

analysisV2

August 16, 2025

0.1 Hybrid Model Evaluation

Jakob Balkovec Fri, Aug 15th 2025

0.2 Data

```
[138]: import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd

METRICS_LOG_CSV = "metrics_log.csv"
NPZ_DIR = "npz"
MASK_VISUALS_DIR = "mask_visuals"

LOSS_LAMBDA = 1.0
```

```
[139]: df = pd.read_csv(METRICS_LOG_CSV)
df.head()
```

```
[139]:   epoch      lr  train_loss  val_loss  train_f1_micro  train_f1_macro \
0       1  0.00010    0.532256  20.283897      0.905405      0.891202
1       2  0.00010    0.185695  33.477348      0.977486      0.971653
2       3  0.00010    0.140118  11.572494      0.989590      0.986913
3       4  0.00005    0.117095  18.454541      0.995027      0.994114
4       5  0.00005    0.118745  25.787544      0.998485      0.998359

      train_precision_micro  train_recall_micro  train_iou_micro \
0            0.830594        0.995027        0.827161
1            0.957675        0.998133        0.955963
2            0.980439        0.998914        0.979395
3            0.990569        0.999524        0.990103
4            0.997180        0.999794        0.996974

      train_roc_auc_macro ...  train_coord_r2  val_coord_mae  val_coord_rmse \
0             NaN  ...       -4.662195      0.246274      0.293481
1             NaN  ...       -4.778428      0.328962      0.399748
2             NaN  ...       -4.501243      0.232830      0.274532
```

```

3          NaN ...      -4.486558      0.227600      0.267413
4          NaN ...      -4.562735      0.239165      0.284681

  val_coord_r2  train_seg_iou  train_seg_dice  val_seg_iou  val_seg_dice \
0   -0.238092      0.005400      0.010743      0.000233      0.000467
1   -1.297030      0.039032      0.075132      0.000991      0.001981
2   -0.083374      0.053155      0.100944      0.002302      0.004594
3   -0.027917      0.066257      0.124280      0.001957      0.003906
4   -0.164959      0.089778      0.164763      0.001600      0.003196

    monitor  monitor_value
0  val_f1_micro      0.582181
1  val_f1_micro      0.395720
2  val_f1_micro      0.396729
3  val_f1_micro      0.540343
4  val_f1_micro      0.360014

[5 rows x 60 columns]

```

```
[140]: def split_train_val(df):
    base = pd.DataFrame({"epoch": df["epoch"]})

    train_cols = [c for c in df.columns if c.startswith("train_")]
    val_cols   = [c for c in df.columns if c.startswith("val_")]

    def strip_prefix(cols, prefix):
        out = {}
        for c in cols:
            key = c[len(prefix):]
            out[key] = c
        return out

    train_map = strip_prefix(train_cols, "train_")
    val_map   = strip_prefix(val_cols, "val_")

    shared = sorted(set(train_map.keys()) & set(val_map.keys()))

    df_train = base.copy()
    df_val   = base.copy()
    for k in shared:
        df_train[k] = df[train_map[k]]
        df_val[k]   = df[val_map[k]]

    return df_train, df_val, shared

df_train, df_val, shared_metrics = split_train_val(df)
```

```
[141]: df_train.head()
```

```
[141]:    epoch  coord_mae  coord_r2  coord_rmse  f1_class_0  f1_class_1  f1_class_2  \
0         1   0.513849 -4.662195   0.627393   0.905399   0.951106   0.928947
1         2   0.520783 -4.778428   0.633754   0.977795   0.989902   0.987272
2         3   0.509247 -4.501243   0.618395   0.990240   0.995090   0.993056
3         4   0.508798 -4.486558   0.617571   0.994136   0.997534   0.996419
4         5   0.511363 -4.562735   0.621819   0.997156   0.999347   0.999252

      f1_class_3  f1_macro  f1_micro  ...  precision_class_3  precision_micro  \
0     0.779356  0.891202  0.905405  ...          0.643213        0.830594
1     0.931642  0.971653  0.977486  ...          0.875144        0.957675
2     0.969265  0.986913  0.989590  ...          0.942166        0.980439
3     0.988370  0.994114  0.995027  ...          0.977634        0.990569
4     0.997679  0.998359  0.998485  ...          0.995888        0.997180

  recall_class_0  recall_class_1  recall_class_2  recall_class_3  \
0     0.997473       0.994716       0.996141       0.988603
1     0.998575       0.998381       0.998503       0.995938
2     0.999287       0.998920       0.999002       0.997969
3     0.999546       0.999574       0.999534       0.999345
4     0.999708       0.999915       0.999900       0.999476

  recall_micro  roc_auc_macro  seg_dice  seg_iou
0     0.995027           NaN  0.010743  0.005400
1     0.998133           NaN  0.075132  0.039032
2     0.998914           NaN  0.100944  0.053155
3     0.999524           NaN  0.124280  0.066257
4     0.999794           NaN  0.164763  0.089778
```

[5 rows x 29 columns]

```
[142]: df_val.head()
```

```
[142]:    epoch  coord_mae  coord_r2  coord_rmse  f1_class_0  f1_class_1  f1_class_2  \
0         1   0.246274 -0.238092   0.293481   0.026753   0.773063   0.797500
1         2   0.328962 -1.297030   0.399748   0.000000   0.001372   0.797500
2         3   0.232830 -0.083374   0.274532   0.000000   0.732393   0.000000
3         4   0.227600 -0.027917   0.267413   0.000000   0.867433   0.482190
4         5   0.239165 -0.164959   0.284681   0.001822   0.002581   0.774378

      f1_class_3  f1_macro  f1_micro  ...  precision_class_3  precision_micro  \
0     0.000000  0.399329  0.582181  ...          0.000000        0.732015
1     0.330118  0.282248  0.395720  ...          0.265548        0.523550
2     0.496302  0.307174  0.396729  ...          0.330054        0.524277
3     0.385476  0.433775  0.540343  ...          0.383438        0.702840
4     0.000000  0.194695  0.360014  ...          0.000000        0.655062
```

```

recall_class_0  recall_class_1  recall_class_2  recall_class_3 \
0      0.013559      0.722577      1.000000      0.000000
1      0.000000      0.000687      1.000000      0.436176
2      0.000000      0.627060      0.000000      1.000000
3      0.000000      1.000000      0.332743      0.387535
4      0.000912      0.001292      0.947854      0.000000

recall_micro  roc_auc_macro  seg_dice  seg_iou
0      0.483263      NaN  0.000467  0.000233
1      0.318062      NaN  0.001981  0.000991
2      0.319098      NaN  0.004594  0.002302
3      0.438875      NaN  0.003906  0.001957
4      0.248215      NaN  0.003196  0.001600

```

[5 rows x 29 columns]

```

[143]: def LOSS_BANDS(loss_lambda: float = 1.0):
    low_thr = 0.7 + 0.4 * loss_lambda
    high_thr = 1.5 + 0.4 * loss_lambda
    max_thr = high_thr + 1.0
    return [
        (0.00, low_thr, "green", f"Low (< {low_thr:.2f})"),
        (low_thr, high_thr, "yellow", f"Moderate ({low_thr:.2f}-{high_thr:.2f})"),
        (high_thr, max_thr, "red", f"High (> {high_thr:.2f})"),
    ]

BANDS = {
    "f1_micro": [
        (0.00, 0.60, "red", "Poor (< 0.60)"),
        (0.60, 0.75, "yellow", "Moderate (0.60-0.75)"),
        (0.75, 1.00, "green", "Good (> 0.75)"),
    ],
    "f1_macro": [
        (0.00, 0.60, "red", "Poor (< 0.60)"),
        (0.60, 0.75, "yellow", "Moderate (0.60-0.75)"),
        (0.75, 1.00, "green", "Good (> 0.75)"),
    ],
    "precision_micro": [
        (0.00, 0.60, "red", "Low (< 0.60)"),
        (0.60, 0.80, "yellow", "Moderate (0.60-0.80)"),
        (0.80, 1.00, "green", "High (> 0.80)"),
    ],
    "recall_micro": [
        (0.00, 0.60, "red", "Low (< 0.60)"),
        (0.60, 0.80, "yellow", "Moderate (0.60-0.80)"),

```

```

        (0.80, 1.00, "green", "High (> 0.80")),
    ],
    "iou_micro": [
        (0.00, 0.50, "red", "Low (< 0.50)"),
        (0.50, 0.70, "yellow", "Moderate (0.50-0.70)"),
        (0.70, 1.00, "green", "High (> 0.70)"),
    ],
    "roc_auc_macro": [
        (0.50, 0.70, "red", "Below target (< 0.70)"),
        (0.70, 0.85, "yellow", "Decent (0.70-0.85)"),
        (0.85, 1.00, "green", "Strong (> 0.85)"),
    ],
    "seg_iou": [
        (0.00, 0.50, "red", "Low (< 0.50)"),
        (0.50, 0.70, "yellow", "Moderate (0.50-0.70)"),
        (0.70, 1.00, "green", "High (> 0.70)"),
    ],
    "seg_dice": [
        (0.00, 0.60, "red", "Poor (< 0.60)"),
        (0.60, 0.75, "yellow", "Fair (0.60-0.75)"),
        (0.75, 1.00, "green", "Good (> 0.75)"),
    ],
    "coord_r2": [
        (-5.00, 0.00, "red", "Worse than baseline (< 0)"),
        (0.00, 0.50, "yellow", "Moderate (0-0.5)"),
        (0.50, 1.00, "green", "Good (> 0.5)"),
    ],
    "coord_mae": [
        (0.20, 10.0, "red", "High error (> 0.20)"),
        (0.10, 0.20, "yellow", "Moderate (0.10-0.20)"),
        (0.00, 0.10, "green", "Low (< 0.10)"),
    ],
    "coord_rmse": [
        (0.25, 10.0, "red", "High error (> 0.25)"),
        (0.15, 0.25, "yellow", "Moderate (0.15-0.25)"),
        (-5.0, 0.15, "green", "Low (< 0.15)"),
    ],
},
}

for c in range(4):
    # F1 per class
    BANDS[f"f1_class_{c}"] = [
        (0.00, 0.60, "red", "Poor (< 0.60)"),
        (0.60, 0.75, "yellow", "Moderate (0.60-0.75)"),
        (0.75, 1.00, "green", "Good (> 0.75)"),

```

```

]
# Precision per class
BANDS[f"precision_class_{c}"] = [
    (0.00, 0.60, "red", "Low (< 0.60)"),
    (0.60, 0.80, "yellow", "Moderate (0.60-0.80)"),
    (0.80, 1.00, "green", "High (> 0.80)"),
]
# Recall per class
BANDS[f"recall_class_{c}"] = [
    (0.00, 0.60, "red", "Low (< 0.60)"),
    (0.60, 0.80, "yellow", "Moderate (0.60-0.80)"),
    (0.80, 1.00, "green", "High (> 0.80)"),
]
# IoU per class
BANDS[f"iou_class_{c}"] = [
    (0.00, 0.50, "red", "Low (< 0.50)"),
    (0.50, 0.70, "yellow", "Moderate (0.50-0.70)"),
    (0.70, 1.00, "green", "High (> 0.70)"),
]

# Optional: helper to get bands for a metric (handles dynamic loss bands)
def get_bands(metric_name: str, *, loss_lambda: float = 1.0):
    if metric_name == "loss":
        return LOSS_BANDS(loss_lambda)
    return BANDS.get(metric_name, None)

```

```

[144]: def plot_train_val(df, metric_name, show_trend=True):
    train_col = f"train_{metric_name}"
    val_col = f"val_{metric_name}"
    if train_col not in df.columns or val_col not in df.columns:
        raise ValueError(f"{metric_name} not found in DataFrame.")

    x = df["epoch"]
    train_y = df[train_col]
    val_y = df[val_col]

    fig, axes = plt.subplots(2, 1, figsize=(10, 8), sharex=True)

    # --- Train plot
    axes[0].plot(x, train_y, marker='o', label="Train", linewidth=2)
    axes[0].set_ylabel(metric_name)
    axes[0].set_title(f"{metric_name} - Train")
    axes[0].grid(True, linestyle="--", alpha=0.5)

    # --- Val plot
    axes[1].plot(x, val_y, marker='o', label="Validation", linewidth=2)
    axes[1].set_xlabel("Epoch")

