PROBLEM STATEMENT

Problem: There's limited theoretical understanding of how model size, data size, and compute affect performance.

Research Goal: Empirically and theoretically characterize scaling laws in vision similar to those in NLP.

Approaches: Power-law fitting, scaling law estimation, empirical studies with varied model/data sizes.

Benchmark: ImageNet subsets.

INTRODUCTION

Scaling laws explain how model performance changes with increasing model size, dataset size, or compute.
They typically follow a power-law relation:

$$L = k \cdot N^{-\alpha}$$

where L is loss, N is scale (like parameters or data) and α is the scaling exponent.
In simple terms ,as we scale up models or data, the loss decreases in a predictable trend until it plateaus.
This concept helps identify how far scaling remains beneficial and when returns start diminishing.

DATASET DESCRIPTION

Source: All experiments were conducted using ImageNet subsets derived from a single class – n01494475 (tench, a species of fish).
Using a single class helps eliminate label imbalance and focus purely on the effects of model size and dataset size on performance.

Setup:
Two independent experiments were performed using different dataset configurations:

Model Loss Analysis:
Used 200 images from the class n01494475.
Purpose: To study how ResNet model depth (ResNet18, ResNet50, ResNet101) affects loss on a fixed dataset.

Dataset Loss Analysis:
Used three datasets of increasing size:
100 images → IDs 1–100
200 images → IDs 1–200
300 images → IDs 1–300

Purpose: To analyze how dataset size influences model performance and loss scaling trends.

TOOLS AND FRAMEWORKS

- **Programming Language:** Python 3.x – primary language for data handling, model training, and analysis.
- **Deep Learning Framework:** PyTorch – for loading pretrained ResNet models, customizing layers, and training.

Computer Vision Libraries:

- **OpenCV –** for image preprocessing, grayscale conversion, and contour detection.
- **PIL (Pillow) –** for image loading and basic transformations.
- Pretrained Models: ResNet18, ResNet50, ResNet101 from torchvision.models, pretrained on ImageNet.
- **Segmentation/Object Detection:** SAM (Segment Anything Model) – used for automatic region detection in images.

Data Handling and Analysis:

- **pandas –** for CSV creation and storage of per-image/model/dataset losses.
- **NumPy –** array operations and numerical computations.
- **Matplotlib and Seaborn –** for loss distributions, correlations, and scaling law plots.

OBJECT DETECTION PIPELINE

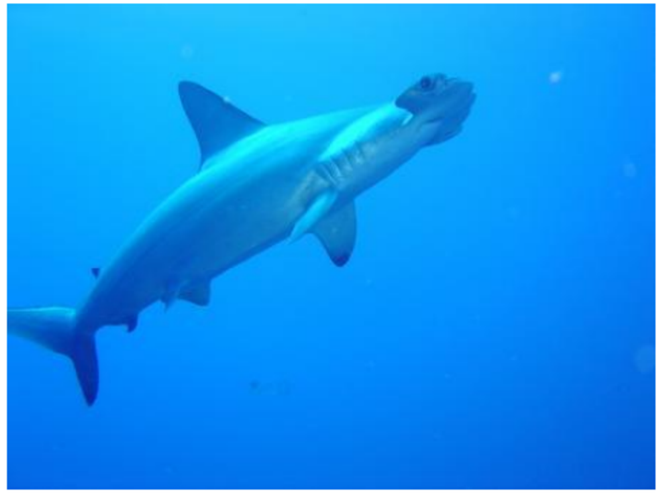
- Segment Anything Model (ViT-B) used for image segmentation.
- Each input image undergoes preprocessing before segmentation.
- Otsu’s thresholding applied to detect object regions automatically.
- Morphological operations (open, close) used to refine binary masks.
- HSV-based filtering ensures color-level region detection.
- Fallback points guarantee segmentation even in low-feature images.

