Image Processing HW3

Hamidreza Mafi
610399209

Question 1:

Dataset Import and Visualization

The dataset consists of training, validation, and test directories. We used `torchvision.datasets.Image` to load these directories and applied basic tensor transformations. A few sample images from the training set were visualized using `matplotlib` to ensure the dataset was loaded correctly.

Figure 1: Sample images from the training dataset visualized with their corresponding class labels.

SportVGG: Model Architecture, Training, and Results

We use a modified VGG19-based model, named **SportVGG**, for 100-class classification. The model consists of the first 36 layers of pretrained VGG19 as a frozen feature extractor, followed by a 1×1 convolutional layer to reduce dimensionality. A custom fully connected classifier maps the features to output probabilities.

Stage 1 of training freezes the VGG backbone and trains only the classifier and 1×1 convolution using Adam optimizer (lr $= 1e$-4) for 25 epochs. Accuracy improved steadily from 16.30% to 84.57% on the training set, and from 27.00% to 77.20% on the validation set.

In **Stage 2**, the last 10 layers of VGG19 are unfrozen and fine-tuned with a reduced learning rate ($1e$-5) and cosine annealing scheduler for 15 epochs. The model achieved final training and validation accuracies of 91.63% and 82.00%, respectively.

These results demonstrate that our two-stage approach effectively adapts the pre-trained VGG19 for fine-grained multi-class classification tasks. The final model accuracy on the test dataset: 86.40%

Model Explainability using Grad-CAM

To interpret the predictions of our SportVGG model, we employ the **Gradient-weighted Class Activation Mapping (Grad-CAM)** technique. Grad-CAM provides visual explanations by highlighting regions in the input image that contribute most to the model's prediction.

Grad-CAM works by computing the gradients of the target class score with respect to the feature maps of a convolutional layer. These gradients are globally averaged to obtain

weights, which are then combined with the feature maps to produce a class-discriminative heatmap. This heatmap is overlaid on the original image to visualize the focus area of the model.

We apply Grad-CAM to the output of our 1×1 convolutional layer. The heatmaps are generated for randomly selected validation samples, helping us qualitatively assess whether the model focuses on relevant image regions.

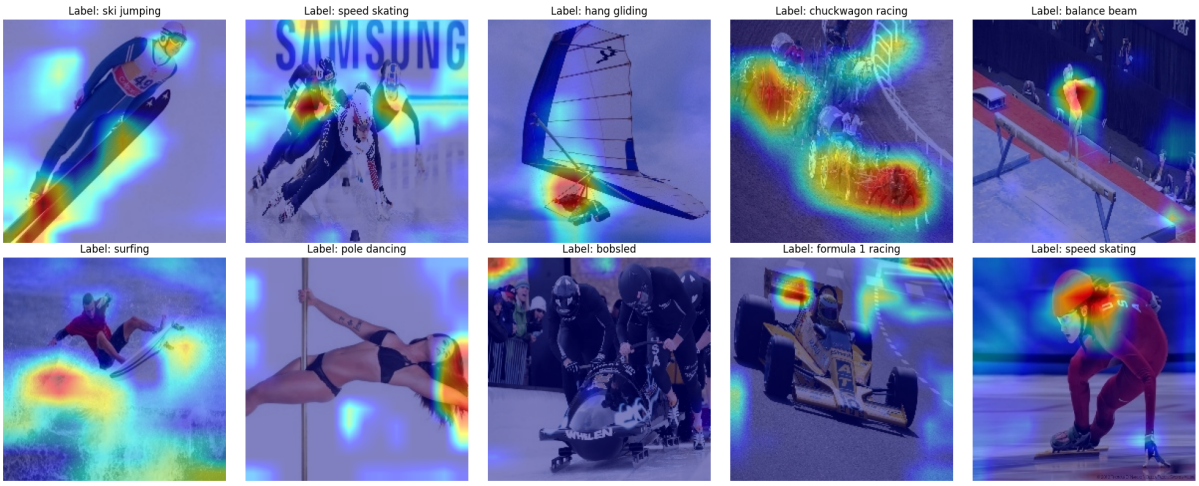


Figure 2: Grad-CAM visualizations for 10 validation samples. The heatmaps show the model's focus areas for each prediction.

