Methodology_Visuals

[1]:
```python
import os
os.getcwd()
```

[1]: 'C:\\Users\\Chrollo\\Master Block1\\Master Thesis'

1 Methodology Visuals

[ ]:
```python
import os
import json
import joblib
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib
import seaborn as sns
import optuna
import optuna.visualization as vis
from sklearn.metrics import classification_report, confusion_matrix, roc_curve,␣
↪auc
from sklearn.preprocessing import label_binarize
```

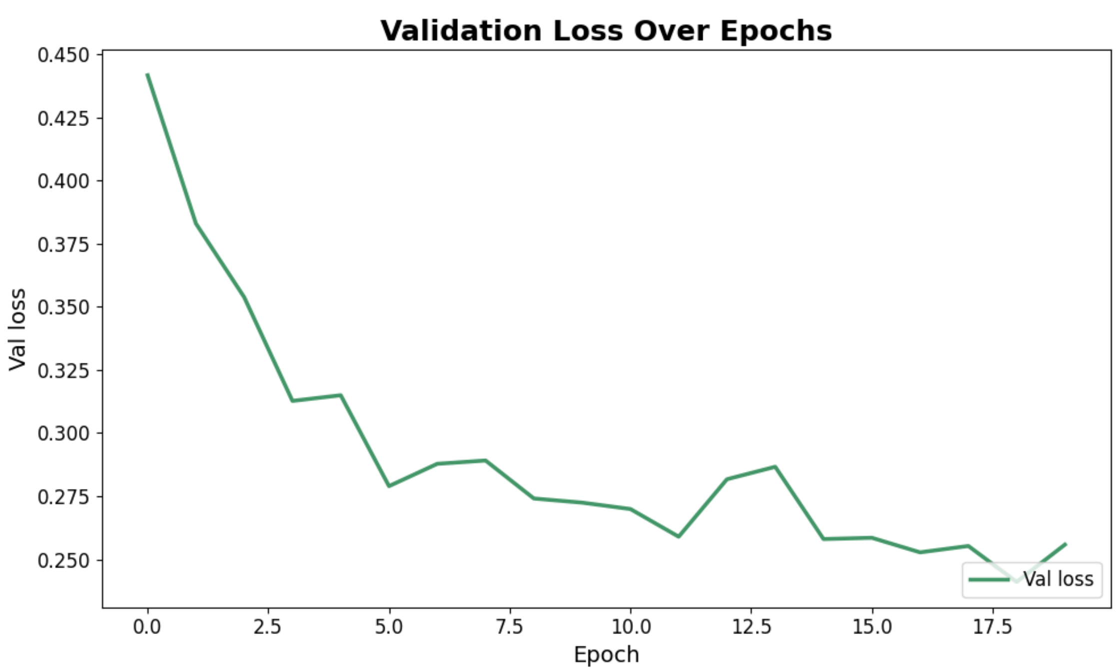
1.0.1 Densenet-121 Hyperparameter Tuning

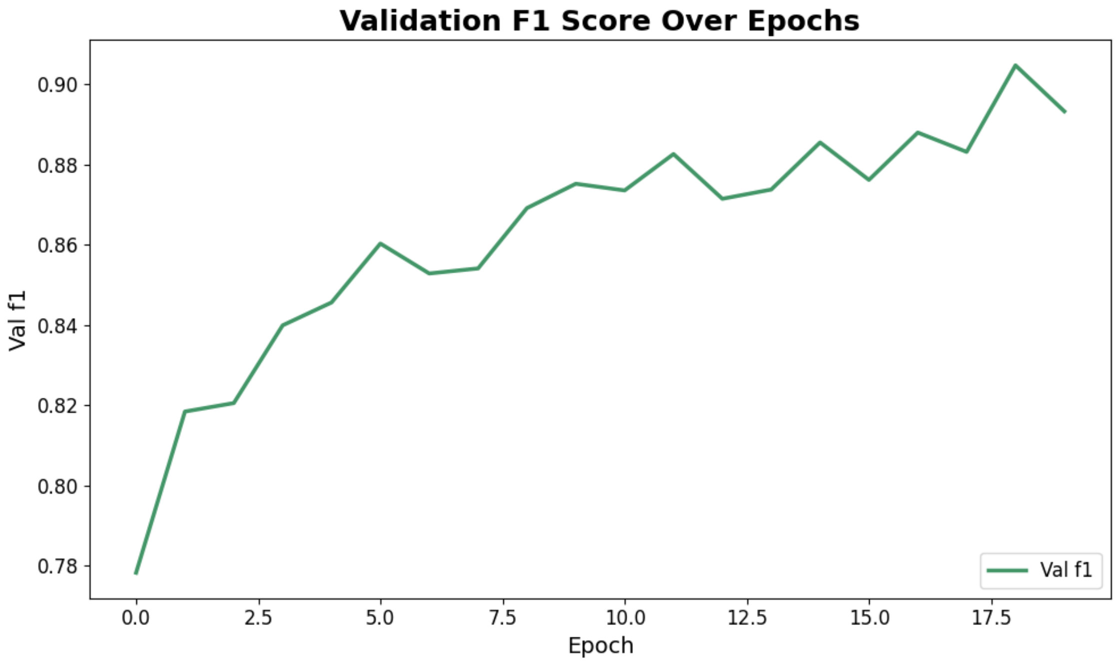
[12]:
```python
# Configuration
output_dir = r"C:/Users/Xuxu/Desktop/Master Thesis/OptunaDensenetFull"
trial_logs_dir = os.path.join(output_dir, "trial_logs")


# Load all metrics.csv logs from Optuna trials
all_trials = []
for trial_name in sorted(os.listdir(trial_logs_dir)):
    trial_path = os.path.join(trial_logs_dir, trial_name, "version_0", "metrics.
↪csv")
    if os.path.exists(trial_path):
        df_trial = pd.read_csv(trial_path)
        trial_id = trial_name.replace("trial_", "")
        df_trial["trial"] = trial_id
```

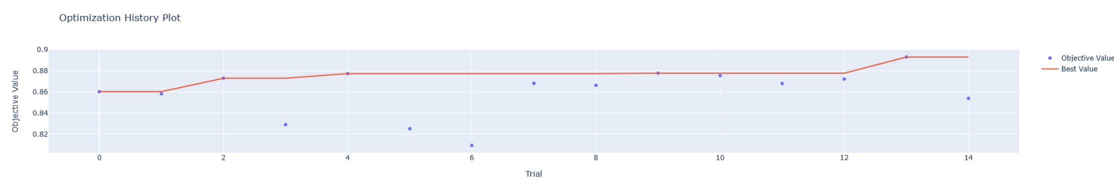
```python
        all_trials.append(df_trial)

if not all_trials:
    print("No metrics.csv files found in:", trial_logs_dir)
    exit()

df_all = pd.concat(all_trials, ignore_index=True)

# Identify the best trial based on final validation F1 score
if "val_f1" not in df_all.columns:
    print("The 'val_f1' column was not found in the metric logs.")
    exit()

last_val_f1_per_trial = (
    df_all.dropna(subset=["val_f1"])
    .sort_values(by=["trial", "epoch"])
    .groupby("trial")
    .tail(1)
)

sorted_trials = last_val_f1_per_trial.sort_values("val_f1", ascending=False).
 ↪reset_index(drop=True)

# Display best trial based on final validation F1 score
best_trial_row = sorted_trials.iloc[0]
best_trial_id = best_trial_row["trial"]
print("\nBest Trial Based on Final Validation F1 Score:")
print(f"Trial ID: {best_trial_id} | Final Val F1: {best_trial_row['val_f1']:.
 ↪4f}\n")

# Display summary of all trials sorted by final F1 score
print("Summary of All Trials (Sorted by Final Val F1):")
for idx, row in sorted_trials.iterrows():
    print(f"Trial {row['trial']} | Val F1: {row['val_f1']:.4f}")


# Define improved plotting function for metrics
def plot_metric(df_trial, metric_col, title):
    if df_trial is None or metric_col not in df_trial.columns:
        print(f"Metric '{metric_col}' not found.")
        return

    df_plot = df_trial.sort_values(by="epoch").dropna(subset=[metric_col])
    if df_plot.empty:
        print(f"No data available to plot for '{metric_col}'.")
        return
```

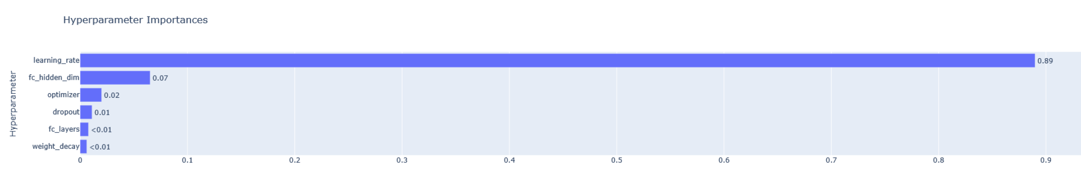
```python
    plt.figure(figsize=(10, 6))
    plt.plot(
        df_plot["epoch"],
        df_plot[metric_col],
        color="#2E8B57",  # SeaGreen for main line
        linestyle="-",
        linewidth=2.5,
        alpha=0.9,
        label=f"{metric_col.replace('_', ' ').capitalize()}"
    )

    plt.title(f"{title}", fontsize=18, fontweight="bold")
    plt.xlabel("Epoch", fontsize=14)
    plt.ylabel(metric_col.replace("_", " ").capitalize(), fontsize=14)
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
    plt.legend(fontsize=12, loc="lower right")
    plt.tight_layout()
    plt.show()

# Plot validation loss and validation F1 score for the best trial
df_best = df_all[df_all["trial"] == best_trial_id]

metrics_to_plot = {
    "val_loss": "Validation Loss Over Epochs",
    "val_f1": "Validation F1 Score Over Epochs"
}

for metric_col, title in metrics_to_plot.items():
    plot_metric(df_best, metric_col, title)

# Load your study
study = joblib.load(r"C:\Users\Xuxu\Desktop\Master␣
↪Thesis\OptunaDensenetFull\new_densenet_study.pkl")

# Plot optimization history
fig1 = vis.plot_optimization_history(study)
fig1.show()

# Plot hyperparameter importance
fig2 = vis.plot_param_importances(study)
fig2.show()

# Plot parallel coordinates
fig3 = vis.plot_parallel_coordinate(study)
fig3.show()
```

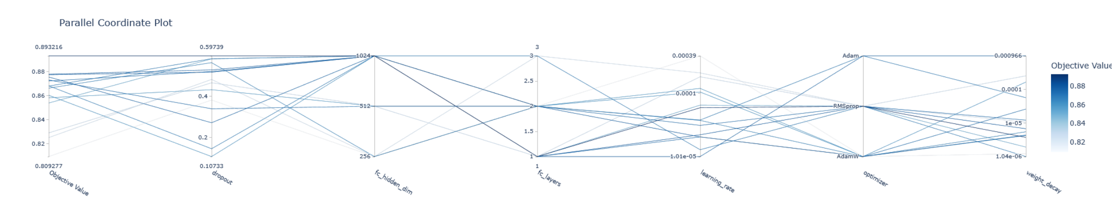
```
Best Trial Based on Final Validation F1 Score:
Trial ID: 13 | Final Val F1: 0.8932

Summary of All Trials (Sorted by Final Val F1):
Trial 13 | Val F1: 0.8932
Trial 9 | Val F1: 0.8779
Trial 4 | Val F1: 0.8775
Trial 10 | Val F1: 0.8756
Trial 2 | Val F1: 0.8731
Trial 12 | Val F1: 0.8723
Trial 7 | Val F1: 0.8683
Trial 11 | Val F1: 0.8680
Trial 8 | Val F1: 0.8663
Trial 0 | Val F1: 0.8604
Trial 1 | Val F1: 0.8583
Trial 14 | Val F1: 0.8539
Trial 3 | Val F1: 0.8290
Trial 5 | Val F1: 0.8251
Trial 6 | Val F1: 0.8093

Best Hyperparameters (from Optuna):
learning_rate: 5.94360847333523e-05
fc_layers: 1
dropout: 0.5973891128630341
weight_decay: 3.816225528011092e-06
fc_hidden_dim: 1024
optimizer: RMSprop
```




Validation F1 Score Over Epochs
0.90
0.88
0.86
0.84
0.82
0.80
0.78
Val f1
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch
Val f1


Optimization History Plot
Objective Value
0.9
0.88
0.86
0.84
0.82
0
2
4
6
8
10
12
14
Trial
Objective Value
Best Value


Hyperparameter Importances
Hyperparameter
learning_rate
fc_hidden_dim
optimizer
dropout
fc_layers
weight_decay
0.89
0.07
0.02
0.01
<0.01
<0.01
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Hyperparameter Importance


Parallel Coordinate Plot
Objective Value
dropout
fc_hidden_dim
fc_layers
learning_rate
optimizer
weight_decay
Adam
RMSprop
AdamW
Objective Value

[41]:
```
#get DataFrame
trials_df = study.trials_dataframe()
trials_df
```

[41]:
```
    number     value             datetime_start          datetime_complete  \
0        0  0.860382 2025-04-15 21:35:05.035791 2025-04-15 21:43:01.992542
1        1  0.858326 2025-04-15 21:43:01.994099 2025-04-15 21:51:39.359135
2        2  0.873119 2025-04-15 21:51:39.359135 2025-04-15 22:03:23.395702
3        3  0.829012 2025-04-15 22:03:23.395702 2025-04-15 22:09:45.924728
4        4  0.877488 2025-04-15 22:09:45.924728 2025-04-15 22:24:50.769611
5        5  0.825104 2025-04-15 22:24:50.769611 2025-04-15 22:34:49.092575
6        6  0.809277 2025-04-15 22:34:49.092575 2025-04-15 22:46:26.467869
7        7  0.868266 2025-04-15 22:46:26.467869 2025-04-15 22:56:47.318181
8        8  0.866303 2025-04-15 22:56:47.318181 2025-04-15 23:10:42.992636
9        9  0.877906 2025-04-15 23:10:42.992636 2025-04-15 23:26:06.342946
10      10  0.875609 2025-04-15 23:26:06.342946 2025-04-15 23:41:47.663797
11      11  0.868041 2025-04-15 23:41:47.663797 2025-04-15 23:57:26.864957
12      12  0.872270 2025-04-15 23:57:26.864957 2025-04-16 00:09:46.893251
13      13  0.893216 2025-04-16 00:09:46.893251 2025-04-16 00:25:18.705512
14      14  0.853925 2025-04-16 00:25:18.707514 2025-04-16 00:33:20.565847

                 duration  params_dropout  params_fc_hidden_dim  \
0  0 days 00:07:56.956751        0.107335                  1024
1  0 days 00:08:37.365036        0.432731                   512
2  0 days 00:11:44.036567        0.340012                   512
3  0 days 00:06:22.529026        0.466376                   512
4  0 days 00:15:04.844883        0.517950                  1024
5  0 days 00:09:58.322964        0.482748                   256
6  0 days 00:11:37.375294        0.382394                   256
7  0 days 00:10:20.850312        0.145488                  1024
8  0 days 00:13:55.674455        0.564494                   256
9  0 days 00:15:23.350310        0.529065                  1024
10 0 days 00:15:41.320851        0.271944                  1024
11 0 days 00:15:39.201160        0.583499                  1024
12 0 days 00:12:20.028294        0.521056                  1024
13 0 days 00:15:31.812261        0.597389                  1024
14 0 days 00:08:01.858333        0.582743                  1024

    params_fc_layers  params_learning_rate params_optimizer  \
0                  2              0.000105            AdamW
1                  2              0.000120            AdamW
2                  2              0.000031          RMSprop
3                  1              0.000183          RMSprop
4                  2              0.000021            AdamW
5                  3              0.000210          RMSprop
```

```
6                    2              0.000390            AdamW
7                    2              0.000038             Adam
8                    2              0.000038            AdamW
9                    1              0.000022          RMSprop
10                   1              0.000010             Adam
11                   3              0.000013          RMSprop
12                   1              0.000021            AdamW
13                   1              0.000059          RMSprop
14                   1              0.000065          RMSprop

    params_weight_decay     state
0              0.000163  COMPLETE
1              0.000004  COMPLETE
2              0.000012  COMPLETE
3              0.000249  COMPLETE
4              0.000004  COMPLETE
5              0.000011  COMPLETE
6              0.000001  COMPLETE
7              0.000056  COMPLETE
8              0.000006  COMPLETE
9              0.000007  COMPLETE
10             0.000966  COMPLETE
11             0.000001  COMPLETE
12             0.000026  COMPLETE
13             0.000004  COMPLETE
14             0.000002  COMPLETE
```

