analysis

August 15, 2025

0.1 Hybrid Model Evaluation

Jakob Balkovec Fri, Aug 15th 2025

0.2 Data

[2]:
```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd

CSV_TRAIN_FILE = "/Users/jbalkovec/Desktop/DR/experiments/model_logs/
 ↪hybrid_train_history.csv"
CSV_VAL_FILE = "/Users/jbalkovec/Desktop/DR/experiments/model_logs/
 ↪hybrid_val_history.csv"
```

[3]:
```
train_df = pd.read_csv(CSV_TRAIN_FILE)
train_df.head()
```

[3]:

	loss	f1_micro	roc_auc	subset_acc	hamming	iou_score	dice_score	\
0	1.696087	0.542398	0	0.1	0.400	0.558333	0.689048	
1	1.659583	0.633307	0	0.2	0.350	0.600000	0.716667	
2	1.635090	0.828151	0	0.6	0.125	0.833333	0.890000	
3	1.604202	0.784636	0	0.4	0.150	0.783333	0.866667	
4	1.542188	0.797794	0	0.5	0.125	0.833333	0.900000	

	pearson_r	epoch
0	-0.160528	1
1	0.534618	2
2	0.693521	3
3	0.699530	4
4	0.739126	5

[4]:
```
val_df = pd.read_csv(CSV_VAL_FILE)
val_df.head()
```

[4]:

	loss	f1_micro	roc_auc	subset_acc	hamming	iou_score	dice_score	\
0	1.656705	0.542398	0	0.1	0.400	0.558333	0.689048	
1	1.661326	0.570175	0	0.2	0.325	0.625000	0.735714	

```
2  1.649466  0.584064        0         0.3    0.250   0.683333    0.782381
3  1.633023  0.584064        0         0.3    0.250   0.683333    0.782381
4  1.605927  0.459064        0         0.3    0.275   0.658333    0.763333

   pearson_r  epoch
0   0.367985      1
1   0.508813      2
2   0.602116      3
3   0.649964      4
4   0.676452      5
```

[20]:

```python
_METRIC_ALIASES = {
    "f1": "f1_micro",
    "f1_micro": "f1_micro",
    "acc": "subset_acc",
    "subset_acc": "subset_acc",
    "ham": "hamming",
    "hamming": "hamming",
    "loss": "loss",
    "r": "pearson_r",
    "pearson_r": "pearson_r",
    "dice": "dice_score",
    "dice_score": "dice_score",
    "iou": "iou_score",
    "iou_score": "iou_score",
    "auc": "roc_auc",
    "roc_auc": "roc_auc",
}

_DEFAULT_BANDS = {
    "f1_micro": [
        (0.0, 0.60, "red",    "Poor (< 0.60)"),
        (0.60, 0.75, "yellow","Moderate (0.60-0.75)"),
        (0.75, 1.00, "green", "Good (> 0.75)"),
    ],
    "subset_acc": [
        (0.0, 0.40, "red",    "Low (< 0.40)"),
        (0.40, 0.65, "yellow","Moderate (0.40-0.65)"),
        (0.65, 1.00, "green", "Good (> 0.65)"),
    ],
    "hamming": [
        (0.00, 0.15, "green", "Low Error (< 0.15)"),
        (0.15, 0.25, "yellow","Moderate Error (0.15-0.25)"),
        (0.25, 1.00, "red",   "High Error (> 0.25)"),
    ],
    "pearson_r": [
        (-1.00, 0.30, "red",    "Weak (< 0.30)"),
```

```
        (0.30, 0.60, "yellow","Moderate (0.30-0.60)"),
        (0.60, 1.00, "green", "Strong (> 0.60)"),
    ],
    "dice_score": [
        (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)"),
    ],
    "iou_score": [
        (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": [
        (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)"),
    ],
}

_BOUNDED_01 = {"f1_micro", "subset_acc", "hamming", "pearson_r", "dice_score",␣
 ↪"iou_score", "roc_auc"}

def _norm_metric_name(metric: str) -> str:
    key = metric.strip().lower()
    if key not in _METRIC_ALIASES:
        raise ValueError(f"Unknown metric '{metric}'. Known:␣
 ↪{sorted(_METRIC_ALIASES.keys())}")
    return _METRIC_ALIASES[key]

def _loss_bands_for_lambda(loss_lambda: float):
    """
    Heuristic bands for total_loss = BCEWithLogitsLoss +   * DiceLoss.
    Calibrated so that for  =1, 'moderate' covers roughly 0.9-1.8 and 'high' >1.
 ↪8.
