Deep Learning for Liver CT Classification and Grad-CAM-based Tumor Localization

Harald Karlsen

Spring 2026

Abstract

This report studies deep learning approaches for classifying liver CT slices into three classes (background, liver present, tumor present) and explores the use of Grad-CAM for tumor localization and pseudo-segmentation. A baseline ResNet50 trained from scratch is compared to a DINO-pretrained ResNet50 and a ConvNeXT model, and the limitations of using Grad-CAM for pixelwise segmentation are analyzed.

1 Introduction

Computed tomography (CT) of the abdomen is a central tool in modern diagnostics of liver disease, including tumor detection. The task considered here is supervised classification of 2D CT slices of size 512×512 pixels into three classes: 0 (no liver), 1 (liver present), and 2 (tumor present). Residual networks such as ResNet50 and more recent ConvNeXT architectures are studied for this purpose, together with explainable AI (XAI) via Grad-CAM to highlight tumor regions in the images.

2 ResNet50 From Scratch

2.1 Motivation for ResNet50

ResNet50 [1] is a residual network architecture that implements *skip connections*, which help stabilize optimization of deep networks and mitigate vanishing or exploding gradients. This is particularly important when training the network from scratch rather than fine-tuning a pretrained model. Our task requires capturing high-level context such as organ boundaries and tumor regions in CT slices, while still exploiting local texture and shape information. As a convolutional neural network (CNN), ResNet50 is well suited for learning such patterns and has become a widely-used, well-tested architecture in computer vision.

2.2 Model and Data Setup

We start from a pre-built ResNet50 with randomly initialized weights and adapt the final layer to three output classes:

```
# Randomly initialized ResNet50
model = resnet50(weights=None)
# Changing number of output classes to 3
n_classes = 3
model.fc = nn.Linear(model.fc.in_features, n_classes)
```

ResNet50 expects input of shape $(N, C = 3, H, W)$, so we replicate the single CT channel to form 3 channels. The validation data are given with pixelwise labels, which must be converted to a single image-level label. This is done with the following function:

```python
def pixelwise_to_class(img_dir, label_dir):
    X = []
    y = []
    img_files = sorted(img_dir.glob("*.npy"))
    label_files = sorted(label_dir.glob("*.npy"))

    for img_path, label_path in zip(img_files, label_files):
        img = np.load(img_path)
        pixelwise = np.load(label_path)
        X.append(img)

        # If any pixel labelled 2 => class 2
        if np.any(pixelwise == 2):
            y.append(2)
        # If no pixel has 2, but 1 exists => class 1
        elif np.any(pixelwise == 1):
            y.append(1)
        else:
            y.append(0)

    # Converting X and y to tensors
    X = np.stack(X)
    y = np.array(y)
    X = torch.from_numpy(X).float()
    y = torch.from_numpy(y).long()
    X = X.unsqueeze(1).repeat(1, 3, 1, 1)
    return X, y
```

For training, we use batch size = 32, learning rate = 10^{-4}, and 100 epochs, with AdamW as optimizer and cross-entropy as the loss function. The training loop is:

```python
for epoch in tqdm(range(n_epochs)):
    model.train()
    curr_loss = 0
    samples = 0

    # Training
    for batch_x, batch_y in train_loader:
        batch_x = batch_x.to(device)
        batch_y = batch_y.to(device)
        optimizer.zero_grad()
        # forward pass
        logits = model(batch_x)
        # Cross-entropy loss
        loss = loss_func(logits, batch_y)
        # Backward pass
        loss.backward()
        optimizer.step()

        curr_loss += loss.item() * batch_x.size(0)
        samples += batch_x.size(0)

    training_loss.append(curr_loss / samples)

    # Validation
    model.eval()
    total_samples = 0
    total_correct = 0
    correct = np.array([0, 0, 0])
    n_class = np.array([0, 0, 0])

    with torch.no_grad():
        for batch_x, batch_y in val_loader:
            batch_x = batch_x.to(device)
            batch_y = batch_y.to(device)
            logits = model(batch_x)
            preds = torch.argmax(logits, dim=1)

            total_samples += batch_x.size(0)
            total_correct += (preds == batch_y).sum().item()

            # Classwise accuracy counts
            for pred, label in zip(preds, batch_y):
                if label == 0:
                    n_class[0] += 1
                    if pred == 0:
                        correct[0] += 1
                elif label == 1:
                    n_class[1] += 1
                    if pred == 1:
                        correct[1] += 1
                elif label == 2:
                    n_class[2] += 1
                    if pred == 2:
                        correct[2] += 1

    val_acc0.append(correct[0] / n_class[0])
    val_acc1.append(correct[1] / n_class[1])
    val_acc2.append(correct[2] / n_class[2])
    val_acc_tot.append(total_correct / total_samples)

    current_val_acc = val_acc_tot[-1]
    if current_val_acc > best_val_acc:
        best_val_acc = current_val_acc
        best_epoch = epoch + 1
        best_checkpoint = {
            "epoch": n_epochs,
            "model_state_dict": model.state_dict(),
            "train_losses": training_loss,
            "val_acc_tot": val_acc_tot,
```

```
            "val_acc0": val_acc0,
            "val_acc1": val_acc1,
            "val_acc2": val_acc2
        }
        torch.save(best_checkpoint, "best_problem3_checkpoint.pth")
```