1.0.2 ConvNext-Tiny Hyperparameter Tuning

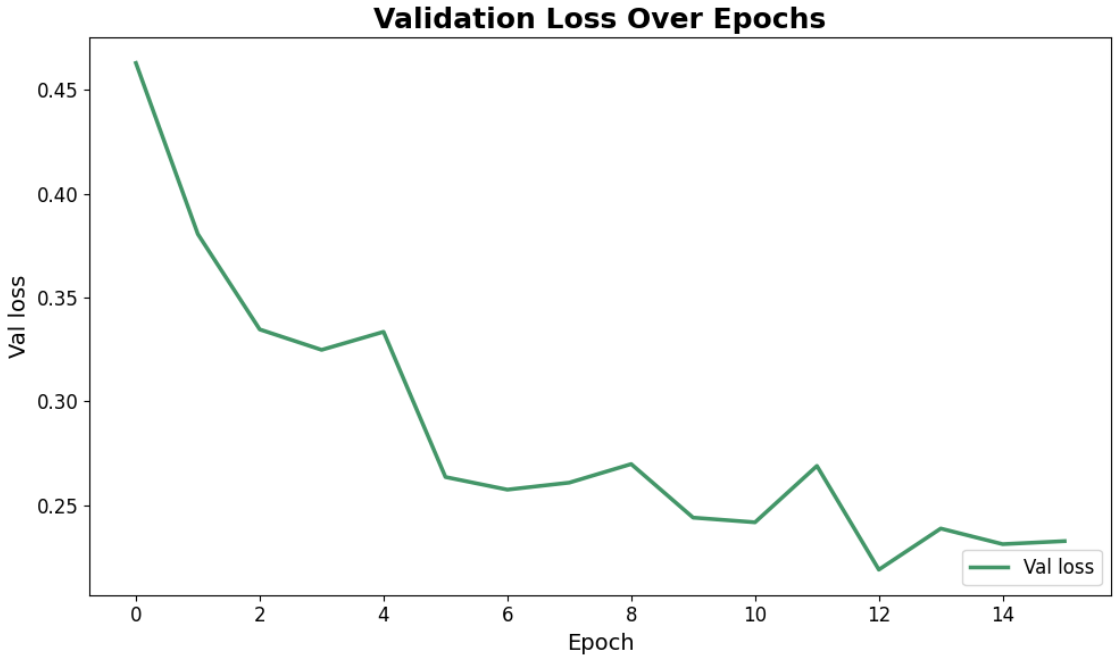
[11]:
```python
# configuration
output_dir = r"C:/Users/Xuxu/Desktop/Master Thesis/OptunaConvNeXtFull"
trial_logs_dir = os.path.join(output_dir, "trial_logs")

# load all metrics.csv logs from optuna trials
all_trials = []
for trial_name in sorted(os.listdir(trial_logs_dir)):
    trial_path = os.path.join(trial_logs_dir, trial_name, "version_0", "metrics.
↪csv")
    if os.path.exists(trial_path):
        df_trial = pd.read_csv(trial_path)
        trial_id = trial_name.replace("trial_", "")
        df_trial["trial"] = trial_id
        all_trials.append(df_trial)

if not all_trials:
    print("no metrics.csv files found in:", trial_logs_dir)
    exit()
```

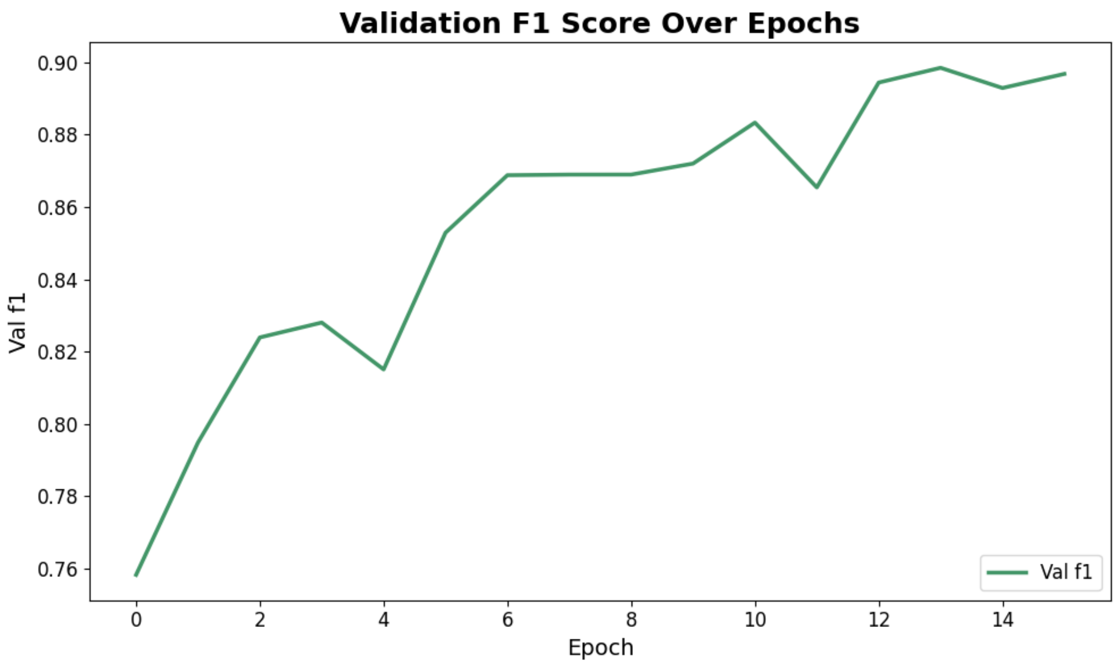
```
df_all = pd.concat(all_trials, ignore_index=True)

# identify the best trial based on final validation f1 score
if "val_f1" not in df_all.columns:
    print("the 'val_f1' column was not found in the metric logs.")
    exit()

last_val_f1_per_trial = (
    df_all.dropna(subset=["val_f1"])
    .sort_values(by=["trial", "epoch"])
    .groupby("trial")
    .tail(1)
)

sorted_trials = last_val_f1_per_trial.sort_values("val_f1", ascending=False).
 ↪reset_index(drop=True)

# display best trial based on final validation f1 score
best_trial_row = sorted_trials.iloc[0]
best_trial_id = best_trial_row["trial"]
print("\nbest trial based on final validation f1 score:")
print(f"trial id: {best_trial_id} | final val f1: {best_trial_row['val_f1']:.
 ↪4f}\n")

# display summary of all trials sorted by final f1 score
print("summary of all trials (sorted by final val f1):")
for idx, row in sorted_trials.iterrows():
    print(f"trial {row['trial']} | val f1: {row['val_f1']:.4f}")

# load optuna study and best hyperparameters
pkl_path = r"C:/Users/Xuxu/Desktop/Master Thesis/OptunaConvNeXtFull/
 ↪new_convnext_study.pkl"
study = joblib.load(pkl_path)

print("\nbest hyperparameters (from optuna):")
for key, value in study.best_trial.params.items():
    print(f"{key}: {value}")

# define improved plotting function for metrics
def plot_metric(df_trial, metric_col, title):
    if df_trial is None or metric_col not in df_trial.columns:
        print(f"metric '{metric_col}' not found.")
        return

    df_plot = df_trial.sort_values(by="epoch").dropna(subset=[metric_col])
    if df_plot.empty:
```

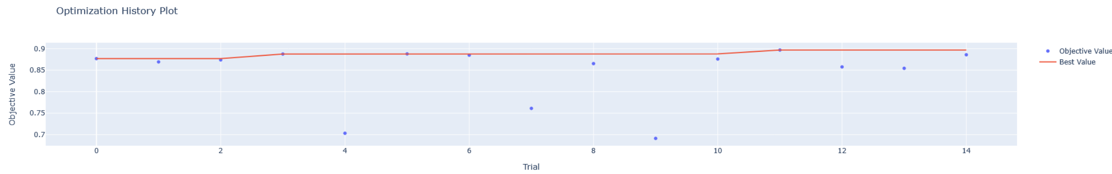
```
        print(f"no data available to plot for '{metric_col}'.")
        return

    plt.figure(figsize=(10, 6))
    plt.plot(
        df_plot["epoch"],
        df_plot[metric_col],
        color="#2E8B57",  # seagreen for main line
        linestyle="-",
        linewidth=2.5,
        alpha=0.9,
        label=f"{metric_col.replace('_', ' ').capitalize()}"
    )

    plt.title(f"{title}", fontsize=18, fontweight="bold")
    plt.xlabel("epoch", fontsize=14)
    plt.ylabel(metric_col.replace("_", " ").capitalize(), fontsize=14)
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
    plt.legend(fontsize=12, loc="lower right")
    plt.tight_layout()
    plt.show()

# plot validation loss and validation f1 score for the best trial
df_best = df_all[df_all["trial"] == best_trial_id]

metrics_to_plot = {
    "val_loss": "validation loss over epochs",
    "val_f1": "validation f1 score over epochs"
}

for metric_col, title in metrics_to_plot.items():
    plot_metric(df_best, metric_col, title)

# load your study
study = joblib.load(r"C:\Users\Xuxu\Desktop\Master␣
↪Thesis\OptunaConvNeXtFull\new_convnext_study.pkl")

# plot optimization history
fig1 = vis.plot_optimization_history(study)
fig1.show()

# plot hyperparameter importance
fig2 = vis.plot_param_importances(study)
fig2.show()

# plot parallel coordinates
```

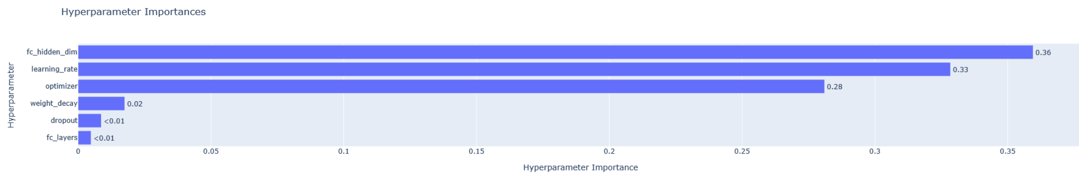
```
fig3 = vis.plot_parallel_coordinate(study)
fig3.show()
```

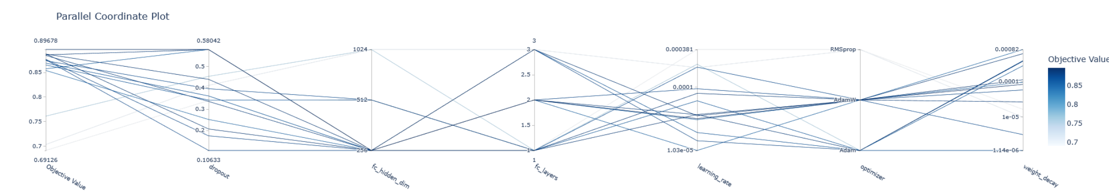
```
Best Trial Based on Final Validation F1 Score:
Trial ID: 11 | Final Val F1: 0.8968

Summary of All Trials (Sorted by Final Val F1):
Trial 11 | Val F1: 0.8968
Trial 5 | Val F1: 0.8878
Trial 3 | Val F1: 0.8875
Trial 14 | Val F1: 0.8859
Trial 6 | Val F1: 0.8847
Trial 0 | Val F1: 0.8768
Trial 10 | Val F1: 0.8759
Trial 2 | Val F1: 0.8739
Trial 1 | Val F1: 0.8692
Trial 8 | Val F1: 0.8653
Trial 12 | Val F1: 0.8575
Trial 13 | Val F1: 0.8542
Trial 7 | Val F1: 0.7611
Trial 4 | Val F1: 0.7033
Trial 9 | Val F1: 0.6913

Best Hyperparameters (from Optuna):
learning_rate: 3.2244442224520205e-05
fc_layers: 2
dropout: 0.5804239833977702
weight_decay: 8.458353625551204e-05
fc_hidden_dim: 256
optimizer: AdamW
```


Validation Loss Over Epochs
0.45
0.40
0.35
0.30
0.25
Val loss
0
2
4
6
8
10
12
14
Epoch
Val loss


Validation F1 Score Over Epochs
0.90
0.88
0.86
0.84
0.82
0.80
0.78
0.76
Val f1
0
2
4
6
8
10
12
14
Epoch
Val f1







[38]:

```python
# Get DataFrame
trials_df = study.trials_dataframe()
trials_df
```

