Computer Vision 2025
(CSE344/ CSE544/ ECE344/ ECE544)
Assignment-1

Max Marks: 110 + (10 Bonus) **Due Date**: 20-Feb-2025, 11:59 PM

Instructions

- Keep collaborations at high-level discussions. Copying/plagiarism will be dealt with strictly.
- Your submission should be a single zip file **HW[n]_Roll_Number.zip**. Include only the **relevant files** arranged with proper names. A single **.pdf report** explaining your codes with relevant graphs, visualization and solution to theory questions.
- Note that you will be provided a separate PDF asking deliverables for each question. You will **only be graded for deliverables** provided in the submitted report.
- Remember to **turn in** after uploading on Google Classroom. No justifications would be taken regarding this after the deadline.
- Start the assignment early. Resolve all your doubts from TAs during their office hours **two days before the deadline.**
- Kindly **document** your code. Don't forget to include all the necessary plots in your report.
- All [**PG**] questions, if any, are **optional for UG** students but are **mandatory for PG** students. UG students will get BONUS marks for solving that question.
- All [**BONUS**] questions, if any, are optional for all the students. As the name suggests, BONUS marks will be awarded to all the students who solve these questions.
- Your submission **must include all the code used for solving the questions**. You can submit *.ipynb* along with the *.py* files.
- For each of the programming questions **you must set the seed with your roll number for reproducibility**. Use `torch.manual_seed(seed: int)` to set the seed value, where *seed = #RollNumber*. If you roll number starts with a letter such as `MT` or `PhD`, you must use the ASCII values of the characters. Submitted codes without this will incur a **penalty**.

Please submit all the theory questions in the report, they will be scored based on the solution provided in the report. We have provided the deliverables for each question. You will be evaluated based on the submitted deliverables. For problems that mention deliverables as *None*, they must be in the report to fetch points. For some of

the questions, you are provided a boilerplate code or a pseudo-code instruction. Follow the same for your implementation. Failing to follow the same for your implementation will incur a penalty, unless otherwise noted. NOTE: For questions which involve saving models, please save the **state_dict** to save only the weights, and not the model file itself. Please use the first method on this page. Failure to do so will incur a penalty. Your submission's directory structure must follow the following:

```
HW1_[Roll_Number]
├── [Roll_Number]_Report.pdf
├── Segmentation/
│   ├── model_class.py
│   ├── decoder.pth
│   └── deeplabv3.pth
├── Classification/
│   ├── model_class.py
│   ├── dataset_class.py
│   └── weights/
│       ├── convnet.pth
│       ├── resnet.pth
│       └── resnet_aug.pth
├── Detection/
│   ├── Detection-Starter-Notebook.ipynb
│   └── coco_predictions.json
└── Tracking/
    ├── IOU_Tracker.py
    ├── Byte.pkl
    ├── IouTracker.pkl
    └── tracking.py
```

1. (43 points) **Image Classification**

    1. (5 points) Refer to the Russian Wildlife Dataset.

        (a) (1 point) Download the dataset and use the following mapping as the class labels: {'amur_leopard': 0, 'amur_tiger': 1, 'birds': 2, 'black_bear': 3, 'brown_bear': 4, 'dog': 5, 'roe_deer': 6, 'sika_deer': 7, 'wild_boar': 8, 'people': 9} Perform a stratified random split of the data in the ratio 0.8:0.2 to get the train and validation sets. Create a custom Dataset class for the data. Initialize Weights & Biases (WandB)(Video Tutorial).
        Deliverable: a screenshot of your Dataset class in your report.

        (b) (2 points) Create data loaders for all the splits (train and validation) using PyTorch.
        Deliverable: None

        (c) (2 points) Visualize the data distribution across class labels for training and validation sets.
        Deliverable: None

    2. (13 Points) Training a CNN from scratch (Tutorial):

(a) (3.5 points) Create a CNN architecture with 3 Convolution Layers having a kernel size of 3×3 and padding and stride of 1. Use 32 feature maps for the first layer, 64 for the second and 128 for the last convolution layer. Use a Max pooling layer having kernel size of 4×4 with stride 4 after the first convolution layer and a Max pooling layer having kernel size of 2×2 with stride 2 after the second and third convolution layers. Finally flatten the output of the final Max pooling layer and add a classification head on top of it. Use ReLU activation functions wherever applicable.
Deliverable: A screenshot of your model class in your report.

