

Practices in visual computing 1

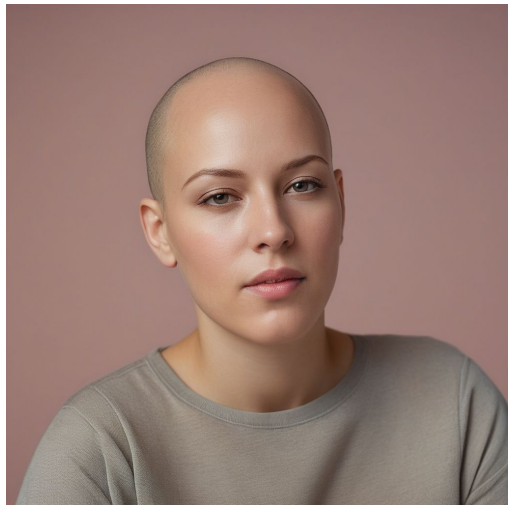
Lab5: Image Classification 1

Simon Fraser University
Fall 2025

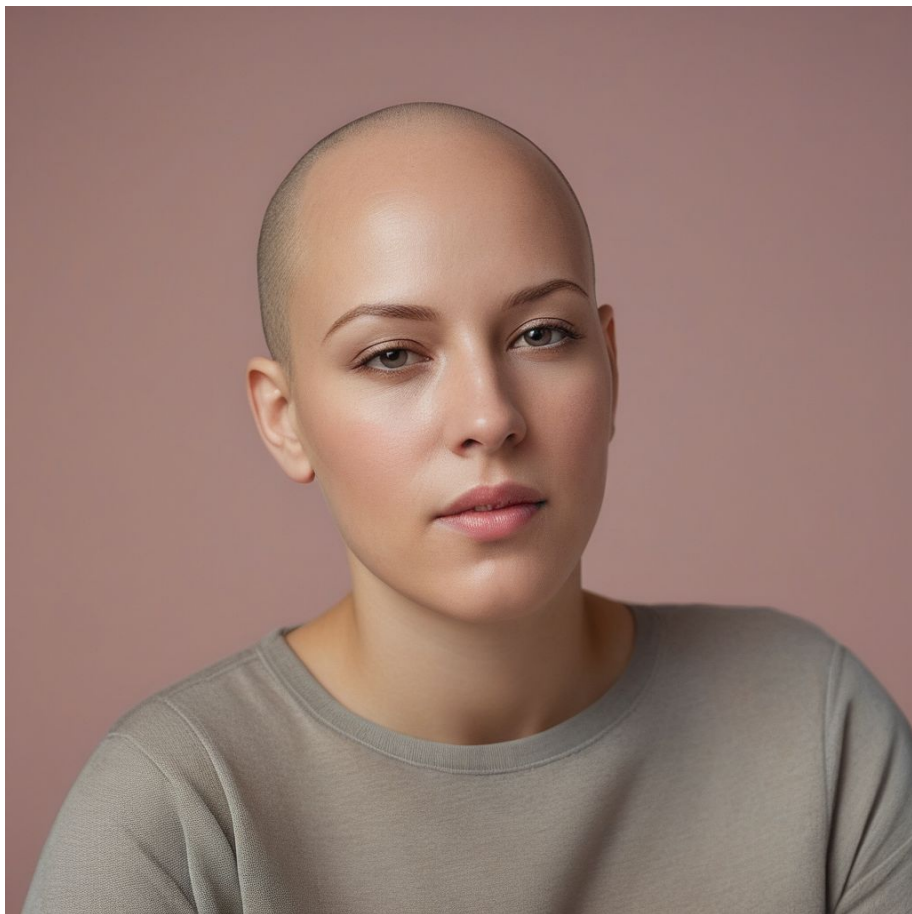
Some application of Poisson blending

We wanted to make a pair dataset of people with and without hair.

The problem was identity preservation.



Some application of Poisson blending



Some application of Poisson blending

Using Poisson Blending



Some application of Poisson blending

Using Poisson Blending + generative refinement



Next Steps: Moving Beyond the Basics

Now that we have covered the fundamentals, it is time to dive into more practical, useful, and exciting applications!

Image Classification

Image Classification

Image Segmentation

Image Classification

Image Segmentation

Object Detection

Why These Are Crucial Skills to Master?

Image Classification

Identifies **what is in an image** (e.g., dog, cat, car).

Used in **facial recognition**, **medical diagnosis**, and

Necessary for more advanced tasks like detection and tracking.



Image Segmentation

Determines where objects are located in an image, assigning labels at the pixel level.

Applied in autonomous driving, medical imaging (tumor detection), and augmented reality.



Object Detection

Identifies both what objects are and where they are in an image.

Used in surveillance, robotics, self-driving cars, and traffic monitoring.

Powers tasks like object tracking and interaction in AR/VR environments.



Fundamentals for Advanced Computer Vision

Classification, Segmentation, and Object Detection are essential components for solving many complex vision problems.

Why PyTorch is Essential for Computer Vision?

Preferred framework for **developing and deploying** deep learning models in academia and industry.

Why PyTorch is Essential for Computer Vision?

Preferred framework for **developing and deploying** deep learning models in academia and industry.

Supports **dynamic computational graphs**, making it easier to debug, experiment, and iterate on models.

