# Progetto di Segmentazione delle Immagini Mediche con U-Net (1° Codice)

### 1) Descrizione del Progetto:

Il progetto ha come obiettivo la segmentazione delle immagini mediche usando una rete neurale **U-Net**, una delle architetture più diffuse per questo tipo di task.
In particolare, il modello è stato addestrato per segmentare immagini mediche, come quelle relative alla rilevazione di polipi o altre anomalie, da un dataset contenente immagini e maschere associate.
Il modello è in grado di generare maschere predette che evidenziano le aree di interesse nelle immagini originali, supportando potenzialmente l'analisi automatizzata dei dati medici.

---

# Librerie, Tecnologie e Motivazioni

### 1) TensorFlow e Keras

- **Utilizzo:**
  **TensorFlow** è utilizzato per la costruzione e l'addestramento della rete neurale U-Net, mentre **Keras**, che fa parte di TensorFlow, fornisce un'interfaccia semplice per definire il modello e gestire i processi di addestramento e validazione.
  Inoltre, TensorFlow viene configurato per rilevare e utilizzare una TPU (Tensor Processing Unit) se disponibile, garantendo un'esecuzione rapida dei calcoli.

- **Motivazione:**
  TensorFlow è una delle librerie più potenti e scalabili per il deep learning. L'uso delle TPU accelera notevolmente il training, specialmente per modelli complessi come la U-Net.

---

### 2) Numpy

- **Utilizzo:**
  Usata per manipolare i dati numerici, come la conversione delle immagini e maschere in array numerici e il calcolo di metriche come **IoU** (Intersection over Union) e **Dice Coefficient**.

- **Motivazione:**
  **Numpy** è una libreria essenziale per il trattamento di array e matrici multidimensionali, fornendo un supporto rapido ed efficiente per il **preprocessing** e il calcolo di metriche personalizzate.

---

### 3) Matplotlib

- **Utilizzo:**
  Utilizzata per visualizzare grafici delle metriche durante l'addestramento (es. **loss**, **accuracy**, **IoU**, e **Dice Coefficient**).

- **Motivazione:**
  **Matplotlib** è ideale per generare visualizzazioni personalizzate che aiutano a monitorare e comprendere il comportamento del modello durante l'addestramento.

---

### 4) Scikit-learn

- **Utilizzo:**
  Impiegata per suddividere il dataset in training e validation set utilizzando il metodo **train_test_split.**

- **Motivazione:**
  **Scikit-learn** fornisce strumenti consolidati e affidabili per la gestione dei dati, come la creazione di set di addestramento e validazione in modo casuale, ma bilanciato.

---

## 5) Funzioni e Strategie di TPU

- **Utilizzo:**
  Nel codice fornito, viene configurata una TPU (Tensor Processing Unit) se disponibile, utilizzando il modulo **tf.distribute.cluster_resolver.TPUClusterResolver**. Questo permette di sfruttare una potenza computazionale superiore rispetto a una CPU o GPU standard, particolarmente utile per allenare modelli complessi come una U-Net.
  In assenza di TPU, viene utilizzata una strategia **fallback** che sfrutta CPU o GPU grazie a **tf.distribute.get_strategy().**

- **Motivazione:**
  L'uso di TPU garantisce una significativa accelerazione nei calcoli, permettendo di addestrare modelli più complessi in tempi ridotti rispetto all'utilizzo di CPU/GPU.

---

## 6) Funzioni Personalizzate

### 6.1) Preprocessing del Dataset

Il dataset è costituito da immagini e maschere localizzate in due directory diverse. Attraverso la funzione **load_images_and_masks**, le immagini e le maschere vengono:

- Caricate e ridimensionate a 256x256 pixel con **tf.keras.preprocessing.image.load_img.**
- Convertite in array e normalizzate tra 0 e 1 usando **tf.keras.preprocessing.image.img_to_array.**

I dati vengono successivamente suddivisi in training e validation set tramite train_test_split di Scikit-learn, mantenendo un bilanciamento tra i due insiemi.