```

```

axes[1].set_ylabel(metric_name)
axes[1].set_title(f"{metric_name} - Validation")
axes[1].grid(True, linestyle="--", alpha=0.5)

# --- Trend line on val
if show_trend and val_y.notna().sum() >= 3:
    z = np.polyfit(x, val_y, 2)
    p = np.poly1d(z)
    x_fit = np.linspace(x.min(), x.max(), 200)
    axes[1].plot(x_fit, p(x_fit), "--", label="Trend")

# --- Bands
bands = get_bands(metric_name, loss_lambda=1.0) # set your lambda
for ax in axes:
    if bands:
        for lo, hi, color, lab in bands:
            ax.axhspan(lo, hi, color=color, alpha=0.10, label=lab)

# --- Best val point
val_clean = val_y.dropna()
if not val_clean.empty:
    if metric_name in ["loss"]:
        best_idx = val_clean.idxmin()
    else:
        best_idx = val_clean.idxmax()
    best_epoch = x[best_idx]
    best_value = val_clean.loc[best_idx]
    axes[1].scatter(best_epoch, best_value, s=100, zorder=5, label=f"Best:{best_value:.3f}")
    axes[1].axvline(best_epoch, linestyle=":", linewidth=1.5)

# --- Legends
for ax in axes:
    handles, labels = ax.get_legend_handles_labels()
    by_label = dict(zip(labels, handles))
    ax.legend(by_label.values(), by_label.keys(), loc="best")

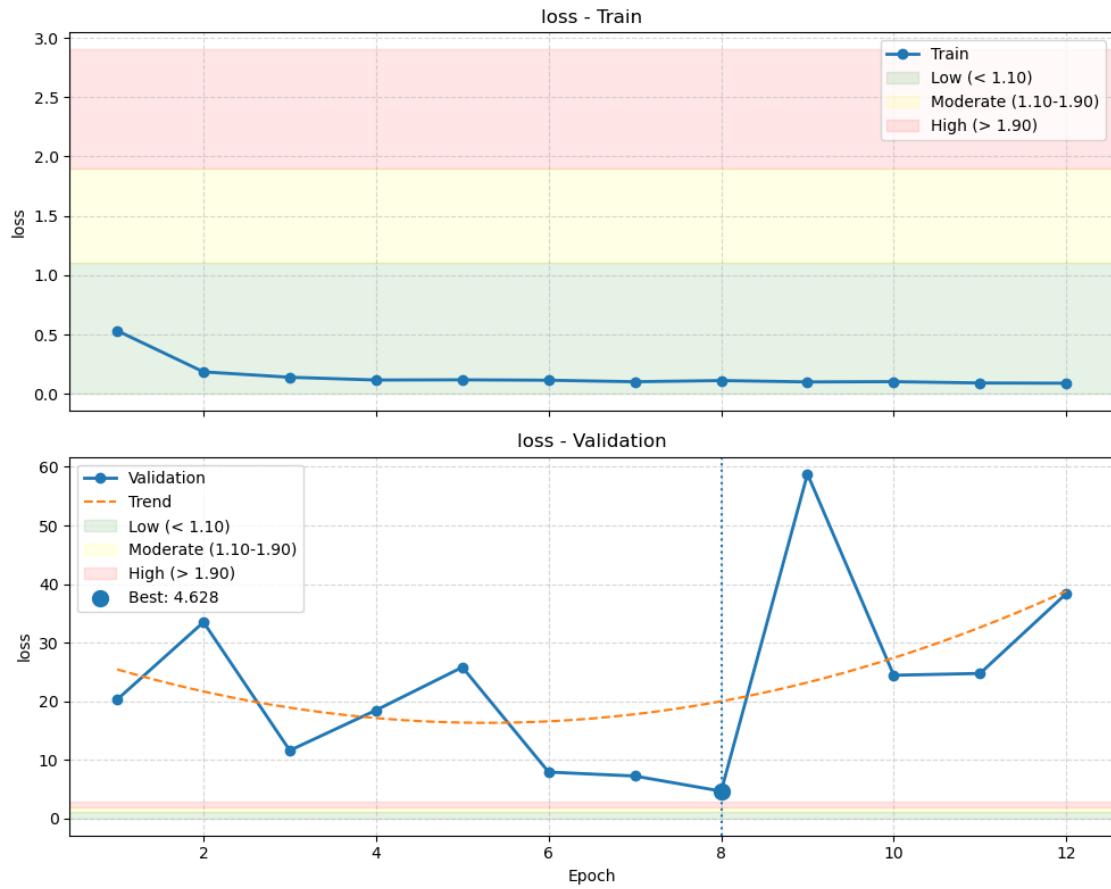
plt.tight_layout()
plt.show()

```

1 Global Classification Metrics

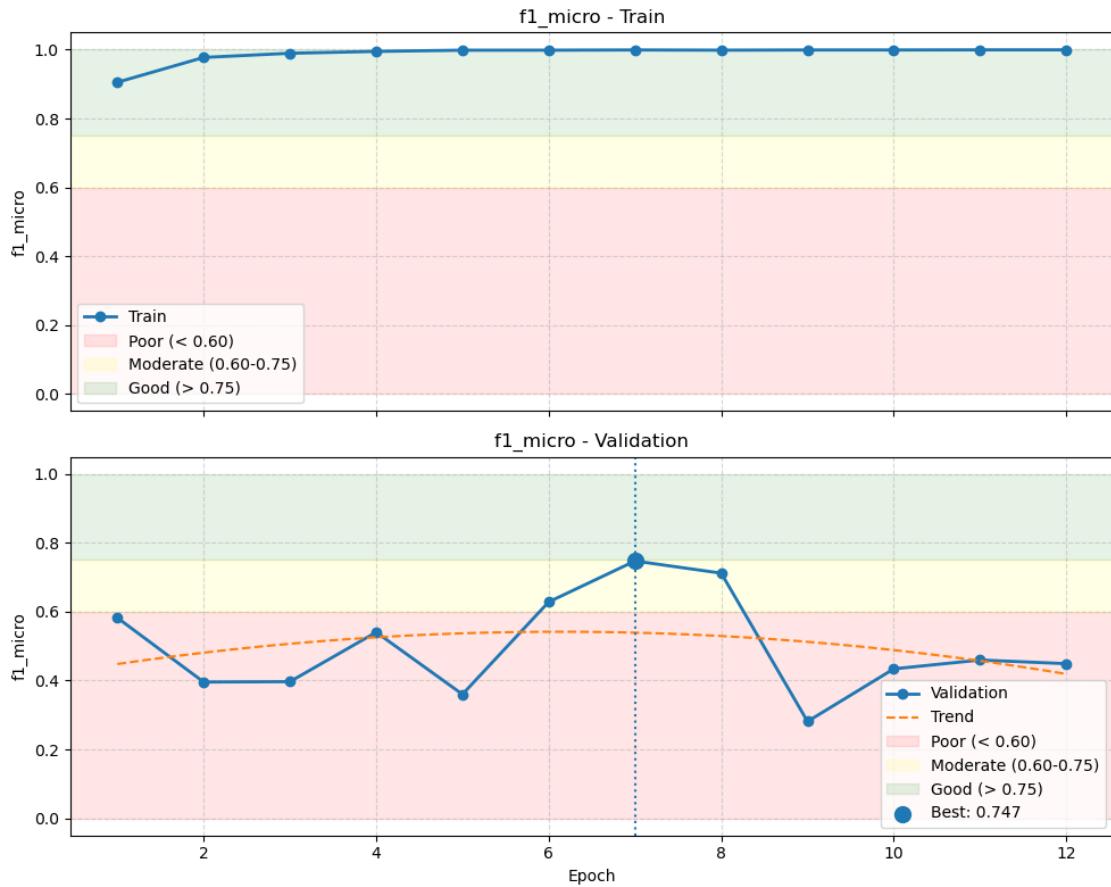
1.1 Loss

[145]: plot_train_val(df, "loss") # fix scale



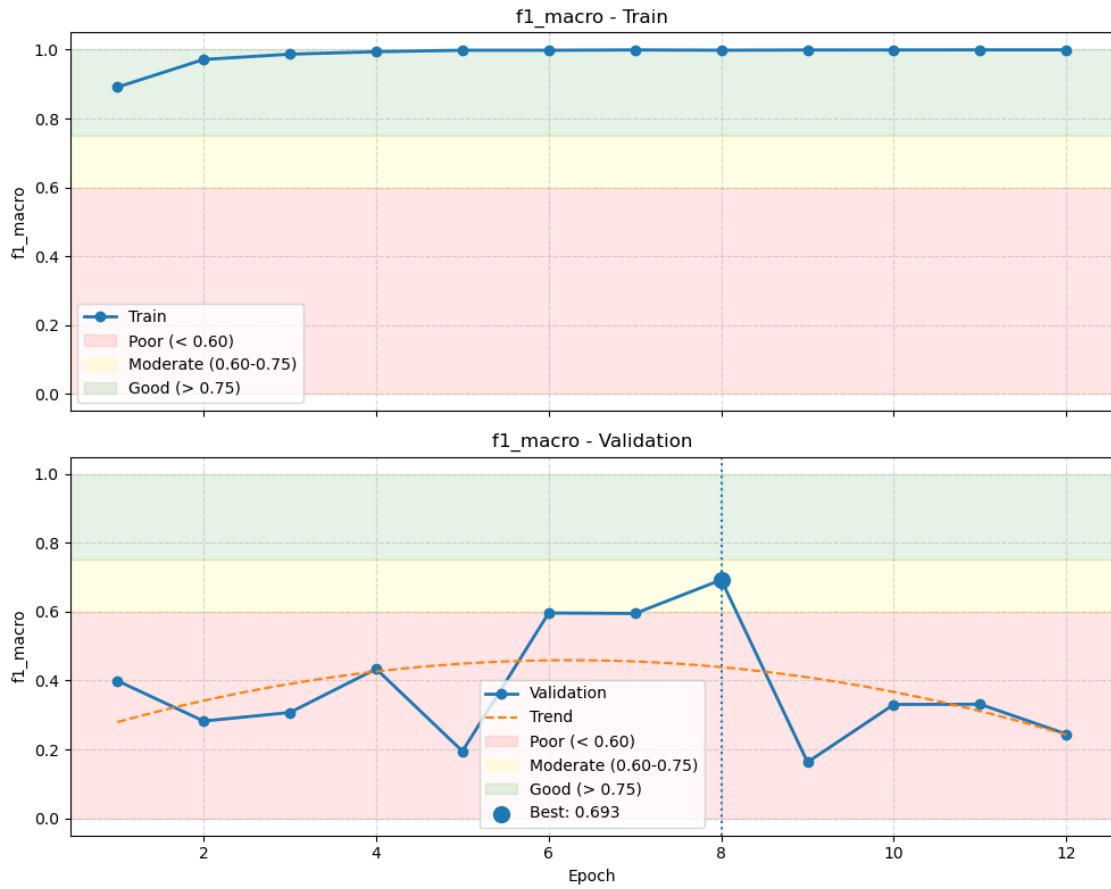
1.2 F1 - Micro

```
[146]: plot_train_val(df, "f1_micro")
```



