))
bars = plt.bar(class_counts.keys(), class_counts.values(), color=class_colors)
plt.xlabel('Class')
plt.ylabel('Count')
plt.title('Number of Images per Class')
plt.xticks(rotation=45)
# Add a legend for class colors
legend_labels = [plt.Line2D([0], [0], color=class_colors[i], lw=4, label=class_labels[i]) for i in range(len(class_labels))]
plt.legend(handles=legend_labels, title="Classes")
plt.show()
```



3. Visualizing the images:

```python
[ ] def Show_Images(target_gen):
        # Get a batch of images and labels
        batch_images, batch_labels = next(target_gen)

        # Get class labels
        class_labels = list(target_gen.class_indices.keys())

        # Display images with labels
        plt.figure(figsize=(20, 20))
        for n , i in enumerate(list(np.random.randint(0,len(batch_images),36))):
            plt.subplot(6, 6, n + 1)
            plt.imshow(batch_images[i])
            plt.title(class_labels[np.argmax(batch_labels[i])])  # Display the class label
            plt.axis('off')
        plt.show()
```

```python
[ ] Show_Images(train_generator)
```

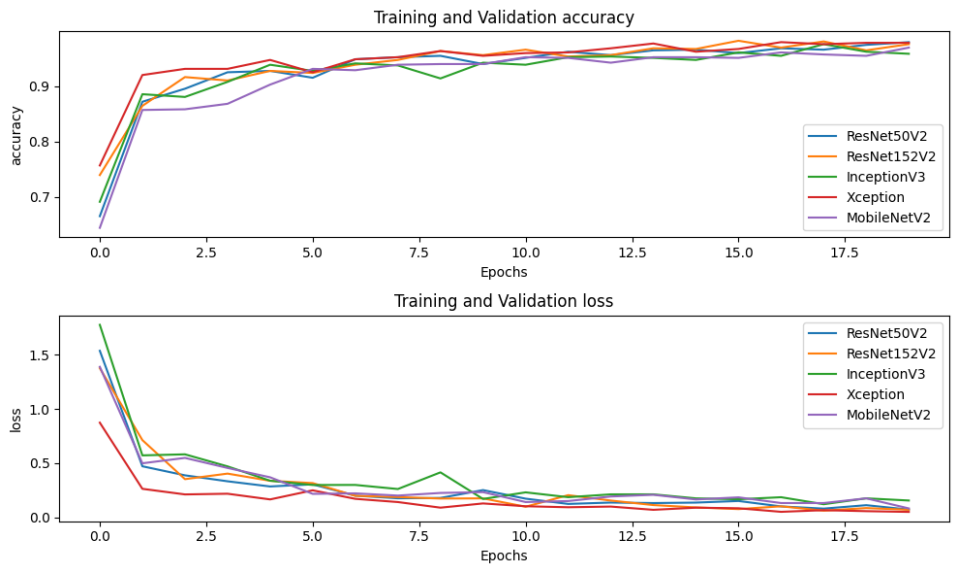


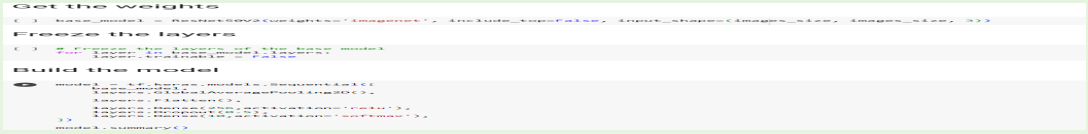
4. Initializing the models

```
[ ]  # Collect all TL models
     TL_Models =[
         ResNet50V2(input_shape=(images_size, images_size, 3), weights='imagenet', include_top=False),
         ResNet152V2(input_shape=(images_size, images_size, 3), weights='imagenet', include_top=False),
         InceptionV3(input_shape=(images_size, images_size, 3), weights='imagenet', include_top=False),
         Xception(input_shape=(images_size, images_size, 3), weights='imagenet', include_top=False),
         MobileNetV2(input_shape=(images_size, images_size, 3), weights='imagenet', include_top=False),
     ]

     # Define all the TL models names. This will be later used during visualization
     TL_Models_NAMES = [
         'ResNet50V2',
         'ResNet152V2',
         'InceptionV3',
         'Xception',
         'MobileNetV2',
     ]

     # Freeze the weights of all the TL models
     for tl_model in TL_Models:
         tl_model.trainable = False
```

5. Training all and comparing. Saving weights of best model.

