**🖐️ Hand Gesture Recognition — week 3/4**

**Project name:** Hand Gesture Recognition (CNN + MLflow)

**Goal of this week task :** Build part pf project for a clear and reproducible pipeline to train a Convolutional Neural Network (CNN) that classifies static hand gestures (images). Use MLflow to track experiments and save the final model for delivery.

**Scope (initial):** Static images of hand gestures (alphabet/words). Dynamic gestures / sentence-level translation are out of scope for the first version.

**Full Pipeline (high-level)**

1. **Data collection & organization** — gather images or use an existing dataset. Keep metadata (subject id, session id).
2. **Preprocessing** — resize, normalize, optional cropping, color handling.
3. **Data splitting** — split by subject/session (to avoid leakage), stratified by class → train/val/test.
4. **Augmentation** — applied only to the *training* set.
5. **Model training** — baseline CNN; add regularization and tuning.
6. **Evaluation** — evaluate on hold-out test set; compute metrics and confusion matrix.
7. **Experiment tracking** — use MLflow: log params, metrics, artifacts.
8. **Save & package** — export model file(s) and environment description.
9. **Deliver & demo** —code, model, MLflow runs, short demo or instruction to run.

**Data & Preprocessing (detailed)**

**1. Image format & color**

* Keep RGB images if color information is helpful; otherwise grayscale reduces compute.

**2. Resize**

* Choose a fixed resolution: **64×64** or **128×128**.
  + 64×64: faster training, less memory, may lose fine details.
  + 128×128: better for finer finger poses but slower.

**3. Normalization**

* Scale pixel values to [0, 1] (divide by 255). Optionally use per-channel mean/std normalization if using pretrained backbones.

**4. Augmentation (apply only to training set)**

* Rotation: ±15°
* Width/height shift: up to 10% (0.1)
* Zoom range: 0.9–1.1
* Brightness variation: ±20%
* Horizontal flip: use with caution — only if flips do not change label meaning (many sign gestures are asymmetric).
* Minor shear or random crop if needed.

**5. Extra preprocessing considerations**

* If dataset has bounding boxes or hand crops, center/align images so hand occupies a consistent region.
* Keep a reproducible script that performs preprocessing and writes processed images or a single .npz/.npy file.

**🧩 Avoiding Data Leakage — Recommended Practice**

* **Split by subject/session**: ensure each subject (person) appears in exactly one of train/val/test. This prevents the model from learning person-specific clues.
* **Stratify** by class when splitting, to keep class distribution consistent.
* **Augment after splitting**: do augmentation only on training set, never on validation/test.
* **Fix random seeds** for reproducibility and store the indices of each split (save a CSV listing the test filenames).
* **Test set must be untouched** until the final evaluation (no hyperparameter tuning using test).

**Model Design (baseline CNN)**

**Architecture (example, high-level):**

* Conv2D(32, 3×3) → ReLU → MaxPool(2×2)
* Conv2D(64, 3×3) → ReLU → MaxPool(2×2)
* Conv2D(128, 3×3) → ReLU → MaxPool(2×2)
* Flatten → Dense(128) → ReLU → Dropout(0.4)
* Output Dense(num\_classes) → Softmax

**Enhancements to try:**

* Batch Normalization after conv blocks.
* Dropout in dense layer(s) to reduce overfitting.
* Weight decay (L2 regularization) if overfitting persists.
* Transfer learning: MobileNetV2 / EfficientNet/B0 as feature extractor if dataset small.

**Why this baseline?**

* Simple, fast to train, interpretable. Good starting point before trying heavier models.

**⚙️ Training Procedure & Hyperparameters**

**Loss & optimizer**

* Loss: categorical\_crossentropy (for multi-class).
* Optimizer: Adam (initial lr = 1e-3).

**Training schedule**

* Batch size: 16–64 (try 32 as starting point).
* Epochs: 30–50 with early stopping (monitor val\_loss with patience=5).
* Callbacks: EarlyStopping, ModelCheckpoint (save best by val\_loss), ReduceLROnPlateau.

**Validation**

* Use validation set during training to select best model and tune hyperparameters.

**Metrics to track**

* Accuracy, Precision, Recall, F1-score (per-class), Confusion Matrix.

**📦 Model Saving & Formats**

**Framework-native formats**

* **Keras/TensorFlow**: .h5 (HDF5) or SavedModel directory.

**Interchange formats**

* **ONNX**: export if you need cross-framework compatibility.
* **TensorFlow Lite**: for mobile deployment (optional later).

**Important**

* Avoid using pickle to save heavy DL models. Use framework serializers.
* Save also the *preprocessing metadata* (image size, normalization scheme) and the label map (class → index).

**📊 Evaluation & Reporting**

* Evaluate the final model only on the untouched **test set**.
* Produce:
  + Test accuracy and loss.
  + Per-class precision/recall/F1.
  + Confusion matrix image.
  + Training curves (loss & accuracy) for train vs val.
* Save evaluation artifacts and log them to MLflow.

**🔍 MLflow — Experiment Tracking (what to log)**

**What to log**

* Parameters: architecture name, learning rate, batch size, epochs, augmentation used.
* Metrics: train/val accuracy & loss per epoch, final test metrics.
* Artifacts: saved model file, training plots (png), confusion matrix (png), sample predictions csv.
* Dataset version/hash (or metadata.csv) and git commit id for reproducibility.

**How to use**

* Create an experiment for this project.
* For each run, log params, metrics and artifacts.
* Use mlflow ui to compare runs and pick the best configuration.

**🧾 Deliverables (before submission)**

* Well-structured repository with README.md.
* src/ scripts: preprocess, train, evaluate, inference utility.
* models/ directory with the final model file(s).
* requirements.txt (or environment.yml).
* MLflow experiment with logged runs and the best run marked.
* Short report (PDF or Markdown) summarizing results and decisions.
* Optional: short demo video or Streamlit app to show inference.

**⚙️ Environment & Dependencies (suggestion)**

* Python = 3.11
* TensorFlow
* OpenCV (opencv-python)
* scikit-learn
* pandas, matplotlib
* mlflow

Put exact package versions in requirements.txt for reproducibility.

**🔭 Next steps & Improvements (post-submission)**

* Move to landmark-based approach (MediaPipe) for lighter-weight real-time inference.
* Support dynamic gestures using temporal models (LSTM/Transformer / 3D-CNN).
* Try transfer learning with MobileNet / EfficientNet for better accuracy with limited data.
* Convert model to TensorFlow Lite for mobile deployment.

**📌 Notes & Best Practices**

* Keep a single source of truth for preprocessing; inference must apply exactly the same steps.
* Save random seeds and the split lists to avoid accidental leakage.
* Document any manual cleaning or filtering you do on the dataset.

**📆 Week 3–4: Model Development & Initial Prototyping**

**Team Members’ Roles**

* **Mahmoud**: Oversee overall model development, ensuring alignment with project goals.
* **Yousef**: Focus on model deployment, integration, and user-facing aspects of the application.
* **Mazen**: Implement and train the initial CNN model, and educate team members on its functionality.
* **Samy**: Develop the Streamlit application for the user interface.
* **Ahmed**: Implement and train Mediapipe-based recognition, and educate team members on its functionality.
* **Kerolos**: Assist with model evaluation and testing, and educate team members on model performance metrics.
* **Fouad**: Document all aspects of model development, training, and deployment.