[38]:

	number	value	datetime_start	datetime_complete	\
0	0	0.876827	2025-04-16 03:20:20.492022	2025-04-16 03:33:01.544538	
1	1	0.869207	2025-04-16 03:33:01.544538	2025-04-16 03:41:58.009991	
2	2	0.873937	2025-04-16 03:41:58.025623	2025-04-16 03:49:34.758889	
3	3	0.887534	2025-04-16 03:49:34.758889	2025-04-16 04:02:03.073885	
4	4	0.703310	2025-04-16 04:02:03.073885	2025-04-16 04:05:25.722288	
5	5	0.887820	2025-04-16 04:05:25.723295	2025-04-16 04:13:38.988839	
6	6	0.884657	2025-04-16 04:13:38.988839	2025-04-16 04:21:20.104420	
7	7	0.761095	2025-04-16 04:21:20.104420	2025-04-16 04:25:21.287224	
8	8	0.865312	2025-04-16 04:25:21.287224	2025-04-16 04:33:39.901703	
9	9	0.691264	2025-04-16 04:33:39.902649	2025-04-16 04:43:48.351275	
10	10	0.875940	2025-04-16 04:43:48.351275	2025-04-16 04:52:06.200594	
11	11	0.896784	2025-04-16 04:52:06.200594	2025-04-16 05:02:14.715406	
12	12	0.857467	2025-04-16 05:02:14.715406	2025-04-16 05:11:48.747401	
13	13	0.854163	2025-04-16 05:11:48.747401	2025-04-16 05:17:04.646691	
14	14	0.885886	2025-04-16 05:17:04.646691	2025-04-16 05:25:23.562713	

	duration	params_dropout	params_fc_hidden_dim
0	0 days 00:12:41.052516	0.174981	256
1	0 days 00:08:56.465453	0.362265	256
2	0 days 00:07:36.733266	0.396210	512
3	0 days 00:12:28.314996	0.440397	256
4	0 days 00:03:22.648403	0.412031	1024
5	0 days 00:08:13.265544	0.207101	256
6	0 days 00:07:41.115581	0.336519	256
7	0 days 00:04:01.182804	0.454363	1024
8	0 days 00:08:18.614479	0.342891	512
9	0 days 00:10:08.448626	0.340564	512
10	0 days 00:08:17.849319	0.106332	256
11	0 days 00:10:08.514812	0.580424	256
12	0 days 00:09:34.031995	0.580390	256
13	0 days 00:05:15.899290	0.251827	256
14	0 days 00:08:18.916022	0.580085	256

	params_fc_layers	params_learning_rate	params_optimizer
0	3	0.000015	Adam
1	3	0.000020	Adam
2	1	0.000203	AdamW
3	3	0.000036	AdamW
4	3	0.000199	RMSprop
5	1	0.000037	AdamW
6	2	0.000037	Adam
7	1	0.000225	Adam
8	1	0.000061	Adam
9	1	0.000381	RMSprop
10	2	0.000094	AdamW
11	2	0.000032	AdamW
12	2	0.000010	AdamW
13	2	0.000031	AdamW
14	1	0.000080	AdamW

	params_weight_decay	state
0	0.000394	COMPLETE
1	0.000001	COMPLETE
2	0.000003	COMPLETE
3	0.000114	COMPLETE
4	0.000010	COMPLETE
5	0.000624	COMPLETE
6	0.000381	COMPLETE
7	0.000388	COMPLETE
8	0.000284	COMPLETE
9	0.000017	COMPLETE
10	0.000059	COMPLETE
11	0.000085	COMPLETE

```
12            0.000097   COMPLETE
13            0.000820   COMPLETE
14            0.000027   COMPLETE
```

1.0.3 EfficientNet-B0 Hyperparameter Tuning

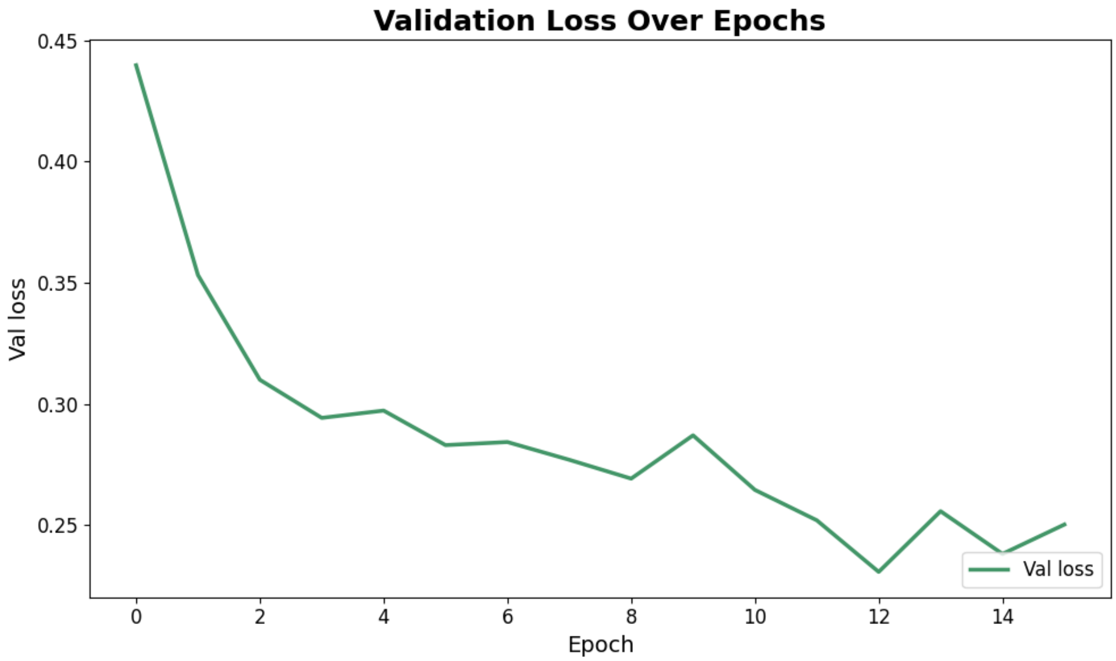
[10]:
```python
# configuration
output_dir = r"C:/Users/Xuxu/Desktop/Master Thesis/OptunaEfficientNetB0Full"
trial_logs_dir = os.path.join(output_dir, "trial_logs")

# load all metrics.csv logs from optuna trials
all_trials = []
for trial_name in sorted(os.listdir(trial_logs_dir)):
    trial_path = os.path.join(trial_logs_dir, trial_name, "version_0", "metrics.
↪csv")
    if os.path.exists(trial_path):
        df_trial = pd.read_csv(trial_path)
        trial_id = trial_name.replace("trial_", "")
        df_trial["trial"] = trial_id
        all_trials.append(df_trial)

if not all_trials:
    print("no metrics.csv files found in:", trial_logs_dir)
    exit()

df_all = pd.concat(all_trials, ignore_index=True)

# identify the best trial based on final validation f1 score
if "val_f1" not in df_all.columns:
    print("the 'val_f1' column was not found in the metric logs.")
    exit()

last_val_f1_per_trial = (
    df_all.dropna(subset=["val_f1"])
    .sort_values(by=["trial", "epoch"])
    .groupby("trial")
    .tail(1)
)

sorted_trials = last_val_f1_per_trial.sort_values("val_f1", ascending=False).
↪reset_index(drop=True)

# display best trial based on final validation f1 score
best_trial_row = sorted_trials.iloc[0]
best_trial_id = best_trial_row["trial"]
print("\nbest trial based on final validation f1 score:")
```

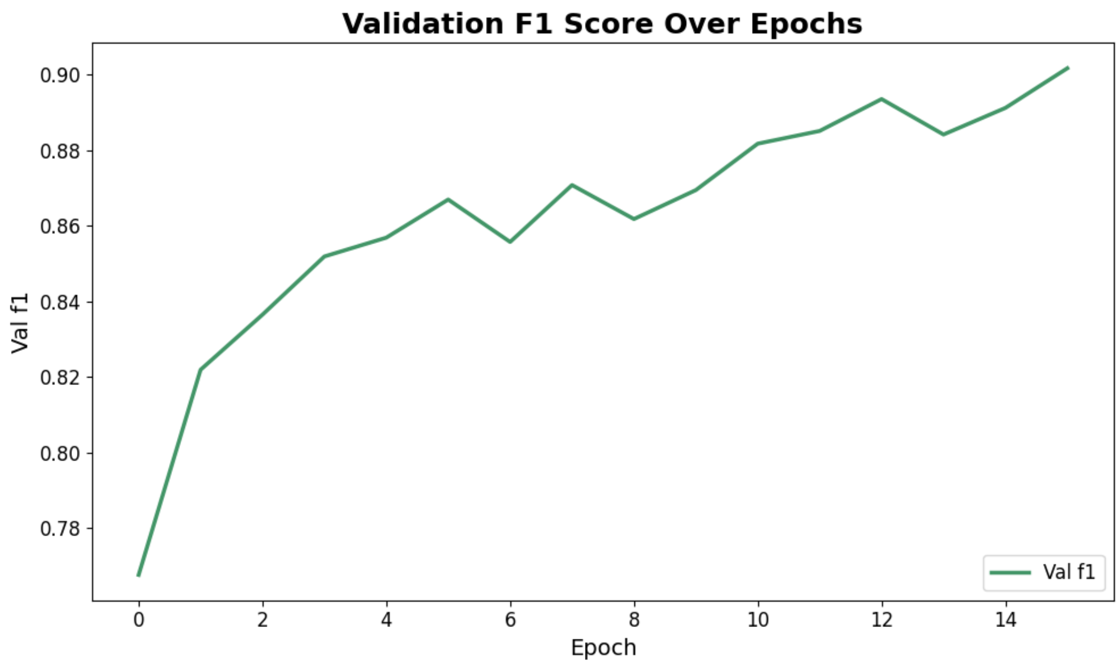
```
print(f"trial id: {best_trial_id} | final val f1: {best_trial_row['val_f1']:.
 ↪4f}\n")

# display summary of all trials sorted by final f1 score
print("summary of all trials (sorted by final val f1):")
for idx, row in sorted_trials.iterrows():
    print(f"trial {row['trial']} | val f1: {row['val_f1']:.4f}")

# load optuna study and best hyperparameters
pkl_path = r"C:/Users/Xuxu/Desktop/Master Thesis/OptunaEfficientNetB0Full/
 ↪new_efficientnet_study.pkl"
study = joblib.load(pkl_path)

print("\nbest hyperparameters (from optuna):")
for key, value in study.best_trial.params.items():
    print(f"{key}: {value}")

# define improved plotting function for metrics
def plot_metric(df_trial, metric_col, title):
    if df_trial is None or metric_col not in df_trial.columns:
        print(f"metric '{metric_col}' not found.")
        return

    df_plot = df_trial.sort_values(by="epoch").dropna(subset=[metric_col])
    if df_plot.empty:
        print(f"no data available to plot for '{metric_col}'.")
        return

    plt.figure(figsize=(10, 6))
    plt.plot(
        df_plot["epoch"],
        df_plot[metric_col],
        color="#2E8B57",  # seagreen for main line
        linestyle="-",
        linewidth=2.5,
        alpha=0.9,
        label=f"{metric_col.replace('_', ' ').capitalize()}"
    )

    plt.title(f"{title}", fontsize=18, fontweight="bold")
    plt.xlabel("epoch", fontsize=14)
    plt.ylabel(metric_col.replace("_", " ").capitalize(), fontsize=14)
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
    plt.legend(fontsize=12, loc="lower right")
    plt.tight_layout()
    plt.show()
```

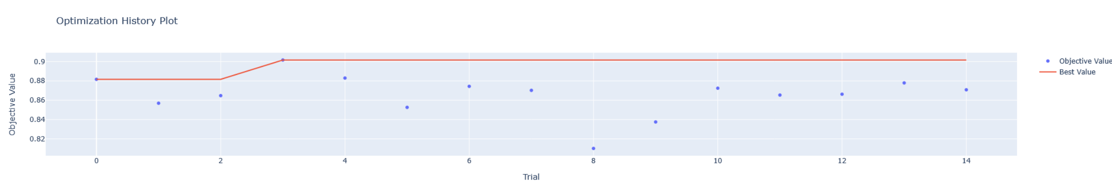
```python
# plot validation loss and validation f1 score for the best trial
df_best = df_all[df_all["trial"] == best_trial_id]

metrics_to_plot = {
    "val_loss": "validation loss over epochs",
    "val_f1": "validation f1 score over epochs"
}

for metric_col, title in metrics_to_plot.items():
    plot_metric(df_best, metric_col, title)

# load your study
study = joblib.load(r"C:\Users\Xuxu\Desktop\Master␣
 ↪Thesis\OptunaEfficientNetB0Full\new_efficientnet_study.pkl")

# plot optimization history
fig1 = vis.plot_optimization_history(study)
fig1.show()

# plot hyperparameter importance
fig2 = vis.plot_param_importances(study)
fig2.show()

# plot parallel coordinates
fig3 = vis.plot_parallel_coordinate(study)
fig3.show()
```

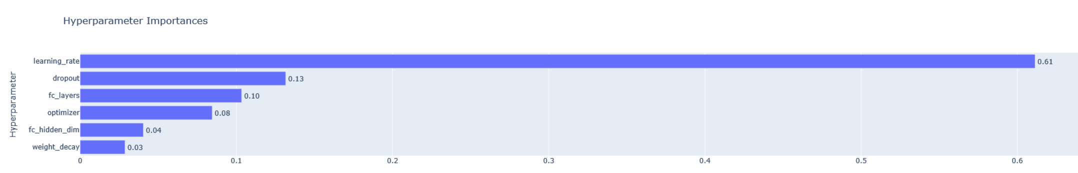
```
Best Trial Based on Final Validation F1 Score:
Trial ID: 3 | Final Val F1: 0.9016

Summary of All Trials (Sorted by Final Val F1):
Trial 3 | Val F1: 0.9016
Trial 4 | Val F1: 0.8830
Trial 0 | Val F1: 0.8817
Trial 13 | Val F1: 0.8780
Trial 6 | Val F1: 0.8744
Trial 10 | Val F1: 0.8725
Trial 14 | Val F1: 0.8709
Trial 7 | Val F1: 0.8703
Trial 12 | Val F1: 0.8663
Trial 11 | Val F1: 0.8654
Trial 2 | Val F1: 0.8647
Trial 1 | Val F1: 0.8569
Trial 5 | Val F1: 0.8527
Trial 9 | Val F1: 0.8375
Trial 8 | Val F1: 0.8101
```

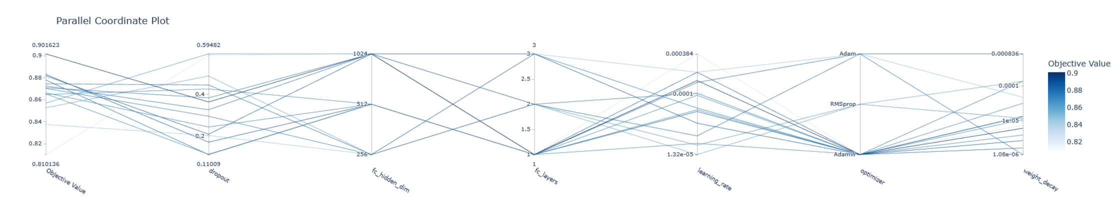
```
Best Hyperparameters (from Optuna):
learning_rate: 0.00015711821204231546
fc_layers: 1
dropout: 0.36330619574900347
weight_decay: 6.1282266824256e-06
fc_hidden_dim: 1024
optimizer: AdamW
```




Validation F1 Score Over Epochs
Val f1
0.90
0.88
0.86
0.84
0.82
0.80
0.78
0 2 4 6 8 10 12 14
Epoch
Val f1


Optimization History Plot
Objective Value
0.9
0.88
0.86
0.84
0.82
0 2 4 6 8 10 12 14
Trial
Objective Value
Best Value


Hyperparameter Importances
Hyperparameter
learning_rate 0.61
dropout 0.13
fc_layers 0.10
optimizer 0.08
fc_hidden_dim 0.04
weight_decay 0.03
0 0.1 0.2 0.3 0.4 0.5 0.6
Hyperparameter Importance


Parallel Coordinate Plot
0.901623
0.9
0.88
0.86
0.84
0.82
0.810136
Objective Value
0.59482
0.4
0.2
0.11009
dropout
1024
512
256
fc_hidden_dim
3
2.5
2
1.5
1
fc_layers
0.000384
0.0001
1.32e-05
learning_rate
Adam
RMSprop
AdamW
optimizer
0.000836
0.0001
1e-05
1.08e-06
weight_decay
Objective Value
0.9
0.88
0.86
0.84
0.82

[41]:
```python
# Get DataFrame
trials_df = study.trials_dataframe()
trials_df
```

