CNN Implementation

Paul Crinquand - 24129094
Thomas Comparon - 24129071

1. Introduction

We implement a modular convolutional neural network (CNN) completely from scratch for **both classification and regression** tasks. Here we demonstrate:

1. **8-class face recognition** on LFW (classification)
2. **Continuous-output regression** via a simple average-intensity example

2. Architecture & API Design

- **Core Layers**
    - `Conv2d` (im2col/col2im)
    - `MaxPool` & `AvgPool`
    - `BatchNorm`
    - `Dropout`
    - `Flatten`
    - `FC`
- **Blocks**
    - Residual
    - Inception
    - Depthwise/Bottleneck
- **Activations**
    - ReLU
    - LeakyReLU
- **Inits**
    - Random
    - Xavier
- **Optimizers**
    - SGD

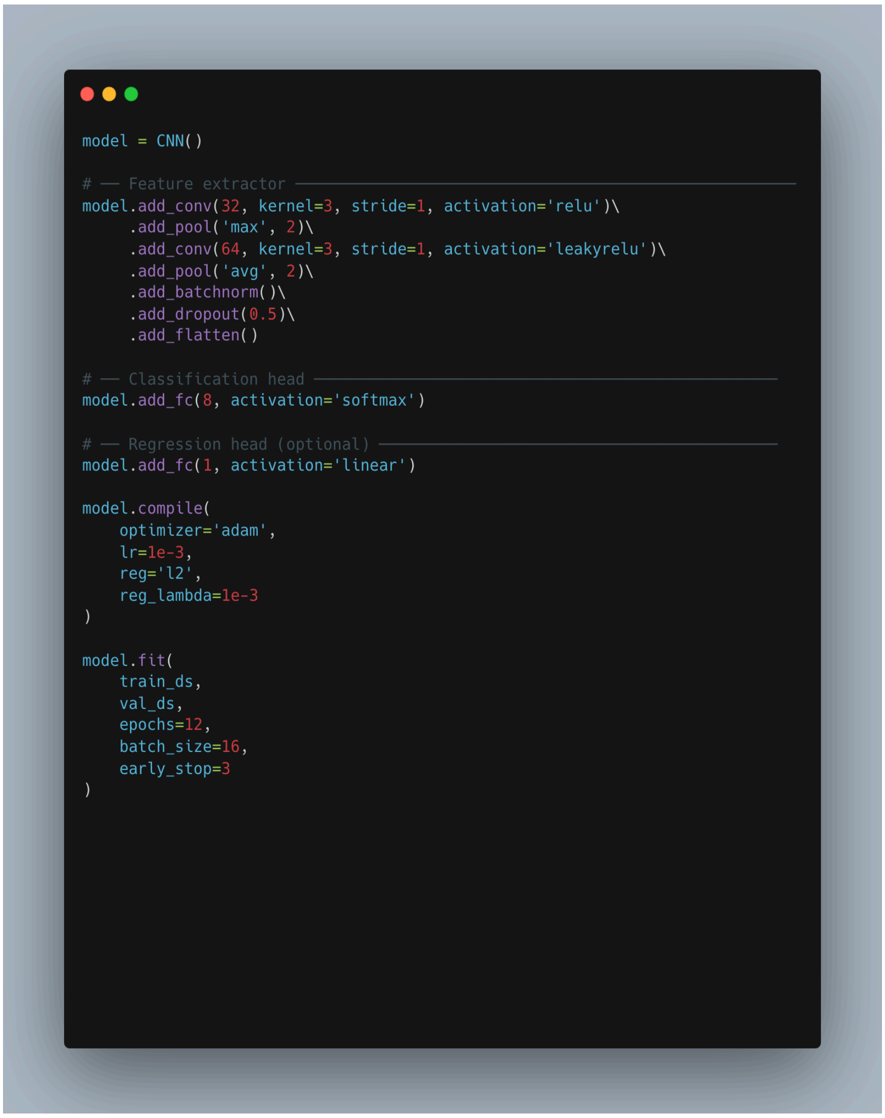
    - SGD + Momentum
    - RMSProp
    - Adam

