**INTERNSHIP PROGRESS REPORT**

**Project Title:**

Image Size Optimization for Patient Data (Background Removal & Compression)

**Organization:**

Newro Kaaya (under PRS Neurosciences)

**Internship Duration:**

July – September 2025

**Team Members:**

* Ankit Ghosh
* Kironmoy Mukherjee
* Mayukh Tilak

**Supervisor:**

Mr Stafford Michahial

Mr Pratitah Sharan

**WEEK 1**

**Objective**

* Build a foundational understanding of **Artificial Intelligence (AI)** and **Machine Learning (ML)**.
* Get familiar with **TensorFlow**, the main framework we plan to use.
* Establish a common knowledge base for the team before moving into project-specific tasks.

**Activities & Work Done**

1. **Introduction to Core Concepts**
   * Understood the differences between **AI, ML, and Deep Learning**.
   * Discussed the types of ML: **supervised learning, unsupervised learning, and reinforcement learning**.
   * Explored how models are trained using **datasets**, and the importance of **training vs testing sets**.
2. **Understanding TensorFlow**
   * Installed and set up TensorFlow in our working environment.
   * Learned about **tensors** as the basic data structure in TensorFlow.
   * Studied the role of **layers, models, optimizers, and loss functions** in building ML models.
3. **Team Learning Sessions**
   * Each member explored different tutorials and documentation.
   * Shared insights within the group to ensure everyone had a clear grasp of the basics.
   * Discussed simple examples like **linear regression** and **classification** to see how models learn patterns.

**WEEK 2**

**Objective**

* To take our **first practical step** into applying the AI/ML basics learned in Week 1.
* Try out simple **image classification** and **face detection** tasks, not for perfect results but to **understand the end-to-end process** of building a model.
* Get familiar with the workflow: **dataset → preprocessing → model → training → testing → saving**.

**Activities & Work Done**

**1. Image Classification (Happy/Sad Classifier)**

* We started with a **basic CNN model** to classify images into two categories (happy vs sad).
* The main goal was to see how images are loaded, preprocessed (resizing, scaling), and fed into a deep learning pipeline.
* We experimented with a simple **Convolutional Neural Network (CNN)** to learn the idea of feature extraction through convolution and pooling layers.
* Training was done for multiple epochs, and we monitored how **loss decreases** and **accuracy improves**.
* The model was then tested on some new images and saved for reuse.

👉 This exercise helped us understand how a **classification problem** is structured in practice.

**2. Face Detection (First Attempt)**

* We then attempted a **face detection model**, which was more complex since it required not just classifying but also **locating faces with bounding boxes**.
* We collected images and annotated them using **MakeSense.ai**, exported in **YOLO format**.
* Learned how to **organize datasets** into training, validation, and test sets.
* Tried **data augmentation** (flipping, rotation, scaling) to increase dataset variety.
* Built a model using **VGG16 as a backbone** and customized it to give both classification and bounding box outputs.
* Defined separate loss functions for classification and localization — a new concept for us at this stage.
* Ran training and did some initial testing, even trying **real-time webcam input** to see it work.

👉 This gave us our first experience of how **object detection tasks** differ from simple classification.

**Findings & Challenges**

* We realized this was our **first “dip” into real ML model building**, and much of the time went into just understanding **how the pieces fit together**.
* Handling datasets and annotations was trickier than expected — especially making sure augmented images and labels matched correctly.
* Performance wasn’t our focus yet; the **main takeaway** was getting comfortable with the **model development pipeline**.

**Next Steps**

* Use the learnings from this week to move into more refined experiments.
* Start exploring **Mediapipe** as a tool for face detection and compare its workflow with our custom models.
* Look into efficiency and accuracy improvements once we’re confident with the process.

Link to work:-   
<https://github.com/mukhokironmoy/Tensorflow-Face-Detection-v1.git>  
https://github.com/mukhokironmoy/Deep-CNN-Image-Classifier.git

**WEEK 3**

**Objective**

* Understand how **Mediapipe** works for human body tracking and face detection.
* Learn how it **maps landmark points** to different parts of the human body and how this data can be used for downstream ML tasks.
* Explore potential applications such as **pose estimation, face detection/recognition, and background removal**.

