Cow Face ID – End-to-End Model (YOLOv5 + ViT + ArcFace)

This repo-style guide gives you **ready-to-run code** to replicate the dairy-cow facial identification system described in your paper: **YOLOv5 for face detection** + **ViT-small with ArcFace** for identification. Copy the code blocks into files with the indicated filenames, follow the steps, and you'll have training + real-time inference.

---

0) Project Structure

```
cow-face-id/
├─ env.yml                    # (optional) conda environment
├─ requirements.txt           # pip deps
├─ README.md                  # (this file)
├─ data/
│  ├─ detection/              # YOLO dataset (images/, labels/)
│  │  ├─ images/{train,val,test}/
│  │  └─ labels/{train,val,test}/
│  └─ recognition/            # cropped faces (class folders) for ViT
│     ├─ train/<cow_id>/*.jpg
│     ├─ val/<cow_id>/*.jpg
│     └─ test/<cow_id>/*.jpg
├─ scripts/
│  ├─ build_crops_from_yolo.py
│  ├─ train_recognition_vit_arcface.py
│  ├─ infer_realtime.py
│  └─ utils_arcface.py
└─ yolo/                      # YOLOv5 repo will live here (git clone)
```

> **Tip:** Put your raw barn videos/jpegs anywhere (e.g., `raw/`). Create YOLO labels for faces. Then run the crop builder to generate the recognition dataset.

---

1) Environment & Dependencies

Option A: Conda

```
# env.yml
name: cow-face-id
channels:
  - pytorch
  - nvidia
  - conda-forge
dependencies:
  - python=3.10
  - pytorch
  - torchvision
  - torchaudio
  - pytorch-cuda=12.1  # if you have NVIDIA GPU
  - pip
  - pip:
      - -r requirements.txt
```

Pip requirements

```
# requirements.txt
opencv-python
numpy
pandas
scikit-learn
matplotlib
tqdm
albumentations
einops
torchmetrics
```

> **YOLOv5** is used via its own repo scripts for training/inference.

Install & clone YOLOv5:

```
conda env create -f env.yml  # or: python -m venv .venv && source .venv/bin/
activate
conda activate cow-face-id

# clone YOLOv5 into ./yolo
cd yolo
git clone https://github.com/ultralytics/yolov5 .
pip install -r requirements.txt  # of YOLOv5 repo
cd ..

# install this project's reqs
pip install -r requirements.txt
```

---

2) Prepare Data

2.1 YOLO Detection Dataset

- **Goal:** Detect cow faces in general frames.
- **Format:** Standard YOLO: one `.txt` per image with lines: `class x y w h` (normalized).
- **Split:** `train/`, `val/`, `test/` under both `images/` and `labels/`.

Example YAML for YOLOv5:

```yaml
# data/detection/cow_face.yaml
path: ../data/detection
train: images/train
val: images/val
names:
  0: cow_face
```

2.2 Build Recognition Dataset (COW77-like)

Run the script below to **crop faces** from detection labels and build folders per cow id (you provide a CSV mapping image→cow_id).

Create file: `scripts/build_crops_from_yolo.py`

```python
import os
import cv2
import csv
from pathlib import Path

IMAGES_ROOT = Path('data/detection/images')  # source full frames
LABELS_ROOT = Path('data/detection/labels')
OUT_ROOT = Path('data/recognition')
SPLITS = ['train', 'val', 'test']

# CSV mapping for identity labels per frame: image_name,cow_id,split
# Example row: DSC_000123.jpg, COW_017, train
ID_CSV = Path('data/identity_map.csv')

# crops will be resized to 224x224 for ViT
TARGET = 224

os.makedirs(OUT_ROOT, exist_ok=True)

# helper to convert normalized xywh -> pixel xyxy
def xywhn_to_xyxy(label_path, img_w, img_h):
    boxes = []
    with open(label_path, 'r') as f:
        for line in f:
            parts = line.strip().split()
            if len(parts) != 5:
                continue
            cls, x, y, w, h = parts
            x, y, w, h = map(float, (x, y, w, h))
            x1 = int((x - w/2) * img_w)
            y1 = int((y - h/2) * img_h)
            x2 = int((x + w/2) * img_w)
            y2 = int((y + h/2) * img_h)
            boxes.append((x1, y1, x2, y2))
    return boxes

# read identity map
meta = {}
with open(ID_CSV, 'r', newline='') as f:
    reader = csv.DictReader(f)
    for r in reader:
        meta[r['image_name']] = (r['cow_id'], r['split'])

for split in SPLITS:
    img_dir = IMAGES_ROOT / split
    for img_name in os.listdir(img_dir):
        if not img_name.lower().endswith(('.jpg','.jpeg','.png')):
            continue
        if img_name not in meta:
            # skip frames not labeled with identity
            continue
        cow_id, split2 = meta[img_name]
        # allow override split from CSV
        split_out = split2 if split2 in SPLITS else split

        img_path = img_dir / img_name
        lbl_path = LABELS_ROOT / split / (Path(img_name).stem + '.txt')
        if not lbl_path.exists():
            continue

        img = cv2.imread(str(img_path))
        if img is None:
            continue
        h, w = img.shape[:2]
        boxes = xywhn_to_xyxy(lbl_path, w, h)
        if not boxes:
            continue

        # Pick the largest face box (closest cow) or loop over all
        for (x1,y1,x2,y2) in boxes:
            x1 = max(0, x1); y1 = max(0, y1)
            x2 = min(w-1, x2); y2 = min(h-1, y2)
            if x2 <= x1 or y2 <= y1:
                continue
            crop = img[y1:y2, x1:x2]
            crop = cv2.resize(crop, (TARGET, TARGET),
interpolation=cv2.INTER_AREA)
            out_dir = OUT_ROOT / split_out / cow_id
            out_dir.mkdir(parents=True, exist_ok=True)
            out_name = out_dir / f"{Path(img_name).stem}_{x1}_{y1}.jpg"
            cv2.imwrite(str(out_name), crop)
```

Run:

```
python scripts/build_crops_from_yolo.py
```

---

3) Train YOLOv5 Face Detector

From the `yolo/` folder you cloned:

```
cd yolo
python train.py \
  --img 640 \
  --batch 16 \
  --epochs 150 \
  --data ../data/detection/cow_face.yaml \
  --weights yolov5s.pt \
  --project runs/cow_face \
  --name y5s_detect
cd ..
```

This produces weights like `yolo/runs/cow_face/y5s_detect/weights/best.pt`.