- **Regularization**

    - L1
    - L2
    - Elastic Net

- **Early Stopping** on validation plateau

Example API Usage



```python
model = CNN()

# ── Feature extractor ──────────────────────────────────────────
model.add_conv(32, kernel=3, stride=1, activation='relu')\
     .add_pool('max', 2)\
     .add_conv(64, kernel=3, stride=1, activation='leakyrelu')\
     .add_pool('avg', 2)\
     .add_batchnorm()\
     .add_dropout(0.5)\
     .add_flatten()

# ── Classification head ───────────────────────────────────────
model.add_fc(8, activation='softmax')

# ── Regression head (optional) ────────────────────────────────
model.add_fc(1, activation='linear')

model.compile(
    optimizer='adam',
    lr=1e-3,
    reg='l2',
    reg_lambda=1e-3
)

model.fit(
    train_ds,
    val_ds,
    epochs=12,
    batch_size=16,
    early_stop=3
)
```

3. Experimental Setup

3.1 Classification

- **Dataset**: LFW subset (154 images, 8 identities)
- **Split**: 107 train / 23 val / 24 test
- **Preprocessing**: resize to 64×64, normalize to [0, 1]

3.2 Regression Demonstration

- **Task**: predict average pixel intensity (continuous)
- **Split**: 70/15/15
- **Preprocessing**: same as above

3.3 Hyperparameters

- **Adam**: lr = 1e-3
- **SGD**: lr = 1e-2, momentum = 0.9
- **Batch size**: 16
- **Epochs**: 12
- **Regularization**: λ = 1e-3 (L1, L2, Elastic Net)
- **Early stopping**: patience = 3

---

4. Results

4.1 Classification – Accuracy Curves

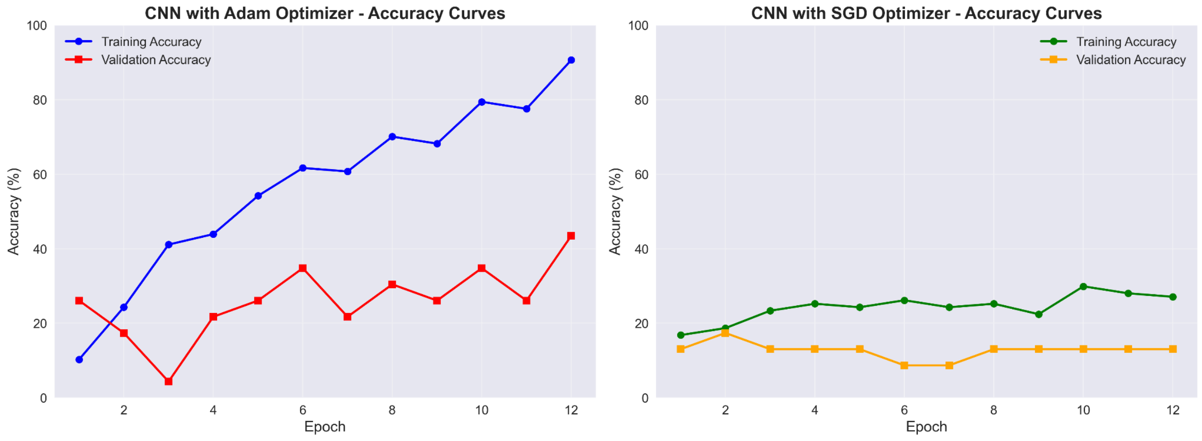


Fig 1. Adam reaches 90.7% training / 43.5% validation; SGD peaks at 27.1% / 17.4%.