Checkpointing is used to store the model state with the highest validation accuracy.

2.3 Results: Training From Scratch

Figure 1 shows the training loss and validation accuracies. The total validation accuracy peaks at epoch 46 with **78.85%**. The class-wise accuracies at this epoch are 96% (class 0), 72% (class 1), and 43% (class 2). This large spread is consistent with class imbalance in the dataset: about 50% of samples are class 0, while 33.8% and 16.2% are class 1 and 2, respectively.

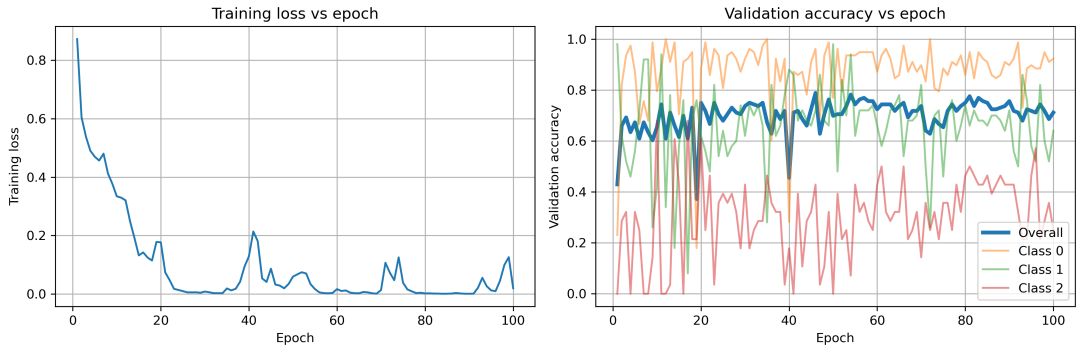


Figure 1: ResNet50 trained from scratch. Left: Training loss vs. epochs. Right: Total validation accuracy (blue), class 0 (yellow), class 1 (green), class 2 (red).

3 Pretrained DINO ResNet50

3.1 Model and Training

To improve performance, we use a DINO-pretrained ResNet50 [2], which outputs a 2048-dimensional feature vector. A linear classifier is added:

```
dino = torch.hub.load('facebookresearch/dino:main',
                      'dino_resnet50', pretrained=True)
model = nn.Sequential(dino, nn.Linear(2048, 3))
```

Initially AdamW is used as optimizer, but this yields a peak validation accuracy of 78.18%, worse than training from scratch. The DINO documentation instead uses SGD with weight decay, so we switch to SGD with learning rate 0.03, weight decay 10^{-4}, and train for 300 epochs.[1]

3.2 Results: DINO ResNet50

The results are shown in Figure 2. The total validation accuracy improves substantially, peaking at **85.26%** (epoch 243), a significant gain over the scratch model. The corresponding class-wise accuracies are 96% (class 0), 86% (class 1), and 54% (class 2). Compared to the scratch model, classes 1 and 2 improve by 14% and 11%, respectively, while class 0 remains stable.

Ideally the scratch model would also be retrained with the same optimizer and number of epochs for a fully fair comparison, but the present results already demonstrate the benefits of self-supervised pretraining in this setting.

[1] Hyperparameters follow the original solution and DINO documentation.

