Esperimenti

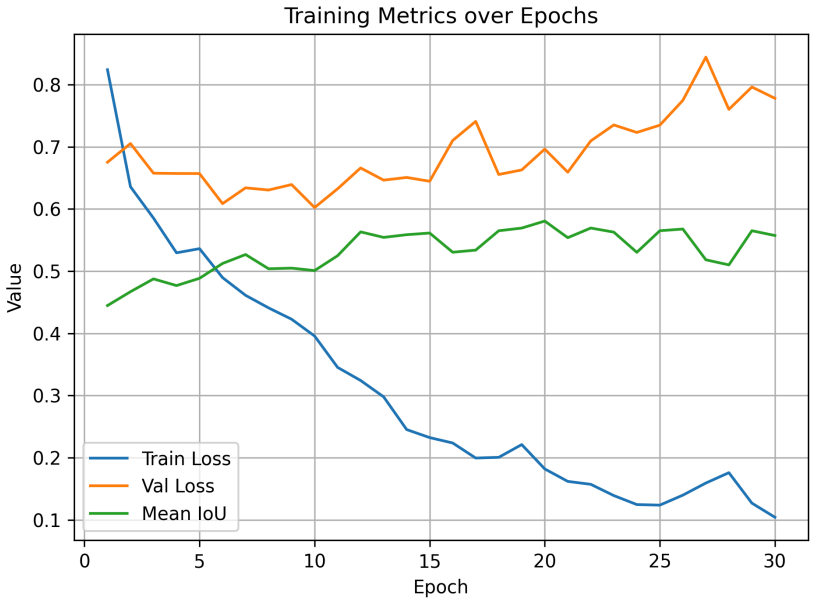
deeplabv3plus_resnet101

✳ Esperimento: `deeplabv3plus_resnet101 - dlprn1010000`

- **Architettura**: `deeplabv3plus_resnet101`
- **Backbone Weights**: best_deeplabv3plus_resnet101_cityscapes_os16.pth
- **Input size**: 256x256
- **Batch size**: 2
- **Augmentation**: off
- **Scheduler**: off
- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy.
- **Main classifier Kernel size:** 1
- **Valutazione risultati**:
    - Migliore Validation Loss ottenuta all'epoca 10:
    - 📊 Train Loss: 0.3957 | Val Loss: 0.6023 | Mean IoU: 0.5009
        - ▸ IoU classe 0: 0.5078
        - ▸ IoU classe 1: 0.5474
        - ▸ IoU classe 2: 0.5975
        - ▸ IoU classe 3: 0.5411
        - ▸ IoU classe 4: 0.0020
        - ▸ IoU classe 5: 0.3486
        - ▸ IoU classe 6: 0.2300
        - ▸ IoU classe 7: 0.8328
        - ▸ IoU classe 8: 0.9079
    -
    - Migliore mean IoU ottenuta all'epoca 20:
    - 📊 Train Loss: 0.1819 | Val Loss: 0.6963 | Mean IoU: 0.5805
        - ▸ IoU classe 0: 0.4769

    - ▸ IoU classe 1: 0.6141
    - ▸ IoU classe 2: 0.5972
    - ▸ IoU classe 3: 0.5399
    - ▸ IoU classe 4: 0.3949
    - ▸ IoU classe 5: 0.4916
    - ▸ IoU classe 6: 0.2641
    - ▸ IoU classe 7: 0.8350
    - ▸ IoU classe 8: 0.9075

- **Rappresentazione grafica del training:**

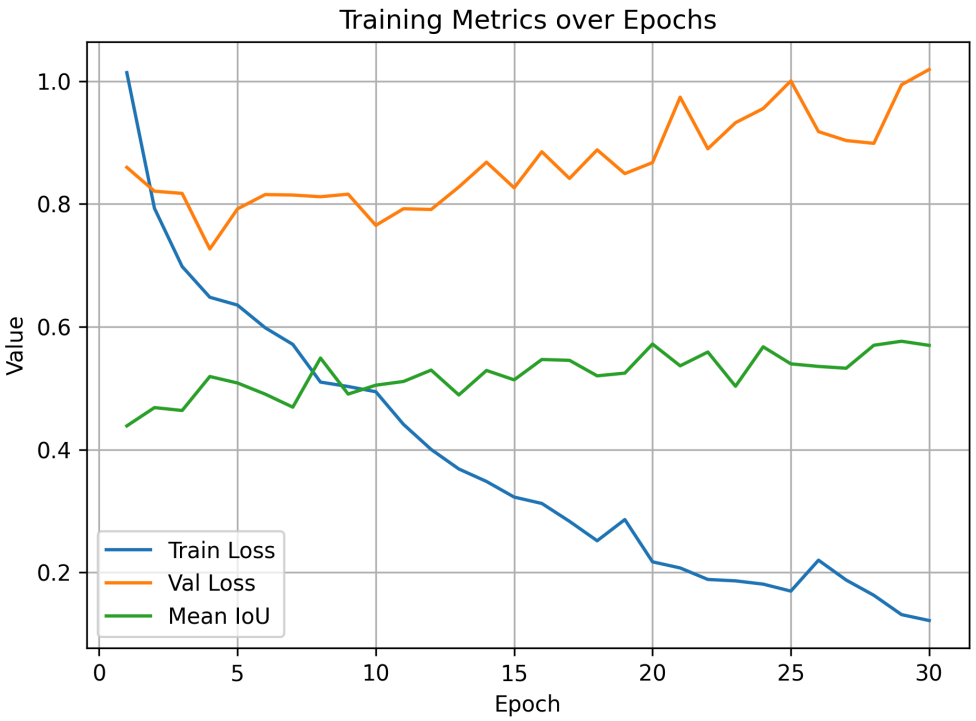


- **Massima memoria RAM occupata durante il training:** 1.28GB
- **Log di training**: `risultati_training\deeplabv3plus\resnet101\dlprn1010000\training_log_dlrn1010000.json`
- **Note / Bug Fix / Annotazioni**:

✳ Esperimento: `deeplabv3plus_resnet101 - dlprn1010001`

- **Architettura**: `deeplabv3plus_resnet101`
- **Backbone Weights**: best_deeplabv3plus_resnet101_cityscapes_os16.pth
- **Input size**: 256x256
- **Batch size**: 2

- **Augmentation**: off
- **Scheduler**: off
- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy+ class weights euristici [1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8].
- **Main classifier Kernel size:** 1
- **Valutazione risultati**:
    - Migliore mean IoU ottenuta all'epoca 29:
        - 📊 Train Loss: 0.1314 | Val Loss: 0.9941 | Mean IoU: 0.5765
        - ▸ IoU classe 0: 0.5415
        - ▸ IoU classe 1: 0.5790
        - ▸ IoU classe 2: 0.5913
        - ▸ IoU classe 3: 0.5590
        - ▸ IoU classe 4: 0.4902
        - ▸ IoU classe 5: 0.5384
        - ▸ IoU classe 6: 0.1770
        - ▸ IoU classe 7: 0.8164
        - ▸ IoU classe 8: 0.8604
    - Migliore Validation Loss ottenuta all'epoca 4:
        - 📊 Train Loss: 0.6481 | Val Loss: 0.7267 | Mean IoU: 0.5191
        - ▸ IoU classe 0: 0.5164
        - ▸ IoU classe 1: 0.5790
        - ▸ IoU classe 2: 0.5396
        - ▸ IoU classe 3: 0.5323
        - ▸ IoU classe 4: 0.3134
        - ▸ IoU classe 5: 0.3297
        - ▸ IoU classe 6: 0.1812
        - ▸ IoU classe 7: 0.7911
        - ▸ IoU classe 8: 0.8862
- **Rappresentazione grafica del training:**

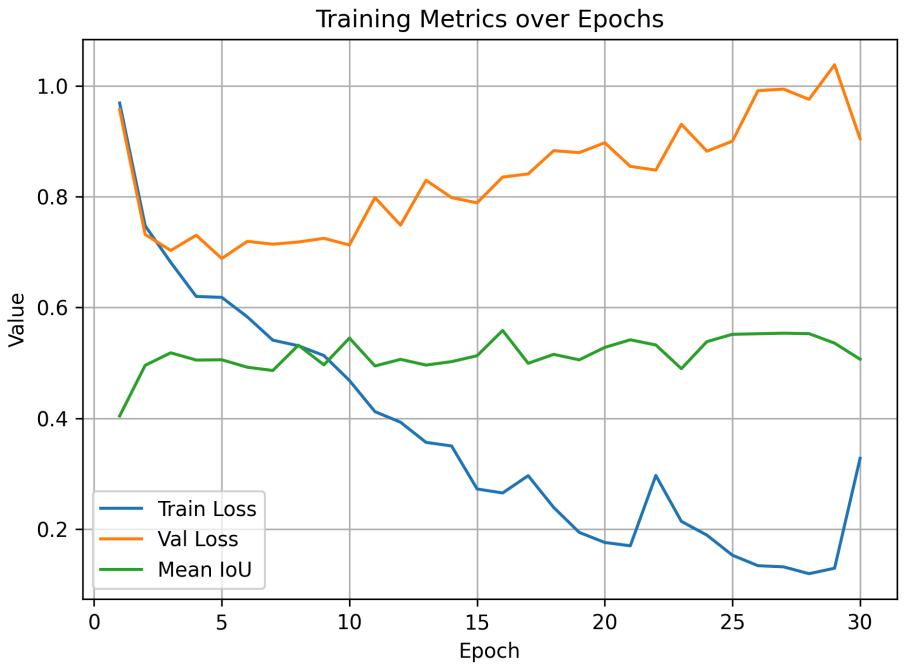


- **Massima memoria RAM occupata durante il training:** 1.28GB
- **Log di training**: `risultati_training\deeplabv3plus\resnet101\dlprn1010001\training_log_dlrn1010001.json`
- **Note / Bug Fix / Annotazioni**:
    - Bisogna mirare a occupare più memoria GPU, tuttavia raddoppiare la batch size ci porta a 5+GB. Si propone di aumentare la dimensione delle immagini a 512x512

✳ Esperimento: `deeplabv3plus_resnet101 - dlprn1010002`

- **Architettura**: `deeplabv3plus_resnet101`
- **Backbone Weights**: best_deeplabv3plus_resnet101_cityscapes_os16.pth
- **Input size**: 512x512
- **Batch size**: 2
- **Augmentation**: off
- **Scheduler**: off

- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy+ class weights euristici [1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8].
- **Main classifier Kernel size:** 1
- **Valutazione risultati**:
    - Migliore Validation Loss ottenuta all'epoca 5:
        - 📊 Train Loss: 0.6183 | Val Loss: 0.6884 | Mean IoU: 0.5057
        - ▸ IoU classe 0: 0.5540
        - ▸ IoU classe 1: 0.5372
        - ▸ IoU classe 2: 0.5092
        - ▸ IoU classe 3: 0.5227
        - ▸ IoU classe 4: 0.1871
        - ▸ IoU classe 5: 0.4341
        - ▸ IoU classe 6: 0.1759
        - ▸ IoU classe 7: 0.7924
        - ▸ IoU classe 8: 0.8871
    - Migliore mean IoU ottenuta all'epoca 16:
        - 📊 Train Loss: 0.2655 | Val Loss: 0.8353 | Mean IoU: 0.5586
        - ▸ IoU classe 0: 0.5718
        - ▸ IoU classe 1: 0.5898
        - ▸ IoU classe 2: 0.5835
        - ▸ IoU classe 3: 0.5237
        - ▸ IoU classe 4: 0.3945
        - ▸ IoU classe 5: 0.4986
        - ▸ IoU classe 6: 0.1603
        - ▸ IoU classe 7: 0.8198
        - ▸ IoU classe 8: 0.8990
- **Rappresentazione grafica del training:**

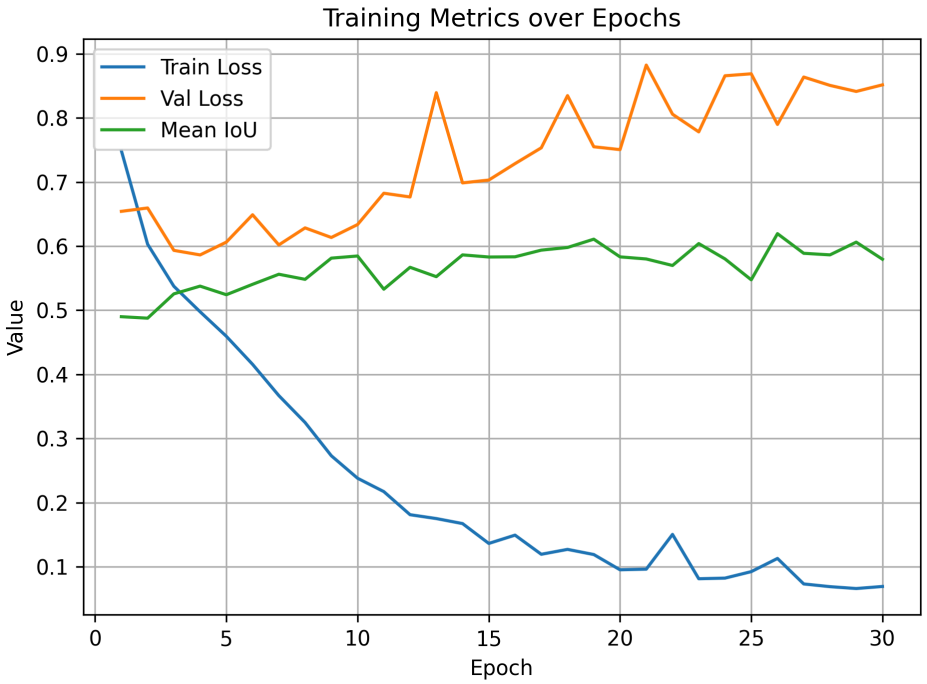


- **Massima memoria RAM occupata durante il training:** 2.88GB
- **Log di training**: `risultati_training\deeplabv3plus\resnet101\dlprn1010002\training_log_dlrn1010002.json`
- **Note / Bug Fix / Annotazioni**:

✳ Esperimento: `deeplabv3plus_resnet101 - dlprn1010003`

- **Architettura**: `deeplabv3plus_resnet101`
- **Backbone Weights**: best_deeplabv3plus_resnet101_cityscapes_os16.pth
- **Input size**: 512x512
- **Batch size**: 2
- **Augmentation**: on
- **Scheduler**: off
- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: off

- **Loss Function / Weighting**: cross-entropy.
- **Main classifier Kernel size:** 1
- **Valutazione risultati**:
    - Migliore Validation Loss ottenuta all'epoca 4:
        - 📊 Train Loss: 0.4977 | Val Loss: 0.5863 | Mean IoU: 0.5376
        - ▸ IoU classe 0: 0.4975
        - ▸ IoU classe 1: 0.6046
        - ▸ IoU classe 2: 0.5578
        - ▸ IoU classe 3: 0.5504
        - ▸ IoU classe 4: 0.0461
        - ▸ IoU classe 5: 0.5470
        - ▸ IoU classe 6: 0.2577
        - ▸ IoU classe 7: 0.8245
        - ▸ IoU classe 8: 0.9130
    - Migliore mean IoU ottenuta all'epoca 26:
        - 📊 Train Loss: 0.1132 | Val Loss: 0.7897 | Mean IoU: 0.6194
        - ▸ IoU classe 0: 0.5176
        - ▸ IoU classe 1: 0.6358
        - ▸ IoU classe 2: 0.6137
        - ▸ IoU classe 3: 0.5713
        - ▸ IoU classe 4: 0.5040
        - ▸ IoU classe 5: 0.6013
        - ▸ IoU classe 6: 0.2633
        - ▸ IoU classe 7: 0.8464
        - ▸ IoU classe 8: 0.9194
- **Rappresentazione grafica del training:**

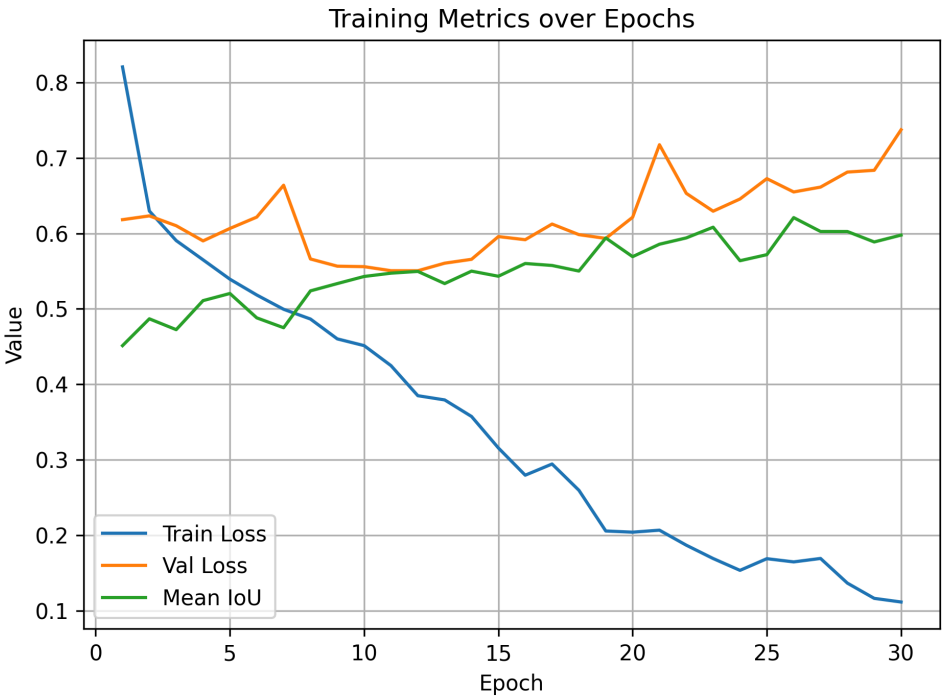


- **Massima memoria RAM occupata durante il training:** 2.42GB
- **Log di training**: `risultati_training\deeplabv3plus\resnet101\dlprn1010003\training_log_dlrn1010003.json`
- **Note / Bug Fix / Annotazioni**:

✳ Esperimento: `deeplabv3plus_resnet101 - dlprn1010004`

- **Architettura**: `deeplabv3plus_resnet101`
- **Backbone Weights**: best_deeplabv3plus_resnet101_cityscapes_os16.pth
- **Input size**: 512x512
- **Batch size**: 2
- **Augmentation**: off
- **Scheduler**: off
- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none

- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy.
- **Main classifier Kernel size:** 1
- **Valutazione risultati**:
    - Migliore Validation Loss ottenuta all’epoca 12:
        - 📊 Train Loss: 0.3850 | Val Loss: 0.5505 | Mean IoU: 0.5496
        - ▸ IoU classe 0: 0.5188
        - ▸ IoU classe 1: 0.6292
        - ▸ IoU classe 2: 0.5853
        - ▸ IoU classe 3: 0.5909
        - ▸ IoU classe 4: 0.1127
        - ▸ IoU classe 5: 0.5059
        - ▸ IoU classe 6: 0.2224
        - ▸ IoU classe 7: 0.8431
        - ▸ IoU classe 8: 0.9075
    - Migliore mean IoU ottenuta all’epoca 26:
        - 📊 Train Loss: 0.1648 | Val Loss: 0.6548 | Mean IoU: 0.6209
        - ▸ IoU classe 0: 0.5104
        - ▸ IoU classe 1: 0.6407
        - ▸ IoU classe 2: 0.6047
        - ▸ IoU classe 3: 0.6071
        - ▸ IoU classe 4: 0.5819
        - ▸ IoU classe 5: 0.4862
        - ▸ IoU classe 6: 0.2926
        - ▸ IoU classe 7: 0.8411
        - ▸ IoU classe 8: 0.9124
- **Rappresentazione grafica del training:**



- **Massima memoria RAM occupata durante il training:** 2.42GB
- **Log di training**: `risultati_training\deeplabv3plus\resnet101\dlprn1010004\training_log_dlrn1010004.json`
- **Note / Bug Fix / Annotazioni**:
  - Questo training (img size 512x512) mostra risultati migliori rispetto al corrispettivo con Augmentation e al corrispettivo con pesi euristici per la Loss.

✳ Esperimento: `deeplabv3plus_resnet101 - dlprn1010005`

- **Architettura**: `deeplabv3plus_resnet101`
- **Backbone Weights**: best_deeplabv3plus_resnet101_cityscapes_os16.pth
- **Input size**: 512x512
- **Batch size**: 2
- **Splitting:** on

```
import numpy as np
from PIL import Image
import os
from iterstrat.ml_stratifiers import MultilabelStratifiedShuffleSplit
import json

```

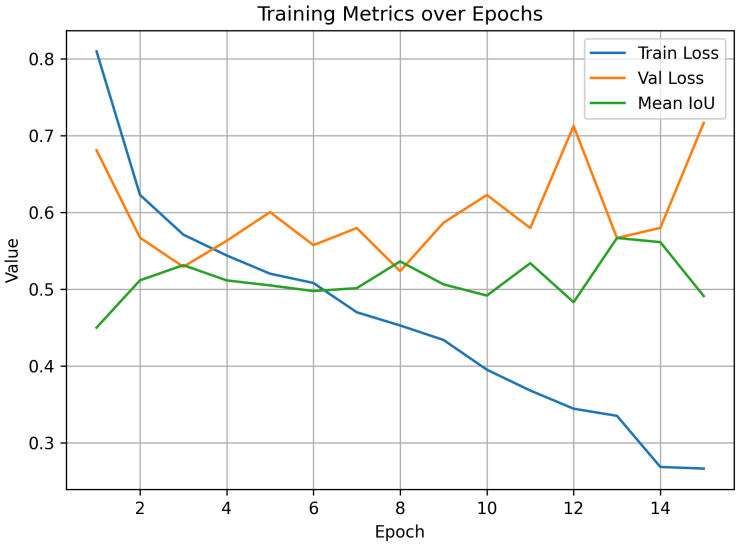
```
# sample male etichettati
excluded_dirs = {
    "0007", "0009", "0090", "0095", "0101", "0104",
"0105",
    "0162", "0163", "0284", "0305", "0306", "0307",
"0308", "0309", "0310", "0311",
    "0351", "0372", "0373", "0376",
    "0498", "0499", "0500", "0501", "0526", "0527",
"0530", "0531", "0542",
    "0564", "0585", "0586", "0587", "0588", "0589",
"0590",
    #seguono nuovo immagini da escludere
    "0000", "0001", "0052"
}

# Elenco di cartelle numeriche valide
all_dirs = sorted([
    d for d in os.listdir(DATASET_DIR)
    if os.path.isdir(os.path.join(DATASET_DIR, d)) and
d.isdigit() and d not in excluded_dirs
])

# ------------------------------
#  COSTRUZIONE DELLA MATRICE MULTILABEL
# ------------------------------
def build_presence_matrix(dir_list, n_classes=9):
    """Return np.array (n_samples, n_classes) with
booleans indicating class presence"""
    presence = np.zeros((len(dir_list), n_classes),
dtype=int)

    for idx, d in enumerate(dir_list):
        label_path = os.path.join(DATASET_DIR, d,
"labels.png")
        lbl = np.array(Image.open(label_path))
        uniq = np.unique(lbl)
        presence[idx, uniq] = 1          # segna le
classi presenti
    return presence

Y = build_presence_matrix(all_dirs, NUM_CLASSES)

# ------------------------------
#  MULTILABEL STRATIFIED SPLIT
# ------------------------------
```

```
msss = MultilabelStratifiedShuffleSplit(n_splits=1,
test_size=0.20, random_state=42)
train_idx, val_idx =
next(msss.split(np.zeros(len(all_dirs)), Y))

train_ids = [all_dirs[i] for i in train_idx]
val_ids   = [all_dirs[i] for i in val_idx]

print("Campioni:", len(train_ids), "train |",
len(val_ids), "val")
```

- **Augmentation**: off
- **Scheduler**: off
- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy.
- **Main classifier Kernel size:** 1
- **Valutazione risultati**:
    - Migliore Validation Loss ottenuta all'epoca 8:
        - 📊 Train Loss: 0.4527 | Val Loss: 0.5234 | Mean IoU: 0.5361
        - ▸ IoU classe 0: 0.5547
        - ▸ IoU classe 1: 0.6503
        - ▸ IoU classe 2: 0.5166
        - ▸ IoU classe 3: 0.6187
        - ▸ IoU classe 4: 0.0004
        - ▸ IoU classe 5: 0.4538
        - ▸ IoU classe 6: 0.3110
        - ▸ IoU classe 7: 0.8308
        - ▸ IoU classe 8: 0.9075
    - Migliore mean IoU ottenuta all'epoca 13:
        - 📊 Train Loss: 0.3349 | Val Loss: 0.5666 | Mean IoU: 0.5666
        - ▸ IoU classe 0: 0.5491
        - ▸ IoU classe 1: 0.6573
        - ▸ IoU classe 2: 0.5543
        - ▸ IoU classe 3: 0.5907
        - ▸ IoU classe 4: 0.2466
        - ▸ IoU classe 5: 0.4339

        - ▸ IoU classe 6: 0.3033
        - ▸ IoU classe 7: 0.8390
        - ▸ IoU classe 8: 0.9078

- **Rappresentazione grafica del training:**



- **Massima memoria RAM occupata durante il training:** 2.42GB

- **Log di training**: `risultati_training\deeplabv3plus\resnet101\dlprn1010005\training_log_dlrn1010005.json`

- **Note / Bug Fix / Annotazioni**:
    - Questo training (aggiunta di splitting proporzionale alle classi tra training e validation) mostra risultati migliori rispetto alla sua versione originale (1010004).

✳ Esperimento: `deeplabv3plus_resnet101 - dlprn1010006`

- **Architettura**: `deeplabv3plus_resnet101`

- **Backbone Weights**: best_deeplabv3plus_resnet101_cityscapes_os16.pth

- **Input size**: 512x512

- **Batch size**: 2

- **Splitting:** on

```
import numpy as np
from PIL import Image
import os
from iterstrat.ml_stratifiers import
MultilabelStratifiedShuffleSplit
```

```
import json

# sample male etichettati
excluded_dirs = {
    "0007", "0009", "0090", "0095", "0101", "0104",
"0105",
    "0162", "0163", "0284", "0305", "0306", "0307",
"0308", "0309", "0310", "0311",
    "0351", "0372", "0373", "0376",
    "0498", "0499", "0500", "0501", "0526", "0527",
"0530", "0531", "0542",
    "0564", "0585", "0586", "0587", "0588", "0589",
"0590",
    #seguono nuovo immagini da escludere
    "0000", "0001", "0052"
}

# Elenco di cartelle numeriche valide
all_dirs = sorted([
    d for d in os.listdir(DATASET_DIR)
    if os.path.isdir(os.path.join(DATASET_DIR, d)) and
d.isdigit() and d not in excluded_dirs
])

# ------------------------------
#  COSTRUZIONE DELLA MATRICE MULTILABEL
# ------------------------------
def build_presence_matrix(dir_list, n_classes=9):
    """Return np.array (n_samples, n_classes) with
booleans indicating class presence"""
    presence = np.zeros((len(dir_list), n_classes),
dtype=int)

    for idx, d in enumerate(dir_list):
        label_path = os.path.join(DATASET_DIR, d,
"labels.png")
        lbl = np.array(Image.open(label_path))
        uniq = np.unique(lbl)
        presence[idx, uniq] = 1          # segna le
classi presenti
    return presence

Y = build_presence_matrix(all_dirs, NUM_CLASSES)

# ------------------------------
```

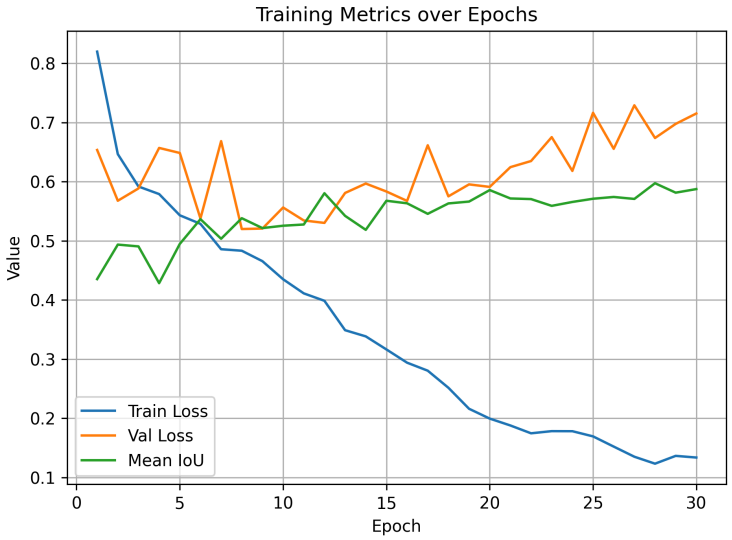
```
#  MULTILABEL STRATIFIED SPLIT
# ----------------------------
msss = MultilabelStratifiedShuffleSplit(n_splits=1,
test_size=0.20, random_state=42)
train_idx, val_idx =
next(msss.split(np.zeros(len(all_dirs)), Y))

train_ids = [all_dirs[i] for i in train_idx]
val_ids   = [all_dirs[i] for i in val_idx]

print("Campioni:", len(train_ids), "train |",
len(val_ids), "val")
```

- **Augmentation**: off
- **Scheduler**: off
- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy.
- **Main classifier Kernel size:** 1
- **Kernel size:**

```
class DeepLabHeadV3Plus(nn.Module):
    def __init__(self, in_channels, low_level_channels,
num_classes, aspp_dilate=[4, 8, 12]): #aspp_dilate=[12, 24,
36]
        super(DeepLabHeadV3Plus, self).__init__()
        self.project = nn.Sequential(
            nn.Conv2d(low_level_channels, 48, 1, bias=False),
            nn.BatchNorm2d(48),
            nn.ReLU(inplace=True),
        )

        self.aspp = ASPP(in_channels, aspp_dilate)

        self.classifier = nn.Sequential(
            nn.Conv2d(304, 256, 5, padding=2, bias=False),
#kernel size era 3, padding 1
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, num_classes, 1)
```

```
        )
        self._init_weight()
```

- **Valutazione risultati**:
    - Migliore Validation Loss ottenuta all'epoca 8:
        - 📊 Train Loss: 0.4832 | Val Loss: 0.5199 | Mean IoU: 0.5382
        - ▸ IoU classe 0: 0.5486
        - ▸ IoU classe 1: 0.6654
        - ▸ IoU classe 2: 0.5843
        - ▸ IoU classe 3: 0.5862
        - ▸ IoU classe 4: 0.1720
        - ▸ IoU classe 5: 0.3127
        - ▸ IoU classe 6: 0.2387
        - ▸ IoU classe 7: 0.8411
        - ▸ IoU classe 8: 0.9047
    - Migliore mean IoU ottenuta all'epoca 28:
        - 📊 Train Loss: 0.1231 | Val Loss: 0.6738 | Mean IoU: 0.5973
        - ▸ IoU classe 0: 0.4991
        - ▸ IoU classe 1: 0.6931
        - ▸ IoU classe 2: 0.5652
        - ▸ IoU classe 3: 0.6404
        - ▸ IoU classe 4: 0.3634
        - ▸ IoU classe 5: 0.4661
        - ▸ IoU classe 6: 0.2992
        - ▸ IoU classe 7: 0.8389
        - ▸ IoU classe 8: 0.9121
- **Rappresentazione grafica del training:**



- **Massima memoria RAM occupata durante il training:** 2.45GB

- **Log di training**: `risultati_training\deeplabv3plus\resnet101\dlprn1010006\training_log_dlrn1010006.json`

- **Note / Bug Fix / Annotazioni**:
  - Questo training aggiunge un utilizzo custom del kernel e di ASPP ma peggiora le prestazioni.

