Spiegazione del codice fornito (2° Codice)

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.

Configurazione del dispositivo di esecuzione (TPU/GPU/CPU)

Il codice inizia configurando il dispositivo di esecuzione. Si tenta di rilevare una TPU (Tensor Processing Unit) e, in caso di indisponibilità, si utilizza la strategia predefinita basata su GPU o CPU. Questa configurazione sfrutta le funzionalità distribuite di TensorFlow per accelerare il processo di addestramento del modello.

- **Perché è importante?**
  Le TPU accelerano enormemente i calcoli, soprattutto per operazioni su array numerici di grandi dimensioni come immagini. Se una TPU non è disponibile, il fallback su GPU/CPU garantisce comunque l'esecuzione del codice.

```
# Configurazione TPU
try:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()  # Verifica se TPU è disponibile
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    tpu_strategy = tf.distribute.TPUStrategy(tpu)
    print("TPU rilevata e configurata correttamente!")
except ValueError:
    tpu_strategy = tf.distribute.get_strategy()  # Usa CPU/GPU come fallback
    print("TPU non rilevata. Utilizzo CPU/GPU.")
```

Caricamento e pre-elaborazione dei dati

Il dataset utilizzato proviene da **Kvasir-SEG**, una raccolta di immagini mediche e relative maschere di segmentazione. La funzione **load_images_and_masks** esegue il caricamento delle immagini, la ridimensiona a 256x256 pixel e normalizza i valori dei pixel nell'intervallo [0, 1].

- **Perché ridimensionare e normalizzare?**
    - Il ridimensionamento garantisce che tutte le immagini abbiano la stessa dimensione, necessaria per un input coerente alla rete neurale.
    - La normalizzazione facilita la convergenza durante l'addestramento, poiché i modelli convergono più rapidamente quando i dati hanno valori compresi in un intervallo piccolo e standard.

Dopo il caricamento, il dataset viene diviso in **training set** (80%) e **validation set** (20%) usando la funzione train_test_split di sklearn.

Architettura del modello: U-Net

Il modello utilizza una variante della rete neurale U-Net, particolarmente adatta per compiti di segmentazione semantica. La rete è composta da:

- **Encoder**: Estrae feature attraverso convoluzioni e downsampling.
- **Bottleneck**: Una rappresentazione compatta delle feature.
- **Decoder**: Ricostruisce l'immagine segmentata tramite transposed convolutions e concatenazioni con le feature degli strati encoder corrispondenti.
- **Perché U-Net?**
  U-Net è altamente efficace per la segmentazione semantica, in quanto combina informazioni globali (encoder) e locali (skip connections) per ottenere maschere dettagliate.

Addestramento del modello

Il modello viene compilato con:

- **Ottimizzatore:** Adam con un learning rate di 10^{-4} ,scelto per la sua velocità di convergenza.
- **Funzione di perdita:** binary_crossentropy, adatta per la segmentazione binaria.
- **Metriche**: Accuratezza standard e metriche personalizzate calcolate post-addestramento.

Durante l'addestramento, i dati di training e validation vengono forniti in batch di 128 immagini per 100 epoche.

```
# Compilazione e addestramento del modello
with tpu_strategy.scope():
    model = unet_model()
    model.compile(optimizer=Adam(learning_rate=1e-4), loss="binary_crossentropy", metrics=["accuracy"])

    # Addestramento del modello
    history = model.fit(
        X_train, y_train,
        validation_data=(X_val, y_val),
        batch_size=BATCH_SIZE,
        epochs=EPOCHS
    )
```

Metriche di valutazione: IoU e DICE

Due metriche chiave per valutare la qualità della segmentazione sono:

1. **IoU (Intersection over Union)**: La sovrapposizione tra la maschera predetta e quella reale.
2. **DICE coefficient**: Una misura di similarità, particolarmente sensibile per immagini con squilibri tra classi.

- **Perché usare IoU e DICE?**
  Entrambe le metriche sono robuste e forniscono un'idea chiara della qualità della segmentazione.

Valutazione e visualizzazione

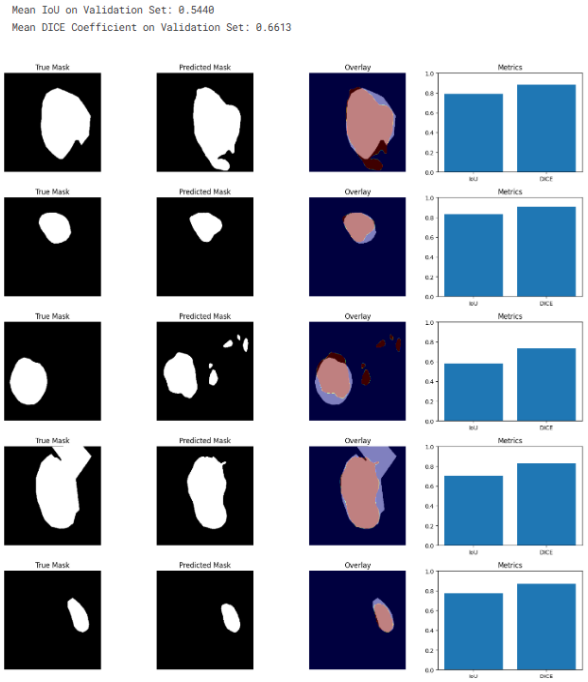
Le maschere predette vengono confrontate con quelle reali per calcolare le metriche di IoU e DICE su tutto il validation set. Inoltre, il codice include una funzione per plottare:

1. Le maschere reali e predette.
2. Le sovrapposizioni tra maschere.
3. Grafici a barre di IoU e DICE per immagini campione.