Question 2:

Dataset Preparation and Splitting

We used the Weizmann Horse Database from Kaggle for our segmentation task. The dataset was downloaded using the Kaggle API and contains images of horses along with their corresponding binary segmentation masks.

After extracting the data, we sorted and paired the image and mask files. The dataset was then split into 80% training and 20% validation sets using a fixed random seed for reproducibility.

To prepare the data for training, we defined a custom `HorseDataset` class. Each image was resized to 512×512 pixels and converted to a tensor. Masks were binarized to ensure values were either 0 or 1. Data augmentations were kept minimal for simplicity, and the same random seed was used to apply identical transformations to both the image and its mask.

Finally, PyTorch `DataLoaders` were created for efficient batching during training and evaluation.

Image 171

Mask 171

Image 94

Mask 94

Image 239

Mask 239

Image 199

Mask 199

Image 193

Mask 193

Loss Functions for Segmentation

In image segmentation tasks, selecting an appropriate loss function is critical for accurate mask prediction. We used and compared the following loss functions:

- **Binary Cross-Entropy (BCE) Loss**: This loss treats segmentation as a pixel-wise binary classification task. It computes the cross-entropy between the predicted probability and the true label at each pixel. BCE is effective but can be sensitive to class imbalance.
- **Dice Loss**: Dice Loss measures the overlap between predicted and ground truth masks. It is derived from the Dice coefficient and is especially useful when dealing with imbalanced datasets, as it directly optimizes for the intersection-over-union metric.
- **Focal Loss**: Focal Loss is an extension of BCE that down-weights easy examples and focuses training on hard, misclassified pixels. This is particularly helpful in segmentation tasks with severe foreground-background imbalance.

Each loss function contributes differently to model learning, and their effectiveness depends on the specific characteristics of the dataset.

Model Architecture and Training Results

U-Net Model Design

In this project, we implemented a U-Net-based semantic segmentation model using PyTorch. U-Net is a popular convolutional neural network architecture specifically designed for biomedical image segmentation, but it generalizes well to other domains due to its encoder-decoder structure with skip connections.

Our U-Net model consists of the following key components:

- **Encoder (Contracting Path)**: Three convolutional blocks that extract features from the input image. Each block uses two convolutional layers followed by ReLU activation and optional dropout. Downsampling is performed using max-pooling layers.
- **Bottleneck**: A convolutional block with the highest feature dimensionality, incorporating dropout to reduce overfitting.
- **Decoder (Expanding Path)**: Three upsampling blocks that progressively reconstruct the segmentation mask. Each upsampled feature map is concatenated with its corresponding encoder output (via skip connections) and passed through a convolutional block.
- **Final Output Layer**: A 1x1 convolution to produce a single-channel output, followed by a sigmoid activation to yield pixel-wise probabilities.

The network was trained using Binary Cross-Entropy (BCE) loss, and we experimented with Focal Loss for comparison. Performance metrics used include Intersection-over-Union (IoU), Accuracy, Precision, Recall, and F1 Score.

Training and Evaluation Pipeline

The training process was conducted over multiple epochs using the Adam optimizer with learning rate scheduling (ReduceLROnPlateau). The model's performance was evaluated on a validation set after each epoch, and the best model was saved based on validation IoU.

The training loop included:

- Forward propagation through the U-Net model.
- Computation of BCE loss between predicted and ground truth masks.
- Backpropagation and optimizer step.
- Thresholding outputs to obtain binary predictions.
- Evaluation of all key metrics per batch and average aggregation.
- Visualization of one validation example after every epoch.

