

```
import torch

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

if torch.cuda.is_available():
    print("GPU Name:", torch.cuda.get_device_name(0))
else:
    print("No GPU detected")
```

Using device: cuda
GPU Name: NVIDIA RTX PRO 6000 Blackwell Server Edition

```
from google.colab import drive
drive.mount('/content/drive')

import os
import pandas as pd
from sklearn.model_selection import train_test_split

ZIP_PATH = "/content/drive/MyDrive/APTOS2019/aptos2019-blindness-detection.zip"

!rm -rf /content/aptos2019
!mkdir -p /content/aptos2019
!unzip -q "$ZIP_PATH" -d /content/aptos2019

DATA_DIR = "/content/aptos2019"
CSV_PATH = os.path.join(DATA_DIR, "train.csv")
IMAGE_DIR = os.path.join(DATA_DIR, "train_images")

df = pd.read_csv(CSV_PATH)

df["image_path"] = df["id_code"].apply(
    lambda x: os.path.join(IMAGE_DIR, x + ".png")
)

print("Total records:", len(df))
print("Missing images:", df["image_path"].apply(lambda x: not os.path.exists(x)).sum())

print("\nClass distribution:")
print(df["diagnosis"].value_counts().sort_index())

train_df, temp_df = train_test_split(
    df,
    test_size=0.30,
    random_state=42,
    stratify=df["diagnosis"]
)

val_df, test_df = train_test_split(
    temp_df,
    test_size=0.50,
    random_state=42,
    stratify=temp_df["diagnosis"]
)
```

```
)  
  
print("\nTrain size:", len(train_df))  
print("Validation size:", len(val_df))  
print("Test size:", len(test_df))  
  
print("\nTrain distribution:")  
print(train_df["diagnosis"].value_counts().sort_index())  
  
print("\nValidation distribution:")  
print(val_df["diagnosis"].value_counts().sort_index())  
  
print("\nTest distribution:")  
print(test_df["diagnosis"].value_counts().sort_index())
```

Mounted at /content/drive
Total records: 3662
Missing images: 0

Class distribution:
diagnosis
0 1805
1 370
2 999
3 193
4 295
Name: count, dtype: int64

Train size: 2563
Validation size: 549
Test size: 550

Train distribution:
diagnosis
0 1263
1 259
2 699
3 135
4 207
Name: count, dtype: int64

Validation distribution:
diagnosis
0 271
1 55
2 150
3 29
4 44
Name: count, dtype: int64

Test distribution:
diagnosis
0 271
1 56
2 150
3 29
4 44
Name: count, dtype: int64

```
import torch
from torch.utils.data import Dataset, DataLoader
from PIL import Image
import torchvision.transforms as transforms

IMG_SIZE = 300
BATCH_SIZE = 16

train_transform_convnext = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(degrees=15),
    transforms.ColorJitter(
        brightness=0.2,
        contrast=0.2,
        saturation=0.1
    ),
    transforms.RandomAffine(
        degrees=0,
        translate=(0.05, 0.05),
        scale=(0.95, 1.05)
    ),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])

val_test_transform_convnext = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])

class DRDatasetConvNeXt(Dataset):
    def __init__(self, dataframe, transform=None):
        self.dataframe = dataframe.reset_index(drop=True)
        self.transform = transform

    def __len__(self):
        return len(self.dataframe)

    def __getitem__(self, idx):
        img_path = self.dataframe.loc[idx, "image_path"]
        label = int(self.dataframe.loc[idx, "diagnosis"])

        image = Image.open(img_path).convert("RGB")

        if self.transform:
            image = self.transform(image)
```

```
return image, label
```

```
train_dataset_convnext = DRDatasetConvNext(train_df, transform=train_transform_convnext)
val_dataset_convnext = DRDatasetConvNext(val_df, transform=val_test_transform_convnext)
test_dataset_convnext = DRDatasetConvNext(test_df, transform=val_test_transform_convnext)
```

```
train_loader_convnext = DataLoader(
    train_dataset_convnext,
    batch_size=BATCH_SIZE,
    shuffle=True,
    num_workers=2,
    pin_memory=True
)
```

```
val_loader_convnext = DataLoader(
    val_dataset_convnext,
    batch_size=BATCH_SIZE,
    shuffle=False,
    num_workers=2,
    pin_memory=True
)
```

```
test_loader_convnext = DataLoader(
    test_dataset_convnext,
    batch_size=BATCH_SIZE,
    shuffle=False,
    num_workers=2,
    pin_memory=True
)
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)
```

```
images, labels = next(iter(train_loader_convnext))
print("Images batch shape:", images.shape)
print("Labels batch shape:", labels.shape)
print("Sample labels:", labels[:16])
```

```
Using device: cuda
Images batch shape: torch.Size([16, 3, 300, 300])
Labels batch shape: torch.Size([16])
Sample labels: tensor([0, 4, 0, 2, 0, 0, 0, 0, 2, 4, 2, 0, 0, 2, 4, 2])
```

```
import torch
import torch.nn as nn
import torchvision.models as models
import numpy as np
from sklearn.utils.class_weight import compute_class_weight
import os
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)
```

```
weights = models.ConvNeXt_Tiny_Weights.DEFAULT
convnext_model = models.convnext_tiny(weights=weights)
```

```

in_features = convnext_model.classifier[2].in_features
convnext_model.classifier[2] = nn.Linear(in_features, 5)

convnext_model = convnext_model.to(device)

classes = np.array([0, 1, 2, 3, 4])

class_weights = compute_class_weight(
    class_weight="balanced",
    classes=classes,
    y=train_df["diagnosis"].values
)

class_weights = torch.tensor(class_weights, dtype=torch.float).to(device)

criterion_convnext = nn.CrossEntropyLoss(weight=class_weights)

optimizer_convnext = torch.optim.AdamW(
    convnext_model.parameters(),
    lr=5e-5,
    weight_decay=1e-4
)

scheduler_convnext = torch.optim.lr_scheduler.CosineAnnealingLR(
    optimizer_convnext,
    T_max=12,
    eta_min=1e-6
)

SAVE_DIR = "/content/drive/MyDrive/DR_Results_ConvNeXt"
os.makedirs(SAVE_DIR, exist_ok=True)

best_convnext_model_path = os.path.join(
    SAVE_DIR,
    "best_convnext_tiny.pth"
)

print("ConvNeXt-Tiny model is ready.")
print("Class weights:", class_weights)