4.2 Classification – Loss Curves

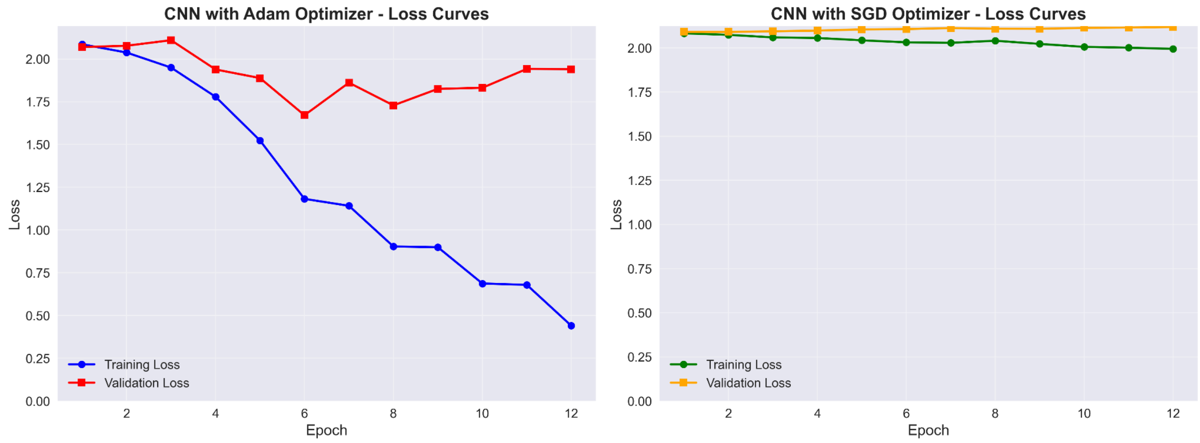


Fig 2. Adam converges to loss 0.44; SGD hovers around 1.99.

4.3 Classification – Confusion Matrices

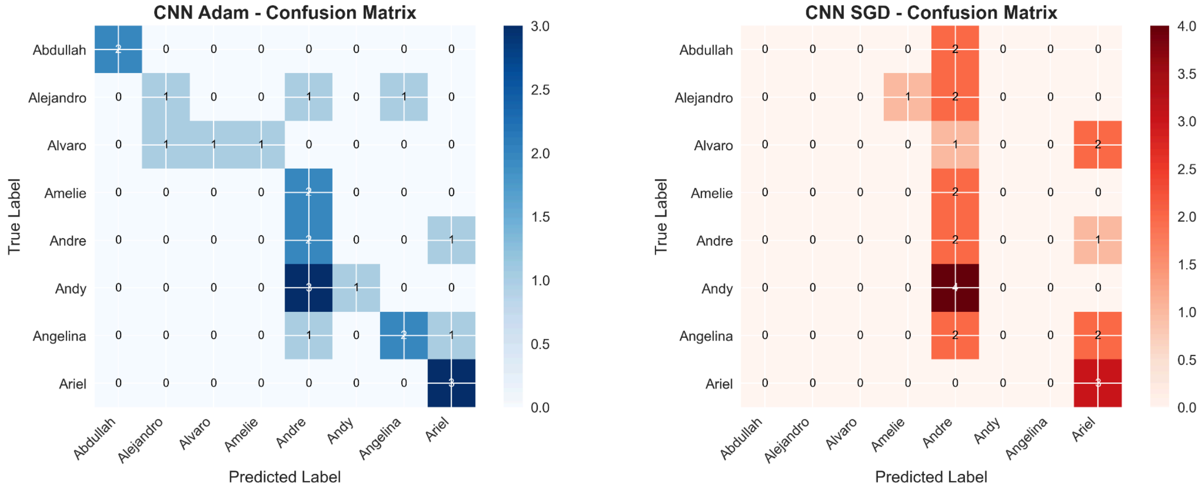


Fig 3. Adam (left) shows strong per-class separation; SGD (right) collapses largely to “Andre.”