✳ Esperimento: `deeplabv3plus_resnet101 - dlprn1010007`

- **Architettura**: `deeplabv3plus_resnet101`

- **Backbone Weights**: best_deeplabv3plus_resnet101_cityscapes_os16.pth

- **Input size**: 512x512

- **Batch size**: 2

- **Splitting:** on

```
import numpy as np
from PIL import Image
import os
from iterstrat.ml_stratifiers import MultilabelStratifiedShuffleSplit
import json

# sample male etichettati
excluded_dirs = {
    "0007", "0009", "0090", "0095", "0101", "0104", "0105",
    "0162", "0163", "0284", "0305", "0306", "0307", "0308", "0309", "0310", "0311",
    "0351", "0372", "0373", "0376",
    "0498", "0499", "0500", "0501", "0526", "0527", "0530", "0531", "0542",
    "0564", "0585", "0586", "0587", "0588", "0589", "0590",
    #seguono nuovo immagini da escludere
    "0000", "0001", "0052"
}

# Elenco di cartelle numeriche valide
all_dirs = sorted([
    d for d in os.listdir(DATASET_DIR)
```

```
    if os.path.isdir(os.path.join(DATASET_DIR, d)) and
d.isdigit() and d not in excluded_dirs
])

# ------------------------------
#  COSTRUZIONE DELLA MATRICE MULTILABEL
# ------------------------------
def build_presence_matrix(dir_list, n_classes=9):
    """Return np.array (n_samples, n_classes) with
booleans indicating class presence"""
    presence = np.zeros((len(dir_list), n_classes),
dtype=int)

    for idx, d in enumerate(dir_list):
        label_path = os.path.join(DATASET_DIR, d,
"labels.png")
        lbl = np.array(Image.open(label_path))
        uniq = np.unique(lbl)
        presence[idx, uniq] = 1          # segna le
classi presenti
    return presence

Y = build_presence_matrix(all_dirs, NUM_CLASSES)

# ------------------------------
#  MULTILABEL STRATIFIED SPLIT
# ------------------------------
msss = MultilabelStratifiedShuffleSplit(n_splits=1,
test_size=0.20, random_state=42)
train_idx, val_idx =
next(msss.split(np.zeros(len(all_dirs)), Y))

train_ids = [all_dirs[i] for i in train_idx]
val_ids   = [all_dirs[i] for i in val_idx]

print("Campioni:", len(train_ids), "train |",
len(val_ids), "val")
```

- **Augmentation**: off
- **Scheduler**: off
- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none

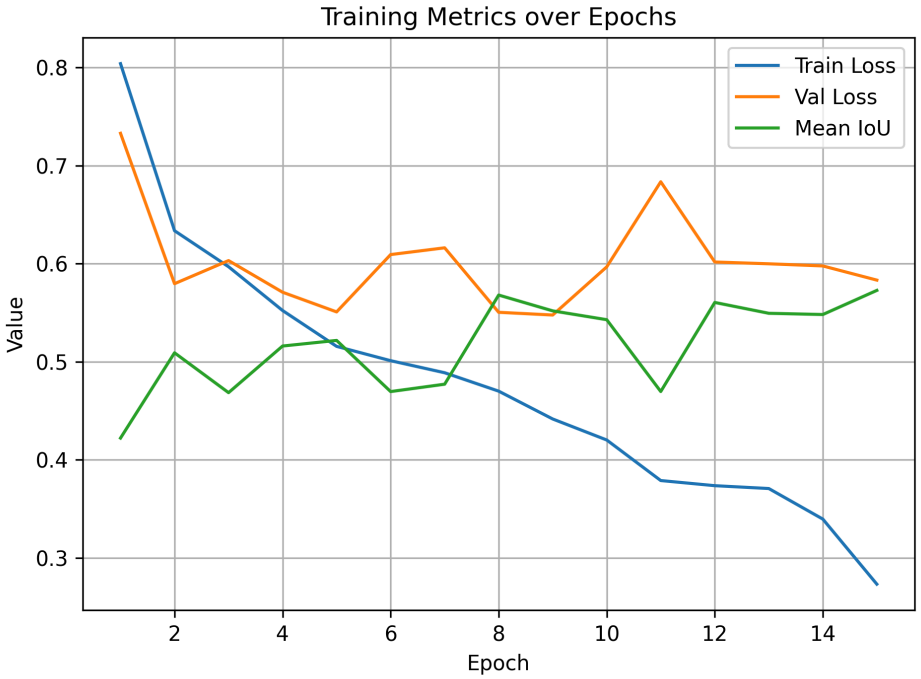
- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy.
- **Main classifier Kernel size:** 1
- **Kernel size:**

```
def __init__(self, in_channels, low_level_channels, num_classes, aspp_dilate=[3, 7, 3]): #aspp_dilate=[12, 24, 36]
        super(DeepLabHeadV3Plus, self).__init__()
        self.project = nn.Sequential(
            nn.Conv2d(low_level_channels, 48, 1, bias=False),
            nn.BatchNorm2d(48),
            nn.ReLU(inplace=True),
        )

        self.aspp = ASPP(in_channels, aspp_dilate)

        self.classifier = nn.Sequential(
            nn.Conv2d(304, 256, 5, padding=2, bias=False),
#kernel size era 3, padding 1
            #nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.Dropout(0.1),
            nn.Conv2d(256, num_classes, 1)
        )
        self._init_weight()
```

- **Valutazione risultati**:
    - Migliore Validation Loss ottenuta all'epoca 9:
        - 📊 Train Loss: 0.4417 | Val Loss: 0.5476 | Mean IoU: 0.5518
        - ▸ IoU classe 0: 0.5367
        - ▸ IoU classe 1: 0.6628
        - ▸ IoU classe 2: 0.5508
        - ▸ IoU classe 3: 0.6003
        - ▸ IoU classe 4: 0.2714
        - ▸ IoU classe 5: 0.3063
        - ▸ IoU classe 6: 0.2910
        - ▸ IoU classe 7: 0.8303
        - ▸ IoU classe 8: 0.9017

- Migliore mean IoU ottenuta all'epoca 15:
  - 📊 Train Loss: 0.2733 | Val Loss: 0.5832 | Mean IoU: 0.5727
  - ▸ IoU classe 0: 0.5586
  - ▸ IoU classe 1: 0.7004
  - ▸ IoU classe 2: 0.5607
  - ▸ IoU classe 3: 0.6475
  - ▸ IoU classe 4: 0.3450
  - ▸ IoU classe 5: 0.2963
  - ▸ IoU classe 6: 0.2879
  - ▸ IoU classe 7: 0.8315
  - ▸ IoU classe 8: 0.9122

- **Rappresentazione grafica del training:**



- **Massima memoria RAM occupata durante il training:** 2.45GB

- **Log di training**: `risultati_training\deeplabv3plus\resnet101\dlprn1010007\training_log_dlrn1010007.json`

- **Note / Bug Fix / Annotazioni**:

✳ Esperimento: `deeplabv3plus_resnet101 - dlprn1010008`

- **Architettura**: `deeplabv3plus_resnet101`

- **Backbone Weights**: best_deeplabv3plus_resnet101_cityscapes_os16.pth

- **Input size**: 512x512

- **Batch size**: 2
- **Splitting:** on

```
import numpy as np
from PIL import Image
import os
from iterstrat.ml_stratifiers import MultilabelStratifiedShuffleSplit
import json

# sample male etichettati
excluded_dirs = {
    "0007", "0009", "0090", "0095", "0101", "0104", "0105",
    "0162", "0163", "0284", "0305", "0306", "0307", "0308", "0309", "0310", "0311",
    "0351", "0372", "0373", "0376",
    "0498", "0499", "0500", "0501", "0526", "0527", "0530", "0531", "0542",
    "0564", "0585", "0586", "0587", "0588", "0589", "0590",
    #seguono nuovo immagini da escludere
    "0000", "0001", "0052"
}

# Elenco di cartelle numeriche valide
all_dirs = sorted([
    d for d in os.listdir(DATASET_DIR)
    if os.path.isdir(os.path.join(DATASET_DIR, d)) and d.isdigit() and d not in excluded_dirs
])

# ------------------------------
#  COSTRUZIONE DELLA MATRICE MULTILABEL
# ------------------------------
def build_presence_matrix(dir_list, n_classes=9):
    """Return np.array (n_samples, n_classes) with booleans indicating class presence"""
    presence = np.zeros((len(dir_list), n_classes), dtype=int)

    for idx, d in enumerate(dir_list):
        label_path = os.path.join(DATASET_DIR, d, "labels.png")
        lbl = np.array(Image.open(label_path))
```

```
        uniq = np.unique(lbl)
        presence[idx, uniq] = 1          # segna le classi presenti
    return presence

Y = build_presence_matrix(all_dirs, NUM_CLASSES)

# ------------------------------
#  MULTILABEL STRATIFIED SPLIT
# ------------------------------
msss = MultilabelStratifiedShuffleSplit(n_splits=1, test_size=0.20, random_state=42)
train_idx, val_idx = next(msss.split(np.zeros(len(all_dirs)), Y))

train_ids = [all_dirs[i] for i in train_idx]
val_ids   = [all_dirs[i] for i in val_idx]

print("Campioni:", len(train_ids), "train |", len(val_ids), "val")
```

- **Augmentation**: off
- **Scheduler**: off
- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy.
- **Main classifier Kernel size:** 1
- **Kernel size:**

```
def __init__(self, in_channels, low_level_channels, num_classes, aspp_dilate=[1, 3, 5]):

        super(DeepLabHeadV3Plus, self).__init__()
        self.project = nn.Sequential(
            nn.Conv2d(low_level_channels, 48, 1, bias=False),
            nn.BatchNorm2d(48),
            nn.ReLU(inplace=True),
        )
```

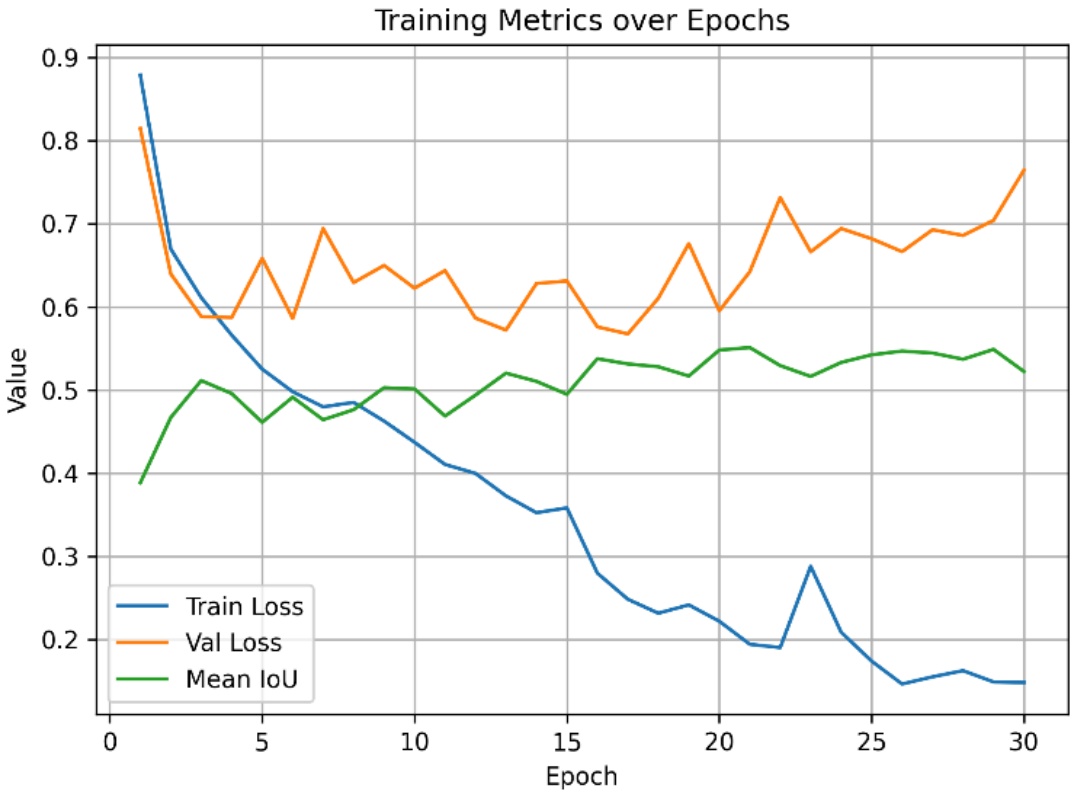
```
        self.aspp = ASPP(in_channels, aspp_dilate)

        self.classifier = nn.Sequential(
            nn.Conv2d(304, 256, 5, padding=2, bias=False),
#kernel size era 3, padding 1
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
           nn.Conv2d(256, 256, 5, padding=2, bias=False),
            nn.ReLU(inplace=True),
            nn.Dropout(0.1),
            nn.Conv2d(256, num_classes, 1)
        )
        self._init_weight()
```

- **Valutazione risultati**:

Migliore Validation Loss ottenuta all'epoca 17:
📊 Train Loss: 0.2485 | Val Loss: 0.5676 | Mean IoU: 0.5315
- IoU classe 0: 0.6081
- IoU classe 1: 0.6245
- IoU classe 2: 0.5524
- IoU classe 3: 0.6435
- IoU classe 4: 0.0000
- IoU classe 5: 0.4427
- IoU classe 6: 0.2647
- IoU classe 7: 0.8195
- IoU classe 8: 0.9046

Migliore mean IoU ottenuta all'epoca 21:
📊 Train Loss: 0.1944 | Val Loss: 0.6422 | Mean IoU: 0.5512
- IoU classe 0: 0.6414
- IoU classe 1: 0.6371
- IoU classe 2: 0.5762
- IoU classe 3: 0.6136
- IoU classe 4: 0.0000
- IoU classe 5: 0.4852
- IoU classe 6: 0.3657
- IoU classe 7: 0.8310
- IoU classe 8: 0.9011

- **Rappresentazione grafica del training:**



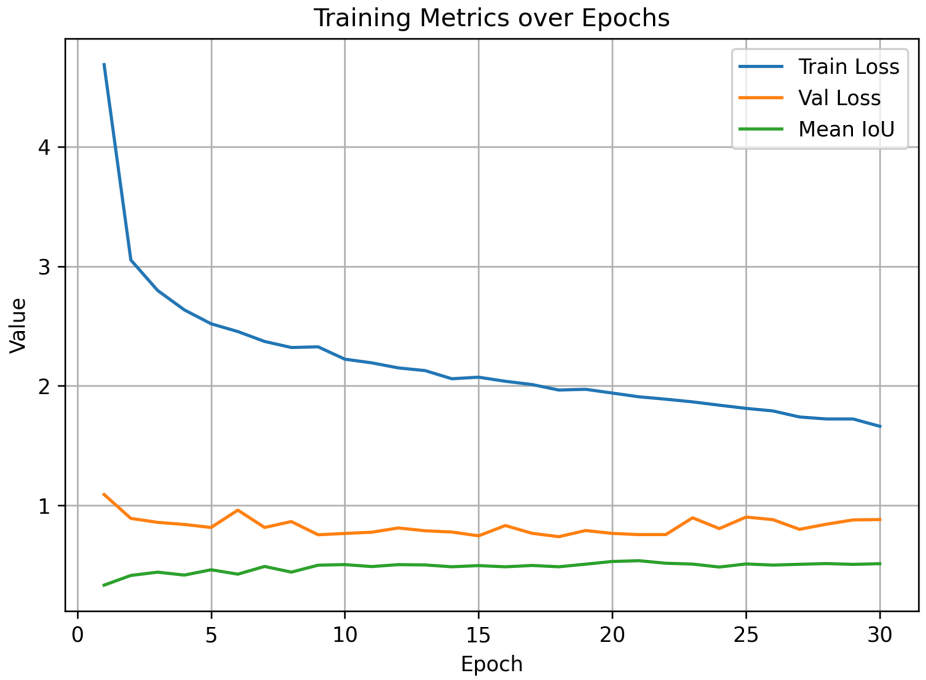
- **Massima memoria RAM occupata durante il training:** 2.48GB
- **Log di training**: `risultati_training\deeplabv3plus\resnet101\dlprn1010007\training_log_dlrn1010008.json`
- **Note / Bug Fix / Annotazioni**:

BiSeNetV2

✳ Esperimento: `BiSeNetV2 - bsnv20001`

- **Architettura**: `BiSeNetV2`
- **Backbone Weights**: model_final_rugd.pth
- **Input size**: 256x256
- **Batch size**: 2
- **Augmentation**: off
- **Scheduler**: off

- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy+ class weights euristici [1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8].
- **Main classifier Kernel size:** 1
- **Valutazione risultati**:
    - Migliore Validation Loss ottenuta all'epoca 18:
        - 📊 Train Loss: 1.9656 | Val Loss: 0.7396 | Mean IoU: 0.4882
        - ▸ IoU classe 0: 0.5773
        - ▸ IoU classe 1: 0.5773
        - ▸ IoU classe 2: 0.4750
        - ▸ IoU classe 3: 0.4569
        - ▸ IoU classe 4: 0.1991
        - ▸ IoU classe 5: 0.4224
        - ▸ IoU classe 6: 0.1655
        - ▸ IoU classe 7: 0.7791
        - ▸ IoU classe 8: 0.8919
    - Migliore mean IoU ottenuta all'epoca 21:
        - 📊 Train Loss: 1.9089 | Val Loss: 0.7570 | Mean IoU: 0.5388
        - ▸ IoU classe 0: 0.5848
        - ▸ IoU classe 1: 0.4562
        - ▸ IoU classe 2: 0.5351
        - ▸ IoU classe 3: 0.5078
        - ▸ IoU classe 4: 0.3807
        - ▸ IoU classe 5: 0.5554
        - ▸ IoU classe 6: 0.1715
        - ▸ IoU classe 7: 0.8067
        - ▸ IoU classe 8: 0.8966
- **Rappresentazione grafica del training:**



- **Massima memoria RAM occupata durante il training:** 1.96GB
- **Log di training**: `risultati_training\deeplabv3plus\resnet101\bsnv20001\training_log_bsnv20001.json`
- **Note / Bug Fix / Annotazioni**:
  - -

✳ Esperimento: `BiSeNetV2 - bsnv20002`

- **Architettura**: `BiSeNetV2`
- **Backbone Weights**: model_final_rellis3d.pth
- **Input size**: 256x256
- **Batch size**: 2
- **Augmentation**: off
- **Scheduler**: off
- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: off

- **Loss Function / Weighting**: cross-entropy+ class weights euristici [1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8].

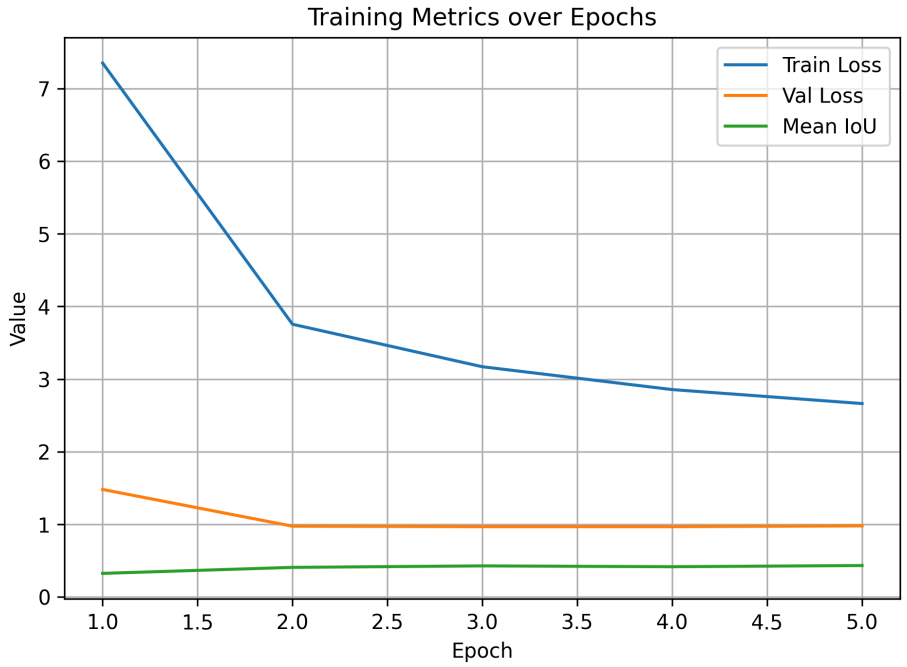
- **Main classifier Kernel size:** 1

- **Valutazione risultati**:

    - Migliore Validation Loss ottenuta all'epoca 4:
        - 📊 Train Loss: 2.8555 | Val Loss: 0.9681 | Mean IoU: 0.4166
        - ▸ IoU classe 0: 0.4735
        - ▸ IoU classe 1: 0.4346
        - ▸ IoU classe 2: 0.3779
        - ▸ IoU classe 3: 0.4268
        - ▸ IoU classe 4: 0.0000
        - ▸ IoU classe 5: 0.3164
        - ▸ IoU classe 6: 0.1450
        - ▸ IoU classe 7: 0.7682
        - ▸ IoU classe 8: 0.8637

    - Migliore mean IoU ottenuta all'epoca 5:
        - 📊 Train Loss: 2.6632 | Val Loss: 0.9786 | Mean IoU: 0.4331
        - ▸ IoU classe 0: 0.4779
        - ▸ IoU classe 1: 0.4879
        - ▸ IoU classe 2: 0.4225
        - ▸ IoU classe 3: 0.3771
        - ▸ IoU classe 4: 0.0000
        - ▸ IoU classe 5: 0.3911
        - ▸ IoU classe 6: 0.1437
        - ▸ IoU classe 7: 0.7777
        - ▸ IoU classe 8: 0.8644

- **Rappresentazione grafica del training:**



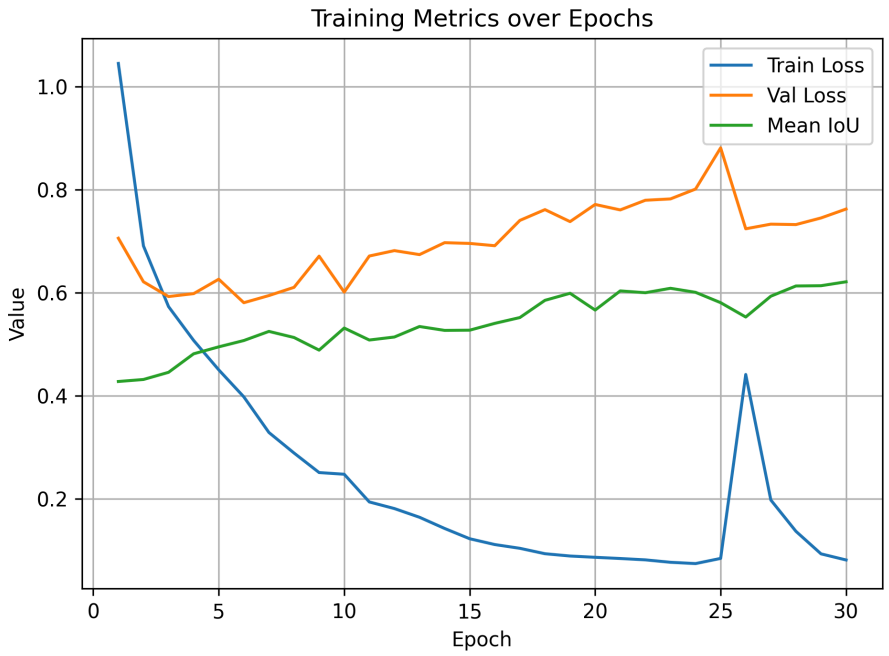
- **Massima memoria RAM occupata durante il training:** 1.96GB
- **Log di training**: `risultati_training\deeplabv3plus\resnet101\bsnv20002\training_log_bsnv20002.json`
- **Note / Bug Fix / Annotazioni**:
  - -

deeplabv3_resnet50

✳ Esperimento: `deeplabv3_resnet50 - dlrn500000`

- **Architettura**: `deeplabv3plus_resnet50`
- **Backbone Weights**: DeepLabV3_ResNet50_Weights.DEFAULT
- **Input size**: 256x256
- **Batch size**: 4
- **Augmentation**: off
- **Scheduler**: off
- **Optimizer:** Adama con lr=1e-4

- **Freezing**: none
- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy
- **Main classifier Kernel size:** 1
- **Valutazione risultati**:
    - Migliore Validation Loss ottenuta all'epoca 6:
        - 📊 Train Loss: 0.3976 | Val Loss: 0.5804 | Mean IoU: 0.5070
        - ▸ IoU classe 0: 0.4787
        - ▸ IoU classe 1: 0.5573
        - ▸ IoU classe 2: 0.6179
        - ▸ IoU classe 3: 0.5495
        - ▸ IoU classe 4: 0.0000
        - ▸ IoU classe 5: 0.4713
        - ▸ IoU classe 6: 0.1508
        - ▸ IoU classe 7: 0.8290
        - ▸ IoU classe 8: 0.8799
    - Migliore `mean IoU` ottenuta all'epoca 30:
        - 📊 Train Loss: 0.0809 | Val Loss: 0.7622 | Mean IoU: 0.6210
        - ▸ IoU classe 0: 0.5350
        - ▸ IoU classe 1: 0.6204
        - ▸ IoU classe 2: 0.6361
        - ▸ IoU classe 3: 0.5720
        - ▸ IoU classe 4: 0.6263
        - ▸ IoU classe 5: 0.6018
        - ▸ IoU classe 6: 0.1782
        - ▸ IoU classe 7: 0.8382
        - ▸ IoU classe 8: 0.8949
- **Rappresentazione grafica del training:**

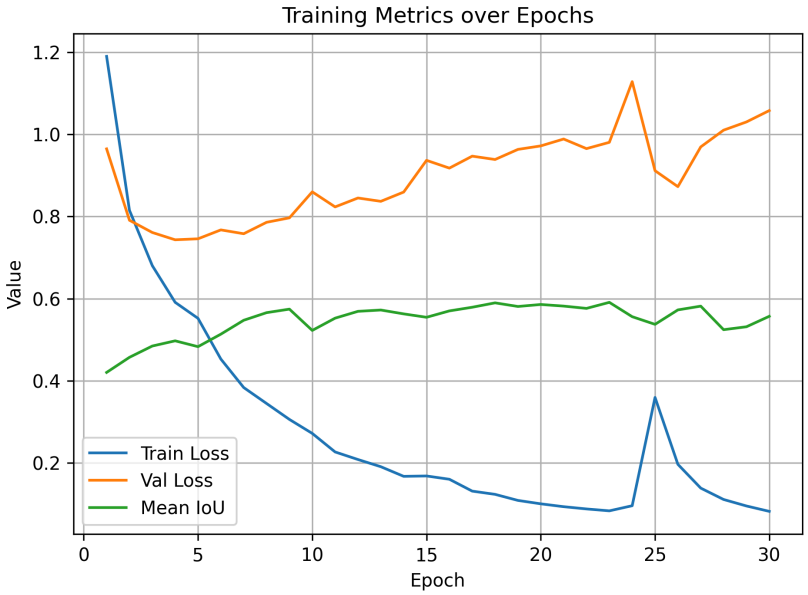


- **Massima memoria RAM occupata durante il training:** 1.54GB
- **Log di training**: `risultati_training\deeplabv3\resnet50\dlrn500000\training_log_dlrn500000.json`
- **Note / Bug Fix / Annotazioni**:
  - —

✳ Esperimento: `deeplabv3_resnet50 - dlrn500001`

- **Architettura**: `deeplabv3plus_resnet50`
- **Backbone Weights**: DeepLabV3_ResNet50_Weights.DEFAULT
- **Input size**: 256x256
- **Batch size**: 4
- **Augmentation**: off
- **Scheduler**: off
- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none

- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy + class weights euristici [1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8].
- **Main classifier Kernel size:** 1
- **Valutazione risultati**:
    - Migliore Validation Loss ottenuta all'epoca 4:
        - 📊 Train Loss: 0.5912 | Val Loss: 0.7434 | Mean IoU: 0.4975
        - ▸ IoU classe 0: 0.4978
        - ▸ IoU classe 1: 0.5695
        - ▸ IoU classe 2: 0.5240
        - ▸ IoU classe 3: 0.5543
        - ▸ IoU classe 4: 0.0000
        - ▸ IoU classe 5: 0.4847
        - ▸ IoU classe 6: 0.1588
        - ▸ IoU classe 7: 0.8133
        - ▸ IoU classe 8: 0.8751
    - Migliore `mean IoU` ottenuta all'epoca 23:
        - 📊 Train Loss: 0.0836 | Val Loss: 0.9807 | Mean IoU: 0.5911
        - ▸ IoU classe 0: 0.5239
        - ▸ IoU classe 1: 0.6022
        - ▸ IoU classe 2: 0.6362
        - ▸ IoU classe 3: 0.5781
        - ▸ IoU classe 4: 0.4076
        - ▸ IoU classe 5: 0.5827
        - ▸ IoU classe 6: 0.1911
        - ▸ IoU classe 7: 0.8366
        - ▸ IoU classe 8: 0.8943
- **Rappresentazione grafica del training:**



- **Massima memoria RAM occupata durante il training:** 1.54GB
- **Log di training**: `risultati_training\deeplabv3\resnet50\dlrn500001\training_log_dlrn500001.json`
- **Note / Bug Fix / Annotazioni**:
  - —