[41]:
```
    number     value             datetime_start          datetime_complete  \
0        0  0.881693 2025-04-16 05:25:34.129115 2025-04-16 05:42:49.294764
1        1  0.856862 2025-04-16 05:42:49.311074 2025-04-16 06:00:05.093702
2        2  0.864706 2025-04-16 06:00:05.093702 2025-04-16 06:17:02.659615
3        3  0.901623 2025-04-16 06:17:02.659615 2025-04-16 06:30:38.225577
4        4  0.883041 2025-04-16 06:30:38.225577 2025-04-16 06:45:03.940303
5        5  0.852651 2025-04-16 06:45:03.940303 2025-04-16 07:02:04.766856
6        6  0.874399 2025-04-16 07:02:04.766856 2025-04-16 07:19:12.054285
7        7  0.870292 2025-04-16 07:19:12.054285 2025-04-16 07:33:37.552547
8        8  0.810136 2025-04-16 07:33:37.552547 2025-04-16 07:39:39.963143
9        9  0.837455 2025-04-16 07:39:39.963143 2025-04-16 07:45:45.873727
10      10  0.872457 2025-04-16 07:45:45.873727 2025-04-16 07:57:48.398021
11      11  0.865388 2025-04-16 07:57:48.398021 2025-04-16 08:10:34.306670
12      12  0.866279 2025-04-16 08:10:34.306670 2025-04-16 08:23:19.933418
13      13  0.877960 2025-04-16 08:23:19.933418 2025-04-16 08:36:56.794675
14      14  0.870880 2025-04-16 08:36:56.794675 2025-04-16 08:48:53.440917

                 duration  params_dropout  params_fc_hidden_dim  \
0  0 days 00:17:15.165649        0.207850                  1024
1  0 days 00:17:15.782628        0.594822                  1024
2  0 days 00:16:57.565913        0.425911                   512
3  0 days 00:13:35.565962        0.363306                  1024
4  0 days 00:14:25.714726        0.169852                   512
5  0 days 00:17:00.826553        0.488134                   256
6  0 days 00:17:07.287429        0.445353                   256
7  0 days 00:14:25.498262        0.294429                   256
8  0 days 00:06:02.410596        0.586961                  1024
9  0 days 00:06:05.910584        0.201819                   256
10 0 days 00:12:02.524294        0.326032                  1024
11 0 days 00:12:45.908649        0.111321                   512
12 0 days 00:12:45.626748        0.242842                   512
13 0 days 00:13:36.861257        0.110090                   512
14 0 days 00:11:56.646242        0.382918                  1024

    params_fc_layers  params_learning_rate params_optimizer  \
0                  3              0.000038            AdamW
1                  2              0.000018          RMSprop
2                  1              0.000019            AdamW
3                  1              0.000157            AdamW
4                  1              0.000056            AdamW
5                  2              0.000013          RMSprop
6                  2              0.000025             Adam
7                  3              0.000063            AdamW
```

```
8                   1             0.000384            AdamW
9                   3             0.000208             Adam
10                  1             0.000148             Adam
11                  1             0.000096            AdamW
12                  1             0.000059            AdamW
13                  1             0.000208            AdamW
14                  2             0.000103            AdamW

    params_weight_decay     state
0              0.000003  COMPLETE
1              0.000012  COMPLETE
2              0.000011  COMPLETE
3              0.000006  COMPLETE
4              0.000013  COMPLETE
5              0.000137  COMPLETE
6              0.000836  COMPLETE
7              0.000138  COMPLETE
8              0.000007  COMPLETE
9              0.000048  COMPLETE
10             0.000001  COMPLETE
11             0.000032  COMPLETE
12             0.000004  COMPLETE
13             0.000002  COMPLETE
14             0.000014  COMPLETE
```

[ ]:

1.0.4 Densenet-121 + Best Hyperparameter

[7]:

```python
# configuration
save_dir = r"C:/Users/Xuxu/Desktop/Master Thesis/BestHyperDensenet121Full"
class_idx_path = r"C:/Users/Xuxu/Desktop/Master Thesis/OptunaDensenetFull/
  ↪class_to_idx.json"
num_folds = 5

# load class mapping
with open(class_idx_path, "r") as f:
    class_to_idx = json.load(f)
idx_to_class = {v: k for k, v in class_to_idx.items()}
class_names = [idx_to_class[i] for i in range(len(idx_to_class))]

# load predictions across folds
all_preds_all_folds = []
all_targets_all_folds = []

for fold in range(num_folds):
    fold_dir = os.path.join(save_dir, f"fold_{fold}", "version_0")
```

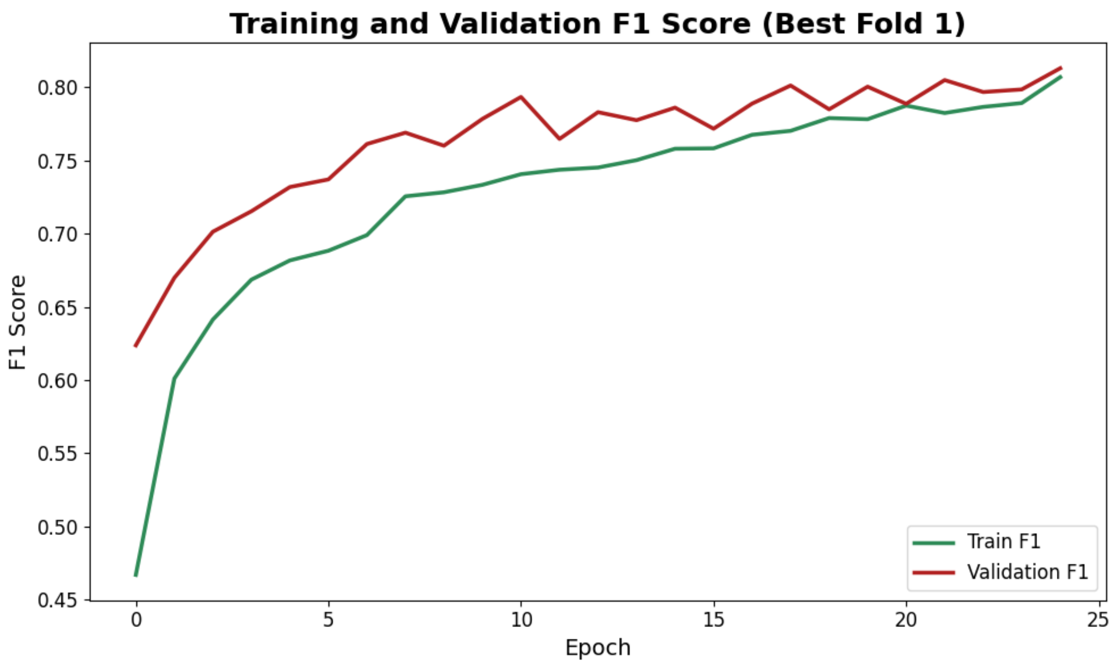
```python
    preds_path = os.path.join(fold_dir, "all_preds.npy")
    targets_path = os.path.join(fold_dir, "all_targets.npy")

    if os.path.exists(preds_path) and os.path.exists(targets_path):
        all_preds_all_folds.extend(np.load(preds_path))
        all_targets_all_folds.extend(np.load(targets_path))

# convert predictions and targets to numpy arrays
all_preds_all_folds = np.array(all_preds_all_folds)
all_targets_all_folds = np.array(all_targets_all_folds)

# fold-wise final metrics summary
metrics_cols = [
    "train_f1", "train_precision", "train_recall", "train_loss",
    "val_f1", "val_precision", "val_recall", "val_loss"
]
metrics_summary = []

for fold in range(num_folds):
    metrics_file = os.path.join(save_dir, f"fold_{fold}", "version_0", "metrics.
↪csv")
    if os.path.exists(metrics_file):
        df = pd.read_csv(metrics_file)

        df_val = df.dropna(subset=["val_f1"])
        df_train = df.dropna(subset=["train_f1"])

        last_val = df_val.sort_values("epoch").groupby("epoch").tail(1).iloc[-1]
        last_train = df_train.sort_values("epoch").groupby("epoch").tail(1).
↪iloc[-1]

        row = [
            last_train.get("train_f1", np.nan),
            last_train.get("train_precision", np.nan),
            last_train.get("train_recall", np.nan),
            last_train.get("train_loss", np.nan),
            last_val.get("val_f1", np.nan),
            last_val.get("val_precision", np.nan),
            last_val.get("val_recall", np.nan),
            last_val.get("val_loss", np.nan),
        ]
        metrics_summary.append(row)

# convert to dataframe
df_summary = pd.DataFrame(metrics_summary, columns=metrics_cols)
df_summary.index = [f"fold {i}" for i in range(num_folds)]
```

```python
# display fold-wise metrics summary
print("\n=== fold-wise metrics summary ===")
for idx, row in df_summary.iterrows():
    print(f"{idx}: "
          f"train f1: {row['train_f1']:.4f}, "
          f"val f1: {row['val_f1']:.4f}, "
          f"val precision: {row['val_precision']:.4f}, "
          f"val recall: {row['val_recall']:.4f}, "
          f"val loss: {row['val_loss']:.4f}")

# display mean and standard deviation across folds
print("\n=== mean and standard deviation across folds ===")
mean_std = df_summary.agg(["mean", "std"]).round(4)

print(f"mean train f1: {mean_std.loc['mean', 'train_f1']:.4f}, std: {mean_std.loc['std', 'train_f1']:.4f}")
print(f"mean val f1:   {mean_std.loc['mean', 'val_f1']:.4f}, std: {mean_std.loc['std', 'val_f1']:.4f}")
print(f"mean val precision: {mean_std.loc['mean', 'val_precision']:.4f}, std: {mean_std.loc['std', 'val_precision']:.4f}")
print(f"mean val recall:    {mean_std.loc['mean', 'val_recall']:.4f}, std: {mean_std.loc['std', 'val_recall']:.4f}")
print(f"mean val loss:      {mean_std.loc['mean', 'val_loss']:.4f}, std: {mean_std.loc['std', 'val_loss']:.4f}")
```

```
=== Fold-wise Metrics Summary ===
Fold 0: Train F1: 0.7961, Val F1: 0.8004, Val Precision: 0.8139, Val Recall:
0.8199, Val Loss: 0.4160
Fold 1: Train F1: 0.8070, Val F1: 0.8130, Val Precision: 0.8271, Val Recall:
0.8272, Val Loss: 0.3719
Fold 2: Train F1: 0.7902, Val F1: 0.8088, Val Precision: 0.8262, Val Recall:
0.8292, Val Loss: 0.3966
Fold 3: Train F1: 0.8014, Val F1: 0.7849, Val Precision: 0.8037, Val Recall:
0.8028, Val Loss: 0.4301
Fold 4: Train F1: 0.7890, Val F1: 0.8014, Val Precision: 0.8226, Val Recall:
0.8220, Val Loss: 0.4070

=== Mean and Standard Deviation Across Folds ===
Mean Train F1: 0.7967, Std: 0.0076
Mean Val F1:   0.8017, Std: 0.0107
Mean Val Precision: 0.8187, Std: 0.0099
Mean Val Recall:    0.8202, Std: 0.0104
Mean Val Loss:      0.4043, Std: 0.0219
```

[13]:
```python
# identify best fold based on final validation f1
best_fold = None
```

```
best_val_f1 = -1

for fold in range(num_folds):
    path = os.path.join(save_dir, f"fold_{fold}", "version_0", "metrics.csv")
    if os.path.exists(path):
        df = pd.read_csv(path)
        df_val = df[df["val_f1"].notna()].copy().reset_index(drop=True)
        if not df_val.empty:
            final_val_f1 = df_val.iloc[-1]["val_f1"]
            if final_val_f1 > best_val_f1:
                best_val_f1 = final_val_f1
                best_fold = fold

# plot training and validation f1 for the best fold
if best_fold is not None:
    path = os.path.join(save_dir, f"fold_{best_fold}", "version_0", "metrics.
↪csv")
    df = pd.read_csv(path)
    df_train = df[df["train_f1"].notna()].copy().reset_index(drop=True)
    df_val = df[df["val_f1"].notna()].copy().reset_index(drop=True)

    plt.figure(figsize=(10, 6))
    plt.plot(df_train["epoch"], df_train["train_f1"], label="train f1",␣
↪color="#2E8B57", linewidth=2.5)
    plt.plot(df_val["epoch"], df_val["val_f1"], label="validation f1",␣
↪color="#B22222", linewidth=2.5)

    plt.xlabel("epoch", fontsize=14)
    plt.ylabel("f1 score", fontsize=14)
    plt.title(f"training and validation f1 score (best fold {best_fold})",␣
↪fontsize=18, fontweight="bold")
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)

    # updated legend position
    plt.legend(fontsize=12, loc="lower right")

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



1.0.5 EfficientNet-B0 + Best Hyperparameter

[23]:
```
# configuration
save_dir = r"C:/Users/Xuxu/Desktop/Master Thesis/BestHyperEfficientNetB0Full"
class_idx_path = r"C:/Users/Xuxu/Desktop/Master Thesis/OptunaDensenetFull/
  ↪class_to_idx.json"
num_folds = 5

# load class mapping
with open(class_idx_path, "r") as f:
    class_to_idx = json.load(f)
idx_to_class = {v: k for k, v in class_to_idx.items()}
class_names = [idx_to_class[i] for i in range(len(idx_to_class))]

# load predictions across folds
all_preds_all_folds = []
all_targets_all_folds = []

for fold in range(num_folds):
    fold_dir = os.path.join(save_dir, f"fold_{fold}", "version_0")
    preds_path = os.path.join(fold_dir, "all_preds.npy")
    targets_path = os.path.join(fold_dir, "all_targets.npy")

    if os.path.exists(preds_path) and os.path.exists(targets_path):
        all_preds_all_folds.extend(np.load(preds_path))
```

```
        all_targets_all_folds.extend(np.load(targets_path))

# convert predictions and targets to numpy arrays
all_preds_all_folds = np.array(all_preds_all_folds)
all_targets_all_folds = np.array(all_targets_all_folds)

# fold-wise final metrics summary
metrics_cols = [
    "train_f1", "train_precision", "train_recall", "train_loss",
    "val_f1", "val_precision", "val_recall", "val_loss"
]
metrics_summary = []

for fold in range(num_folds):
    metrics_file = os.path.join(save_dir, f"fold_{fold}", "version_0", "metrics.
↪csv")
    if os.path.exists(metrics_file):
        df = pd.read_csv(metrics_file)

        df_val = df.dropna(subset=["val_f1"])
        df_train = df.dropna(subset=["train_f1"])

        last_val = df_val.sort_values("epoch").groupby("epoch").tail(1).iloc[-1]
        last_train = df_train.sort_values("epoch").groupby("epoch").tail(1).
↪iloc[-1]

        row = [
            last_train.get("train_f1", np.nan),
            last_train.get("train_precision", np.nan),
            last_train.get("train_recall", np.nan),
            last_train.get("train_loss", np.nan),
            last_val.get("val_f1", np.nan),
            last_val.get("val_precision", np.nan),
            last_val.get("val_recall", np.nan),
            last_val.get("val_loss", np.nan),
        ]
        metrics_summary.append(row)

# convert to dataframe
df_summary = pd.DataFrame(metrics_summary, columns=metrics_cols)
df_summary.index = [f"fold {i}" for i in range(num_folds)]

