

Model Performance Analysis

August 5, 2025

0.1 Hybrid Model Evaluation

Jakob Balkovec Tue, Aug 5th 2025

This document evaluates the performance of the hybrid model, solely on metrics/numbers I got whilst training it. The model was trained on a dataset of 1841 images, and 184100 patches. Due to the stack mismatch issue, I didn't get a chance to filter out the noise (black patches) but I think the results are still quite promising.

The model is a dual-branch DL architecture (global and local), it leverages two ResNet18 backbones: one processes the full fundus image to extract global context, the other encodes localized lesion patches.

The output features from each branch are then projected into a shared latent space via fully connected layers and fused using a multi-head self-attention mechanism...allowing the model to align and weigh patch-level signals relative to the image context.

The fused representation is passed through a lightweight MLP classifier with a sigmoid activation to predict the presence of four lesion types.

0.2 Training

The model was trained with a learning rate of 0.0001, and I used the Adam optimizer (work to be done here). The training loss decreased steadily over time, and the validation loss also showed a similar trend.

Since I couldn't access Texas A&M's GPU cluster, I had to train the model on Google Colab. I got booted off after about 9.5 epochs, but f1 was still improving, meaning there is performance left on the table.

I designed a new training function that allows for early stopping as well as continuing training from a checkpoint. This way, I can resume training on the cluster without losing progress. I haven't tested any of this yet, but I'm hoping it works...

Google Colab for Education is not available in my country. I emailed Google support, but they said they can't help me. I'll make the most of what I have...

0.3 Data

```
[5]: import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

```

import pandas as pd

CSV_FILE = "/Users/jbalkovec/Desktop/DR/experiments/model_logs/hybrid_model_log.
˓→CSV"

```

[14]: df = pd.read_csv(CSV_FILE)
df.head(n=len(df))

[14]:

	timestamp	epoch	f1_micro	hamming	subset_acc	loss
0	2025-08-04 16:43:37	1	0.740259	0.252307	0.302932	0.515967
1	2025-08-04 16:51:09	1	0.658971	0.232763	0.337676	NaN
2	2025-08-04 17:01:44	2	0.759698	0.225841	0.354506	0.470941
3	2025-08-04 17:09:43	2	0.742369	0.206433	0.396308	NaN
4	2025-08-04 17:20:31	3	0.777327	0.206298	0.386536	0.440251
5	2025-08-04 17:28:39	3	0.818846	0.185532	0.425081	NaN
6	2025-08-04 17:39:26	4	0.789693	0.195847	0.409338	0.414448
7	2025-08-04 17:47:38	4	0.823207	0.168838	0.459826	NaN
8	2025-08-04 17:58:20	5	0.792866	0.187704	0.431053	0.406103
9	2025-08-04 18:06:23	5	0.858121	0.147394	0.527687	NaN
10	2025-08-04 18:16:45	6	0.813307	0.178339	0.447883	0.383413
11	2025-08-04 18:24:47	6	0.783430	0.191775	0.426167	NaN
12	2025-08-04 18:35:23	7	0.839157	0.156488	0.513029	0.345170
13	2025-08-04 18:43:39	7	0.868800	0.141151	0.548317	NaN
14	2025-08-04 18:54:12	8	0.845825	0.146851	0.530402	0.323287
15	2025-08-04 19:02:19	8	0.888501	0.113871	0.629750	NaN
16	2025-08-04 19:12:45	9	0.873666	0.123507	0.592291	0.279986
17	2025-08-04 19:20:51	9	0.905516	0.100027	0.659609	NaN

[15]:

```

def plot_metric(df, metric: str):
    if metric not in df.columns or 'epoch' not in df.columns:
        raise ValueError("DataFrame must contain 'epoch' and the requested
˓→metric column.")

    df_train = df.iloc[:2].copy()
    df_val = df.iloc[1::2].copy()

    plt.figure(figsize=(10, 6))
    plt.plot(df_train["epoch"], df_train[metric], marker='o', label="Train",
˓→linewidth=2, color='tab:blue')
    plt.plot(df_val["epoch"], df_val[metric], marker='o', label="Validation",
˓→linewidth=2, color='tab:orange')

    # trend line (optional: only on val)
    if len(df_val["epoch"].unique()) >= 3:
        z = np.polyfit(df_val["epoch"], df_val[metric], 2)
        p = np.poly1d(z)
        x_vals = np.linspace(df_val["epoch"].min(), df_val["epoch"].max(), 100)

```