✳ Esperimento: `deeplabv3_resnet50 - dlrn500002`

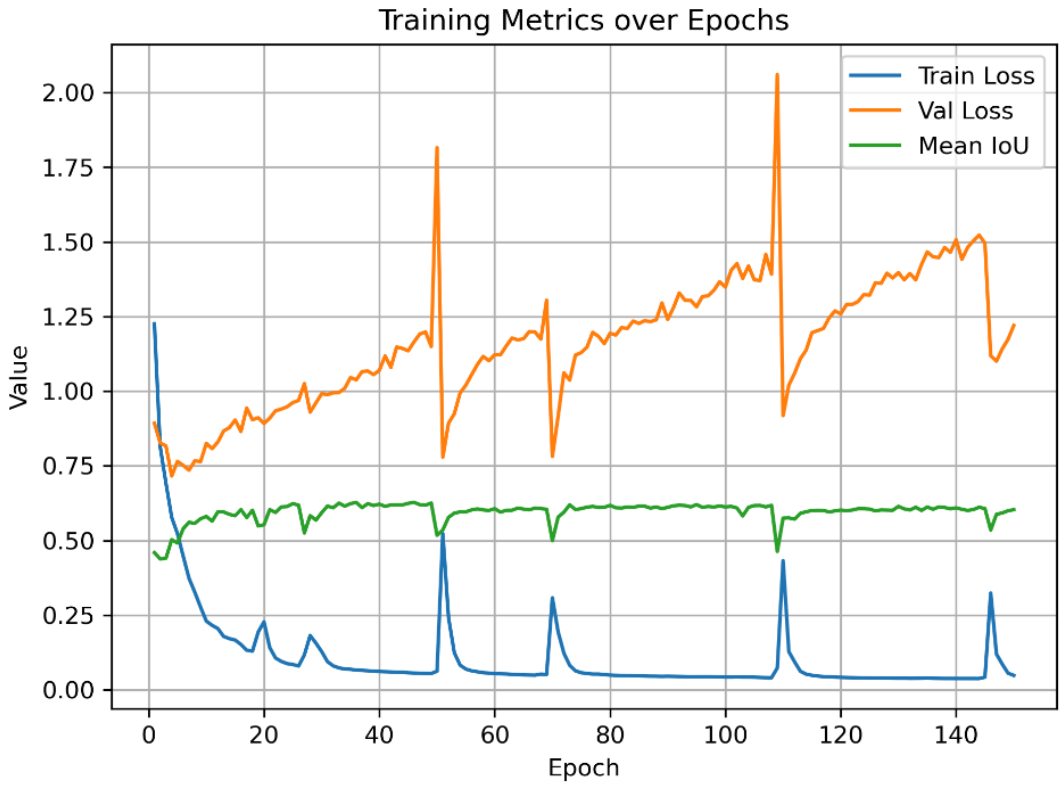
- **Architettura**: `deeplabv3plus_resnet50`
- **Backbone Weights**: DeepLabV3_ResNet50_Weights.DEFAULT
- **Input size**: 256x256
- **Batch size**: 4
- **Augmentation :** on

```
# Augmentazione per immagini con classi rare
rare_aug = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.4),
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])
```

```
o
o  # Augmentazione di base
o  base_aug = A.Compose([
o      A.Resize(256, 256),
o      A.Normalize(mean=(0.485, 0.456, 0.406),
o                  std=(0.229, 0.224, 0.225)),
o      ToTensorV2()
o  ])
o
o
o  train_loader = DataLoader(train_set,
   batch_size=BATCH_SIZE, shuffle=True, drop_last=True)
o  def __init__(self, root_dir, is_val = False,
   sample_ids=None, allow_augmentation = False):
o      self.samples = []
o      self.is_val = is_val
o      self.sample_ids = sample_ids
o      self.allow_augmentation = allow_augmentation
o      self._load_and_augment(root_dir)
o
o  def _load_and_augment(self, root_dir):
o      sample_dirs = []
o      for id in self.sample_ids:
o          sample_dirs.append(os.path.join(root_dir, id))
o
o      for dir_path in sample_dirs:
o          img_path = os.path.join(dir_path, 'rgb.jpg')
o          label_path = os.path.join(dir_path, 'labels.png')
o
o          label_img = Image.open(label_path)
o          is_rare =
   SegmentationDataset.contains_rare_classes(label_img,
   RARE_CLASSES)
o
o          if not self.allow_augmentation:
o              is_rare = False
o
o          # Sempre aggiungi l'originale
o          self.samples.append((img_path, label_path,
   is_rare))
o
o          # Duplicazione per oversampling (solo se contiene
   classi rare)
o          if is_rare and not self.is_val:
o              for _ in range(DUPLICATE_FACTOR - 1):
```

```
                self.samples.append((img_path,
label_path, is_rare))
```

- **Scheduler**: off
- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy + class weights euristici [1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8].
- **Main classifier Kernel size:** 1
- **Valutazione risultati**:
  - Migliore Validation Loss ottenuta all'epoca 4:
    - 📊 Train Loss: 0.5772 | Val Loss: 0.7160 | Mean IoU: 0.5025
      - ▸ IoU classe 0: 0.5101
      - ▸ IoU classe 1: 0.5496
      - ▸ IoU classe 2: 0.5533
      - ▸ IoU classe 3: 0.5210
      - ▸ IoU classe 4: 0.0
      - ▸ IoU classe 5: 0.5110
      - ▸ IoU classe 6: 0.1892
      - ▸ IoU classe 7: 0.8196
      - ▸ IoU classe 8: 0.8759
  - Migliore `mean IoU` ottenuta all'epoca 46:
    - 📊 Train Loss: 0.0560 | Val Loss: 1.1647 | Mean IoU: 0.6277
      - ▸ IoU classe 0: 0.5581
      - ▸ IoU classe 1: 0.6294
      - ▸ IoU classe 2: 0.6459
      - ▸ IoU classe 3: 0.5936
      - ▸ IoU classe 4: 0.5776
      - ▸ IoU classe 5: 0.6460
      - ▸ IoU classe 6: 0.1898
      - ▸ IoU classe 7: 0.8455
      - ▸ IoU classe 8: 0.8937
- **Rappresentazione grafica del training:**



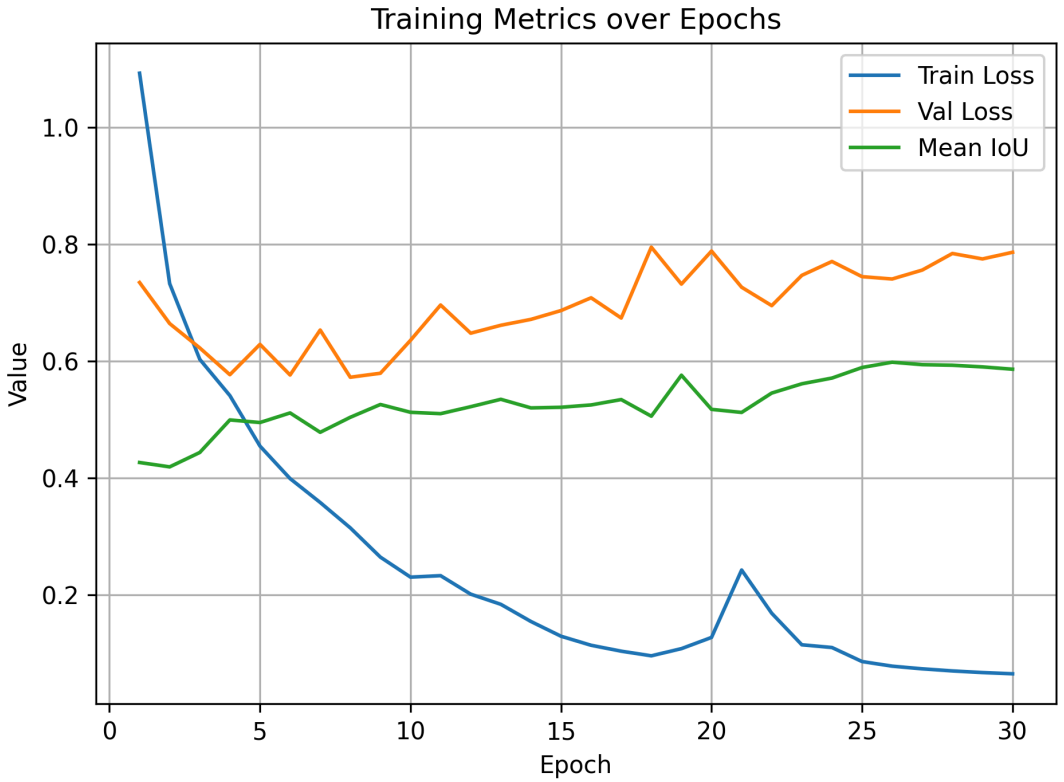
- **Massima memoria RAM occupata durante il training:** —GB
- **Log di training**: `risultati_training\deeplabv3\resnet50\dlrn500001\training_log_dlrn500002.json`
- **Note / Bug Fix / Annotazioni**:
    - grafico con picchi anomali ogni 30 epoche

deeplabv3_resnet101

✳ Esperimento: `deeplabv3_resnet101 - dlrn101000`

- **Architettura**: `deeplabv3plus_resnet101`
- **Backbone Weights**: DeepLabV3_ResNet101_Weights.DEFAULT
- **Input size**: 256x256
- **Batch size**: 4
- **Augmentation**: off

- **Scheduler**: off
- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy
- **Main classifier Kernel size:** 1
- **Valutazione risultati**:
    - Migliore Validation Loss ottenuta all'epoca 7:
        - 📊 Train Loss: 0.3993 | Val Loss: 0.5766 | Mean IoU: 0.5117
        - ▸ IoU classe 0: 0.5169
        - ▸ IoU classe 1: 0.5172
        - ▸ IoU classe 2: 0.6249
        - ▸ IoU classe 3: 0.5701
        - ▸ IoU classe 4: 0.0000
        - ▸ IoU classe 5: 0.5097
        - ▸ IoU classe 6: 0.1662
        - ▸ IoU classe 7: 0.8303
        - ▸ IoU classe 8: 0.8747
    - Migliore `mean IoU` ottenuta all'epoca 26:
        - 📊 Train Loss: 0.0782 | Val Loss: 0.7409 | Mean IoU: 0.5985
        - ▸ IoU classe 0: 0.5401
        - ▸ IoU classe 1: 0.6034
        - ▸ IoU classe 2: 0.6366
        - ▸ IoU classe 3: 0.5879
        - ▸ IoU classe 4: 0.3909
        - ▸ IoU classe 5: 0.6283
        - ▸ IoU classe 6: 0.2004
        - ▸ IoU classe 7: 0.8454
        - ▸ IoU classe 8: 0.8949
- **Rappresentazione grafica del training:**

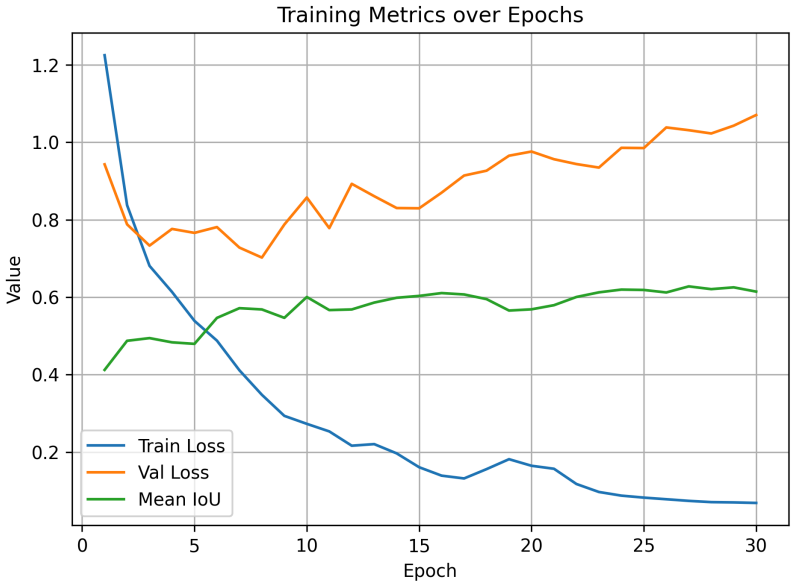


- **Massima memoria RAM occupata durante il training:** 2.62GB
- **Log di training**: `risultati_training\deeplabv3\resnet101\dlrn1010000\training_log_dlrn1010001.json`
- **Note / Bug Fix / Annotazioni**:
  - —

✳ Esperimento: `deeplabv3_resnet101 - dlrn101001`

- **Architettura**: `deeplabv3plus_resnet101`
- **Backbone Weights**: DeepLabV3_ResNet101_Weights.DEFAULT
- **Input size**: 256x256
- **Batch size**: 4
- **Augmentation**: off
- **Scheduler**: off
- **Optimizer:** Adama con lr=1e-4

- **Freezing**: none
- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy + class weights euristici [1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8].
- **Main classifier Kernel size:** 1
- **Valutazione risultati**:
    - Migliore Validation Loss ottenuta all'epoca 8:
        - 📊 Train Loss: 0.3480 | Val Loss: 0.7024 | Mean IoU: 0.5685
        - ▸ IoU classe 0: 0.5130
        - ▸ IoU classe 1: 0.6119
        - ▸ IoU classe 2: 0.5904
        - ▸ IoU classe 3: 0.5739
        - ▸ IoU classe 4: 0.3759
        - ▸ IoU classe 5: 0.5182
        - ▸ IoU classe 6: 0.1781
        - ▸ IoU classe 7: 0.8242
        - ▸ IoU classe 8: 0.8751
    - Migliore `mean IoU` ottenuta all'epoca 27:
        - 📊 Train Loss: 0.0740 | Val Loss: 1.0314 | Mean IoU: 0.6280
        - ▸ IoU classe 0: 0.5373
        - ▸ IoU classe 1: 0.6072
        - ▸ IoU classe 2: 0.6516
        - ▸ IoU classe 3: 0.5739
        - ▸ IoU classe 4: 0.6328
        - ▸ IoU classe 5: 0.6454
        - ▸ IoU classe 6: 0.1861
        - ▸ IoU classe 7: 0.8358
        - ▸ IoU classe 8: 0.8912
- **Rappresentazione grafica del training:**



- **Massima memoria RAM occupata durante il training:** 2.62GB
- **Log di training**: `risultati_training\deeplabv3\resnet101\dlrn1010001\training_log_dlrn1010001.json`
- **Note / Bug Fix / Annotazioni**:
  - —

✳ Esperimento: `deeplabv3_resnet101 - dlrn101002`

- **Architettura**: `deeplabv3plus_resnet101`
- **Backbone Weights**: DeepLabV3_ResNet101_Weights.DEFAULT
- **Input size**: 256x256
- **Batch size**: 4
- **Augmentation**: on

```
# Augmentazione per immagini con classi rare
rare_aug = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.4),
    #A.Rotate(limit=20, p=0.3), potenzialmente inutile
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
```

```
    ToTensorV2()
])

# Augmentazione di base
base_aug = A.Compose([
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])
```

```
def _load_and_augment(self, root_dir):
    sample_dirs = []
    for id in self.sample_ids:
        sample_dirs.append(os.path.join(root_dir, id))

    for dir_path in sample_dirs:
        img_path = os.path.join(dir_path, 'rgb.jpg')
        label_path = os.path.join(dir_path, 'labels.png')

        label_img = Image.open(label_path)
        is_rare =
SegmentationDataset.contains_rare_classes(label_img,
RARE_CLASSES)

        if not self.allow_augmentation:
            is_rare = False

        # Sempre aggiungi l'originale
        self.samples.append((img_path, label_path,
is_rare))

        # Duplicazione per oversampling (solo se contiene
classi rare)
        if is_rare and not self.is_val:
            for _ in range(DUPLICATE_FACTOR - 1):
                self.samples.append((img_path,
label_path, is_rare))
```

- **Scheduler**: off
- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none

- **Auxiliary Head**: off

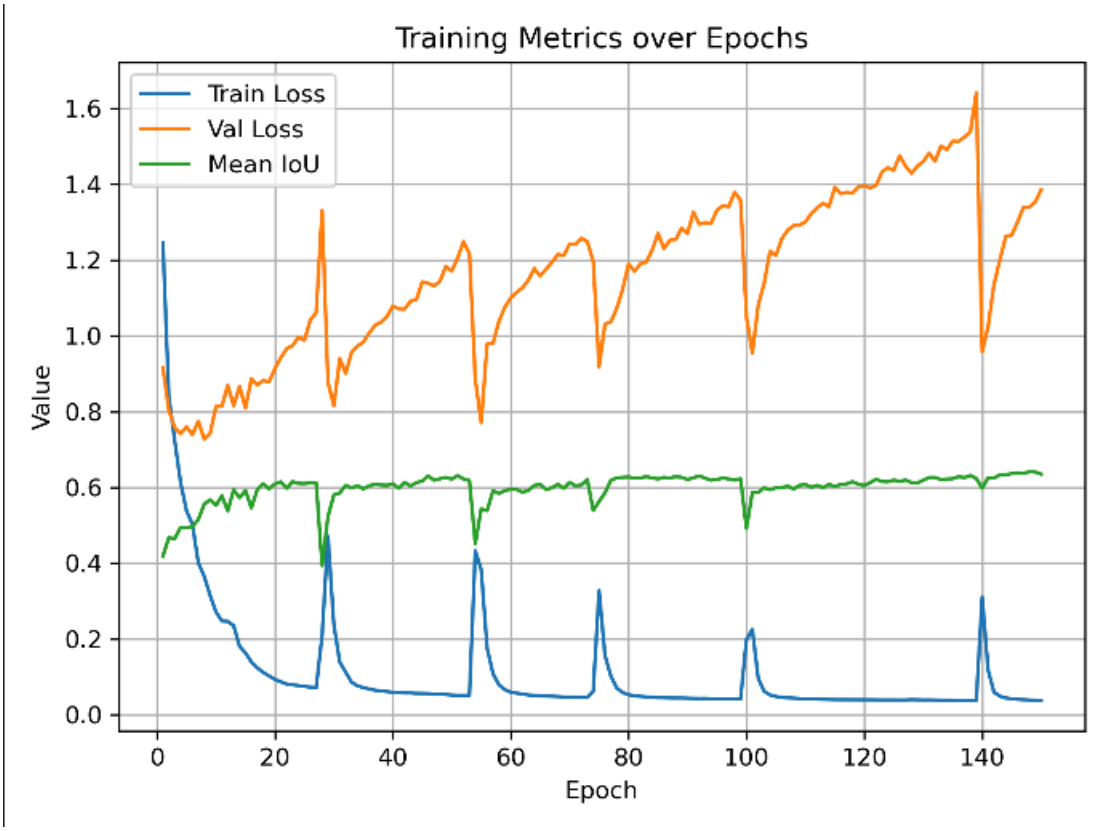
- **Loss Function / Weighting**: cross-entropy + class weights euristici [1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8].

- **Main classifier Kernel size:** 1

- **Valutazione risultati**:
    - Migliore Validation Loss ottenuta all'epoca 4:
        - Train Loss: 0.3602 | Val Loss: 0.7229 | Mean IoU: 0.5917
        - IoU classe 0: 0.4873
        - IoU classe 1: 0.5685
        - IoU classe 2: 0.5609
        - IoU classe 3: 0.5867
        - IoU classe 4: 0.5617
        - IoU classe 5: 0.5661
        - IoU classe 6: 0.1902
        - IoU classe 7: 0.8245
        - IoU classe 8: 0.8751

    - Migliore mean IoU ottenuta all'epoca 18:
    - Train Loss: 0.0781 | Val Loss: 0.9812 | Mean IoU: 0.6285
    - IoU classe 0: 0.5387
    - IoU classe 1: 0.6149
    - IoU classe 2: 0.6248
    - IoU classe 3: 0.5763
    - IoU classe 4: 0.6435
    - IoU classe 5: 0.6451
    - IoU classe 6: 0.1964
    - IoU classe 7: 0.8354
    - IoU classe 8: 0.8913

- **Rappresentazione grafica del training:**



- **Massima memoria RAM occupata durante il training:** 2.61GB
- **Log di training**: `risultati_training\deeplabv3\resnet101\dlrn1010001\training_log_dlrn1010002.json`
- **Note / Bug Fix / Annotazioni**:
    - picchi anomali ogni 25 epoche circa

✳ Esperimento: `deeplabv3_resnet101 - dlrn101003`

- **Architettura**: `deeplabv3plus_resnet101`
- **Backbone Weights**: DeepLabV3_ResNet101_Weights.DEFAULT
- **Input size**: 256x256
- **Test size**: 512x512

```
        # Augmentazione di base
        val_aug = A.Compose([
            A.Resize(512, 512),
            A.Normalize(mean=(0.485, 0.456, 0.406),
                        std=(0.229, 0.224, 0.225)),
```

```
            ToTensorV2()
        ])
```

- **Batch size**: 4
- **Augmentation**: on

```
# Augmentazione per immagini con classi rare
rare_aug = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.4),
    #A.Rotate(limit=20, p=0.3), potenzialmente inutile
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])

# Augmentazione di base
base_aug = A.Compose([
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])
```

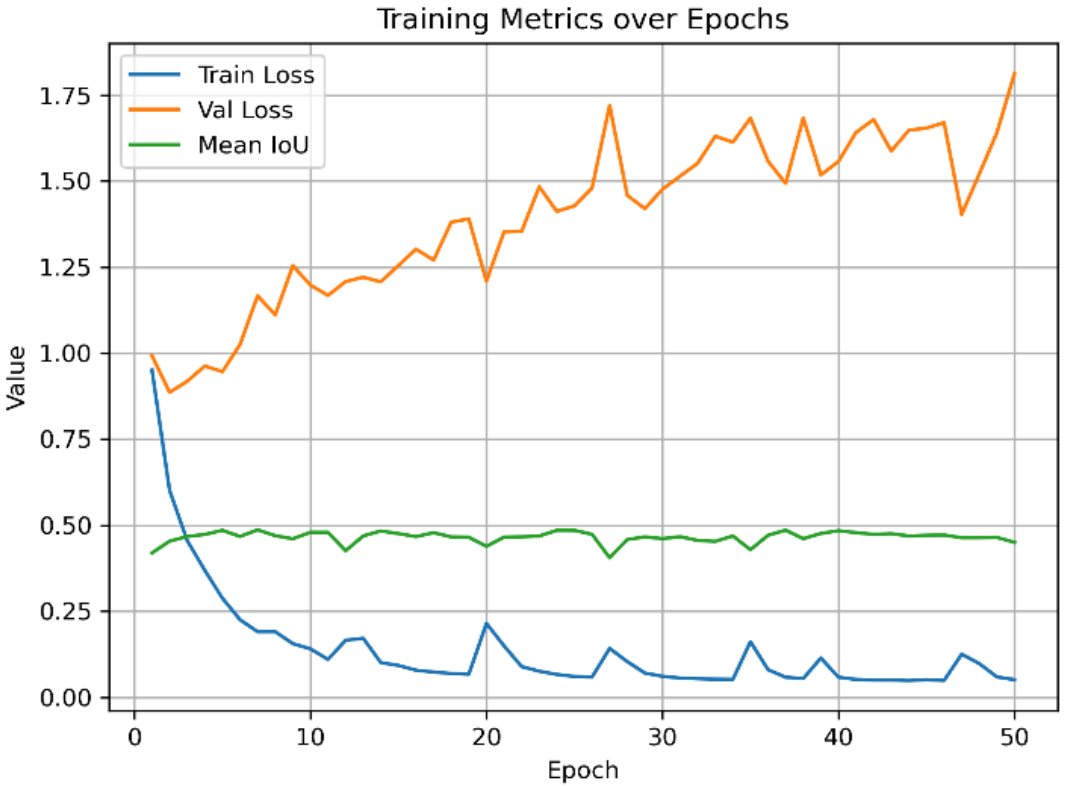
```
def _load_and_augment(self, root_dir):
    sample_dirs = []
    for id in self.sample_ids:
        sample_dirs.append(os.path.join(root_dir, id))

    for dir_path in sample_dirs:
        img_path = os.path.join(dir_path, 'rgb.jpg')
        label_path = os.path.join(dir_path, 'labels.png')

        label_img = Image.open(label_path)
        is_rare =
SegmentationDataset.contains_rare_classes(label_img,
RARE_CLASSES)

        if not self.allow_augmentation:
            is_rare = False
```

```
        # Sempre aggiungi l'originale
        self.samples.append((img_path, label_path,
is_rare))

        # Duplicazione per oversampling (solo se contiene
classi rare)
        if is_rare and not self.is_val:
            for _ in range(DUPLICATE_FACTOR - 1):
                self.samples.append((img_path,
label_path, is_rare))
```

- **Scheduler**: off
- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy + class weights euristici [1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8].
- **Main classifier Kernel size:** 1
- **Valutazione risultati**:
    - Migliore Validation Loss ottenuta all'epoca 2:
        - Train Loss: 0.6010 | Val Loss: 0.8861 | Mean IoU: 0.4538
        - IoU classe 0: 0.4030
        - IoU classe 1: 0.3244
        - IoU classe 2: 0.4519
        - IoU classe 3: 0.4907
        - IoU classe 4: 0.1171
        - IoU classe 5: 0.4992
        - IoU classe 6: 0.1440
        - IoU classe 7: 0.7271
        - IoU classe 8: 0.8758

    - Migliore mean IoU ottenuta all'epoca 7:
        - Train Loss: 0.1906 | Val Loss: 1.1668 | Mean IoU: 0.4866
        - IoU classe 0: 0.2706
        - IoU classe 1: 0.3583
        - IoU classe 2: 0.4325
        - IoU classe 3: 0.4450
        - IoU classe 4: 0.2861
        - IoU classe 5: 0.5752

    - IoU classe 6: 0.1388
    - IoU classe 7: 0.7701
    - IoU classe 8: 0.8867
- **Rappresentazione grafica del training:**



- **Massima memoria RAM occupata durante il training:** 2.61GB
- **Log di training**: `risultati_training\deeplabv3\resnet101\dlrn1010003\training_log_dlrn1010003.json`
- **Note / Bug Fix / Annotazioni**:
    - picchi anomali ogni 25 epoche circa

✳ Esperimento: `deeplabv3_resnet101 - dlrn101004`

- **Architettura**: `deeplabv3plus_resnet101`
- **Backbone Weights**: DeepLabV3_ResNet101_Weights.DEFAULT
- **Input size**: 512x512
- **Test size**: 512x512

```
    # Augmentazione di base
    val_aug = A.Compose([
```

```
            A.Resize(512, 512),
            A.Normalize(mean=(0.485, 0.456, 0.406),
                        std=(0.229, 0.224, 0.225)),
            ToTensorV2()
        ])
```

- **Batch size**: 4
- **Augmentation**: on

```
# Augmentazione per immagini con classi rare
rare_aug = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.4),
    #A.Rotate(limit=20, p=0.3), potenzialmente inutile
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])

# Augmentazione di base
base_aug = A.Compose([
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])
```

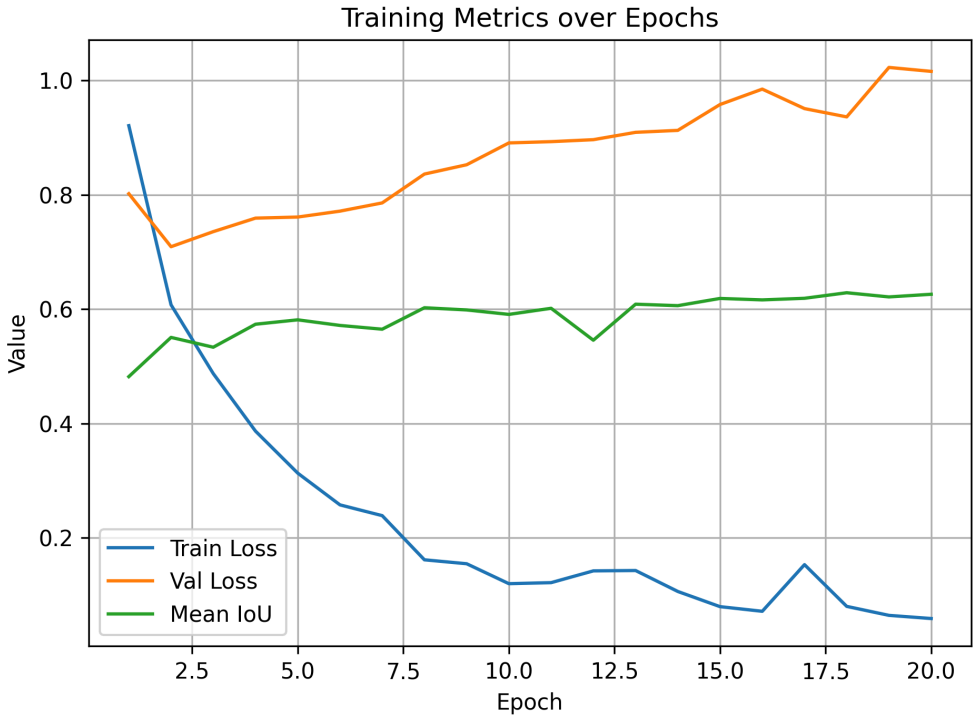
```
def _load_and_augment(self, root_dir):
    sample_dirs = []
    for id in self.sample_ids:
        sample_dirs.append(os.path.join(root_dir, id))

    for dir_path in sample_dirs:
        img_path = os.path.join(dir_path, 'rgb.jpg')
        label_path = os.path.join(dir_path, 'labels.png')

        label_img = Image.open(label_path)
        is_rare =
SegmentationDataset.contains_rare_classes(label_img,
RARE_CLASSES)
```

```
        if not self.allow_augmentation:
            is_rare = False

        # Sempre aggiungi l'originale
        self.samples.append((img_path, label_path,
is_rare))

        # Duplicazione per oversampling (solo se contiene
classi rare)
        if is_rare and not self.is_val:
            for _ in range(DUPLICATE_FACTOR - 1):
                self.samples.append((img_path,
label_path, is_rare))
```

- **Scheduler**: off
- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy + class weights euristici [1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8].
- **Main classifier Kernel size:** 1
- **Valutazione risultati**:
    - Migliore Validation Loss ottenuta all'epoca 2:
        - 📊 Train Loss: 0.6075 | Val Loss: 0.7091 | Mean IoU: 0.5504
        - ▸ IoU classe 0: 0.5259
        - ▸ IoU classe 1: 0.5440
        - ▸ IoU classe 2: 0.5052
        - ▸ IoU classe 3: 0.5481
        - ▸ IoU classe 4: 0.4803
        - ▸ IoU classe 5: 0.4454
        - ▸ IoU classe 6: 0.1824
        - ▸ IoU classe 7: 0.8097
        - ▸ IoU classe 8: 0.8885
    - Migliore mean IoU ottenuta all'epoca 18:
        - 📊 Train Loss: 0.0798 | Val Loss: 0.9362 | Mean IoU: 0.6286
        - ▸ IoU classe 0: 0.5765
        - ▸ IoU classe 1: 0.6088

    - ▸ IoU classe 2: 0.6368
    - ▸ IoU classe 3: 0.6030
    - ▸ IoU classe 4: 0.5410
    - ▸ IoU classe 5: 0.6921
    - ▸ IoU classe 6: 0.2046
    - ▸ IoU classe 7: 0.8388
    - ▸ IoU classe 8: 0.9034

- **Rappresentazione grafica del training:**



- **Massima memoria RAM occupata durante il training:** 8.25GB

- **Log di training**: `risultati_training\deeplabv3\resnet101\dlrn1010004\training_log_dlrn1010004.json`

- **Note / Bug Fix / Annotazioni**:

    - con size 512x512 sia sul test set che sul training set, questo modello è stato scartato in quanto troppo esoso in termini di memoria RAM

✳ Esperimento: `deeplabv3_resnet101 - dlrn101005`

- **Architettura**: `deeplabv3plus_resnet101`

- **Backbone Weights**: DeepLabV3_ResNet101_Weights.DEFAULT
- **Input size**: 256x256
- **Batch size**: 4
- **Augmentation**: on

```
# Augmentazione per immagini con classi rare
rare_aug = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.4),
    #A.Rotate(limit=20, p=0.3), potenzialmente inutile
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])

# Augmentazione di base
base_aug = A.Compose([
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])
```

```
def _load_and_augment(self, root_dir):
    sample_dirs = []
    for id in self.sample_ids:
        sample_dirs.append(os.path.join(root_dir, id))

    for dir_path in sample_dirs:
        img_path = os.path.join(dir_path, 'rgb.jpg')
        label_path = os.path.join(dir_path, 'labels.png')

        label_img = Image.open(label_path)
        is_rare =
SegmentationDataset.contains_rare_classes(label_img,
RARE_CLASSES)

        if not self.allow_augmentation:
            is_rare = False

        # Sempre aggiungi l'originale
```

```
        self.samples.append((img_path, label_path,
is_rare))

        # Duplicazione per oversampling (solo se contiene
classi rare)
        if is_rare and not self.is_val:
            for _ in range(DUPLICATE_FACTOR - 1):
                self.samples.append((img_path,
label_path, is_rare))
```