```
[ ] # Initialize an empty list to hold the histories of each TL_models architecture.
    HISTORIES = []

    # Loop over every backbone in the BACKBONES list.
    for tl_model in tqdm(TL_Models, desc="Training Tl Models"):

        # Create the simplest model architecture using the current backbone.
        model = keras.Sequential([
            tl_model,
            layers.GlobalAveragePooling2D(),
            layers.Dropout(0.5),
            layers.Dense(10, activation='softmax')
        ])

        # Compile the model with the specified loss function, optimizer, and metrics.
        model.compile(
            loss='categorical_crossentropy',
            optimizer=Adam(learning_rate = learning_rate_schedule),
            metrics='accuracy'
        )

        # Train the model on a subset of the training data.
        history = model.fit(
            X_sub, y_sub,
            epochs=20,
            validation_split=0.2,
            batch_size=batch_size
        )

        # Store the history of the trained model.
        HISTORIES.append(history.history)
```





6. Training the model

- The better performing model among five is ResNet50V2.
- It is selected for further training.

- Extra layers are added and model is compiled with required metrics.

```
[ ] model = tf.keras.models.Sequential([
        base_model,
        layers.GlobalAveragePooling2D(),

        layers.Flatten(),

        layers.Dense(256,activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(10,activation='softmax'),
    ])

    model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50v2 (Functional)     (None, 7, 7, 2048)        23564800

 global_average_pooling2d_5  (None, 2048)              0
  (GlobalAveragePooling2D)

 flatten (Flatten)           (None, 2048)              0

 dense_5 (Dense)             (None, 256)               524544

 dropout_5 (Dropout)         (None, 256)               0

 dense_6 (Dense)             (None, 10)                2570

=================================================================
Total params: 24091914 (91.90 MB)
Trainable params: 527114 (2.01 MB)
Non-trainable params: 23564800 (89.89 MB)
```

- adam optimizer is used.

```
[ ] from tensorflow.keras import optimizers
    optimizer = optimizers.Adam(learning_rate=learning_rate_schedule)

[ ] model.compile(optimizer=optimizer,
                  loss="categorical_crossentropy",
                  metrics=['accuracy']
                 )
```

Training the model

```
[ ] history = model.fit(
        train_generator,
        steps_per_epoch=train_generator.samples // batch_size,
        epochs=20,
        validation_data=valid_generator,
        batch_size=batch_size,
        validation_steps=valid_generator.samples // batch_size,
        callbacks=[callback]
    )
```

- The model is trained for 20 epochs. Arguments:
    - steps_per_epoch: it specifies the total number of steps taken from the generator as soon as one epoch is finished and the next epoch has started. We can calculate the value of steps_per_epoch as the total number of samples in your dataset divided by the batch size.
    - Epochs: an integer and number of epochs we want to train our model for.
    - validation_data can be either: an inputs and targets list a generator an inputs, targets, and sample_weights list which can be used to evaluate the loss and metrics for any model after any epoch has ended.

- validation_steps: only if the validation_data is a generator then only this argument can be used. It specifies the total number of steps taken from the generator before it is stopped at every epoch and its value is calculated as the total number of validation data points in your dataset divided by the validation batch size.

```
[ ]  from tensorflow.keras import optimizers
     optimizer = optimizers.Adam(learning_rate=learning_rate_schedule)

[ ]  model.compile(optimizer=optimizer,
                   loss="categorical_crossentropy",
                   metrics=['accuracy']
                   )
```

Training the model