Fine-Tuning Results

We performed a series of fine-tuning experiments by modifying hyperparameters such as learning rate, channel depth factor (k), loss function, and image size. The table below summarizes the best validation IoU scores achieved under different configurations:

Learning Rate	k	**Loss Function**	**Epochs**	**Image Size**	**Best Val IoU**
1e-4	64	BCE	75	512×512	0.7964
1e-4	128	BCE	30	512×512	0.7328
1e-4	128	BCE	100	512×512	0.7957
1e-4	180	BCE	15	512×512	0.4812
1e-4	100	BCE	100	512×512	0.7943
5e-5	128	BCE	100	512×512	0.7617
1e-4	110	BCE	15	512×512	0.7957
1e-4	100	Focal Loss	50	512×512	0.5449
1e-3	64	BCE	70	512×512	0.5591
1e-4	80	BCE	100	800×800	0.7491
1e-3	32	BCE	100	512×512	0.7692
8e-5	200	BCE	100	512×512	**0.8023**

Table 1: Summary of Fine-Tuning Experiments

The best performance was obtained with a learning rate of 8×10^{-5}, $k = 200$, 100 epochs, BCE loss, and image size of 512×512, achieving a validation IoU of 0.8023.

Training History Visualization

To better understand the training dynamics, we tracked all key metrics throughout the epochs. Figure 4 shows the plots for loss, IoU, accuracy, precision, recall, and F1 score for both training and validation phases.

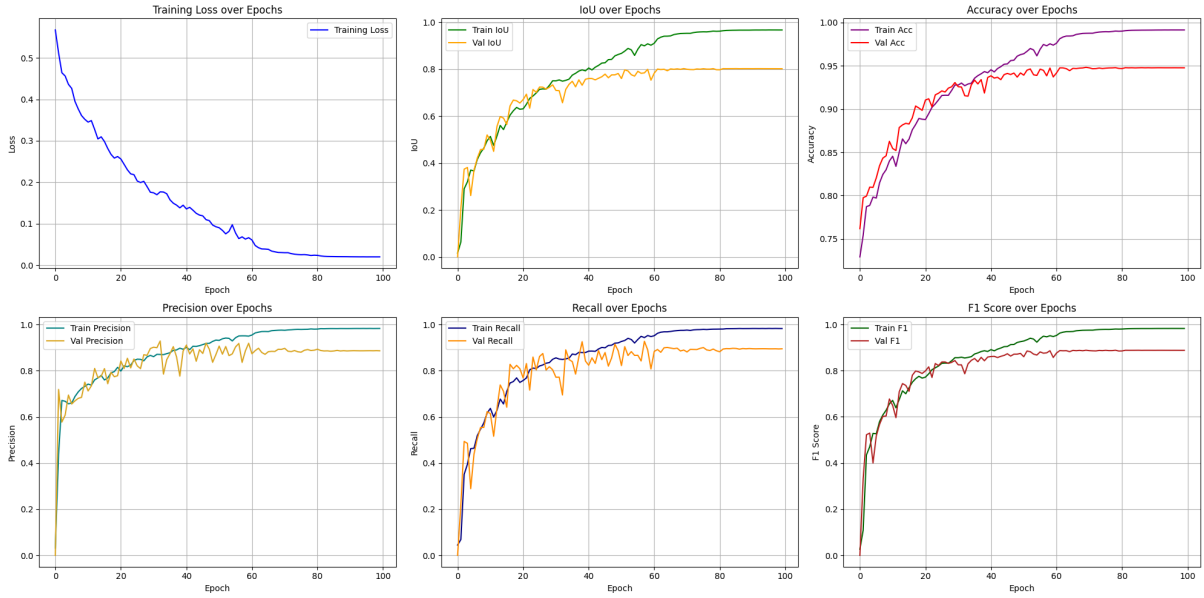


Figure 4: Training and Validation Metrics over Epochs

Qualitative Evaluation on Validation Set

To visually assess the model's segmentation performance, we present qualitative results on a subset of the validation images. Each sample includes the input image, ground truth mask, and predicted mask, as shown in Figure 5.

