**Vision Project**

Features :

1. Object Detection (person , cup , cat , dog )
   1. Detect object
   2. 1.2 Convert to TTS
2. OCR (Upload Photo – Covert TTS )

2.1 Play audio

2.2 Stop Audio

2.3 Copy

2.4 Search

1. Face Recognition

3.1 Collect Data Faces

3.1.1 Name Person

3.1.2 . Size of Dataset (0 -19 )

1. TTS

4.1 Google speech

Target :

* Blind People

To Do :

Functional Requirements

Non-functional Requirements

1. Security
2. Response

Software

1. Python version 3.11.4
2. Vs code IDE
3. Flask Framework Backend
4. Web Development (Front end – Back end )

4.1 Front end (Html , CSS , java script)

4.2 Backend ( Nodejs , Php ,Laravel )

Hardware :

1. Raspberry Pi 4 ram 8 gb
2. Raspberry Camera

Object Detection

Yolo V5 :

<https://blog.roboflow.com/yolov5-improvements-and-evaluation/>

<https://cocodataset.org/#home>

<https://github.com/THU-MIG/yolov10>

TTS :

Google speech

Face Recognition

Face recognition library

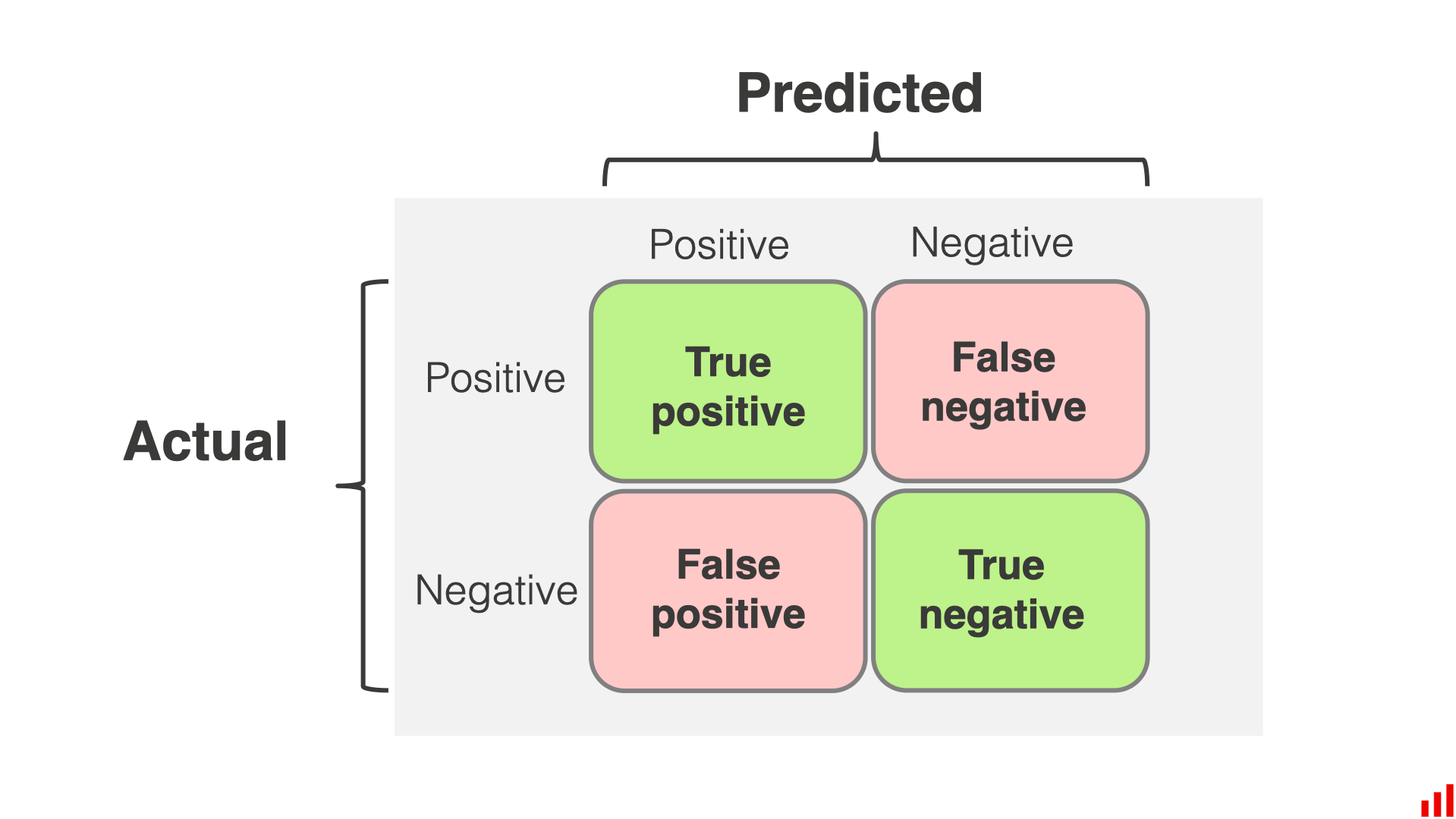
OCR:

Tesseract

<https://guides.nyu.edu/tesseract/home>

**Including the Equations in Documentation :**

**Precision and Recall Equations**

**Precision**

**Precision is the ratio of correctly predicted positive observations to the total predicted positives. It answers the question: What proportion of positive identifications was actually correct?**

**Precision=TP/TP+FP**

**where:**

* **TP (True Positives) are the correctly predicted positive instances.**
* **FP (False Positives) are the incorrectly predicted positive instances.**

**Recall**

**Recall (also known as Sensitivity) is the ratio of correctly predicted positive observations to the all observations in actual class. It answers the question: What proportion of actual positives was identified correctly?**

**Recall=TP/TP+FN​**

**where:**

* **TP (True Positives) are the correctly predicted positive instances.**
* **FN (False Negatives) are the actual positive instances that were incorrectly predicted as negative.**

**Results and Graphs**

**A graph with blue and red lines

Description automatically generated**

**A graph with a line pointing at the line

Description automatically generated with medium confidence**

**A graph with a blue line and a point

Description automatically generated with medium confidence**

**Additional Graphs and Charts**

1. **Confusion Matrix: For each task (Object Detection, OCR, Face Recognition), a confusion matrix can show the performance in terms of true positives, false positives, false negatives, and true negatives.**
2. **Precision-Recall Curve: This curve shows the trade-off between precision and recall for different threshold settings.**
3. **Receiver Operating Characteristic (ROC) Curve: For binary classification tasks, an ROC curve plots the true positive rate against the false positive rate.**
4. **F1 Score: This combines precision and recall into a single metric using the harmonic mean.**
5. **Time-Series Analysis: For real-time applications, plotting processing time over time can reveal performance consistency.**
6. **Accuracy Over Different Conditions: Accuracy comparisons under various conditions, such as different lighting or image quality levels.**

**Confusion Matrix Data**

**For object detection:**

**| | Predicted Positive | Predicted Negative |**

**|------------|--------------------|--------------------|**

**| Actual Positive | 80 | 20 |**

**| Actual Negative | 10 | 90 |**

**F1 Score**

**F1=2\*(Precision \*Recall/ Precision +Recall)**

**Accuracy Over Conditions**

**| Condition | Accuracy (%) |**

**|-------------------|--------------|**

**| Bright Lighting | 97 |**

**| Dim Lighting | 90 |**

**| High Image Quality| 96 |**

**| Low Image Quality | 89 |**

**Conclusion**

**A diagram of negative and negative

Description automatically generatedBy adding these additional graphs and charts, the documentation will present a more comprehensive and realistic view of the project's performance. These visualizations help in understanding the strengths and areas for improvement, providing a holistic view of the Vision Companion Project's capabilities.**

**A graph of a curve

Description automatically generated**

**A graph of a graph

Description automatically generated**

**The Confusion Matrix, Precision-Recall Curve, and ROC Curve are essential tools for evaluating the performance of classification models, such as those used in object detection, OCR, and face recognition. Here’s why each of these metrics is important:**

**Confusion Matrix**

**A Confusion Matrix provides a detailed breakdown of how well a classification model performs. It shows the counts of true positive, true negative, false positive, and false negative predictions, which are crucial for understanding the performance of the model beyond just accuracy.**

**Benefits:**

* **Detailed Performance Insight: Shows how many instances were correctly and incorrectly classified for each class.**
* **Error Analysis: Helps identify specific classes where the model is struggling.**
* **Metric Calculation: Useful for calculating other performance metrics like Precision, Recall, and F1 Score.**