```
[ ]  history = model.fit(
         train_generator,
         steps_per_epoch=train_generator.samples // batch_size,
         epochs=20,
         validation_data=valid_generator,
         batch_size=batch_size,
         validation_steps=valid_generator.samples // batch_size,
         callbacks=[callback]
     )
```

  - Analyzing performance metrics

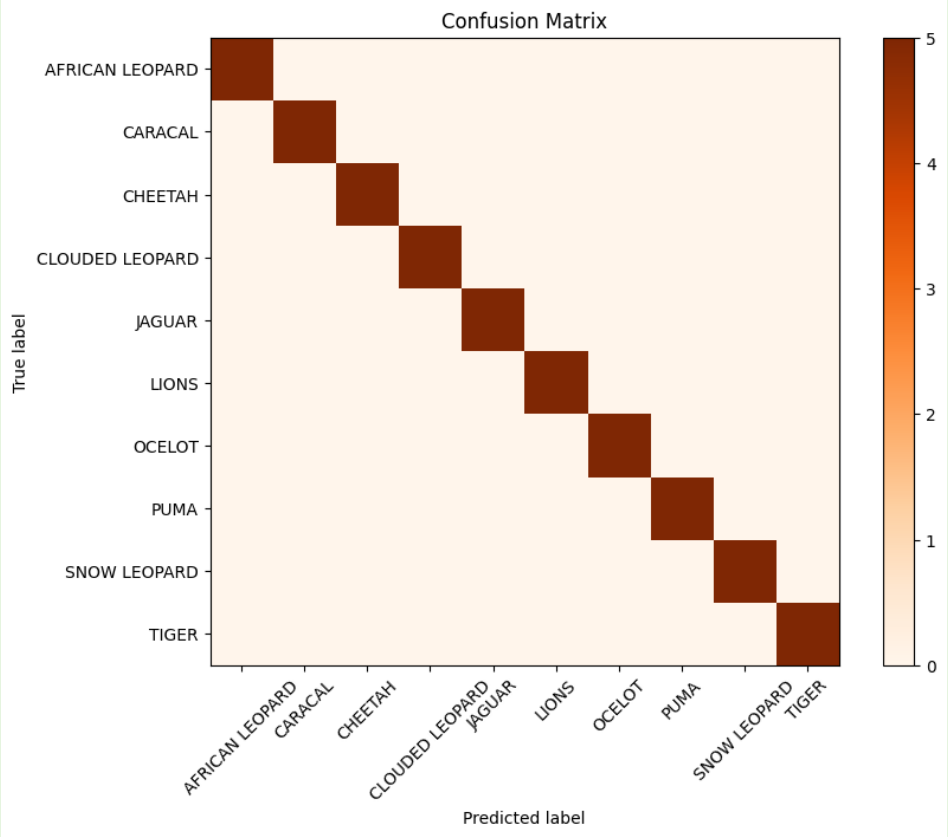
ii. Evaluating on test data.

```
[ ]  # Evaluate on test dataset
     score = model.evaluate(test_generator, verbose=False)
     print('Test loss:', score[0])
     print('Test accuracy:', score[1])
```

```
Test loss: 0.02155800350010395
Test accuracy: 1.0
```

iii. Observing the conclusion matrix to see how well the model is able to classify correctly. All the images have been correctly classified.

```
[ ]  true_labels = []
     predicted_labels = []

     num_batches = len(test_generator)
     for i in range(num_batches):
         x_batch, y_batch = test_generator[i]
         predictions = model.predict(x_batch)
         true_labels.extend(np.argmax(y_batch, axis=1))  # Convert one-hot encoded labels to class indices
         predicted_labels.extend(np.argmax(predictions, axis=1))


     class_names = test_generator.class_indices.keys()
     # Calculate the confusion matrix
     confusion = confusion_matrix(true_labels, predicted_labels)

     plt.figure(figsize=(9, 7))
     # Create a visualization of the confusion matrix
     classes = [str(i) for i in range(len(test_generator.class_indices))]
     plt.imshow(confusion, interpolation='nearest', cmap=plt.get_cmap('Oranges'))
     plt.title('Confusion Matrix')
     plt.colorbar()
     tick_marks = np.arange(len(class_names))
     plt.xticks(tick_marks, class_names, rotation=45)
     plt.yticks(tick_marks, class_names)

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



iv. Studying the classification report. The model is performing highly precise for the test data.

```
# Calculate the classification report
report = classification_report(true_labels, predicted_labels, target_names=test_generator.class_indices)

# Print the classification report
print(report)
```

```
                precision    recall  f1-score   support

AFRICAN LEOPARD       1.00      1.00      1.00         5
        CARACAL       1.00      1.00      1.00         5
       CHEETAH       1.00      1.00      1.00         5
CLOUDED LEOPARD       1.00      1.00      1.00         5
        JAGUAR       1.00      1.00      1.00         5
         LIONS       1.00      1.00      1.00         5
        OCELOT       1.00      1.00      1.00         5
          PUMA       1.00      1.00      1.00         5
   SNOW LEOPARD       1.00      1.00      1.00         5
         TIGER       1.00      1.00      1.00         5

      accuracy                           1.00        50
     macro avg       1.00      1.00      1.00        50
  weighted avg       1.00      1.00      1.00        50
```