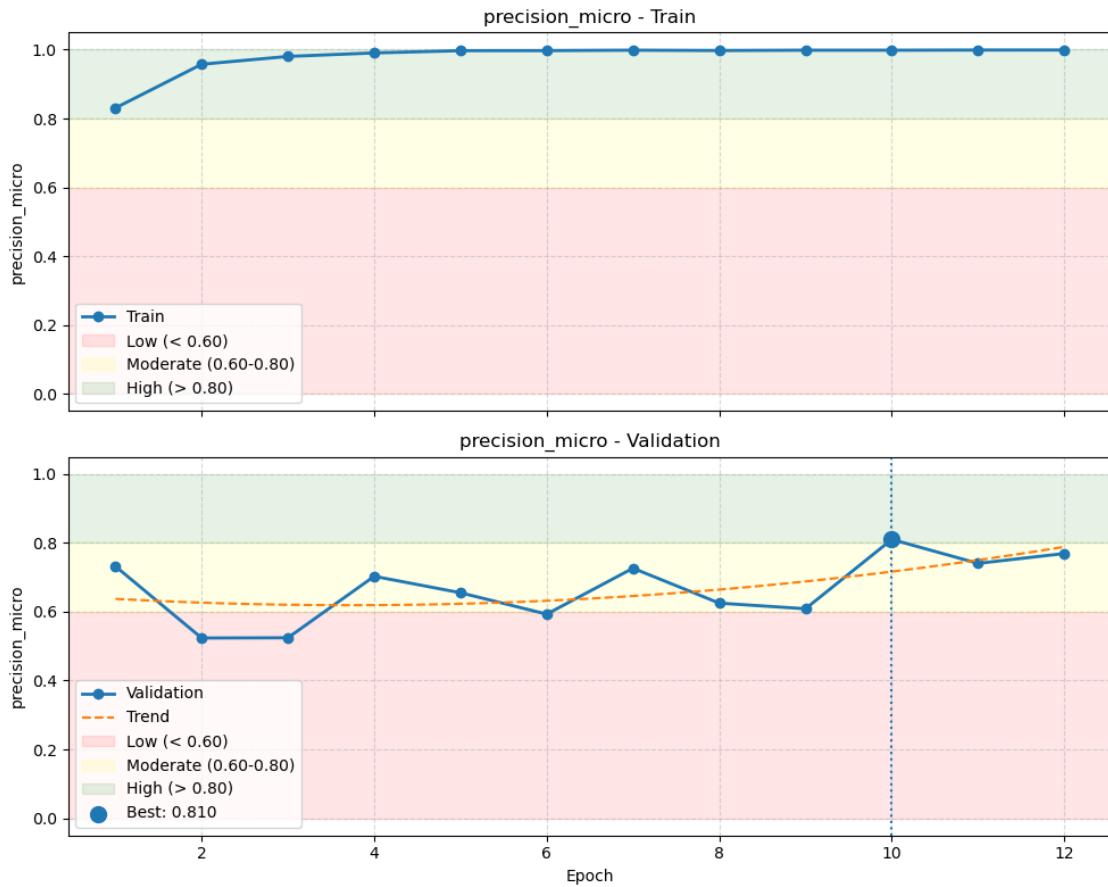
1.3 F1 - Macro

```
[147]: plot_train_val(df, "f1_macro")
```



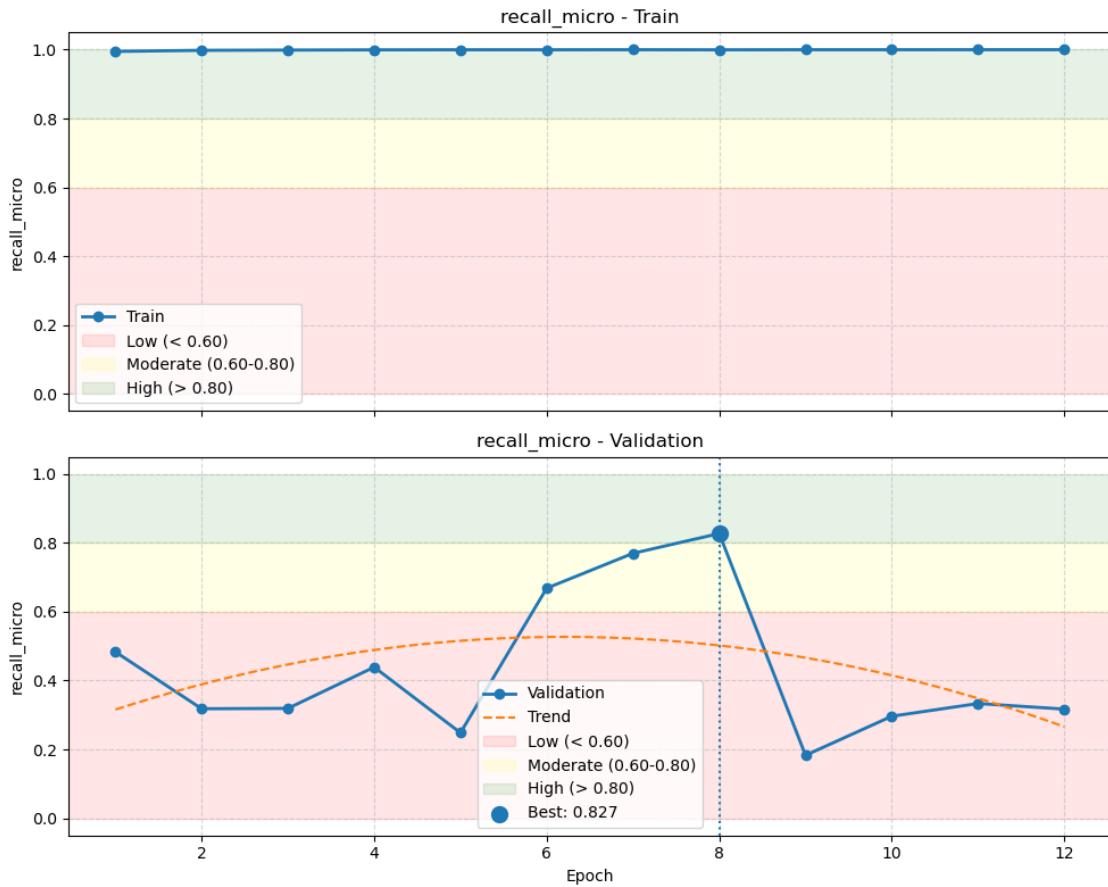
1.4 Percision Micro

```
[148]: plot_train_val(df, "precision_micro")
```



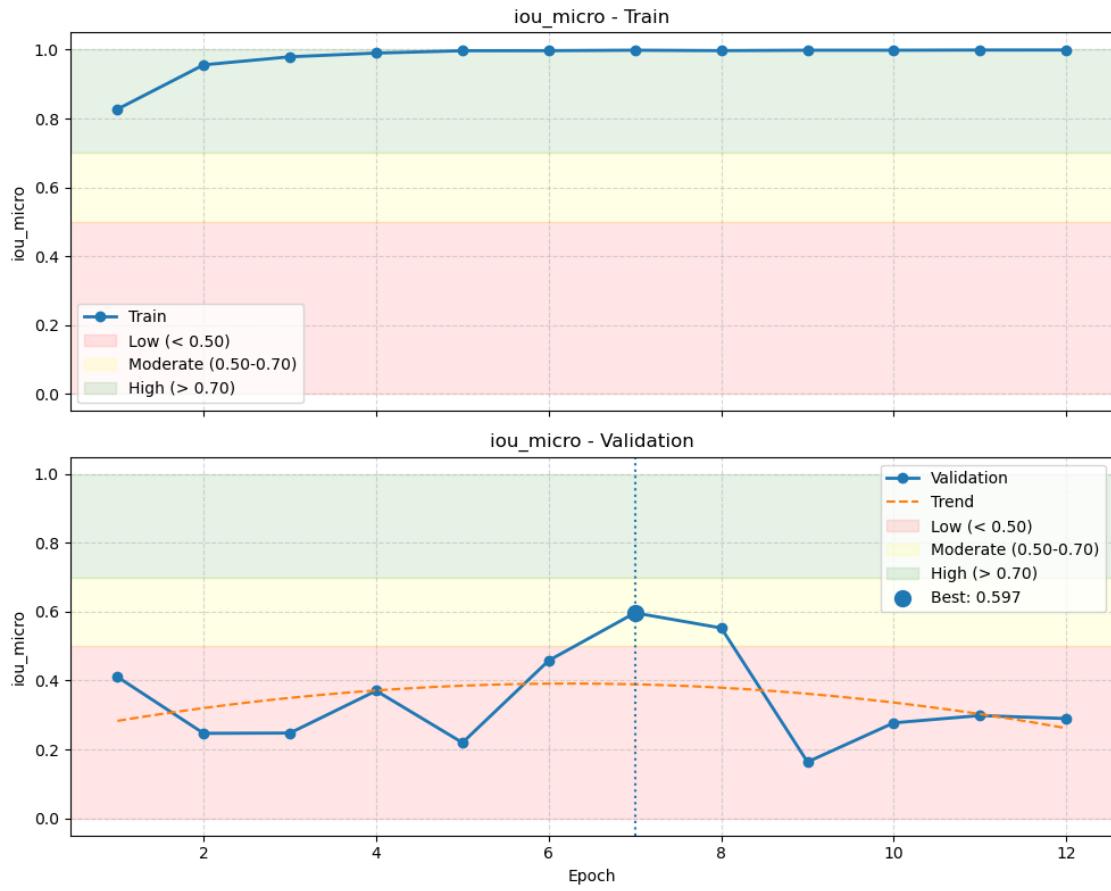
1.5 Recall Micro

```
[149]: plot_train_val(df, "recall_micro")
```



1.6 IoU Micro

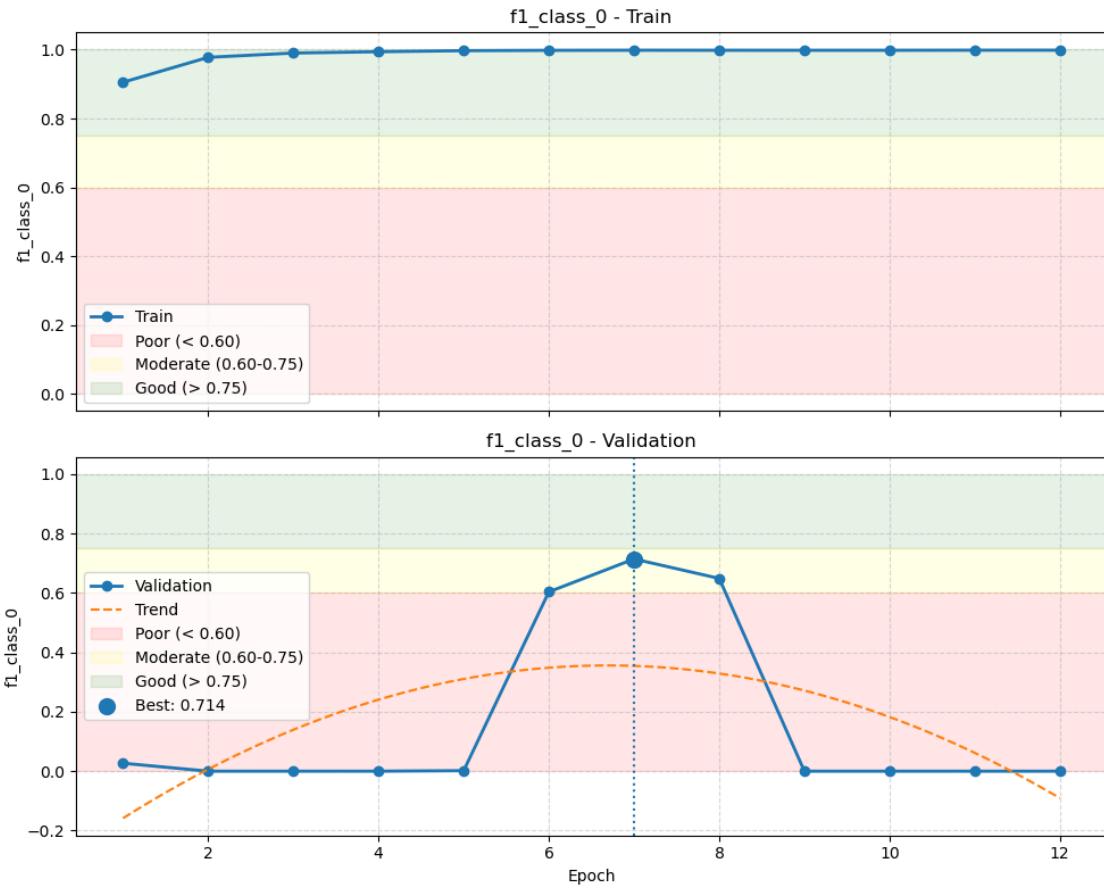
```
[150]: plot_train_val(df, "iou_micro")
```