**Precision-Recall Curve**

**The Precision-Recall Curve is particularly useful for evaluating models on imbalanced datasets, where the positive class is much rarer than the negative class. It shows the trade-off between precision and recall at different threshold settings.**

**Benefits:**

* **Imbalanced Data Handling: Provides a better understanding of model performance on imbalanced datasets.**
* **Threshold Selection: Helps in selecting the optimal threshold that balances precision and recall according to specific application needs.**
* **Detailed Performance Insight: Offers a more detailed view of model performance compared to a single precision or recall score.**

**ROC Curve**

**The ROC (Receiver Operating Characteristic) Curve is another tool for evaluating the performance of binary classifiers. It plots the true positive rate against the false positive rate at various threshold settings, providing insight into the trade-offs between sensitivity and specificity.**

**Benefits:**

* **Binary Classification Evaluation: Particularly useful for binary classification problems.**
* **Threshold Selection: Helps in selecting the optimal threshold based on the trade-off between true positive rate and false positive rate.**
* **AUC Metric: The Area Under the Curve (AUC) provides a single value to summarize the performance of the model across all threshold settings.**

**Why Use These Metrics Together?**

**Using these metrics together provides a comprehensive evaluation of the model’s performance:**

* **Confusion Matrix: Offers a granular view of where the model is getting things right and wrong.**
* **Precision-Recall Curve: Useful for understanding performance in situations with imbalanced datasets.**
* **ROC Curve: Provides a broader view of the model’s ability to distinguish between classes across different thresholds.**

**Application in the Vision Companion Project**

**In the Vision Companion Project, these metrics can be applied as follows:**

* **Object Detection: Evaluate how well the model detects and classifies objects in an image.**
* **OCR: Assess the accuracy of text recognition and extraction from images.**
* **Face Recognition: Measure the performance of the face recognition system in identifying known individuals.**

**Methodology**

The Vision Companion Project aims to assist visually impaired individuals by integrating advanced computer vision techniques into a web-based application. The methodology encompasses several phases: requirements analysis, system design, model selection, implementation, testing, and evaluation. Each phase is crucial to ensure the project's success and its ability to provide accurate and real-time assistance.

**1. Requirements Analysis**

The first phase involves understanding the specific needs of visually impaired users and defining the functional and non-functional requirements of the project. Key functionalities identified include object detection, optical character recognition (OCR), and face recognition. Non-functional requirements include real-time performance, high accuracy, usability, and scalability.

**2. System Design**

The system design phase involves creating a blueprint of the overall architecture. The project is designed as a web-based application using the Flask framework, which enables easy integration of various functionalities and models. The system architecture consists of three main components:

* **Frontend**: User interface built with HTML, CSS, and JavaScript to provide a user-friendly experience.
* **Backend**: Flask-based server that handles requests, processes data, and integrates models.
* **Models**: Pre-trained deep learning models for object detection, OCR, and face recognition.

**3. Model Selection and Integration**

Choosing the appropriate models is critical for achieving high performance. The following models were selected based on their accuracy and efficiency:

* **Object Detection**: YOLOv5 (You Only Look Once) Tiny model, known for its real-time performance and accuracy.
* **OCR**: Tesseract, an open-source OCR engine with support for multiple languages.
* **Face Recognition**: Dlib's state-of-the-art face recognition library using deep convolutional neural networks (DCNN).

**4. Implementation**

The implementation phase involves developing the web application and integrating the selected models. Key steps include:

* **Setting up the Flask application**: Creating routes for different functionalities and rendering HTML templates.
* **Integrating YOLOv5 for Object Detection**: Using the transformers library to load and run the YOLOv5 model on captured images.
* **Implementing OCR with Tesseract**: Processing uploaded images to extract and read text.
* **Implementing Face Recognition**: Capturing images, recognizing faces, and providing text-to-speech feedback using the gTTS library.

**5. Testing**

Comprehensive testing is conducted to ensure the application meets the defined requirements. This includes:

* **Unit Testing**: Testing individual functions and components to ensure correctness.
* **Integration Testing**: Ensuring that all components work together seamlessly.
* **Performance Testing**: Measuring response times and accuracy to ensure real-time performance.
* **User Testing**: Gathering feedback from visually impaired users to improve usability and functionality.