- Testing some samples

```python
# Collect true labels and model predictions
true_labels = []
predicted_labels = []
class_names = test_generator.class_indices.keys()
class_names = list(class_names)
num_batches = len(train_generator)
for i in range(10):
    x_batch, y_batch = train_generator[i]
    predictions = model.predict(x_batch)
    true_labels.extend(np.argmax(y_batch, axis=1))  # Convert one-hot encoded labels to class indices
    predicted_labels.extend(np.argmax(predictions, axis=1))


true_class_labels = [class_names[i] for i in true_labels]
predicted_class_labels = [class_names[i] for i in predicted_labels]

# Plot true labels and predicted labels
plt.figure(figsize=(10, 5))
num_samples_to_display = min(10, len(x_batch))  # Display up to 10 samples or less if available
for i in range(num_samples_to_display):
    plt.subplot(2, 5, i + 1)
    plt.imshow(x_batch[i])
    plt.title(f'True: {true_class_labels[i]}\nPredicted: {predicted_class_labels[i]}', fontsize=8)
    plt.axis('off')

plt.tight_layout()
plt.show()
```

- Saving the model in appropriate format to integrate using Flask

```
[ ] import pickle
    pickle.dump(model, open('model.pkl', 'wb'))

[ ] model.save('model.h5')

    /usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend usi
      saving_api.save_model(
```

Milestone 3 – Application Building

- Now that we have trained our model, let us build our flask application which will be running in our local browser with a user interface.

- In the flask application, the input parameters are taken from the HTML page These factors are then given to the model to know to predict the type of wild cat and showcase on the HTML page to notify the user.

a. Create HTML Pages

- We use HTML to create the front-end part of the web page.
- We also use Java Script and CSS to enhance our functionality and view of HTML pages.

b. Create app.py (Python Flask) file

app.py > ...

```python
from flask import Flask, render_template, request
from keras.models import load_model
from keras.preprocessing import image
import numpy as np
app = Flask(__name__)
cats = ["AFRICAN LEOPARD", "CARACAL", "CHEETAH", "CLOUDED LEOPARD", "JAGUAR", "LION", "OCELOT", "PUMA", "SNOW LEOPARD", "TIGER"]
model = load_model('model.h5')
def predict_label(img_path):
    i = image.load_img(img_path, target_size=(224,224))
    i = image.img_to_array(i)/255.0
    i = i.reshape(1,224,224,3)
    p = model.predict(i)
    l = np.argmax(p, axis=1)
    return cats[l[0]]
@app.route("/", methods=['GET', 'POST'])
def main():
    return render_template("index.html")
@app.route("/submit", methods = ['GET', 'POST'])
def get_output():
    if request.method == 'POST':
        img = request.files['image']
        img_path = "static/" + img.filename
        img.save(img_path)
        p = predict_label(img_path)
    return render_template("index.html", prediction = p, img_path = img_path)
if __name__ =='__main__':
    app.run(debug = True)
```

8. PERFORMANCE TESTING

8.1 Performance Metrics

Model Summary

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50v2 (Functional)     (None, 7, 7, 2048)        23564800

 global_average_pooling2d_5  (None, 2048)              0
  (GlobalAveragePooling2D)

 flatten (Flatten)           (None, 2048)              0

 dense_5 (Dense)             (None, 256)               524544

 dropout_5 (Dropout)         (None, 256)               0

 dense_6 (Dense)             (None, 10)                2570

=================================================================
Total params: 24091914 (91.90 MB)
Trainable params: 527114 (2.01 MB)
Non-trainable params: 23564800 (89.89 MB)
_________________________________________________________________
```

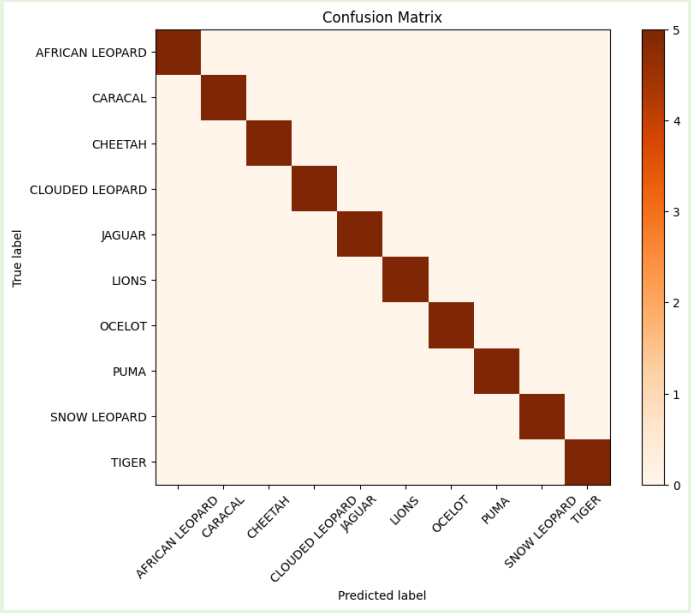
Accuracy

Training Accuracy – 91%

Validation Accuracy – 95%