# display fold-wise metrics summary
print("\nmetrics summary per fold")
for idx, row in df_summary.iterrows():
    print(f"{idx}: "
          f"train f1: {row['train_f1']:.4f}, "
```

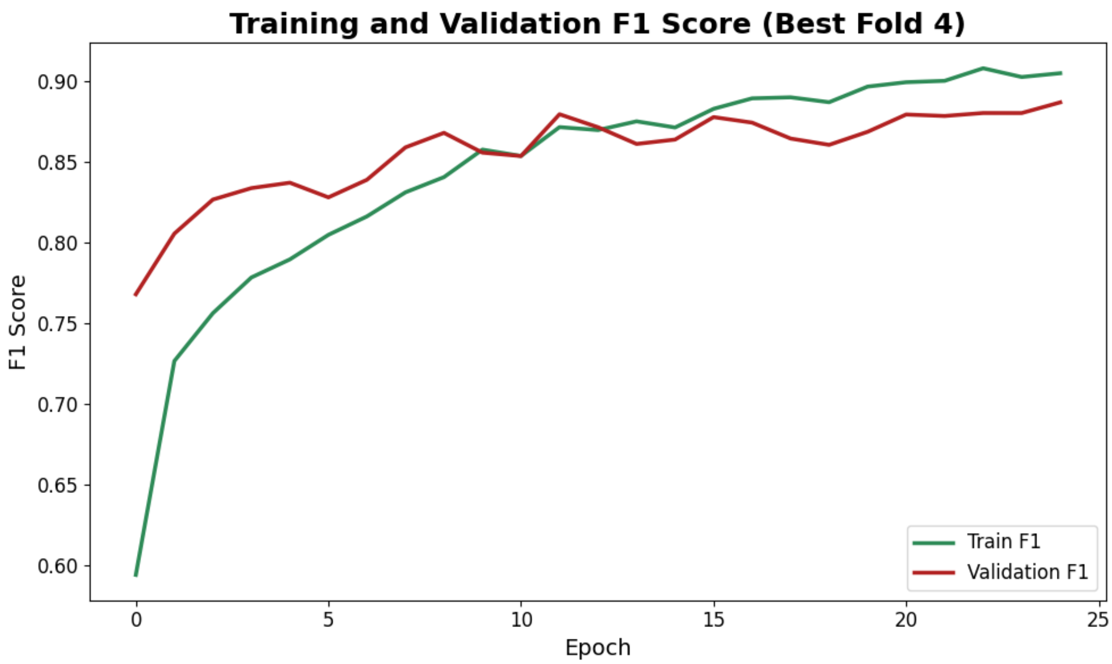
```
          f"val f1: {row['val_f1']:.4f}, "
          f"val precision: {row['val_precision']:.4f}, "
          f"val recall: {row['val_recall']:.4f}, "
          f"val loss: {row['val_loss']:.4f}")

# display mean and standard deviation across folds
print("\nmean and standard deviation across folds")
mean_std = df_summary.agg(["mean", "std"]).round(4)

print(f"mean train f1: {mean_std.loc['mean', 'train_f1']:.4f}, std: {mean_std.
 ↪loc['std', 'train_f1']:.4f}")
print(f"mean val f1:   {mean_std.loc['mean', 'val_f1']:.4f}, std: {mean_std.
 ↪loc['std', 'val_f1']:.4f}")
print(f"mean val precision: {mean_std.loc['mean', 'val_precision']:.4f}, std:␣
 ↪{mean_std.loc['std', 'val_precision']:.4f}")
print(f"mean val recall:    {mean_std.loc['mean', 'val_recall']:.4f}, std:␣
 ↪{mean_std.loc['std', 'val_recall']:.4f}")
print(f"mean val loss:      {mean_std.loc['mean', 'val_loss']:.4f}, std:␣
 ↪{mean_std.loc['std', 'val_loss']:.4f}")
```

```
Metrics Summary Per Fold
Fold 0: Train F1: 0.9048, Val F1: 0.8801, Val Precision: 0.8894, Val Recall:
0.8913, Val Loss: 0.3201
Fold 1: Train F1: 0.9099, Val F1: 0.8795, Val Precision: 0.8897, Val Recall:
0.8893, Val Loss: 0.3106
Fold 2: Train F1: 0.9194, Val F1: 0.8555, Val Precision: 0.8673, Val Recall:
0.8683, Val Loss: 0.3483
Fold 3: Train F1: 0.9097, Val F1: 0.8653, Val Precision: 0.8754, Val Recall:
0.8760, Val Loss: 0.3582
Fold 4: Train F1: 0.9052, Val F1: 0.8871, Val Precision: 0.8972, Val Recall:
0.8969, Val Loss: 0.3034

Mean and Standard Deviation Across Folds
Mean Train F1: 0.9098, Std: 0.0059
Mean Val F1:   0.8735, Std: 0.0128
Mean Val Precision: 0.8838, Std: 0.0121
Mean Val Recall:    0.8844, Std: 0.0118
Mean Val Loss:      0.3281, Std: 0.0239
```

[15]:

```
# identify best fold based on final validation f1
best_fold = None
best_val_f1 = -1

for fold in range(num_folds):
    path = os.path.join(save_dir, f"fold_{fold}", "version_0", "metrics.csv")
    if os.path.exists(path):
```

```
        df = pd.read_csv(path)
        df_val = df[df["val_f1"].notna()].copy().reset_index(drop=True)
        if not df_val.empty:
            final_val_f1 = df_val.iloc[-1]["val_f1"]
            if final_val_f1 > best_val_f1:
                best_val_f1 = final_val_f1
                best_fold = fold

# plot training and validation f1 for the best fold
if best_fold is not None:
    path = os.path.join(save_dir, f"fold_{best_fold}", "version_0", "metrics.
 ↪csv")
    df = pd.read_csv(path)
    df_train = df[df["train_f1"].notna()].copy().reset_index(drop=True)
    df_val = df[df["val_f1"].notna()].copy().reset_index(drop=True)

    plt.figure(figsize=(10, 6))
    plt.plot(df_train["epoch"], df_train["train_f1"], label="train f1",␣
 ↪color="#2E8B57", linewidth=2.5)
    plt.plot(df_val["epoch"], df_val["val_f1"], label="validation f1",␣
 ↪color="#B22222", linewidth=2.5)

    plt.xlabel("epoch", fontsize=14)
    plt.ylabel("f1 score", fontsize=14)
    plt.title(f"training and validation f1 score (best fold {best_fold})",␣
 ↪fontsize=18, fontweight="bold")
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)

    # updated legend position
    plt.legend(fontsize=12, loc="lower right")

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



1.0.6 ConvNeXtTiny + Best Hyperparameter

[24]:
```
# imports
import os
import json
import numpy as np
import pandas as pd

# configuration
save_dir = r"C:/Users/Xuxu/Desktop/Master Thesis/BestHyperConvNeXtFull"
class_idx_path = r"C:/Users/Xuxu/Desktop/Master Thesis/OptunaDensenetFull/
↪class_to_idx.json"
num_folds = 5

# load class mapping
with open(class_idx_path, "r") as f:
    class_to_idx = json.load(f)
idx_to_class = {v: k for k, v in class_to_idx.items()}
class_names = [idx_to_class[i] for i in range(len(idx_to_class))]

# load predictions across folds
all_preds_all_folds = []
all_targets_all_folds = []

for fold in range(num_folds):
```

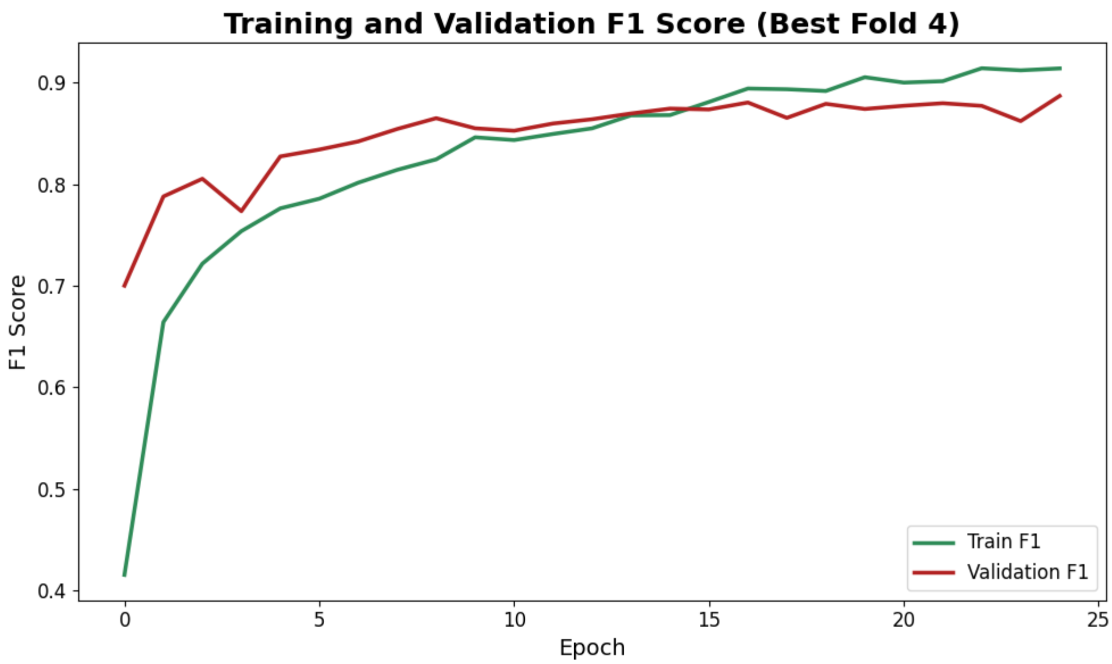
```python
    fold_dir = os.path.join(save_dir, f"fold_{fold}", "version_0")
    preds_path = os.path.join(fold_dir, "all_preds.npy")
    targets_path = os.path.join(fold_dir, "all_targets.npy")

    if os.path.exists(preds_path) and os.path.exists(targets_path):
        all_preds_all_folds.extend(np.load(preds_path))
        all_targets_all_folds.extend(np.load(targets_path))

# convert predictions and targets to numpy arrays
all_preds_all_folds = np.array(all_preds_all_folds)
all_targets_all_folds = np.array(all_targets_all_folds)

# fold-wise final metrics summary
metrics_cols = [
    "train_f1", "train_precision", "train_recall", "train_loss",
    "val_f1", "val_precision", "val_recall", "val_loss"
]
metrics_summary = []

for fold in range(num_folds):
    metrics_file = os.path.join(save_dir, f"fold_{fold}", "version_0", "metrics.
↪csv")
    if os.path.exists(metrics_file):
        df = pd.read_csv(metrics_file)

        df_val = df.dropna(subset=["val_f1"])
        df_train = df.dropna(subset=["train_f1"])

        last_val = df_val.sort_values("epoch").groupby("epoch").tail(1).iloc[-1]
        last_train = df_train.sort_values("epoch").groupby("epoch").tail(1).
↪iloc[-1]

        row = [
            last_train.get("train_f1", np.nan),
            last_train.get("train_precision", np.nan),
            last_train.get("train_recall", np.nan),
            last_train.get("train_loss", np.nan),
            last_val.get("val_f1", np.nan),
            last_val.get("val_precision", np.nan),
            last_val.get("val_recall", np.nan),
            last_val.get("val_loss", np.nan),
        ]
        metrics_summary.append(row)

# convert to dataframe
df_summary = pd.DataFrame(metrics_summary, columns=metrics_cols)
df_summary.index = [f"fold {i}" for i in range(num_folds)]
```

```python
# display fold-wise metrics summary
print("\nmetrics summary per fold")
for idx, row in df_summary.iterrows():
    print(f"{idx}: "
          f"train f1: {row['train_f1']:.4f}, "
          f"val f1: {row['val_f1']:.4f}, "
          f"val precision: {row['val_precision']:.4f}, "
          f"val recall: {row['val_recall']:.4f}, "
          f"val loss: {row['val_loss']:.4f}")

# display mean and standard deviation across folds
print("\nmean and standard deviation across folds")
mean_std = df_summary.agg(["mean", "std"]).round(4)

print(f"mean train f1: {mean_std.loc['mean', 'train_f1']:.4f}, std: {mean_std.
 ↪loc['std', 'train_f1']:.4f}")
print(f"mean val f1:   {mean_std.loc['mean', 'val_f1']:.4f}, std: {mean_std.
 ↪loc['std', 'val_f1']:.4f}")
print(f"mean val precision: {mean_std.loc['mean', 'val_precision']:.4f}, std:␣
 ↪{mean_std.loc['std', 'val_precision']:.4f}")
print(f"mean val recall:    {mean_std.loc['mean', 'val_recall']:.4f}, std:␣
 ↪{mean_std.loc['std', 'val_recall']:.4f}")
print(f"mean val loss:      {mean_std.loc['mean', 'val_loss']:.4f}, std:␣
 ↪{mean_std.loc['std', 'val_loss']:.4f}")
```

```
Metrics Summary Per Fold
Fold 0: Train F1: 0.9149, Val F1: 0.8730, Val Precision: 0.8849, Val Recall:
0.8831, Val Loss: 0.2966
Fold 1: Train F1: 0.9163, Val F1: 0.8613, Val Precision: 0.8712, Val Recall:
0.8731, Val Loss: 0.3134
Fold 2: Train F1: 0.9168, Val F1: 0.8662, Val Precision: 0.8765, Val Recall:
0.8779, Val Loss: 0.2894
Fold 3: Train F1: 0.9137, Val F1: 0.8829, Val Precision: 0.8941, Val Recall:
0.8938, Val Loss: 0.2917
Fold 4: Train F1: 0.9137, Val F1: 0.8867, Val Precision: 0.8972, Val Recall:
0.8974, Val Loss: 0.2541

Mean and Standard Deviation Across Folds
Mean Train F1: 0.9151, Std: 0.0015
Mean Val F1:   0.8740, Std: 0.0108
Mean Val Precision: 0.8848, Std: 0.0111
Mean Val Recall:    0.8851, Std: 0.0103
Mean Val Loss:      0.2890, Std: 0.0217
```