```

plt.plot(x_vals, p(x_vals), "--", color="gray", label="Val Trend Line")

# thresholds
if metric == 'f1_micro':
    plt.axhspan(0.0, 0.6, color='red', alpha=0.1, label="Poor (< 0.6)")
    plt.axhspan(0.6, 0.75, color='yellow', alpha=0.1, label="Moderate (0.6-0.75)")
    plt.axhspan(0.75, 1.0, color='green', alpha=0.1, label="Good (> 0.75)")

elif metric == 'subset_acc':
    plt.axhspan(0.0, 0.4, color='red', alpha=0.1, label="Low (< 0.4)")
    plt.axhspan(0.4, 0.65, color='yellow', alpha=0.1, label="Moderate (0.4-0.65)")
    plt.axhspan(0.65, 1.0, color='green', alpha=0.1, label="Good (> 0.65)")

elif metric == 'hamming':
    plt.axhspan(0.0, 0.15, color='green', alpha=0.1, label="Low Error (< 0.15)")
    plt.axhspan(0.15, 0.25, color='yellow', alpha=0.1, label="Moderate Error (0.15-0.25)")
    plt.axhspan(0.25, 1.0, color='red', alpha=0.1, label="High Error (> 0.25)")

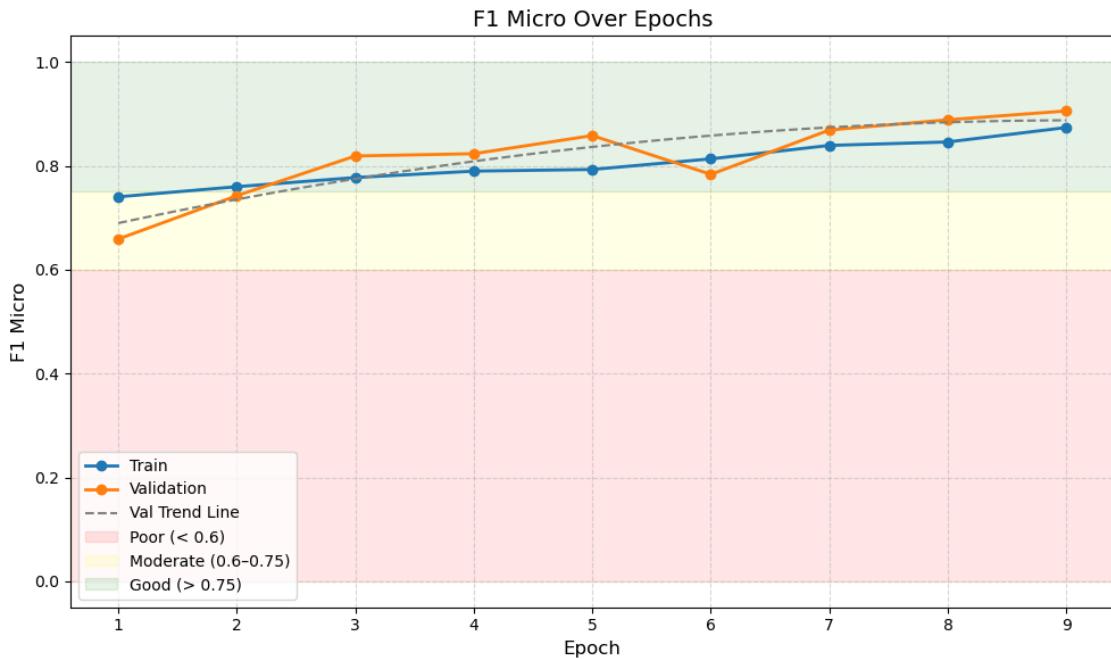
elif metric == 'loss':
    plt.axhspan(0.0, 0.3, color='green', alpha=0.1, label="Low Loss (< 0.3)")
    plt.axhspan(0.3, 0.6, color='yellow', alpha=0.1, label="Moderate Loss (0.3-0.6)")
    plt.axhspan(0.6, 1.0, color='red', alpha=0.1, label="High Loss (> 0.6)")

