โครงงานรายวิชา

เรื่อง เทรนโมเดลจำแนกท่าทางมือ

เสนอ

ผู้ช่วยศาสตราจารย์ ดร.ภารุจ รัตนวรพันธุ์

จัดทำโดย

นางสาวเปรมมิกา เนียมเปรม รหัสนิสิต 6610505489

นางสาววริษฐา ภิรมย์เกียรติ์ รหัสนิสิต 6610505551

รายงานนี้เป็นส่วนหนึ่งของรายวิชา

ระบบปฏิบัติการ Deep Learning (01204466)

คณะวิศวกรรมศาสตร์ ภาควิชาวิศวกรรมคอมพิวเตอร์

มหาวิทยาลัยเกษตรศาสตร์ วิทยาเขตบางเขน

ภาคเรียนที่ 1 ปีการศึกษา 2568

สารบัญ

1. หัวข้อ final project

การจำแนกท่ามือ 6 คลาส {dislike, fist, like, ok, palm, peace} ใช้ได้จริงกับระบบสั่งงานไร้สัมผัส AR/VR และช่วยการเข้าถึงสำหรับผู้ใช้พิเศษ ความท้าทายคือท่ามือคล้ายกัน (เช่น ok vs like), สภาพแสง/พื้นหลังหลากหลาย และมือที่มีขนาดเล็กในภาพ จึงเป็นโจทย์ที่เหมาะกับงานสุดท้ายเพราะต้องออกแบบทั้งสถาปัตยกรรม การฝึก และการประเมินผลอย่างรอบด้าน

1.1. ทำไมหัวข้อนี้จึงต้องใช้ deep learning ในการแก้ปัญหา

1.1.1. จำแนกท่ามือจากภาพจริงที่มีความแปรผันสูง (แสง เป้าหมายขนาดเล็กอย่างนิ้วมือ มุมกล้อง พื้นหลัง คนต่างผิว/อุปกรณ์)

1.1.2. CNN เรียนรู้ฟีเจอร์ลำดับชั้นจากภาพโดยตรง รองรับความแปรผันสูง และใช้ pretrained ช่วยให้เรียนรู้เร็วขึ้นด้วยข้อมูลไม่มาก

1.1.3. ข้อเด่นของ Deep Learning

1.1.3.1. ความแม่นยำและความยืดหยุ่นสูง: แยกท่าที่คล้ายกัน (OK vs Palm vs Peace) ได้ดีเมื่อใช้ข้อมูลหลากหลาย + augmentation

1.1.3.2. Transfer learning: ใช้ ResNet/ViT ที่ pretrain บน ImageNet ช่วยให้ต้องการข้อมูล gesture น้อยลง และ converge เร็ว