[49]:

```python
# identify best fold based on final validation f1
best_fold = None
best_val_f1 = -1

for fold in range(num_folds):
    path = os.path.join(save_dir, f"fold_{fold}", "version_0", "metrics.csv")
    if os.path.exists(path):
        df = pd.read_csv(path)
        df_val = df[df["val_f1"].notna()].copy().reset_index(drop=True)
        if not df_val.empty:
            final_val_f1 = df_val.iloc[-1]["val_f1"]
            if final_val_f1 > best_val_f1:
                best_val_f1 = final_val_f1
                best_fold = fold

# plot training and validation f1 for the best fold
if best_fold is not None:
    path = os.path.join(save_dir, f"fold_{best_fold}", "version_0", "metrics.
↪csv")
    df = pd.read_csv(path)
    df_train = df[df["train_f1"].notna()].copy().reset_index(drop=True)
    df_val = df[df["val_f1"].notna()].copy().reset_index(drop=True)

    plt.figure(figsize=(10, 6))
    plt.plot(df_train["epoch"], df_train["train_f1"], label="train f1",␣
↪color="#2E8B57", linewidth=2.5)
    plt.plot(df_val["epoch"], df_val["val_f1"], label="validation f1",␣
↪color="#B22222", linewidth=2.5)

    plt.xlabel("epoch", fontsize=14)
    plt.ylabel("f1 score", fontsize=14)
    plt.title(f"training and validation f1 score (best fold {best_fold})",␣
↪fontsize=18, fontweight="bold")
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)

    # updated legend position
    plt.legend(fontsize=12, loc="lower right")

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



1.0.7 ConvNext-Tiny(Supervised)

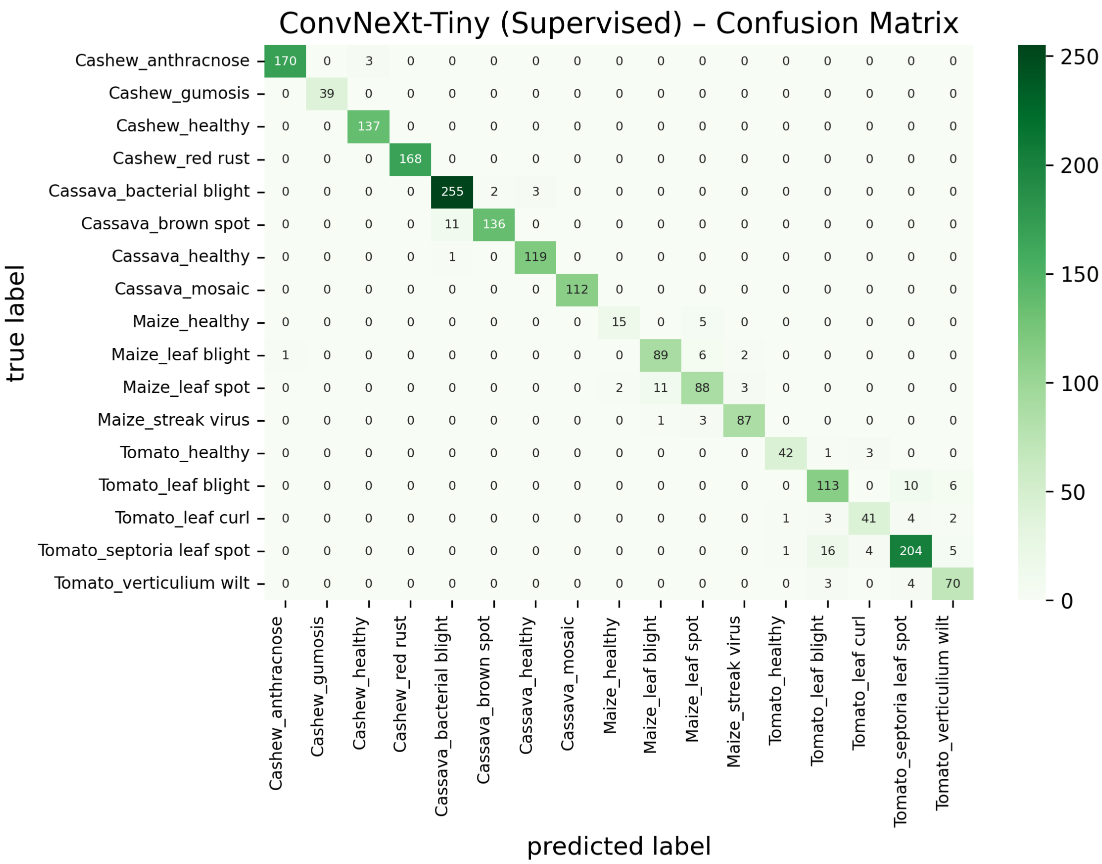
[50]:
```
# paths
version_dir = r"C:/Users/Xuxu/Desktop/Master Thesis/
  ↪SupervisedBaselineVer2Epoch100/single_run/version_0"
index_dir = r"C:/Users/Xuxu/Desktop/Master Thesis/OptunaDensenetFull"
save_dir = os.path.join(version_dir, "figures")

# create save_dir if needed
os.makedirs(save_dir, exist_ok=True)

# file paths
preds_path = os.path.join(version_dir, "all_preds.npy")
targets_path = os.path.join(version_dir, "all_targets.npy")
probs_path = os.path.join(version_dir, "all_probs.npy")
metrics_path = os.path.join(version_dir, "test_metrics.json")

# load class mapping
with open(os.path.join(index_dir, "class_to_idx.json")) as f:
    class_to_idx = json.load(f)
idx_to_class = {v: k for k, v in class_to_idx.items()}
class_names = [idx_to_class[i] for i in range(len(idx_to_class))]

# load predictions and targets
all_preds = np.load(preds_path)
```

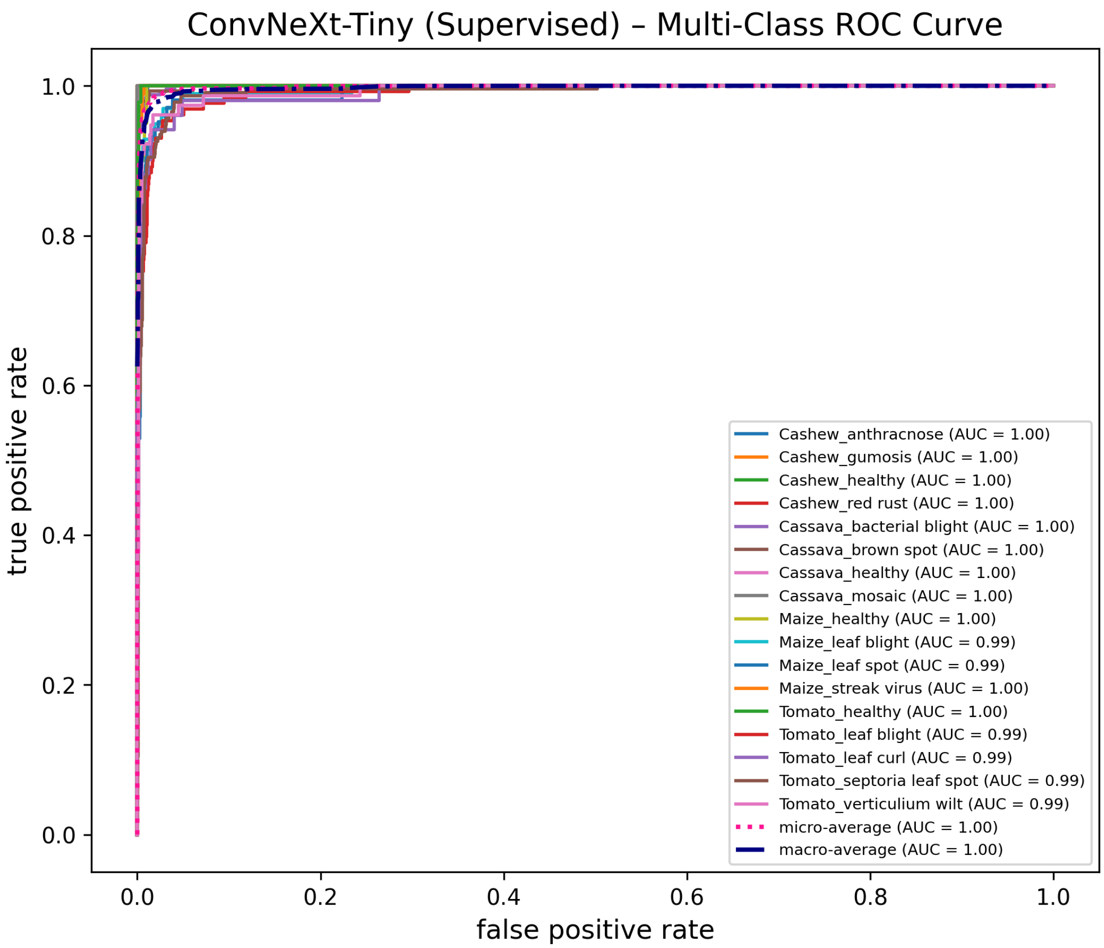
```
all_targets = np.load(targets_path)
all_probs = np.load(probs_path)

# classification report
print("\n# classification report")
print(classification_report(all_targets, all_preds, target_names=class_names,␣
↪digits=4))

# confusion matrix
cm = confusion_matrix(all_targets, all_preds)
plt.figure(figsize=(8, 6), dpi=300)
sns.heatmap(
    cm,
    annot=True,
    fmt="d",
    cmap="Greens",
    xticklabels=class_names,
    yticklabels=class_names,
    annot_kws={"fontsize": 6}
)
plt.title("convnext-tiny (supervised) - confusion matrix", fontsize=14)
plt.xlabel("predicted label", fontsize=12)
plt.ylabel("true label", fontsize=12)
plt.xticks(rotation=90, ha='right', fontsize=8)
plt.yticks(rotation=0, fontsize=8)
plt.tight_layout(pad=1.0)
plt.savefig(os.path.join(save_dir, "confusion_matrix_absolute_cleaned.png"),␣
↪bbox_inches="tight")
plt.show()

# roc curves (multi-class)
n_classes = len(class_names)
y_true_bin = label_binarize(all_targets, classes=list(range(n_classes)))

# compute roc and auc
fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_true_bin[:, i], all_probs[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# micro-average
fpr["micro"], tpr["micro"], _ = roc_curve(y_true_bin.ravel(), all_probs.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# macro-average
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
```

```
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

# save auc values
roc_auc_json = {class_names[i]: roc_auc[i] for i in range(n_classes)}
roc_auc_json["micro"] = roc_auc["micro"]
roc_auc_json["macro"] = roc_auc["macro"]
with open(os.path.join(save_dir, "roc_auc_scores.json"), "w") as f:
    json.dump(roc_auc_json, f, indent=4)

# plot roc curves
plt.figure(figsize=(7, 6), dpi=300)
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f"{class_names[i]} (auc = {roc_auc[i]:.
 ↪2f})", linewidth=1.5)
plt.plot(fpr["micro"], tpr["micro"], label=f"micro-average (auc =␣
 ↪{roc_auc['micro']:.2f})", color='deeppink', linestyle=':', linewidth=2)
plt.plot(fpr["macro"], tpr["macro"], label=f"macro-average (auc =␣
 ↪{roc_auc['macro']:.2f})", color='navy', linestyle='-.', linewidth=2)
plt.title("convnext-tiny (supervised) - multi-class roc curve", fontsize=14)
plt.xlabel("false positive rate", fontsize=12)
plt.ylabel("true positive rate", fontsize=12)
plt.legend(loc="lower right", fontsize=7)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.tight_layout(pad=0.5)
plt.savefig(os.path.join(save_dir, "roc_curve_multiclass.png"),␣
 ↪bbox_inches="tight")
plt.show()
```

```
# classification report
                           precision    recall  f1-score   support

       Cashew_anthracnose     0.9942    0.9827    0.9884       173
           Cashew_gumosis     1.0000    1.0000    1.0000        39
           Cashew_healthy     0.9786    1.0000    0.9892       137
          Cashew_red rust     1.0000    1.0000    1.0000       168
 Cassava_bacterial blight     0.9551    0.9808    0.9677       260
       Cassava_brown spot     0.9855    0.9252    0.9544       147
          Cassava_healthy     0.9754    0.9917    0.9835       120
           Cassava_mosaic     1.0000    1.0000    1.0000       112
            Maize_healthy     0.8824    0.7500    0.8108        20
```

Maize_leaf blight	0.8812	0.9082	0.8945	98
Maize_leaf spot	0.8627	0.8462	0.8544	104
Maize_streak virus	0.9457	0.9560	0.9508	91
Tomato_healthy	0.9545	0.9130	0.9333	46
Tomato_leaf blight	0.8309	0.8760	0.8528	129
Tomato_leaf curl	0.8542	0.8039	0.8283	51
Tomato_septoria leaf spot	0.9189	0.8870	0.9027	230
Tomato_verticulium wilt	0.8434	0.9091	0.8750	77
accuracy			0.9416	2002
macro avg	0.9331	0.9253	0.9286	2002
weighted avg	0.9421	0.9416	0.9415	2002

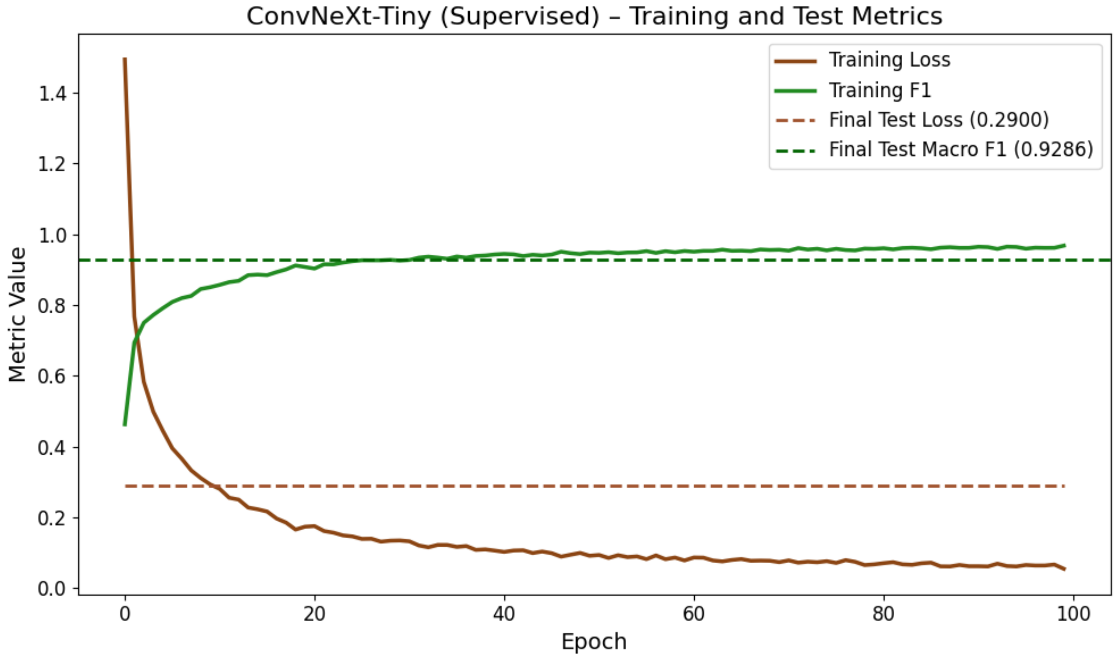




[51]:
```
# load training metrics and drop any rows missing training loss or f1
df = pd.read_csv(metrics_path)
df_epoch = df.dropna(subset=["train_loss", "train_f1"]).reset_index(drop=True)

# load predicted and true labels, then calculate macro f1-score
all_preds = np.load(preds_path)
all_targets = np.load(targets_path)
report = classification_report(all_targets, all_preds, output_dict=True,
 ↪zero_division=0)
macro_f1 = report["macro avg"]["f1-score"]

# create plot for training and test metrics
plt.figure(figsize=(10, 6))

# plot training loss over epochs
plt.plot(
    df_epoch["epoch"],
```

```
    df_epoch["train_loss"],
    label="training loss",
    linestyle="-",
    color="#8B4513",  # saddlebrown
    linewidth=2.5
)

# plot training f1 score over epochs
plt.plot(
    df_epoch["epoch"],
    df_epoch["train_f1"],
    label="training f1",
    linestyle="-",
    color="#228B22",  # forestgreen
    linewidth=2.5
)