**Activities & Work Done**

1. **Team Learning (Basics of Mediapipe)**
   * Watched the Mediapipe tutorial video ([YouTube link](https://youtu.be/01sAkU_NvOY?feature=shared)) which explained how to:
     + Load Mediapipe’s built-in **Pose solution**.
     + Detect **33 key landmarks** on the human body (like shoulders, elbows, knees).
     + Retrieve landmark coordinates (x, y, z, visibility) as structured data.
     + Overlay these landmarks on a live video stream using OpenCV .
   * Practiced running simple examples to understand how Mediapipe captures real-time movement.
   * Focused on the idea that these landmarks can later be used as **inputs for training models** or for performing tasks like gesture recognition.
2. **Independent Exploration by Team Member**
   * One teammate (with prior AIML experience) went beyond the basics:
     + Experimented with **Mediapipe’s Face Mesh** for detecting 468 facial landmarks.
     + Attempted an early version of **face recognition** by comparing facial landmark embeddings.
     + Began testing **background removal** using Mediapipe’s segmentation capabilities.
3. **Hands-on Exercises (Group)**
   * Extracted pose landmark data from images and video.
   * Learned how Mediapipe outputs landmark data as structured lists (with normalized coordinates).
   * Discussed how this numerical data could be stored in a dataset and later used for ML tasks such as **action recognition** or **movement classification**.

**Findings & Challenges**

* Realized that **Mediapipe simplifies complex CV tasks**, since pose and face detection models are pre-trained and optimized for real-time use.
* Learned how landmark data is accessible as arrays, which makes it easier to integrate into training pipelines.
* Main challenge: balancing **depth vs breadth** — while two members focused on understanding basics, the third member’s deeper dive into face recognition/background removal highlighted the variety of possible directions.

**Next Steps**

* As a group, consolidate our understanding of Mediapipe’s **pose and face modules**.
* Continue experimenting with background removal and compare Mediapipe’s built-in segmentation with other methods (custom CNNs, MODNet).
* Decide on a **consistent workflow** for the team so we progress in sync while still encouraging individual exploration.

Link to work:  
<https://github.com/mukhokironmoy/Mediapipe-basics.git>

**WEEK 4 & 5**

**Objective**

* Begin our first structured attempts at **removing backgrounds from images and video frames**.
* Explore **multiple approaches** using Mediapipe for segmentation and pose detection.
* Test how background removal affects **file size reduction** and **dataset usefulness**.

**Activities & Work Done**

We experimented with several methods. For each, we noted the workflow, tested outputs, and saved code/scripts for reference.

**Method 1 – Pose Cropping (Bounding Box around Person)**

* **Idea:** Use Mediapipe Pose to detect human landmarks, compute a bounding box, and crop only the region containing the subject.
* **Process:**
  + Detect visible body keypoints.
  + Calculate minimum and maximum x,y coordinates to form a bounding box.
  + Add a fixed margin (converted from cm to pixels).
  + Save cropped frames as JPEG with compression.
* **Expected Advantage:** Reduces storage by cutting out background pixels entirely.

📂 **Code Link:** [pose\_cropping.py](https://github.com/mukhokironmoy/NewroKaaya_Alpha_Matting/blob/c04956b1dbe2d6e180b70cdd48d3c5961f5dc6bc/Week_4/Method%201%20-%20Pose%20Cropping/pose_cropping.py)

📝 **Remarks:** *Dynamic cropping working but compression needs to be better. Filesize needs to be reduced way more. Not reliably for low res photos that come out of poise. Would lead to garbage in garbage out results.*

**Method 2 – Pose Erasing with Alpha Matte**

* **Idea:** Create a **mask of body landmarks and connections**, then erase/remove the background by applying an alpha channel.
* **Process:**
  + Build a mask by filling circles around each joint and connecting bones.
  + Smooth and dilate the mask to create a continuous silhouette.
  + Merge mask with original frame to produce RGBA images with transparent backgrounds.
* **Expected Advantage:** Allows saving **PNG with transparency** for flexibility in downstream ML tasks.