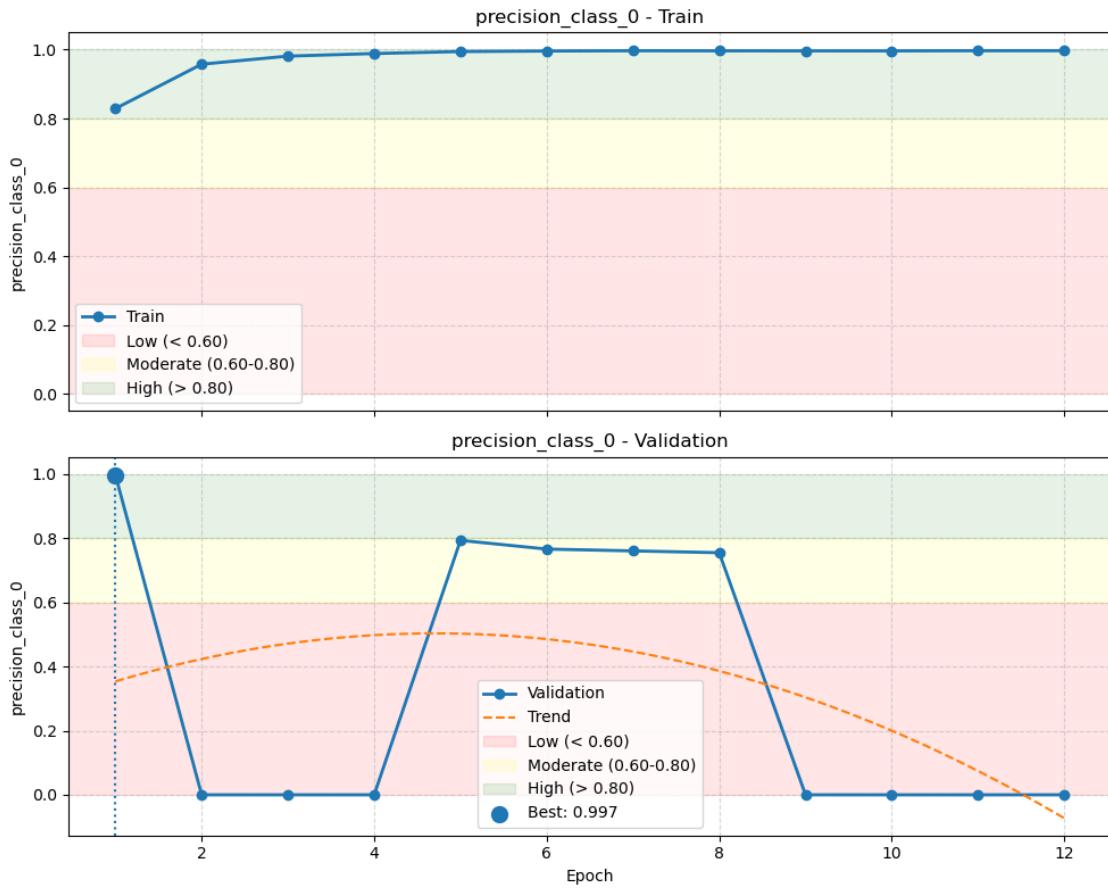
2 Per-Class Classification Metrics

2.1 Micro Aneurysms (MA)

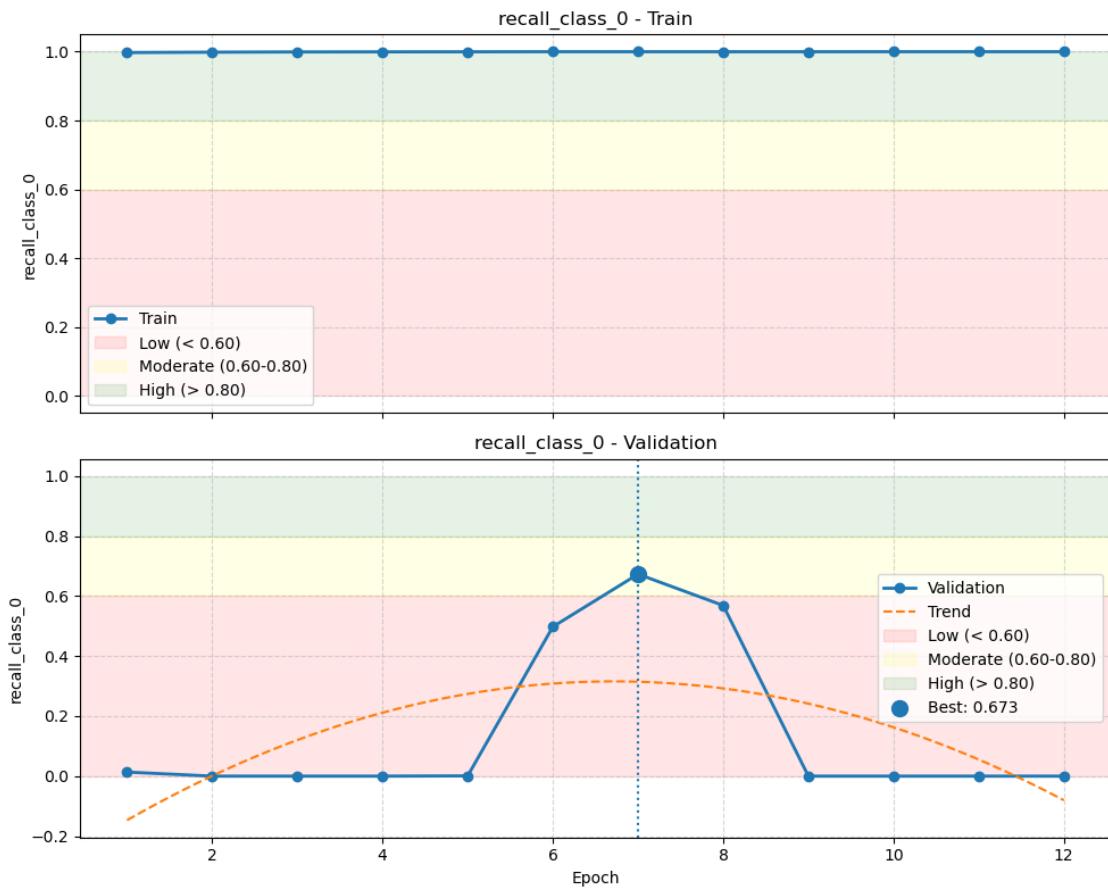
```
[151]: plot_train_val(df, "f1_class_0")
```



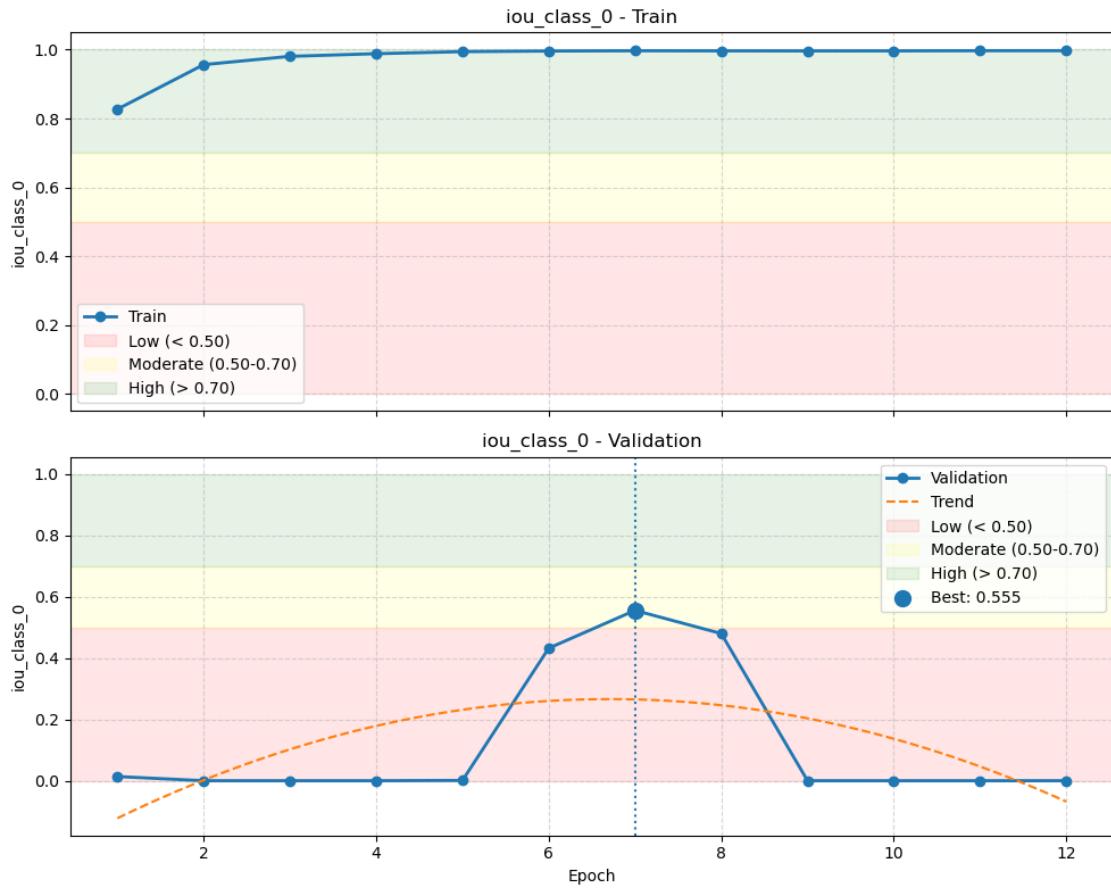
```
[152]: plot_train_val(df, "precision_class_0")
```



```
[153]: plot_train_val(df, "recall_class_0")
```

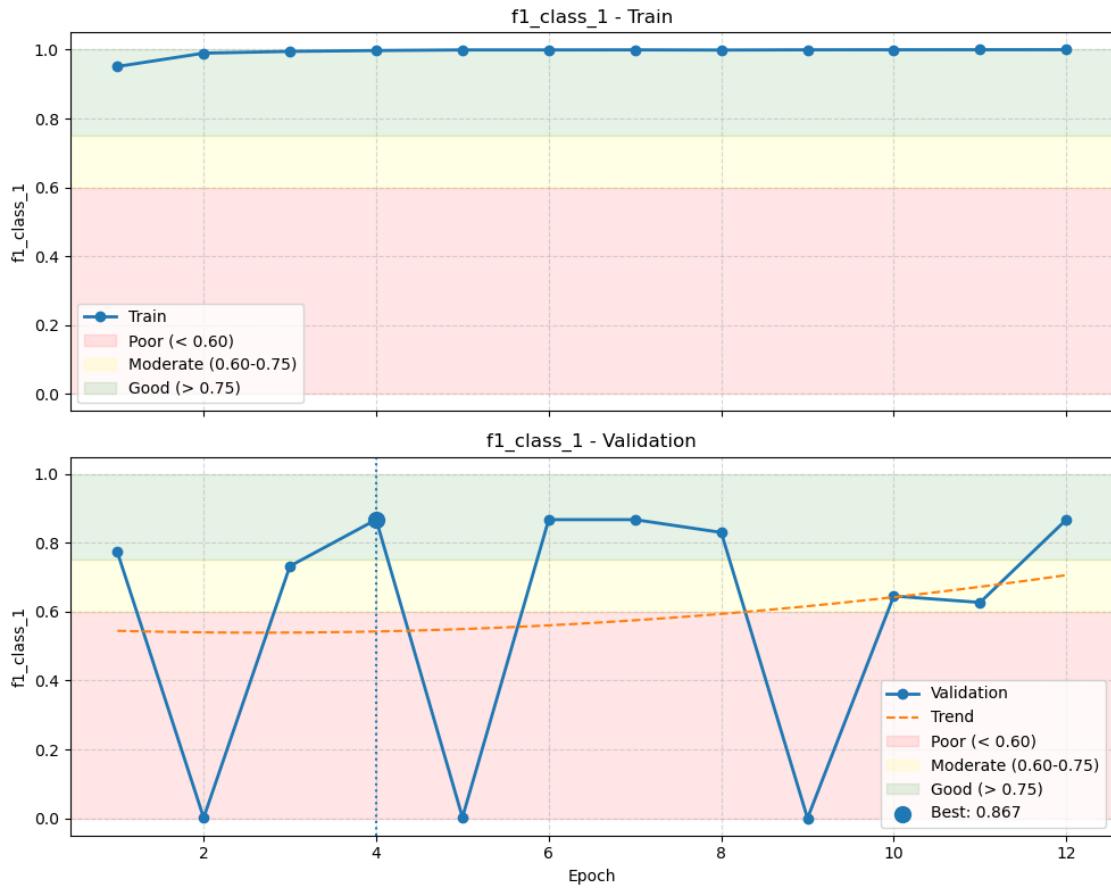


```
[154]: plot_train_val(df, "iou_class_0")
```