(b) (3 points) Train the model using the Cross-Entropy Loss and Adam optimizer for 10 epochs. Use wandb to log the training and validation losses and accuracies.
Deliverable: (1) Screenshots of your **WandB plots** in the report. WandB plots are mandatory. (2) Your model's **state_dict** saved as **convnet.pth** under the **weights/** folder. You MUST follow this directory structure and naming convention to obtain points.

(c) (0.5 points) Look at the training and validation loss plots and comment whether the model is overfitting or not.
Deliverable: None

(d) (3 points) Report the Accuracy and F1-Score on the validation set. Also, log the confusion matrix using wandb.
Deliverable: None

(e) (3 points) For each class in the validation set, visualize any 3 images that were misclassified along with the predicted class label. Analyze why the model could possibly be failing in these cases. Is this due to the fact that the image does not contain the ground truth class or it looks more similar to the predicted class or something else? Can you think of any workaround for such samples?
Deliverable: None

3. (10 points) Fine-tuning a pretrained model

(a) (3.5 points) Train another classifier with a fine-tuned Resnet-18 (pre-trained on ImageNet) architecture using the same strategy used in Question 2.2.(b) and again use wandb for logging the loss and accuracy.
Deliverable: (1) WandB plots in the report; (2) Your model's **state_dict** saved as **resnet.pth** under the **weights/** folder. You MUST follow this directory structure and naming convention to obtain points.

(b) (0.5 points) Look at the training and validation loss plots and comment whether the model is overfitting or not.
Deliverable: None

(c) (3 points) Report the Accuracy and F1-Score on the validation set. Also, log the confusion matrix using wandb.
Deliverable: None

(d) (3 points) For deep neural networks, typically, the backbone is the part of a model (initial layers) that is used to extract *feature representations* (or simply

features) from the raw input data, which can then be used for classification or some other related task. These features are expressed as an n-dimensional vector, also known as a feature vector and the corresponding vector space is referred to as the feature space. As the training progresses and the classifier learns to classify the input, the data samples belonging to the same class lie closer to each other in the feature space than other data samples. For input samples from the training and validation sets, extract the feature vectors using the backbone (ResNet-18 in this case) and visualize them in the feature space using the tSNE plot in a 2-D Space. Also, visualize the tSNE plot of the validation set in a 3D-Space.
Deliverable: None

4. (10 points) Data augmentation techniques
   (a) (3.5 points) Use any 3 (or more) Data Augmentation techniques that are suitable for this problem. Remember that data augmentation techniques are used for synthetically adding more training data so that the model can train on more variety of data samples. Visualize 4-5 augmented images and add them to your report.
   Deliverable: (1) Screenshot of the code for augmentations you are applying; (2) The visualized augmented images.
   (b) (3 points) Follow the same steps as in Question 2.3.(a) to train the model.
   Deliverable: (1) Same as per Q2.3(a); (2) our model's **state_dict** saved as **resnet_aug.pth** under the **weights/** folder. You MUST follow this directory structure and naming convention to obtain points.
   (c) (0.5 point) Look at the training and validation loss plots now and comment if the problem of overfitting is getting resolved or not.
   Deliverable: WandB plots, report.
   (d) (3 points) Report the Accuracy and F1-Score on the validation set. Also, log the confusion matrix using wandb.
   Deliverable: None
5. (5 points) Compare and comment on the performance of all three models.
   Deliverable: None

2. (35 points) **Image Segmentation**

   1. (5 points) Download the CAMVid dataset.
      (a) (1 point) Download the dataset and write the dataloader. The image size in the dataset is (960, 720) shape; you need to resize the image to (480, 360) and normalize the input image by applying mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225].
      Deliverable: Screenshot of the dataset class code in the report.
      (b) (2 points) Visualize the class distribution across the provided dataset.
      Deliverable: None
      (c) (2 points) Visualize two images along with their mask for each class.
      Deliverable: None