📂 **Code Link:** [pose\_erase.py](https://github.com/mukhokironmoy/NewroKaaya_Alpha_Matting/blob/c04956b1dbe2d6e180b70cdd48d3c5961f5dc6bc/Week_4/Method%202%20-%20Pose%20Erasing/pose_erase.py)  
📝 **Remarks:**

* *png makes the file size massive at the cost of transparency*
* *loss of data near chest area*
* *tracking of body parts isn’t accurate*

**Method 3 – Real-time Segmentation and Cropping**

* **Idea:** Use **Mediapipe Selfie Segmentation** in live webcam feed to isolate person from background.
* **Process:**
  + Generate segmentation mask in real time.
  + Apply binary threshold to extract person.
  + Crop around the detected region with margin.
  + Save output frames as JPEG with compression.
* **Expected Advantage:** Useful for **real-time applications** and efficient dataset collection.

📂 **Code Link:** [realtime.py](https://github.com/mukhokironmoy/NewroKaaya_Alpha_Matting/blob/c04956b1dbe2d6e180b70cdd48d3c5961f5dc6bc/Week_4/Method%203%20-%20Real%20time%20segmentation%20and%20cropping/realtime.py)  
  
📝 **Remarks:**

* *File size is much smaller than png due to jpg output*
* But still needs more compression to even compare to poise output requirements
* Background removal is not reliable. Lots of data loss.

**Method 4 – Batch Segmentation and Compression**

* **Idea:** Process entire folders of saved frames with segmentation + JPEG compression.
* **Process:**
  + Loop through stored images.
  + Apply Mediapipe segmentation mask for background removal.
  + Crop subject and save with **JPEG optimization (quality 95 → 65 → 55)**.
  + Combines both background removal and lossy compression for maximum file size reduction.
* **Expected Advantage:** Scalable for **large datasets**; integrates background removal + compression in one pipeline.

📂 **Code Link:** [segment\_and\_compress.py,](https://github.com/mukhokironmoy/NewroKaaya_Alpha_Matting/blob/c04956b1dbe2d6e180b70cdd48d3c5961f5dc6bc/Week_4/Method%204%20-%20Batch%20segmentation%20and%20compress/segment_and_compress.py)   
📝 **Remarks:**

* Relatively decent bg removal but still a few traces of data loss for a couple frames
* Filesize is decent with jpg
* Can be better with further compression

**Method 5 – Combined Background Removal + Compression Pipeline**

* **Idea:** Create a single pipeline that performs both background removal and multi-stage JPEG compression for maximum file size reduction.
* **Process:**
  + Step 1: Apply an initial compression (simulating Guetzli-like optimization).
  + Step 2: Run Mediapipe Pose with segmentation enabled to remove background.
  + Step 3: If segmentation fails, use a convex hull fallback based on visible landmarks to approximate the body region.
  + Step 4: Save the processed output as JPEG.
  + Step 5: Apply a second, stronger compression step to further reduce file size.
* **Expected Advantage:**
  + Produces significant space savings while still keeping the subject pixels.
  + Can be used as a batch processor for entire datasets.
  + Flexible fallback ensures at least partial results even when segmentation is imperfect.

**📂 Code Link:** [**imagecompression.py**](https://github.com/mukhokironmoy/NewroKaaya_Alpha_Matting/blob/c04956b1dbe2d6e180b70cdd48d3c5961f5dc6bc/Week_4/Method%205%20-%20Image%20Compression/imagecompression.ipynb) **📝 Remarks:**

* Much more stable than other methods in terms of bg removal and file size
* Still a bit of data loss
* Needs very lenient confidence threshold for getting results. Needs to be worked on.

**Findings & Challenges**

* **Pose Cropping** worked well for quick storage reduction, but cropping alone does not eliminate complex backgrounds.
* **Alpha Matte Erasing** produced cleaner outputs but PNGs were larger in size compared to JPEG.
* **Real-time Segmentation** was efficient but requires tuning confidence thresholds to avoid false negatives.
* **Batch Compression** gave the best balance between **automation and size reduction**, but requires further testing on large datasets.