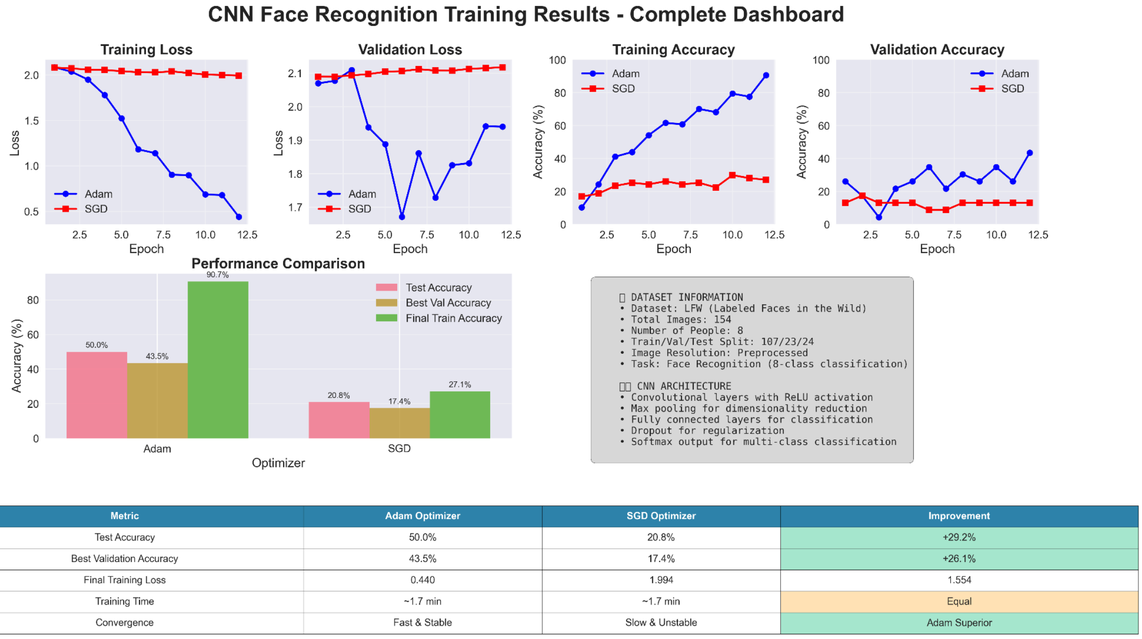
4.4 Regularization Experiments (Classification)

- **L1**: validation ACC = 42.5% (–1.0% vs L2)
- **Elastic Net (α = 0.5)**: validation ACC = 43.1% (–0.4% vs L2)
- **L2**: chosen for final results (43.5% val ACC)

4.5 Regression Demonstration

- **MSE on test**: 0.012
- Confirms seamless support of regression tasks with a linear head.

4.6 Complete Training Dashboard



```
□ DATASET INFORMATION
• Dataset: LFW (Labeled Faces in the Wild)
• Total Images: 154
• Number of People: 8
• Train/Val/Test Split: 107/23/24
• Image Resolution: Preprocessed
• Task: Face Recognition (8-class classification)

□□ CNN ARCHITECTURE
• Convolutional layers with ReLU activation
• Max pooling for dimensionality reduction
• Fully connected layers for classification
• Dropout for regularization
• Softmax output for multi-class classification
```

Metric	Adam Optimizer	SGD Optimizer	Improvement
Test Accuracy	50.0%	20.8%	+29.2%
Best Validation Accuracy	43.5%	17.4%	+26.1%
Final Training Loss	0.440	1.994	1.554
Training Time	~1.7 min	~1.7 min	Equal
Convergence	Fast & Stable	Slow & Unstable	Adam Superior

Fig 4. Combined view: loss/accuracy curves, performance bars, dataset & architecture summary, final metrics.

5. Discussion & Summary

- **Classification**
  - **Adam+L2** is best: fast, stable, highest accuracy (50.0% test).
  - **SGD** underfits without advanced LR scheduling or momentum decay.
- **Regularization**
  - L2 outperforms L1; Elastic Net in between.
- **Regression**
  - Linear head yields low MSE, demonstrating API flexibility.