Code:

```python
# Visualizzazione grafica di IoU e DICE per alcune immagini
def plot_metrics(y_true, y_pred, predictions, num_samples=5):
    indices = np.random.choice(len(y_true), num_samples, replace=False)
    plt.figure(figsize=(15, 15))

    for i, idx in enumerate(indices):
        plt.subplot(num_samples, 4, i * 4 + 1)
        plt.imshow(y_true[idx].squeeze(), cmap='gray')
        plt.title("True Mask")
        plt.axis("off")

        plt.subplot(num_samples, 4, i * 4 + 2)
        plt.imshow(predictions[idx].squeeze(), cmap='gray')
        plt.title("Predicted Mask")
        plt.axis("off")

        plt.subplot(num_samples, 4, i * 4 + 3)
        plt.imshow(y_true[idx].squeeze(), cmap='gray')
        plt.imshow(predictions[idx].squeeze(), cmap='jet', alpha=0.5)
        plt.title("Overlay")
        plt.axis("off")

        # IoU e DICE per questa immagine
        img_iou = iou_score(y_true[idx], y_pred[idx])
        img_dice = dice_coefficient(y_true[idx], y_pred[idx])

        plt.subplot(num_samples, 4, i * 4 + 4)
        plt.bar(["IoU", "DICE"], [img_iou, img_dice])
        plt.title("Metrics")
        plt.ylim(0, 1)

    plt.tight_layout()
    plt.show()
```

Plots:



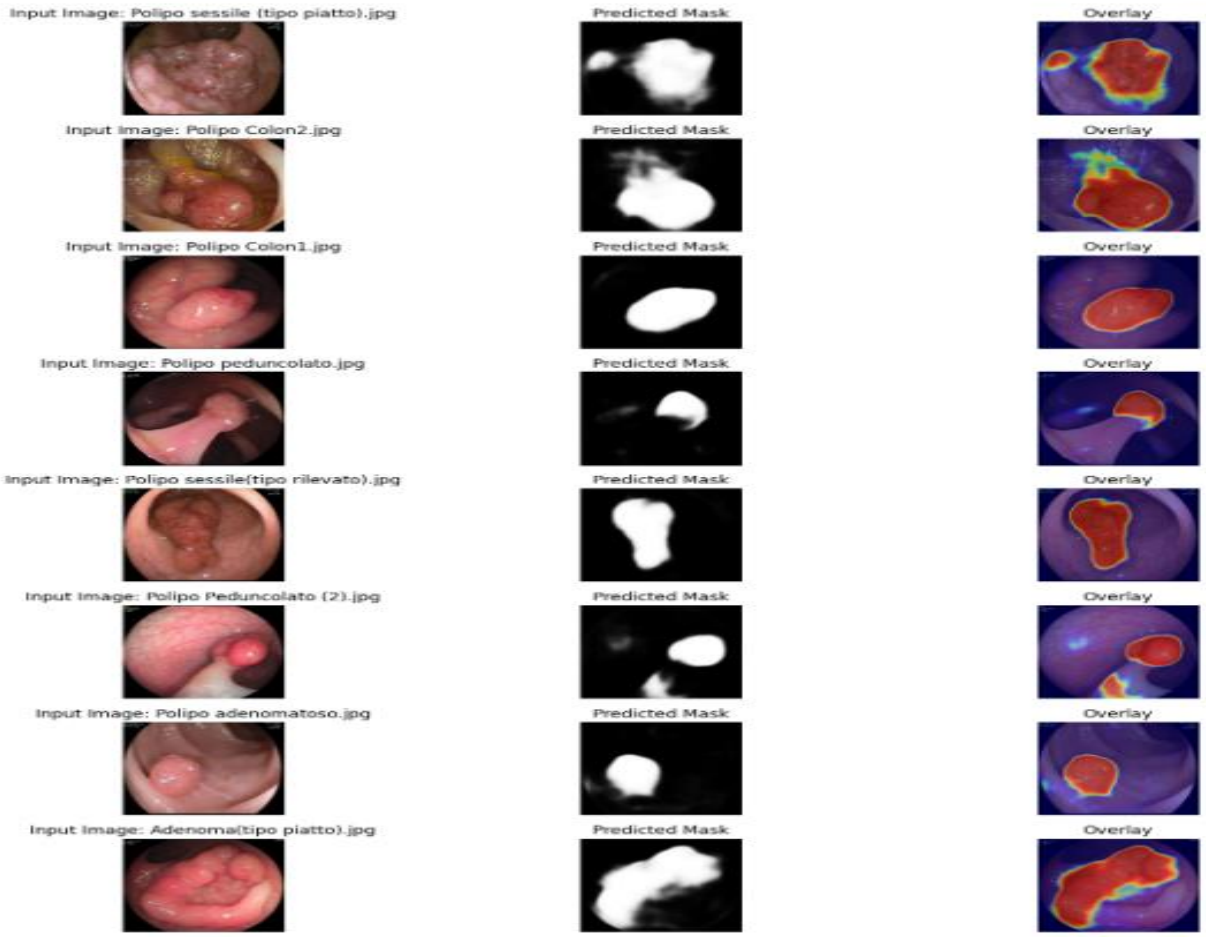
- **Perché visualizzare i risultati?**
  La visualizzazione è cruciale per comprendere qualitativamente l'output del modello e identificare eventuali aree di miglioramento.

Salvataggio del modello

Il modello addestrato viene salvato in formato Keras, consentendo di caricarlo per uso futuro o ulteriori esperimenti.

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.



Conclusione

Il codice realizza un'intera pipeline per la segmentazione semantica basata su U-Net, ottimizzando sia l'addestramento sia la valutazione. Grazie all'uso di TPU/GPU, metriche robuste (IoU e DICE) e visualizzazioni dettagliate, si ha un esempio pratico implementato per problemi di segmentazione medica.