**Next Steps**

* Compare results of each method quantitatively (file size, quality retention, ML usability).
* Decide on a **primary workflow** for dataset processing.
* Explore whether advanced models like **MODNet** provide better background removal than Mediapipe.

**WEEK 6**

**Objective**

* Explore whether there are better tools than MediaPipe for **background removal**.
* Investigate both **free** and **paid** options for image compression and storage optimization.
* Understand the **practical limits of PNG with transparency**, and whether it fits our use case.

**Activities & Work Done**

**1. PNG with Transparency – Size Limits**

* We checked if **PNG with alpha channels** can realistically reach ~5KB file sizes.
* Findings:
  + Apparently possible only when images are **very small (≤160×160 px)**, have **large transparent regions**, or are **extremely simple**.
  + Even with aggressive compression, most usable PNGs with transparency end up in the **15–60KB range**.
  + Conclusion: **PNG is not the best choice** if the primary goal is minimizing file size.

**2. Free & Open-Source Options**

1. **MODNet / RVM (Robust Video Matting)**
   * Advanced open-source models for background removal.
   * Produce cleaner and tighter subject masks compared to MediaPipe.
   * Can be combined with cropping to further cut file size.
   * ➡️ Improves accuracy, but not compression directly.
2. **Image Compression Utilities**
   * jpegoptim → Optimizes JPEGs with lossy/lossless modes.
   * cwebp → Converts to WebP with strong compression.
   * oxipng → Optimizes PNG files, good for alpha transparency.
   * ImageMagick → Flexible tool for cropping, resizing, and conversions.
   * ➡️ All tools are scriptable and batch-friendly.
3. **MediaPipe Selfie-Segmentation**
   * Lighter and faster than Pose-based segmentation.
   * Works well for **waist-up videos and faces/hands**.
   * ➡️ Faster = more scalable and efficient.
4. **Guetzli (Experimental)**
   * Google’s JPEG encoder, very aggressive compression.
   * ⚠️ Too slow for batch jobs → not practical for large-scale use.

**3. Paid Options (Enterprise-Ready)**

1. **remove.bg**
   * High-quality background removal API.
   * Handles fine details (hair, fingers) much better than free models.
   * Supports PNG with transparency.
   * Cost: ~$0.20 per image (bulk discounts available).
2. **Krikey AI / RunwayML**
   * Focused on **video background removal**.
   * Useful if we want to process entire motion capture sessions directly, not just frames.
   * More expensive; suited for professional media pipelines.
3. **TinyPNG Pro**
   * API for compressing PNGs and WebPs.
   * Can reduce large PNGs (~100KB+) to ~15KB while keeping transparency.
   * Very useful if organization wants to keep transparent images.
4. **Cloudinary / Imgix**
   * Cloud-based image optimization and hosting.
   * Provides on-the-fly optimized delivery (different resolutions, formats).
   * Best suited if images are to be displayed online in addition to storage.

**Findings & Key Takeaways**

* **PNG transparency at 5KB is not realistic** for most patient data images; JPEG/WebP are more storage-efficient.
* **WebP with cwebp compression** gives the best balance of size and quality.
* **remove.bg** provides the cleanest background removal if paid options are allowed.
* For free pipelines, combining **MODNet/RVM for masks** with **cwebp/jpegoptim for compression** seems most promising.

**Next Steps**

* Benchmark MODNet/RVM against our current MediaPipe workflow.
* Test cwebp compression on cropped outputs to measure file size savings.
* Prepare a decision matrix (accuracy vs size vs cost) for supervisor approval.

👉 Do you want me to also prepare that **comparison/decision matrix** (Free vs Paid methods with columns for Accuracy, Speed, File Size, Cost) for Week 6, so it’s easy for your supervisor to make a call?

**WEEK 7 & 8**

**Objective**

* Reduce storage size of Poise session frames (~10 KB JPEGs at 30 fps).
* Preserve all clinically relevant body parts (hands, arms, legs) while removing unnecessary background.
* Explore and benchmark different background removal and compression methods.
* Move towards a stable, automated pipeline for large-scale processing.