1.1.3.3. End-to-end: รวมการเรียนรู้ฟีเจอร์ + ตัวจำแนกในโมเดลเดียว ลดการประกอบท่อหลายขั้น

1.1.4. ข้อด้อยของ Deep Learning

1.1.4.1. ต้องการทรัพยากร: เทรนต้องใช้ GPU เวลา infer ถ้าโมเดลใหญ่ก็หน่วง

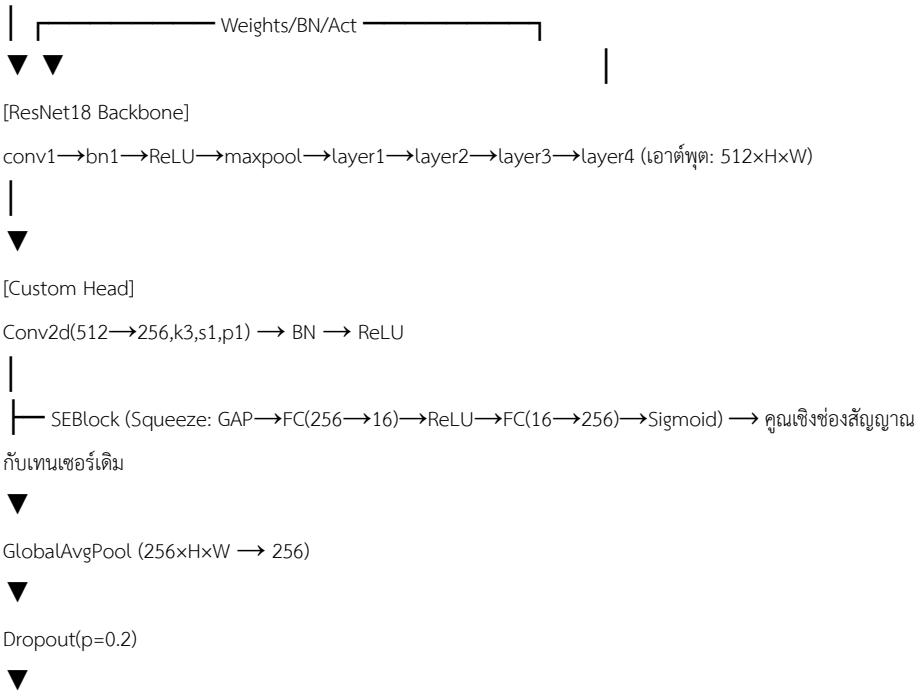
1.1.4.2. ต้องการข้อมูลคุณภาพ: ถ้าข้อมูลไม่พอ/ลำเอียง จะเกิด bias และ overfit

1.1.4.3. วิศวกรรมปฏิบัติ: ต้องดูแล pipeline (augmentation, normalization, lr schedule, early stopping) ให้ดี

1.2. อธิบายสถาปัตยกรรม deep learning

1.2.1. ชนิดโมเดล: CNN (Convolutional Neural Network) แบบ Transfer Learning

1.2.2. แผนภาพโครงสร้าง



Linear(256⟶6) + Bias

▼

Softmax (ความน่าจะเป็น 6 คลาส)

1.2.3. ResNet18 (Backbone): เป็น CNN ที่มีชื่อเสียง ถูกออกแบบมาเพื่อแก้ปัญหา vanishing/exploding gradients ในเครือข่ายที่ลึกมากๆ โดยใช้ "residual connections"ที่ช่วยให้ข้อมูลไหลผ่านหลายชั้นได้ง่ายขึ้น

1.2.3.1. base = models.resnet18(weights=weights) โหลด ResNet18 ที่อาจจะถูก pre-trained บนชุดข้อมูล ImageNet มาแล้ว (use_pretrained=True)

1.2.3.2. self.backbone ประกอบด้วยส่วนเริ่มต้นของ ResNet18: conv1, bn1, relu, maxpool และ layer1, layer2, layer3, layer4 ซึ่งเป็นกลุ่มของ Residual Blocks แต่ละกลุ่มมีจำนวนชั้นและความซับซ้อนเพิ่มขึ้น

1.2.3.3. เอาต์พุตจาก layer4 ของ ResNet18 จะมีขนาดช่องสัญญาณ (channels) เท่ากับ 512 ซึ่งจะถูกส่งต่อไปยังส่วนหัว (GestureHead)

1.2.4. Custom Head :

1.2.4.1. ConvBNAct (Convolutional Layer):

1.2.4.1.1. nn.Conv2d(in_ch, mid_ch, k=3, s=1, p=1, bias=False): ชั้น Conv2d ที่มีขนาด kernel 3x3, stride 1, padding 1

1.2.4.1.1.1. in_ch คือ 512 (จาก ResNet18 layer4)

1.2.4.1.1.2. mid_ch คือ 256 (ค่าเริ่มต้น)

1.2.4.1.1.3. bias=False เนื่องจากมี Batch Normalization

1.2.4.1.2. nn.BatchNorm2d(mid_ch): Batch Normalization ช่วยลด covariate shift และทำให้การฝึกเร็วขึ้น

1.2.4.1.3. nn.ReLU(inplace=True): Activation function แบบ ReLU (Rectified Linear Unit)

1.2.4.2. SEBlock (Squeeze-and-Excitation Block):

1.2.4.2.1. เป็นกลไก Attention ที่ช่วยให้โมเดลเรียนรู้ที่จะให้ความสำคัญกับช่องสัญญาณ (channel) ที่สำคัญ และลดความสำคัญของช่องสัญญาณที่ไม่เกี่ยวข้อง

1.2.4.2.2. nn.AdaptiveAvgPool2d(1): ทำ Global Average Pooling เพื่อบีบมิติเชิงพื้นที่ให้เหลือ 1x1