```

```

Using device: cuda
Downloading: "https://download.pytorch.org/models/convnext_tiny-983f1562.pth" to /root/.cache/torch/hub/checkpoints/convnext_tiny-983f1562.pth
100%|██████████| 109M/109M [00:04<00:00, 28.4MB/s]
ConvNeXt-Tiny model is ready.
Class weights: tensor([0.4059, 1.9792, 0.7333, 3.7970, 2.4763], device='cuda:0')

```

```

from tqdm import tqdm
import numpy as np
import torch

def train_one_epoch_convnext(model, dataloader, criterion, optimizer, device):
    model.train()

    running_loss = 0.0
    correct = 0

```

```

total = 0

for images, labels in tqdm(dataloader):
    images = images.to(device)
    labels = labels.to(device)

    optimizer.zero_grad()

    outputs = model(images)
    loss = criterion(outputs, labels)

    loss.backward()
    optimizer.step()

    running_loss += loss.item() * images.size(0)

    _, predicted = torch.max(outputs, 1)
    correct += (predicted == labels).sum().item()
    total += labels.size(0)

return running_loss / total, correct / total

def evaluate_convnext(model, dataloader, criterion, device):
    model.eval()

    running_loss = 0.0

```

```

EPOCHS = 12
PATIENCE = 4

best_val_acc = 0.0
patience_counter = 0

for epoch in range(EPOCHS):
    print(f"\nEpoch {epoch+1}/{EPOCHS}")

    train_loss, train_acc = train_one_epoch_convnext(
        convnext_model,
        train_loader_convnext,
        criterion_convnext,
        optimizer_convnext,
        device
    )

    val_loss, val_acc, _, _, _ = evaluate_convnext(
        convnext_model,
        val_loader_convnext,
        criterion_convnext,
        device
    )

    scheduler_convnext.step()

    current_lr = optimizer_convnext.param_groups[0]["lr"]

```

```

print(f"Train Loss: {train_loss:.4f} | Train Accuracy: {train_acc:.4f}")
print(f"Val Loss: {val_loss:.4f} | Val Accuracy: {val_acc:.4f}")
print(f"Learning Rate: {current_lr:.8f}")

if val_acc > best_val_acc:
    best_val_acc = val_acc
    patience_counter = 0
    torch.save(convnext_model.state_dict(), best_convnext_model_path)
    print("Best ConvNeXt-Tiny model saved.")
else:
    patience_counter += 1
    print(f"No improvement. Patience: {patience_counter}/{PATIENCE}")

if patience_counter >= PATIENCE:
    print("Early stopping triggered.")
    break

print("Training completed.")
print("Best Validation Accuracy:", best_val_acc)