- **Motivazione:**
  La normalizzazione aiuta la rete a convergere più rapidamente, riducendo i problemi di instabilità numerica. Il ridimensionamento garantisce uniformità tra i dati in input.

- **6.2) Funzioni per il calcolo delle metriche**
  - **iou_score:** Calcola la metrica Intersection over Union.
  - **dice_coefficient:** Calcola il Dice Coefficient.
  - **evaluate_metrics:** Valuta IoU e Dice Coefficient su un dataset.

```python
# Funzione per calcolare IoU
def iou_score(y_true, y_pred, smooth=1e-7):
    intersection = np.sum(y_true * y_pred)
    union = np.sum(y_true) + np.sum(y_pred) - intersection
    return (intersection + smooth) / (union + smooth)

# Funzione per calcolare DICE coefficient
def dice_coefficient(y_true, y_pred, smooth=1e-7):
    intersection = np.sum(y_true * y_pred)
    return (2. * intersection + smooth) / (np.sum(y_true) + np.sum(y_pred) + smooth)

# Funzione per calcolare le metriche su un dataset
def evaluate_metrics(model, X, y, threshold=0.5):
    predictions = model.predict(X)
    predictions = (predictions > threshold).astype(np.float32)  # Binarizzazione delle previsioni

    iou_scores = []
    dice_scores = []

    for true, pred in zip(y, predictions):
        iou_scores.append(iou_score(true, pred))
        dice_scores.append(dice_coefficient(true, pred))

    mean_iou = np.mean(iou_scores)
    mean_dice = np.mean(dice_scores)

    return mean_iou, mean_dice, predictions
```

**Motivazione:** Queste funzioni personalizzate consentono di calcolare metriche avanzate specifiche per i problemi di segmentazione.

- **6.3) MetricsCallback:**

    - Un **callback** personalizzato di TensorFlow per calcolare e memorizzare IoU e Dice Coefficient al termine di ogni epoca di addestramento

```
# Callback personalizzato per memorizzare IoU e DICE durante l'addestramento
class MetricsCallback(tf.keras.callbacks.Callback):
    def __init__(self, val_data):
        self.val_data = val_data
        self.iou_scores = []
        self.dice_scores = []

    def on_epoch_end(self, epoch, logs=None):
        X_val, y_val = self.val_data
        predictions = self.model.predict(X_val)
        predictions = (predictions > 0.5).astype(np.float32)  # Binarizzazione

        iou = np.mean([iou_score(y_true, y_pred) for y_true, y_pred in zip(y_val, predictions)])
        dice = np.mean([dice_coefficient(y_true, y_pred) for y_true, y_pred in zip(y_val, predict
ions)])

        self.iou_scores.append(iou)
        self.dice_scores.append(dice)

        print(f"Epoch {epoch+1}: Mean IoU = {iou:.4f}, Mean DICE = {dice:.4f}")
```

**Motivazione:** Monitorare metriche avanzate durante il training aiuta a valutare in tempo reale l'efficacia del modello e identificare eventuali problemi di overfitting o underfitting.

---

## 7) Architettura U-Net

- **Utilizzo:**
  La rete **U-Net** è composta da un **encoder** (per estrarre caratteristiche dalle immagini) e un **decoder** (per ricostruire la maschera segmentata), con skip connections per preservare dettagli spaziali.
  Le convoluzioni sono implementate con **Conv2D**, i pool con **MaxPooling2D** e gli upsample con **Conv2DTranspose**.

**La U-Net implementata segue un design classico:**

1. **Encoder:**
   Utilizza blocchi convolutivi e operazioni di pooling per estrarre feature rappresentative riducendo progressivamente la risoluzione spaziale.

2. **Decoder:**
   Ricostruisce l'immagine segmentata utilizzando convoluzioni trasposte (Conv2DTranspose) e skip connections per recuperare i dettagli persi durante l'encoding.