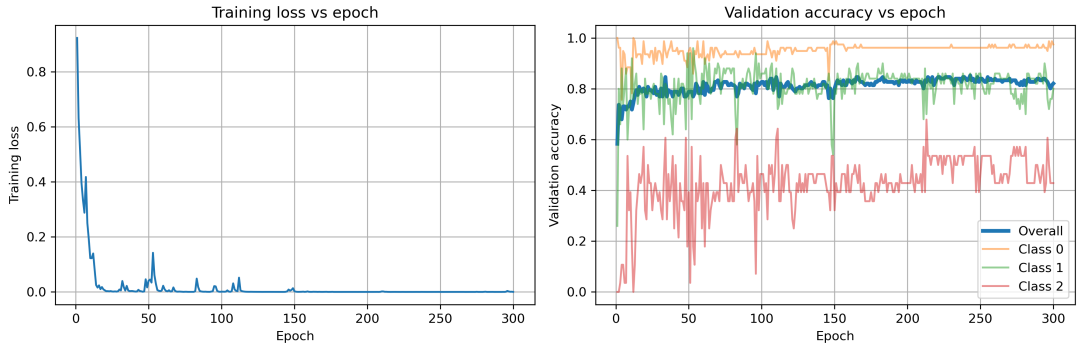


Figure 2: DINO-pretrained ResNet50. Left: Training loss vs. epochs. Right: Total validation accuracy (blue), class 0 (yellow), class 1 (green), class 2 (red).

4 Explainable AI With Grad-CAM

4.1 Motivation and Method

In medical applications, interpretability is crucial, since wrong decisions can have severe consequences. XAI aims to provide explanations for model predictions, for example by highlighting relevant regions in an image for a given classification. In our case, we want to understand why an image is classified as "tumor present" and whether the model focuses on the right regions.

Gradient-weighted Class Activation Mapping (Grad-CAM) uses gradients with respect to a target class c to highlight influential regions in a chosen feature map layer A^k [3]. For class score y^c (pre-softmax), the class-specific weights are

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k}, \tag{1}$$

where Z is a normalization constant and (i, j) indexes spatial positions. The Grad-CAM heatmap is then

$$L_{\text{Grad-CAM}}^c = \sum_k \alpha_k^c A^k, \tag{2}$$

followed by a ReLU to retain only positive influences.

4.2 Implementation for Tumor Classification

We use the PyTorch Grad-CAM library and select the last convolutional layer (layer4) of the DINO ResNet50 backbone as the target layer:

```
backbone = model[0]  # ResNet50 backbone
layer = backbone.layer4[-1]
grad_cam = GradCAM(model=model, target_layers=[layer])
```

Grad-CAM is applied to five validation images that are predicted as tumor. For each image, we choose the predicted class as the Grad-CAM target:

```
for i in range(0, 5):
    idx = val_idxs[i]
    img_path = val_file_names[idx]
    img = np.load(img_path)
    lbl = val_y[idx]

    # Predicted class as target
    target = val_preds[idx]
    targets = [ClassifierOutputTarget(target)]

    # Prepare tensor input
    X = torch.tensor(img, dtype=torch.float32) \
            .unsqueeze(0).unsqueeze(0).to(device)
    X = X.repeat(1, 3, 1, 1)

    gs_cam = grad_cam(input_tensor=X, targets=targets)
    # ... plotting code omitted
```

Figure 3 shows the resulting Grad-CAM overlays. The first two examples are misclassified (true label "liver"), while the last three are correctly predicted as "tumor". To an untrained eye, the highlighted blobs look quite similar across all examples, making it difficult to fully diagnose model failure modes without medical expertise.

5 From Grad-CAM to Pixelwise Masks

5.1 Thresholding and IoU Computation

To obtain pseudo-segmentation masks from Grad-CAM, we normalize each heatmap and apply a threshold in $[0, 1]$. Pixels above threshold form a binary mask, optionally refined with morphological operations. The Grad-CAM mask and IoU are computed as:

```python
def gradcam_pixelmask(gradcam, threshold):
    # Normalize GradCAM heatmap
    gradcam = (gradcam - gradcam.min()) / \
              (gradcam.max() - gradcam.min() + 1e-9)
    # Thresholding
    mask = np.where(gradcam >= threshold, 1, 0)

    # Morphological smoothing
    mask = binary_opening(mask, structure=np.ones((5, 5)))
    mask = binary_closing(mask, structure=np.ones((5, 5)))
    return mask

def iou_score(pred, true):
    pred_mask = pred.astype(bool)
    true_mask = true.astype(bool)
    inter = np.logical_and(pred_mask, true_mask).sum()
    union = np.logical_or(pred_mask, true_mask).sum()
    iou = inter / (union + 1e-9)
    return iou
```