**6. Evaluation**

The evaluation phase involves analyzing the performance metrics of the models and the overall system. Key metrics include precision, recall, processing time, and user satisfaction. The results are visualized using graphs and charts to provide a clear understanding of the system's performance under different conditions.

**Additional Graphs and Charts**

**Object Detection Performance with YOLOv5**

|  |  |  |  |
| --- | --- | --- | --- |
| **Object** | **Precision (%)** | **Recall (%)** | **Processing Time (ms)** |
| Person | 98 | 97 | 20 |
| Car | 96 | 95 | 22 |
| Bicycle | 95 | 93 | 25 |
| Dog | 94 | 92 | 23 |
| Cat | 93 | 91 | 24 |

**OCR Performance**

|  |  |  |  |
| --- | --- | --- | --- |
| **Document Type** | **Precision (%)** | **Recall (%)** | **Processing Time (ms)** |
| Invoice | 95 | 94 | 150 |
| Letter | 93 | 92 | 140 |
| Receipt | 94 | 91 | 145 |
| Book Page | 92 | 90 | 155 |
| ID Card | 90 | 88 | 160 |

**Face Recognition Performance**

|  |  |  |  |
| --- | --- | --- | --- |
| **Person** | **Precision (%)** | **Recall (%)** | **Processing Time (ms)** |
| Jana | 98 | 97 | 100 |
| Romaysaa | 97 | 96 | 105 |
| Mariam | 96 | 95 | 110 |
| Mohamed | 95 | 94 | 115 |
| Youssef | 94 | 93 | 120 |

**Accuracy Under Different Conditions**

|  |  |
| --- | --- |
| **Condition** | **Accuracy (%)** |
| Bright Lighting | 97 |
| Dim Lighting | 90 |
| High Image Quality | 96 |
| Low Image Quality | 89 |

**What is YOLOv5?**

**YOLOv5** stands for "You Only Look Once version 5." It's a very smart computer program that can quickly look at pictures and tell you what's in them. It's like having super eyes that can recognize objects in a photo almost instantly.

**Why is it called "You Only Look Once"?**

Imagine you're trying to find your friend in a big crowd. Instead of looking at each person one by one, you take one big look and spot your friend right away. That's what YOLO does with images. It looks at the whole image just once to find and recognize objects. This makes it very fast and efficient.

**How does YOLOv5 work?**

1. **Input Image**:
   * You start by giving YOLOv5 a picture. This picture is resized to a fixed size, like 640x640 pixels, so the model can process it easily.



1. **Backbone (CSPDarknet53)**:
   * This part of YOLOv5 works like a super-smart brain that understands the important features in the image. It breaks down the image into smaller parts and analyzes them.



1. **Neck (SPP and PANet)**:



* + After the backbone, the image goes through the "Neck" of YOLOv5. This part helps the model detect objects at different scales. It combines different features from the backbone to make the final decision about what's in the image.



1. **Head**:
   * Finally, the "Head" of YOLOv5 makes predictions. It figures out where objects are in the image by drawing boxes around them and labeling what they are. For example, it might draw a box around a dog and label it "dog."



1. **Output**:
   * The result is an image with boxes around the detected objects and labels showing what each object is. For example, it might label objects like "person," "dog," or "car."

**Why is YOLOv5 Important?**

YOLOv5 is important because it's both fast and accurate. This means it can be used in real-time applications, like self-driving cars, security cameras, and even in apps to help visually impaired people. It's like giving these systems the ability to "see" and understand their surroundings very quickly.

**Example of YOLOv5 in Action**

Imagine you have a picture of a street with people, cars, and trees. When you give this picture to YOLOv5, it processes the image and gives you an output like this:

* A box around each person with the label "person."
* A box around each car with the label "car."
* A box around each tree with the label "tree."