Input Image → Preprocessing → Blob Detection → SAM Segmentation → Mask Output

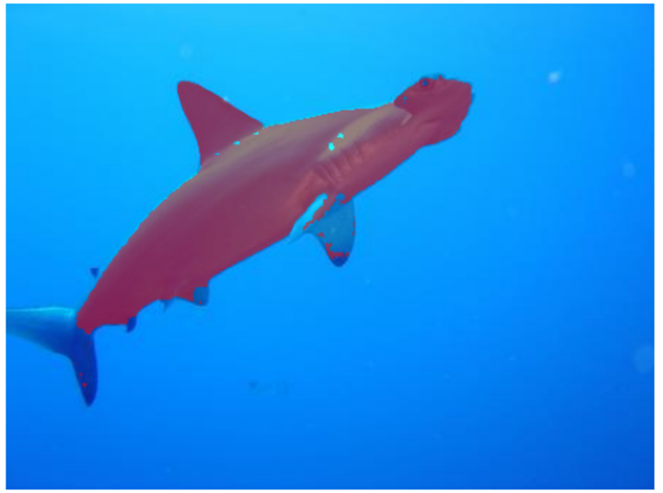
```python
def detect_blobs_otsu(gray):
    _, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
    kernel = np.ones((3,3), np.uint8)
    cleaned = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel, iterations=2)
    merged = cv2.morphologyEx(cleaned, cv2.MORPH_CLOSE, kernel, iterations=2)
    contours, _ = cv2.findContours(merged, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    points = []
    for cnt in contours:
        area = cv2.contourArea(cnt)
        perimeter = cv2.arcLength(cnt, True)
        circularity = 4 * np.pi * area / (perimeter**2 + 1e-6)
        if MIN_AREA < area < MAX_AREA and 0.5 < circularity < 1.5:
            M = cv2.moments(cnt)
            if M["m00"] != 0:
                cx = int(M["m10"]/M["m00"])
                cy = int(M["m01"]/M["m00"])
                points.append((cx, cy))
    return points
```

```python
def detect_blobs_hsv(image):
    hsv = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
    lower = np.array([0, 50, 50])
    upper = np.array([179, 255, 255])
    mask = cv2.inRange(hsv, lower, upper)
    kernel = np.ones((3,3), np.uint8)
    cleaned = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel, iterations=1)
    contours, _ = cv2.findContours(cleaned, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    points = []
    for cnt in contours:
        area = cv2.contourArea(cnt)
        if MIN_AREA < area < MAX_AREA:
            M = cv2.moments(cnt)
            if M["m00"] != 0:
                cx = int(M["m10"]/M["m00"])
                cy = int(M["m01"]/M["m00"])
                points.append((cx, cy))
    return points
```

Input Image



Segmented Overlay



LOSS CALCULATION

Different loss functions were considered, but MSE was chosen as the most suitable for analyzing scaling behavior.

- Cross-Entropy Loss: Designed for classification tasks; outputs discrete probability distributions. Not meaningful for continuous pixel or feature-level comparisons.
- MAE (Mean Absolute Error): Treats all deviations equally; less sensitive to large errors and produces less distinct scaling patterns.
- Huber Loss: Hybrid of MSE and MAE, robust but adds unnecessary complexity since outliers weren’t dominant in our data.
- MSE (Mean Squared Error): Best suited for continuous-valued regression; penalizes large deviations more, offering a clearer quantitative measure of model performance relative to scale.

MODEL LOSS COMPUTATION PIPELINE

Dataset Handling:

- Images are loaded from a single folder (200Images) and preprocessed to 224×224 using PyTorch transforms.
- A custom Dataset class ensures numeric sorting of filenames for reproducibility.

Model Preparation:

- Three ResNet architectures (ResNet18, ResNet50, ResNet101) are loaded with pretrained ImageNet weights.
- All layers are frozen except the fully connected (FC) layer.
- The FC layer is modified to output a vector of the same size as input, acting like an identity mapping.

Training Strategy:

- Only the FC layer is trained for a few epochs using Mean Squared Error (MSE) loss.
- MSE is used here because it provides a scalar measure of the difference between the FC output and its target, suitable for creating a uniform per-image metric across models.
- Purpose: generate “pre-loss” values representing model-specific output magnitudes per image, without affecting pretrained features.