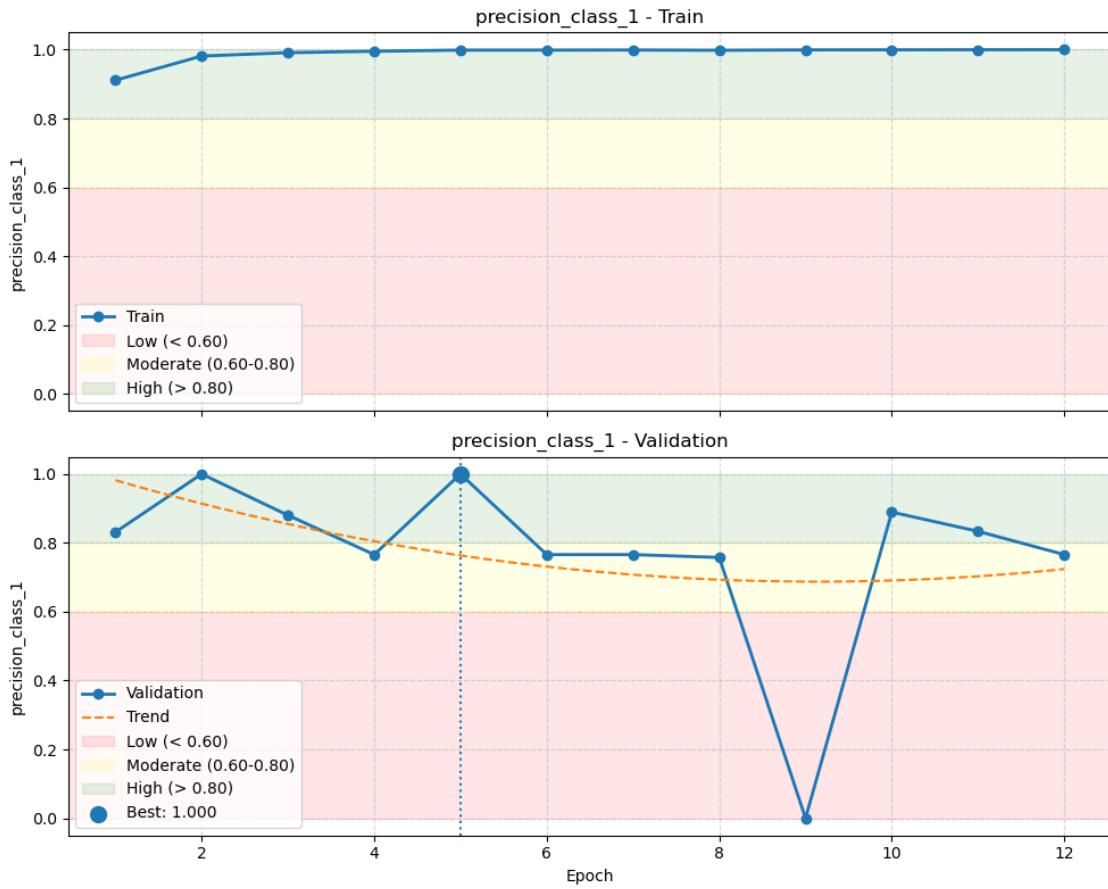


2.2 Hemorrhages (HE)

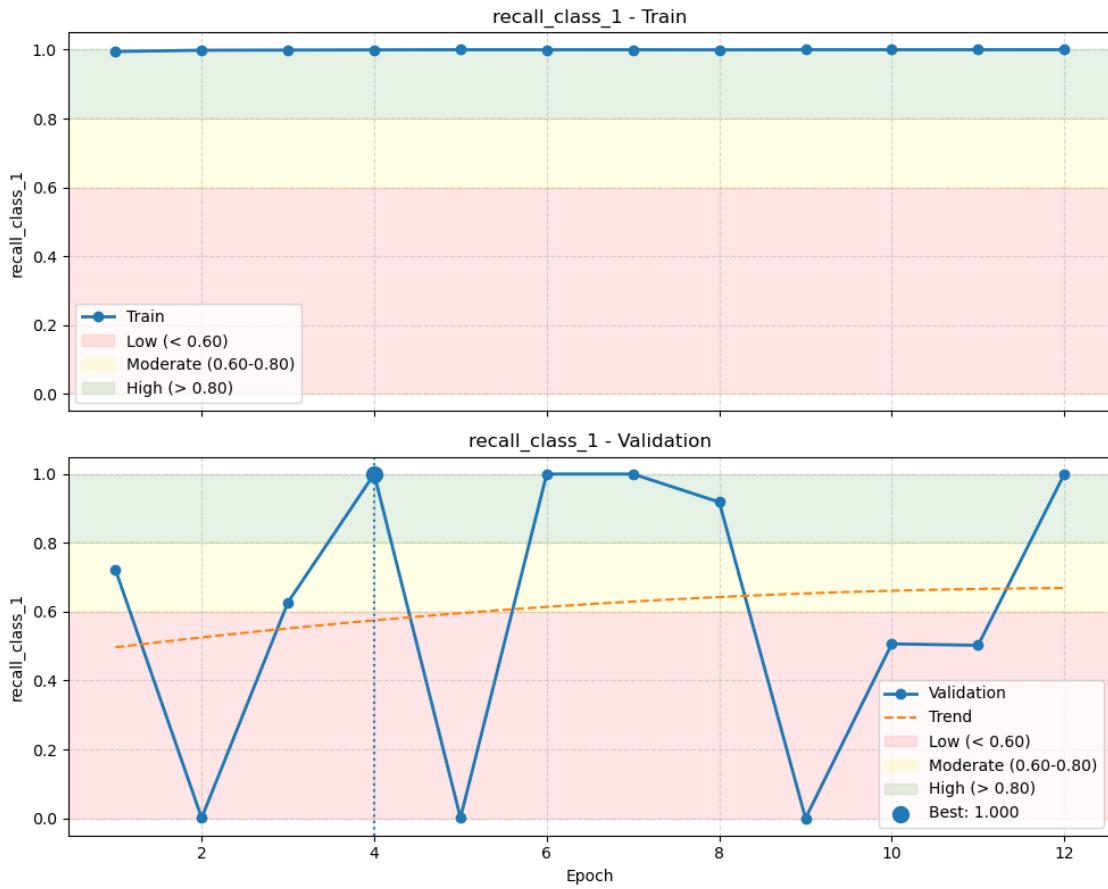
```
[155]: plot_train_val(df, "f1_class_1")
```



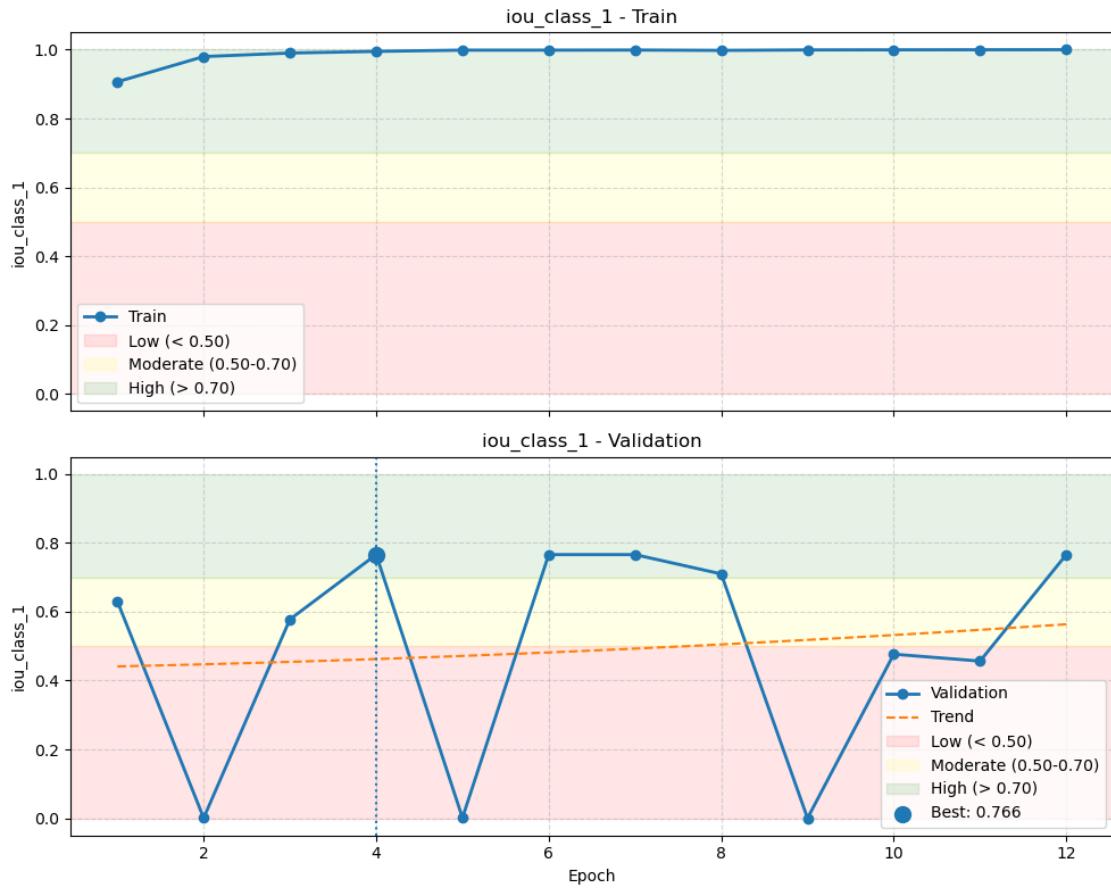
```
[156]: plot_train_val(df, "precision_class_1")
```



```
[157]: plot_train_val(df, "recall_class_1")
```

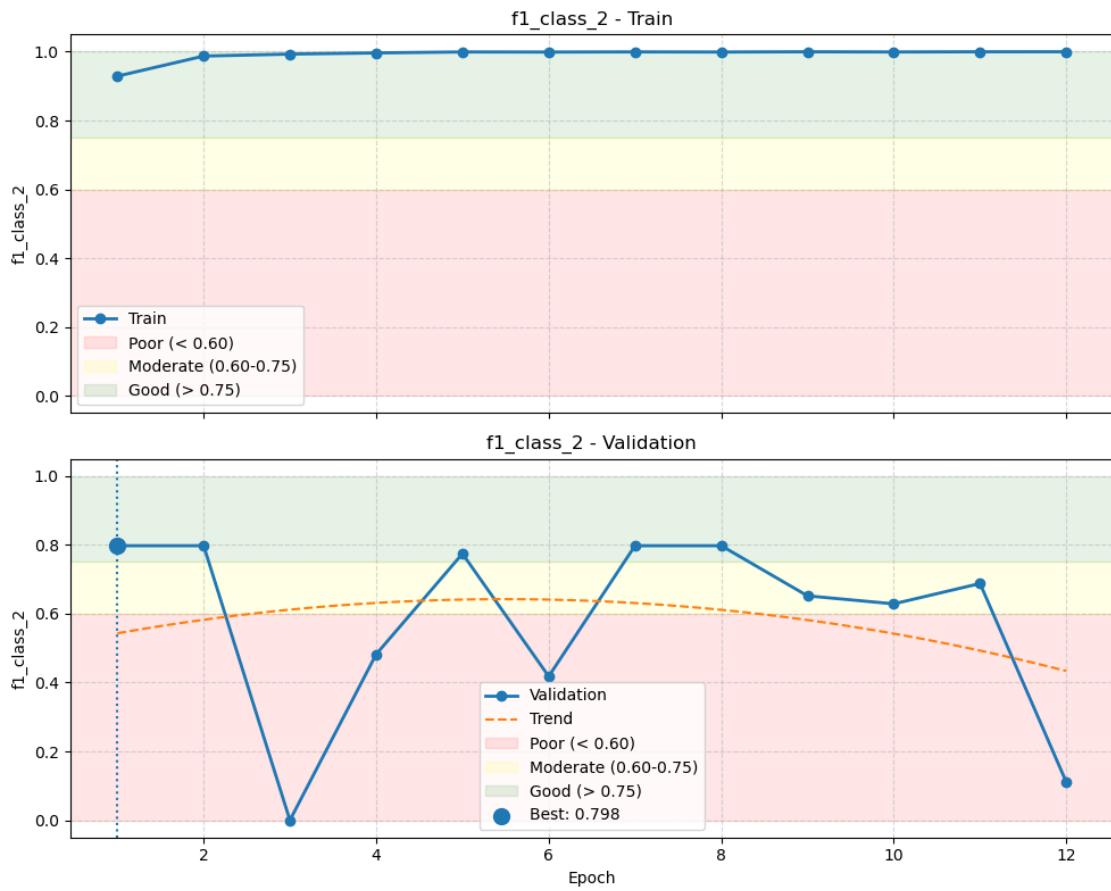


```
[158]: plot_train_val(df, "iou_class_1")
```