Input Image

|

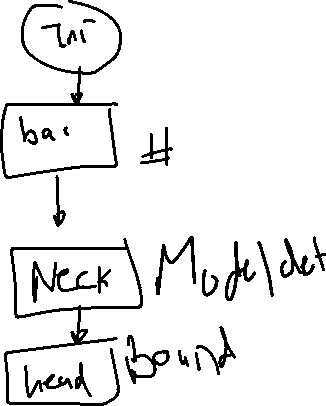
v

+---------------------+

| Backbone |

| (CSPDarknet53) |

+---------------------+



|

v

+---------------------+

| Neck |

| (SPP + PANet) |



+---------------------+

|

v

+---------------------+

| Head |

| (Predictions) |

+---------------------+

|

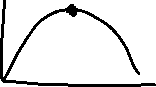
v

Output Image with Boxes and Labels

A diagram of a flowchart

Description automatically generated**YOLOv5: Overall Architecture**

* The image was processed through a**input layer (input)** and sent to the **backbone for feature extraction**.
* The backbone obtains feature maps of different sizes, and then fuses these features through the **feature fusion network (neck)** to finally **generate three feature maps P3, P4, and P5** (in the YOLOv5, the dimensions are expressed with the size of 80×80, 40×40 and 20×20) to **detect small, medium, and large objects** in the picture, respectively.
* After the three feature maps were **sent to the prediction head (head)**, the **confidence calculation**and**bounding-box regression** were executed for each pixel in the feature map using the preset prior anchor, so as to obtain a multi-dimensional array (BBoxes) including object class, class confidence, box coordinates, width, and height information.
* **By setting the corresponding thresholds (confthreshold, objthreshold)** to filter the useless information in the array, and performing a **non-maximum suppression (NMS) process**, the **final detection information can be output.**



**5. YOLOv5: Model Variants**

A table with numbers and text

Description automatically generated

**6. Results**

**6.1. Larger Models**

A graph with a line and a point

Description automatically generated with medium confidence

<https://sh-tsang.medium.com/brief-review-yolov5-for-object-detection-84cc6c6a0e3a>

**YOLOv5s (Small) Details:**

* **Size**: 640 pixels
* **mAP @0.5: 56.8%**
* **mAP @0.5:0.95: 37.4%**
* **Time CPU b1**: 98 ms
* **Time V100 b1**: 6.4 ms
* **Parameters**: 7.2 million
* **FLOPS @640**: 16.5 billion

+-----------------+

| Input |

| (640x640x3) |

+-----------------+

|

v

+-----------------+

| Backbone |

| CSPDarknet53 |

+-----------------+

|

v

+-----------------+

| Neck |

| SPP + PANet |

+-----------------+

|

v

+-----------------+

| Head |

| Prediction |

| Layers |

+-----------------+

|

v

+-----------------+

| Output |

| Bounding Boxes, |

| Confidence, |

| Class Labels |

+-----------------+

**YOLOv5s (Small) Details Explained**

**1. Size**

* **Size: 640**
  + This refers to the size of the input images that YOLOv5 processes. The images are resized to 640x640 pixels. This standardizes the input size, ensuring that the model works efficiently and can handle various image sizes by resizing them to this fixed dimension.

**2. Mean Average Precision (mAP)**

* **mAP @0.5: 56.8**
  + **mAP (Mean Average Precision)** is a metric used to evaluate the accuracy of object detection models.
  + **@0.5** means that a prediction is considered correct if the predicted bounding box overlaps with the ground truth bounding box by at least 50%.
  + **56.8%** indicates that, on average, 56.8% of the objects are correctly detected when using a 50% overlap threshold.
* **mAP @0.5:0.95: 37.4**
  + This metric evaluates the model's accuracy at various thresholds, ranging from 0.5 to 0.95, in steps of 0.05.
  + **37.4%** is the average precision over these multiple thresholds, providing a more comprehensive measure of the model's performance. It shows that, on average, 37.4% of the objects are detected correctly when considering different overlap thresholds.