Why PyTorch is Essential for Computer Vision?

Preferred framework for **developing and deploying** deep learning models in academia and industry.

Supports **dynamic computational graphs**, making it easier to debug, experiment, and iterate on models.

Seamlessly **integrates with GPUs** for fast, efficient model training and inference.

Why PyTorch is Essential for Computer Vision?

Preferred framework for **developing and deploying** deep learning models in academia and industry.

Supports **dynamic computational graphs**, making it easier to debug, experiment, and iterate on models.

Seamlessly **integrates with GPUs** for fast, efficient model training and inference.

Backed by a large, active community with **extensive resources**, tutorials, and **pre-trained models** for quick development.

What techniques we need to know?

Defining and Preparing the Dataset

Data Preprocessing:

- Cleaning and normalizing data is crucial for effective model training. Techniques include normalization, data augmentation, and resizing images.

Defining and Preparing the Dataset

Data Preprocessing:

- Cleaning and normalizing data is crucial for effective model training. Techniques include normalization, data augmentation, and resizing images.

Data Loading:

- Use frameworks like PyTorch's DataLoader to efficiently load and batch your dataset, enabling faster training by processing batches in parallel.

Building the Model

Model Architecture:

- Choose or design the appropriate model architecture (e.g., Convolutional Neural Networks (CNNs) for images, Recurrent Neural Networks (RNNs) for sequences).

Transfer Learning:

- Leverage pre-trained models and fine-tune them on your specific dataset. This is particularly useful when your dataset is small or your task is complex, as it allows your model to benefit from knowledge gained on large datasets like ImageNet.

Choosing the Right Loss Function and Optimizer

Loss Function:

The choice of loss function depends on the type of task:

- Cross-Entropy Loss for classification.
- Mean Squared Error (MSE) for regression.
- Dice Loss for segmentation tasks.

Choosing the Right Loss Function and Optimizer

Optimizer:

- Popular optimizers include Stochastic Gradient Descent (SGD), Adam, and RMSprop. Choosing the right optimizer and tuning learning rates can significantly affect training speed and performance.
- Techniques like learning rate scheduling or adaptive learning rates can help improve convergence.

Tracking Training with TensorBoard or Other Tools

TensorBoard:

Use TensorBoard to visualize metrics like loss, accuracy, and other key indicators during training. It's useful for:

- Tracking how your model's performance evolves.
- Debugging vanishing or exploding gradients.
- Comparing multiple experiments easily.

Using Evaluation Metrics

- Accuracy and Loss
- Confusion Matrix
- Precision, Recall, and F1-Score
- Intersection over Union (IoU)

Handling Overfitting and Underfitting

Regularization Techniques:

Apply L2 regularization (weight decay) or dropout to prevent overfitting.

Handling Overfitting and Underfitting

Regularization Techniques:

Apply L2 regularization (weight decay) or dropout to prevent overfitting.

Early Stopping:

Implement early stopping to halt training once the validation performance stops improving, avoiding overfitting.

Handling Overfitting and Underfitting

Regularization Techniques:

Apply L2 regularization (weight decay) or dropout to prevent overfitting.

Early Stopping:

Implement early stopping to halt training once the validation performance stops improving, avoiding overfitting.

Data Augmentation:

Expanding your dataset with augmentation techniques can help the model generalize better.

Tuning Hyperparameters

Learning Rate:

Arguably the most important hyperparameter. Use techniques like learning rate decay or cyclical learning rates to adjust learning rates dynamically during training.

Tuning Hyperparameters

Learning Rate:

Arguably the most important hyperparameter. Use techniques like learning rate decay or cyclical learning rates to adjust learning rates dynamically during training.

Batch Size:

Finding the right balance in batch size can significantly affect training time and model performance.

Tuning Hyperparameters

Learning Rate:

Arguably the most important hyperparameter. Use techniques like learning rate decay or cyclical learning rates to adjust learning rates dynamically during training.

Batch Size:

Finding the right balance in batch size can significantly affect training time and model performance.

Epochs:

Set an appropriate number of epochs to ensure the model trains long enough to capture patterns but not so long that it overfits.

Model Checkpointing and Saving

Saving Checkpoints:

Save model checkpoints periodically during training to avoid losing progress in case of interruptions. You can also use checkpoints to resume training from where it left off.

Checkpointing is especially useful when training large models over days or weeks.

Model Export:

After training, export the model in a format that can be used for inference, such as PyTorch's .pt or .pth formats.

Inference and Deployment

Post-Training Inference:

Test the model on unseen data and ensure it works effectively in real-world scenarios.

Key Takeaways

Master the Fundamentals: Understand the essential tasks like classification, segmentation, and object detection.

Leverage Powerful Tools: Use frameworks like PyTorch to develop, train, and experiment with deep learning models efficiently. Utilize tools like TensorBoard for tracking performance.

Apply Essential Techniques: Focus on data preparation, hyperparameter tuning, and real-world deployment to ensure your models are efficient, well-optimized, and production-ready.

Resources

<https://pytorch.org/vision/2.0/models.html#classification>

#TODO

1- different models

2- checking different augmentations

3- checking different learning rates and maybe optimizers (momentum and weight_decay)

4- train with different num of epochs