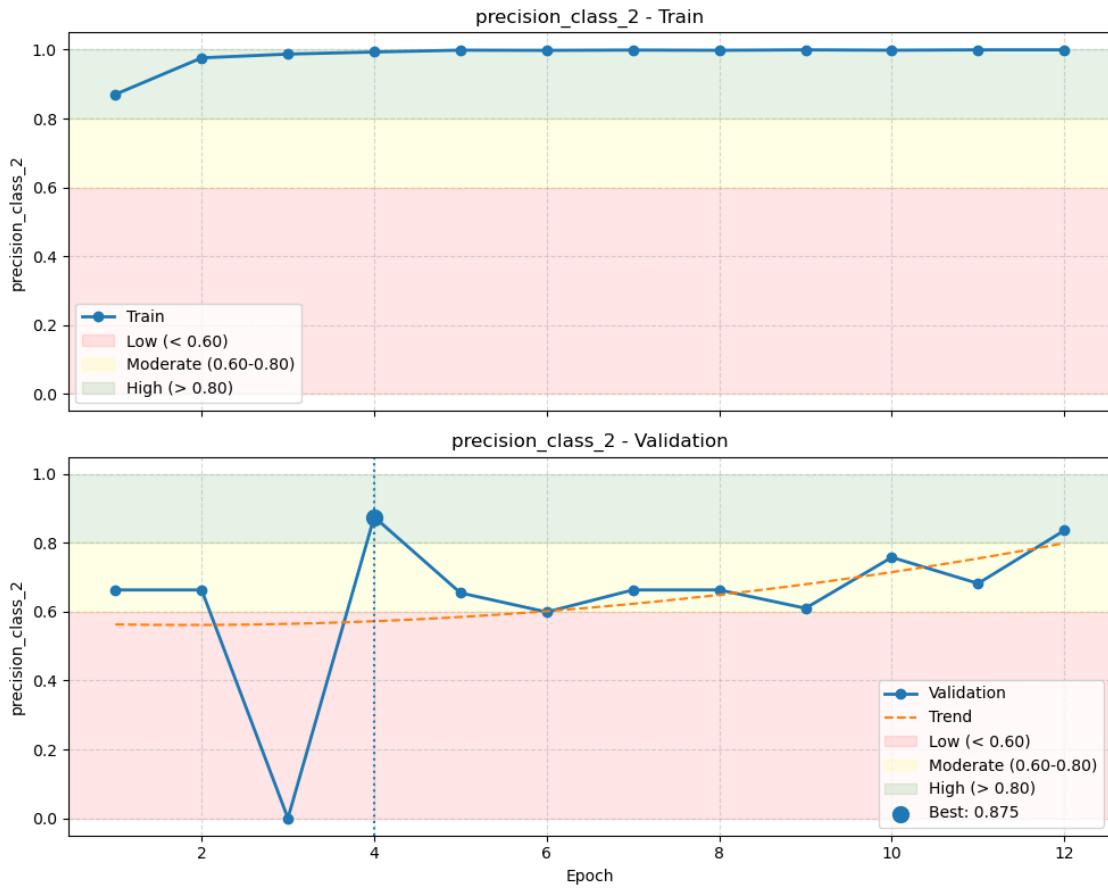


2.3 Hard Exudates (EX)

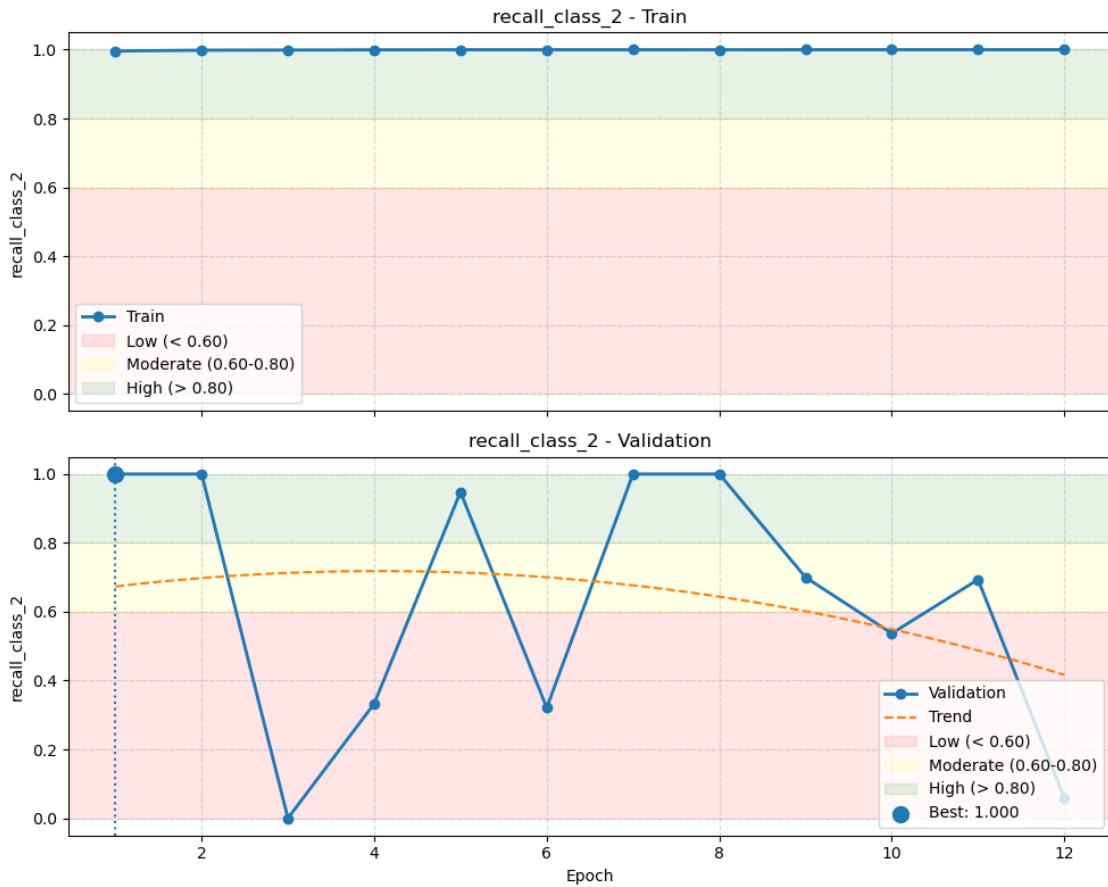
```
[159]: plot_train_val(df, "f1_class_2")
```



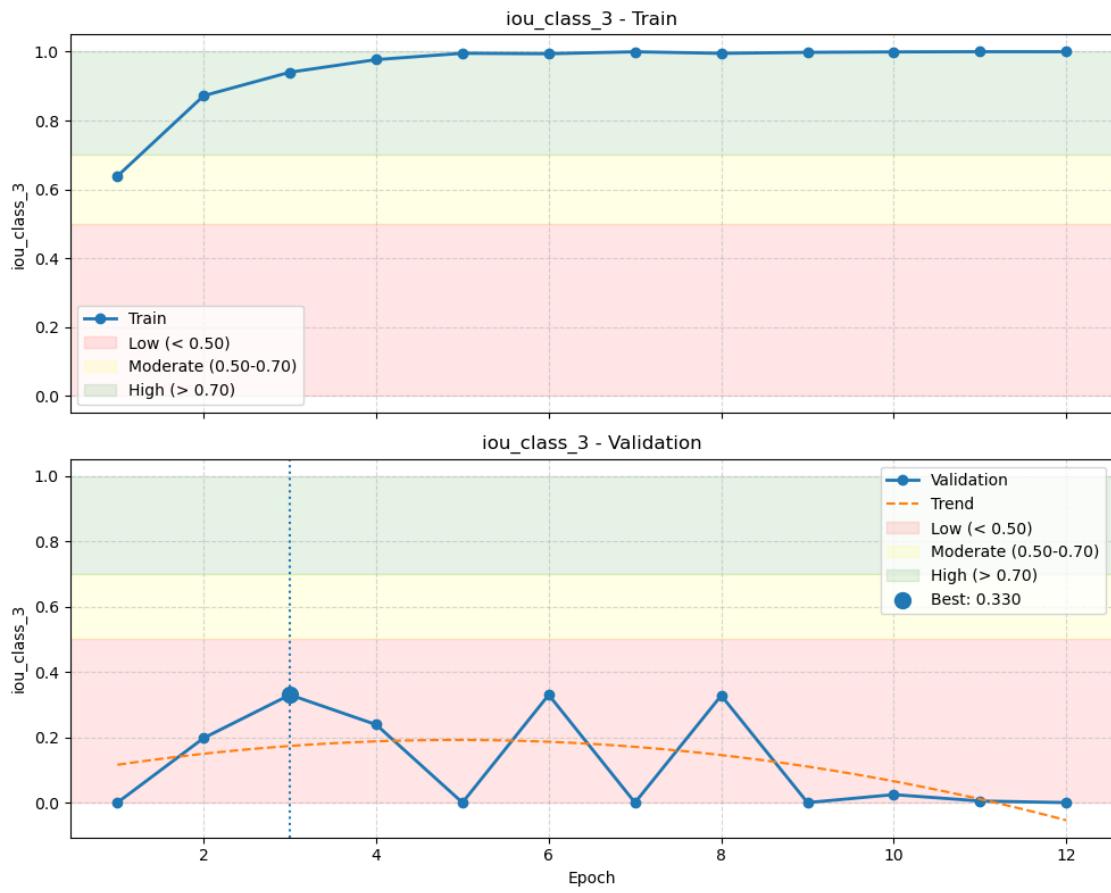
```
[160]: plot_train_val(df, "precision_class_2")
```



```
[161]: plot_train_val(df, "recall_class_2")
```