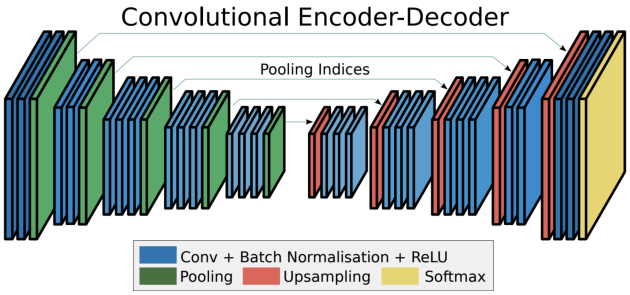

Figure 1: Encoder-decoder architecture for segmentation.

2. (15 points) Train SegNet Decoder from scratch:
   (a) (6 points) We provide a pre-trained SegNet encoder, implemented in the given model_classes.py file, which has been trained on the **CamVid** dataset. Additionally, the file includes a skeleton structure for the decoder that you will need to implement from scratch.
   Your task is to complete the decoder implementation by following the provided instructions. Once implemented, train the decoder by using the **segnet_pretrained**, use the **cross-entropy loss** function, with a **Batch Normalization** momentum of 0.5, and optimize the model using the **Adam optimizer**. Also, log your training loss on wandb.
   Deliverable:(1) WandB plots in the report. (2) The model class implemented in **model_class.py** as **SegNet_Decoder**. (3) Your model's **state_dict** saved as **decoder.pth**. You MUST follow this directory structure and exact naming convention to obtain points.
   (b) (4 points) Report the classwise performance of the test set in terms of pixel-wise accuracy, dice coefficient, IoU (Intersection Over Union), and also mIoU. Additionally, report precision and recall. Use the IoUs within the range [0, 1] with 0.1 interval size for computation of the above metrics. You may refer to this article to learn more about the evaluation of segmentation models. Include all your findings in the submitted report.
   Deliverable: None
   (c) (5 points) For each class in the test set, visualize any three images with IoU ≤ 0.5 and the predicted and ground truth masks; the visualization of masks should be in proper color code . Comment on why the model could possibly be failing in these cases with the help of IoU visualizations. Is the object occluded, is it being misclassified, or is it due to the environment (surroundings) where the object is present?
   Deliverable: None
3. (15 points) Fine-tuning DeepLabV3 on the CamVID Dataset.
   (a) (6 points) Use a pre-trained DeepLabv3 trained on the Pascal VOC dataset, fine-tune it for CamVID Dataset using the cross-entropy loss function and the Adam optimizer. Also, log your training loss on wandb.
   Deliverable: (1) WandB plots in the report; (2) The model class implemented

in **model_class.py** as **Deeplabv3**. (3) Your model's **state_dict** saved as **deeplabv3.pth**. You MUST follow this directory structure and exact naming convention to obtain points.

(b) (4 points) Report the classwise performance of the test set in terms of pixel-wise accuracy, dice coefficient, IoU (Intersection Over Union), and also mIoU. Additionally, report precision and recall. Use the IoUs within the range [0, 1] with 0.1 interval size for computation of the above metrics and compare it with your implemented SegNet Model. Include all your findings in the submitted report.
Deliverable: None

(c) (5 points) For each class in the test set, visualize any three images with IoU ≤ 0.5 and the predicted and ground truth masks; the visualization of masks should be in proper color code same as done in the part-1 of this question. Comment on why the model could possibly be failing in these cases with the help of IoU visualizations. Is the object occluded, is it being misclassified, or is it due to the environment (surroundings) where the object is present?
Deliverable: None

3. (28 points) **Object Detection and Multi-Object Tracking**

1. (18 points) The *Object Detection* problem involves two tasks – *localizing* the object, i.e. determining the coordinates of the valid bounding boxes in the image, and *classifying* the objects within those boxes from a vocabulary of object classes. By completing this task, you will learn how to use detection models and interpret their outputs. More importantly, you will learn how to perform a deep analysis of prediction errors.