1.2.4.2.3. self.fc: เป็น MLP ขนาดเล็กที่ประกอบด้วย Conv2d สองชั้นคั่นด้วย ReLU และจบด้วย Sigmoid

1.2.4.2.3.1. nn.Conv2d(ch, ch // r, 1, bias=True): ลดจำนวนช่องสัญญาณ

1.2.4.2.3.2. nn.ReLU(inplace=True)

1.2.4.2.3.3. nn.Conv2d(ch // r, ch, 1, bias=True): กู้คืนจำนวนช่องสัญญาณเดิม

1.2.4.2.3.4. nn.Sigmoid(): สร้างค่า weight ระหว่าง 0 ถึง 1 สำหรับแต่ละช่องสัญญาณ

1.2.4.2.4. เอาต์พุตของ SEBlock จะถูกนำไปคูณกับอินพุตเดิม (x * w) เพื่อปรับขนาดความสำคัญของแต่ละช่องสัญญาณ

1.2.4.3. nn.AdaptiveAvgPool2d(1) (Global Average Pooling - GAP):

1.2.4.3.1. ลดมิติของ feature map จาก (mid_ch, H, W) เหลือ (mid_ch, 1, 1) โดยการหาค่าเฉลี่ยของแต่ละ feature map

1.2.4.3.2. จากนั้น .flatten(1) จะเปลี่ยนให้เป็นเวกเตอร์ขนาด (mid_ch,)

1.2.4.4. nn.Dropout(p=drop_p):

1.2.4.4.1. ชั้น Dropout ที่จะสุ่มปิดโหนดเป็นจำนวน drop_p (0.2 โดยค่าเริ่มต้น) เพื่อป้องกัน overfitting

1.2.4.5. nn.Linear(mid_ch, num_classes) (Fully Connected Layer - FC):

1.2.4.5.1. ชั้นสุดท้ายที่เป็น Linear layer ที่เชื่อมต่อกับทุกโหนดในชั้นก่อนหน้า

1.2.4.5.2. แปลงเวกเตอร์ขนาด mid_ch ให้เป็นเวกเตอร์เอาต์พุตขนาด num_classes (6)

1.2.4.5.3. เอาต์พุตนี้โดยปกติจะถูกป้อนเข้าสู่ softmax function ในภายหลังเพื่อคำนวณความน่าจะเป็นของแต่ละคลาส (แม้ว่าจะไม่ได้แสดงในโค้ดนี้ แต่เป็นวิธีปฏิบัติทั่วไปสำหรับงานจำแนกประเภท)

1.3. อธิบายโค้ด

1.3.1. การเตรียมข้อมูล (Data pipeline)

1.3.1.1. เลือกและสร้างโฟลเดอร์ train/val/test จากชุด dataset “HaGRID Sample 30k 384p” ที่บันทึกลงใน Google Drive

1.3.1.1.1. ค่าตั้งต้น: DATA_ROOT="/content/drive/MyDrive/hagrid-sample-30k-384p", โฟลเดอร์ภาพอยู่ที่ hagrid_30k/

1.3.1.1.2. แบ่งสัดส่วน: test = 20% แล้วเอาครึ่งของ test ไปทำ val ⇒ val ≈ 10%, test ≈ 10%

1.3.1.1.3. เลือกเฉพาะ 6 คลาสที่ต้องการประกอบด้วย {"like","dislike","fist","palm","peace","ok","okay"}

1.3.1.2. ฟังก์ชันหลัก: build_transforms, build_dataloaders

1.3.1.2.1. **Train transforms**: RandomResizedCrop, HorizontalFlip, ColorJitter, ToTensor, Normalize(ImageNet mean/std)

1.3.1.2.2. **Eval/Test transforms**: Resize/CenterCrop (หรือ Resize คงสเกล) + Normalize

1.3.1.2.3. ใช้ torchvision.datasets.ImageFolder + DataLoader (มี pin_memory, persistent_workers)

1.3.2. ส่วนสร้างโมเดล (Model)

1.3.2.1. คลาส: ConvBNAct, SEBlock, GestureHead, GestureResNet18Custom

1.3.2.2. ฟังก์ชัน: build_model(num_classes, use_pretrained=True, mid_ch=256, drop_p=0.2)