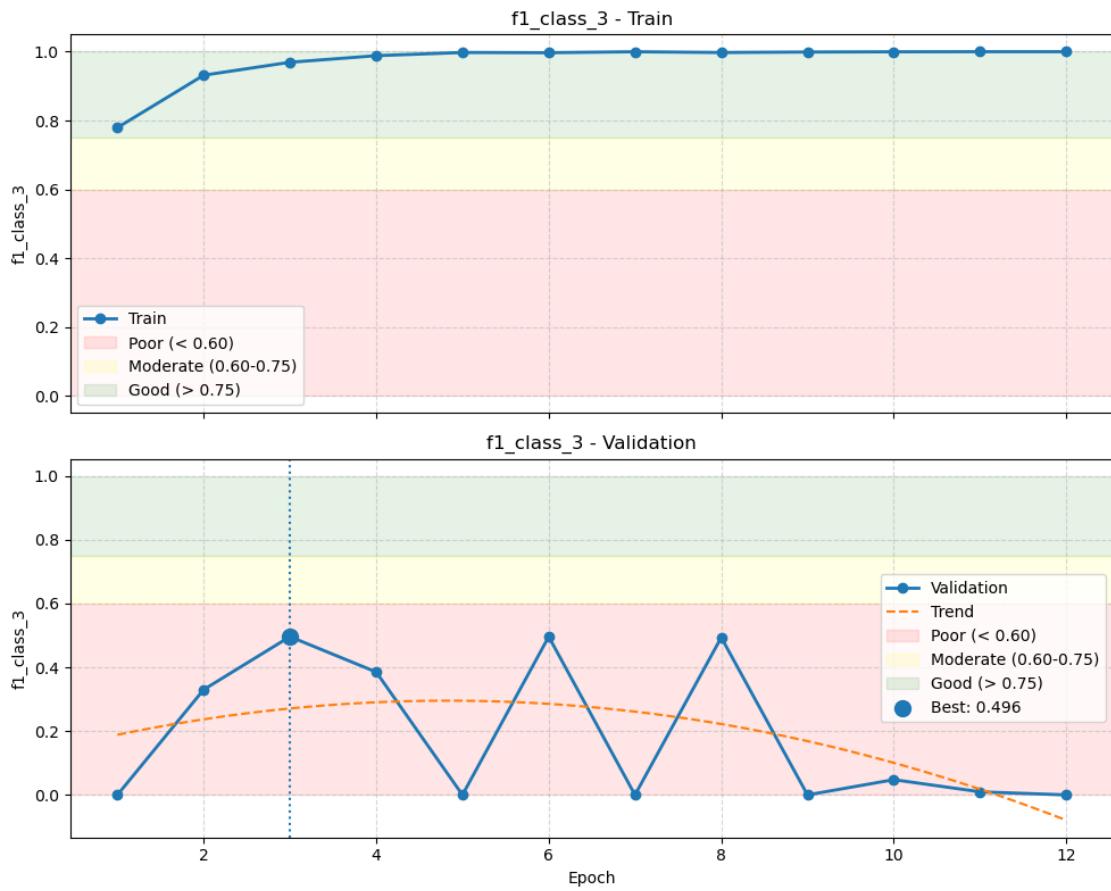


```
[162]: plot_train_val(df, "iou_class_3")
```

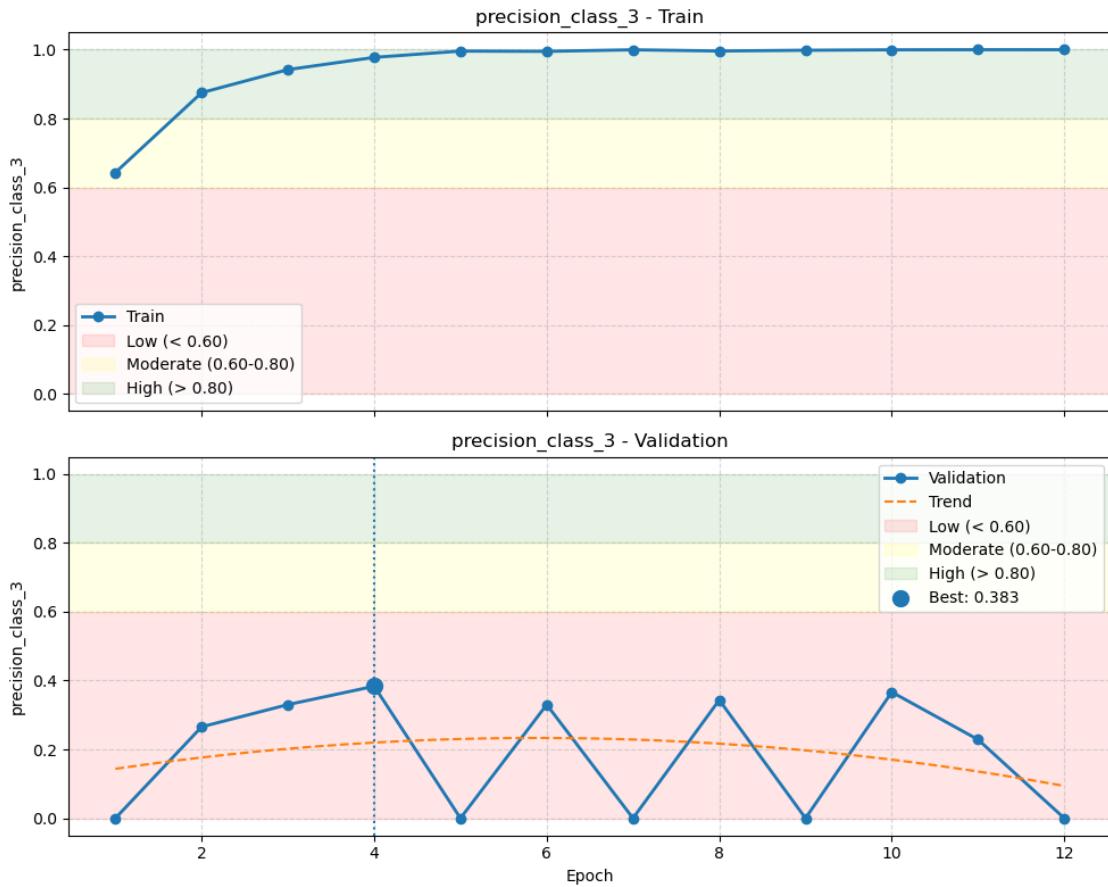


2.4 Soft Exudates (SE)

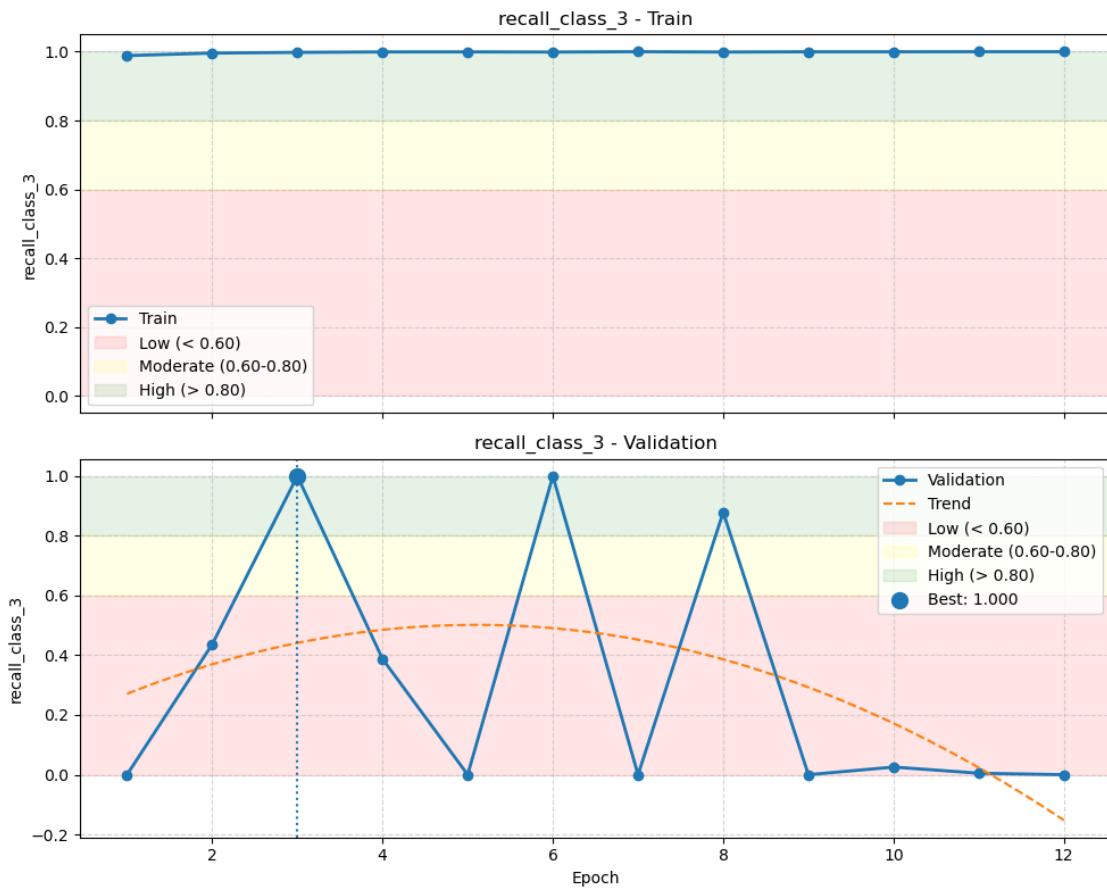
```
[163]: plot_train_val(df, "f1_class_3")
```



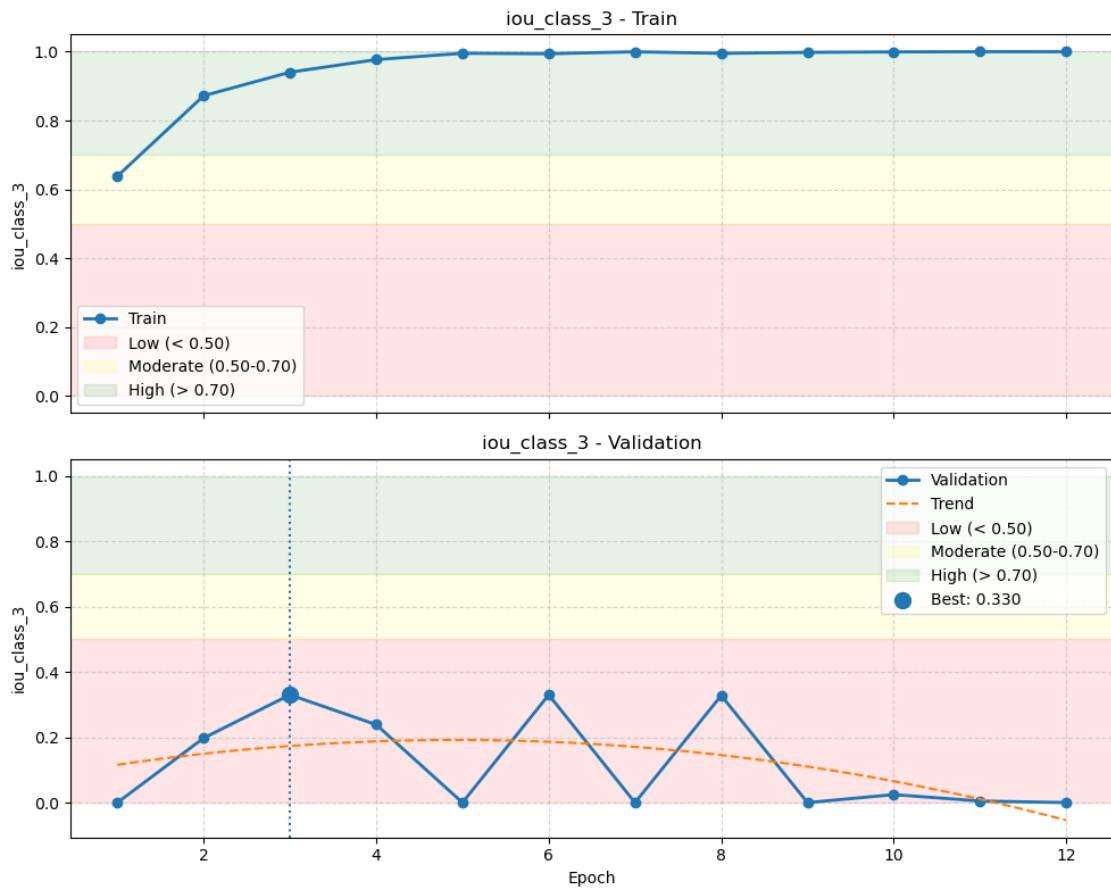
```
[164]: plot_train_val(df, "precision_class_3")
```



```
[165]: plot_train_val(df, "recall_class_3")
```



