18/09/2025

PROYECTO
MATCHAI

A01710550 - Maxime Vilcocq Parra
A01710791 - Galo Alejandro del Rio Viggiano
A01369687 - Ana Karen Toscano Díaz
A01710367 - José Antonio López Saldaña

EQUIPO DE TRABAJO

Galo Alejandro del Rio Viggiano

A01710791

Maxime Vilcocq Parra

A01710550

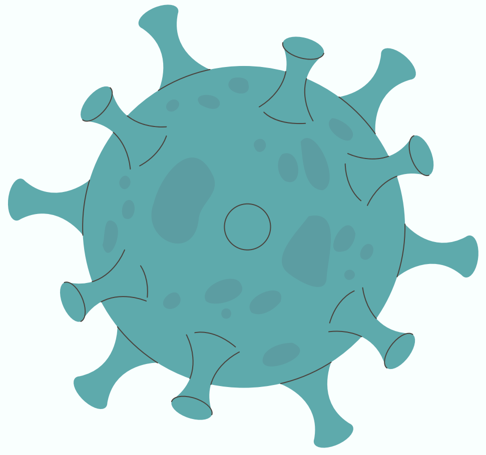
José Antonio López Saldaña

A01710367

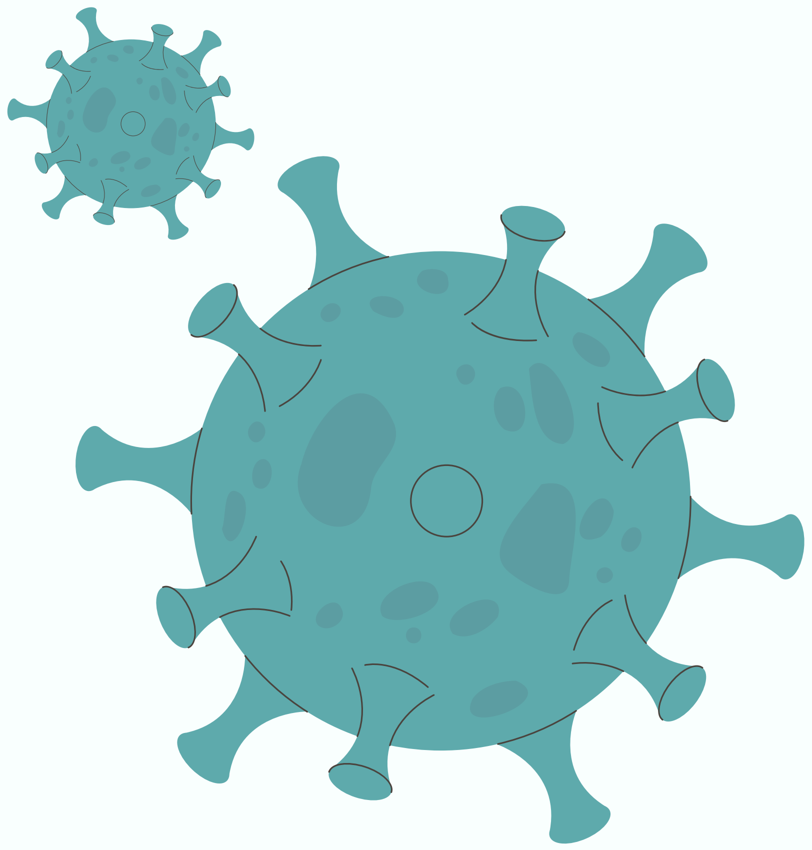
Ana Karen (Max) Toscano

A01369687

Sobre el Proyecto



CONTEXTO



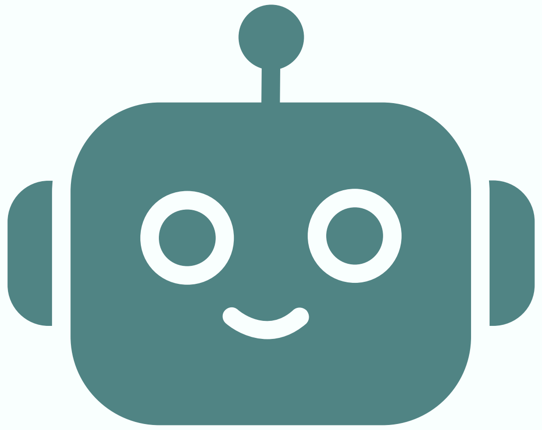
Sobre el Proyecto

NUESTRA SOLUCIÓN

OBJETIVOS

Detección

Precisión



Robustez

CÓDIGO IMPLEMENTADO

1. CARGA Y PREPROCESAMIENTO DE DATOS

- Carga de imágenes TAC y máscaras desde archivos .npy
- Normalización con windowing tipo CT
- Clipping intensidades (-1500 a 500 HU)

```python
# Load data

prefix = "/kaggle/input/covid-segmentation/"
images_medseg = np.load(prefix + "images_medseg.npy").astype(np.float32)
masks_medseg  = np.load(prefix + "masks_medseg.npy").astype(np.int16)

NUM_CLASSES = 4


# Preprocess

def preprocess(images: np.ndarray) -> np.ndarray:
    images = np.clip(images, -1500, 500)
    mean, std = images.mean(), images.std()
    return (images - mean) / (std + 1e-8)

images_medseg = preprocess(images_medseg)
```

2. DATA SET Y DATA LOADER

Creación de la clase LungDataset
- Convierte imágenes a tensores
- Asegura máscaras indexadas (H,W)

Separación en entrenamiento y validación

Uso de WeightedRandomSampler para balancear clases minoritarias

```python
# Dataset

class LungDataset(Dataset):
    def __init__(self, images, masks, aug=None):
        self.images, self.masks, self.aug = images, masks, aug
        self.to_tensor = T.ToTensor()  # HWC -> CHW, preserva float32

    def __len__(self):
        return len(self.images)

    def __getitem__(self, i):
        img, mask = self.images[i], self.masks[i]

        if self.aug: # aplicar augmentations si están definidos
            sample = self.aug(image=img, mask=mask)
            img, mask = sample["image"], sample["mask"]

        if img.ndim == 2: # si no hay canal explícito añadir (H, W, 1)