1.3.2.3. มี set_backbone_trainable(model, trainable) สำหรับ freeze/unfreeze backbone

1.3.3. ส่วนเทรน (Train Loop)

1.3.3.1. ฟังก์ชัน:

1.3.3.1.1. run_epoch(model, loader, ...) ⟶ วน batch, AMP (torch.amp.autocast / fallback torch.cuda.amp), CrossEntropyLoss, AdamW, optional grad clipping

1.3.3.1.2. train_model(model, dls, device, epochs=15, lr=3e-4, weight_decay=1e-4, freeze_epochs=3, early_patience=5, clip_norm=1.0, out_dir=...)

1.3.3.1.2.1. **Phase 1**: freeze backbone แล้วเทรนเฉพาะ head

1.3.3.1.2.2. **Phase 2**: unfreeze (เมื่อ epoch == freeze_epochs+1) แล้ว train ทั้งโมเดล

1.3.3.1.2.3. Scheduler: ReduceLROnPlateau (ตรวจ val_loss)

1.3.3.1.2.4. **Early Stopping** จาก val_loss พร้อมบันทึก best.pt

1.3.3.1.2.5. เก็บ hist = {train_loss, val_loss, train_acc, val_acc} และ plot กราฟ loss_curve.png, acc_curve.png

1.3.4. ส่วนประเมินผล (Evaluate & Report)

1.3.4.1. evaluate_and_report(model, test_loader, classes, device, out_dir)

1.3.4.1.1. คำนวณ Accuracy, Precision/Recall/F1 (macro & per class) ด้วย sklearn.metrics

1.3.4.1.2. สร้าง Confusion Matrix ทั้ง นับจริง และ normalized (cm_test_counts.png, cm_test_norm.png)

1.3.4.1.3. สร้าง classification_report_test.(txt|csv) และ test_summary.txt

1.3.5. ส่วนต่อเชื่อม/ส่งออกไปโมเดลอื่น

1.3.5.1. มี Export ONNX: torch.onnx.export(...) สร้าง model.onnx เพื่อใช้ใน runtime อื่น ๆ

1.4. อธิบายวิธีการเทรนโมเดล

1.4.1. เตรียมโครงสร้างโฟลเดอร์ภาพของ HaGRID 30k ไว้ใน Google Drive ตามพาธที่ระบุ

1.4.2. รันโค้ดเพื่อสร้าง train/val/test (symlink/copy/move ได้ตาม MODE)

1.4.3. ตั้งค่าหลัก (ค่าดีฟอลต์ในไฟล์):

1.4.3.1. IMG_SIZE=224, BATCH=32, EPOCHS=15, LR=3e-4, WEIGHT_DECAY=1e-4

1.4.3.2. MID_CH=256, DROP_P=0.2, FREEZE_EPOCHS=3, EARLY_PATIENCE=5, CLIP_NORM=1.0

1.4.3.3. USE_PRETRAINED=True

1.4.4. สร้าง DataLoader ด้วย build_dataloaders(DATA_ROOT, IMG_SIZE, BATCH, NUM_WORKERS)

1.4.5. สร้างโมเดลด้วย build_model(num_classes=len(classes), ...) แล้ว model.to(device)

1.4.6. ฝึกด้วย train_model(...) (freeze⟶unfreeze) จน early stop พร้อมบันทึกกราฟ loss_curve.png, acc_curve.png และ checkpoint best.pt

1.4.7. ประเมินด้วย evaluate_and_report(...) ได้ไฟล์ classification_report_test.(txt|csv), cm_test_counts.png, cm_test_norm.png, test_summary.txt

1.4.8. (ตัวเลือกเสริม) ส่งออก model.onnx

1.5. อธิบาย dataset

1.5.1. แหล่งข้อมูล : HaGRID Sample 30k 384p (Kaggle) เซฟลงใน drive

1.5.2. เลือกคลาส like, dislike, fist, ok, palm, peace เพื่อนำมาเทรนและทดสอบโมเดล

1.5.3. แบ่งชุดทดสอบและฝึกโมเดล :