# plot horizontal line for final test loss if available
if "test_loss" in df.columns and not df["test_loss"].isnull().all():
    test_loss = df["test_loss"].dropna().values[-1]
    plt.hlines(
        y=test_loss,
        xmin=df_epoch["epoch"].min(),
        xmax=df_epoch["epoch"].max(),
        label=f"final test loss ({test_loss:.4f})",
        colors="#A0522D",  # sienna
        linestyles="--",
        linewidth=2.0
    )

# plot horizontal line for final test macro f1 score
plt.axhline(
    y=macro_f1,
    xmin=0,
    xmax=1,
    label=f"final test macro f1 ({macro_f1:.4f})",
    color="#006400",  # darkgreen
    linestyle="--",
    linewidth=2.0
)

# format axes and layout
plt.xlabel("epoch", fontsize=14)
plt.ylabel("metric value", fontsize=14)
plt.title("convnext-tiny (supervised) - training and test metrics", fontsize=16)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
```

```python
plt.legend(fontsize=12, loc="upper right")
plt.tight_layout()

# save the plot and display it
save_path = os.path.join(save_dir, "convnext_training_metrics.png")
plt.savefig(save_path, dpi=300, bbox_inches="tight")
plt.show()
```



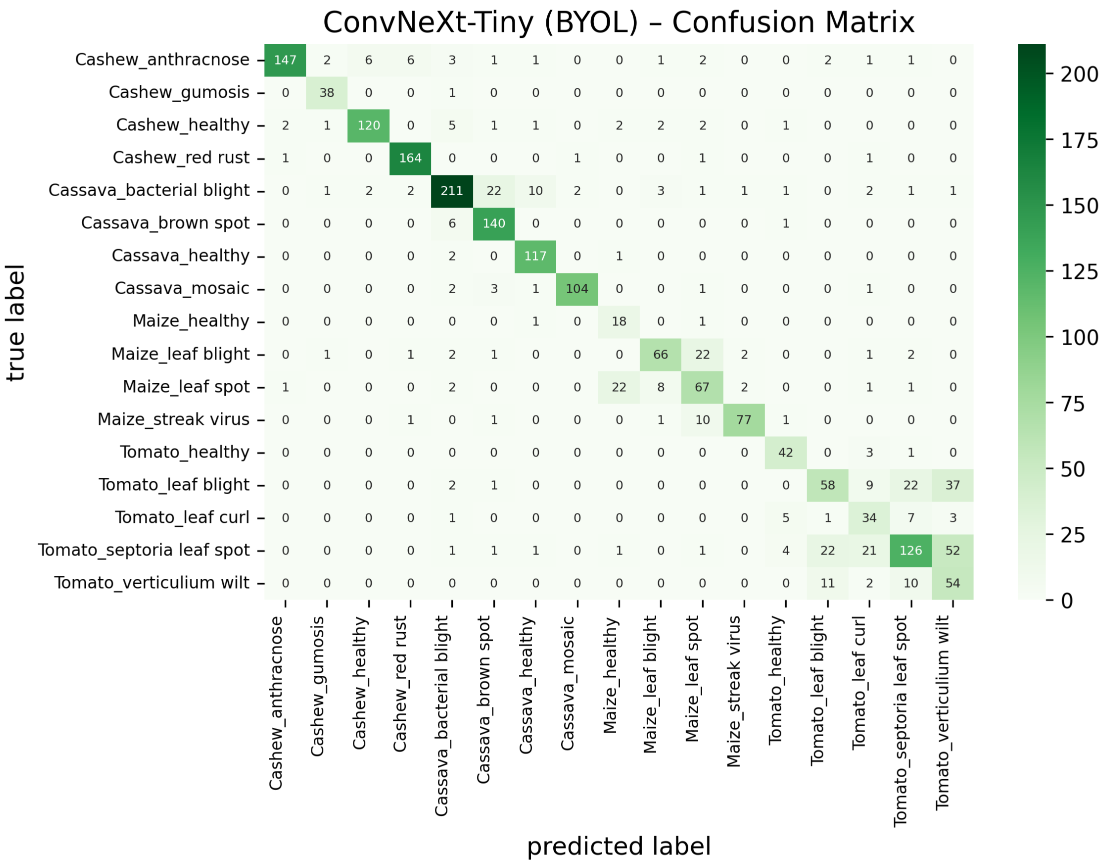
1.0.8 ConvNext-Tiny(BYOL)

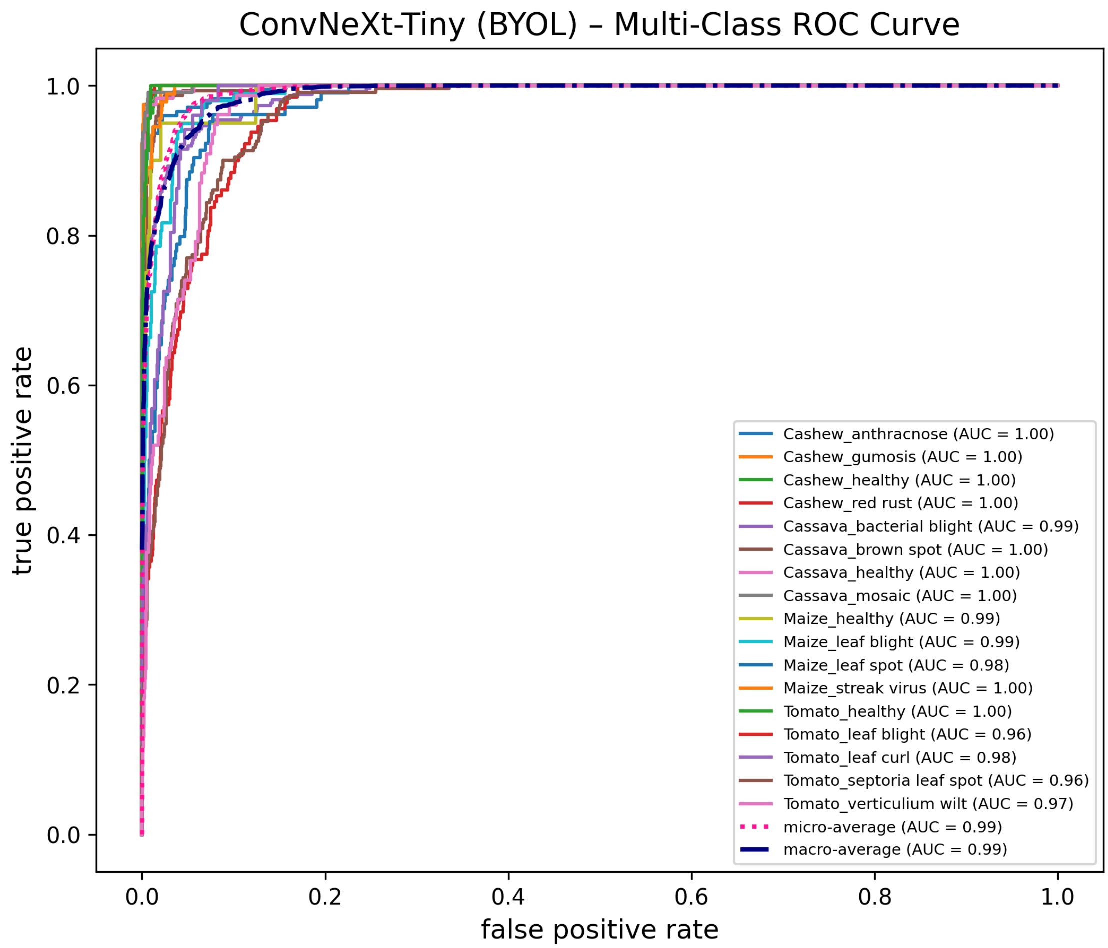
[52]:
```python
# paths
version_dir = r"C:/Users/Xuxu/Desktop/Master Thesis/BYOLBaselineVer2Epoch100/
↪single_run/version_0"
index_dir = r"C:/Users/Xuxu/Desktop/Master Thesis/OptunaDensenetFull"
save_dir = os.path.join(version_dir, "figures")
os.makedirs(save_dir, exist_ok=True)

preds_path = os.path.join(version_dir, "all_preds.npy")
targets_path = os.path.join(version_dir, "all_targets.npy")
probs_path = os.path.join(version_dir, "all_probs.npy")
metrics_path = os.path.join(version_dir, "test_metrics.json")

# load class names
with open(os.path.join(index_dir, "class_to_idx.json")) as f:
    class_to_idx = json.load(f)
```

```
idx_to_class = {v: k for k, v in class_to_idx.items()}
class_names = [idx_to_class[i] for i in range(len(idx_to_class))]

# load predictions and test metrics
all_preds = np.load(preds_path)
all_targets = np.load(targets_path)
all_probs = np.load(probs_path)

print("\n# classification report")
print(classification_report(all_targets, all_preds, target_names=class_names,␣
↪digits=4))

# confusion matrix (absolute)
cm = confusion_matrix(all_targets, all_preds)
plt.figure(figsize=(8, 6), dpi=300)
sns.heatmap(cm, annot=True, fmt="d", cmap="Greens",
            xticklabels=class_names, yticklabels=class_names,
            annot_kws={"fontsize": 6})
plt.title("convnext-tiny (byol) - confusion matrix", fontsize=14)
plt.xlabel("predicted label", fontsize=12)
plt.ylabel("true label", fontsize=12)
plt.xticks(rotation=90, ha='right', fontsize=8)
plt.yticks(rotation=0, fontsize=8)
plt.tight_layout(pad=1.0)
plt.savefig(os.path.join(save_dir, "confusion_matrix_absolute_cleaned.png"),␣
↪bbox_inches="tight")
plt.show()

# roc curve setup
n_classes = len(class_names)
y_true_bin = label_binarize(all_targets, classes=list(range(n_classes)))

# compute per-class roc and auc
fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_true_bin[:, i], all_probs[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# micro-average
fpr["micro"], tpr["micro"], _ = roc_curve(y_true_bin.ravel(), all_probs.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# macro-average
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
```

```python
mean_tpr /= n_classes
fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

# save auc scores
roc_auc_json = {class_names[i]: roc_auc[i] for i in range(n_classes)}
roc_auc_json["micro"] = roc_auc["micro"]
roc_auc_json["macro"] = roc_auc["macro"]
with open(os.path.join(save_dir, "roc_auc_scores.json"), "w") as f:
    json.dump(roc_auc_json, f, indent=4)

# plot roc curves
plt.figure(figsize=(7, 6), dpi=300)
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i],
             label=f"{c
```

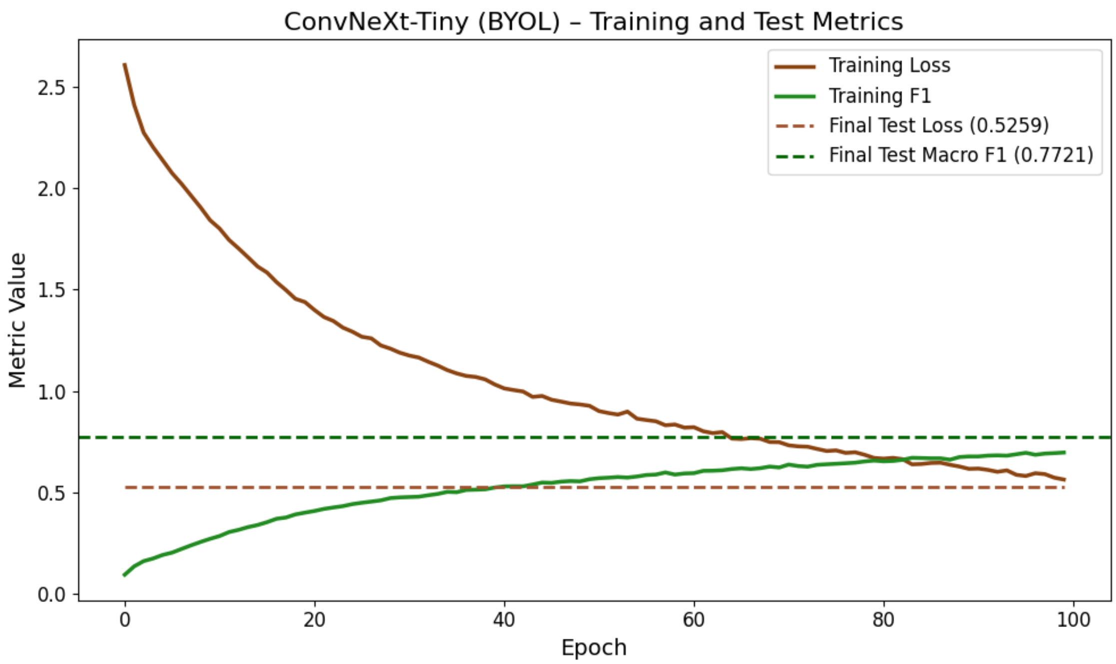
```
# classification report
                             precision    recall  f1-score   support

        Cashew_anthracnose      0.9735    0.8497    0.9074       173
            Cashew_gumosis      0.8837    0.9744    0.9268        39
            Cashew_healthy      0.9375    0.8759    0.9057       137
           Cashew_red rust      0.9425    0.9762    0.9591       168
  Cassava_bacterial blight      0.8866    0.8115    0.8474       260
        Cassava_brown spot      0.8187    0.9524    0.8805       147
           Cassava_healthy      0.8864    0.9750    0.9286       120
            Cassava_mosaic      0.9720    0.9286    0.9498       112
             Maize_healthy      0.4091    0.9000    0.5625        20
          Maize_leaf blight     0.8148    0.6735    0.7374        98
           Maize_leaf spot      0.6204    0.6442    0.6321       104
        Maize_streak virus      0.9390    0.8462    0.8902        91
            Tomato_healthy      0.7636    0.9130    0.8317        46
        Tomato_leaf blight      0.6170    0.4496    0.5202       129
          Tomato_leaf curl      0.4474    0.6667    0.5354        51
Tomato_septoria leaf spot      0.7368    0.5478    0.6284       230
   Tomato_verticulium wilt      0.3673    0.7013    0.4821        77

                  accuracy                          0.7907      2002
                 macro avg      0.7657    0.8051    0.7721      2002
              weighted avg      0.8137    0.7907    0.7944      2002
```


ConvNeXt-Tiny (BYOL) – Confusion Matrix
true label
predicted label
Cashew_anthracnose
Cashew_gumosis
Cashew_healthy
Cashew_red rust
Cassava_bacterial blight
Cassava_brown spot
Cassava_healthy
Cassava_mosaic
Maize_healthy
Maize_leaf blight
Maize_leaf spot
Maize_streak virus
Tomato_healthy
Tomato_leaf blight
Tomato_leaf curl
Tomato_septoria leaf spot
Tomato_verticulium wilt
200
175
150
125
100
75
50
25
0



[53]:
```
# load training metrics and filter rows with valid loss and f1 values
df = pd.read_csv(metrics_path)
df_epoch = df.dropna(subset=["train_loss", "train_f1"]).reset_index(drop=True)

# load predicted and true labels, then compute macro-averaged f1 score
all_preds = np.load(preds_path)
all_targets = np.load(targets_path)
report = classification_report(all_targets, all_preds, output_dict=True,␣
 ↪zero_division=0)
macro_f1 = report["macro avg"]["f1-score"]