Image | Ground Truth Mask | Predicted Mask

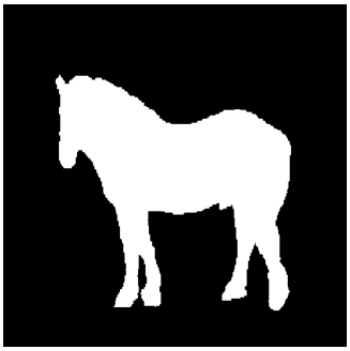
Image 1

Ground Truth

Prediction

Image 2

Ground Truth



Prediction

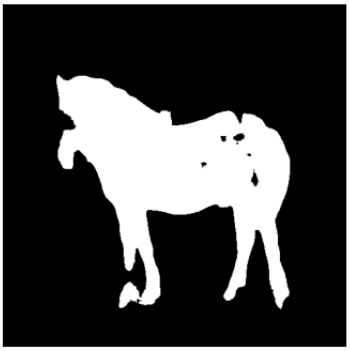


Image 3

Ground Truth

Prediction

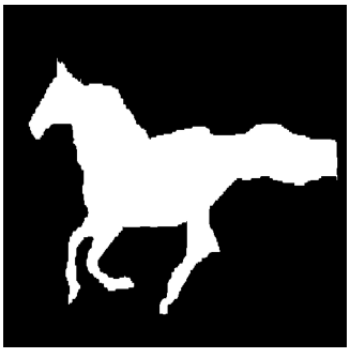
Image 4

Ground Truth

Prediction

Image 5

Ground Truth



Prediction

Seg-Grad-CAM: Visualizing Model Attention in U-Net

Introduction to Seg-Grad-CAM

Understanding how a deep neural network makes decisions is critical for building trust in its predictions, especially in tasks like semantic segmentation. Gradient-weighted Class Activation Mapping (Grad-CAM) is a popular visualization technique that highlights important regions in the input that influence the model's output. Seg-Grad-CAM is an adaptation of this technique for segmentation models, allowing pixel-wise attention visualization at various levels of the network.

In our project, we implemented Seg-Grad-CAM for U-Net to visualize which regions of the input image activate the different feature stages (`x1` through `x6`) and how they contribute to the predicted segmentation mask.

How Seg-Grad-CAM Works

The Seg-Grad-CAM function works as follows:

1. The input image is passed through the U-Net model, and the predicted mask is thresholded to identify foreground pixels.
2. Feature maps from intermediate layers (`x1` to `x6`) are retained for gradient tracking.
3. The sum of the predicted mask pixels is computed and backpropagated to each retained feature map to get gradients.
4. A weighted average of the gradients across spatial dimensions is computed for each feature map, producing importance weights.
5. These weights are used to generate activation maps (Grad-CAM heatmaps) which highlight regions in the input image that contributed most to the prediction at each stage.
6. The heatmaps are upsampled to match the input image size and overlayed with the original image using a `jet` color map.

Implementation and Usage

The function `seg_grad_cam()` implements this procedure in PyTorch. For evaluation, we selected five random validation samples and applied the Grad-CAM process at all six decoder stages of U-Net. This gives insight into how attention is distributed across the network's depth, from shallow to deeper features.

Heatmap Visualization

Figures 6 through 10 show Grad-CAM heatmaps for five randomly selected validation samples. Each figure displays six subplots corresponding to stages `x1` through `x6`. Red regions represent areas with higher contribution to the predicted segmentation.

Figure 6: Seg-Grad-CAM visualizations for Validation Sample 1 (Stages x1 to x6)