1.5.3.1. แบบที่หนึ่ง: ตั้ง TEST=0.20 แล้วนำ ครึ่งหนึ่งของ test ไปเป็น val ⟶ ได้ train≈80% / val≈10% / test≈10% (ใช้วางไฟล์ด้วยโหมด symlink/copy/move)

1.5.3.2. แบบที่สอง : สุ่มตรงๆ เป็น 80/10/10 แล้วคัดลอกไฟล์จริงไปเก็บถาวรบน drive

1.6. การประเมินผลและ metric ที่ใช้

1.6.1. Loss: Cross-Entropy Loss (ทั้ง train/val เก็บเป็นกราฟ)

1.6.2. Metrics:

1.6.2.1. Accuracy (train/val/test)

1.6.2.2. Macro Precision/Recall/F1 (รวมทั้ง per-class report)

1.6.3. Visualization:

1.6.3.1. Confusion Matrix (counts/normalized)

1.6.3.2. กราฟ loss/acc

1.6.3.3. ตาราง per-class metrics

1.6.3.4. PR Curve/PR AUC (macro AP)

1.6.4. รูปภาพ ไฟล์ประกอบผลลัพธ์ จัดเก็บอยู่ในโฟลเดอร์ /content/run_outputs หลังจากการฝึกโมเดล :