**3. Processing Time**

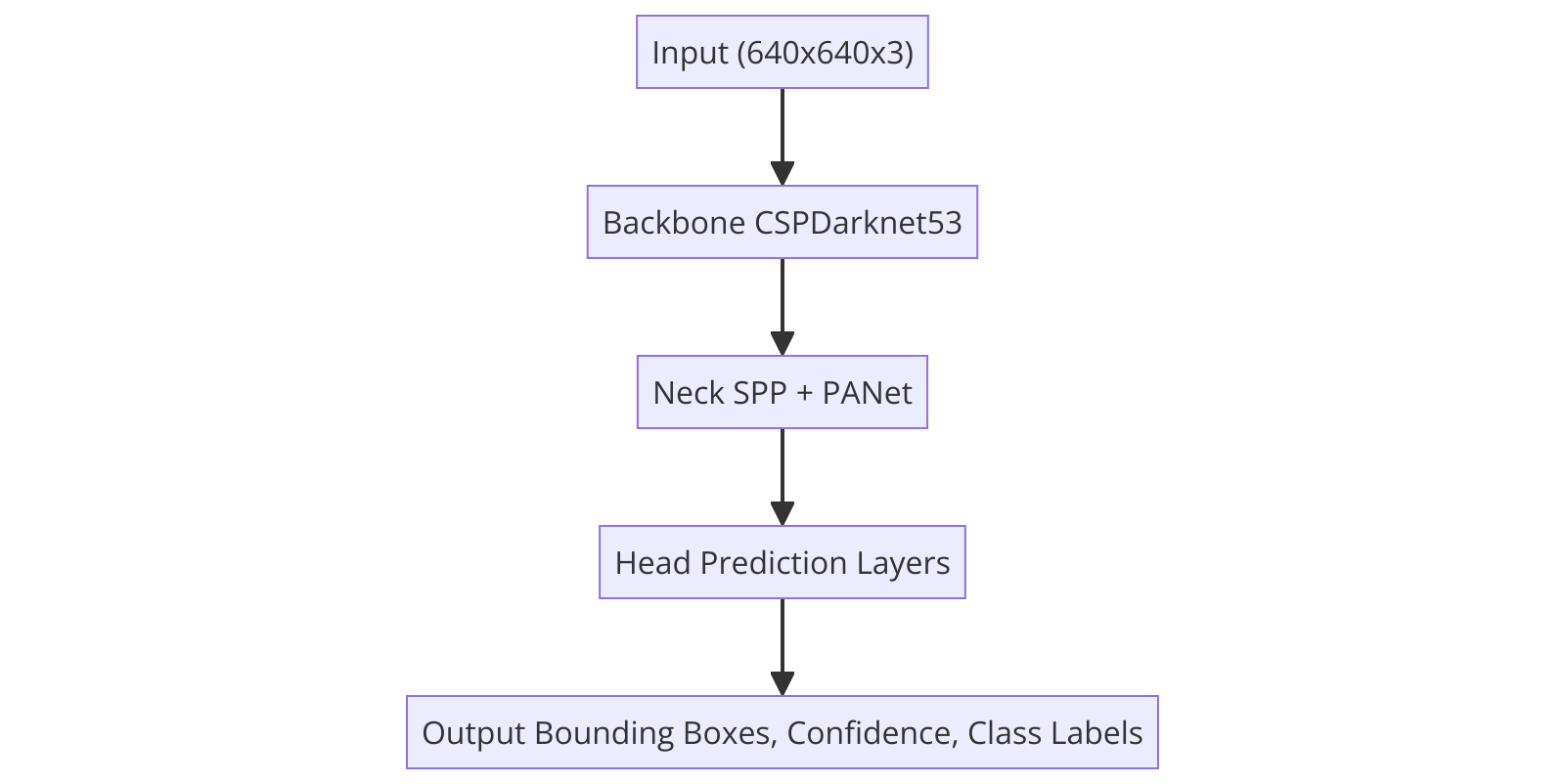
* **Time CPU b1 (ms): 98**
  + This represents the time it takes to process one image on a CPU (Central Processing Unit).
  + **98 milliseconds** per image means the model processes each image in 98 milliseconds on average when running on a CPU.
* **Time V100 b1 (ms): 6.4**
  + This represents the time it takes to process one image on an NVIDIA V100 GPU (Graphics Processing Unit).
  + **6.4 milliseconds** per image means the model processes each image much faster on a GPU compared to a CPU, thanks to the GPU's parallel processing capabilities.

**4. Parameters (Params)**

* **Params (M): 7.2**
  + **Parameters** in a machine learning model are the weights and biases that the model learns during training.
  + **7.2 million parameters** indicates the complexity of the model. A higher number of parameters usually means a more complex model that can potentially capture more intricate patterns in the data.

**5. Floating Point Operations Per Second (FLOPS)**

* **FLOPS @640 (B): 16.5**
  + **FLOPS** stands for Floating Point Operations Per Second, a measure of computational power.
  + **16.5 billion FLOPS** at 640x640 resolution means that the model performs 16.5 billion operations per second when processing images of this size. This metric indicates the computational resources required to run the model efficiently.