    """
    # Base BCE bands (no segmentation): low<0.7, moderate 0.7-1.5, high>1.5
    # Shift upward by the expected segmentation contribution  0.4*  (mid␣
 ↪DiceLoss)
    low_thr  = 0.7 + 0.4 * loss_lambda
    high_thr = 1.5 + 0.4 * loss_lambda

    # Keep some headroom for rougher phases
    max_thr = high_thr + 1.0  # only for band shading upper bound

    return [
        (0.00,   low_thr,  "green",  f"Low (< {low_thr:.2f})"),
```

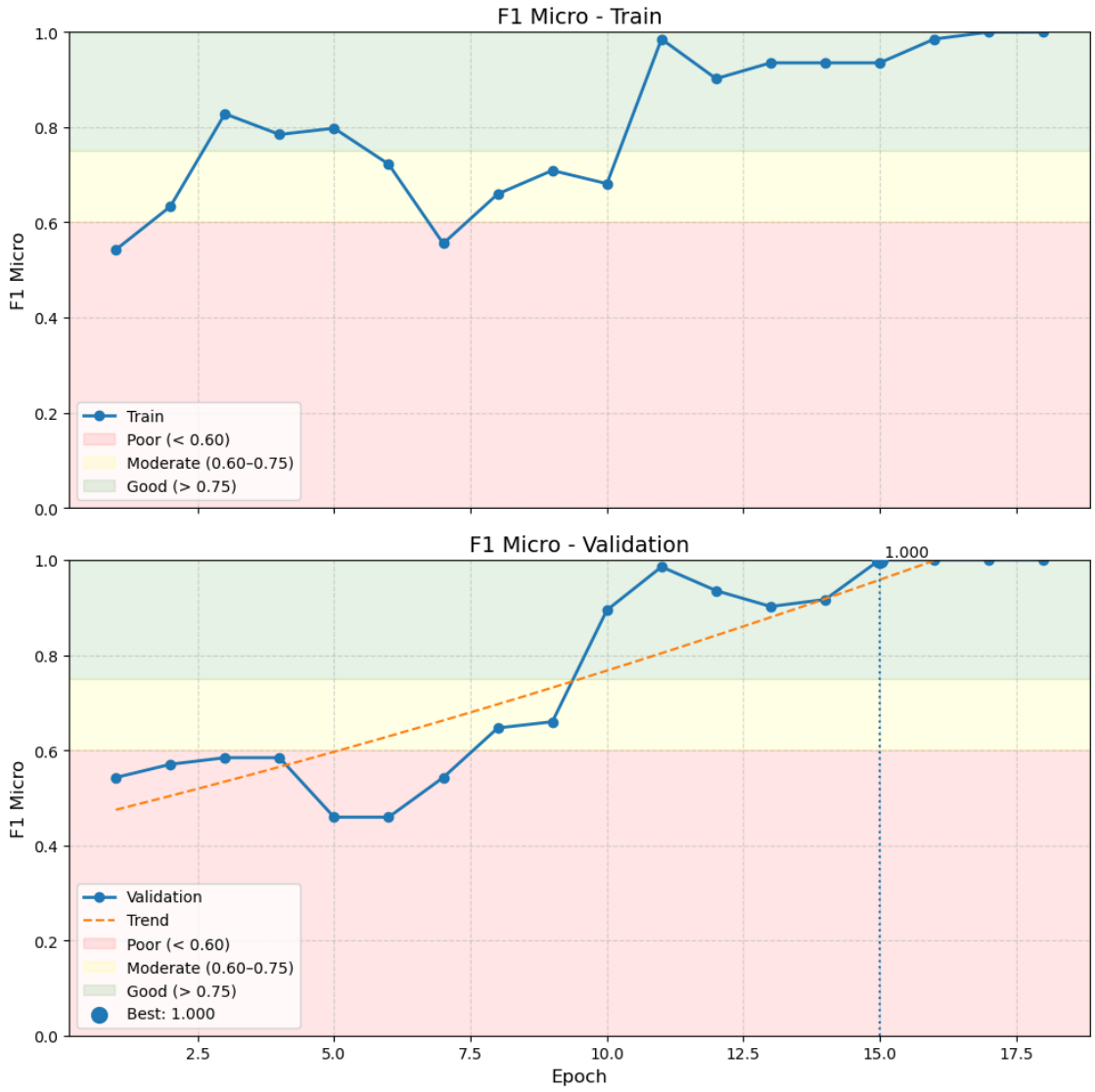
```
        (low_thr, high_thr,"yellow", f"Moderate ({low_thr:.2f}-{high_thr:.