We evaluate IoU over a range of thresholds and select the best mean IoU:

```python
thresholds = np.linspace(0.5, 0.99, 20)
best_iou_liver = 0
best_iou_tumor = 0
best_threshold_tumor = 0
best_threshold_liver = 0

for threshold in tqdm(thresholds):
    iou_tumor = []

    # Tumor-class images
    for i in range(len(val_file_names)):
        if val_y[i] == 2:
            img = np.load(val_file_names[i])
            true_labels = np.load(val_label_file_names[i])

            X = torch.tensor(img, dtype=torch.float32) \
                    .unsqueeze(0).unsqueeze(0).to(device) \
                    .repeat(1, 3, 1, 1)

            target_tumor = ClassifierOutputTarget(2)
            gradcam_tumor = grad_cam(input_tensor=X,
                                     targets=[target_tumor])[0]

            pred_mask_tumor = gradcam_pixelmask(gradcam_tumor, threshold)
            true_mask_tumor = np.where(true_labels == 2, 1, 0)

            iou_tumor.append(iou_score(pred_mask_tumor, true_mask_tumor))

    iou_tumor = np.array(iou_tumor)
    mean_iou_tumor = np.mean(iou_tumor)

    if mean_iou_tumor >= best_iou_tumor:
        best_iou_tumor = mean_iou_tumor
        best_threshold_tumor = threshold

    # ... analogous loop for liver class
```

5.2 Results and Discussion

The mean IoU scores for the best thresholds are summarized in Table 1. The performance is poor: 13.96% IoU for tumors and 11.69% for liver. This is expected, since Grad-CAM heatmaps are low-resolution blobs designed for visualization rather than fine-grained segmentation; tumor regions are typically small and fuzzy, whereas Grad-CAM tends to produce coarse, diffuse activations.

Method (Class)	Mean IoU
Grad-CAM (Tumor)	0.1396
Grad-CAM (Liver)	0.1169

Table 1: Mean IoU scores for Grad-CAM-based pseudo-segmentation masks on the validation set.

Across all validation samples, the highest IoU observed is 60.29%, while some samples achieve 0% IoU. The corresponding examples are shown in Figure 4. In the best case, the tumor location aligns well with the densest region of the Grad-CAM heatmap, but spurious weak activations in other regions reduce the IoU. In the worst case, the tumor is extremely small and the highlighted region is slightly offset.

Overall, these results support the view that Grad-CAM is better suited as a decision-support and visualization tool rather than a replacement for dedicated segmentation models such as U-Net or SAM [4, 5].

6 ConvNeXT for Liver CT Classification

6.1 Motivation and Model Setup

To further improve classification performance, we adopt ConvNeXT, a modern convolutional architecture that can be seen as an evolution of ResNet50 with several design improvements [6]. ConvNeXT uses modifications such as GELU activations, depthwise convolutions, larger kernels, and reduced number of activation and normalization layers. These changes yield consistent gains on ImageNet-1k and naturally motivate testing ConvNeXT on this CT classification task.

Class imbalance is handled by incorporating class weights into the cross-entropy loss, and label smoothing with factor 0.1 is used to reduce overconfidence. Hyperparameters are: learning rate $= 10^{-4}$, weight decay $= 0.01$, batch size $= 8$, and 50 epochs with checkpointing. AdamW is chosen as optimizer.

The ConvNeXT model is initialized as follows:

```
weights = ConvNeXt_Base_Weights.DEFAULT
model = convnext_base(weights=weights)
# Adapt the classifier to 3 classes
in_features = model.classifier[2].in_features
model.classifier[2] = nn.Linear(in_features, 3)
```

6.2 Data Preprocessing

ConvNeXT is pretrained on ImageNet, so we follow the same normalization pipeline. CT slices are converted to three channels and normalized using ImageNet mean and standard deviation:

```
# (N, 1, H, W) -> (N, 3, H, W)
train_x = train_x.unsqueeze(1).repeat(1, 3, 1, 1)

# Normalize to [0, 1]
train_x = train_x / 255

# ImageNet RGB mean and std
mean = torch.tensor([0.485, 0.456, 0.406]).view(1, 3, 1, 1)
std = torch.tensor([0.229, 0.224, 0.225]).view(1, 3, 1, 1)

train_x.sub_(mean)
train_x.div_(std)
```

6.3 Results: ConvNeXT

Training curves for ConvNeXT are shown in Figure 5. The peak validation accuracy is **88.46%** at epoch 3. Class-wise validation accuracies are summarized in Table 2. On the Kaggle test set, ConvNeXT achieves an accuracy of **90.909%**, clearly outperforming the previous ResNet-based models.