Logging Losses:

- After training, the MSE values are computed per image for each model.
- Stored in a dictionary and exported to CSV for subsequent scaling law analysis.
- Provides a consistent metric to compare performance across models and support empirical study of scaling laws.

```
for model_name in models_to_run:
    print(f"\nTraining {model_name}...")
    model = get_model(model_name)
    optimizer = torch.optim.Adam(model.fc.parameters(), lr=learning_rate)
    criterion = nn.MSELoss()

    # Training FC layer
    for epoch in range(num_epochs):
        for inputs, _ in dataloader:  # labels not used
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, outputs.detach())  # dummy backward to train FC
            loss.backward()
            optimizer.step()
```

DATASET LOSS COMPUTATION PIPELINE

Dataset Handling:

- Three datasets are used, created from ImageNet subsets: 100, 200, and 300 images.
- Images are preprocessed to 224×224 using PyTorch transforms.
- Images are sorted numerically/alphabetically to maintain consistent ordering.

Model Preparation:

- ResNet18 is loaded with pretrained ImageNet weights.
- All layers are frozen except the fully connected (FC) layer.
- The FC layer is modified to output a vector of size 512, providing a uniform representation for loss computation.

Training Strategy:

- Only the FC layer is trained for a few epochs using Mean Squared Error (MSE) loss.
- MSE is chosen because it provides a scalar measure of difference between the FC output and input vector, creating a consistent per-image metric across datasets.
- Purpose: generate per-image loss values for each dataset size, while keeping pretrained features intact.

Logging Losses:

- After training, the MSE loss for each image is computed and logged for each dataset size.
- Data is stored in a CSV with columns: image_name, loss_100, loss_200, loss_300.
- These values serve as the dataset-specific loss metric for scaling law analysis.

```
resnet18 = models.resnet18(weights=models.ResNet18_Weights.IMAGENET1K_V1)
for param in resnet18.parameters():
    param.requires_grad = False
resnet18.fc = nn.Linear(resnet18.fc.in_features, 512)

criterion = nn.MSELoss()
optimizer = optim.SGD(resnet18.fc.parameters(), lr=0.001)
```

```
try:
    output_vec = resnet18(input_vec)
    loss_val = criterion(output_vec.view(-1), input_vec.view(-1)[:output_vec.numel()])
    loss_val.backward()
    optimizer.step()
except Exception as e:
    print(f"Error processing {img_name}: {e}")
    loss_val = None
```

EMPIRICAL OBSERVATIONS

- Model depth impacts per-image loss: ResNet50 and ResNet101 generally show lower median and maximum losses than ResNet18.
- Dataset scaling shows minimal aggregate effect: loss slightly increases with larger datasets ($\alpha \approx -0.02$), but per-image variability is significant.
- Correlation analysis: deeper models (ResNet50 vs ResNet101) are highly correlated, whereas correlation across dataset sizes is weaker.
- Scaling exponents highlight trends: negative α for model depth suggests possible overfitting on smaller datasets; near-zero α for dataset size indicates minimal scaling impact.
- Per-image variations emphasize need for targeted optimization: preprocessing or selective data augmentation could improve outcomes.

REFERENCES

1. Kaplan, J., McCandlish, S., Henighan, T., et al. Scaling Laws for Neural Language Models.

2. Bahri, Y., Dyer, E., Kaplan, J., et al. Explaining Neural Scaling Laws.

3. Hoffmann, J., Borgeaud, S., Mensch, A., et al. Training Compute-Optimal Large Language Models.

4. He, K., Zhang, X., Ren, S., & Sun, J. Deep Residual Learning for Image Recognition (ResNet). Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

5. Kirillov, A., Mintun, E., Ravi, N., et al. Segment Anything.

6. Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2009.

7. PyTorch Documentation. torchvision.models.resnet – Pretrained Models and API Reference.

8. PyTorch Documentation. torch.nn.MSELoss – Mean Squared Error Loss Function.

9. Loshchilov, I., & Hutter, F. Decoupled Weight Decay Regularization (AdamW Optimizer).

10. Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. Improving Language Understanding by Generative Pre-Training. OpenAI, 2018. (Referenced for scaling law methodology inspiration.)