↪2f})"),
        (high_thr, max_thr,"red",    f"High (> {high_thr:.2f})"),
    ]

def plot_metric(df_train,
                df_val,
                metric: str,
                *,
                bands: dict | None = None,
                show_trend: bool = True,
                loss_lambda: float = 1.0):

    # desc: plot a metric for train and validation using two DataFrames.

    # arg1: df_train, df_val - DataFrames with columns ['epoch', metric]
    # arg2: metric - name or alias (e.g., 'iou', 'dice', 'f1', 'acc', 'auc',␣
↪'r', 'hamming', 'loss')
    # arg3: bands - optional dict to override or extend threshold bands
    # arg4: show_trend - fit a quadratic trend on validation and draw it

    m = _norm_metric_name(metric)
    label = m.replace('_', ' ').title()

    if "epoch" not in df_train or "epoch" not in df_val:
        raise ValueError("Both DataFrames must include an 'epoch' column.")

    if m not in df_train or m not in df_val:
        raise ValueError(f"Metric '{m}' not found in DataFrames.")

    band_cfg = dict(_DEFAULT_BANDS)
    if bands:
        band_cfg.update(bands)

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

    ax_tr = axes[0]
    ax_tr.plot(df_train["epoch"], df_train[m], marker='o', label='Train',␣
↪linewidth=2)
    ax_tr.set_ylabel(label, fontsize=12)
    ax_tr.set_title(f"{label} - Train", fontsize=14)
    ax_tr.grid(True, linestyle='--', alpha=0.5)

    ax_va = axes[1]
    ax_va.plot(df_val["epoch"], df_val[m], marker='o', label='Validation',␣
↪linewidth=2)
    ax_va.set_xlabel("Epoch", fontsize=12)
```

```
    ax_va.set_ylabel(label, fontsize=12)
    ax_va.set_title(f"{label} - Validation", fontsize=14)