**Activities & Work Done**

1. **Initial Experiments with File Formats**
   * Converted Poise JPEG frames to **WebP**.
     + Finding: On our low-detail frames, WebP **increased file size** instead of reducing.
     + Decision: WebP not viable as a direct replacement for JPEG.
2. **Background Removal with Mediapipe**
   * Tried **Mediapipe Selfie Segmentation** for extracting subjects.
     + Advantage: lightweight and real-time capable.
     + Issues:
       - Low-quality (10 KB) images → masks were inaccurate.
       - Cropping based on masks often cut away limbs or key body parts.
   * Outcome: Not reliable for production but useful for quick tests.
3. **Cropping Approaches Tested**
   * Naïve cropping from segmentation masks.
     + Issue: **unstable bounding boxes** and high risk of cutting important regions.
   * Lesson: **Landmark-aware cropping** (using Poise-provided coordinates) is needed instead of mask-only cropping.

**Proposed Workflow (Pipeline)**

1. **Ingest**
   * Input: Frame folder (JPEGs) + Poise landmark data (coordinates of body joints).
   * Validation: Check missing landmarks, frame consistency.
2. **Background Removal**
   * **Primary:** Use **MODNet** for alpha matte → better isolation of subject compared to Mediapipe.
   * **Fallback:** Mediapipe (with denoise + relaxed thresholds) if MODNet unavailable.
3. **Landmark-Aware Cropping**
   * Compute bounding box from Poise landmarks (shoulders, hips, wrists, ankles).
   * Add configurable margins to ensure no limb truncation.
   * Apply temporal smoothing to reduce jitter between consecutive frames.
4. **Compression**
   * Run cropped frames through **CLI optimizers** (jpegoptim / mozjpeg).
   * Tune JPEG quality (55–70) to balance size vs clarity.
   * WebP only for cases with very simple crops (optional).
5. **Reporting & QA**
   * Track size reduction per frame and per session.
   * Verify all landmarks remain inside the cropped region.
   * Create side-by-side samples for quick human review.

**Findings & Challenges**

* **WebP** is not automatically better — in our data, it performed worse than JPEG.
* **Mediapipe** segmentation struggles with small low-quality frames, producing unstable masks.
* **Naïve Cropping** caused frequent data loss; landmark-guided cropping is essential.
* Key challenge: Balancing **accuracy (preserve subject)** with **compression (reduce size)**.

**Next Steps**

* Implement MODNet on sample sessions to test matte quality.
* Develop **landmark-aware cropping logic** with margins + smoothing.
* Benchmark compression settings to find best quality/size tradeoff.
* Automate reporting pipeline for session-level results.
* Run pilot tests on multiple sessions to validate both storage savings and clinical usability.

Link to Code:

https://github.com/mukhokironmoy/NewroKaaya\_Alpha\_Matting/tree/7bfabff8dce0147bc2c57983e1e197e1a4d6451d/Week\_7