            img = img[..., None]

        x = self.to_tensor(img)

        if mask.ndim == 3 and mask.shape[-1] > 1: # si está one-hot pasar a índices
            mask = np.argmax(mask, axis=-1)
        y = torch.tensor(mask, dtype=torch.long)
        return x, y
```

```python
# Train/Val split

n_total = len(images_medseg)
n_val   = int(0.20 * n_total)
idxs = np.arange(n_total); np.random.shuffle(idxs)
train_idx, val_idx = idxs[n_val:], idxs[:n_val]

train_ds = LungDataset(images_medseg[train_idx], masks_medseg[train_idx], train_aug)
val_ds   = LungDataset(images_medseg[val_idx],   masks_medseg[val_idx],   val_aug)
```

```python
# Weighted sampler

def mask_has_lesion(mask):
    if mask.ndim == 3 and mask.shape[-1] > 1:
        mask = np.argmax(mask, axis=-1)
    return int(np.any(mask > 0))

minor_presence = np.array([mask_has_lesion(m) for m in masks_medseg[train_idx]])
sample_weights_np = np.where(minor_presence == 1, 3.0, 1.0).astype(np.float32)
sample_weights = torch.as_tensor(sample_weights_np, dtype=torch.double)
sampler = WeightedRandomSampler(sample_weights, num_samples=len(sample_weights), replacement=True)

BATCH = 6
pin = torch.cuda.is_available()
train_dl = DataLoader(train_ds, batch_size=BATCH, sampler=sampler, num_workers=2, pin_memory=pin)
val_dl   = DataLoader(val_ds,   batch_size=BATCH, shuffle=False, num_workers=2, pin_memory=pin)
```

3. AUGMENTATIONS

Entrenamiento (más variabilidad):

- Resize a 320×320
- Flips (horizontal, vertical)
- Rotaciones aleatorias (90°)
- Afine (escala, rotación, shear, traslación)
- Brillo y contraste aleatorio

Validación (solo resize para consistencia)

```python
# Augmentations

IMG_SIZE = 384
train_aug = A.Compose([
    A.Resize(IMG_SIZE, IMG_SIZE),
    A.HorizontalFlip(p=0.5),
    A.VerticalFlip(p=0.3),
    A.RandomRotate90(p=0.3),
    A.Affine(scale=(0.95,1.05), rotate=(-10,10), shear=(-5,5),
             translate_percent=(0.0,0.03), p=0.4),
    A.RandomBrightnessContrast(brightness_limit=0.10, contrast_limit=0.10, p=0.2),
])
val_aug = A.Compose([A.Resize(IMG_SIZE, IMG_SIZE)])
```