    ax_va.grid(True, linestyle='--', alpha=0.5)

    if show_trend and df_val[m].notna().sum() >= 3:
        try:
            z = np.polyfit(df_val["epoch"].to_numpy(), df_val[m].to_numpy(), 2)
            p = np.poly1d(z)
            x_vals = np.linspace(df_val["epoch"].min(), df_val["epoch"].max(),␣
↪200)
            ax_va.plot(x_vals, p(x_vals), "--", label="Trend")
        except Exception:
            pass  # be robust

    def draw_bands(ax):

        if m == "loss":
            for lo, hi, color, lab in _loss_bands_for_lambda(loss_lambda):
                ax.axhspan(lo, hi, color=color, alpha=0.10, label=lab)
            return

        if m in band_cfg:
            for lo, hi, color, lab in band_cfg[m]:
                ax.axhspan(lo, hi, color=color, alpha=0.10, label=lab)

    draw_bands(ax_tr)
    draw_bands(ax_va)

    if m in _BOUNDED_01:
        ax_tr.set_ylim(0.0, 1.0)
        ax_va.set_ylim(0.0, 1.0)
    elif m == "pearson_r":
        ax_tr.set_ylim(-1.0, 1.0)
        ax_va.set_ylim(-1.0, 1.0)
    elif m == "loss":
        # auto with margin
        for ax in (ax_tr, ax_va):
            vals = np.concatenate([df_train[m].dropna().to_numpy(), df_val[m].
↪dropna().to_numpy()])
            if vals.size:
                lo, hi = float(np.nanmin(vals)), float(np.nanmax(vals))
                span = max(1e-6, hi - lo)
                ax.set_ylim(max(0.0, lo - 0.1 * span), hi + 0.1 * span)

    val_clean = df_val[df_val[m].notna()]
    if not val_clean.empty:
        if m in {"loss", "hamming"}:
```

```
            best_idx = val_clean[m].idxmin()
        else:
            best_idx = val_clean[m].idxmax()
        best_epoch = val_clean.loc[best_idx, "epoch"]