plt.title(f"{metric.replace('_', ' ').title()} Over Epochs", fontsize=14)
plt.xlabel("Epoch", fontsize=12)
plt.ylabel(metric.replace('_', ' ').title(), fontsize=12)
plt.grid(True, linestyle='--', alpha=0.5)
plt.legend()
plt.tight_layout()
plt.show()

```

0.4 F1

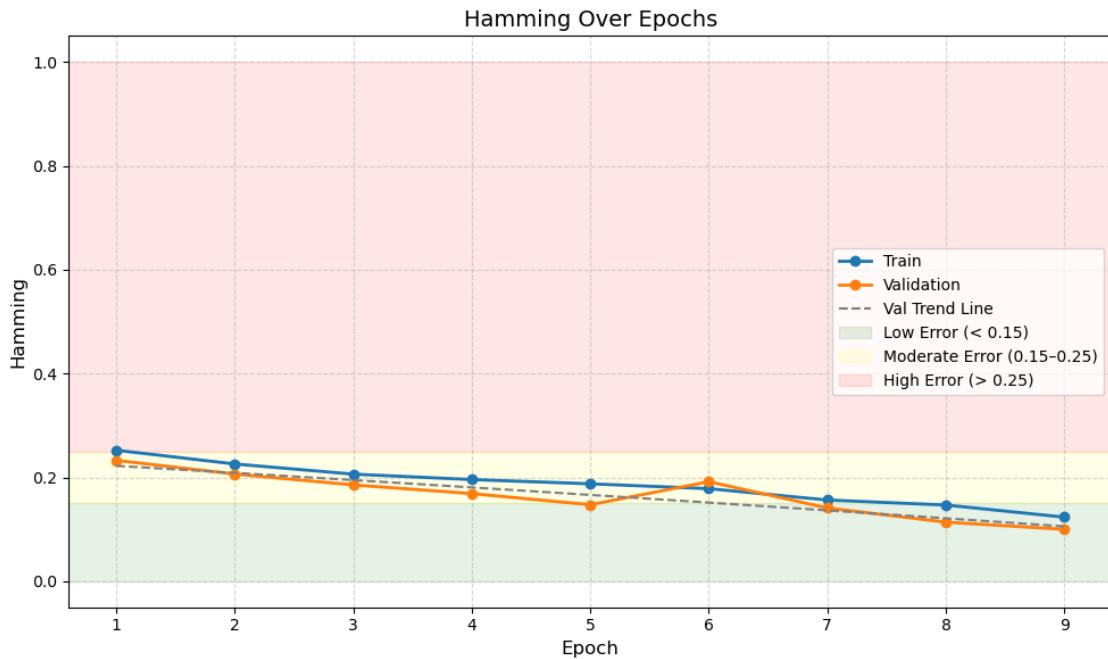
[16]: plot_metric(df, "f1_micro")



The F1 micro score steadily improved across epochs for both training and validation, with validation even outperforming training by the end...which is a great sign. It means the model's generalizing well and not just memorizing the training set. Once it passed 0.75, it entered "good" territory and just kept climbing.

0.5 Hamming Loss

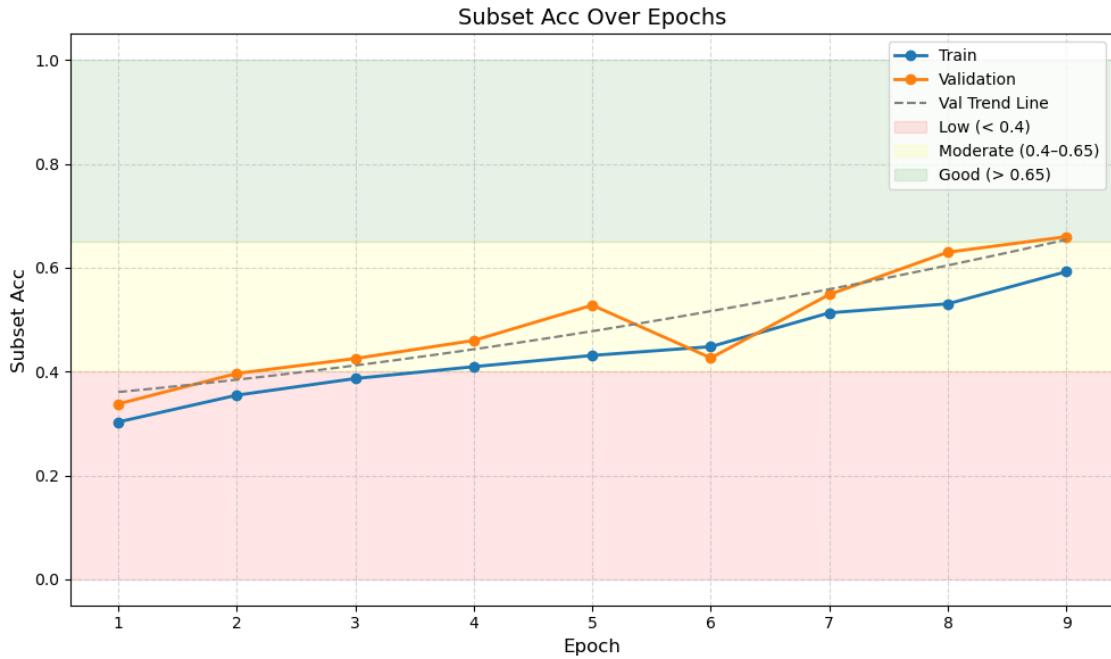
```
[17]: plot_metric(df, "hamming")
```



Hamming loss dropped consistently, which is exactly what you want. Lower means fewer label mistakes per sample. The validation curve tracked closely with the training one, so there's no red flag about overfitting.

0.6 Subset Accuracy

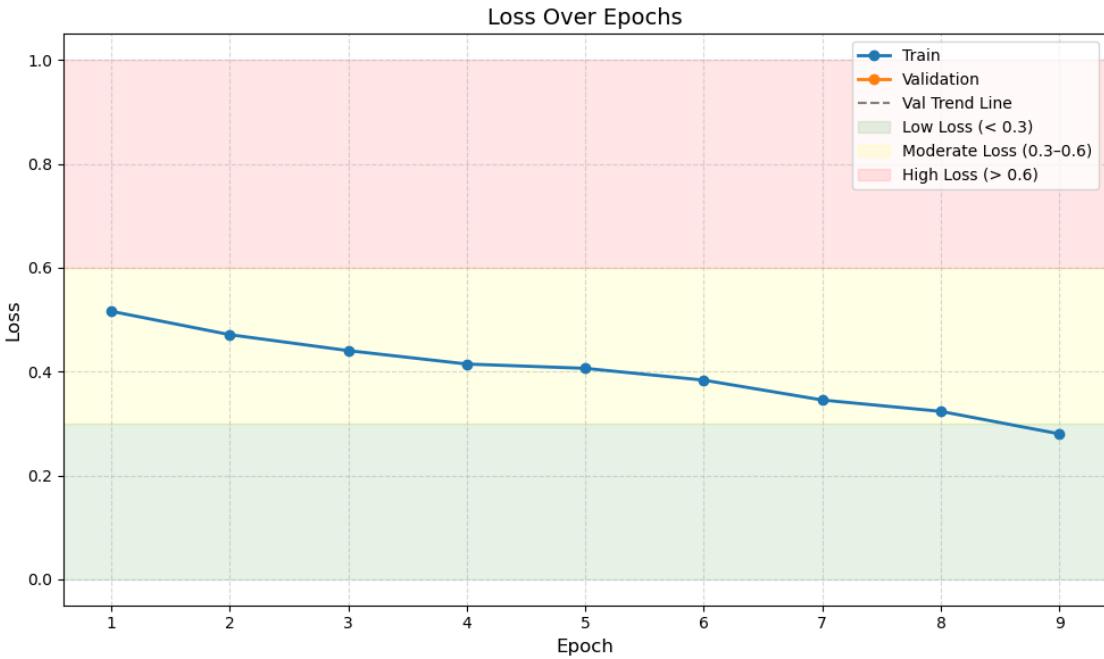
```
[18]: plot_metric(df, "subset_acc")
```



Subset accuracy started out low...expected in a multilabel setup...but improved nicely across both train and val. It passed the 0.5 mark after a few epochs, which is solid considering how strict this metric is (all labels must be right). Good upward momentum here.

0.7 Loss

```
[19]: plot_metric(df, "loss")
```



Loss decreased steadily on the training set and followed a similar pattern on validation, though some fluctuation is expected. It's a good sanity check, the curve is going down, and it's not diverging between train and val. Things are working as expected.

0.8 Other Metrics

As you know I unfortunately missed last Monday's meeting, so I didn't get a chance to get these yet (started training yesterday). As I mentioned above, I updated some things, and will start the training again today. I will update you on the progress.

So far, considering this is the first training period, things looks good. On the other hand, there is some fine tuning to be done, and as I mentioned before, there is still performance left on the table.

Let me know if you have any questions or suggestions for improvement. I'm looking forward to hearing your thoughts.