```
Epoch 17/20
146/146 [==============================] - ETA: 0s - loss: 0.2753 - accuracy: 0.9100WARNING:tensorflow:Learning rate reduction is conditioned on m
146/146 [==============================] - 38s 261ms/step - loss: 0.2753 - accuracy: 0.9100 - val_loss: 0.1466 - val_accuracy: 0.9583 - lr: 0.0018
```

Confusion Matrix



Test Accuracy Scores

```
Test loss: 0.02155800350010395
Test accuracy: 1.0
```

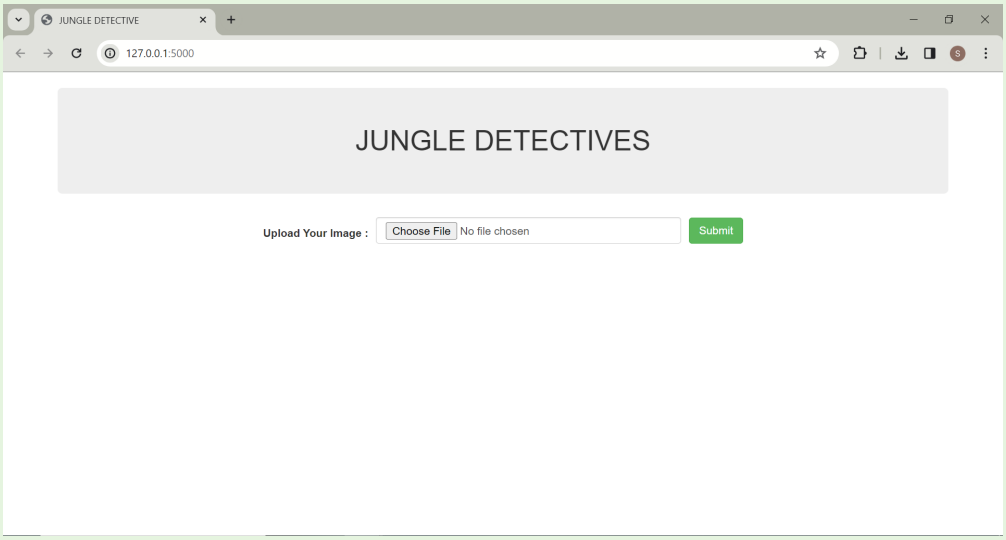
Classification Report

	precision	recall	f1-score	support
AFRICAN LEOPARD	1.00	1.00	1.00	5
CARACAL	1.00	1.00	1.00	5
CHEETAH	1.00	1.00	1.00	5
CLOUDED LEOPARD	1.00	1.00	1.00	5
JAGUAR	1.00	1.00	1.00	5
LIONS	1.00	1.00	1.00	5
OCELOT	1.00	1.00	1.00	5
PUMA	1.00	1.00	1.00	5
SNOW LEOPARD	1.00	1.00	1.00	5
TIGER	1.00	1.00	1.00	5
accuracy			1.00	50
macro avg	1.00	1.00	1.00	50
weighted avg	1.00	1.00	1.00	50

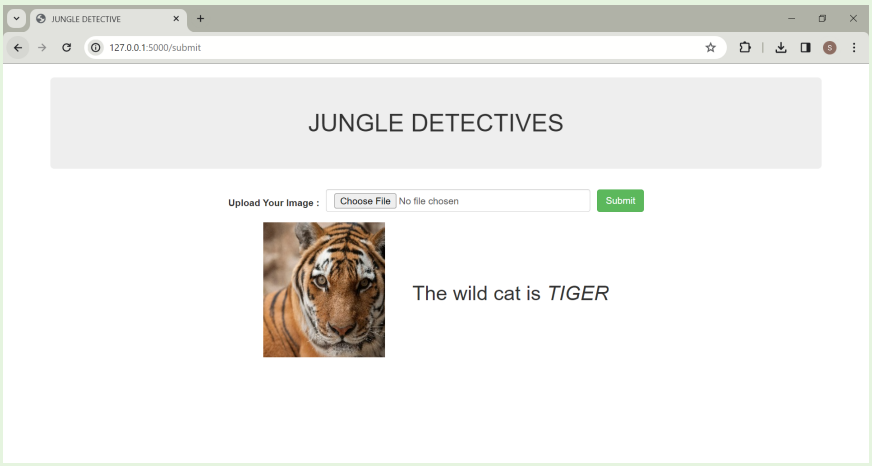
9. RESULTS

9.1 Output Screenshots

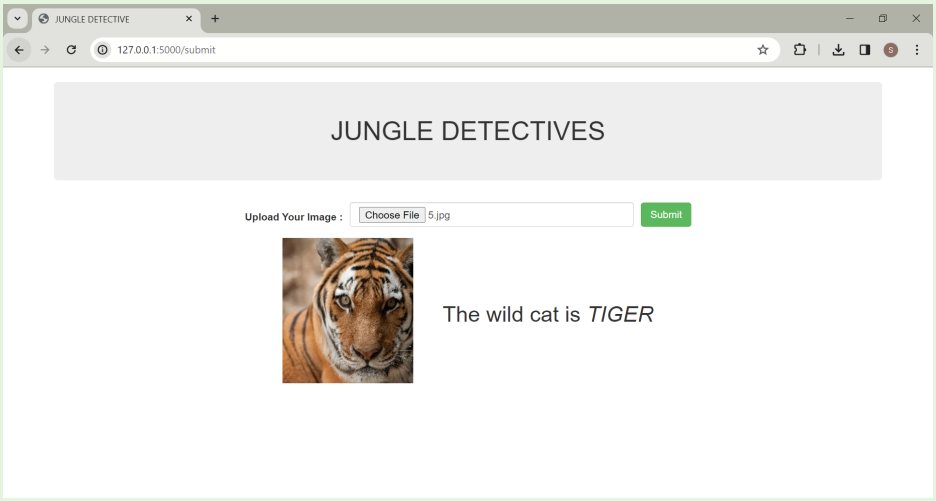
Initial page.



Output for an image submitted.



Submitting another image.



Output for second submitted image.

10. ADVANTAGES & DISADVANTAGES

Advantages:

1. Accurate Identification: The AI-powered image classification system provides accurate identification of ten different big cat species, aiding researchers, and conservationists in their work.
2. Efficient Monitoring: The technology enables efficient monitoring of big cat populations, allowing for better understanding of their behaviour, distribution, and overall well-being in their natural habitats.
3. Time-Saving: Automated image classification through AI is much faster than manual identification, saving valuable time for researchers and allowing them to process a large volume of data more quickly.
4. Conservation Impact: The project contributes directly to wildlife conservation efforts by providing a tool that helps in the preservation and protection of big cat species, which may face threats such as poaching and habitat loss.