        best_value = val_clean.loc[best_idx, m]
        ax_va.scatter(best_epoch, best_value, s=100, zorder=5, label=f"Best:␣
↪{best_value:.3f}")
        ax_va.axvline(best_epoch, linestyle=":", linewidth=1.5)
        ax_va.text(best_epoch, best_value, f" {best_value:.3f}", fontsize=10,␣
↪va="bottom")

    def dedup_legend(ax):
        handles, labels = ax.get_legend_handles_labels()
        seen = {}
        new_h, new_l = [], []
        for h, l in zip(handles, labels):
            if l not in seen:
                seen[l] = True
                new_h.append(h)
                new_l.append(l)
        ax.legend(new_h, new_l, loc="best")

    dedup_legend(ax_tr)
    dedup_legend(ax_va)

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

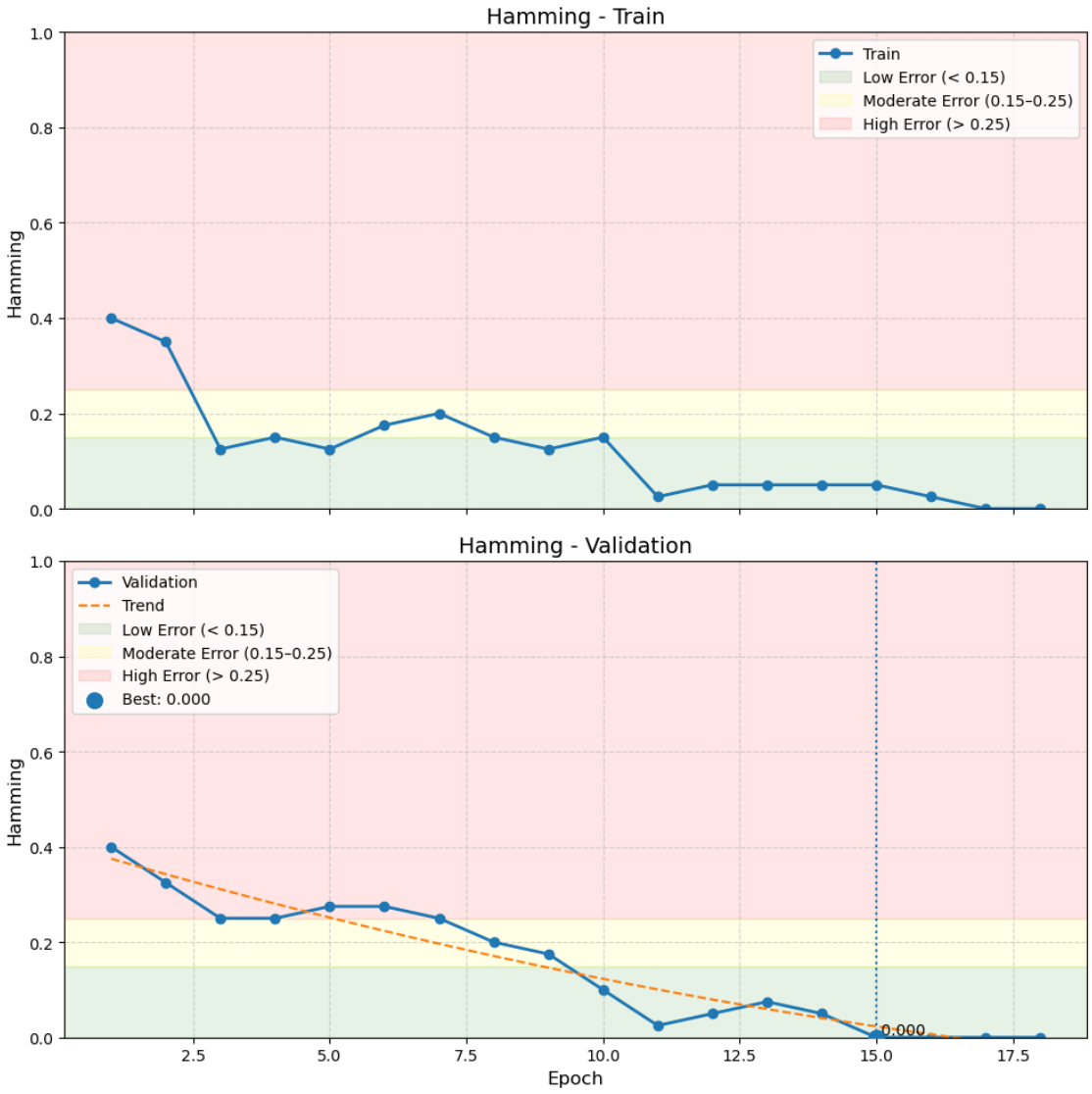
0.3 F1

[ ]:
```
plot_metric(train_df, val_df, "f1")