4. MODELO

- Arquitectura UNet++
- Encoder: EfficientNet-B3 preentrenado en ImageNet
- Entrada: 1 canal (grayscale)
- Salida: 4 clases (fondo + 2 lesiones)

```python
# Model

device = "cuda" if torch.cuda.is_available() else "cpu"
model = smp.UnetPlusPlus(
    encoder_name="timm-efficientnet-b3",
    encoder_weights="imagenet",
    in_channels=1,
    classes=NUM_CLASSES,
).to(device)
```

5. FUNCIONES DE PÉRDIDA

- CrossEntropy con pesos balanceados.
- Dice Loss para segmentación.
- Focal Loss para énfasis en clases difíciles ($\gamma = 2.5$).
- Ponderación: 0.25*CE + 0.5*Dice + 0.25*Focal

```python
# Definir combinación de pérdidas = CrossEntropy ponderada + Dice + Focal
ce_loss    = nn.CrossEntropyLoss(weight=ce_weights)
dice_loss  = smp.losses.DiceLoss(mode="multiclass")
focal_loss = smp.losses.FocalLoss(mode="multiclass", gamma=2.5)

def criterion(y_pred, y_true, a=0.25, b=0.5, c=0.25):
    return a*ce_loss(y_pred, y_true) + b*dice_loss(y_pred, y_true) + c*focal_loss(y_pred, y_true)
```

6. OPTIMIZACIÓN Y MÉTRICAS

- Optimizador: AdamW
- Scheduler: OneCycleLR (ajuste dinámico del LR)
- AMP (Mixed Precision) para acelerar entrenamiento
- Métrica principal: Coeficiente Dice por clase

```python
# Optimizer & Scheduler

opt = torch.optim.AdamW(model.parameters(), lr=1e-4, weight_decay=1e-4)
EPOCHS = 60
scheduler = torch.optim.lr_scheduler.OneCycleLR(
    opt, max_lr=5e-4, steps_per_epoch=len(train_dl), epochs=EPOCHS
)

# Configuración para entrenamiento mixto (AMP) con nueva API.
use_amp = (device == "cuda")
scaler = torch.amp.GradScaler('cuda') if use_amp else None


# Metrics

@torch.no_grad()
def dice_per_class(logits, target, num_classes=NUM_CLASSES):
    pred = logits.argmax(1)
    dices = []
    for c in range(num_classes):
        p = (pred == c).float()
        t = (target == c).float()
        inter = (p*t).sum()
        denom = p.sum() + t.sum()
        dices.append(1.0 if denom == 0 else (2*inter/denom).item())
    return dices
```

7. LOOP DE ENTRENAMIENTO

- Forward y backward pass con AMP
- Validación al final de cada epoch
- Guardado del mejor modelo (checkpoint) según Dice promedio

```python
# Training loop

best_mean_dice = -1.0
ckpt_path = "best_unetpp_b3_384.pth"

for epoch in range(1, EPOCHS+1):
    model.train()
    running = 0.0
    for x, y in train_dl:
        x, y = x.to(device, non_blocking=True), y.to(device, non_blocking=True)
        opt.zero_grad(set_to_none=True)

        if use_amp:
            with torch.amp.autocast('cuda'):
                out  = model(x)
                loss = criterion(out, y)
            scaler.scale(loss).backward()
            scaler.step(opt)
            scaler.update()
        else:
            out  = model(x)
            loss = criterion(out, y)
            loss.backward()
            opt.step()

        scheduler.step()
        running += loss.item()
    train_loss = running / max(len(train_dl), 1)
```

```python
model.eval()
vloss, vdices = 0.0, []
autocast_ctx = torch.amp.autocast('cuda') if use_amp else contextlib.nullcontext()
with torch.no_grad(), autocast_ctx:
    for x, y in val_dl:
        x, y = x.to(device), y.to(device)
        out  = model(x)
        loss = criterion(out, y)
        vloss += loss.item()
        vdices.append(dice_per_class(out, y))
vloss /= max(len(val_dl), 1)
vdices = np.mean(vdices, axis=0)
mean_dice = float(np.mean(vdices))

if mean_dice > best_mean_dice:
    best_mean_dice = mean_dice
    torch.save(model.state_dict(), ckpt_path)

print(f"Epoch {epoch:3d}/{EPOCHS} | Train {train_loss:.3f} | Val {vloss:.3f} | "
      f"Dice {np.round(vdices, 4)} | Mean {mean_dice:.4f} | Best {best_mean_dice:.4f}")
```

8. VISUALIZACIÓN DE RESULTADOS

Función visualize_batch

Muestra:

- Imagen TAC
- Máscara ground truth
- Predicción del modelo

```python
# Visualization

model.load_state_dict(torch.load(ckpt_path, map_location=device))
model.eval()

@torch.no_grad()
def visualize_batch(x_cpu, y_cpu, pred_cpu, max_images=6):
    n = min(max_images, x_cpu.size(0))
    for i in range(n):
        plt.figure(figsize=(12,4))
        plt.subplot(1,3,1); plt.imshow(x_cpu[i,0], cmap="gray"); plt.title("Image"); plt.axis("off")
        plt.subplot(1,3,2); plt.imshow(y_cpu[i], cmap="jet", alpha=0.7); plt.title("Mask GT"); plt.axis("off")
        plt.subplot(1,3,3); plt.imshow(pred_cpu[i], cmap="jet", alpha=0.7); plt.title("Prediction"); plt.axis("off")
        plt.show()

with torch.no_grad():
    for b_idx, (x, y) in enumerate(val_dl):
        x = x.to(device)
        out = model(x)
        pred = out.argmax(1).cpu()
        visualize_batch(x.cpu(), y, pred, max_images=8)
        if b_idx >= 4:  # mostrar primeras 5 batches
            break
```

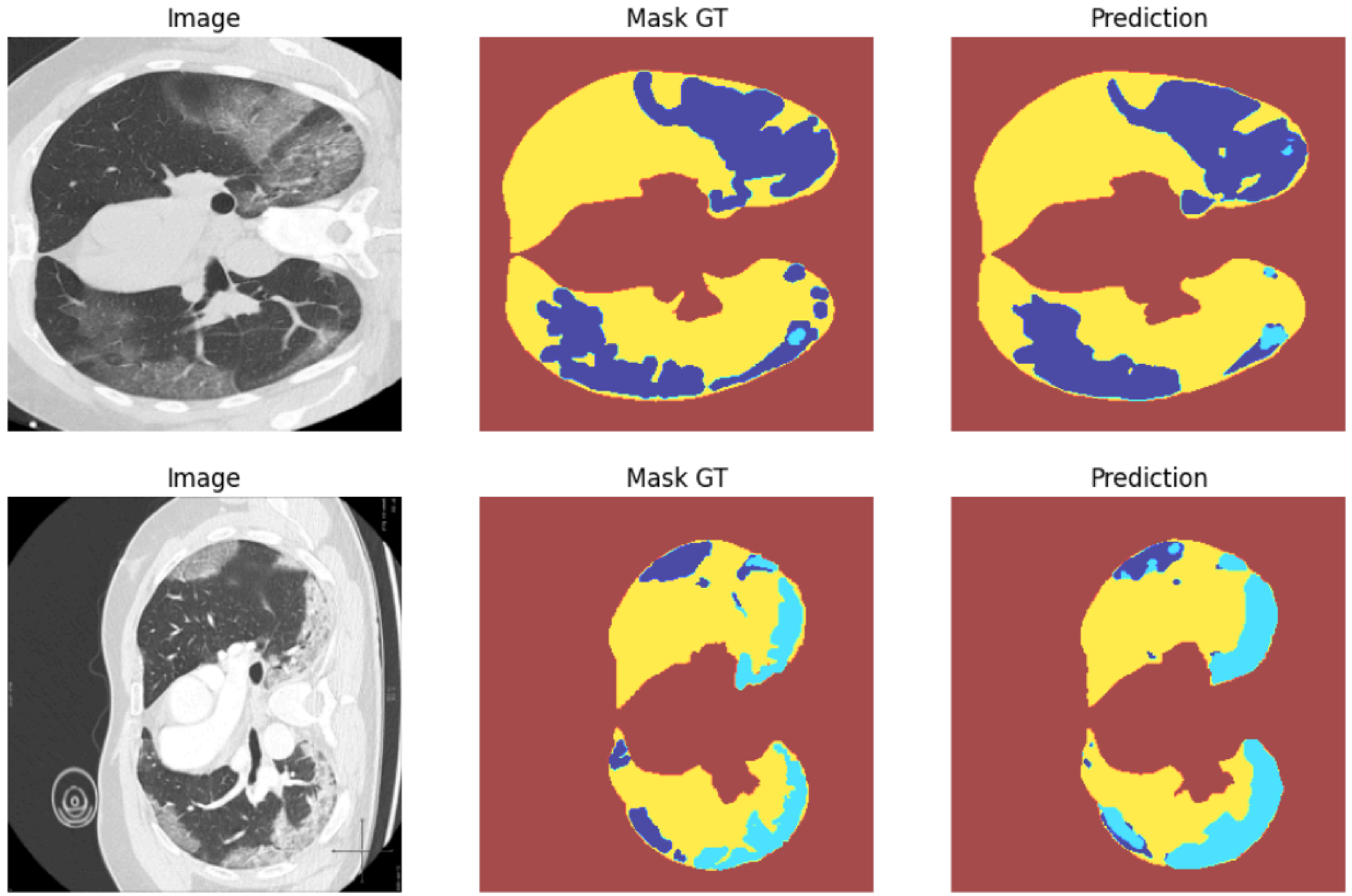


Image
Mask GT
Prediction
Image
Mask GT
Prediction

RESULTADOS

01 SE LOGRÓ UNA MEJORA PROGRESIVA EN LA MÉTRICA DICE A LO LARGO DE 60 ÉPOCAS.