Class	Validation Accuracy (%)
Total	88.46
Background (0)	100.00
Liver (1)	78.00
Tumor (2)	75.00

Table 2: Validation accuracy of the ConvNeXT model per class.

7 Conclusion

This report presented a complete workflow for liver CT slice classification and Grad-CAM-based tumor localization. A ResNet50 trained from scratch provided a reasonable baseline, but DINO-pretrained ResNet50 and, especially, ConvNeXT significantly improved validation and test accuracies. Grad-CAM offered useful qualitative insight into model focus regions, but its low-resolution heatmaps limited its effectiveness as a stand-alone segmentation method. For clinical use, Grad-CAM is best seen as a visual decision-support tool, to be complemented with dedicated segmentation networks and expert radiologist review.

References

[1] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778.

[2] Mathilde Caron et al. "Emerging Properties in Self-Supervised Vision Transformers". In: *Proceedings of the International Conference on Computer Vision (ICCV)*. 2021.

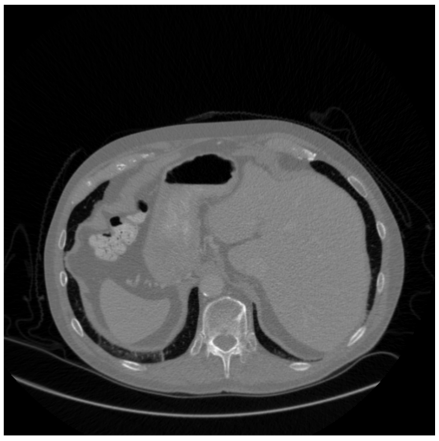
[3] Ramprasaath R. Selvaraju et al. "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization". In: *arXiv preprint arXiv:1610.02391* (2016).

[4] Tillmann Rheude et al. "Leveraging CAM Algorithms for Explaining Medical Semantic Segmentation". In: *arXiv preprint arXiv:2409.20287* (2024). URL: https://arxiv.org/abs/2409.20287.

[5] Wenjian Yao et al. "From CNN to Transformer: A Review of Medical Image Segmentation Models". In: *Journal of Imaging Informatics in Medicine* 37.4 (2024), pp. 1529–1547. DOI: 10.1007/s10278-024-00981-7. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC11300773/.

[6] Zhuang Liu et al. "A ConvNet for the 2020s". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022.

Image 12
True: liver, Pred: tumor



Grad-CAM target: tumor

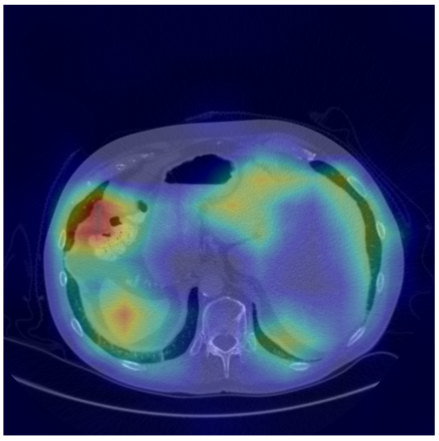
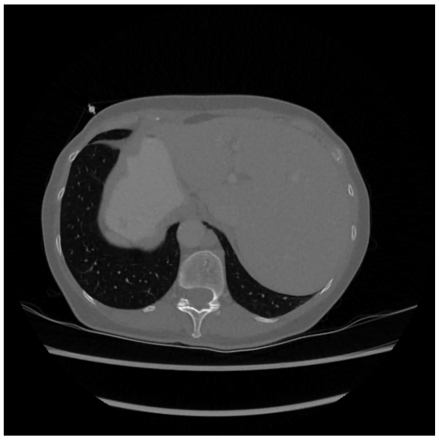