5. Educational Tool: The project serves as an educational tool for enthusiasts, raising awareness about big cat species and their importance in the ecosystem.

Disadvantages:

1. Data Bias: If the training data used for the model is biased, the system may not perform well in recognizing variations within each species or might struggle with images from different geographical locations.
2. Limited to Trained Species: The model may not be applicable to identify other species beyond the ten big cat species it was trained on, limiting its broader ecological applications.
3. Dependency on Image Quality: The accuracy of the system could be influenced by the quality of input images. Low-resolution or unclear images may lead to misclassifications.
4. Maintenance and Updates: The need for continuous maintenance and updates to the model to ensure its accuracy over time may pose a challenge, especially if there are changes in the populations or habitats of the big cat species.
5. Resource Intensive: Developing and maintaining such advanced AI models can

be resource-intensive, requiring significant computational power, storage, and expertise.

11. CONCLUSION

Hence, the primary goal of this project to develop a robust image classification system capable of accurately identifying and categorizing different types of big cats from large datasets of images has been achieved. The system leverages transfer learning approaches, that can accurately identify and classify ten big cat species from their images, despite the challenges of natural variation in appearance, pose, and lighting conditions.

12. FUTURE SCOPE

- Incorporating a larger dataset.
- Expanding the projects for other wild life.
- Developing mobile application .

13. APPENDIX

GitHub Link: https://github.com/smartinternz02/SI-GuidedProject-609379-1698293666

Project Demo Link: https://drive.google.com/file/d/1tkuFVZM7r_8bB0m42p0xf4N-eaDN0Ca8/view?usp=sharing