1.6.4.1. loss_curve.png

1.6.4.2. acc_curve.png

1.6.4.3. classification_report_test.txt

1.6.4.4. classification_report_test.csv

1.6.4.5. cm_test_counts.png

1.6.4.6. cm_test_norm.png

1.6.4.7. test_summary.txt

1.6.4.8. best.pt

1.6.4.9. classes.json

1.6.4.10. history.json

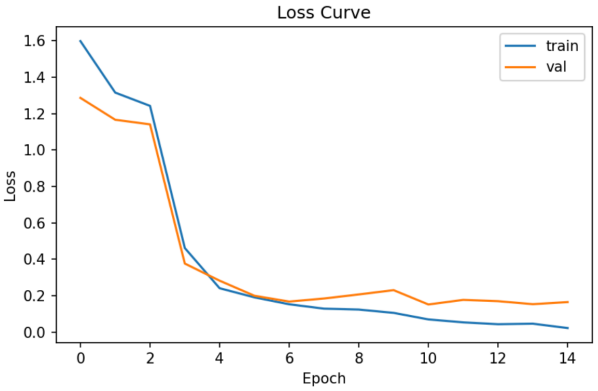
1.6.4.11. train_summary.txt

1.7. รูปภาพประกอบผลลัพธ์

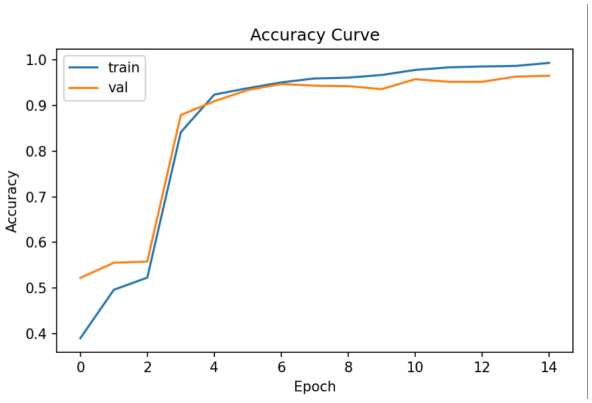
1.7.1. รูปแสดงจำนวนโหนด weight bias

1.7.2. รูปภาพประกอบผลลัพธ์การเทรน

1.7.2.1. กราฟแสดง Loss curve

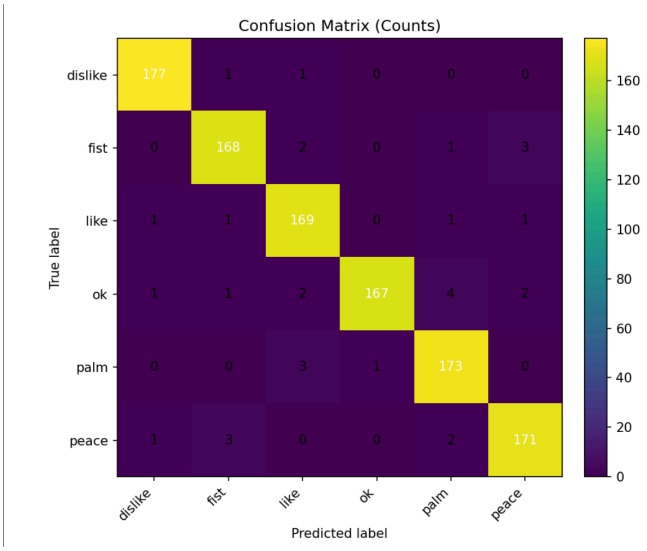


1.7.2.2. กราฟแสดง Accuracy curve

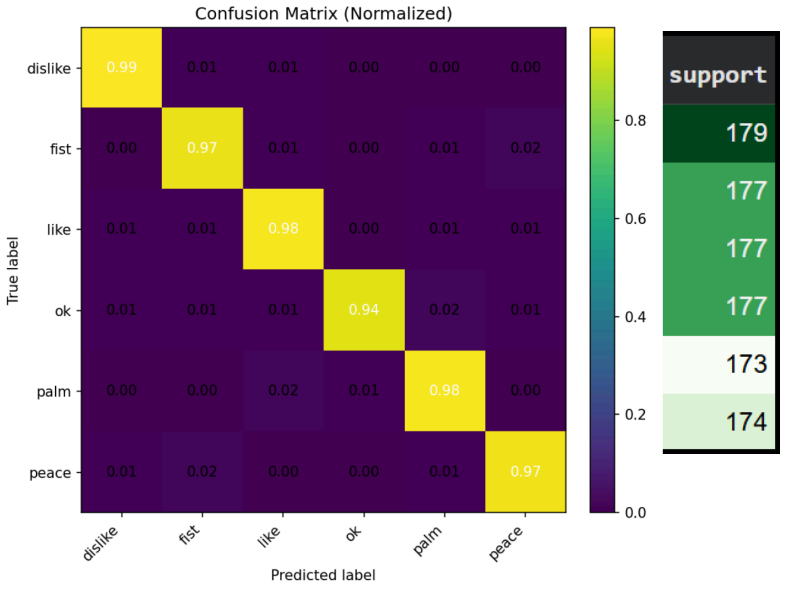


1.7.2.3. กราฟแสดง Confusion Matrix (Counts)

1.7.2.4. กราฟแสดง Confusion Matrix (Normalized)