3. **Output:**
   L'ultimo livello usa una convoluzione con attivazione **sigmoide** per produrre una maschera binaria di probabilità.

- **Motivazione:**
  La U-Net è una delle architetture più efficienti per la segmentazione di immagini mediche grazie alla sua capacità di combinare contesto globale (attraverso il bottleneck) e dettagli locali (grazie alle skip connections).

---

## 8) Ottimizzatore Adam

- **Utilizzo:**

  Il modello usa l'ottimizzatore **Adam** con un learning rate iniziale di $1 \times 10^{-4}$ e la funzione di perdita **binary_crossentropy**.
  Adam è scelto per la sua capacità di adattarsi dinamicamente ai gradienti, mentre la cross-entropia binaria è adatta a problemi di classificazione binaria come la segmentazione.

- **Motivazione:**
  Queste scelte garantiscono stabilità e una rapida convergenza durante l'addestramento.

- **Motivazione:**
  Adam è un ottimizzatore adattivo che garantisce una discesa rapida del gradiente, ideale per reti profonde come la U-Net.

---

## 9) Funzione di perdita

- **Utilizzo:**
  La **binary_crossentropy** è scelta come funzione di perdita, essendo specifica per problemi di classificazione binaria (es. presenza o assenza di una regione segmentata).

- **Motivazione:**
  La binary_crossentropy è particolarmente efficace per minimizzare le discrepanze tra maschere predette e maschere reali in problemi di segmentazione binaria.

---

# 10) Salvataggio del Modello

- **Utilizzo:**
  Il modello addestrato viene salvato in formato **.keras** per un utilizzo futuro.

  **Motivazione:**
  Il salvataggio del modello consente di riutilizzarlo per inferenze o ulteriori addestramenti senza doverlo ricostruire da zero.

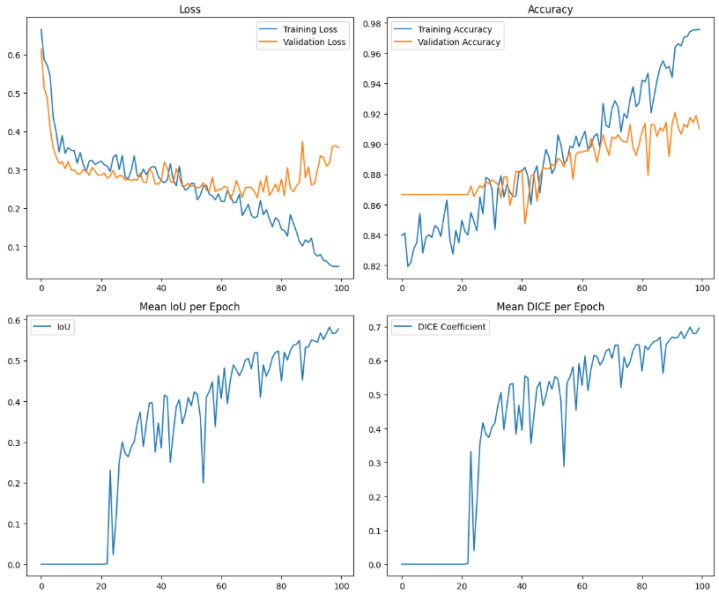
---

# 11)Visualizzazione delle Metriche

Alla fine dell'addestramento, il codice genera grafici per:

- **Perdite:** Training e validation loss per osservare eventuali divergenze.
- **Accuratezza:** Differenza tra performance su training e validation set.
- **IoU e Dice:** Evoluzione delle metriche specifiche per la segmentazione.
- **Motivazione:**
  La visualizzazione è uno strumento essenziale per interpretare e ottimizzare il comportamento del modello