**WEEK 9-10**

**Week 9 & 10 – Landmark Mapping and JPEG Behavior Experiments**

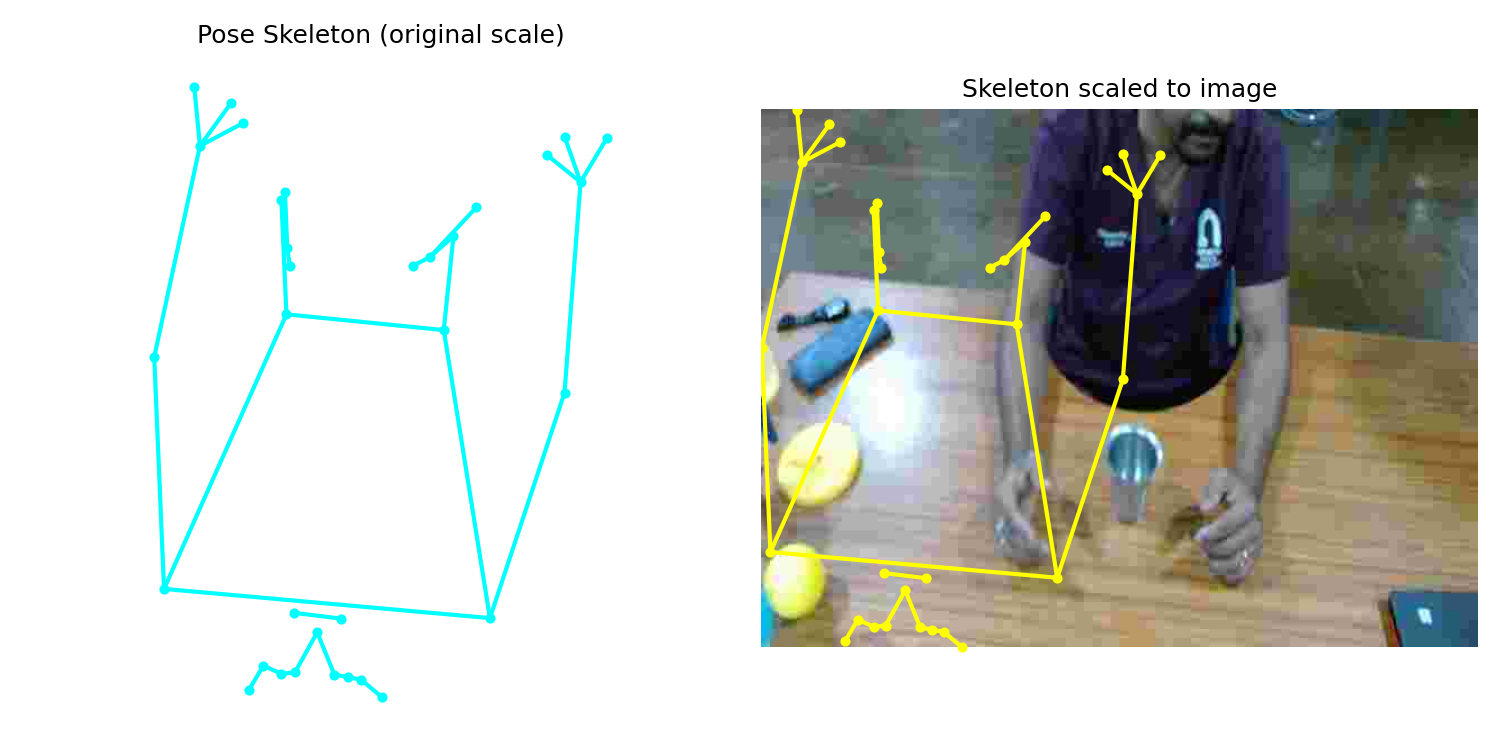
**Objective**

* Verify and validate **Poise-provided landmark data** against the actual frames.
* Explore issues with **landmark-to-image alignment** caused by scaling differences.
* Study the behavior of the **JPEG format** during cropping, recompression, and manual optimization.
* Identify a practical workflow for **extreme file size reduction (target 2–5 KB)** while preserving clinically useful details.

**Activities & Work Done**

**1. Landmark–Image Mapping**

* Implemented a script to **parse Poise landmark files** and align them with corresponding frame images.
* Generated a mapping.json file storing:
  + Image path
  + Associated list of landmarks ([x, y] coordinates).
* Tested on a **sample of 100 frames** from one Poise session.
* Attempted to overlay landmarks on images for visual verification.
* **Issue Discovered:**
  + Landmarks appear **misaligned** when plotted, because the landmarks were captured on **full-resolution frames**, while the images we received had been **downscaled**.
  + This caused skeletons to drift outside the actual body in the frames (see attached figure: original scale vs image scale).



**2. JPEG Cropping and File Size Behavior**

* Conducted experiments to understand why cropped JPEGs sometimes end up **larger in size** (e.g., 5 KB → 11 KB).
* Findings (confirmed via GPT-assisted research):
  + **Recompression Settings:** Editors often default to higher quality factors on save, inflating file size.
  + **Compression Efficiency:** Cropping away smooth regions leaves more complex details, making compression less effective.
  + **Metadata Overhead:** Some tools reinsert EXIF/color profiles.
  + **Pixel Complexity:** JPEG size depends on detail + quality factor, not just pixel count.