```



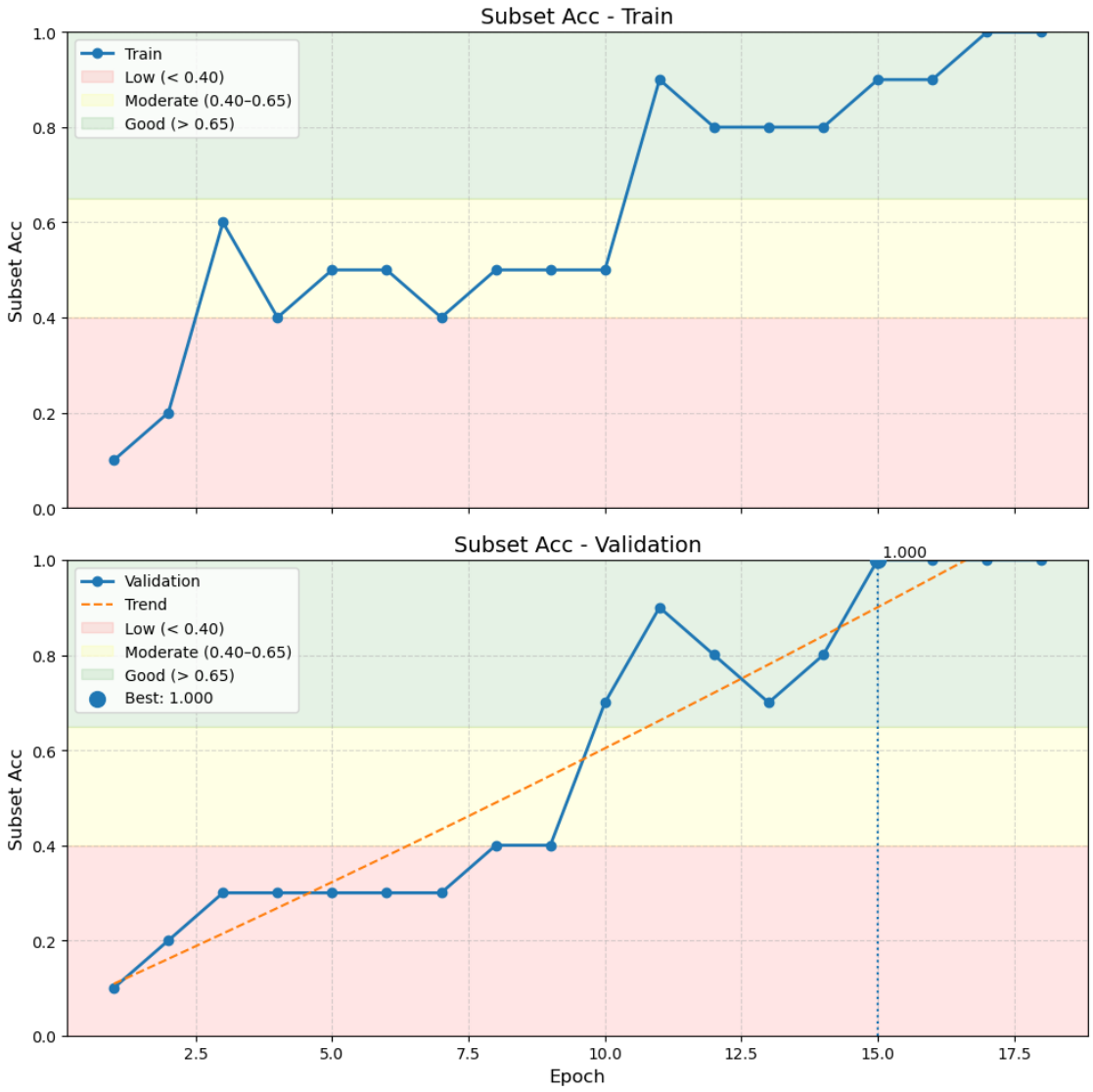
0.4 Hamming Loss

[9]:
```
plot_metric(train_df, val_df, "hamming")
```



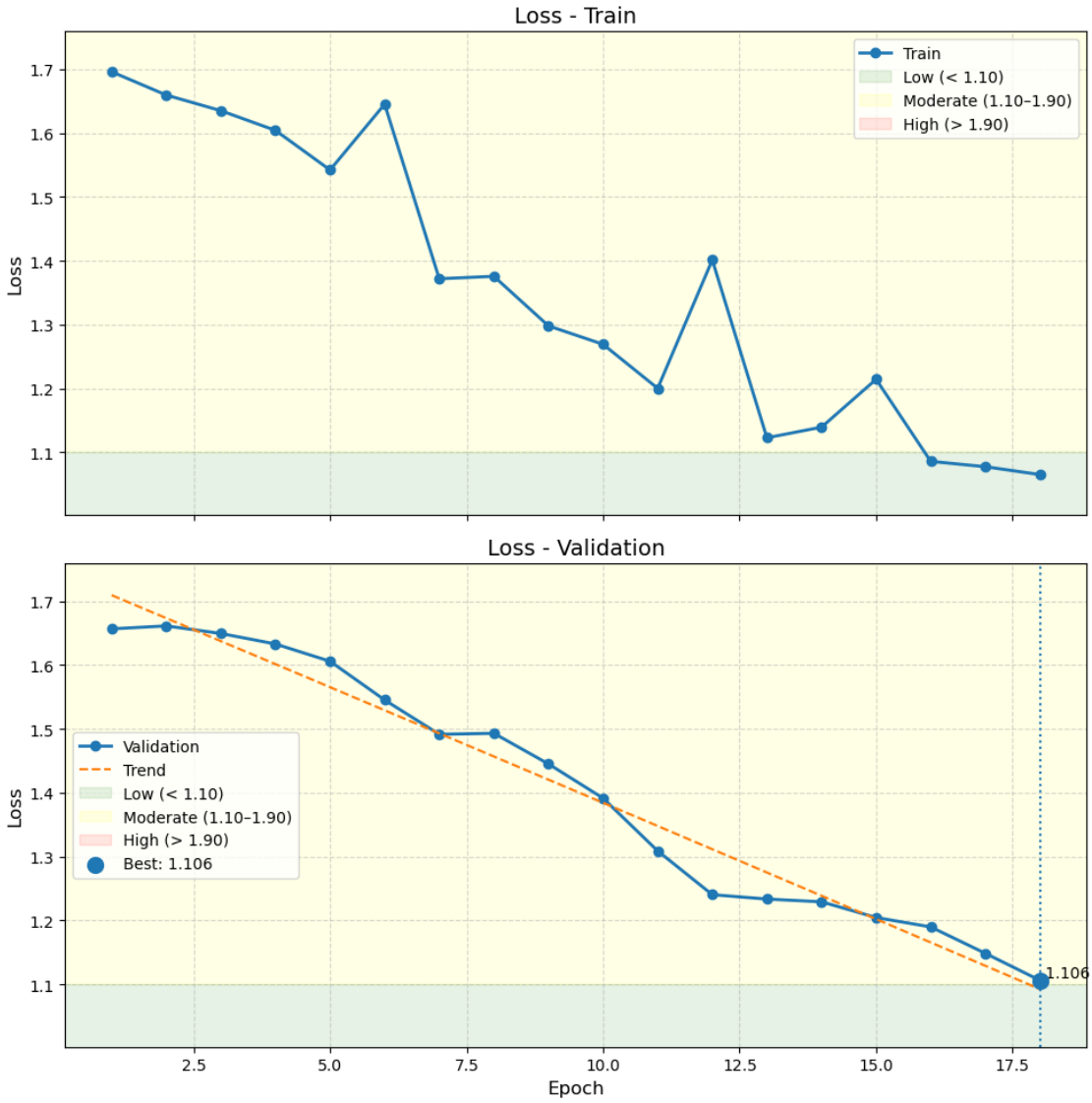
0.5 Subset Accuracy

[10]:
```
plot_metric(train_df, val_df, "subset_acc")
```



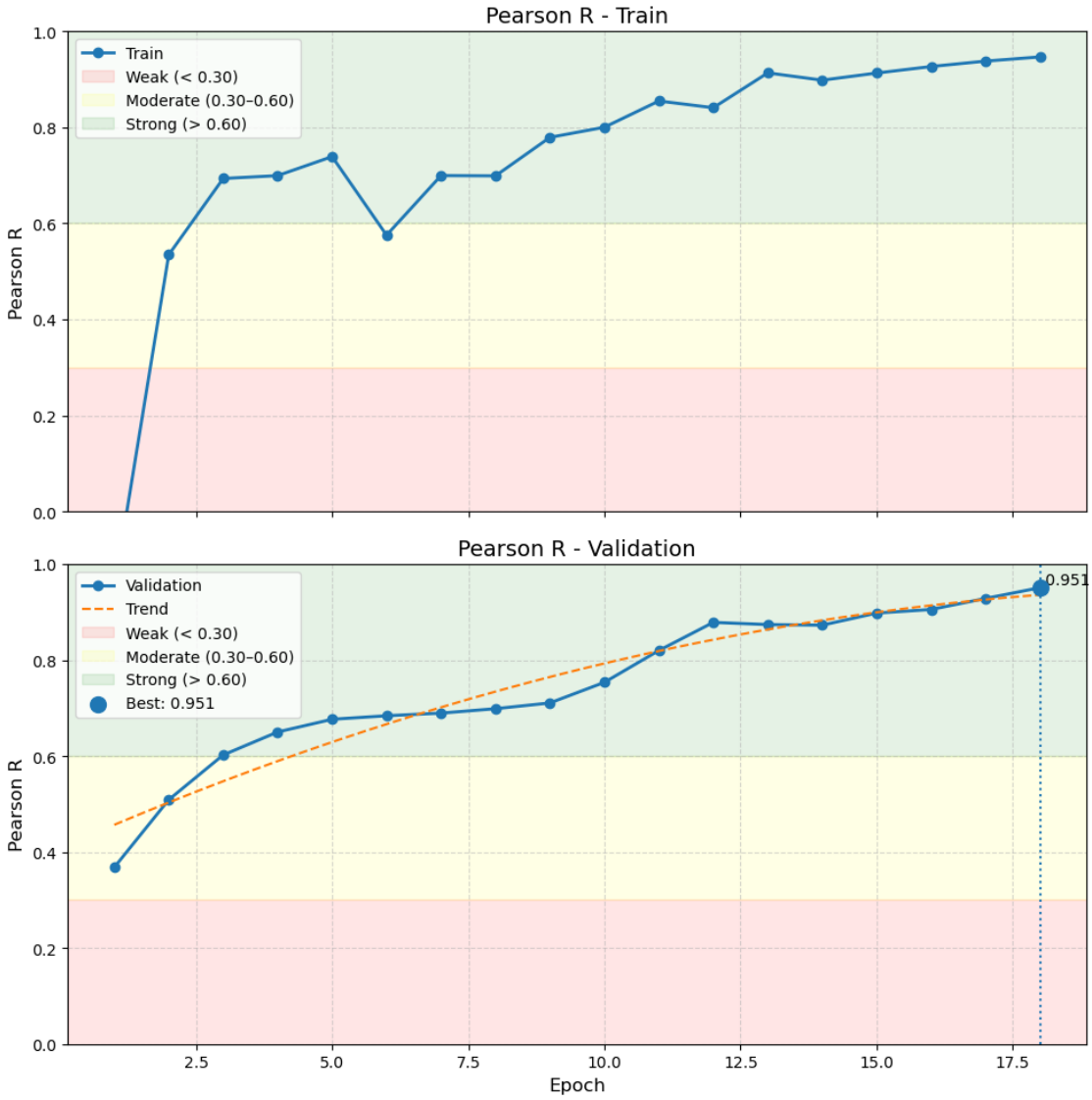
0.6 Loss

[21]:
```
plot_metric(train_df, val_df, "loss")
```



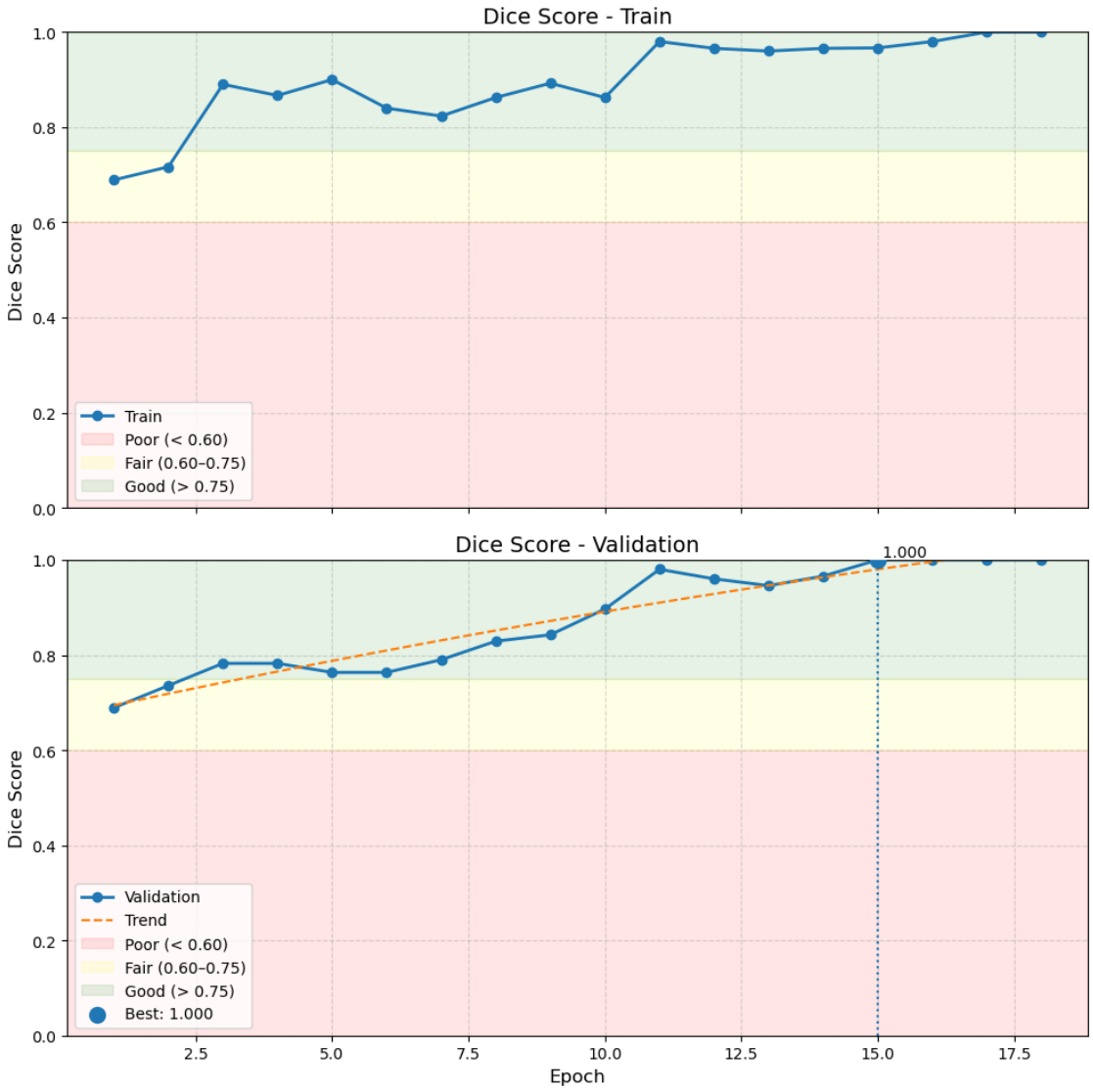
0.7 Pearson Correlation (R)

[12]:
```
plot_metric(train_df, val_df, "r")
```



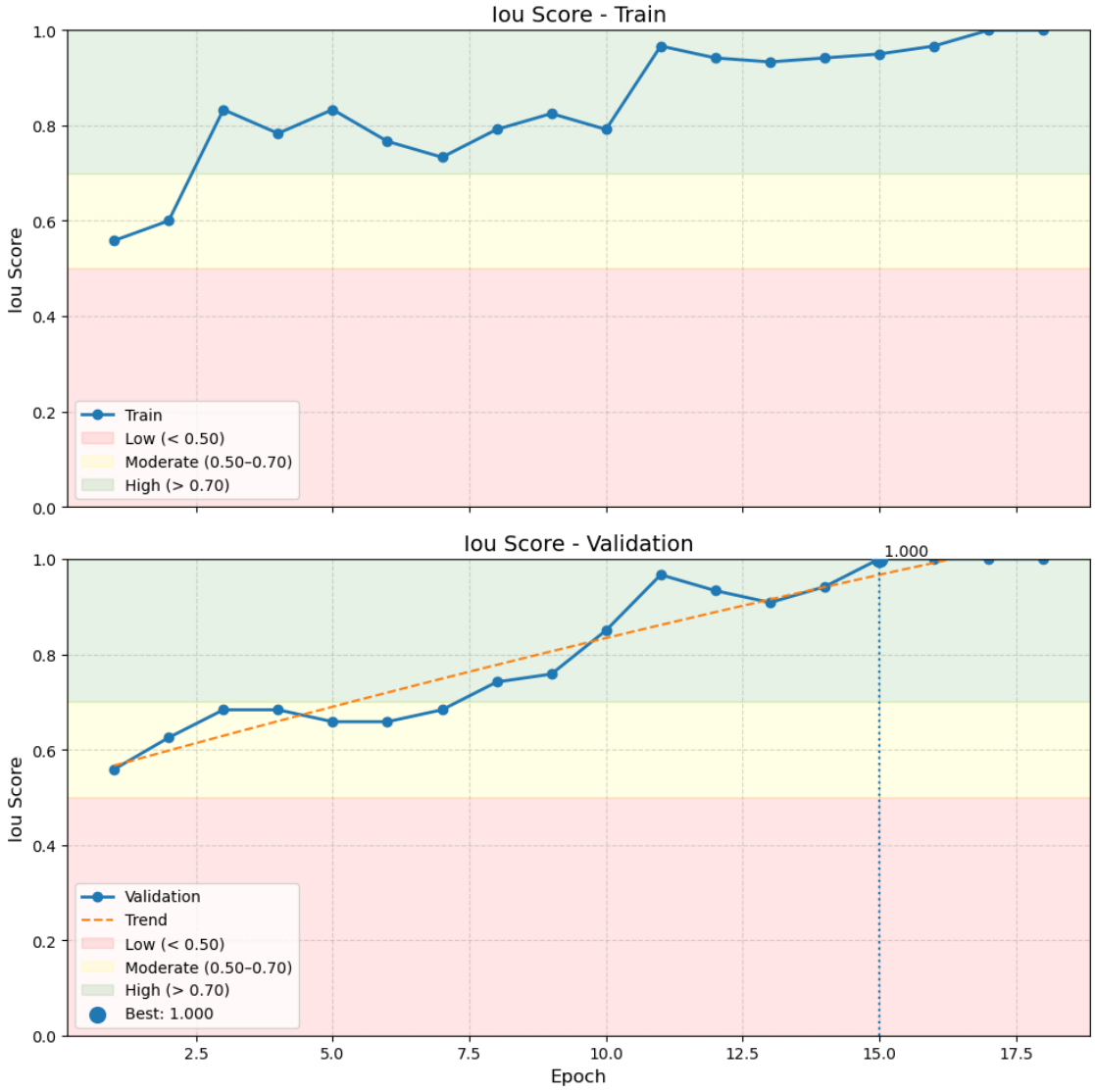
0.8 Dice

[13]:
```
plot_metric(train_df, val_df, "dice")
```



0.9 IoU

[14]:
```
plot_metric(train_df, val_df, "iou")
```


Iou Score - Train
Iou Score
Train
Low (< 0.50)
Moderate (0.50–0.70)
High (> 0.70)
Iou Score - Validation
1.000
Iou Score
Epoch
Validation
Trend
Low (< 0.50)
Moderate (0.50–0.70)
High (> 0.70)
Best: 1.000