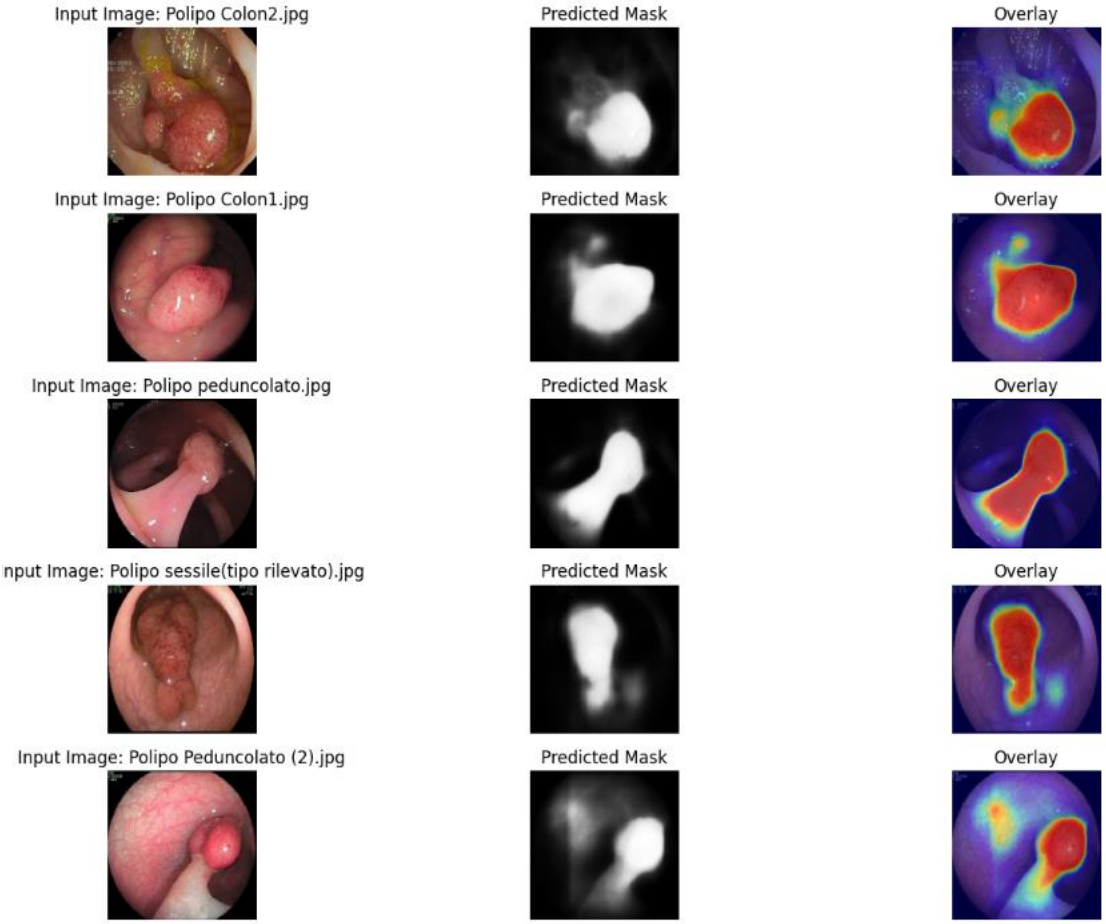
```
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 76ms/step  - accuracy: 0.9759 - loss
Epoch 100: Mean IoU = 0.5776, Mean DICE = 0.6955
7/7 ━━━━━━━━━━━━━━━━━━━━ 5s 688ms/step - accuracy: 0.9759 - loss: 0.0472 - val_accuracy: 0.9102
- val_loss: 0.3581
```




---

## 12) Predizioni su Nuove Immagini:

Il modello viene poi richiamato per fare previsioni su nuove immagini.
Le maschere predette vengono visualizzate affiancate alle immagini originali per una facile interpretazione.



