**ELEC 475 Lab 2**

**Name 1:** Matteo Van Der Platt; **Student number:** INSERT

**Name 2:** Jacob Badali; **Student number**: 20290739

# Network Architecture

The network architecture can be seen in the bullet points below.

* **Input**
  + RGB image tensor of shape 3×227×227.
  + Images are resized to 227×227; target nose coordinates (u, v) are geometrically scaled to the same 227×227 space.
  + Pixel value normalization: default [0,1] scaling (ToTensor); optional ImageNet mean/std normalization is supported but not required for SnoutNet.
* **Convolutional backbone**
  + All convolutions use kernel size 3×3, stride 1, padding 1; ReLU after each conv; no batch norm; no dropout.
  + Max pooling uses kernel size 4×4, stride 4, with ceil\_mode=True (this is important to match the exact downstream feature map size).
  + Layer-by-layer with spatial sizes (H×W×C), starting from 227×227×3:
    1. Conv1: 3→64, 3×3, pad=1 → ReLU → MaxPool(4×4, s=4, ceil)  
       227×227×3 → 227×227×64 → 57×57×64
    2. Conv2: 64→128, 3×3, pad=1 → ReLU → MaxPool(4×4, s=4, ceil)  
       57×57×64 → 57×57×128 → 15×15×128
    3. Conv3: 128→256, 3×3, pad=1 → ReLU → MaxPool(4×4, s=4, ceil)  
       15×15×128 → 15×15×256 → 4×4×256
* Head (fully connected regression)
  + Flatten: 4×4×256 = 4096 features.
  + FC1: 4096 → 1024, ReLU.
  + FC2: 1024 → 1024, ReLU.
  + FC3: 1024 → 2 (linear output).
  + No activation after the final layer; outputs are the predicted pixel coordinates (û, v̂) in the 227×227 image coordinate system (origin at top-left).
* Output and loss
  + Output: 2-D vector per image: (û, v̂), in pixel units aligned with the resized image.
  + Training loss: Mean Squared Error (MSE) between predicted and ground-truth (u, v) coordinates (both in the resized 227×227 space).
* Notes and rationale
  + The use of ceil\_mode=True in 4×4 pooling is critical to get exact feature map sizes of 57→15→4, which in turn ensures the flatten dimension is exactly 4096 for FC1.
  + No dropout or batch normalization are used in SnoutNet.
  + Parameter count is approximately 5.62 million.

# Dataset Implementation + Sanity Check

## Dataset Implementation

The dataset is implemented in dataset.py, and the dataloader is implemented in the datamodule.py, in the ‘get\_dataloaders’ function.

* **Data format**
  + Labels are provided as lines of the form: “filename, (u, v)”, where u is the horizontal pixel coordinate and v is the vertical pixel coordinate in the original image.
  + The parser accepts both a strict quoted format and a tolerant fallback without quotes. Each line is parsed into a tuple (filename, u, v). Invalid lines or missing files cause a clear error early.
* **Image loading and coordinate handling**
  + For a given index, the image is loaded and converted to RGB.
  + The image is deterministically resized to a square of S×S pixels (S = 227).
  + The target coordinates are scaled to match the resized image using the original image dimensions once the image is transformed (W, H):
    - u' = u × (S / W)
    - v' = v × (S / H)
  + The coordinate system uses the top-left corner as the origin; u increases to the right and v increases downward. Returned coordinates are in pixel units of the resized S×S space.
* **Geometry-safe augmentation**
  + A horizontal flip can be applied with probability p during training-time augmentation.
  + When a flip occurs, the image is mirrored left-right and the target is updated consistently:
    - u' = (S − 1) − u
    - v' = v
  + Because resizing to S×S occurs first, the flip formula is simple and consistent.
* **Tensor conversion and normalization**
  + The image is converted to a float tensor with values in [0, 1].
  + Optionally, ImageNet mean/std normalization can be applied (mean = [0.485, 0.456, 0.406]; std = [0.229, 0.224, 0.225]) to match backbones that expect it. This is used in AlexNet and VGGNet backbones.
  + The dataset returns:
    - Image tensor of shape (3, S, S)
    - Target tensor of shape (2,) containing (u, v) in pixel units of the resized image.
* **DataLoader setup**
  + Separate train and test splits are constructed using the same resizing and normalization, with augmentation enabled only for the training split.
  + The training loader shuffles samples; the test loader does not.
  + Batch size and number of workers are configurable, and pinned memory is enabled for efficient host-to-device transfers.

## Sanity Check

The sanity check was implemented to check the coordinates, ensuring they are manipulated correctly. It can be found in the sanity\_check.py file.

* **Shape and range checks**
  + Print the dataset sizes for both splits.
  + Randomly sample a few items from each split and verify:
    - Image tensor shape is (3, 227, 227)
    - Target is a length-2 tensor (u, v)
    - Bounds: 0 ≤ u < 227 and 0 ≤ v < 227
  + If using augmentation with horizontal flips, verify the flip invariant on paired examples:
    - Before/after flip, u + u\_flipped ≈ S − 1 (and v remains the same).
* **Visual spot checks**
  + Save a couple of samples to disk by converting the tensor back to an image and overlaying a small dot at (u, v).
  + Inspect a few training and test images to confirm that the dot lands on the nose after resizing and any flips.
* **DataLoader integrity**
  + Iterate one or two batches from the training loader and confirm batch shapes:
    - Images: (B, 3, 227, 227)
    - Targets: (B, 2)
  + Confirm that shuffling is active for training and disabled for testing by sampling indices across two consecutive training batches.
* **Failure modes caught early**
  + Missing label/image files cause immediate, readable errors.
  + Malformed label lines are rejected with a clear parse error.
  + Any out-of-bounds target after transforms triggers an assertion during the sanity pass.

The sanity check saves to ‘\_vis\_check’. Images with red dots on snouts can be seen in this folder.

# Snoutnet Hyperparameters/Hardware/Time

We trained SnoutNet as a coordinate regression model that maps 227×227 RGB images to two pixel coordinates (u, v) identifying the nose location. Images were resized to 227×227 and their ground-truth coordinates were scaled into the same pixel space before being fed to the network. During training, we optimized the mean squared error (MSE) between predicted and ground-truth coordinates using the Adam optimizer with an initial learning rate of 1e-3 and no weight decay. To encourage steady convergence, we employed a ReduceLROnPlateau scheduler that halves the learning rate when the validation loss plateaus (patience of five epochs). We trained for 50 epochs with a batch size of 32, selecting the best checkpoint by the lowest validation loss. Inputs were normalized by simple [0, 1] scaling; we did not apply ImageNet mean/std normalization for SnoutNet.

For the augmented run, we used a geometry-consistent horizontal flip with probability 0.5 (updating u accordingly while keeping v unchanged), and a light color jitter (brightness/contrast/saturation ±0.2 and hue ±0.1). Evaluation focuses on Euclidean distance in pixels across the test set, reporting min/mean/median/std and percentiles.

Hardware and time:

* GPU: Google Colab NVIDIA A100, 40GB VRAM]; CUDA enabled
* CPU: Google Colab CPU, not trained on.
* Google Colab System RAM: 32 GB
* Total training time (50 epochs): [hh:mm:ss]
* Average time per epoch: [mm:ss]
* Peak GPU memory usage (optional): [\_\_ GB]

Loss plot (paste below):