**3. Manual Optimization Experiments**

* Tried **downscaling → cropping → re-saving at lowest quality factor**.
* Tools used: Windows default Photos editor.
* Observations:
  + Downscaling drastically reduces file size while retaining subject outline.
  + Maintaining **minimum quality factor** is essential to avoid size bloat.
  + Cropping alone without controlling quality factor is unreliable.
* **Result Achieved:**
  + Reduced ~10 KB Poise JPEGs down to **2–5 KB** consistently.
  + Shows that aggressive downscaling + controlled compression can meet storage targets.

A screenshot of a computer

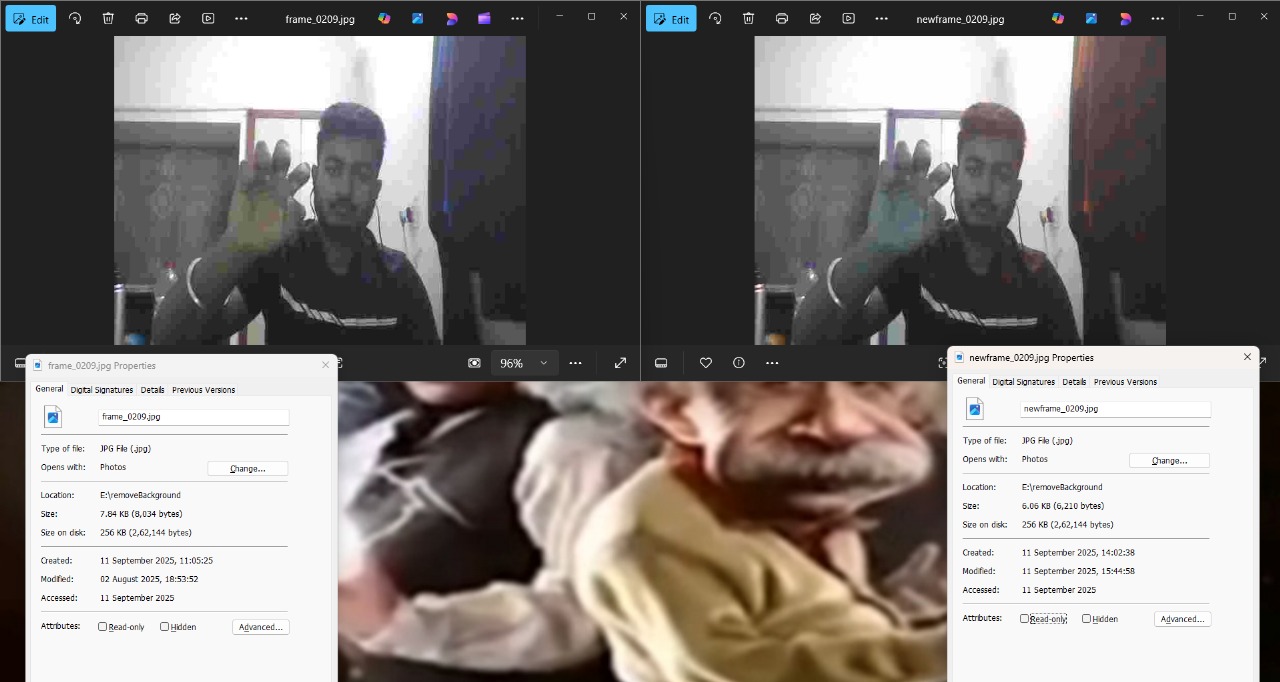
AI-generated content may be incorrect.

**4. Parallel work on Mozjpeg – Background Removal & Compression Pipeline**

While the above work focused on landmark mapping and JPEG behavior, we designed a **stepwise automated pipeline**:

* **Input image → rembg:** Applied rembg (a background removal tool) to generate an initial subject mask.
* **Dilated mask:** Expanded mask boundaries slightly to ensure no body parts were cut.
* **Crop:** Extracted subject tightly based on the refined mask.
* **Solid background replacement:** Replaced transparent/removed areas with a uniform solid background (avoiding PNG alpha bloat).
* **mozjpeg compression:** Finally compressed the cropped output using **mozjpeg**, which applies efficient lossy JPEG optimization.