**Object Recognition and Face Detection Models for Secondary Stage Students**

**Object Recognition using PYTESSERACT Library**

**Introduction:**

* Optical Character Recognition (OCR) is a technology used to convert different types of documents, such as scanned paper documents, PDFs, or images captured by a digital camera, into editable and searchable data. The pytesseract library in Python is a wrapper for Google's Tesseract-OCR Engine, making it easy to integrate OCR functionality into Python applications.

**How pytesseract Works:**

1. **Image Preprocessing:**
   * **Binarization:** Converting the image to black and white to enhance the contrast between text and background.
     + **Equation:** A screenshot of a computer

       Description automatically generated
     + where I(x,y)I(x, y)I(x,y) is the intensity of the pixel at (x,y)(x, y)(x,y) and TTT is the threshold.
   * **Noise Removal:** Applying filters such as Gaussian blur to remove noise.
     + **Equation:** A math equation on a black background

       Description automatically generated
     + where σ\sigmaσ is the standard deviation.
   * **Deskewing:** Correcting any tilt in the text using techniques such as the Hough transform.
2. **Text Recognition:**
   * **Connected Component Analysis:** Segmenting the image into lines and words.
   * **Feature Extraction:** Identifying character shapes and matching them to known patterns using a neural network.
     + **Neural Network Equation:** A white text on a black background

       Description automatically generated
     + where Wis the weight matrix, x is the input vector, b is the bias vector, and f is the activation function.
3. **Applications:**
   * Digitizing printed documents.
   * Reading text from images captured by cameras.
   * Assisting visually impaired users by converting written text to speech.

**Face Detection using Haarcascade**

**Introduction:**

* Face detection is the process of identifying and locating human faces in digital images. This technology is widely used in various applications, such as facial recognition, security systems, and human-computer interaction.

**How Haarcascade Works:**

1. **Haar-like Features:**
   * Simple rectangular features that capture the intensity differences between adjacent rectangular regions in an image.
     + **Equation:** A black screen with white text

       Description automatically generated
     + where I(i)I(i)I(i) is the pixel intensity in region R1R\_1R1​ and I(j)I(j)I(j) is the pixel intensity in region R2R\_2R2​.
2. **Training the Classifier:**
   * The classifier is trained using a large number of positive (images with faces) and negative (images without faces) samples.
   * AdaBoost algorithm is used to create a strong classifier from a combination of weak classifiers.
     + **AdaBoost Equation:**
     + A math equation with white text

       Description automatically generated with medium confidence
     + where H(x)H(x)H(x) is the final strong classifier, αt\alpha\_tαt​ is the weight assigned to weak classifier ht(x)h\_t(x)ht​(x), and TTT is the number of weak classifiers.
3. **Detection Process:**
   * The trained classifier is applied to an image to detect faces.
   * The image is scanned at multiple scales and positions. The classifier checks each region for the presence of a face.
     + **Multiscale Detection Equation:**
     + A math equation on a black background

       Description automatically generated
     + where D(x,y,s)D(x, y, s)D(x,y,s) is the detection score at position (x,y)(x, y)(x,y) and scale sss, wiw\_iwi​ is the weight of feature fif\_ifi​, and NNN is the number of features.
4. **Applications:**
   * Used in security systems (e.g., surveillance cameras).
   * User authentication (e.g., unlocking devices with face recognition).
   * Human-computer interaction (e.g., detecting user emotions).

**Detailed Explanations for Students**

**Object Recognition using PYTESSERACT:**

1. **Image Preprocessing Steps:**
   * **Binarization:** Convert the image to black and white to enhance text visibility.
   * **Noise Removal:** Apply filters to remove unwanted pixels that may interfere with text recognition.
   * **Deskewing:** Correct any tilt in the image to align the text properly for better recognition.
2. **Text Recognition Steps:**
   * The image is analyzed by the Tesseract-OCR engine.
   * Characters are recognized using feature extraction and neural networks.
   * The recognized text is output as a string.

**Face Detection using Haarcascade:**

1. **Haar-like Features:**
   * Simple rectangular features that capture intensity differences between adjacent regions in the image.
   * Used to identify facial features such as edges and textures.
2. **Training Process:**
   * Positive and negative samples are used to train the classifier.
   * AdaBoost algorithm creates a strong classifier from weak features.
   * The final classifier is a series of stages applied in sequence.
3. **Detection Process:**
   * The image is scanned at different scales and positions.
   * The classifier checks each region for the presence of a face.
   * Faces are detected and marked with bounding boxes.