> You can later run detection to create more crops if needed:

```
cd yolo
python detect.py --weights runs/cow_face/y5s_detect/weights/best.pt \
  --img 640 --source ../data/detection/images/val --save-txt --save-conf
cd ..
```

---

4) Train ViT-small + ArcFace for Recognition

Create file: `scripts/utils_arcface.py`

```python
import math
import torch
import torch.nn as nn

class L2Norm(nn.Module):
    def __init__(self, dim=1, eps=1e-12):
        super().__init__()
        self.dim = dim
        self.eps = eps
    def forward(self, x):
        return torch.nn.functional.normalize(x, p=2, dim=self.dim,
eps=self.eps)
```

```
class ArcMarginProduct(nn.Module):
    """Implements ArcFace linear head.
    Args:
        in_features: embedding dim
        out_features: num classes
        s: scale factor
        m: additive angular margin
    """
    def __init__(self, in_features, out_features, s=10.0, m=0.25,
easy_margin=False):
        super().__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.s = s
        self.m = m
        self.weight = nn.Parameter(torch.FloatTensor(out_features,
in_features))
        nn.init.xavier_uniform_(self.weight)
        self.easy_margin = easy_margin
        self.cos_m = math.cos(m)
        self.sin_m = math.sin(m)
        self.th = math.cos(math.pi - m)
        self.mm = math.sin(math.pi - m) * m

    def forward(self, embeddings, labels):
        # normalize features and weights
        embeddings = torch.nn.functional.normalize(embeddings, p=2, dim=1)
        W = torch.nn.functional.normalize(self.weight, p=2, dim=1)
        cosine = torch.matmul(embeddings, W.t())
        sine = torch.sqrt(1.0 - torch.clamp(cosine ** 2, 0, 1))
        phi = cosine * self.cos_m - sine * self.sin_m
        if self.easy_margin:
            phi = torch.where(cosine > 0, phi, cosine)
        else:
            phi = torch.where(cosine > self.th, phi, cosine - self.mm)
        one_hot = torch.zeros_like(cosine)
        one_hot.scatter_(1, labels.view(-1, 1), 1.0)
        logits = (one_hot * phi) + ((1.0 - one_hot) * cosine)
        logits *= self.s
        return logits
```

Create file: `scripts/train_recognition_vit_arcface.py`

```
import os
from pathlib import Path
import random
import numpy as np
from tqdm import tqdm

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import transforms, datasets

from utils_arcface import ArcMarginProduct

# Config
DATA_ROOT = Path('data/recognition')
IMG_SIZE = 224
BATCH_SIZE = 64
EPOCHS = 150
LR = 0.05
MOMENTUM = 0.9
WD = 1e-4
SCALE = 10.0
MARGIN = 0.25
SEED = 42

random.seed(SEED); np.random.seed(SEED); torch.manual_seed(SEED)

def build_transforms():
    train_tf = transforms.Compose([
        transforms.Resize((IMG_SIZE, IMG_SIZE)),
        transforms.RandomApply([transforms.ColorJitter(0.2,0.2,0.2,0.1)],
p=0.5),
        transforms.RandomHorizontalFlip(p=0.2),
        transforms.RandomRotation(10),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
0.225])
    ])
    test_tf = transforms.Compose([
        transforms.Resize((IMG_SIZE, IMG_SIZE)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
0.225])
    ])
    return train_tf, test_tf


def create_loaders():
    train_tf, test_tf = build_transforms()
    train_ds = datasets.ImageFolder(DATA_ROOT / 'train', transform=train_tf)
    val_ds   = datasets.ImageFolder(DATA_ROOT / 'val', transform=test_tf)
    test_ds  = datasets.ImageFolder(DATA_ROOT / 'test', transform=test_tf)

    train_loader = DataLoader(train_ds, batch_size=BATCH_SIZE, shuffle=True,
num_workers=4, pin_memory=True)
    val_loader   = DataLoader(val_ds, batch_size=BATCH_SIZE, shuffle=False,
num_workers=4, pin_memory=True)
```

```
    test_loader  = DataLoader(test_ds, batch_size=BATCH_SIZE, shuffle=False,
num_workers=4, pin_memory=True)
    return train_loader, val_loader, test_loader, train_ds.classes


class ViTSmall(nn.Module):
    def __init__(self, embed_dim=384, num_classes=1000):
        super().__init__()
        # Use torchvision's ViT-S/16
        from torchvision.models.vision_transformer import vit_b_16,
ViT_B_16_Weights, vit_s_16, ViT_S_16_Weights
        self.backbone = vit_s_16(weights=ViT_S_16_Weights.IMAGENET1K_V1)
        # replace head with identity to output embeddings
        self.embedding_dim = self.backbone.heads.head.in_features
        self.backbone.heads.head = nn.Identity()

    def forward(self, x):
        return self.backbone(x)


def evaluate(model, head, loader, device):
    model.eval(); head.eval()
    correct = 0; total = 0
    with torch.no_grad():
        for x, y in loader:
```