- **Scheduler**: off
- **Splitting:** on

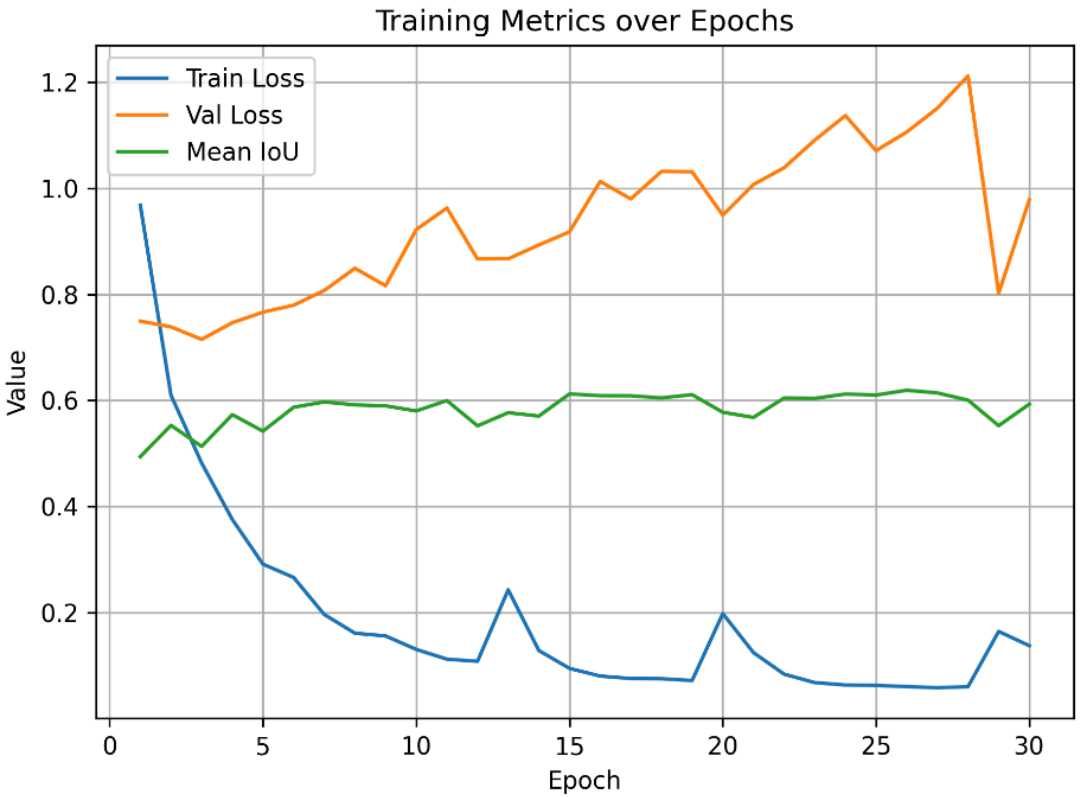
```
import numpy as np
from PIL import Image
import os
from iterstrat.ml_stratifiers import
MultilabelStratifiedShuffleSplit
import json

# sample male etichettati
excluded_dirs = {
    "0007", "0009", "0090", "0095", "0101", "0104",
"0105",
    "0162", "0163", "0284", "0305", "0306", "0307",
"0308", "0309", "0310", "0311",
    "0351", "0372", "0373", "0376",
    "0498", "0499", "0500", "0501", "0526", "0527",
"0530", "0531", "0542",
    "0564", "0585", "0586", "0587", "0588", "0589",
"0590",
    #seguono nuovo immagini da escludere
    "0000", "0001", "0052"
}

# Elenco di cartelle numeriche valide
all_dirs = sorted([
    d for d in os.listdir(DATASET_DIR)
    if os.path.isdir(os.path.join(DATASET_DIR, d)) and
d.isdigit() and d not in excluded_dirs
])

# ------------------------------
#  COSTRUZIONE DELLA MATRICE MULTILABEL
# ------------------------------
def build_presence_matrix(dir_list, n_classes=9):
```

```
    """Return np.array (n_samples, n_classes) with
booleans indicating class presence"""
    presence = np.zeros((len(dir_list), n_classes),
dtype=int)

    for idx, d in enumerate(dir_list):
        label_path = os.path.join(DATASET_DIR, d,
"labels.png")
        lbl = np.array(Image.open(label_path))
        uniq = np.unique(lbl)
        presence[idx, uniq] = 1          # segna le
classi presenti
    return presence

Y = build_presence_matrix(all_dirs, NUM_CLASSES)

# ------------------------------
#  MULTILABEL STRATIFIED SPLIT
# ------------------------------
msss = MultilabelStratifiedShuffleSplit(n_splits=1,
test_size=0.20, random_state=42)
train_idx, val_idx =
next(msss.split(np.zeros(len(all_dirs)), Y))

train_ids = [all_dirs[i] for i in train_idx]
val_ids   = [all_dirs[i] for i in val_idx]

print("Campioni:", len(train_ids), "train |",
len(val_ids), "val")
```

- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy + class weights euristici [1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8].
- **Main classifier Kernel size:** 1
- **Valutazione risultati**:
    - Migliore Validation Loss ottenuta all’epoca 3:
    - 📊 Train Loss: 0.4821 | Val Loss: 0.7150 | Mean IoU: 0.5133
    - ▸ IoU classe 0: 0.4856

- ▸ IoU classe 1: 0.6092
- ▸ IoU classe 2: 0.5384
- ▸ IoU classe 3: 0.6030
- ▸ IoU classe 4: 0.1352
- ▸ IoU classe 5: 0.2619
- ▸ IoU classe 6: 0.3044
- ▸ IoU classe 7: 0.7835
- ▸ IoU classe 8: 0.8706
-
- Migliore mean IoU ottenuta all'epoca 26:
- 📊 Train Loss: 0.0608 | Val Loss: 1.1057 | Mean IoU: 0.6191
- ▸ IoU classe 0: 0.5361
- ▸ IoU classe 1: 0.7011
- ▸ IoU classe 2: 0.6249
- ▸ IoU classe 3: 0.6641
- ▸ IoU classe 4: 0.1964
- ▸ IoU classe 5: 0.7440
- ▸ IoU classe 6: 0.3067
- ▸ IoU classe 7: 0.8261
- ▸ IoU classe 8: 0.8899

- **Rappresentazione grafica del training:**



- **Massima memoria RAM occupata durante il training:** 2.64GB

- **Log di training**: `risultati_training\deeplabv3\resnet101\dlrn1010001\training_log_dlrn1010005.json`
- **Note / Bug Fix / Annotazioni**:
    - picchi anomali ogni 10 epoche circa

✳ Esperimento: `deeplabv3_resnet101 - dlrn101006`

- **Architettura**: `deeplabv3plus_resnet101`
- **Backbone Weights**: DeepLabV3_ResNet101_Weights.DEFAULT
- **Input size**: 256x256
- **Batch size**: 4
- **Augmentation**: on

```
# Augmentazione per immagini con classi rare
rare_aug = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.4),
    #A.Rotate(limit=20, p=0.3), potenzialmente inutile
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])

# Augmentazione di base
base_aug = A.Compose([
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])
```

```
def _load_and_augment(self, root_dir):
    sample_dirs = []
    for id in self.sample_ids:
        sample_dirs.append(os.path.join(root_dir, id))

    for dir_path in sample_dirs:
        img_path = os.path.join(dir_path, 'rgb.jpg')
```

```
        label_path = os.path.join(dir_path, 'labels.png')

        label_img = Image.open(label_path)
        is_rare =
SegmentationDataset.contains_rare_classes(label_img,
RARE_CLASSES)

        if not self.allow_augmentation:
            is_rare = False

        # Sempre aggiungi l'originale
        self.samples.append((img_path, label_path,
is_rare))

        # Duplicazione per oversampling (solo se contiene
classi rare)
        if is_rare and not self.is_val:
            for _ in range(DUPLICATE_FACTOR - 1):
                self.samples.append((img_path,
label_path, is_rare))
```

- **Scheduler**: off
- **Splitting:** on

```
import numpy as np
from PIL import Image
import os
from iterstrat.ml_stratifiers import
MultilabelStratifiedShuffleSplit
import json

# sample male etichettati
excluded_dirs = {
    "0007", "0009", "0090", "0095", "0101", "0104",
"0105",
    "0162", "0163", "0284", "0305", "0306", "0307",
"0308", "0309", "0310", "0311",
    "0351", "0372", "0373", "0376",
    "0498", "0499", "0500", "0501", "0526", "0527",
"0530", "0531", "0542",
    "0564", "0585", "0586", "0587", "0588", "0589",
"0590",
    #seguono nuovo immagini da escludere
    "0000", "0001", "0052"
}
```

```
# Elenco di cartelle numeriche valide
all_dirs = sorted([
    d for d in os.listdir(DATASET_DIR)
    if os.path.isdir(os.path.join(DATASET_DIR, d)) and
d.isdigit() and d not in excluded_dirs
])

# ------------------------------
#  COSTRUZIONE DELLA MATRICE MULTILABEL
# ------------------------------
def build_presence_matrix(dir_list, n_classes=9):
    """Return np.array (n_samples, n_classes) with
booleans indicating class presence"""
    presence = np.zeros((len(dir_list), n_classes),
dtype=int)

    for idx, d in enumerate(dir_list):
        label_path = os.path.join(DATASET_DIR, d,
"labels.png")
        lbl = np.array(Image.open(label_path))
        uniq = np.unique(lbl)
        presence[idx, uniq] = 1          # segna le
classi presenti
    return presence

Y = build_presence_matrix(all_dirs, NUM_CLASSES)

# ------------------------------
#  MULTILABEL STRATIFIED SPLIT
# ------------------------------
msss = MultilabelStratifiedShuffleSplit(n_splits=1,
test_size=0.20, random_state=42)
train_idx, val_idx =
next(msss.split(np.zeros(len(all_dirs)), Y))

train_ids = [all_dirs[i] for i in train_idx]
val_ids   = [all_dirs[i] for i in val_idx]

print("Campioni:", len(train_ids), "train |",
len(val_ids), "val")
```

- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none

- **Auxiliary Head**: off

- **Loss Function / Weighting**: cross-entropy + class weights euristici [1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8].

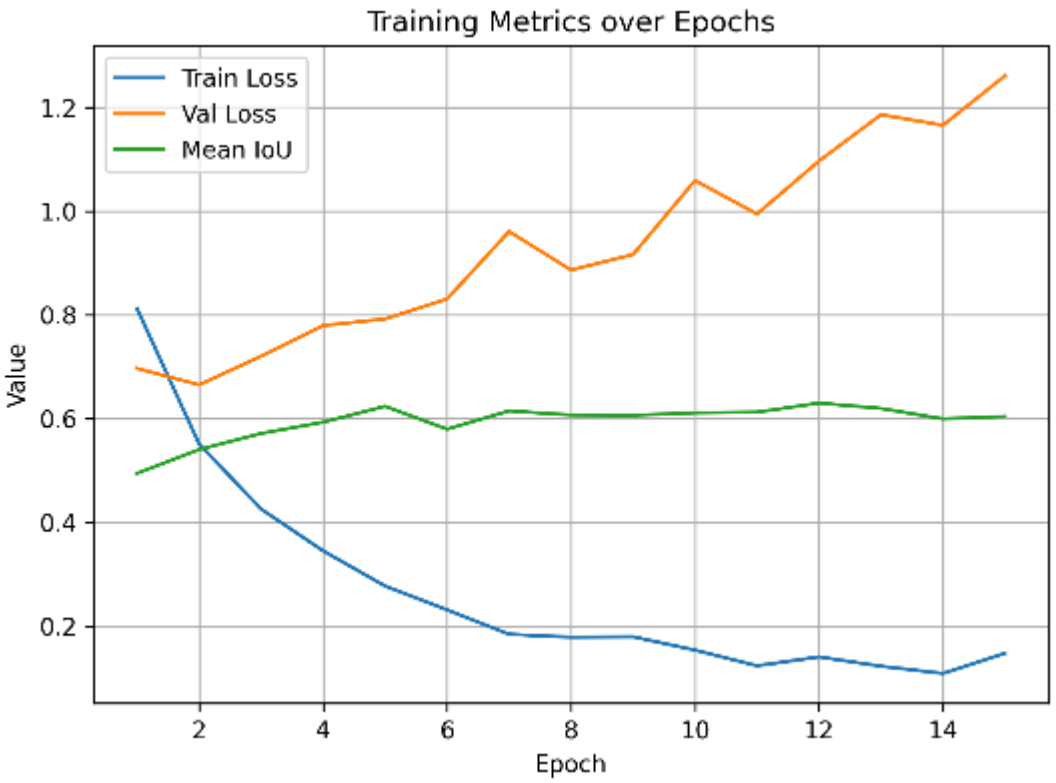
- **Main classifier Kernel size:** 5
- **ASPP:** stride [3, 7, 3]
    - ```
      model = models.segmentation.deeplabv3_resnet101(weights=weights)
      # versione custom del classifier
      model.classifier = CustomDeepLabHead(2048, NUM_CLASSES)
      ```

    ```
    class CustomDeepLabHead(nn.Sequential):
        def __init__(self, in_channels, num_classes):
            super().__init__(
                ASPP(in_channels, [3, 7, 3]),  # ASPP con dilatazioni personalizzate
                nn.Conv2d(256, 256, kernel_size=5, padding=2),  # kernel 5x5
                nn.ReLU(),
                nn.Dropout(0),
                nn.Conv2d(256, num_classes, kernel_size=1)  # output classi
            )
    ```

- **Valutazione risultati**:

  Migliore Validation Loss ottenuta all'epoca 2:

  📊 Train Loss: 0.5513 | Val Loss: 0.6654 | Mean IoU: 0.5404

  - IoU classe 0: 0.5130
  - IoU classe 1: 0.6549
  - IoU classe 2: 0.4952
  - IoU classe 3: 0.6251
  - IoU classe 4: 0.5213
  - IoU classe 5: 0.1592
  - IoU classe 6: 0.2579
    ```

- ▸ IoU classe 7: 0.7432
- ▸ IoU classe 8: 0.8667

Migliore mean IoU ottenuta all'epoca 12:

📊 Train Loss: 0.1404 | Val Loss: 1.0972 | Mean IoU: 0.6302

- ▸ IoU classe 0: 0.5576
- ▸ IoU classe 1: 0.6834
- ▸ IoU classe 2: 0.5835
- ▸ IoU classe 3: 0.6061
- ▸ IoU classe 4: 0.6778
- ▸ IoU classe 5: 0.4874
- ▸ IoU classe 6: 0.2635
- ▸ IoU classe 7: 0.8385
- ▸ IoU classe 8: 0.9014

- **Rappresentazione grafica del training:**



- **Massima memoria RAM occupata durante il training:** 3.57GB
- **Log di training**: `risultati_training\deeplabv3\resnet101\dlrn1010001\training_log_dlrn1010006.json`
- **Note / Bug Fix / Annotazioni**:

✳ Esperimento: `deeplabv3_resnet101 - dlrn101007`

uguale a dlrn101006 ma con dilazione ASPP [1, 3, 5] con risultati peggiori perchè Migliore Validation Loss ottenuta all'epoca 2 con Mean IoU: 0.53

✳ Esperimento: `deeplabv3_resnet101 - dlrn101008`

- **Architettura**: `deeplabv3plus_resnet101`
- **Backbone Weights**: DeepLabV3_ResNet101_Weights.DEFAULT
- **Input size**: 256x256
- **Batch size**: 4
- **Augmentation**: on
  - `# Augmentazione per immagini con classi rare`

```
rare_aug = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.4),
    #A.Rotate(limit=20, p=0.3), potenzialmente inutile
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])

# Augmentazione di base
base_aug = A.Compose([
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])
```

```
def _load_and_augment(self, root_dir):
    sample_dirs = []
    for id in self.sample_ids:
        sample_dirs.append(os.path.join(root_dir, id))

    for dir_path in sample_dirs:
        img_path = os.path.join(dir_path, 'rgb.jpg')
        label_path = os.path.join(dir_path, 'labels.png')

        label_img = Image.open(label_path)
        is_rare =
SegmentationDataset.contains_rare_classes(label_img,
RARE_CLASSES)

        if not self.allow_augmentation:
            is_rare = False

        # Sempre aggiungi l'originale
        self.samples.append((img_path, label_path,
is_rare))

        # Duplicazione per oversampling (solo se contiene
classi rare)
        if is_rare and not self.is_val:
            for _ in range(DUPLICATE_FACTOR - 1):
                self.samples.append((img_path,
label_path, is_rare))
```

- **Scheduler**: off
- **Splitting:** on

```
import numpy as np
from PIL import Image
import os
from iterstrat.ml_stratifiers import MultilabelStratifiedShuffleSplit
import json

# sample male etichettati
excluded_dirs = {
    "0007", "0009", "0090", "0095", "0101", "0104", "0105",
    "0162", "0163", "0284", "0305", "0306", "0307", "0308", "0309", "0310", "0311",
    "0351", "0372", "0373", "0376",
    "0498", "0499", "0500", "0501", "0526", "0527", "0530", "0531", "0542",
    "0564", "0585", "0586", "0587", "0588", "0589", "0590",
    #seguono nuovo immagini da escludere
    "0000", "0001", "0052"
}

# Elenco di cartelle numeriche valide
all_dirs = sorted([
    d for d in os.listdir(DATASET_DIR)
    if os.path.isdir(os.path.join(DATASET_DIR, d)) and d.isdigit() and d not in excluded_dirs
])

# ------------------------------
#  COSTRUZIONE DELLA MATRICE MULTILABEL
# ------------------------------
def build_presence_matrix(dir_list, n_classes=9):
    """Return np.array (n_samples, n_classes) with booleans indicating class presence"""
    presence = np.zeros((len(dir_list), n_classes), dtype=int)

    for idx, d in enumerate(dir_list):
        label_path = os.path.join(DATASET_DIR, d, "labels.png")
        lbl = np.array(Image.open(label_path))
```

```
        uniq = np.unique(lbl)
        presence[idx, uniq] = 1          # segna le
classi presenti
    return presence

Y = build_presence_matrix(all_dirs, NUM_CLASSES)

# ------------------------------
#  MULTILABEL STRATIFIED SPLIT
# ------------------------------
msss = MultilabelStratifiedShuffleSplit(n_splits=1,
test_size=0.20, random_state=42)
train_idx, val_idx =
next(msss.split(np.zeros(len(all_dirs)), Y))

train_ids = [all_dirs[i] for i in train_idx]
val_ids   = [all_dirs[i] for i in val_idx]

print("Campioni:", len(train_ids), "train |",
len(val_ids), "val")
```

- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy + class weights euristici [1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8].
- **Main classifier Kernel size:** 5
- **ASPP e dropout :** stride [3, 7, 3] e 0.1

```
model =
models.segmentation.deeplabv3_resnet101(weights=weights)
# versione custom del classifier
model.classifier = CustomDeepLabHead(2048, NUM_CLASSES)
```

```

class CustomDeepLabHead(nn.Sequential):
    def __init__(self, in_channels, num_classes):
        super().__init__(
            ASPP(in_channels, [3, 7, 3]),  # ASPP con
dilatazioni personalizzate
            nn.Conv2d(256, 256, kernel_size=5,
padding=2),  # kernel 5x5
```

```
            nn.ReLU(),
            nn.Dropout(0.1),
            nn.Conv2d(256, num_classes, kernel_size=1)
# output classi
        )
```

- **Valutazione risultati**:
  - Migliore Validation Loss ottenuta all'epoca 2:
  - 📊 Train Loss: 0.5711 | Val Loss: 0.6978 | Mean IoU: 0.5658
  - ▸ IoU classe 0: 0.4895
  - ▸ IoU classe 1: 0.6365
  - ▸ IoU classe 2: 0.3813
  - ▸ IoU classe 3: 0.5872
  - ▸ IoU classe 4: 0.6044
  - ▸ IoU classe 5: 0.3580
  - ▸ IoU classe 6: 0.2886
  - ▸ IoU classe 7: 0.7859
  - ▸ IoU classe 8: 0.8849
  -
  - Migliore mean IoU ottenuta all'epoca 16:
  - 📊 Train Loss: 0.1089 | Val Loss: 1.0747 | Mean IoU: 0.6474
  - ▸ IoU classe 0: 0.5505
  - ▸ IoU classe 1: 0.6855
  - ▸ IoU classe 2: 0.5956
  - ▸ IoU classe 3: 0.6407
  - ▸ IoU classe 4: 0.7616
  - ▸ IoU classe 5: 0.4601
  - ▸ IoU classe 6: 0.3005
  - ▸ IoU classe 7: 0.8384
  - ▸ IoU classe 8: 0.8966

- **Rappresentazione grafica del training:**

- **Massima memoria RAM occupata durante il training:** 2.658GB
- **Log di training**: `risultati_training\deeplabv3\resnet101\dlrn1010001\training_log_dlrn1010008.json`
- **Note / Bug Fix / Annotazioni**:
    - picchi anomali ogni 10 epoche circa

✳ Esperimento: `deeplabv3_resnet101 - dlrn101009`

- **Architettura**: `deeplabv3plus_resnet101`
- **Backbone Weights**: DeepLabV3_ResNet101_Weights.DEFAULT
- **Input size**: 256x256
- **Batch size**: 4
- **Augmentation**: on

```
# Augmentazione per immagini con classi rare
rare_aug = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.4),
```

```
    #A.Rotate(limit=20, p=0.3), potenzialmente inutile
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])

# Augmentazione di base
base_aug = A.Compose([
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])
```

```
def _load_and_augment(self, root_dir):
    sample_dirs = []
    for id in self.sample_ids:
        sample_dirs.append(os.path.join(root_dir, id))

    for dir_path in sample_dirs:
        img_path = os.path.join(dir_path, 'rgb.jpg')
        label_path = os.path.join(dir_path, 'labels.png')

        label_img = Image.open(label_path)
        is_rare =
SegmentationDataset.contains_rare_classes(label_img,
RARE_CLASSES)

        if not self.allow_augmentation:
            is_rare = False

        # Sempre aggiungi l'originale
        self.samples.append((img_path, label_path,
is_rare))

        # Duplicazione per oversampling (solo se contiene
classi rare)
        if is_rare and not self.is_val:
            for _ in range(DUPLICATE_FACTOR - 1):
                self.samples.append((img_path,
label_path, is_rare))
```

- **Scheduler**: off
- **Splitting:** on

```
import numpy as np
from PIL import Image
import os
from iterstrat.ml_stratifiers import
MultilabelStratifiedShuffleSplit
import json

# sample male etichettati
excluded_dirs = {
    "0007", "0009", "0090", "0095", "0101", "0104",
"0105",
    "0162", "0163", "0284", "0305", "0306", "0307",
"0308", "0309", "0310", "0311",
    "0351", "0372", "0373", "0376",
    "0498", "0499", "0500", "0501", "0526", "0527",
"0530", "0531", "0542",
    "0564", "0585", "0586", "0587", "0588", "0589",
"0590",
    #seguono nuovo immagini da escludere
    "0000", "0001", "0052"
}

# Elenco di cartelle numeriche valide
all_dirs = sorted([
    d for d in os.listdir(DATASET_DIR)
    if os.path.isdir(os.path.join(DATASET_DIR, d)) and
d.isdigit() and d not in excluded_dirs
])

# ------------------------------
#  COSTRUZIONE DELLA MATRICE MULTILABEL
# ------------------------------
def build_presence_matrix(dir_list, n_classes=9):
    """Return np.array (n_samples, n_classes) with
booleans indicating class presence"""
    presence = np.zeros((len(dir_list), n_classes),
dtype=int)

    for idx, d in enumerate(dir_list):
        label_path = os.path.join(DATASET_DIR, d,
"labels.png")
        lbl = np.array(Image.open(label_path))
        uniq = np.unique(lbl)
        presence[idx, uniq] = 1          # segna le
classi presenti
```

```
    return presence

Y = build_presence_matrix(all_dirs, NUM_CLASSES)

# ------------------------------
#  MULTILABEL STRATIFIED SPLIT
# ------------------------------
msss = MultilabelStratifiedShuffleSplit(n_splits=1,
test_size=0.20, random_state=42)
train_idx, val_idx =
next(msss.split(np.zeros(len(all_dirs)), Y))

train_ids = [all_dirs[i] for i in train_idx]
val_ids   = [all_dirs[i] for i in val_idx]

print("Campioni:", len(train_ids), "train |",
len(val_ids), "val")
```

- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy + class weights euristici [1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8].
- **Main classifier Kernel size:** 5
- **ASPP e dropout :** stride [3, 7, 3] e 0.2

```
model =
models.segmentation.deeplabv3_resnet101(weights=weights)
# versione custom del classifier
model.classifier = CustomDeepLabHead(2048, NUM_CLASSES)
```

```
class CustomDeepLabHead(nn.Sequential):
    def __init__(self, in_channels, num_classes):
        super().__init__(
            ASPP(in_channels, [3, 7, 3]),  # ASPP con
dilatazioni personalizzate
            nn.Conv2d(256, 256, kernel_size=5,
padding=2),  # kernel 5x5
            nn.ReLU(),
            nn.Dropout(0.2),
```

```
            nn.Conv2d(256, num_classes, kernel_size=1)
# output classi
        )
```

- **Valutazione risultati**:
  - Migliore Validation Loss ottenuta all'epoca 1:
  - 📊 Train Loss: 0.8053 | Val Loss: 0.7467 | Mean IoU: 0.4639
  - ▸ IoU classe 0: 0.4377
  - ▸ IoU classe 1: 0.5977
  - ▸ IoU classe 2: 0.4377
  - ▸ IoU classe 3: 0.5946
  - ▸ IoU classe 4: 0.0000
  - ▸ IoU classe 5: 0.2050
  - ▸ IoU classe 6: 0.2629
  - ▸ IoU classe 7: 0.7605
  - ▸ IoU classe 8: 0.8529
  -
  - Migliore mean IoU ottenuta all'epoca 15:
  - 📊 Train Loss: 0.0914 | Val Loss: 1.1984 | Mean IoU: 0.6351
  - ▸ IoU classe 0: 0.5441
  - ▸ IoU classe 1: 0.6598
  - ▸ IoU classe 2: 0.5785
  - ▸ IoU classe 3: 0.6332
  - ▸ IoU classe 4: 0.7150
  - ▸ IoU classe 5: 0.4863
  - ▸ IoU classe 6: 0.2818
  - ▸ IoU classe 7: 0.8304
  - ▸ IoU classe 8: 0.8961

- **Rappresentazione grafica del training:**

- **Massima memoria RAM occupata durante il training: 3.57**GB
- **Log di training**: `risultati_training\deeplabv3\resnet101\dlrn1010001\training_log_dlrn1010009.json`
- **Note / Bug Fix / Annotazioni**:

✳ Esperimento: `deeplabv3_resnet101 - dlrn101010`

- **Architettura**: `deeplabv3plus_resnet101`
- **Backbone Weights**: DeepLabV3_ResNet101_Weights.DEFAULT
- **Input size**: 256x256
- **Batch size**: 4
- **Augmentation**: on

```
# Augmentazione per immagini con classi rare
rare_aug = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.4),
    #A.Rotate(limit=20, p=0.3), potenzialmente inutile
```

```
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])

# Augmentazione di base
base_aug = A.Compose([
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])
```

```
def _load_and_augment(self, root_dir):
    sample_dirs = []
    for id in self.sample_ids:
        sample_dirs.append(os.path.join(root_dir, id))

    for dir_path in sample_dirs:
        img_path = os.path.join(dir_path, 'rgb.jpg')
        label_path = os.path.join(dir_path, 'labels.png')

        label_img = Image.open(label_path)
        is_rare = SegmentationDataset.contains_rare_classes(label_img, RARE_CLASSES)

        if not self.allow_augmentation:
            is_rare = False

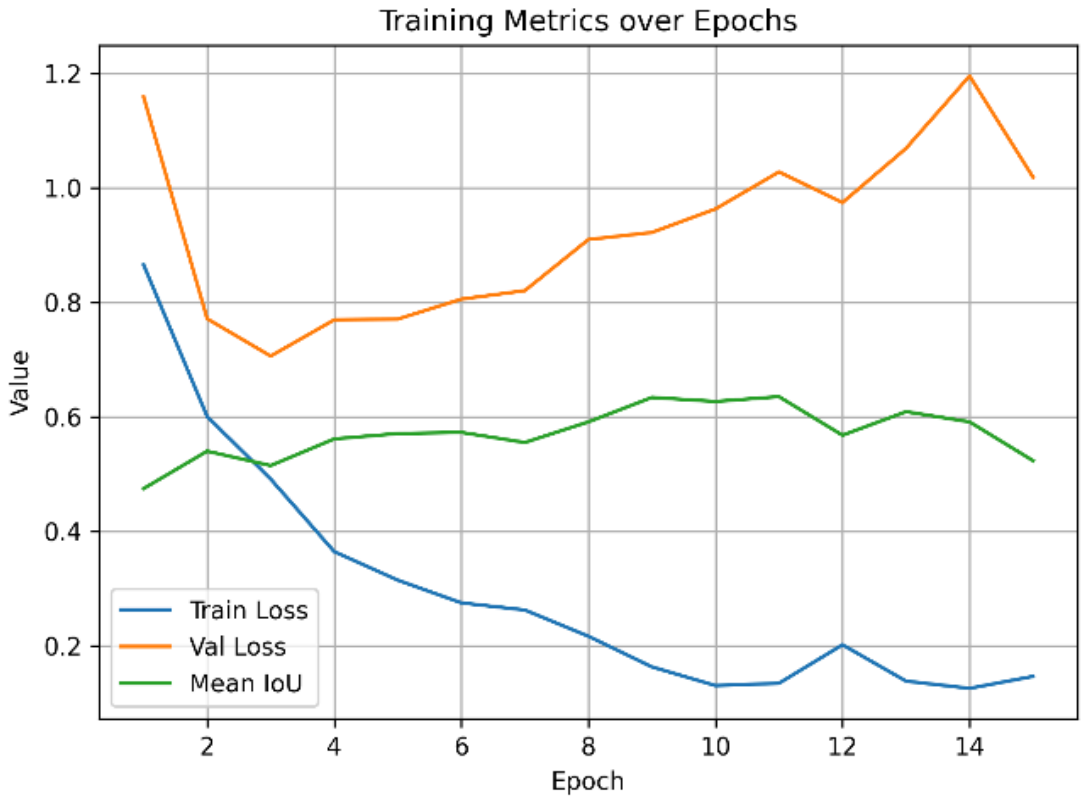
        # Sempre aggiungi l'originale
        self.samples.append((img_path, label_path, is_rare))

        # Duplicazione per oversampling (solo se contiene classi rare)
        if is_rare and not self.is_val:
            for _ in range(DUPLICATE_FACTOR - 1):
                self.samples.append((img_path, label_path, is_rare))
```