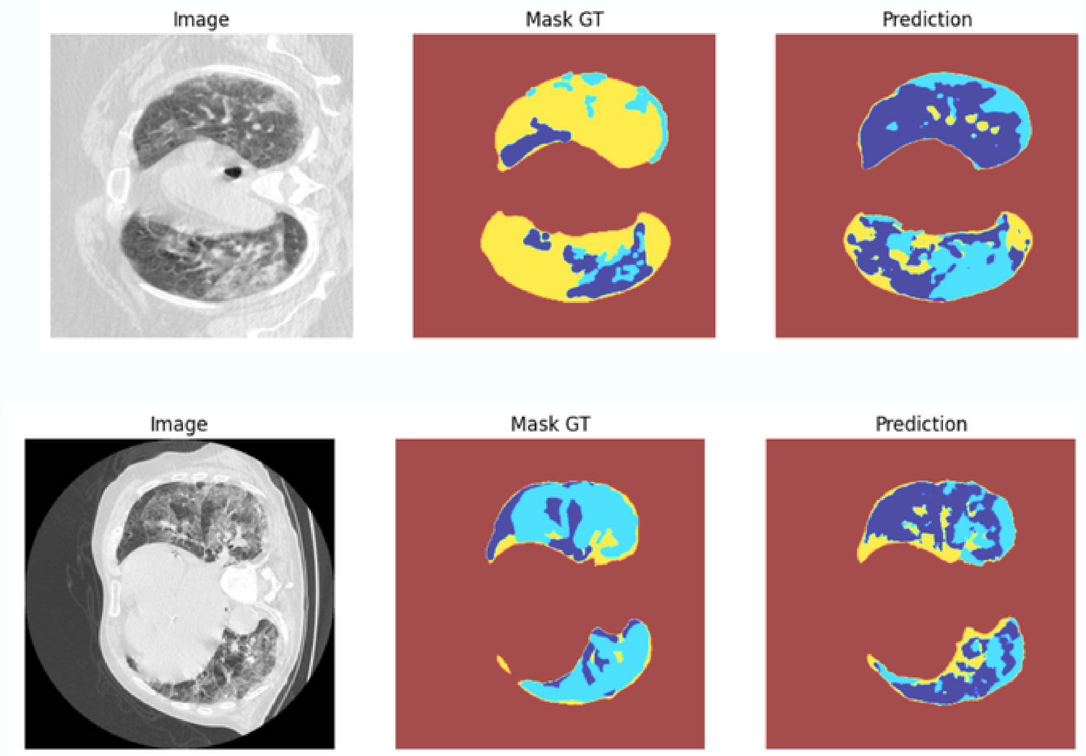
02 EL MODELO ALCANZÓ UN DICE PROMEDIO CERCANO AL 0.84, MOSTRANDO BUENA CAPACIDAD DE SEGMENTACIÓN EN LESIONES PULMONARES.

IMPLICACIONES ÉTICAS

MUCHAS

GRACIAS

Limitaciones del modelo



Why These Outliers Happen

1. **Ambiguity in the CT image**
   - Some slices genuinely look similar across classes (e.g. normal tissue vs. early-stage opacities).
   - Even human radiologists sometimes disagree on labeling.
2. **Label noise in the ground truth**
   - Masks in public COVID datasets are often annotated quickly.
   - Boundary regions (yellow vs. blue vs. cyan in your case) are **hard to separate**, and your model may be predicting something plausible that just doesn't match the annotation perfectly.
3. **Class imbalance still bites**
   - If one class is underrepresented (say cyan lesions are rare), the model may under- or over-predict it in specific cases.
4. **3D context is missing**
   - Your UNet++ only sees **one 2D slice at a time.**
   - In lungs, some patterns are only clear when you see adjacent slices. Without that, the model may confuse diffuse lesions vs. dense opacities.