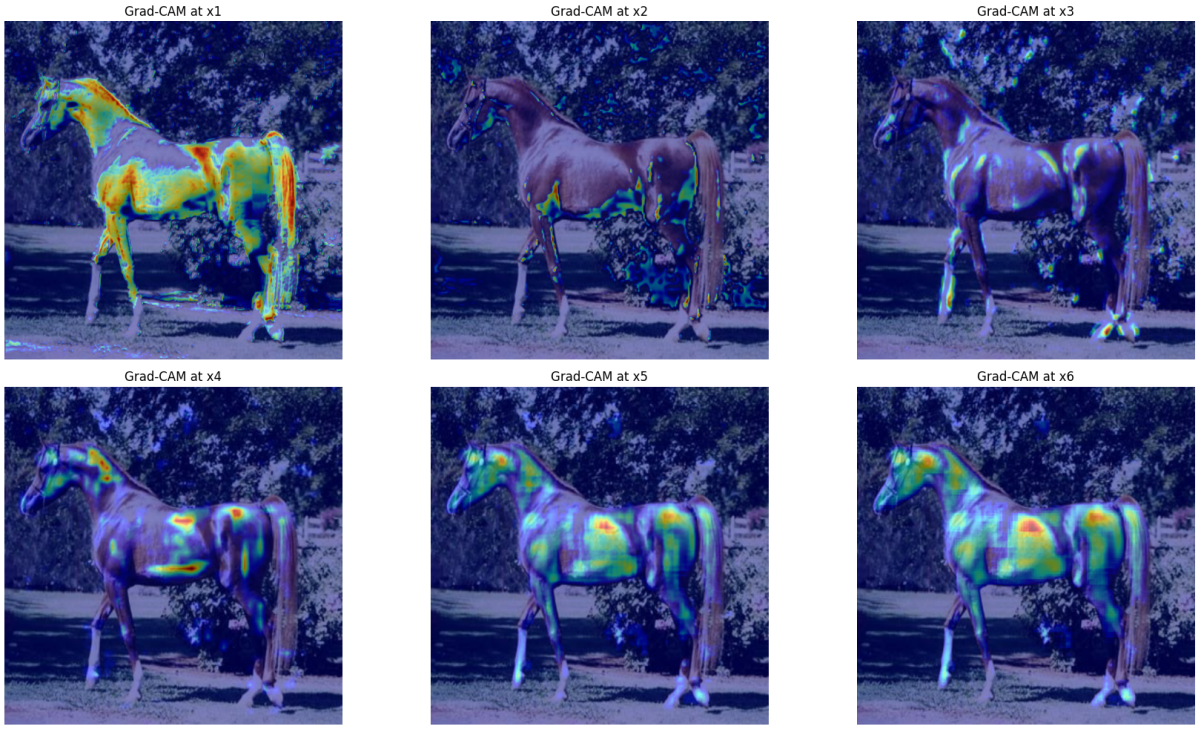


Figure 7: Seg-Grad-CAM visualizations for Validation Sample 2 (Stages x1 to x6)

Figure 8: Seg-Grad-CAM visualizations for Validation Sample 3 (Stages x1 to x6)