   Deliverables: For this problem, you are given a starter notebook called *Detection-Starter-Notebook.ipynb* under the *Detection/* folder. This notebook should help you get started with implementing the metrics. However, you are not required to use it, and you may choose to ignore it and write your own code if you find it more appropriate. However, you must submit all your code.

   (a) (1 point) You will use the Ultralytics API to detect objects in the COCO 2017 validation set. Familiarize yourself with the API and download the COCO 2017 validation set. Note that the Ultralytics repo has a script to automatically download the data – you may refer to their documentation. You will not need the *train* and *test* splits, only the *val* split is required for this task.
   Deliverables: None

   (b) (2 points) Use a COCO-pretrained **YOLOv8** model to make predictions on COCO val2017. Save your predictions in a json file **in the standard COCO format**. You can find this format specification on the COCO website. Report the mAP (Mean Average Precision) of your model predictions.

Deliverables: your predictions json file, saved as **coco_predictions.json** in the **Detection/** folder. The code for generating this file should be in your submitted code. Your json file name **MUST** match this structure and filename to fetch points.

(c) (2 points) Check out the TIDE Toolbox. This tool analyzes detector predictions to classify and quantify the specific types of errors. Use your saved predictions to compute the TIDE statistics on the val set. Moreover, read the TIDE paper and understand what these errors mean. Comment on the error analysis of your model as per your understanding.
Deliverables: Add your results **and corresponding analysis** in the report. The code for generating these results should be in your submitted code.

(d) (5 points) Modern deep neural networks often have poor *confidence calibration* – that is, their predicted probability estimates are not a good representative of the true correctness likelihood. Expected Calibration Error (ECE) is one metric that summarizes this miscalibration error into a scalar statistic. Please read **Section 2** of this paper to learn how to compute ECE (Eqn. 3).
Compute this metric on your COCO val2017 predictions. Report and comment on the significance of the error obtained for your predictions.
Deliverables: Add your results **and corresponding analysis** in the report. The code for generating these results should be in your submitted code.

(e) (5 points) The COCO API reports performance for three scales of objects – small, medium and large. These objects are classified based on the *area* of the bounding box of each object – **small** objects are defined as those with area $< 36 \cdot 36$, **medium** as $36 \cdot 36 \leq$ area $< 96 \cdot 96$, and **large** as area $\geq 96 \cdot 96$. Compute the TIDE statistics at each of the three object scales. For this, you can filter your saved predictions file based on the three scales and save a new file for each scale, and then match each of them against the ground truth annotations file with the TIDE API.
Also, compute the Expected Calibration Error at each of these scales.
Deliverables: Add your results **and corresponding analysis** in the report. The code for generating these results should be in your submitted code.

(f) (3 points) Answer each of the following points separately.

- (1 point) What can you infer from these observations?
- (1 point) Comment on your observations across each of the three scales.
- (1 point) Compare these statistics with the relevant metrics computed with all objects, as you computed in 4.(c) and 4.(d).

Deliverables: Analysis in the report.

2. **[BONUS]** (10 points) Refer to the MOT17 Dataset for object tracking evaluation. You need to evaluate ByteTrack using the training dataset.

(a) (2.5 points) Download the MOT17 dataset. Visualize the first frame of the first video in the training set. Visualize the data distribution using the number of frames. Visualize the first frame of any one video.
Deliverable: None

(b) (1 point) ByteTrack is a simple, fast and strong multi-object tracker. Use ByteTrack to track detection provided in any one video and visualize the output video with bounding boxes.
Deliverable: pickle file of the initialized class.

(c) (4 points) Implement a simple IOU Tracker, and track the detection provided in any one video and visualize the output video with bounding boxes
Deliverable: Implemented IOU Tracker code , pickle file of the initialized class

(d) (2.5 points) Compare the performance of your custom-implemented IoU tracker with ByteTrack. To know more about MOT metrics, click this link.
Deliverable: None