- **Scheduler**: off
- **Splitting:** on

```
import numpy as np
```

```
from PIL import Image
import os
from iterstrat.ml_stratifiers import
MultilabelStratifiedShuffleSplit
import json

# sample male etichettati
excluded_dirs = {
    "0007", "0009", "0090", "0095", "0101", "0104",
"0105",
    "0162", "0163", "0284", "0305", "0306", "0307",
"0308", "0309", "0310", "0311",
    "0351", "0372", "0373", "0376",
    "0498", "0499", "0500", "0501", "0526", "0527",
"0530", "0531", "0542",
    "0564", "0585", "0586", "0587", "0588", "0589",
"0590",
    #seguono nuovo immagini da escludere
    "0000", "0001", "0052"
}

# Elenco di cartelle numeriche valide
all_dirs = sorted([
    d for d in os.listdir(DATASET_DIR)
    if os.path.isdir(os.path.join(DATASET_DIR, d)) and
d.isdigit() and d not in excluded_dirs
])

# ------------------------------
#  COSTRUZIONE DELLA MATRICE MULTILABEL
# ------------------------------
def build_presence_matrix(dir_list, n_classes=9):
    """Return np.array (n_samples, n_classes) with
booleans indicating class presence"""
    presence = np.zeros((len(dir_list), n_classes),
dtype=int)

    for idx, d in enumerate(dir_list):
        label_path = os.path.join(DATASET_DIR, d,
"labels.png")
        lbl = np.array(Image.open(label_path))
        uniq = np.unique(lbl)
        presence[idx, uniq] = 1          # segna le
classi presenti
    return presence
```

```
o
o  Y = build_presence_matrix(all_dirs, NUM_CLASSES)
o
o  # ------------------------------
o  #  MULTILABEL STRATIFIED SPLIT
o  # ------------------------------
o  msss = MultilabelStratifiedShuffleSplit(n_splits=1,
   test_size=0.20, random_state=42)
o  train_idx, val_idx =
   next(msss.split(np.zeros(len(all_dirs)), Y))
o
o  train_ids = [all_dirs[i] for i in train_idx]
o  val_ids   = [all_dirs[i] for i in val_idx]
o
o  print("Campioni:", len(train_ids), "train |",
   len(val_ids), "val")
```

- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy + class weights euristici [1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8].
- **Main classifier Kernel size:** 5 e 3

```
o  # 🔁 Primo layer convoluzionale: kernel 3x3
o  nn.Conv2d(256, 256, kernel_size=3, padding=1),  #
   mantiene dimensione
o  nn.ReLU(),
o  # 🔁 Secondo layer convoluzionale: kernel 5x5
o  nn.Conv2d(256, 256, kernel_size=5, padding=2),  # campo
   visivo più ampio
o  nn.ReLU(),
```

- **ASPP:** stride [3, 7, 3]

```
o  model =
   models.segmentation.deeplabv3_resnet101(weights=weights)
o  # versione custom del classifier
o  model.classifier = CustomDeepLabHead(2048, NUM_CLASSES)
```

```
-
-  class CustomDeepLabHead(nn.Sequential):
-      def __init__(self, in_channels, num_classes):
-          super().__init__(
```

```
            ASPP(in_channels, [3, 7, 3]),  # ASPP con
dilatazioni personalizzate
            nn.Conv2d(256, 256, kernel_size=5,
padding=2),  # kernel 5x5
            nn.ReLU(),
            nn.Dropout(0),
            nn.Conv2d(256, num_classes, kernel_size=1)
# output classi
        )
```

- **Valutazione risultati**:
  Migliore Validation Loss ottenuta all'epoca 3:
  📊 Train Loss: 0.4911 | Val Loss: 0.7059 | Mean IoU: 0.5146
  - IoU classe 0: 0.5091
  - IoU classe 1: 0.6581
  - IoU classe 2: 0.5004
  - IoU classe 3: 0.6067
  - IoU classe 4: 0.1908
  - IoU classe 5: 0.1882
  - IoU classe 6: 0.2894
  - IoU classe 7: 0.8116
  - IoU classe 8: 0.8712

  Migliore mean IoU ottenuta all'epoca 11:
  📊 Train Loss: 0.1342 | Val Loss: 1.0277 | Mean IoU: 0.6350
  - IoU classe 0: 0.5409
  - IoU classe 1: 0.6768
  - IoU classe 2: 0.6031
  - IoU classe 3: 0.6327
  - IoU classe 4: 0.6595
  - IoU classe 5: 0.4610
  - IoU classe 6: 0.3177
  - IoU classe 7: 0.8350
  - IoU classe 8: 0.8941

- **Rappresentazione grafica del training:**



- **Massima memoria RAM occupata durante il training:** 4.03GB
- **Log di training**: `risultati_training\deeplabv3\resnet101\dlrn1010001\training_log_dlrn1010010.json`
- **Note / Bug Fix / Annotazioni**:

✳ Esperimento: `deeplabv3_resnet101 - dlrn101011`

- **Architettura**: `deeplabv3plus_resnet101`
- **Backbone Weights**: DeepLabV3_ResNet101_Weights.DEFAULT
- **Input size**: 256x256
- **Batch size**: 4
- **Augmentation**: on

```
# Augmentazione per immagini con classi rare
rare_aug = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.4),
    #A.Rotate(limit=20, p=0.3), potenzialmente inutile
```

```
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])

# Augmentazione di base
base_aug = A.Compose([
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])
```

```
def _load_and_augment(self, root_dir):
    sample_dirs = []
    for id in self.sample_ids:
        sample_dirs.append(os.path.join(root_dir, id))

    for dir_path in sample_dirs:
        img_path = os.path.join(dir_path, 'rgb.jpg')
        label_path = os.path.join(dir_path, 'labels.png')

        label_img = Image.open(label_path)
        is_rare = SegmentationDataset.contains_rare_classes(label_img, RARE_CLASSES)

        if not self.allow_augmentation:
            is_rare = False

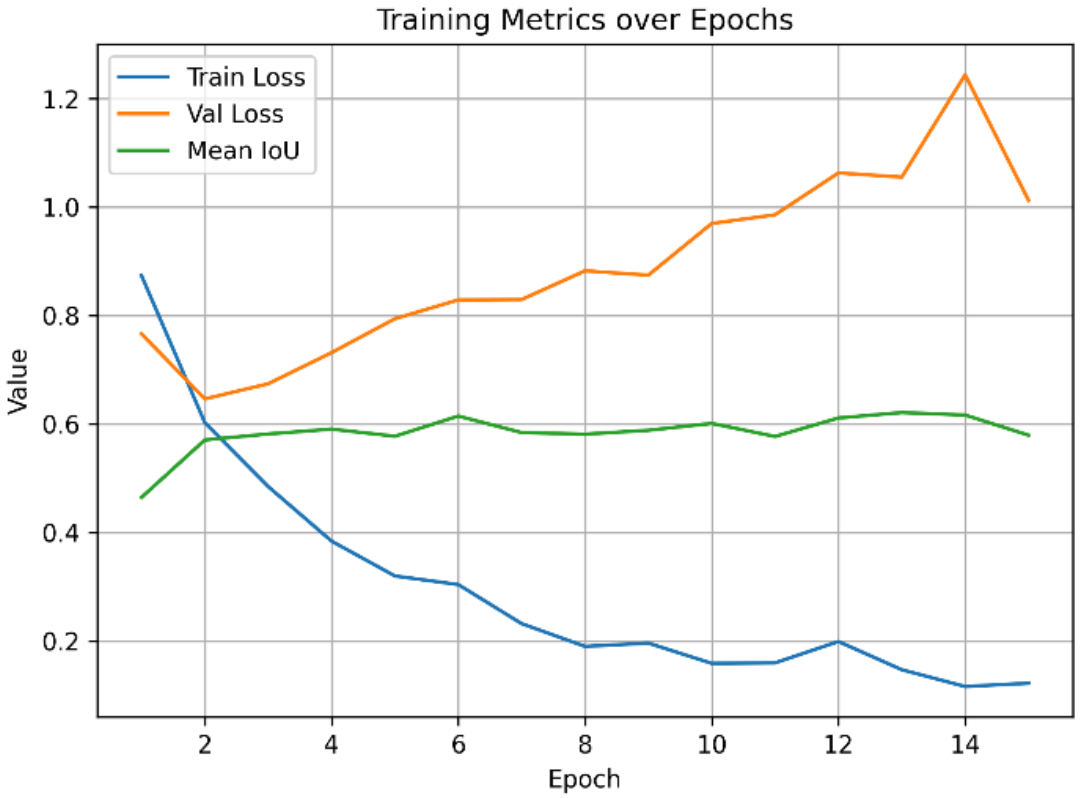
        # Sempre aggiungi l'originale
        self.samples.append((img_path, label_path, is_rare))

        # Duplicazione per oversampling (solo se contiene classi rare)
        if is_rare and not self.is_val:
            for _ in range(DUPLICATE_FACTOR - 1):
                self.samples.append((img_path, label_path, is_rare))
```

- **Scheduler**: off
- **Splitting:** on

```
import numpy as np
```

```
from PIL import Image
import os
from iterstrat.ml_stratifiers import
MultilabelStratifiedShuffleSplit
import json

# sample male etichettati
excluded_dirs = {
    "0007", "0009", "0090", "0095", "0101", "0104",
"0105",
    "0162", "0163", "0284", "0305", "0306", "0307",
"0308", "0309", "0310", "0311",
    "0351", "0372", "0373", "0376",
    "0498", "0499", "0500", "0501", "0526", "0527",
"0530", "0531", "0542",
    "0564", "0585", "0586", "0587", "0588", "0589",
"0590",
    #seguono nuovo immagini da escludere
    "0000", "0001", "0052"
}

# Elenco di cartelle numeriche valide
all_dirs = sorted([
    d for d in os.listdir(DATASET_DIR)
    if os.path.isdir(os.path.join(DATASET_DIR, d)) and
d.isdigit() and d not in excluded_dirs
])

# ------------------------------
#  COSTRUZIONE DELLA MATRICE MULTILABEL
# ------------------------------
def build_presence_matrix(dir_list, n_classes=9):
    """Return np.array (n_samples, n_classes) with
booleans indicating class presence"""
    presence = np.zeros((len(dir_list), n_classes),
dtype=int)

    for idx, d in enumerate(dir_list):
        label_path = os.path.join(DATASET_DIR, d,
"labels.png")
        lbl = np.array(Image.open(label_path))
        uniq = np.unique(lbl)
        presence[idx, uniq] = 1          # segna le
classi presenti
    return presence
```

```
o
o  Y = build_presence_matrix(all_dirs, NUM_CLASSES)
o
o  # ------------------------------
o  #  MULTILABEL STRATIFIED SPLIT
o  # ------------------------------
o  msss = MultilabelStratifiedShuffleSplit(n_splits=1,
   test_size=0.20, random_state=42)
o  train_idx, val_idx =
   next(msss.split(np.zeros(len(all_dirs)), Y))
o
o  train_ids = [all_dirs[i] for i in train_idx]
o  val_ids   = [all_dirs[i] for i in val_idx]
o
o  print("Campioni:", len(train_ids), "train |",
   len(val_ids), "val")
```

- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy + class weights euristici [1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8].
- **Main classifier Kernel size:** due di dimensione 5

```
-  # 🔁 Primo layer convoluzionale: kernel 3x3
-  nn.Conv2d(256, 256, kernel_size=5, padding=2),  #
   mantiene dimensione
-  nn.ReLU(),
-  # 🔁 Secondo layer convoluzionale: kernel 5x5
-  nn.Conv2d(256, 256, kernel_size=5, padding=2),  # campo
   visivo più ampio
-  nn.ReLU(),
```

- **ASPP:** stride [3, 7, 3]

```
o  model =
   models.segmentation.deeplabv3_resnet101(weights=weights)
o  # versione custom del classifier
o  model.classifier = CustomDeepLabHead(2048, NUM_CLASSES)
```

```
-
-  class CustomDeepLabHead(nn.Sequential):
-     def __init__(self, in_channels, num_classes):
-         super().__init__(
```

```
            ASPP(in_channels, [3, 7, 3]),  # ASPP con
dilatazioni personalizzate
            nn.Conv2d(256, 256, kernel_size=5,
padding=2),  # kernel 5x5
            nn.ReLU(),
            nn.Dropout(0),
            nn.Conv2d(256, num_classes, kernel_size=1)
# output classi
        )
```

- **Valutazione risultati**:

  Migliore Validation Loss ottenuta all'epoca 2:

  📊 Train Loss: 0.6026 | Val Loss: 0.6464 | Mean IoU: 0.5708

  - IoU classe 0: 0.4810
  - IoU classe 1: 0.6194
  - IoU classe 2: 0.5079
  - IoU classe 3: 0.6208
  - IoU classe 4: 0.6019
  - IoU classe 5: 0.2128
  - IoU classe 6: 0.3210
  - IoU classe 7: 0.8015
  - IoU classe 8: 0.8805

  Migliore mean IoU ottenuta all'epoca 13:

  📊 Train Loss: 0.1472 | Val Loss: 1.0552 | Mean IoU: 0.6213

  - IoU classe 0: 0.5387
  - IoU classe 1: 0.6678
  - IoU classe 2: 0.5658
  - IoU classe 3: 0.5696
  - IoU classe 4: 0.7548

- IoU classe 5: 0.3527
- IoU classe 6: 0.3220
- IoU classe 7: 0.8402
- IoU classe 8: 0.8972

- **Rappresentazione grafica del training:**



- **Massima memoria RAM occupata durante il training:** 2.7GB
- **Log di training**: `risultati_training\deeplabv3\resnet101\dlrn1010001\training_log_dlrn101011.json`
- **Note / Bug Fix / Annotazioni**:

✳ Esperimento: `deeplabv3_resnet101 - dlrn101012`

- **Architettura**: `deeplabv3plus_resnet101`
- **Backbone Weights**: DeepLabV3_ResNet101_Weights.DEFAULT
- **Input size**: 256x256

- **Batch size**: 4
- **Augmentation**: on
    -
```
# Augmentazione per immagini con classi rare
rare_aug = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.4),
    #A.Rotate(limit=20, p=0.3), potenzialmente inutile
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])

# Augmentazione di base
base_aug = A.Compose([
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])
```

```
def _load_and_augment(self, root_dir):
    sample_dirs = []
    for id in self.sample_ids:
        sample_dirs.append(os.path.join(root_dir, id))

    for dir_path in sample_dirs:
        img_path = os.path.join(dir_path, 'rgb.jpg')
        label_path = os.path.join(dir_path, 'labels.png')

        label_img = Image.open(label_path)
        is_rare =
SegmentationDataset.contains_rare_classes(label_img,
RARE_CLASSES)

        if not self.allow_augmentation:
            is_rare = False

        # Sempre aggiungi l'originale
        self.samples.append((img_path, label_path,
is_rare))

        # Duplicazione per oversampling (solo se contiene
classi rare)
```

```
        if is_rare and not self.is_val:
            for _ in range(DUPLICATE_FACTOR - 1):
                self.samples.append((img_path,
label_path, is_rare))
```

- **Scheduler**: off
- **Splitting:** on

```
import numpy as np
from PIL import Image
import os
from iterstrat.ml_stratifiers import
MultilabelStratifiedShuffleSplit
import json

# sample male etichettati
excluded_dirs = {
    "0007", "0009", "0090", "0095", "0101", "0104",
"0105",
    "0162", "0163", "0284", "0305", "0306", "0307",
"0308", "0309", "0310", "0311",
    "0351", "0372", "0373", "0376",
    "0498", "0499", "0500", "0501", "0526", "0527",
"0530", "0531", "0542",
    "0564", "0585", "0586", "0587", "0588", "0589",
"0590",
    #seguono nuovo immagini da escludere
    "0000", "0001", "0052"
}

# Elenco di cartelle numeriche valide
all_dirs = sorted([
    d for d in os.listdir(DATASET_DIR)
    if os.path.isdir(os.path.join(DATASET_DIR, d)) and
d.isdigit() and d not in excluded_dirs
])

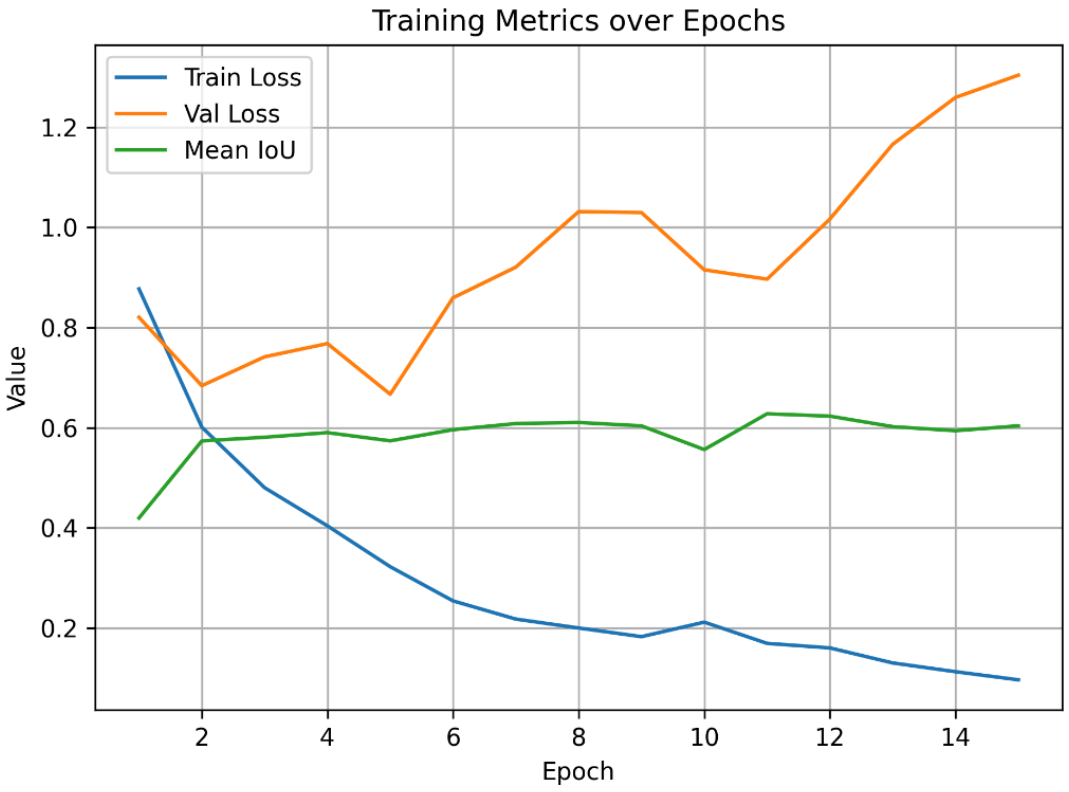
# ------------------------------
#  COSTRUZIONE DELLA MATRICE MULTILABEL
# ------------------------------
def build_presence_matrix(dir_list, n_classes=9):
    """Return np.array (n_samples, n_classes) with
booleans indicating class presence"""
    presence = np.zeros((len(dir_list), n_classes),
dtype=int)

```

```
    for idx, d in enumerate(dir_list):
        label_path = os.path.join(DATASET_DIR, d,
"labels.png")
        lbl = np.array(Image.open(label_path))
        uniq = np.unique(lbl)
        presence[idx, uniq] = 1          # segna le
classi presenti
    return presence

Y = build_presence_matrix(all_dirs, NUM_CLASSES)

# ------------------------------
#  MULTILABEL STRATIFIED SPLIT
# ------------------------------
msss = MultilabelStratifiedShuffleSplit(n_splits=1,
test_size=0.20, random_state=42)
train_idx, val_idx =
next(msss.split(np.zeros(len(all_dirs)), Y))

train_ids = [all_dirs[i] for i in train_idx]
val_ids   = [all_dirs[i] for i in val_idx]

print("Campioni:", len(train_ids), "train |",
len(val_ids), "val")
```

- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy + class weights euristici [1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8].
- **Main classifier Kernel size:** due di dimensione 5

```
# 🔁 Primo layer convoluzionale: kernel 3x3
nn.Conv2d(256, 256, kernel_size=5, padding=2),  #
mantiene dimensione
nn.ReLU(),
# 🔁 Secondo layer convoluzionale: kernel 5x5
nn.Conv2d(256, 256, kernel_size=5, padding=2),  # campo
visivo più ampio
nn.ReLU(),
```

- **ASPP:** stride [3, 7, 3]

```
model = models.segmentation.deeplabv3_resnet101(weights=weights)
# versione custom del classifier
model.classifier = CustomDeepLabHead(2048, NUM_CLASSES)
```

```
class CustomDeepLabHead(nn.Sequential):
    def __init__(self, in_channels, num_classes):
        super().__init__(
            ASPP(in_channels, [3, 7, 3]),  # ASPP con dilatazioni personalizzate
            nn.Conv2d(256, 256, kernel_size=5, padding=2),  # kernel 5x5
            nn.ReLU(),
            nn.Dropout(0.05),
            nn.Conv2d(256, num_classes, kernel_size=1)  # output classi
        )
```

- **Valutazione risultati**:

  Migliore Validation Loss ottenuta all'epoca 5:

  📊 Train Loss: 0.3375 | Val Loss: 0.7033 | Mean IoU: 0.5997

  - IoU classe 0: 0.5237
  - IoU classe 1: 0.6338
  - IoU classe 2: 0.5476
  - IoU classe 3: 0.5924
  - IoU classe 4: 0.6271
  - IoU classe 5: 0.4224
  - IoU classe 6: 0.3016
  - IoU classe 7: 0.7931
  - IoU classe 8: 0.8793

  Migliore mean IoU ottenuta all'epoca 9:

  📊 Train Loss: 0.1911 | Val Loss: 0.9139 | Mean IoU: 0.6395

- ▸ IoU classe 0: 0.5093
- ▸ IoU classe 1: 0.6897
- ▸ IoU classe 2: 0.5568
- ▸ IoU classe 3: 0.6291
- ▸ IoU classe 4: 0.7707
- ▸ IoU classe 5: 0.4354
- ▸ IoU classe 6: 0.2997
- ▸ IoU classe 7: 0.8368
- ▸ IoU classe 8: 0.8975

- **Rappresentazione grafica del training:**



- **Massima memoria RAM occupata durante il training:** 2.7GB

- **Log di training**: `risultati_training\deeplabv3\resnet101\dlrn1010001\training_log_dlrn101012.json`
- **Note / Bug Fix / Annotazioni**:

✳ Esperimento: `deeplabv3_resnet101 - dlrn101013`

- **Architettura**: `deeplabv3plus_resnet101`
- **Backbone Weights**: DeepLabV3_ResNet101_Weights.DEFAULT
- **Input size**: 256x256
- **Batch size**: 4
- **Augmentation**: on

```
# Augmentazione per immagini con classi rare
rare_aug = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.4),
    #A.Rotate(limit=20, p=0.3), potenzialmente inutile
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])

# Augmentazione di base
base_aug = A.Compose([
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])
```

```
def _load_and_augment(self, root_dir):
    sample_dirs = []
    for id in self.sample_ids:
        sample_dirs.append(os.path.join(root_dir, id))

    for dir_path in sample_dirs:
        img_path = os.path.join(dir_path, 'rgb.jpg')
        label_path = os.path.join(dir_path, 'labels.png')
```

```
        label_img = Image.open(label_path)
        is_rare =
SegmentationDataset.contains_rare_classes(label_img,
RARE_CLASSES)

        if not self.allow_augmentation:
            is_rare = False

        # Sempre aggiungi l'originale
        self.samples.append((img_path, label_path,
is_rare))

        # Duplicazione per oversampling (solo se contiene
classi rare)
        if is_rare and not self.is_val:
            for _ in range(DUPLICATE_FACTOR - 1):
                self.samples.append((img_path,
label_path, is_rare))
```

- **Scheduler**: off
- **Splitting:** on

```
import numpy as np
from PIL import Image
import os
from iterstrat.ml_stratifiers import
MultilabelStratifiedShuffleSplit
import json

# sample male etichettati
excluded_dirs = {
    "0007", "0009", "0090", "0095", "0101", "0104",
"0105",
    "0162", "0163", "0284", "0305", "0306", "0307",
"0308", "0309", "0310", "0311",
    "0351", "0372", "0373", "0376",
    "0498", "0499", "0500", "0501", "0526", "0527",
"0530", "0531", "0542",
    "0564", "0585", "0586", "0587", "0588", "0589",
"0590",
    #seguono nuovo immagini da escludere
    "0000", "0001", "0052"
}

# Elenco di cartelle numeriche valide
```

```
all_dirs = sorted([
    d for d in os.listdir(DATASET_DIR)
    if os.path.isdir(os.path.join(DATASET_DIR, d)) and
d.isdigit() and d not in excluded_dirs
])

# ------------------------------
#  COSTRUZIONE DELLA MATRICE MULTILABEL
# ------------------------------
def build_presence_matrix(dir_list, n_classes=9):
    """Return np.array (n_samples, n_classes) with
booleans indicating class presence"""
    presence = np.zeros((len(dir_list), n_classes),
dtype=int)

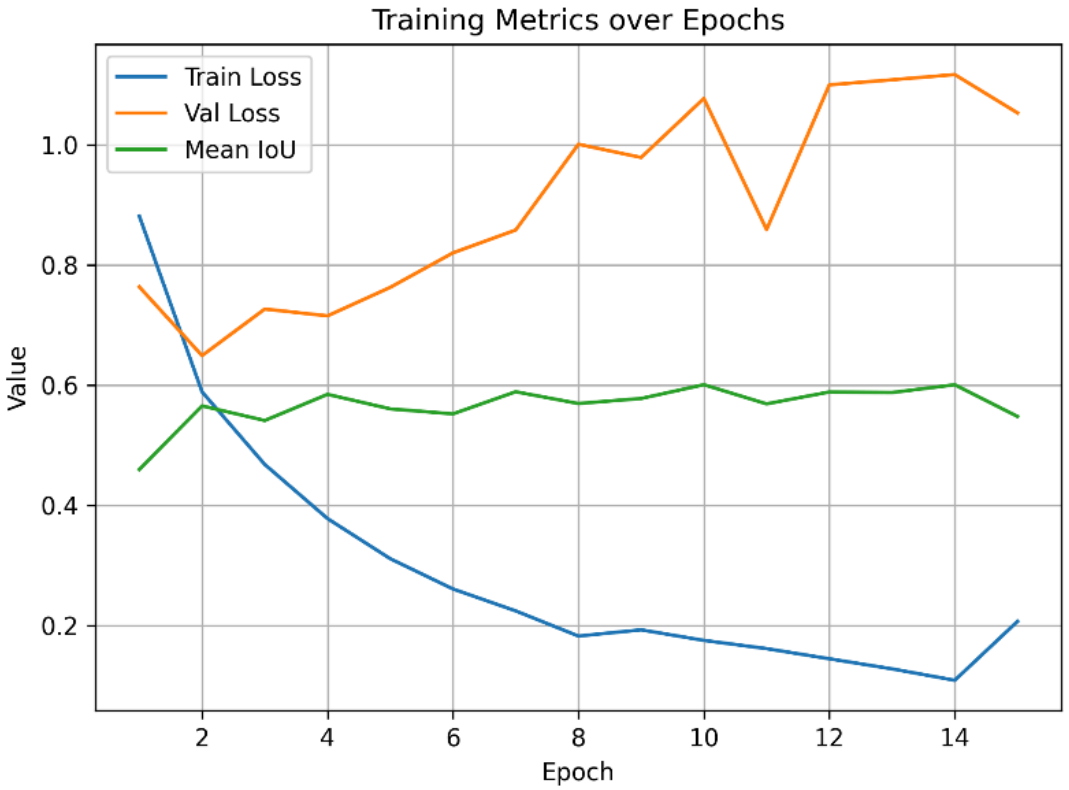
    for idx, d in enumerate(dir_list):
        label_path = os.path.join(DATASET_DIR, d,
"labels.png")
        lbl = np.array(Image.open(label_path))
        uniq = np.unique(lbl)
        presence[idx, uniq] = 1          # segna le
classi presenti
    return presence

Y = build_presence_matrix(all_dirs, NUM_CLASSES)

# ------------------------------
#  MULTILABEL STRATIFIED SPLIT
# ------------------------------
msss = MultilabelStratifiedShuffleSplit(n_splits=1,
test_size=0.20, random_state=42)
train_idx, val_idx =
next(msss.split(np.zeros(len(all_dirs)), Y))

train_ids = [all_dirs[i] for i in train_idx]
val_ids   = [all_dirs[i] for i in val_idx]

print("Campioni:", len(train_ids), "train |",
len(val_ids), "val")
```

- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none

- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy + class weights euristici [1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8].
- **Main classifier Kernel size:** due di dimensione 5

```
# 🔁 Primo layer convoluzionale: kernel 3x3
nn.Conv2d(256, 256, kernel_size=5, padding=2),  # mantiene dimensione
nn.ReLU(),
# 🔁 Secondo layer convoluzionale: kernel 5x5
nn.Conv2d(256, 256, kernel_size=5, padding=2),  # campo visivo più ampio
nn.ReLU(),
```

- **ASPP:** stride [3, 7, 3]

```
model = models.segmentation.deeplabv3_resnet101(weights=weights)
# versione custom del classifier
model.classifier = CustomDeepLabHead(2048, NUM_CLASSES)
```

```

class CustomDeepLabHead(nn.Sequential):
    def __init__(self, in_channels, num_classes):
        super().__init__(
            ASPP(in_channels, [3, 7, 3]),  # ASPP con dilatazioni personalizzate
            nn.Conv2d(256, 256, kernel_size=5, padding=2),  # kernel 5x5
            nn.ReLU(),
            nn.Dropout(0.1),
            nn.Conv2d(256, num_classes, kernel_size=1)  # output classi
        )
```

- **Valutazione risultati**:

  Migliore Validation Loss ottenuta all'epoca 2:

  📊 Train Loss: 0.5890 | Val Loss: 0.6494 | Mean IoU: 0.5656

  - IoU classe 0: 0.4966
  - IoU classe 1: 0.6386
  - IoU classe 2: 0.5564

- ▸ IoU classe 3: 0.6029
- ▸ IoU classe 4: 0.4459
- ▸ IoU classe 5: 0.2749
- ▸ IoU classe 6: 0.3078
- ▸ IoU classe 7: 0.8223
- ▸ IoU classe 8: 0.8765

Migliore mean IoU ottenuta all'epoca 10:

📊 Train Loss: 0.1752 | Val Loss: 1.0776 | Mean IoU: 0.6009

- ▸ IoU classe 0: 0.5302
- ▸ IoU classe 1: 0.6489
- ▸ IoU classe 2: 0.5761
- ▸ IoU classe 3: 0.5994
- ▸ IoU classe 4: 0.5974
- ▸ IoU classe 5: 0.3799
- ▸ IoU classe 6: 0.2832
- ▸ IoU classe 7: 0.8323
- ▸ IoU classe 8: 0.8902

- **Rappresentazione grafica del training:**



- **Massima memoria RAM occupata durante il training:** 2.6GB
- **Log di training**: `risultati_training\deeplabv3\resnet101\dlrn1010001\training_log_dlrn101013.json`
- **Note / Bug Fix / Annotazioni**:

✳ Esperimento: `deeplabv3_resnet101 - dlrn101014`

- **Architettura**: `deeplabv3plus_resnet101`
- **Backbone Weights**: DeepLabV3_ResNet101_Weights.DEFAULT
- **Input size**: 256x256
- **Batch size**: 4
- **Augmentation**: on

```
# Augmentazione per immagini con classi rare
rare_aug = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.4),
    #A.Rotate(limit=20, p=0.3), potenzialmente inutile
```

```
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])

# Augmentazione di base
base_aug = A.Compose([
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])
```

```
def _load_and_augment(self, root_dir):
    sample_dirs = []
    for id in self.sample_ids:
        sample_dirs.append(os.path.join(root_dir, id))

    for dir_path in sample_dirs:
        img_path = os.path.join(dir_path, 'rgb.jpg')
        label_path = os.path.join(dir_path, 'labels.png')

        label_img = Image.open(label_path)
        is_rare = SegmentationDataset.contains_rare_classes(label_img, RARE_CLASSES)

        if not self.allow_augmentation:
            is_rare = False

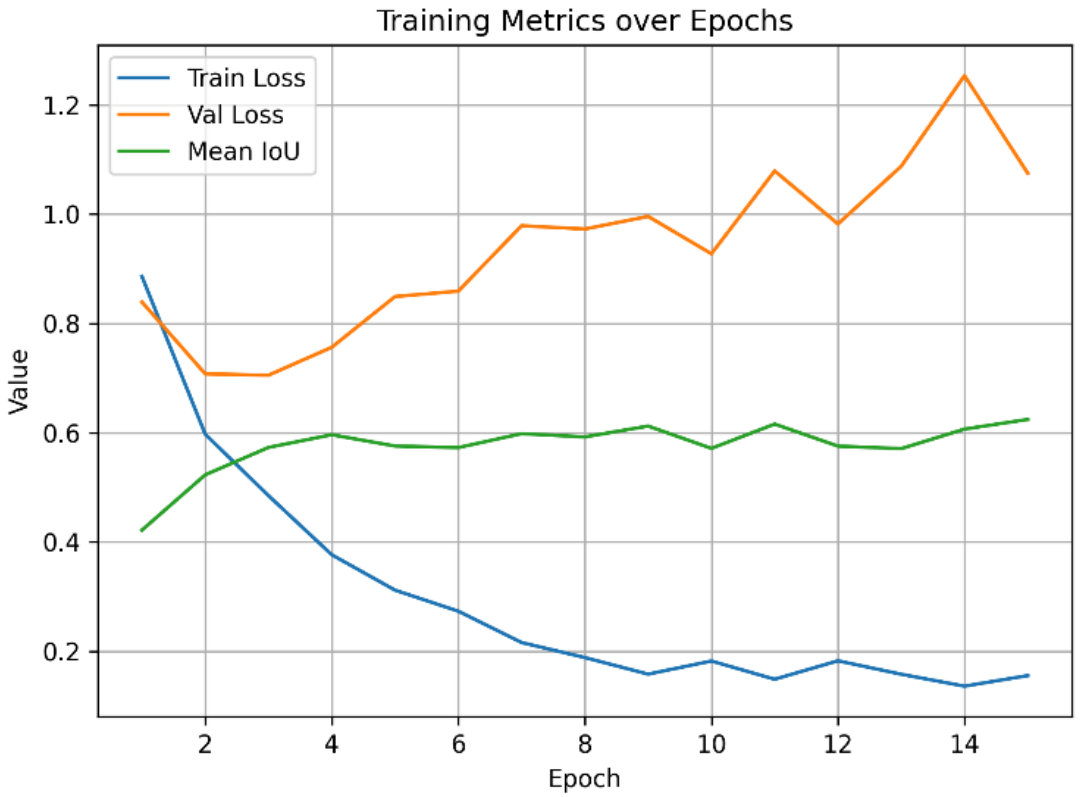
        # Sempre aggiungi l'originale
        self.samples.append((img_path, label_path, is_rare))

        # Duplicazione per oversampling (solo se contiene classi rare)
        if is_rare and not self.is_val:
            for _ in range(DUPLICATE_FACTOR - 1):
                self.samples.append((img_path, label_path, is_rare))
```

- **Scheduler**: off
- **Splitting:** on

```
import numpy as np
```

```
from PIL import Image
import os
from iterstrat.ml_stratifiers import
MultilabelStratifiedShuffleSplit
import json

# sample male etichettati
excluded_dirs = {
    "0007", "0009", "0090", "0095", "0101", "0104",
"0105",
    "0162", "0163", "0284", "0305", "0306", "0307",
"0308", "0309", "0310", "0311",
    "0351", "0372", "0373", "0376",
    "0498", "0499", "0500", "0501", "0526", "0527",
"0530", "0531", "0542",
    "0564", "0585", "0586", "0587", "0588", "0589",
"0590",
    #seguono nuovo immagini da escludere
    "0000", "0001", "0052"
}

# Elenco di cartelle numeriche valide
all_dirs = sorted([
    d for d in os.listdir(DATASET_DIR)
    if os.path.isdir(os.path.join(DATASET_DIR, d)) and
d.isdigit() and d not in excluded_dirs
])

# ------------------------------
#  COSTRUZIONE DELLA MATRICE MULTILABEL
# ------------------------------
def build_presence_matrix(dir_list, n_classes=9):
    """Return np.array (n_samples, n_classes) with
booleans indicating class presence"""
    presence = np.zeros((len(dir_list), n_classes),
dtype=int)

    for idx, d in enumerate(dir_list):
        label_path = os.path.join(DATASET_DIR, d,
"labels.png")
        lbl = np.array(Image.open(label_path))
        uniq = np.unique(lbl)
        presence[idx, uniq] = 1          # segna le
classi presenti
    return presence
```

```
o
o  Y = build_presence_matrix(all_dirs, NUM_CLASSES)
o
o  # ------------------------------
o  #  MULTILABEL STRATIFIED SPLIT
o  # ------------------------------
o  msss = MultilabelStratifiedShuffleSplit(n_splits=1,
   test_size=0.20, random_state=42)
o  train_idx, val_idx =
   next(msss.split(np.zeros(len(all_dirs)), Y))
o
o  train_ids = [all_dirs[i] for i in train_idx]
o  val_ids   = [all_dirs[i] for i in val_idx]
o
o  print("Campioni:", len(train_ids), "train |",
   len(val_ids), "val")
```

- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy + class weights euristici [1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8].
- **Main classifier Kernel size:** due di dimensione 5

```
-  # 🔁 Primo layer convoluzionale: kernel 3x3
-  nn.Conv2d(256, 256, kernel_size=5, padding=2),  #
   mantiene dimensione
-  nn.ReLU(),
-  # 🔁 Secondo layer convoluzionale: kernel 5x5
-  nn.Conv2d(256, 256, kernel_size=5, padding=2),  # campo
   visivo più ampio
-  nn.ReLU(),
```

- **ASPP:** stride [3, 7, 3]

```
o  model =
   models.segmentation.deeplabv3_resnet101(weights=weights)
o  # versione custom del classifier
o  model.classifier = CustomDeepLabHead(2048, NUM_CLASSES)
```

```
-
-  class CustomDeepLabHead(nn.Sequential):
-     def __init__(self, in_channels, num_classes):
-         super().__init__(
```

```
            ASPP(in_channels, [3, 7, 3]),  # ASPP con
dilatazioni personalizzate
            nn.Conv2d(256, 256, kernel_size=5,
padding=2),  # kernel 5x5
            nn.ReLU(),
            nn.Dropout(0.07),
            nn.Conv2d(256, num_classes, kernel_size=1)
# output classi
        )
```

- **Valutazione risultati**:

Migliore Validation Loss ottenuta all'epoca 3:

📊 Train Loss: 0.4850 | Val Loss: 0.7054 | Mean IoU: 0.5732

- IoU classe 0: 0.5110
- IoU classe 1: 0.6382
- IoU classe 2: 0.5217
- IoU classe 3: 0.5848
- IoU classe 4: 0.5679
- IoU classe 5: 0.3303
- IoU classe 6: 0.2764
- IoU classe 7: 0.7965
- IoU classe 8: 0.8696

Migliore mean IoU ottenuta all'epoca 15:

📊 Train Loss: 0.1557 | Val Loss: 1.0749 | Mean IoU: 0.6245

- IoU classe 0: 0.5427
- IoU classe 1: 0.6423
- IoU classe 2: 0.5548
- IoU classe 3: 0.6181
- IoU classe 4: 0.6214

- IoU classe 5: 0.4975
- IoU classe 6: 0.3262
- IoU classe 7: 0.8368
- IoU classe 8: 0.8985

- **Rappresentazione grafica del training:**



- **Massima memoria RAM occupata durante il training:** 2.7GB
- **Log di training**: `risultati_training\deeplabv3\resnet101\dlrn1010001\training_log_dlrn101014.json`
- **Note / Bug Fix / Annotazioni**:

✳ Esperimento: `deeplabv3_resnet101 - dlrn101015`

- **Architettura**: `deeplabv3plus_resnet101`
- **Backbone Weights**: DeepLabV3_ResNet101_Weights.DEFAULT
- **Input size**: 256x256

- **Batch size**: 4
- **Augmentation**: on
    -
```
# Augmentazione per immagini con classi rare
rare_aug = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.4),
    #A.Rotate(limit=20, p=0.3), potenzialmente inutile
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])

# Augmentazione di base
base_aug = A.Compose([
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])
```

```
def _load_and_augment(self, root_dir):
    sample_dirs = []
    for id in self.sample_ids:
        sample_dirs.append(os.path.join(root_dir, id))

    for dir_path in sample_dirs:
        img_path = os.path.join(dir_path, 'rgb.jpg')
        label_path = os.path.join(dir_path, 'labels.png')

        label_img = Image.open(label_path)
        is_rare =
SegmentationDataset.contains_rare_classes(label_img,
RARE_CLASSES)

        if not self.allow_augmentation:
            is_rare = False

        # Sempre aggiungi l'originale
        self.samples.append((img_path, label_path,
is_rare))

        # Duplicazione per oversampling (solo se contiene
classi rare)
```

```
        if is_rare and not self.is_val:
            for _ in range(DUPLICATE_FACTOR - 1):
                self.samples.append((img_path,
label_path, is_rare))
```

- **Scheduler**: off
- **Splitting:** on

```
import numpy as np
from PIL import Image
import os
from iterstrat.ml_stratifiers import
MultilabelStratifiedShuffleSplit
import json

# sample male etichettati
excluded_dirs = {
    "0007", "0009", "0090", "0095", "0101", "0104",
"0105",
    "0162", "0163", "0284", "0305", "0306", "0307",
"0308", "0309", "0310", "0311",
    "0351", "0372", "0373", "0376",
    "0498", "0499", "0500", "0501", "0526", "0527",
"0530", "0531", "0542",
    "0564", "0585", "0586", "0587", "0588", "0589",
"0590",
    #seguono nuovo immagini da escludere
    "0000", "0001", "0052"
}

# Elenco di cartelle numeriche valide
all_dirs = sorted([
    d for d in os.listdir(DATASET_DIR)
    if os.path.isdir(os.path.join(DATASET_DIR, d)) and
d.isdigit() and d not in excluded_dirs
])

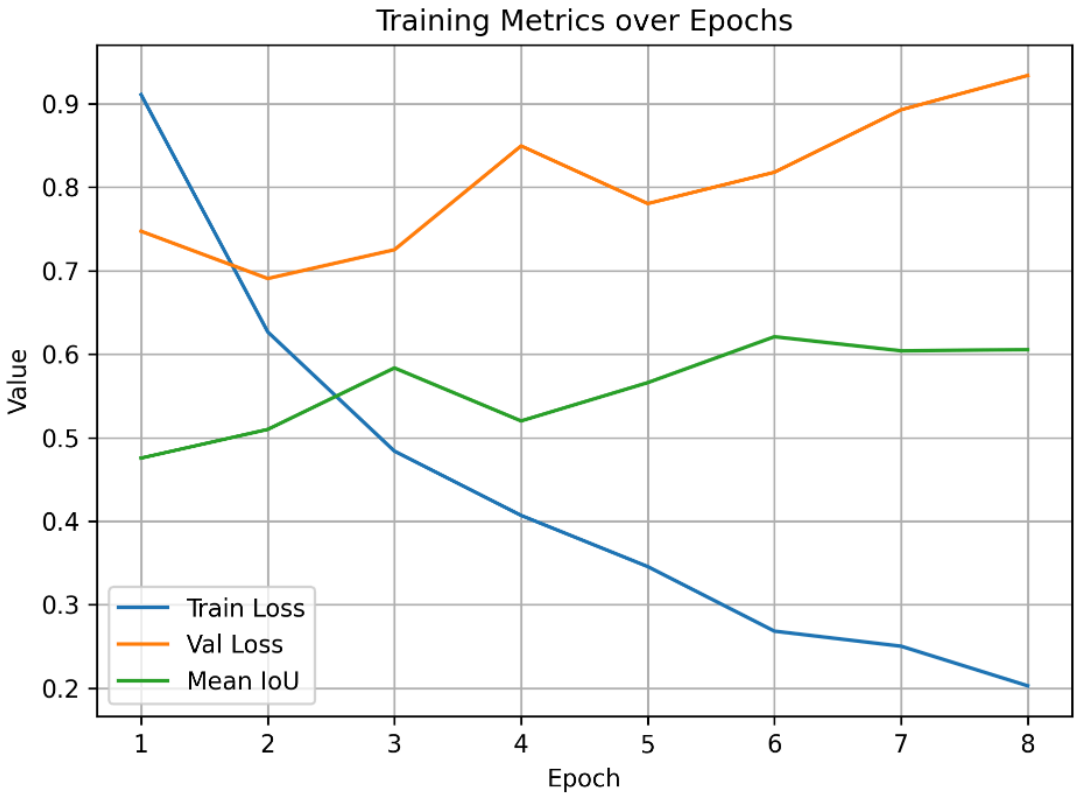
# ------------------------------
#  COSTRUZIONE DELLA MATRICE MULTILABEL
# ------------------------------
def build_presence_matrix(dir_list, n_classes=9):
    """Return np.array (n_samples, n_classes) with
booleans indicating class presence"""
    presence = np.zeros((len(dir_list), n_classes),
dtype=int)

```

```
    for idx, d in enumerate(dir_list):
        label_path = os.path.join(DATASET_DIR, d,
"labels.png")
        lbl = np.array(Image.open(label_path))
        uniq = np.unique(lbl)
        presence[idx, uniq] = 1          # segna le
classi presenti
    return presence

Y = build_presence_matrix(all_dirs, NUM_CLASSES)

# ------------------------------
#  MULTILABEL STRATIFIED SPLIT
# ------------------------------
msss = MultilabelStratifiedShuffleSplit(n_splits=1,
test_size=0.20, random_state=42)
train_idx, val_idx =
next(msss.split(np.zeros(len(all_dirs)), Y))

train_ids = [all_dirs[i] for i in train_idx]
val_ids   = [all_dirs[i] for i in val_idx]

print("Campioni:", len(train_ids), "train |",
len(val_ids), "val")
```

- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy + class weights euristici [1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8].
- **Main classifier Kernel size:** 5, 7

```
# 🔁 Primo layer convoluzionale: kernel 3x3
nn.Conv2d(256, 256, kernel_size=5, padding=2),  #
mantiene dimensione
nn.ReLU(),
# 🔁 Secondo layer convoluzionale: kernel 7x7
nn.Conv2d(256, 256, kernel_size=7, padding=3),  # campo
visivo più ampio
nn.ReLU(),
```

- **ASPP:** stride [3, 7, 3]

```
model = models.segmentation.deeplabv3_resnet101(weights=weights)
# versione custom del classifier
model.classifier = CustomDeepLabHead(2048, NUM_CLASSES)
```

```
class CustomDeepLabHead(nn.Sequential):
    def __init__(self, in_channels, num_classes):
        super().__init__(
            ASPP(in_channels, [3, 7, 3]),  # ASPP con dilatazioni personalizzate
            nn.Conv2d(256, 256, kernel_size=5, padding=2),  # kernel 5x5
            nn.ReLU(),
            nn.Dropout(0),
            nn.Conv2d(256, num_classes, kernel_size=1)  # output classi
        )
```

- **Valutazione risultati**:

  Migliore Validation Loss ottenuta all'epoca 2:

  📊 Train Loss: 0.6266 | Val Loss: 0.6906 | Mean IoU: 0.5099

  - IoU classe 0: 0.4924
  - IoU classe 1: 0.5962
  - IoU classe 2: 0.4811
  - IoU classe 3: 0.5892
  - IoU classe 4: 0.2433
  - IoU classe 5: 0.1892
  - IoU classe 6: 0.3024
  - IoU classe 7: 0.8110
  - IoU classe 8: 0.8667

  Migliore mean IoU ottenuta all'epoca 6:

  📊 Train Loss: 0.2683 | Val Loss: 0.8179 | Mean IoU: 0.6209

- IoU classe 0: 0.5330
- IoU classe 1: 0.6726
- IoU classe 2: 0.5424
- IoU classe 3: 0.6475
- IoU classe 4: 0.7008
- IoU classe 5: 0.3720
- IoU classe 6: 0.3107
- IoU classe 7: 0.8321
- IoU classe 8: 0.8888

- **Rappresentazione grafica del training:**



- **Massima memoria RAM occupata durante il training:** 3.15GB

- **Log di training**: `risultati_training\deeplabv3\resnet101\dlrn1010001\training_log_dlrn101015.json`

- **Note / Bug Fix / Annotazioni**:

✳ Esperimento: `deeplabv3_resnet101 - dlrn101016`

- **Architettura**: `deeplabv3plus_resnet101`
- **Backbone Weights**: DeepLabV3_ResNet101_Weights.DEFAULT
- **Input size**: 256x256
- **Batch size**: 4
- **Augmentation**: on

```
# Augmentazione per immagini con classi rare
rare_aug = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.4),
    #A.Rotate(limit=20, p=0.3), potenzialmente inutile
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])

# Augmentazione di base
base_aug = A.Compose([
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])
```

```
def _load_and_augment(self, root_dir):
    sample_dirs = []
    for id in self.sample_ids:
        sample_dirs.append(os.path.join(root_dir, id))

    for dir_path in sample_dirs:
        img_path = os.path.join(dir_path, 'rgb.jpg')
        label_path = os.path.join(dir_path, 'labels.png')

        label_img = Image.open(label_path)
```

```
        is_rare = SegmentationDataset.contains_rare_classes(label_img, RARE_CLASSES)

        if not self.allow_augmentation:
            is_rare = False

        # Sempre aggiungi l'originale
        self.samples.append((img_path, label_path, is_rare))

        # Duplicazione per oversampling (solo se contiene classi rare)
        if is_rare and not self.is_val:
            for _ in range(DUPLICATE_FACTOR - 1):
                self.samples.append((img_path, label_path, is_rare))
```

- **Scheduler**: off
- **Splitting:** on

```
import numpy as np
from PIL import Image
import os
from iterstrat.ml_stratifiers import MultilabelStratifiedShuffleSplit
import json

# sample male etichettati
excluded_dirs = {
    "0007", "0009", "0090", "0095", "0101", "0104", "0105",
    "0162", "0163", "0284", "0305", "0306", "0307", "0308", "0309", "0310", "0311",
    "0351", "0372", "0373", "0376",
    "0498", "0499", "0500", "0501", "0526", "0527", "0530", "0531", "0542",
    "0564", "0585", "0586", "0587", "0588", "0589", "0590",
    #seguono nuovo immagini da escludere
    "0000", "0001", "0052"
}

# Elenco di cartelle numeriche valide
all_dirs = sorted([
    d for d in os.listdir(DATASET_DIR)
```

```
    if os.path.isdir(os.path.join(DATASET_DIR, d)) and
d.isdigit() and d not in excluded_dirs
])

# ------------------------------
#  COSTRUZIONE DELLA MATRICE MULTILABEL
# ------------------------------
def build_presence_matrix(dir_list, n_classes=9):
    """Return np.array (n_samples, n_classes) with
booleans indicating class presence"""
    presence = np.zeros((len(dir_list), n_classes),
dtype=int)

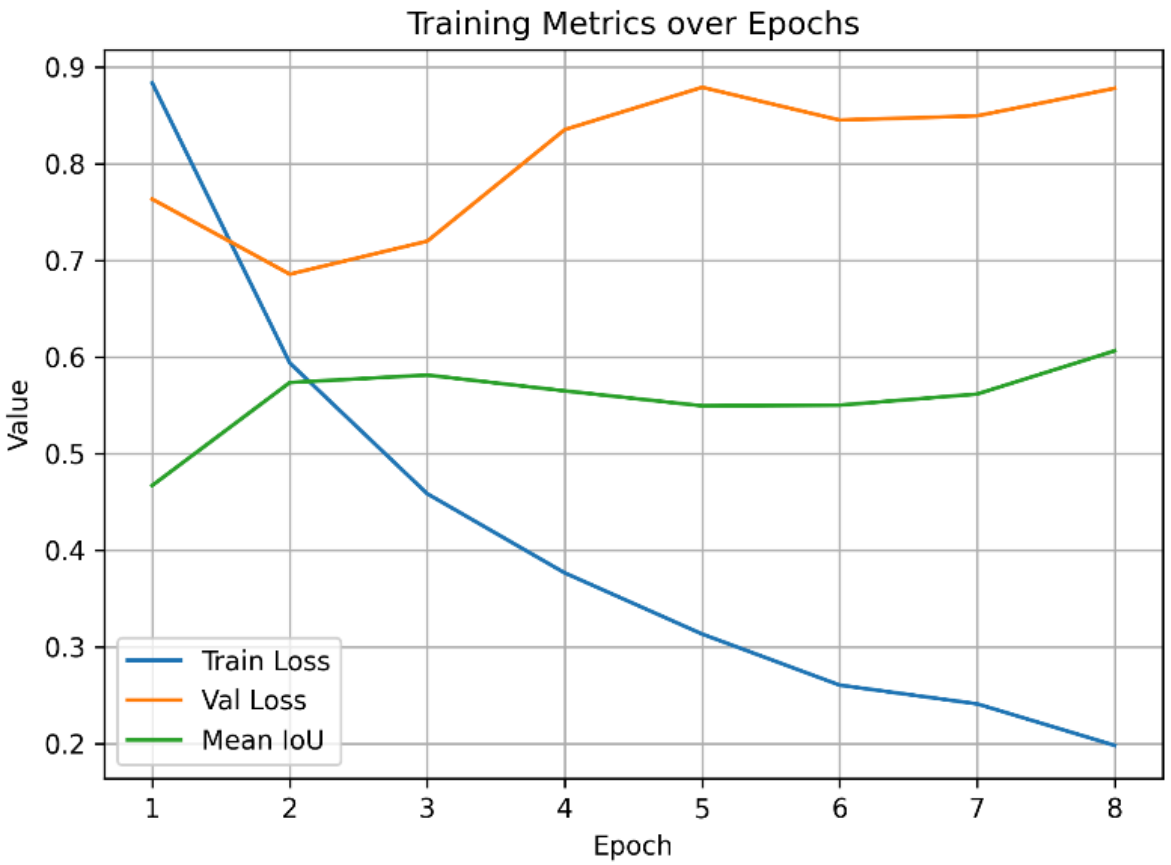
    for idx, d in enumerate(dir_list):
        label_path = os.path.join(DATASET_DIR, d,
"labels.png")
        lbl = np.array(Image.open(label_path))
        uniq = np.unique(lbl)
        presence[idx, uniq] = 1          # segna le
classi presenti
    return presence

Y = build_presence_matrix(all_dirs, NUM_CLASSES)

# ------------------------------
#  MULTILABEL STRATIFIED SPLIT
# ------------------------------
msss = MultilabelStratifiedShuffleSplit(n_splits=1,
test_size=0.20, random_state=42)
train_idx, val_idx =
next(msss.split(np.zeros(len(all_dirs)), Y))

train_ids = [all_dirs[i] for i in train_idx]
val_ids   = [all_dirs[i] for i in val_idx]

print("Campioni:", len(train_ids), "train |",
len(val_ids), "val")
```

- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: off

- **Loss Function / Weighting**: cross-entropy + class weights euristici [1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8].

- **Main classifier Kernel size:** 5, 7

```
# 🔁 Primo layer convoluzionale: kernel 5x5
nn.Conv2d(256, 256, kernel_size=5, padding=2),  # mantiene dimensione
nn.ReLU(),
# 🔁 Secondo layer convoluzionale: kernel 5x5
nn.Conv2d(256, 256, kernel_size=5, padding=2),  # campo visivo più ampio
nn.ReLU(),
```

- **ASPP:** stride [3, 7, 3]

```
model = models.segmentation.deeplabv3_resnet101(weights=weights)
# versione custom del classifier
model.classifier = CustomDeepLabHead(2048, NUM_CLASSES)
```

```

class CustomDeepLabHead(nn.Sequential):
    def __init__(self, in_channels, num_classes):
        super().__init__(
            ASPP(in_channels, [3, 7, 3]),  # ASPP con dilatazioni personalizzate
            nn.Conv2d(256, 256, kernel_size=5, padding=2),  # kernel 5x5
            nn.ReLU(),
            nn.Dropout(0.02),
            nn.Conv2d(256, num_classes, kernel_size=1)  # output classi
        )
```

- **Valutazione risultati**:

  Migliore Validation Loss ottenuta all'epoca 2:

  📊 Train Loss: 0.5938 | Val Loss: 0.6856 | Mean IoU: 0.5735

  - IoU classe 0: 0.4865
  - IoU classe 1: 0.6379
  - IoU classe 2: 0.5090
  - IoU classe 3: 0.5968
  - IoU classe 4: 0.4395

- ▸ IoU classe 5: 0.4210
- ▸ IoU classe 6: 0.3014
- ▸ IoU classe 7: 0.8162
- ▸ IoU classe 8: 0.8660

Migliore mean IoU ottenuta all'epoca 8:

📊 Train Loss: 0.1980 | Val Loss: 0.8780 | Mean IoU: 0.6063

- ▸ IoU classe 0: 0.5290
- ▸ IoU classe 1: 0.6790
- ▸ IoU classe 2: 0.5932
- ▸ IoU classe 3: 0.6270
- ▸ IoU classe 4: 0.5069
- ▸ IoU classe 5: 0.4022
- ▸ IoU classe 6: 0.3390
- ▸ IoU classe 7: 0.8210
- ▸ IoU classe 8: 0.8819

- **Rappresentazione grafica del training:**



- **Massima memoria RAM occupata durante il training:** 2.7GB
- **Log di training**: `risultati_training\deeplabv3\resnet101\dlrn1010001\training_log_dlrn101016.json`
- **Note / Bug Fix / Annotazioni**:

✳ Esperimento: `deeplabv3_resnet101 - dlrn101017`

- **Architettura**: `deeplabv3plus_resnet101`
- **Backbone Weights**: DeepLabV3_ResNet101_Weights.DEFAULT
- **Input size**: 256x256
- **Batch size**: 4
- **Augmentation**: on

```
# Augmentazione per immagini con classi rare
rare_aug = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.4),
```

```
    #A.Rotate(limit=20, p=0.3), potenzialmente inutile
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])

# Augmentazione di base
base_aug = A.Compose([
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])
```

```
def _load_and_augment(self, root_dir):
    sample_dirs = []
    for id in self.sample_ids:
        sample_dirs.append(os.path.join(root_dir, id))

    for dir_path in sample_dirs:
        img_path = os.path.join(dir_path, 'rgb.jpg')
        label_path = os.path.join(dir_path, 'labels.png')

        label_img = Image.open(label_path)
        is_rare =
SegmentationDataset.contains_rare_classes(label_img,
RARE_CLASSES)

        if not self.allow_augmentation:
            is_rare = False

        # Sempre aggiungi l'originale
        self.samples.append((img_path, label_path,
is_rare))

        # Duplicazione per oversampling (solo se contiene
classi rare)
        if is_rare and not self.is_val:
            for _ in range(DUPLICATE_FACTOR - 1):
                self.samples.append((img_path,
label_path, is_rare))
```

- **Scheduler**: off
- **Splitting:** on

```
import numpy as np
from PIL import Image
import os
from iterstrat.ml_stratifiers import
MultilabelStratifiedShuffleSplit
import json

# sample male etichettati
excluded_dirs = {
    "0007", "0009", "0090", "0095", "0101", "0104",
"0105",
    "0162", "0163", "0284", "0305", "0306", "0307",
"0308", "0309", "0310", "0311",
    "0351", "0372", "0373", "0376",
    "0498", "0499", "0500", "0501", "0526", "0527",
"0530", "0531", "0542",
    "0564", "0585", "0586", "0587", "0588", "0589",
"0590",
    #seguono nuovo immagini da escludere
    "0000", "0001", "0052"
}

# Elenco di cartelle numeriche valide
all_dirs = sorted([
    d for d in os.listdir(DATASET_DIR)
    if os.path.isdir(os.path.join(DATASET_DIR, d)) and
d.isdigit() and d not in excluded_dirs
])

# ------------------------------
#  COSTRUZIONE DELLA MATRICE MULTILABEL
# ------------------------------
def build_presence_matrix(dir_list, n_classes=9):
    """Return np.array (n_samples, n_classes) with
booleans indicating class presence"""
    presence = np.zeros((len(dir_list), n_classes),
dtype=int)

    for idx, d in enumerate(dir_list):
        label_path = os.path.join(DATASET_DIR, d,
"labels.png")
        lbl = np.array(Image.open(label_path))
        uniq = np.unique(lbl)
        presence[idx, uniq] = 1          # segna le
classi presenti
```

```
    return presence

Y = build_presence_matrix(all_dirs, NUM_CLASSES)

# ------------------------------
#  MULTILABEL STRATIFIED SPLIT
# ------------------------------
msss = MultilabelStratifiedShuffleSplit(n_splits=1,
test_size=0.20, random_state=42)
train_idx, val_idx =
next(msss.split(np.zeros(len(all_dirs)), Y))

train_ids = [all_dirs[i] for i in train_idx]
val_ids   = [all_dirs[i] for i in val_idx]

print("Campioni:", len(train_ids), "train |",
len(val_ids), "val")
```

- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy + class weights euristici [1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8].
- **Cross entropy loss:**

```
#FOCUSSED CROSS ENTROPY LOSS

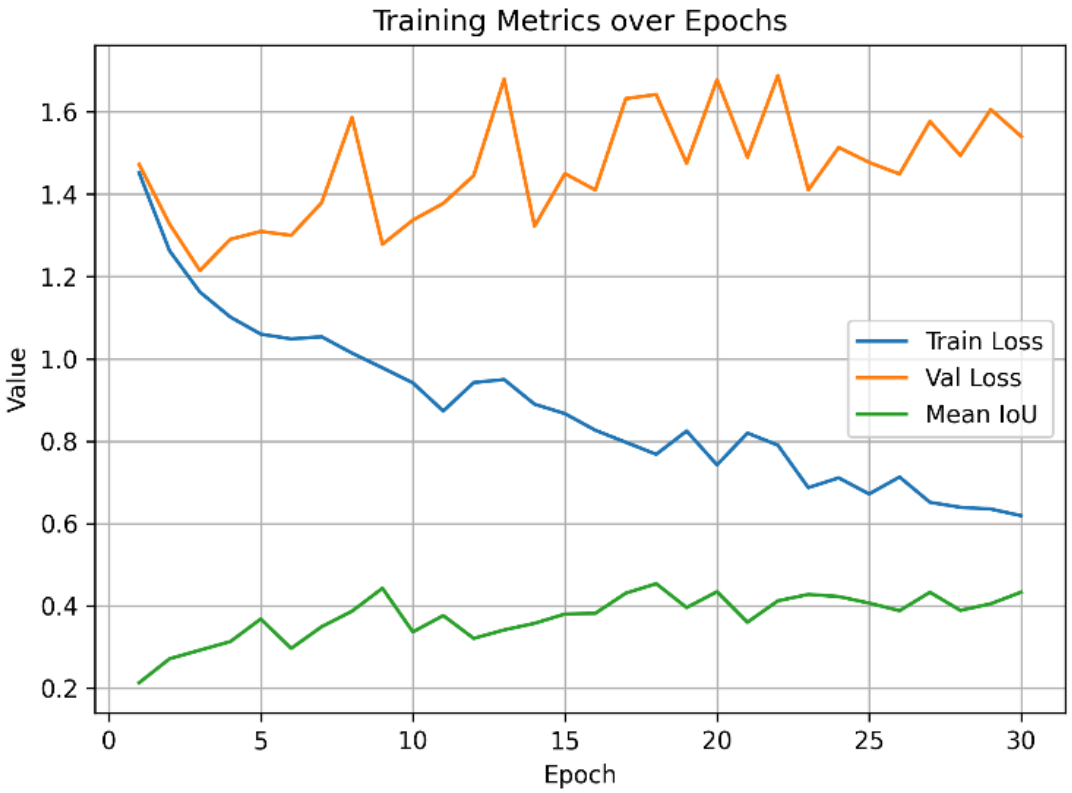

class FocusedCrossEntropyLoss(nn.Module):
    def __init__(self, weight=None, lambda_confusion=2.0,
class_a=1, class_b=3):
        super(FocusedCrossEntropyLoss, self).__init__()
        self.ce_loss = nn.CrossEntropyLoss(weight=weight,
ignore_index=255)
        self.lambda_confusion = lambda_confusion
        self.class_a = class_a
        self.class_b = class_b

    def forward(self, pred, target):
        # pred: (B, C, H, W), logits
        # target: (B, H, W), long
```

```
            ce = self.ce_loss(pred, target)

            probs = F.softmax(pred, dim=1)
            pred_labels = torch.argmax(probs, dim=1)

            confusion_mask = (
                ((target == self.class_a) & (pred_labels ==
self.class_b)) |
                ((target == self.class_b) & (pred_labels ==
self.class_a))
            )

            if confusion_mask.sum() > 0:
                a_probs = probs[:, self.class_a, :,
:][confusion_mask]
                b_probs = probs[:, self.class_b, :,
:][confusion_mask]
                confusion_penalty = (a_probs + b_probs).mean()
            else:
                confusion_penalty = torch.tensor(0.0,
device=pred.device)

            return ce + self.lambda_confusion *
confusion_penalty
```

- **Main classifier Kernel size:** due di dimensione 5

```
# 🔁 Primo layer convoluzionale: kernel 3x3
nn.Conv2d(256, 256, kernel_size=5, padding=2),  #
mantiene dimensione
nn.ReLU(),
# 🔁 Secondo layer convoluzionale: kernel 5x5
nn.Conv2d(256, 256, kernel_size=5, padding=2),  # campo
visivo più ampio
nn.ReLU(),
```

- **ASPP:** stride [3, 7, 3]

```
model =
models.segmentation.deeplabv3_resnet101(weights=weights)
# versione custom del classifier
```

```
model.classifier = CustomDeepLabHead(2048, NUM_CLASSES)
```

```
class CustomDeepLabHead(nn.Sequential):
    def __init__(self, in_channels, num_classes):
        super().__init__(
            ASPP(in_channels, [3, 7, 3]),  # ASPP con dilatazioni personalizzate
            nn.Conv2d(256, 256, kernel_size=5, padding=2),  # kernel 5x5
            nn.ReLU(),
            nn.Dropout(0.05),
            nn.Conv2d(256, num_classes, kernel_size=1)  # output classi
        )
```

- **Valutazione risultati**:
  Migliore Validation Loss ottenuta all'epoca 3:
  📊 Train Loss: 1.1621 | Val Loss: 1.2142 | Mean IoU: 0.2926
  - IoU classe 0: 0.3136
  - IoU classe 1: 0.0633
  - IoU classe 2: 0.2910
  - IoU classe 3: 0.0000
  - IoU classe 4: 0.0290
  - IoU classe 5: 0.0218
  - IoU classe 6: 0.2568
  - IoU classe 7: 0.8052
  - IoU classe 8: 0.8737

  Migliore mean IoU ottenuta all'epoca 18:
  📊 Train Loss: 0.7681 | Val Loss: 1.6418 | Mean IoU: 0.4540
  - IoU classe 0: 0.3728
  - IoU classe 1: 0.4314
  - IoU classe 2: 0.3458
  - IoU classe 3: 0.4196
  - IoU classe 4: 0.0413
  - IoU classe 5: 0.4209
  - IoU classe 6: 0.2676
  - IoU classe 7: 0.8149
  - IoU classe 8: 0.8900

- **Rappresentazione grafica del training:**



- **Massima memoria RAM occupata durante il training:** 3.6GB
- **Log di training**: `risultati_training\deeplabv3\resnet101\dlrn1010001\training_log_dlrn101017.json`
- **Note / Bug Fix / Annotazioni**:

✳ Esperimento: `deeplabv3_resnet101 - dlrn101018`

- **Architettura**: `deeplabv3plus_resnet101`
- **Backbone Weights**: DeepLabV3_ResNet101_Weights.DEFAULT
- **Input size**: 256x256
- **Batch size**: 4
- **Augmentation**: on

```
# Augmentazione per immagini con classi rare
rare_aug = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.4),
    #A.Rotate(limit=20, p=0.3), potenzialmente inutile
```

```
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])

# Augmentazione di base
base_aug = A.Compose([
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])
```

```
def _load_and_augment(self, root_dir):
    sample_dirs = []
    for id in self.sample_ids:
        sample_dirs.append(os.path.join(root_dir, id))

    for dir_path in sample_dirs:
        img_path = os.path.join(dir_path, 'rgb.jpg')
        label_path = os.path.join(dir_path, 'labels.png')

        label_img = Image.open(label_path)
        is_rare = SegmentationDataset.contains_rare_classes(label_img, RARE_CLASSES)

        if not self.allow_augmentation:
            is_rare = False

        # Sempre aggiungi l'originale
        self.samples.append((img_path, label_path, is_rare))

        # Duplicazione per oversampling (solo se contiene classi rare)
        if is_rare and not self.is_val:
            for _ in range(DUPLICATE_FACTOR - 1):
                self.samples.append((img_path, label_path, is_rare))
```

- **Scheduler**: off
- **Splitting:** on

```
import numpy as np
```

```
from PIL import Image
import os
from iterstrat.ml_stratifiers import
MultilabelStratifiedShuffleSplit
import json

# sample male etichettati
excluded_dirs = {
    "0007", "0009", "0090", "0095", "0101", "0104",
"0105",
    "0162", "0163", "0284", "0305", "0306", "0307",
"0308", "0309", "0310", "0311",
    "0351", "0372", "0373", "0376",
    "0498", "0499", "0500", "0501", "0526", "0527",
"0530", "0531", "0542",
    "0564", "0585", "0586", "0587", "0588", "0589",
"0590",
    #seguono nuovo immagini da escludere
    "0000", "0001", "0052"
}

# Elenco di cartelle numeriche valide
all_dirs = sorted([
    d for d in os.listdir(DATASET_DIR)
    if os.path.isdir(os.path.join(DATASET_DIR, d)) and
d.isdigit() and d not in excluded_dirs
])

# ------------------------------
#  COSTRUZIONE DELLA MATRICE MULTILABEL
# ------------------------------
def build_presence_matrix(dir_list, n_classes=9):
    """Return np.array (n_samples, n_classes) with
booleans indicating class presence"""
    presence = np.zeros((len(dir_list), n_classes),
dtype=int)

    for idx, d in enumerate(dir_list):
        label_path = os.path.join(DATASET_DIR, d,
"labels.png")
        lbl = np.array(Image.open(label_path))
        uniq = np.unique(lbl)
        presence[idx, uniq] = 1          # segna le
classi presenti
    return presence
```

```
Y = build_presence_matrix(all_dirs, NUM_CLASSES)

# ------------------------------
#  MULTILABEL STRATIFIED SPLIT
# ------------------------------
msss = MultilabelStratifiedShuffleSplit(n_splits=1,
test_size=0.20, random_state=42)
train_idx, val_idx =
next(msss.split(np.zeros(len(all_dirs)), Y))

train_ids = [all_dirs[i] for i in train_idx]
val_ids   = [all_dirs[i] for i in val_idx]

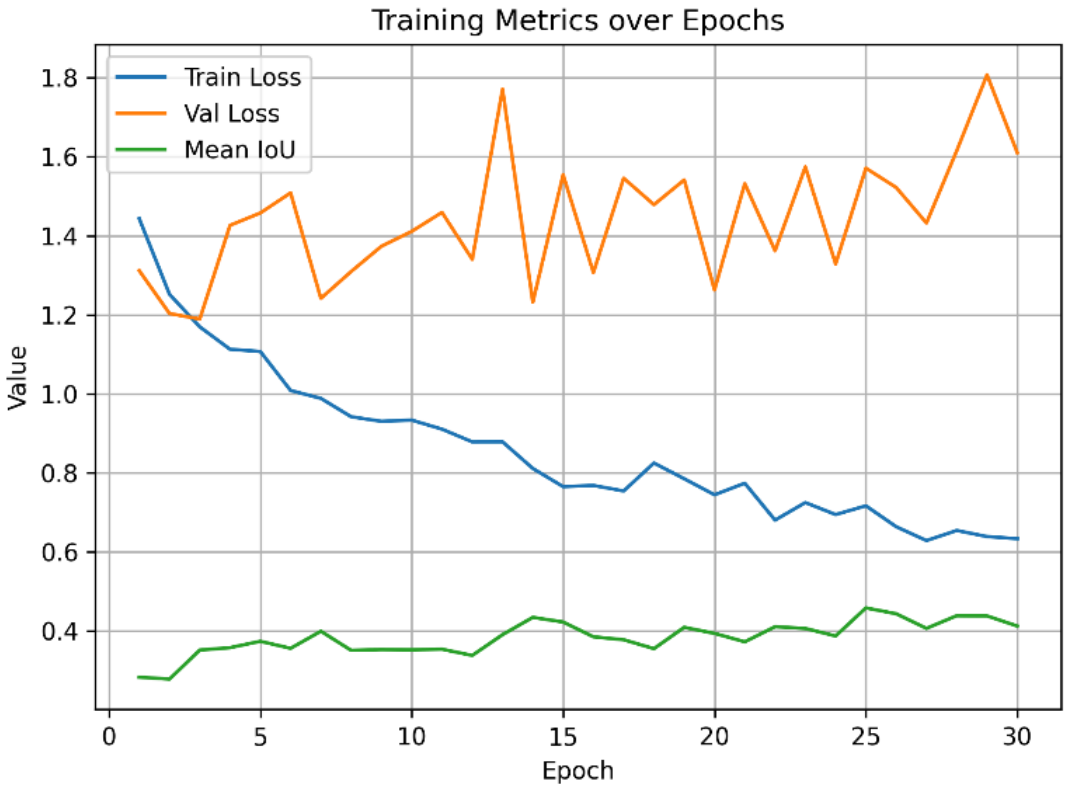
print("Campioni:", len(train_ids), "train |",
len(val_ids), "val")
```

- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy + class weights euristici [1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8].
- **Cross penalty loss:**

```
#FOCUSSED CROSS ENTROPY LOSS


class FocusedCrossEntropyLoss(nn.Module):
    def __init__(self, weight=None, lambda_confusion=0.5,
class_a=1, class_b=3):
        super(FocusedCrossEntropyLoss, self).__init__()
        self.ce_loss = nn.CrossEntropyLoss(weight=weight,
ignore_index=255)
        self.lambda_confusion = lambda_confusion
        self.class_a = class_a
        self.class_b = class_b

    def forward(self, pred, target):
        # pred: (B, C, H, W), logits
        # target: (B, H, W), long
        ce = self.ce_loss(pred, target)
```

```
            probs = F.softmax(pred, dim=1)
            pred_labels = torch.argmax(probs, dim=1)

            confusion_mask = (
                ((target == self.class_a) & (pred_labels ==
self.class_b)) |
                ((target == self.class_b) & (pred_labels ==
self.class_a))
            )

            if confusion_mask.sum() > 0:
                a_probs = probs[:, self.class_a, :,
:][confusion_mask]
                b_probs = probs[:, self.class_b, :,
:][confusion_mask]
                confusion_penalty = (a_probs + b_probs).mean()
            else:
                confusion_penalty = torch.tensor(0.0,
device=pred.device)

            return ce + self.lambda_confusion *
confusion_penalty
```

- **Main classifier Kernel size:** due di dimensione 5

```
# 🔁 Primo layer convoluzionale: kernel 3x3
nn.Conv2d(256, 256, kernel_size=5, padding=2),  #
mantiene dimensione
nn.ReLU(),
# 🔁 Secondo layer convoluzionale: kernel 5x5
nn.Conv2d(256, 256, kernel_size=5, padding=2),  # campo
visivo più ampio
nn.ReLU(),
```

- **ASPP:** stride [3, 7, 3]

```
model =
models.segmentation.deeplabv3_resnet101(weights=weights)
# versione custom del classifier
model.classifier = CustomDeepLabHead(2048, NUM_CLASSES)
```

```
class CustomDeepLabHead(nn.Sequential):
    def __init__(self, in_channels, num_classes):
        super().__init__(
            ASPP(in_channels, [3, 7, 3]),  # ASPP con
dilatazioni personalizzate
            nn.Conv2d(256, 256, kernel_size=5,
padding=2),  # kernel 5x5
            nn.ReLU(),
            nn.Dropout(0.05),
            nn.Conv2d(256, num_classes, kernel_size=1)
# output classi
        )
```

- **Valutazione risultati**:
  📊 Train Loss: 1.1693 | Val Loss: 1.1890 | Mean IoU: 0.3508
  - IoU classe 0: 0.3534
  - IoU classe 1: 0.0094
  - IoU classe 2: 0.3062
  - IoU classe 3: 0.3140
  - IoU classe 4: 0.0145
  - IoU classe 5: 0.2450
  - IoU classe 6: 0.2446
  - IoU classe 7: 0.8077
  - IoU classe 8: 0.8651

  Migliore mean IoU ottenuta all'epoca 25:
  📊 Train Loss: 0.7161 | Val Loss: 1.5710 | Mean IoU: 0.4576
  - IoU classe 0: 0.3657
  - IoU classe 1: 0.4367
  - IoU classe 2: 0.4148
  - IoU classe 3: 0.5200
  - IoU classe 4: 0.0254
  - IoU classe 5: 0.3010
  - IoU classe 6: 0.2411
  - IoU classe 7: 0.8300
  - IoU classe 8: 0.8915

- **Rappresentazione grafica del training:**



- **Massima memoria RAM occupata durante il training:** 3.6GB
- **Log di training**: `risultati_training\deeplabv3\resnet101\dlrn1010001\training_log_dlrn101018.json`
- **Note / Bug Fix / Annotazioni**:

✳ Esperimento: `deeplabv3_resnet101 - dlrn101019`

- **Architettura**: `deeplabv3plus_resnet101`
- **Backbone Weights**: DeepLabV3_ResNet101_Weights.DEFAULT
- **Input size**: 256x256
- **Batch size**: 4
- **Augmentation**: on

```
# Augmentazione per immagini con classi rare
rare_aug = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.4),
    #A.Rotate(limit=20, p=0.3), potenzialmente inutile
```

```
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])

# Augmentazione di base
base_aug = A.Compose([
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])
```

```
def _load_and_augment(self, root_dir):
    sample_dirs = []
    for id in self.sample_ids:
        sample_dirs.append(os.path.join(root_dir, id))

    for dir_path in sample_dirs:
        img_path = os.path.join(dir_path, 'rgb.jpg')
        label_path = os.path.join(dir_path, 'labels.png')

        label_img = Image.open(label_path)
        is_rare = SegmentationDataset.contains_rare_classes(label_img, RARE_CLASSES)

        if not self.allow_augmentation:
            is_rare = False

        # Sempre aggiungi l'originale
        self.samples.append((img_path, label_path, is_rare))

        # Duplicazione per oversampling (solo se contiene classi rare)
        if is_rare and not self.is_val:
            for _ in range(DUPLICATE_FACTOR - 1):
                self.samples.append((img_path, label_path, is_rare))
```

- **Scheduler**: off
- **Splitting:** on

```
import numpy as np
```

```
from PIL import Image
import os
from iterstrat.ml_stratifiers import
MultilabelStratifiedShuffleSplit
import json

# sample male etichettati
excluded_dirs = {
    "0007", "0009", "0090", "0095", "0101", "0104",
"0105",
    "0162", "0163", "0284", "0305", "0306", "0307",
"0308", "0309", "0310", "0311",
    "0351", "0372", "0373", "0376",
    "0498", "0499", "0500", "0501", "0526", "0527",
"0530", "0531", "0542",
    "0564", "0585", "0586", "0587", "0588", "0589",
"0590",
    #seguono nuovo immagini da escludere
    "0000", "0001", "0052"
}

# Elenco di cartelle numeriche valide
all_dirs = sorted([
    d for d in os.listdir(DATASET_DIR)
    if os.path.isdir(os.path.join(DATASET_DIR, d)) and
d.isdigit() and d not in excluded_dirs
])

# ------------------------------
#  COSTRUZIONE DELLA MATRICE MULTILABEL
# ------------------------------
def build_presence_matrix(dir_list, n_classes=9):
    """Return np.array (n_samples, n_classes) with
booleans indicating class presence"""
    presence = np.zeros((len(dir_list), n_classes),
dtype=int)

    for idx, d in enumerate(dir_list):
        label_path = os.path.join(DATASET_DIR, d,
"labels.png")
        lbl = np.array(Image.open(label_path))
        uniq = np.unique(lbl)
        presence[idx, uniq] = 1          # segna le
classi presenti
    return presence
```

```
Y = build_presence_matrix(all_dirs, NUM_CLASSES)

# ------------------------------
#  MULTILABEL STRATIFIED SPLIT
# ------------------------------
msss = MultilabelStratifiedShuffleSplit(n_splits=1,
test_size=0.20, random_state=42)
train_idx, val_idx =
next(msss.split(np.zeros(len(all_dirs)), Y))

train_ids = [all_dirs[i] for i in train_idx]
val_ids   = [all_dirs[i] for i in val_idx]

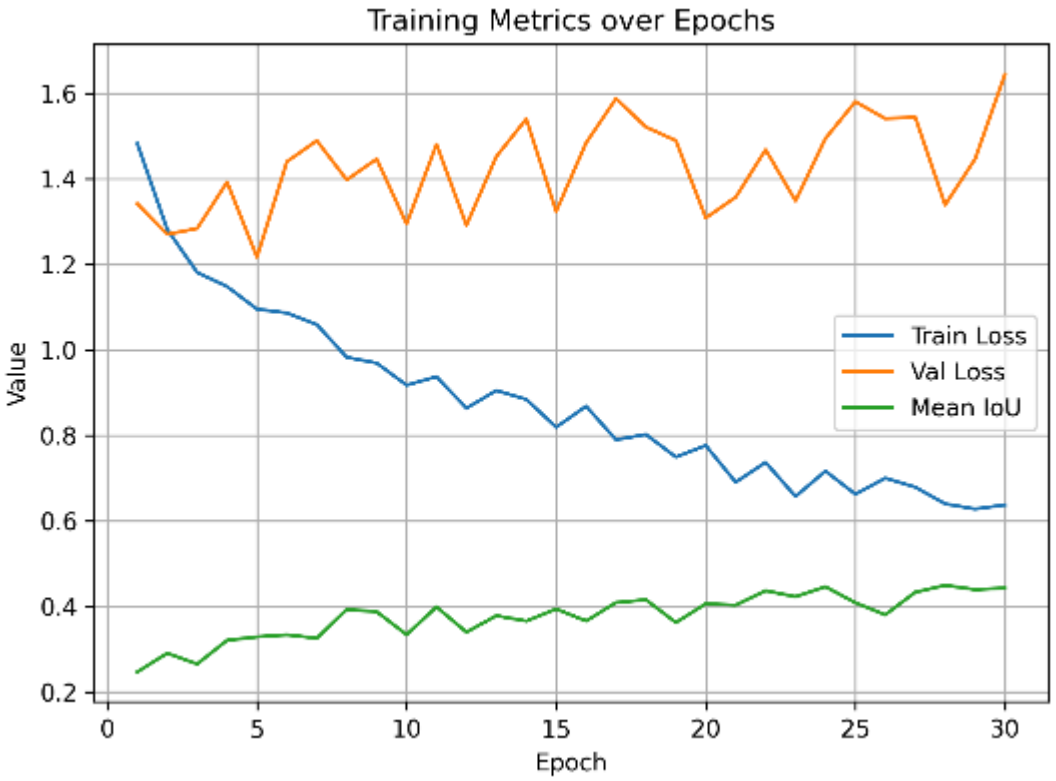
print("Campioni:", len(train_ids), "train |",
len(val_ids), "val")
```

- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy + class weights euristici [1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8].
- **Cross penalty loss:**

```
#FOCUSSED CROSS ENTROPY LOSS


class FocusedCrossEntropyLoss(nn.Module):
    def __init__(self, weight=None, lambda_confusion=0.1,
class_a=1, class_b=3):
        super(FocusedCrossEntropyLoss, self).__init__()
        self.ce_loss = nn.CrossEntropyLoss(weight=weight,
ignore_index=255)
        self.lambda_confusion = lambda_confusion
        self.class_a = class_a
        self.class_b = class_b

    def forward(self, pred, target):
        # pred: (B, C, H, W), logits
        # target: (B, H, W), long
        ce = self.ce_loss(pred, target)
```

```
            probs = F.softmax(pred, dim=1)
            pred_labels = torch.argmax(probs, dim=1)

            confusion_mask = (
                ((target == self.class_a) & (pred_labels ==
self.class_b)) |
                ((target == self.class_b) & (pred_labels ==
self.class_a))
            )

            if confusion_mask.sum() > 0:
                a_probs = probs[:, self.class_a, :,
:][confusion_mask]
                b_probs = probs[:, self.class_b, :,
:][confusion_mask]
                confusion_penalty = (a_probs + b_probs).mean()
            else:
                confusion_penalty = torch.tensor(0.0,
device=pred.device)

            return ce + self.lambda_confusion *
confusion_penalty
```

- **Main classifier Kernel size:** due di dimensione 5

```
# 🔁 Primo layer convoluzionale: kernel 3x3
nn.Conv2d(256, 256, kernel_size=5, padding=2),  #
mantiene dimensione
nn.ReLU(),
# 🔁 Secondo layer convoluzionale: kernel 5x5
nn.Conv2d(256, 256, kernel_size=5, padding=2),  # campo
visivo più ampio
nn.ReLU(),
```

- **ASPP:** stride [3, 7, 3]

```
model =
models.segmentation.deeplabv3_resnet101(weights=weights)
# versione custom del classifier
model.classifier = CustomDeepLabHead(2048, NUM_CLASSES)
```

```
class CustomDeepLabHead(nn.Sequential):
    def __init__(self, in_channels, num_classes):
        super().__init__(
            ASPP(in_channels, [3, 7, 3]),  # ASPP con
dilatazioni personalizzate
            nn.Conv2d(256, 256, kernel_size=5,
padding=2),  # kernel 5x5
            nn.ReLU(),
            nn.Dropout(0.05),
            nn.Conv2d(256, num_classes, kernel_size=1)
# output classi
        )
```

- **Valutazione risultati**:
  📊 Train Loss: 1.0955 | Val Loss: 1.2171 | Mean IoU: 0.3295
  - IoU classe 0: 0.3609
  - IoU classe 1: 0.1535
  - IoU classe 2: 0.2541
  - IoU classe 3: 0.0047
  - IoU classe 4: 0.0114
  - IoU classe 5: 0.3014
  - IoU classe 6: 0.2420
  - IoU classe 7: 0.7911
  - IoU classe 8: 0.8778

  Migliore mean IoU ottenuta all'epoca 28:
  📊 Train Loss: 0.6404 | Val Loss: 1.3389 | Mean IoU: 0.4501
  - IoU classe 0: 0.2255
  - IoU classe 1: 0.4809
  - IoU classe 2: 0.4056
  - IoU classe 3: 0.4222
  - IoU classe 4: 0.0484
  - IoU classe 5: 0.2713
  - IoU classe 6: 0.2620
  - IoU classe 7: 0.8187
  - IoU classe 8: 0.8914

- **Rappresentazione grafica del training:**



- **Massima memoria RAM occupata durante il training:** 4.6GB
- **Log di training**: `risultati_training\deeplabv3\resnet101\dlrn1010001\training_log_dlrn101019.json`
- **Note / Bug Fix / Annotazioni**:

✳ Esperimento: `deeplabv3_resnet101 - dlrn101020`

- **Architettura**: `deeplabv3plus_resnet101`
- **Backbone Weights**: DeepLabV3_ResNet101_Weights.DEFAULT
- **Input size**: 256x256
- **Batch size**: 4
- **Augmentation**: on

```
# Augmentazione per immagini con classi rare
rare_aug = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.4),
    #A.Rotate(limit=20, p=0.3), potenzialmente inutile
```

```
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])

# Augmentazione mirata per immagini che contengono sia classi 1 che 3
confused_aug = A.Compose([
    A.OneOf([
        A.MotionBlur(p=0.5),
        A.GaussianBlur(p=0.5),
        A.MedianBlur(blur_limit=5, p=0.5),
        A.GaussNoise(var_limit=(10.0, 50.0), p=0.5),
        A.Sharpen(p=0.5),
    ], p=1.0),
    A.RandomBrightnessContrast(p=0.3),
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])

# Augmentazione di base
base_aug = A.Compose([
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])
```

```
def _load_and_augment(self, root_dir):
    sample_dirs = []
    for id in self.sample_ids:
        sample_dirs.append(os.path.join(root_dir, id))

    for dir_path in sample_dirs:
        img_path = os.path.join(dir_path, 'rgb.jpg')
        label_path = os.path.join(dir_path, 'labels.png')

        label_img = Image.open(label_path)
        is_rare = SegmentationDataset.contains_rare_classes(label_img, RARE_CLASSES)
```

```
        if not self.allow_augmentation:
            is_rare = False

        # Sempre aggiungi l'originale
        self.samples.append((img_path, label_path,
is_rare))

        # Duplicazione per oversampling (solo se contiene
classi rare)
        if is_rare and not self.is_val:
            for _ in range(DUPLICATE_FACTOR - 1):
                self.samples.append((img_path,
label_path, is_rare))

if self.is_val:
    aug = SegmentationDataset.base_aug
else:
    # Carica la maschera come immagine PIL per
controllare le classi
    label_img = Image.open(label_path)
    is_confused =
SegmentationDataset.contains_confused_classes(label_img)

    if is_confused:
        aug = SegmentationDataset.confused_aug
    elif is_rare:
        aug = SegmentationDataset.rare_aug
    else:
        aug = SegmentationDataset.base_aug
```

- **Scheduler**: off
- **Splitting:** on

```
import numpy as np
from PIL import Image
import os
from iterstrat.ml_stratifiers import
MultilabelStratifiedShuffleSplit
import json

# sample male etichettati
excluded_dirs = {
   "0007", "0009", "0090", "0095", "0101", "0104",
"0105",
   "0162", "0163", "0284", "0305", "0306", "0307",
"0308", "0309", "0310", "0311",
```

```
    "0351", "0372", "0373", "0376",
    "0498", "0499", "0500", "0501", "0526", "0527",
"0530", "0531", "0542",
    "0564", "0585", "0586", "0587", "0588", "0589",
"0590",
    #seguono nuovo immagini da escludere
    "0000", "0001", "0052"
}

# Elenco di cartelle numeriche valide
all_dirs = sorted([
    d for d in os.listdir(DATASET_DIR)
    if os.path.isdir(os.path.join(DATASET_DIR, d)) and
d.isdigit() and d not in excluded_dirs
])

# ------------------------------
#  COSTRUZIONE DELLA MATRICE MULTILABEL
# ------------------------------
def build_presence_matrix(dir_list, n_classes=9):
    """Return np.array (n_samples, n_classes) with
booleans indicating class presence"""
    presence = np.zeros((len(dir_list), n_classes),
dtype=int)

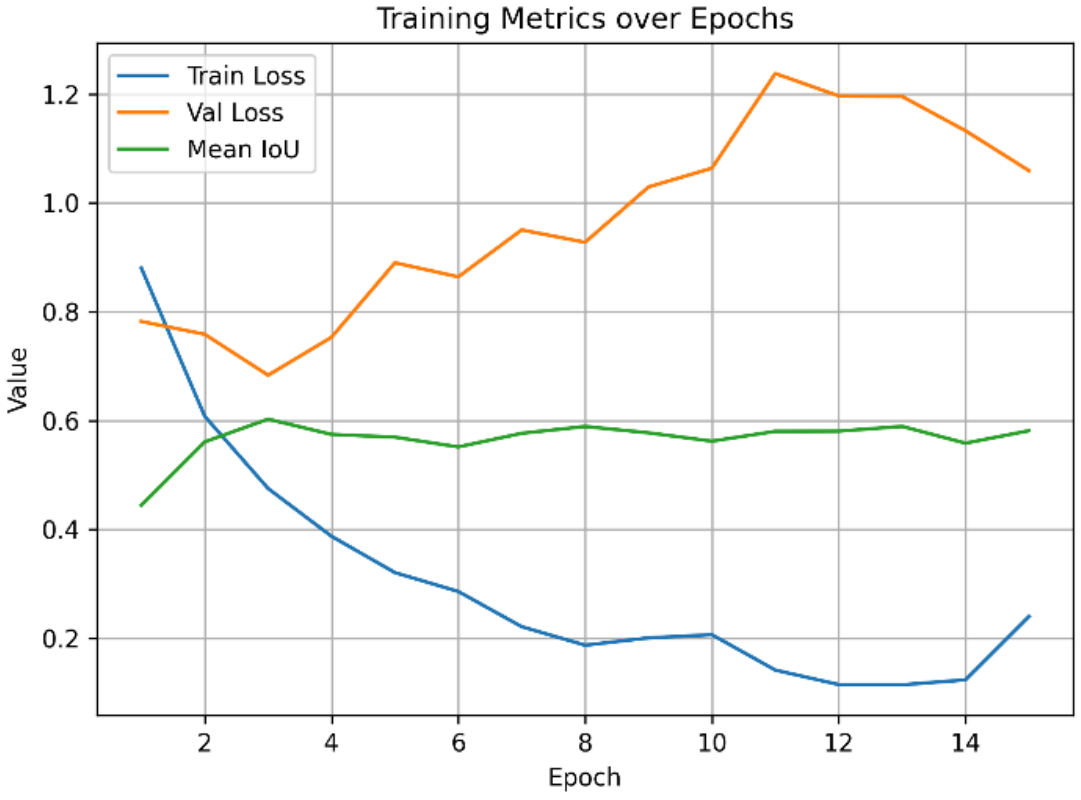
    for idx, d in enumerate(dir_list):
        label_path = os.path.join(DATASET_DIR, d,
"labels.png")
        lbl = np.array(Image.open(label_path))
        uniq = np.unique(lbl)
        presence[idx, uniq] = 1          # segna le
classi presenti
    return presence

Y = build_presence_matrix(all_dirs, NUM_CLASSES)

# ------------------------------
#  MULTILABEL STRATIFIED SPLIT
# ------------------------------
msss = MultilabelStratifiedShuffleSplit(n_splits=1,
test_size=0.20, random_state=42)
train_idx, val_idx =
next(msss.split(np.zeros(len(all_dirs)), Y))

train_ids = [all_dirs[i] for i in train_idx]
```

```
val_ids   = [all_dirs[i] for i in val_idx]

print("Campioni:", len(train_ids), "train |",
len(val_ids), "val")
```

- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy + class weights euristici [1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8].
- **Main classifier Kernel size:** due di dimensione 5

```
# 🔁 Primo layer convoluzionale: kernel 3x3
nn.Conv2d(256, 256, kernel_size=5, padding=2),  #
mantiene dimensione
nn.ReLU(),
# 🔁 Secondo layer convoluzionale: kernel 5x5
nn.Conv2d(256, 256, kernel_size=5, padding=2),  # campo
visivo più ampio
nn.ReLU(),
```

- **ASPP:** stride [3, 7, 3]

```
model =
models.segmentation.deeplabv3_resnet101(weights=weights)
# versione custom del classifier
model.classifier = CustomDeepLabHead(2048, NUM_CLASSES)
```

```

class CustomDeepLabHead(nn.Sequential):
    def __init__(self, in_channels, num_classes):
        super().__init__(
            ASPP(in_channels, [3, 7, 3]),  # ASPP con
dilatazioni personalizzate
            nn.Conv2d(256, 256, kernel_size=5,
padding=2),  # kernel 5x5
            nn.ReLU(),
            nn.Dropout(0.05),
            nn.Conv2d(256, num_classes, kernel_size=1)
# output classi
        )
```

- **Valutazione risultati**:

Migliore Validation Loss ottenuta all'epoca 3:
📊 Train Loss: 0.4752 | Val Loss: 0.6834 | Mean IoU: 0.6026
- IoU classe 0: 0.5033
- IoU classe 1: 0.6373
- IoU classe 2: 0.4822
- IoU classe 3: 0.5902
- IoU classe 4: 0.6699
- IoU classe 5: 0.4330
- IoU classe 6: 0.3208
- IoU classe 7: 0.8155
- IoU classe 8: 0.8722

Migliore mean IoU ottenuta all'epoca 3:
📊 Train Loss: 0.4752 | Val Loss: 0.6834 | Mean IoU: 0.6026
- IoU classe 0: 0.5033
- IoU classe 1: 0.6373
- IoU classe 2: 0.4822
- IoU classe 3: 0.5902
- IoU classe 4: 0.6699
- IoU classe 5: 0.4330
- IoU classe 6: 0.3208
- IoU classe 7: 0.8155
- IoU classe 8: 0.8722

- **Rappresentazione grafica del training:**



- **Massima memoria RAM occupata durante il training:** 4GB
- **Log di training**: `risultati_training\deeplabv3\resnet101\dlrn1010001\training_log_dlrn101020.json`
- **Note / Bug Fix / Annotazioni**:

✳ Esperimento: `deeplabv3_resnet101 - dlrn101021`

- **Architettura**: `deeplabv3plus_resnet101`
- **Backbone Weights**: DeepLabV3_ResNet101_Weights.DEFAULT
- **Input size**: 256x256
- **Batch size**: 4
- **Augmentation**: on

```
# Augmentazione per immagini con classi rare
rare_aug = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.4),
    #A.Rotate(limit=20, p=0.3), potenzialmente inutile
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])

# Augmentazione mirata per immagini che contengono sia classi 1 che 3
confused_aug = A.Compose([
    A.OneOf([
        A.MotionBlur(p=0.5),
        A.GaussianBlur(p=0.5),
        A.MedianBlur(blur_limit=5, p=0.5),
        A.GaussNoise(var_limit=(10.0, 50.0), p=0.5),
        A.Sharpen(p=0.5),
    ], p=1.0),
    A.RandomBrightnessContrast(p=0.3),
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
```

```
    ToTensorV2()
])

# Augmentazione di base
base_aug = A.Compose([
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])
```

```
def _load_and_augment(self, root_dir):
    sample_dirs = []
    for id in self.sample_ids:
        sample_dirs.append(os.path.join(root_dir, id))

    for dir_path in sample_dirs:
        img_path = os.path.join(dir_path, 'rgb.jpg')
        label_path = os.path.join(dir_path, 'labels.png')

        label_img = Image.open(label_path)
        is_rare =
SegmentationDataset.contains_rare_classes(label_img,
RARE_CLASSES)

        if not self.allow_augmentation:
            is_rare = False

        # Sempre aggiungi l'originale
        self.samples.append((img_path, label_path,
is_rare))

        # Duplicazione per oversampling (solo se contiene
classi rare)
        if is_rare and not self.is_val:
            for _ in range(DUPLICATE_FACTOR - 1):
                self.samples.append((img_path,
label_path, is_rare))

if self.is_val:
    aug = SegmentationDataset.base_aug
else:
    # Carica la maschera come immagine PIL per
controllare le classi
    label_img = Image.open(label_path)
```

```
-       is_confused =
    SegmentationDataset.contains_confused_classes(label_img)
-
-       if is_confused:
-           aug = SegmentationDataset.confused_aug
-       elif is_rare:
-           aug = SegmentationDataset.rare_aug
-       else:
-           aug = SegmentationDataset.base_aug
-
```

- **Scheduler**: off
- **Splitting:** on

```
o   import numpy as np
o   from PIL import Image
o   import os
o   from iterstrat.ml_stratifiers import
    MultilabelStratifiedShuffleSplit
o   import json
o
o   # sample male etichettati
o   excluded_dirs = {
o       "0007", "0009", "0090", "0095", "0101", "0104",
    "0105",
o       "0162", "0163", "0284", "0305", "0306", "0307",
    "0308", "0309", "0310", "0311",
o       "0351", "0372", "0373", "0376",
o       "0498", "0499", "0500", "0501", "0526", "0527",
    "0530", "0531", "0542",
o       "0564", "0585", "0586", "0587", "0588", "0589",
    "0590",
o       #seguono nuovo immagini da escludere
o       "0000", "0001", "0052"
o   }
o
o   # Elenco di cartelle numeriche valide
o   all_dirs = sorted([
o       d for d in os.listdir(DATASET_DIR)
o       if os.path.isdir(os.path.join(DATASET_DIR, d)) and
    d.isdigit() and d not in excluded_dirs
o   ])
o
o   # ------------------------------
o   #  COSTRUZIONE DELLA MATRICE MULTILABEL
o   # ------------------------------
o   def build_presence_matrix(dir_list, n_classes=9):
```

```
    """Return np.array (n_samples, n_classes) with
booleans indicating class presence"""
    presence = np.zeros((len(dir_list), n_classes),
dtype=int)

    for idx, d in enumerate(dir_list):
        label_path = os.path.join(DATASET_DIR, d,
"labels.png")
        lbl = np.array(Image.open(label_path))
        uniq = np.unique(lbl)
        presence[idx, uniq] = 1          # segna le
classi presenti
    return presence

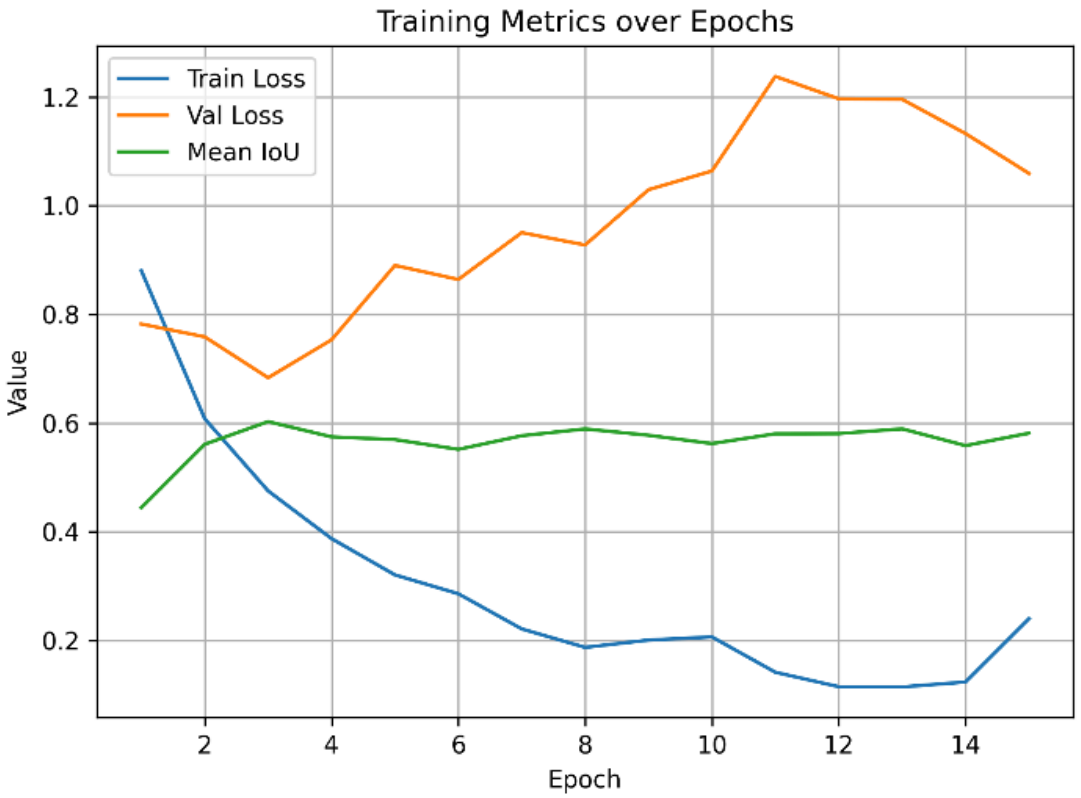
Y = build_presence_matrix(all_dirs, NUM_CLASSES)

# ------------------------------
#  MULTILABEL STRATIFIED SPLIT
# ------------------------------
msss = MultilabelStratifiedShuffleSplit(n_splits=1,
test_size=0.20, random_state=42)
train_idx, val_idx =
next(msss.split(np.zeros(len(all_dirs)), Y))

train_ids = [all_dirs[i] for i in train_idx]
val_ids   = [all_dirs[i] for i in val_idx]

print("Campioni:", len(train_ids), "train |",
len(val_ids), "val")
```

- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: on
  - if AUX_HEAD:
  - model.aux_classifier[-1] = nn.Conv2d(256, NUM_CLASSES, kernel_size=1)
  - else:
  - model.aux_classifier = None
  - if self.use_aux and "aux" in outputs:
  - aux_output = outputs.get("aux")
  - aux_loss = self.criterion(aux_output, masks)
  - loss += 0.4 * aux_loss

```
AUX_HEAD = True

trainer = Trainer(
    model=model,
    train_loader=train_loader,
    val_loader=val_loader,
    device=DEVICE,
    criterion=criterion,
    optimizer=optimizer,
    json_log_path="./training_log.json",
    scheduler=None,
    use_aux=AUX_HEAD
)
```

- **Loss Function / Weighting**: cross-entropy + class weights euristici [1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8].

- **Main classifier Kernel size:** due di dimensione 5

```
# 🔁 Primo layer convoluzionale: kernel 3x3
nn.Conv2d(256, 256, kernel_size=5, padding=2),  # mantiene dimensione
nn.ReLU(),
# 🔁 Secondo layer convoluzionale: kernel 5x5
nn.Conv2d(256, 256, kernel_size=5, padding=2),  # campo visivo più ampio
nn.ReLU(),
```

- **ASPP:** stride [3, 7, 3]

```
model = models.segmentation.deeplabv3_resnet101(weights=weights)
# versione custom del classifier
model.classifier = CustomDeepLabHead(2048, NUM_CLASSES)
```

```

class CustomDeepLabHead(nn.Sequential):
    def __init__(self, in_channels, num_classes):
        super().__init__(
            ASPP(in_channels, [3, 7, 3]),  # ASPP con dilatazioni personalizzate
            nn.Conv2d(256, 256, kernel_size=5, padding=2),  # kernel 5x5
            nn.ReLU(),
            nn.Dropout(0.05),
            nn.Conv2d(256, num_classes, kernel_size=1)  # output classi
        )
```

- **Valutazione risultati**:

  Migliore Validation Loss ottenuta all'epoca 3:
  📊 Train Loss: 0.6645 | Val Loss: 0.6926 | Mean IoU: 0.5837
  - IoU classe 0: 0.5200
  - IoU classe 1: 0.5868
  - IoU classe 2: 0.4574
  - IoU classe 3: 0.5728
  - IoU classe 4: 0.7273
  - IoU classe 5: 0.3785
  - IoU classe 6: 0.2812
  - IoU classe 7: 0.7813
  - IoU classe 8: 0.8846

  Migliore mean IoU ottenuta all'epoca 9:
  📊 Train Loss: 0.2170 | Val Loss: 0.9923 | Mean IoU: 0.6363
  - IoU classe 0: 0.5503
  - IoU classe 1: 0.6650
  - IoU classe 2: 0.5687
  - IoU classe 3: 0.5805
  - IoU classe 4: 0.7819
  - IoU classe 5: 0.4737
  - IoU classe 6: 0.2927
  - IoU classe 7: 0.8277
  - IoU classe 8: 0.9003

- **Rappresentazione grafica del training:**



- **Massima memoria RAM occupata durante il training:** 2.7GB
- **Log di training**: `risultati_training\deeplabv3\resnet101\dlrn1010001\training_log_dlrn101021.json`
- **Note / Bug Fix / Annotazioni**:

✳ Esperimento: `deeplabv3_resnet101 - dlrn101022`

- **Architettura**: `deeplabv3plus_resnet101`
- **Backbone Weights**: DeepLabV3_ResNet101_Weights.DEFAULT
- **Input size**: 256x256
- **Batch size**: 4
- **Augmentation**: on

```
# Augmentazione per immagini con classi rare
rare_aug = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.4),
```

```
    #A.Rotate(limit=20, p=0.3), potenzialmente inutile
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])

# Augmentazione mirata per immagini che contengono sia classi 1 che 3
confused_aug = A.Compose([
    A.OneOf([
        A.MotionBlur(p=0.5),
        A.GaussianBlur(p=0.5),
        A.MedianBlur(blur_limit=5, p=0.5),
        A.GaussNoise(var_limit=(10.0, 50.0), p=0.5),
        A.Sharpen(p=0.5),
    ], p=1.0),
    A.RandomBrightnessContrast(p=0.3),
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])

# Augmentazione di base
base_aug = A.Compose([
    A.Resize(256, 256),
    A.Normalize(mean=(0.485, 0.456, 0.406),
                std=(0.229, 0.224, 0.225)),
    ToTensorV2()
])
```

```
def _load_and_augment(self, root_dir):
    sample_dirs = []
    for id in self.sample_ids:
        sample_dirs.append(os.path.join(root_dir, id))

    for dir_path in sample_dirs:
        img_path = os.path.join(dir_path, 'rgb.jpg')
        label_path = os.path.join(dir_path, 'labels.png')

        label_img = Image.open(label_path)
```

```
        is_rare = SegmentationDataset.contains_rare_classes(label_img, RARE_CLASSES)

        if not self.allow_augmentation:
            is_rare = False

        # Sempre aggiungi l'originale
        self.samples.append((img_path, label_path, is_rare))

        # Duplicazione per oversampling (solo se contiene classi rare)
        if is_rare and not self.is_val:
            for _ in range(DUPLICATE_FACTOR - 1):
                self.samples.append((img_path, label_path, is_rare))

if self.is_val:
    aug = SegmentationDataset.base_aug
else:
    # Carica la maschera come immagine PIL per controllare le classi
    label_img = Image.open(label_path)
    is_confused = SegmentationDataset.contains_confused_classes(label_img)

    if is_confused:
        aug = SegmentationDataset.confused_aug
    elif is_rare:
        aug = SegmentationDataset.rare_aug
    else:
        aug = SegmentationDataset.base_aug
```

- **Scheduler**: off
- **Splitting:** on

```
import numpy as np
from PIL import Image
import os
from iterstrat.ml_stratifiers import MultilabelStratifiedShuffleSplit
import json

# sample male etichettati
excluded_dirs = {
```

```
    "0007", "0009", "0090", "0095", "0101", "0104",
"0105",
    "0162", "0163", "0284", "0305", "0306", "0307",
"0308", "0309", "0310", "0311",
    "0351", "0372", "0373", "0376",
    "0498", "0499", "0500", "0501", "0526", "0527",
"0530", "0531", "0542",
    "0564", "0585", "0586", "0587", "0588", "0589",
"0590",
    #seguono nuovo immagini da escludere
    "0000", "0001", "0052"
}

# Elenco di cartelle numeriche valide
all_dirs = sorted([
    d for d in os.listdir(DATASET_DIR)
    if os.path.isdir(os.path.join(DATASET_DIR, d)) and
d.isdigit() and d not in excluded_dirs
])

# ------------------------------
#  COSTRUZIONE DELLA MATRICE MULTILABEL
# ------------------------------
def build_presence_matrix(dir_list, n_classes=9):
    """Return np.array (n_samples, n_classes) with
booleans indicating class presence"""
    presence = np.zeros((len(dir_list), n_classes),
dtype=int)

    for idx, d in enumerate(dir_list):
        label_path = os.path.join(DATASET_DIR, d,
"labels.png")
        lbl = np.array(Image.open(label_path))
        uniq = np.unique(lbl)
        presence[idx, uniq] = 1          # segna le
classi presenti
    return presence

Y = build_presence_matrix(all_dirs, NUM_CLASSES)

# ------------------------------
#  MULTILABEL STRATIFIED SPLIT
# ------------------------------
msss = MultilabelStratifiedShuffleSplit(n_splits=1,
test_size=0.20, random_state=42)
```

```
train_idx, val_idx =
next(msss.split(np.zeros(len(all_dirs)), Y))

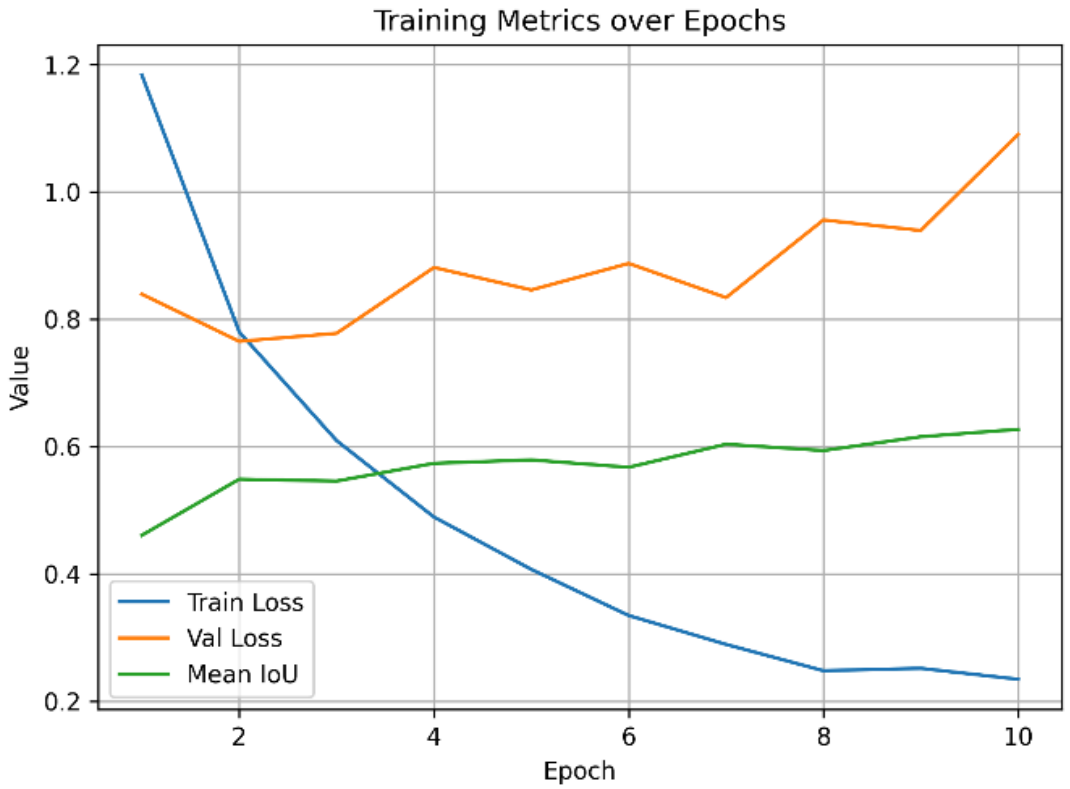
train_ids = [all_dirs[i] for i in train_idx]
val_ids   = [all_dirs[i] for i in val_idx]

print("Campioni:", len(train_ids), "train |",
len(val_ids), "val")
```

- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: on
    - if AUX_HEAD:
    - model.aux_classifier[-1] = nn.Conv2d(256, NUM_CLASSES, kernel_size=1)
    - else:
    - model.aux_classifier = None
    - if self.use_aux and "aux" in outputs:
    - aux_output = outputs.get("aux")
    - aux_loss = self.criterion(aux_output, masks)
    - loss += 0.3 * aux_loss
    - AUX_HEAD = True
    - trainer = Trainer(
    - model=model,
    - train_loader=train_loader,
    - val_loader=val_loader,
    - device=DEVICE,
    - criterion=criterion,
    - optimizer=optimizer,
    - json_log_path="./training_log.json",
    - scheduler=None,
    - use_aux=AUX_HEAD
    -)
- **Loss Function / Weighting**: cross-entropy + class weights euristici [1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8].
- **Main classifier Kernel size:** due di dimensione 5

```
# 🔁 Primo layer convoluzionale: kernel 3x3
nn.Conv2d(256, 256, kernel_size=5, padding=2),  #
mantiene dimensione
```

```
    nn.ReLU(),
    # 🔁 Secondo layer convoluzionale: kernel 5x5
    nn.Conv2d(256, 256, kernel_size=5, padding=2),  # campo
    visivo più ampio
    nn.ReLU(),
```

- **ASPP:** stride [3, 7, 3]
  - ```
    model =
    models.segmentation.deeplabv3_resnet101(weights=weights)
    ```
  - ```
    # versione custom del classifier
    ```
  - ```
    model.classifier = CustomDeepLabHead(2048, NUM_CLASSES)
    ```

```

class CustomDeepLabHead(nn.Sequential):
    def __init__(self, in_channels, num_classes):
        super().__init__(
            ASPP(in_channels, [3, 7, 3]),  # ASPP con
    dilatazioni personalizzate
            nn.Conv2d(256, 256, kernel_size=5,
    padding=2),  # kernel 5x5
            nn.ReLU(),
            nn.Dropout(0.05),
            nn.Conv2d(256, num_classes, kernel_size=1)
    # output classi
        )
```

- **Valutazione risultati**:
  Migliore Validation Loss ottenuta all'epoca 2:
  📊 Train Loss: 0.7801 | Val Loss: 0.7652 | Mean IoU: 0.5487
  - IoU classe 0: 0.5069
  - IoU classe 1: 0.6413
  - IoU classe 2: 0.3969
  - IoU classe 3: 0.5744
  - IoU classe 4: 0.5398
  - IoU classe 5: 0.3108
  - IoU classe 6: 0.2558
  - IoU classe 7: 0.7924
  - IoU classe 8: 0.8784

  Migliore mean IoU ottenuta all'epoca 10:
  📊 Train Loss: 0.2344 | Val Loss: 1.0903 | Mean IoU: 0.6270
  - IoU classe 0: 0.5482
  - IoU classe 1: 0.6693
  - IoU classe 2: 0.5924
  - IoU classe 3: 0.6273
  - IoU classe 4: 0.6607
```

- IoU classe 5: 0.4515
- IoU classe 6: 0.2718
- IoU classe 7: 0.8442
- IoU classe 8: 0.8990

- **Rappresentazione grafica del training:**



- **Massima memoria RAM occupata durante il training:** 2.7GB

- **Log di training**: `risultati_training\deeplabv3\resnet101\dlrn1010001\training_log_dlrn101022.json`

- **Note / Bug Fix / Annotazioni**:

deeplabv3_mobilenet_v3_large

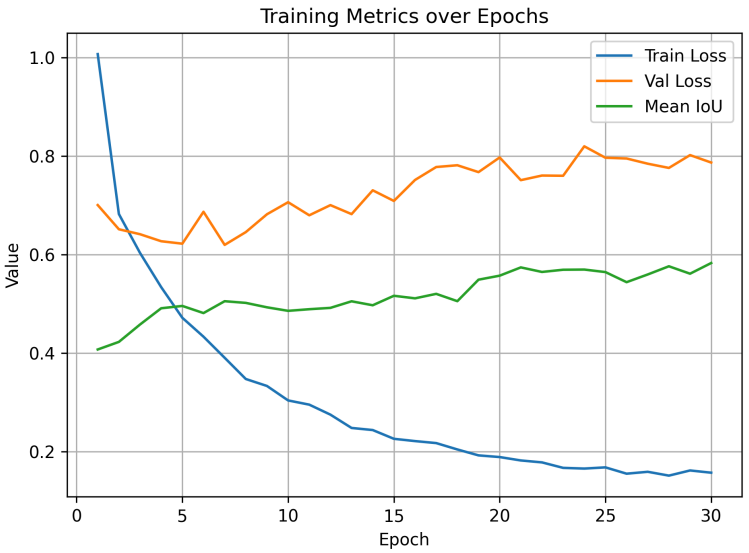
✳ Esperimento: `deeplabv3_mobilenet_v3_large - DLMN000`

- **Architettura**: `deeplabv3plus_mobilenet_v3_large`

- **Backbone Weights**: DeepLabV3_MobileNet_V3_Large_Weights.DEFAULT

- **Input size**: 256x256

- **Batch size**: 4
- **Augmentation**: off
- **Scheduler**: off
- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: off
- **Loss Function / Weighting**: cross-entropy
- **Main classifier Kernel size:** 1
- **Valutazione risultati**:
    - Migliore Validation Loss ottenuta all'epoca 7:
        - 📊 Train Loss: 0.3904 | Val Loss: 0.6196 | Mean IoU: 0.5052
        - ▸ IoU classe 0: 0.3568
        - ▸ IoU classe 1: 0.5695
        - ▸ IoU classe 2: 0.5950
        - ▸ IoU classe 3: 0.5364
        - ▸ IoU classe 4: 0.0000
        - ▸ IoU classe 5: 0.5020
        - ▸ IoU classe 6: 0.1968
        - ▸ IoU classe 7: 0.8017
        - ▸ IoU classe 8: 0.8404
    - Migliore `mean IoU` ottenuta all'epoca 30:
        - 📊 Train Loss: 0.1570 | Val Loss: 0.7868 | Mean IoU: 0.5828
        - ▸ IoU classe 0: 0.4531
        - ▸ IoU classe 1: 0.5564
        - ▸ IoU classe 2: 0.6296
        - ▸ IoU classe 3: 0.5506
        - ▸ IoU classe 4: 0.5626
        - ▸ IoU classe 5: 0.5171
        - ▸ IoU classe 6: 0.1771
        - ▸ IoU classe 7: 0.8165
        - ▸ IoU classe 8: 0.8523
- **Rappresentazione grafica del training:**



- **Massima memoria RAM occupata durante il training:** 0.56GB
- **Log di training**: `risultati_training\deeplabv3\mobilenetv3_large\dlmnv3l0001\training_log_dlmnv3l0000.json`
- **Note / Bug Fix / Annotazioni**:
  - —

✳ Esperimento: `deeplabv3_mobilenet_v3_large - DLMN001`

- **Architettura**: `deeplabv3plus_mobilenet_v3_large`
- **Backbone Weights**: DeepLabV3_MobileNet_V3_Large_Weights.DEFAULT
- **Input size**: 256x256
- **Batch size**: 4
- **Augmentation**: off
- **Scheduler**: off
- **Optimizer:** Adama con lr=1e-4
- **Freezing**: none
- **Auxiliary Head**: off

- **Loss Function / Weighting**: cross-entropy + class weights euristici [1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8].

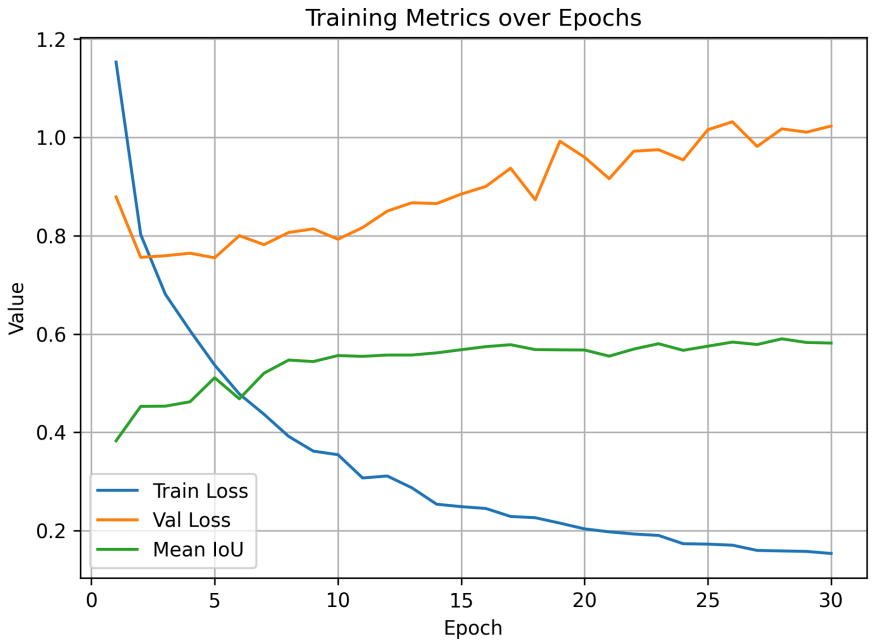
- **Main classifier Kernel size:** 1

- **Valutazione risultati**:

    - Migliore Validation Loss ottenuta all'epoca 5:
        - 📊 Train Loss: 0.5369 | Val Loss: 0.7551 | Mean IoU: 0.5111
        - ▸ IoU classe 0: 0.3274
        - ▸ IoU classe 1: 0.4677
        - ▸ IoU classe 2: 0.5834
        - ▸ IoU classe 3: 0.5334
        - ▸ IoU classe 4: 0.2439
        - ▸ IoU classe 5: 0.4288
        - ▸ IoU classe 6: 0.2185
        - ▸ IoU classe 7: 0.7925
        - ▸ IoU classe 8: 0.8208

    - Migliore `mean IoU` ottenuta all'epoca 26:
        - 📊 Train Loss: 0.1706 | Val Loss: 1.0316 | Mean IoU: 0.5837
        - ▸ IoU classe 0: 0.4317
        - ▸ IoU classe 1: 0.5694
        - ▸ IoU classe 2: 0.5931
        - ▸ IoU classe 3: 0.5216
        - ▸ IoU classe 4: 0.6083
        - ▸ IoU classe 5: 0.5410
        - ▸ IoU classe 6: 0.1721
        - ▸ IoU classe 7: 0.8126
        - ▸ IoU classe 8: 0.8511

- **Rappresentazione grafica del training:**



- **Massima memoria RAM occupata durante il training:** 0.56GB
- **Log di training**: `risultati_training\deeplabv3\mobilenetv3_large\dlmnv3l0001\training_log_dlmnv3l0001.json`
- **Note / Bug Fix / Annotazioni**:
  - —