This pipeline demonstrated a structured way to balance **subject preservation** with **file size reduction**, and avoids the pitfalls of PNG with transparency. It demonstrated a **reduction of 1.5 – 2 KB in the images.**



**Findings & Challenges**

* Landmark misalignment highlights the need for **coordinate rescaling logic** (mapping landmark origin to downscaled frame dimensions).
* JPEG cropping must be paired with **explicit compression controls** — otherwise file size may increase unexpectedly.
* The most effective strategy appears to be:
  + **Downscale → Crop (using corrected landmarks) → Save with minimal quality factor.**
* Manual experiments confirm feasibility, but automation is needed for large-scale sessions.

**Next Steps**

* Develop a **landmark-rescaling function** to correctly align Poise coordinates with downscaled images.
* Automate the **downscale + crop + compression pipeline** to consistently achieve 2–5 KB targets.
* Benchmark quality trade-offs (clarity of limbs/hands vs extreme compression).
* Test pipeline on **multiple Poise sessions** to confirm generalizability.

Link to code:-

Mapping and Downscaling code:  
<https://github.com/mukhokironmoy/NewroKaaya_Alpha_Matting/tree/f6a328137aa0280d75289498c67d7fa176cd4629/Week_9-10>  
  
Mozjpeg:  
<https://github.com/ankitghoshthecreator/rembg.git>

**WEEK 11**

**Objective**  
Refine cropping and compression workflows using Poise landmarks and external tools.  
Investigate format compatibility (WebP vs JPEG) and behavior of different libraries (OpenCV, Pillow, Mozjpeg).  
Push towards automation while minimizing file size without data loss.

**Activities & Work Done**

**1. WebP Conversion & Mediapipe Masking**

* Converted input images from JPEG to WebP for initial storage testing.
* Used Mediapipe landmarks to generate masks and black out unnecessary regions.
* Applied cropping guided by the generated masks.
* **Setback:** Unsure if Poise software can handle WebP format for downstream compatibility.A diagram of a flowchart

  AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**2. Landmark Superimposition & Cropping**

* Implemented a script to overlay Poise-provided landmarks directly onto frames for visual verification.
* Generated a mask using the landmark points to isolate the subject and crop accordingly.
* Reduced file size by lowering JPEG quality factors.
* **Setback:** Both OpenCV and Pillow unexpectedly increased output file sizes when saving cropped images, highlighting inefficiencies in recompression.

A person standing in a room with a green line

AI-generated content may be incorrect.

A person standing in a mirror

AI-generated content may be incorrect. A person wearing a white shirt and shorts

AI-generated content may be incorrect.

**3. Mozjpeg & Grayscale Compression**

* Applied Mozjpeg to optimize cropped frames with stronger JPEG compression.
* Tested grayscale conversion as an additional method to shrink file sizes while retaining subject shape.
* Showed consistent reductions compared to baseline JPEG compression.

**Findings & Challenges**

* WebP provides promising compression, but software compatibility with Poise remains uncertain.
* Standard libraries (OpenCV, Pillow) often inflate file size during save operations due to recompression defaults.
* Mozjpeg remains the most effective tool for consistent reductions.
* Grayscale images reduce size further but may risk losing clinically relevant details if not carefully validated.

Link to code:

<https://github.com/mukhokironmoy/NewroKaaya_Alpha_Matting/tree/034ab79de03819fce3ec5aff4faf1cf1160a5fc9/Week_11>

**🔎 What’s happening when crop=True**

* You crop to just the bounding box around the hull.
* The cropped region contains **almost only the person (lots of detail, colors, edges)** and **very little black space**.
* Compression has to preserve much more “real information” per pixel.
* So even though dimensions are smaller, the **entropy per pixel is higher**, leading to **bigger files** at the same quality=80.
* In other words:
  + crop=False → full frame with tons of empty black = highly compressible.
  + crop=True → tighter crop with mostly complex content = harder to compress.

That’s why **cropped images blow up in size** compared to uncropped ones.