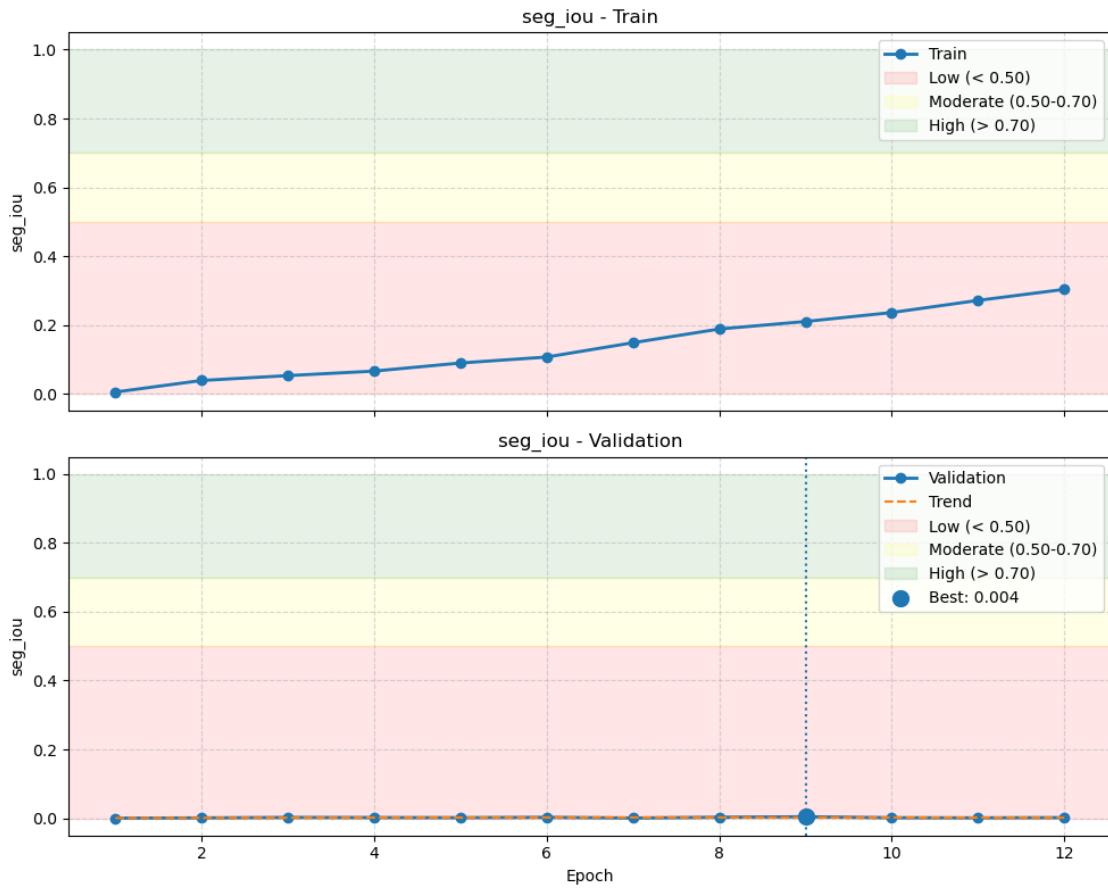
```
[166]: plot_train_val(df, "iou_class_3")
```



2.5 Segmentation Metrics

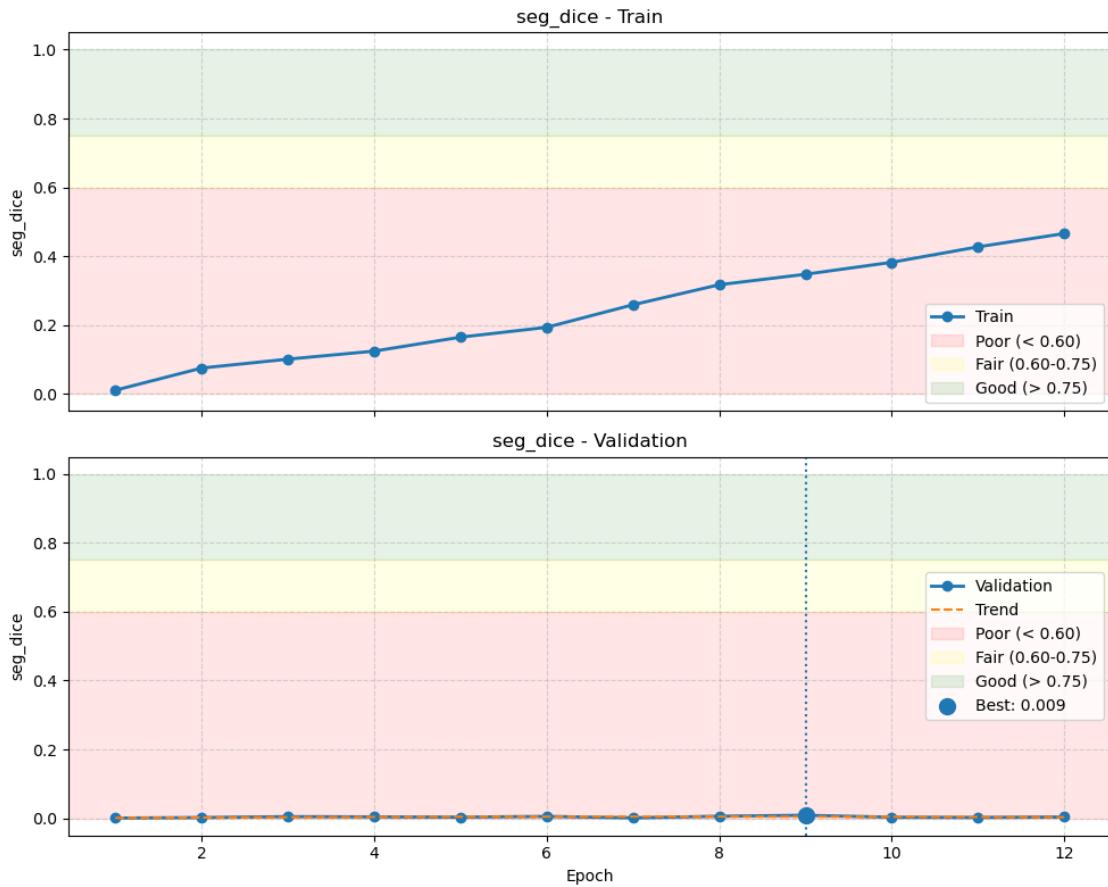
2.5.1 IoU

```
[167]: plot_train_val(df, "seg_iou")
```



2.6 Dice

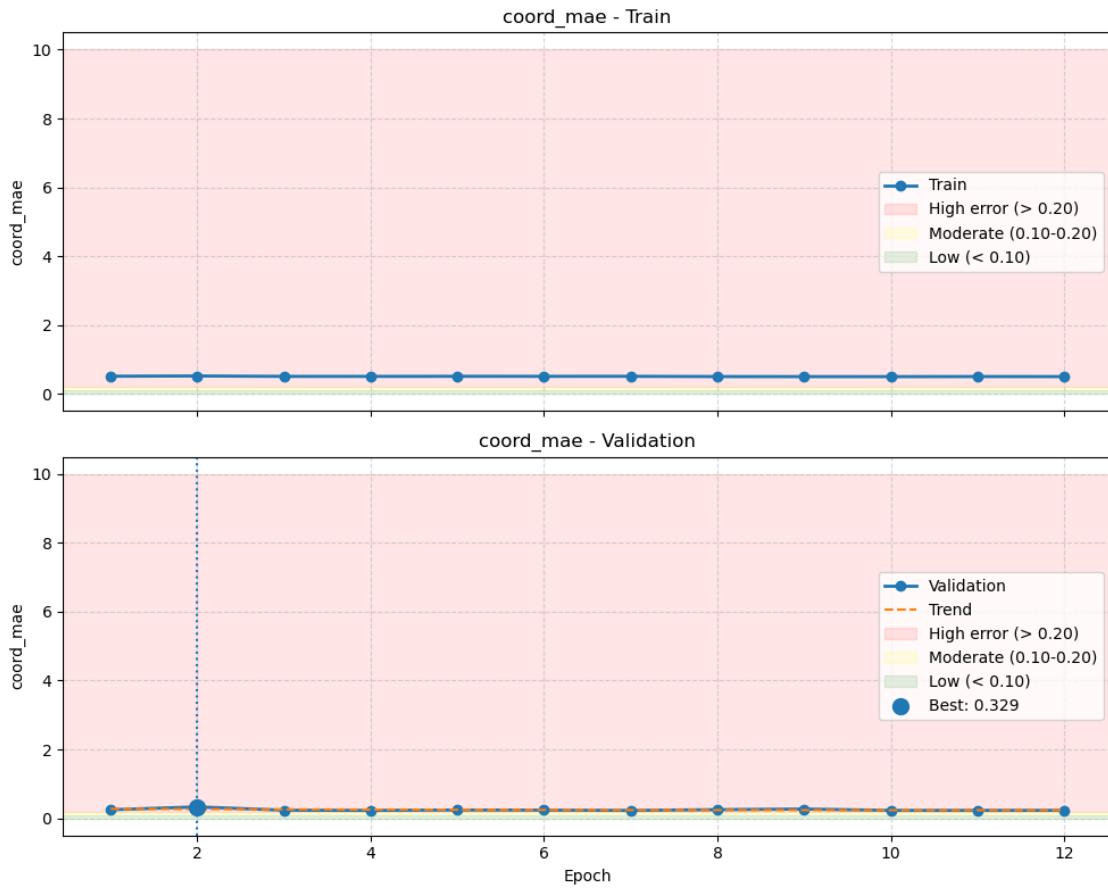
```
[168]: plot_train_val(df, "seg_dice")
```



3 Coordinate Regression Metrics

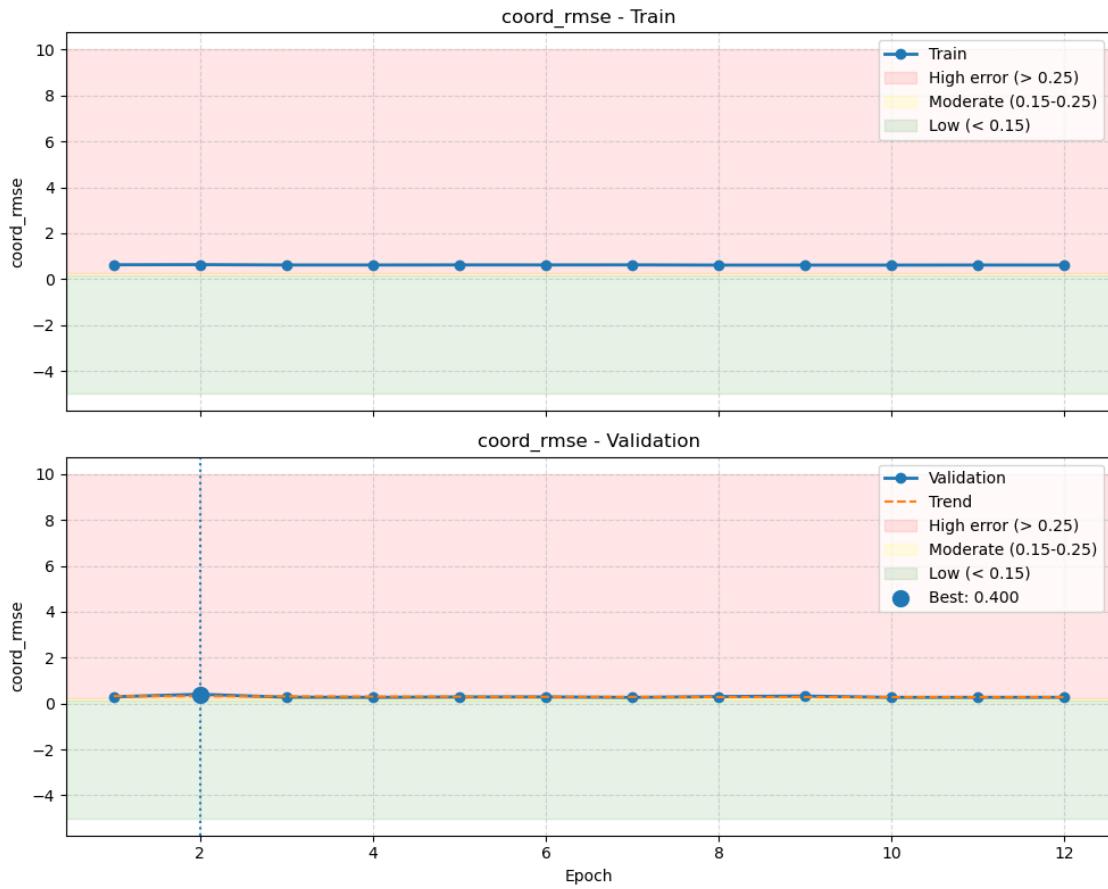
4 MAE

```
[169]: plot_train_val(df, "coord_mae")
```



4.1 RMSE

```
[170]: plot_train_val(df, "coord_rmse")
```



4.2 R^2

```
[171]: plot_train_val(df, "coord_r2")
```