# initialize the plot for training and test metrics
plt.figure(figsize=(10, 6))

# plot training loss across epochs
plt.plot(
    df_epoch["epoch"],
```

```
    df_epoch["train_loss"],
    label="training loss",
    linestyle="-",
    color="#8B4513",  # saddlebrown
    linewidth=2.5
)

# plot training f1 score across epochs
plt.plot(
    df_epoch["epoch"],
    df_epoch["train_f1"],
    label="training f1",
    linestyle="-",
    color="#228B22",  # forestgreen
    linewidth=2.5
)

# draw horizontal line for final test loss if available
if "test_loss" in df.columns and not df["test_loss"].isnull().all():
    test_loss = df["test_loss"].dropna().values[-1]
    plt.hlines(
        y=test_loss,
        xmin=df_epoch["epoch"].min(),
        xmax=df_epoch["epoch"].max(),
        label=f"final test loss ({test_loss:.4f})",
        colors="#A0522D",  # sienna
        linestyles="--",
        linewidth=2.0
    )

# draw horizontal line for final macro-averaged test f1 score
plt.axhline(
    y=macro_f1,
    xmin=0,
    xmax=1,
    label=f"final test macro f1 ({macro_f1:.4f})",
    color="#006400",  # darkgreen
    linestyle="--",
    linewidth=2.0
)

# set axis labels and title
plt.xlabel("epoch", fontsize=14)
plt.ylabel("metric value", fontsize=14)
plt.title("convnext-tiny (by
```



1.0.9 Supervised ConvNextTiny + SimCLR

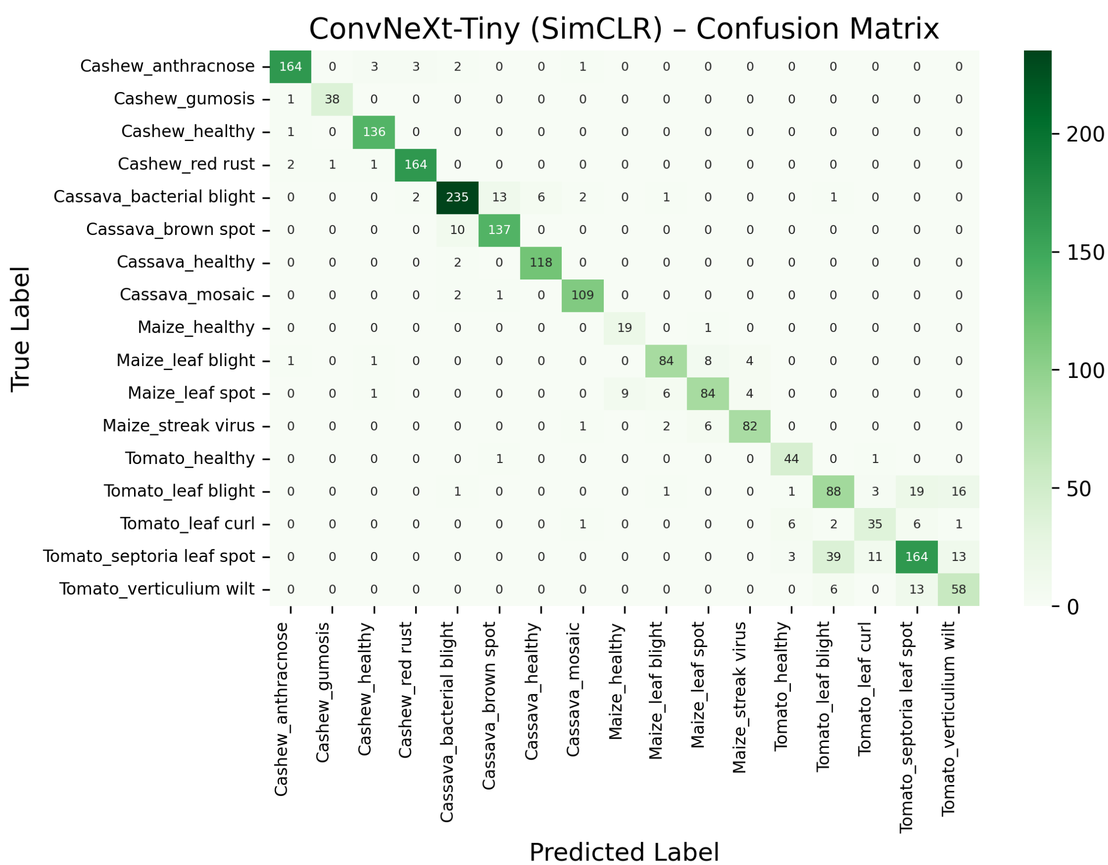
[54]:
```python
# Define paths
VERSION_DIR = r"C:/Users/Xuxu/Desktop/Master Thesis/SIMCLRBaselineEpoch100/
↪single_run/version_0"
INDEX_DIR = r"C:/Users/Xuxu/Desktop/Master Thesis/OptunaDensenetFull"
SAVE_DIR = os.path.join(VERSION_DIR, "figures")

# Create directory to save figures if it doesn't exist
os.makedirs(SAVE_DIR, exist_ok=True)

# Define file paths
PREDS_PATH = os.path.join(VERSION_DIR, "all_preds.npy")
TARGETS_PATH = os.path.join(VERSION_DIR, "all_targets.npy")
PROBS_PATH = os.path.join(VERSION_DIR, "all_probs.npy")
METRICS_PATH = os.path.join(VERSION_DIR, "test_metrics.json")

# Load class names
with open(os.path.join(INDEX_DIR, "class_to_idx.json")) as f:
    class_to_idx = json.load(f)
idx_to_class = {v: k for k, v in class_to_idx.items()}
class_names = [idx_to_class[i] for i in range(len(idx_to_class))]

# Load predictions, targets, and probabilities
all_preds = np.load(PREDS_PATH)
```

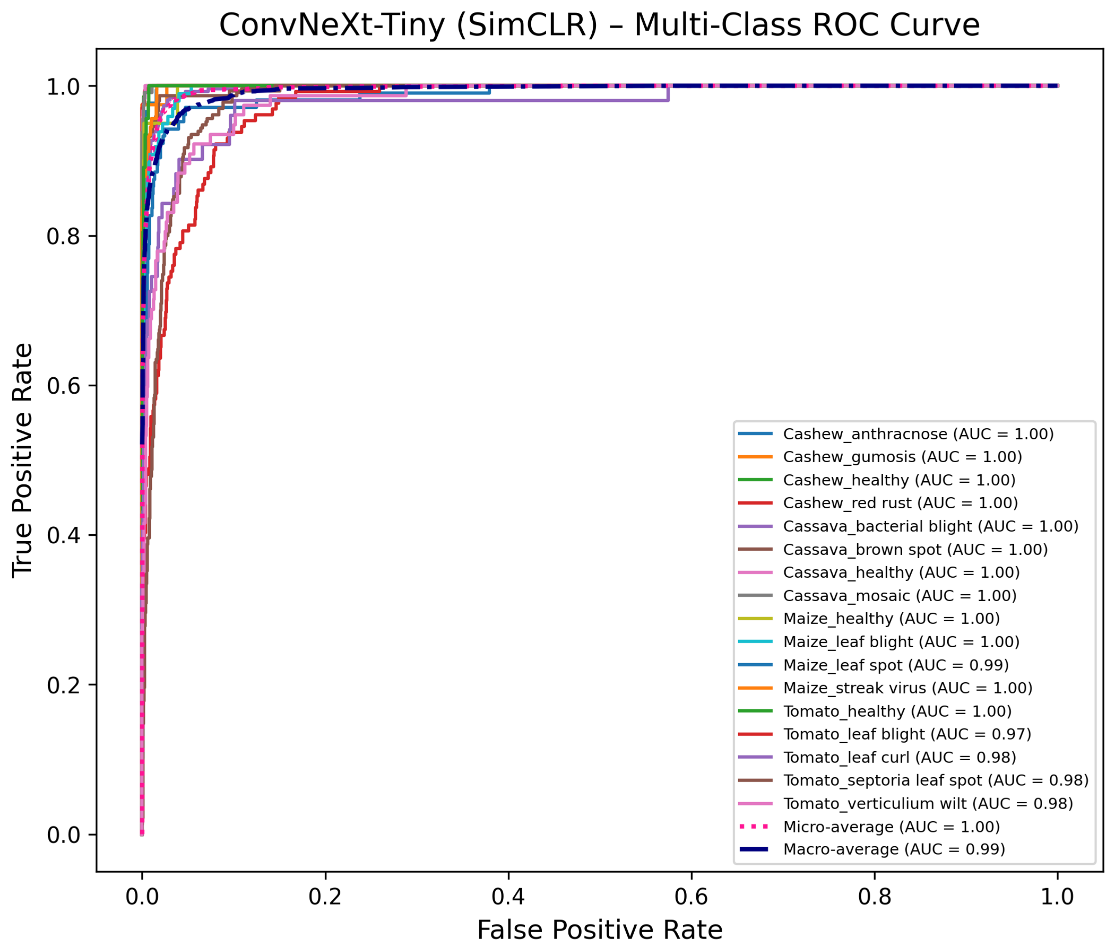
```
all_targets = np.load(TARGETS_PATH)
all_probs = np.load(PROBS_PATH)


# Print classification report
print("\n# Classification Report")
print(classification_report(all_targets, all_preds, target_names=class_names,␣
 ↪digits=4))

# Plot confusion matrix (absolute counts)
cm = confusion_matrix(all_targets, all_preds)
plt.figure(figsize=(8, 6), dpi=300)
sns.heatmap(
    cm,
    annot=True,
    fmt="d",
    cmap="Greens",
    xticklabels=class_names,
    yticklabels=class_names,
    annot_kws={"fontsize": 6}
)
plt.title("ConvNeXt-Tiny (SimCLR) - Confusion Matrix", fontsize=14)
plt.xlabel("Predicted Label", fontsize=12)
plt.ylabel("True Label", fontsize=12)
plt.xticks(rotation=90, ha='right', fontsize=8)
plt.yticks(rotation=0, fontsize=8)
plt.tight_layout(pad=1.0)
plt.savefig(os.path.join(SAVE_DIR, "confusion_matrix_absolute_cleaned.png"),␣
 ↪bbox_inches="tight")
plt.show()

# Prepare for ROC Curve (multi-class)
n_classes = len(class_names)
y_true_bin = label_binarize(all_targets, classes=list(range(n_classes)))

# Compute ROC curve and AUC for each class
fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_true_bin[:, i], all_probs[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and AUC
fpr["micro"], tpr["micro"], _ = roc_curve(y_true_bin.ravel(), all_probs.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Compute macro-average ROC curve and AUC
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
```

```python
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

# Plot ROC curves
plt.figure(figsize=(7, 6), dpi=300)

# Plot ROC for each class
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i],
             label=f"{class_names[i]} (AUC = {roc_auc[i]:.2f})",
             linewidth=1.5)

# Plot micro- and macro-average ROC
plt.plot(fpr["micro"], tpr["micro"],
         label=f"Micro-average (AUC = {roc_auc['micro']:.2f})",
         color='deeppink', linestyle=':', linewidth=2)
plt.plot(fpr["macro"], tpr["macro"],
         label=f"Macro-average (AUC = {roc_auc['macro']:.2f})",
         color='navy', linestyle='-.', linewidth=2)

# Finalize ROC plot
plt.title("ConvNeXt-Tiny (SimCLR) - Multi-Class ROC Curve", fontsize=14)
plt.xlabel("False Positive Rate", fontsize=12)
plt.ylabel("True Positive Rate", fontsize=12)
plt.legend(loc="lower right", fontsize=7)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.tight_layout(pad=0.5)
plt.savefig(os.path.join(SAVE_DIR, "roc_curve_byol.png"), bbox_inches="tight",
 ↪dpi=300)
plt.show()
```

```
# Classification Report
                           precision    recall  f1-score   support

      Cashew_anthracnose      0.9704    0.9480    0.9591       173
          Cashew_gumosis      0.9744    0.9744    0.9744        39
          Cashew_healthy      0.9577    0.9927    0.9749       137
         Cashew_red rust      0.9704    0.9762    0.9733       168
Cassava_bacterial blight      0.9325    0.9038    0.9180       260
      Cassava_brown spot      0.9013    0.9320    0.9164       147
         Cassava_healthy      0.9516    0.9833    0.9672       120
```

Cassava_mosaic	0.9561	0.9732	0.9646	112
Maize_healthy	0.6786	0.9500	0.7917	20
Maize_leaf blight	0.8936	0.8571	0.8750	98
Maize_leaf spot	0.8485	0.8077	0.8276	104
Maize_streak virus	0.9111	0.9011	0.9061	91
Tomato_healthy	0.8148	0.9565	0.8800	46
Tomato_leaf blight	0.6471	0.6822	0.6642	129
Tomato_leaf curl	0.7000	0.6863	0.6931	51
Tomato_septoria leaf spot	0.8119	0.7130	0.7593	230
Tomato_verticulium wilt	0.6591	0.7532	0.7030	77
accuracy			0.8786	2002
macro avg	0.8576	0.8818	0.8675	2002
weighted avg	0.8805	0.8786	0.8785	2002





[55]:
```
# load training metrics and keep only rows with valid training loss and f1 score
df = pd.read_csv(metrics_path)
df_epoch = df.dropna(subset=["train_loss", "train_f1"]).reset_index(drop=True)

# load predicted and true labels, then compute macro-averaged f1 score
all_preds = np.load(preds_path)
all_targets = np.load(targets_path)
report = classification_report(all_targets, all_preds, output_dict=True,␣
 ↪zero_division=0)
macro_f1 = report["macro avg"]["f1-score"]

# create the plot for training and test metrics
plt.figure(figsize=(10, 6))

# plot training loss across epochs
plt.plot(
    df_epoch["epoch"],
```

```
    df_epoch["train_loss"],
    label="training loss",
    linestyle="-",
    color="#8B4513",  # saddlebrown
    linewidth=2.5
)

# plot training f1 score across epochs
plt.plot(
    df_epoch["epoch"],
    df_epoch["train_f1"],
    label="training f1",
    linestyle="-",
    color="#228B22",  # forestgreen
    linewidth=2.5
)

# plot horizontal line for final test loss if available
if "test_loss" in df.columns and not df["test_loss"].isnull().all():
    test_loss = df["test_loss"].dropna().values[-1]
    plt.hlines(
        y=test_loss,
        xmin=df_epoch["epoch"].min(),
        xmax=df_epoch["epoch"].max(),
        label=f"final test loss ({test_loss:.4f})",
        colors="#A0522D",  # sienna
        linestyles="--",
        linewidth=2.0
    )

# plot horizontal line for final macro-averaged test f1 score
plt.axhline(
    y=macro_f1,
    xmin=0,
    xmax=1,
    label=f"final test macro f1 ({macro_f1:.4f})",
    color="#006400",  # darkgreen
    linestyle="--",
    linewidth=2.0
)

# set axis labels and plot title
plt.xlabel("epoch", fontsize=14)
plt.ylabel("metric value", fontsize=14)
plt.title("convnext-tiny (simclr) - training and test metrics", fontsize=16)

# customize tick size and legend
```

```python
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.legend(fontsize=12, loc="upper right")
plt.tight_layout()

# save the plot to file and display it
save_path = os.path.join(save_dir, "convnext_training_metrics.png")
plt.savefig(save_path, dpi=300, bbox_inches="tight")
plt.show()
```