Image 14
True: liver, Pred: tumor



Grad-CAM target: tumor

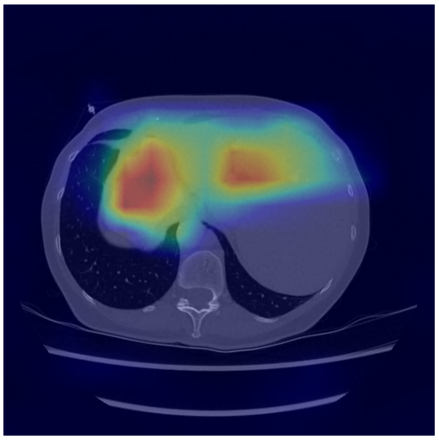
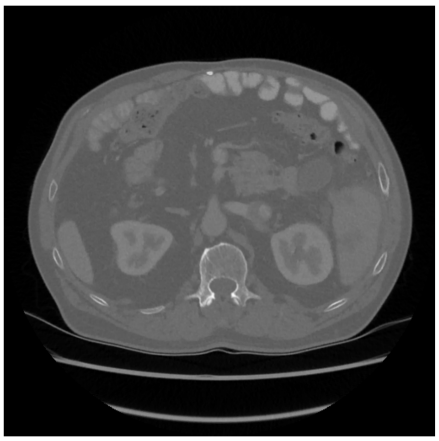


Image 20
True: tumor, Pred: tumor



Grad-CAM target: tumor

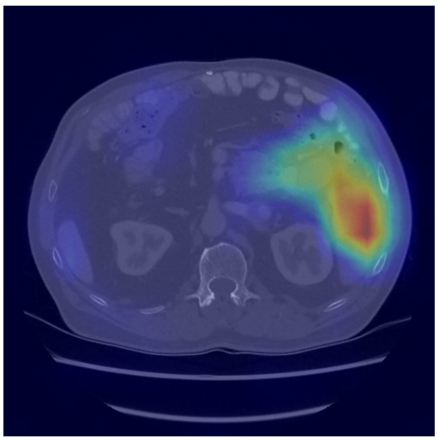
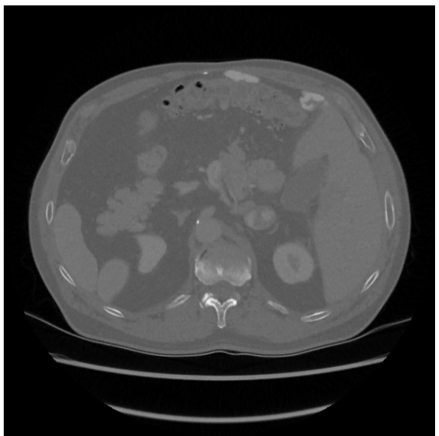
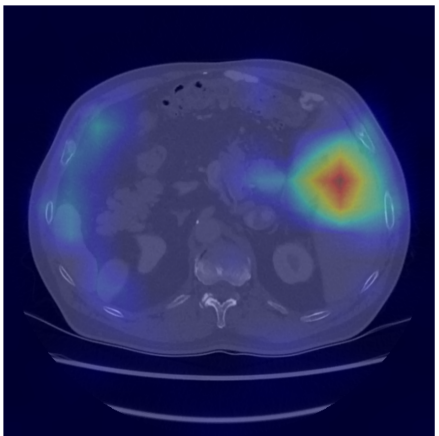