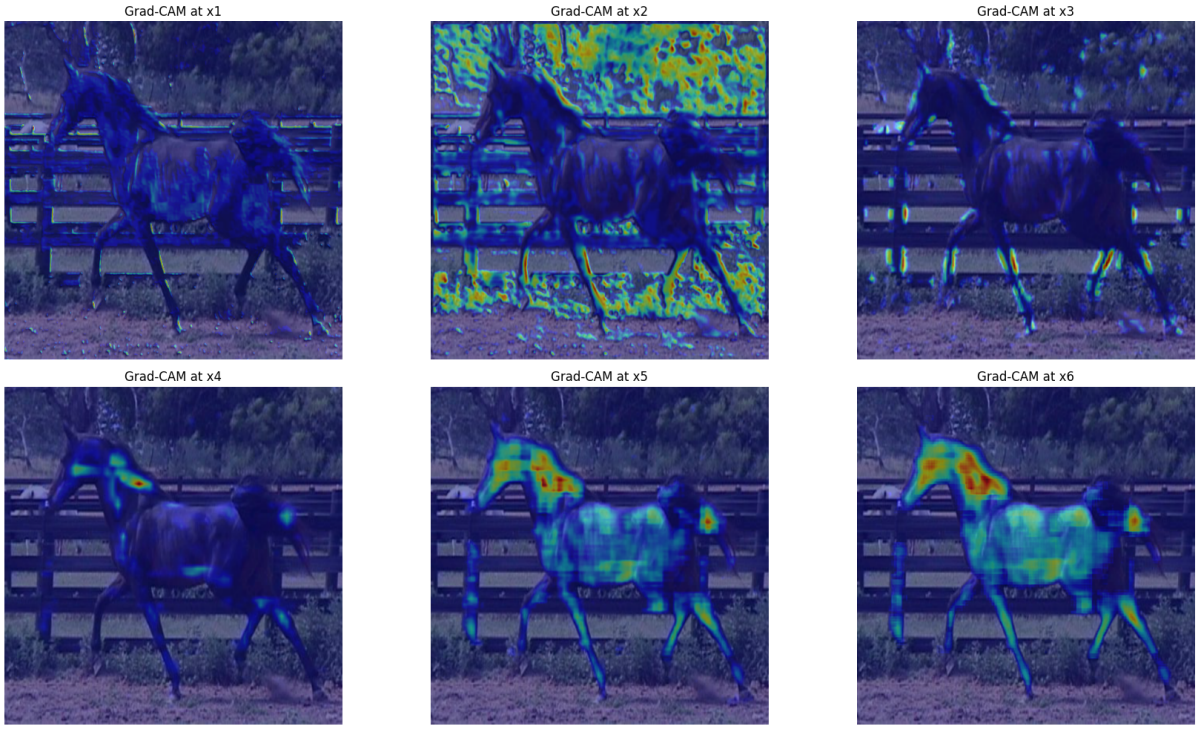


Figure 9: Seg-Grad-CAM visualizations for Validation Sample 4 (Stages x1 to x6)

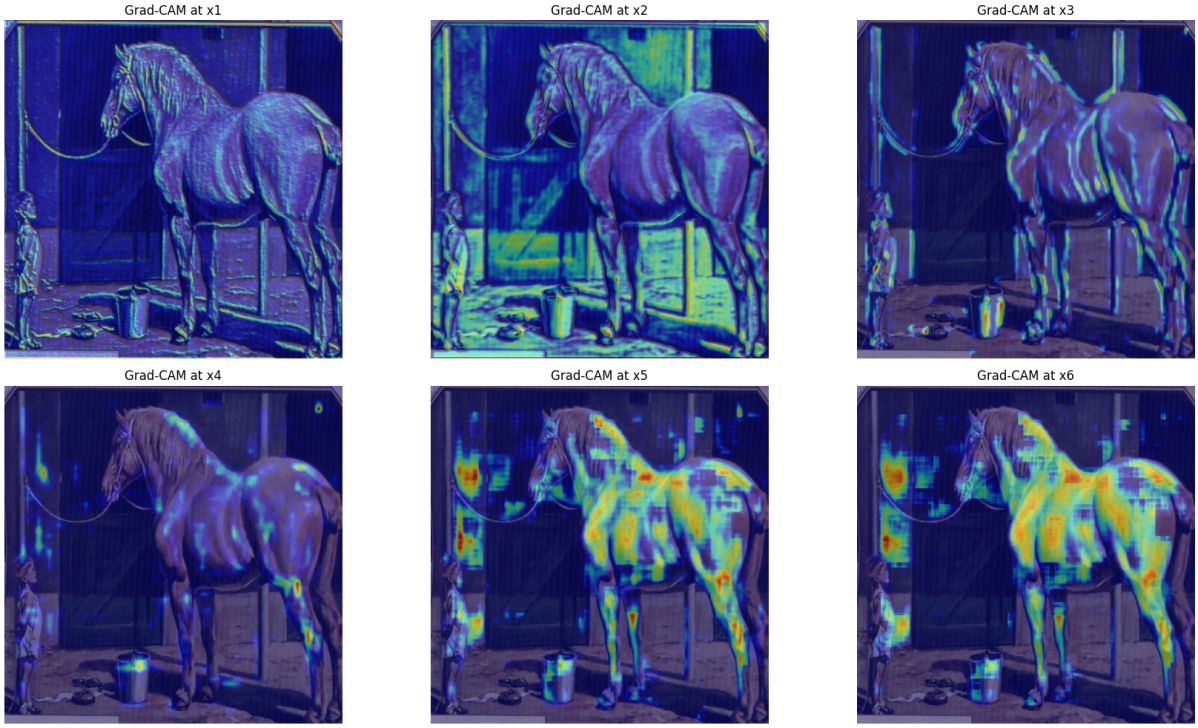


Figure 10: Seg-Grad-CAM visualizations for Validation Sample 5 (Stages x1 to x6)