## 13) Metriche di Performance:

Nel contesto della segmentazione delle immagini, è fondamentale scegliere le metriche giuste per valutare l'efficacia del modello. Come gia visto, per questo progetto sono state utilizzate le seguenti metriche:

## Dice Coefficient

Il Dice Coefficient è una metrica particolarmente adatta per problemi di segmentazione binaria, dove l'obiettivo è confrontare la maschera predetta e quella reale. Varia tra 0 e 1, con 1 che rappresenta una perfetta corrispondenza. La formula è la seguente:

$$\boldsymbol{DICE\ COEFFICIENT} = \frac{\mathbf{2}|\boldsymbol{A} \cap \boldsymbol{B}|}{|\boldsymbol{A}| + |\boldsymbol{B}|}$$

Dove:

- A è la maschera predetta,
- B è la maschera reale,
- |A| e |B| sono le aree di A e B rispettivamente,
- |A∩B|| è l'area di sovrapposizione tra A e B.

### Perché è adatto?

La segmentazione binaria richiede una metrica che valuti quanto precisamente il modello riesce a identificare le aree di interesse.
Il Dice Coefficient è una metrica molto popolare in applicazioni di segmentazione, come la segmentazione di polipi o altre anomalie nelle immagini mediche, poiché penalizza maggiormente le piccole discrepanze tra la maschera predetta e quella reale.
Inoltre, è particolarmente utile quando i dati sono sbilanciati, come accade spesso in segmentazioni di immagini mediche, dove l'area di interesse (ad esempio, un polipo) occupa solo una piccola porzione dell'immagine.

## Intersection over Union (IoU):

L'Intersection over Union (IoU) è un'altra metrica ampiamente utilizzata nella segmentazione. Essa misura l'intersezione tra la maschera predetta e quella reale, divisa per l'unione di entrambe. La formula è la seguente:

$$\boldsymbol{IoU} = \frac{|\boldsymbol{A} \cap \boldsymbol{B}|}{|\boldsymbol{A} \cup \boldsymbol{B}|}$$

Dove:

- A è la maschera predetta,
- B è la maschera reale,
- |A∪B| è l'unione di A e B.

## Perché è adatto?

L'IoU fornisce una valutazione più severa rispetto al Dice Coefficient, poiché calcola direttamente quanto le due maschere (predetta e reale) si sovrappongono rispetto alla loro unione.
Mentre il Dice Coefficient tende a premiare il modello per una maggiore sovrapposizione, l'IoU punisce i falsi positivi in modo più evidente.
Questo lo rende particolarmente utile quando si desidera evitare che il modello produca molte segmentazioni false.

## Accuracy:

La precisione (accuracy) è una metrica generale che misura la percentuale di pixel correttamente classificati come appartenenti all'area di interesse o meno (sia veri positivi che veri negativi). La formula è la seguente:

$$\textbf{Accuracy} = \frac{\boldsymbol{TP+TN}}{\boldsymbol{TP+TN+FP+FN}}$$

Dove:

- **TP** (True Positives) è il numero di pixel correttamente classificati come appartenenti all'area di interesse,
- **TN** (True Negatives) è il numero di pixel correttamente classificati come non appartenenti all'area di interesse,
- **FP** (False Positives) è il numero di pixel erroneamente classificati come appartenenti all'area di interesse,
- **FN** (False Negatives) è il numero di pixel erroneamente classificati come non appartenenti all'area di interesse.

**È adatto?**

Sebbene l'accuracy possa essere una metrica utile in contesti generali, nel caso di segmentazione delle immagini mediche, potrebbe non essere la metrica migliore da sola, specialmente se i dati sono sbilanciati. Tuttavia, è comunque utile come metrica di riferimento, poiché fornisce una misura generale di come il modello stia operando.

## 14)Punti di Forza del Codice

1. Configurazione automatica di TPU/CPU/GPU per ottimizzare le risorse.
2. Uso di metriche avanzate specifiche per la segmentazione.
3. Implementazione di U-Net con un'architettura consolidata per risultati robusti.
4. Salvaguardia e riutilizzo del modello con opzioni di salvataggio.
5. Visualizzazione delle metriche per facilitare l'interpretazione.