Image 21
True: tumor, Pred: tumor



Grad-CAM target: tumor



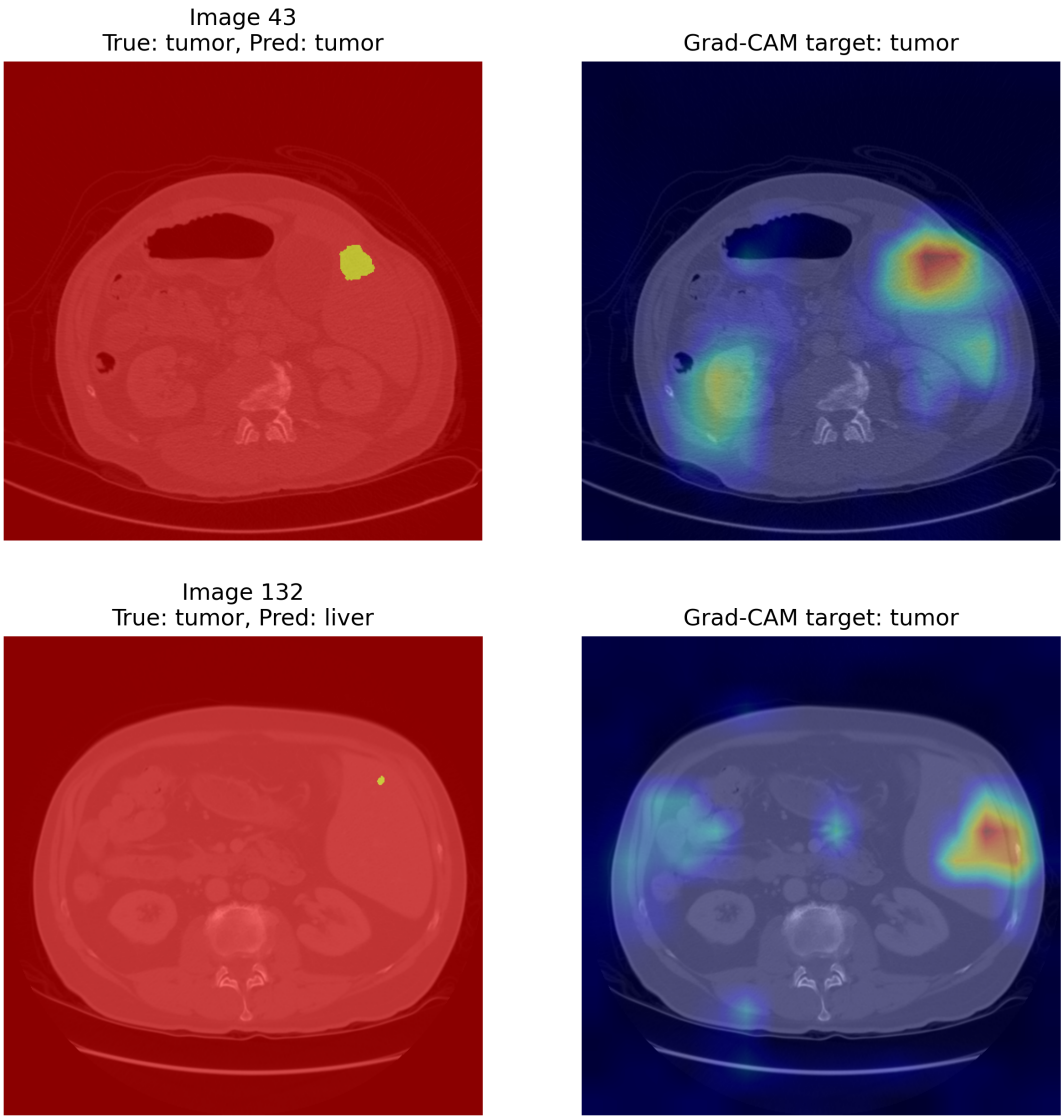


Figure 4: Top: Example with best IoU (left: ground truth tumor mask in yellow, right: Grad-CAM heatmap). Bottom: Example with 0 IoU.

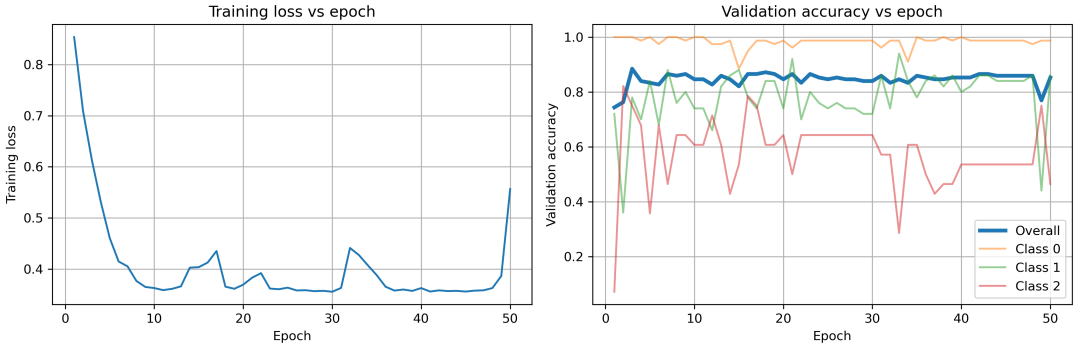


Figure 5: Training curves for the ConvNeXT model: training loss and validation accuracy versus epochs.