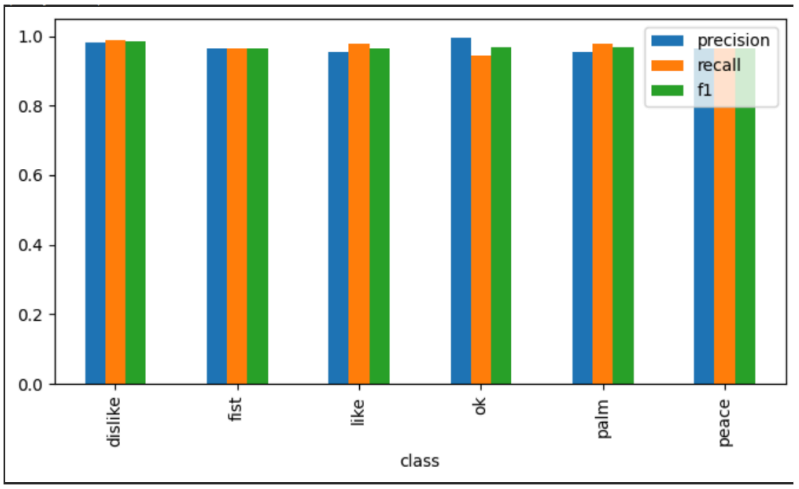


1.7.3. รูปภาพประกอบผลลัพธ์การทดสอบ

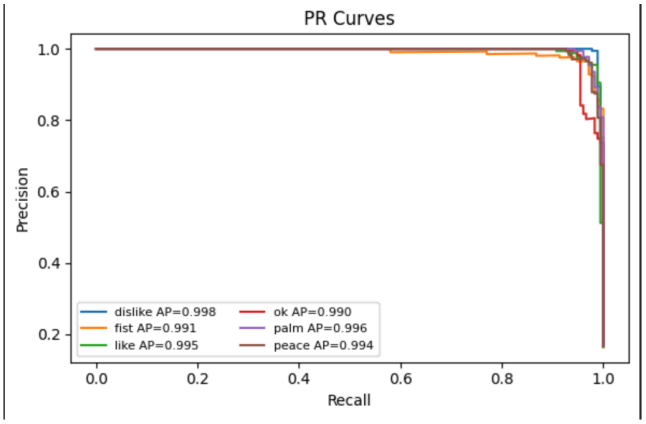


1.7.3.1. ตาราง per-class

1.7.3.2. กราฟแท่ง



1.7.3.3. PR curves



2. อธิบายบทความอ้างอิงและงานที่เกี่ยวข้อง

2.1. ResNet-18 (Backbone มาตรฐานใน torchvision.models)

2.2. Squeeze-and-Excitation (SE) Block สำหรับ channel attention

2.3. ONNX Export สำหรับนำโมเดลไปใช้งานข้ามเฟรมเวิร์ก/รันไทม์

2.4. Dataset: HaGRID Sample 30k 384p (Kaggle)

https://www.kaggle.com/datasets/innominate817/hagrid-sample-30k-384p/data

3. เปอร์เซ็นงานของแต่ละคนที่ทำ

3.1. นางสาวเปรมมิกา เนียมเปรม

3.1.1. ออกแบบสถาปัตยกรรมhead custom (Conv3×3⟶BN⟶ReLU⟶SE(r=16)⟶GAP⟶Dropout⟶FC) — 15%

3.1.2. ประกอบกับ ResNet18 backbone + ฟังก์ชัน build_model, set_backbone_trainable — 8%

3.1.3. เทรนโมเดล: freeze⟶unfreeze, AdamW, ReduceLROnPlateau, AMP, grad clipping, early stopping — 12%

3.1.4. ประเมินผล: Accuracy, Macro-P/R/F1, confusion matrix, per-class report — 8%

3.1.5. วิชวลไลซ์/ความน่าเชื่อถือ: loss/acc curves, PR curves/mAP, misclassified gallery, (optional) ECE/Grad-CAM — 5%

3.1.6. เอกสาร/รายงานฝั่ง Model: ผังสถาปัตยกรรม, จำนวนพารามิเตอร์, เหตุผลเลือก hyperparams — 2%

3.2. นางสาววริษฐา ภิรมย์เกียรติ์

3.2.1. จัดการชุดข้อมูล & สปลิต (train/val/test ≈ 80/10/10, รวม ok/okay) — 15%

3.2.2. Data hygiene: ตรวจซ้ำ/ไฟล์เสีย/เลเบลผิด, สมดุลคลาสเบื้องต้น — 8%

3.2.3. Data pipeline: เขียน build_transforms, build_dataloaders, tune num_workers/pin_memory — 10%

3.2.4. Augmentation: ออกแบบ RandomResizedCrop/Flip/ColorJitter (อธิบายเหตุผล) — 7%

3.2.5. วิเคราะห์ข้อมูล: class distribution, ตัวอย่างภาพ, สถิติขนาดภาพ — 5%

3.2.6. เอกสาร/รายงานฝั่ง Data: วิธีเตรียม, แผนผังโฟลเดอร์, เหตุผลเลือก augment/split — 5%

4. ภาคผนวก

4.1. คำสั่งรัน (Colab/Notebook)

4.1.1. ติดตั้งและเมานต์ไดรฟ์ จากนั้นปรับ DATA_ROOT ให้ถูกต้อง

4.1.2. รันส่วน สร้างสปลิต จนได้โฟลเดอร์ train/val/test

4.1.3. รันส่วน Config ⟶ Data Pipeline ⟶ Model ⟶ Train Loop ⟶ Evaluation ⟶ Export ONNX ตามลำดับ

4.2. ไฟล์ที่ต้องส่ง

4.2.1. ไฟล์ final_report.pdf ⟶ ไฟล์นี้

4.2.2. Github link

4.2.2.1. ไฟล์โค้ด PyTorch final_project.ipynb

4.2.2.2. ไฟล์ README.md ที่แปลงจาก final_report.pdf