```

```

Epoch 1/12
100%|██████████| 161/161 [01:37<00:00, 1.65it/s]
-----
TypeError                                Traceback (most recent call last)
/tmp/ipykernel_1902/982606319.py in <cell line: 0>()
     16     )
     17
--> 18     val_loss, val_acc, _, _, _ = evaluate_convnext(
     19         convnext_model,
     20         val_loader_convnext,

TypeError: cannot unpack non-iterable NoneType object

```

```

from tqdm import tqdm
import numpy as np
import torch

def train_one_epoch_convnext(model, dataloader, criterion, optimizer, device):
    model.train()

    running_loss = 0.0
    correct = 0
    total = 0

    for images, labels in tqdm(dataloader):
        images = images.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()

        outputs = model(images)
        loss = criterion(outputs, labels)

        loss.backward()

```

```
optimizer.step()

running_loss += loss.item() * images.size(0)

_, predicted = torch.max(outputs, 1)
correct += (predicted == labels).sum().item()
total += labels.size(0)

epoch_loss = running_loss / total
epoch_acc = correct / total

return epoch_loss, epoch_acc


def evaluate_convnext(model, dataloader, criterion, device):
    model.eval()

    running_loss = 0.0
    correct = 0
    total = 0

    all_labels = []
    all_preds = []
    all_probs = []

    with torch.no_grad():
        for images, labels in tqdm(dataloader):
            images = images.to(device)
            labels = labels.to(device)

            outputs = model(images)
            loss = criterion(outputs, labels)

            probs = torch.softmax(outputs, dim=1)
            _, predicted = torch.max(outputs, 1)

            running_loss += loss.item() * images.size(0)

            correct += (predicted == labels).sum().item()
            total += labels.size(0)

            all_labels.extend(labels.cpu().numpy())
            all_preds.extend(predicted.cpu().numpy())
            all_probs.extend(probs.cpu().numpy())

    epoch_loss = running_loss / total
    epoch_acc = correct / total

    return (
        epoch_loss,
        epoch_acc,
        np.array(all_labels),
        np.array(all_preds),
        np.array(all_probs)
    )
```

```
print("ConvNeXt training and evaluation functions are fixed.")
```

```
ConvNeXt training and evaluation functions are fixed.
```

```
convnext_model = models.convnext_tiny(weights=weights)
```

```
import torch
import torch.nn as nn
import torchvision.models as models
import numpy as np
from sklearn.utils.class_weight import compute_class_weight
import os

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

weights = models.ConvNeXt_Tiny_Weights.DEFAULT
convnext_model = models.convnext_tiny(weights=weights)

in_features = convnext_model.classifier[2].in_features
convnext_model.classifier[2] = nn.Linear(in_features, 5)

# IMPORTANT: move model to GPU
convnext_model = convnext_model.to(device)

classes = np.array([0, 1, 2, 3, 4])

class_weights = compute_class_weight(
    class_weight="balanced",
    classes=classes,
    y=train_df["diagnosis"].values
)

class_weights = torch.tensor(class_weights, dtype=torch.float).to(device)

criterion_convnext = nn.CrossEntropyLoss(weight=class_weights)

optimizer_convnext = torch.optim.AdamW(
    convnext_model.parameters(),
    lr=5e-5,
    weight_decay=1e-4
)

scheduler_convnext = torch.optim.lr_scheduler.CosineAnnealingLR(
    optimizer_convnext,
    T_max=12,
    eta_min=1e-6
)

SAVE_DIR = "/content/drive/MyDrive/DR_Results_ConvNeXt"
os.makedirs(SAVE_DIR, exist_ok=True)

best_convnext_model_path = os.path.join(
    SAVE_DIR,
```

```

    "best_convnext_tiny.pth"
)

print("ConvNeXt-Tiny model is ready.")
print("Model device:", next(convnext_model.parameters()).device)
print("Class weights device:", class_weights.device)
print("Class weights:", class_weights)

Using device: cuda
ConvNeXt-Tiny model is ready.
Model device: cuda:0
Class weights device: cuda:0
Class weights: tensor([0.4059, 1.9792, 0.7333, 3.7970, 2.4763], device='cuda:0')

```

```

EPOCHS = 12
PATIENCE = 4

best_val_acc = 0.0
patience_counter = 0

for epoch in range(EPOCHS):
    print(f"\nEpoch {epoch+1}/{EPOCHS}")

    train_loss, train_acc = train_one_epoch_convnext(
        convnext_model,
        train_loader_convnext,
        criterion_convnext,
        optimizer_convnext,
        device
    )

    val_loss, val_acc, _, _, _ = evaluate_convnext(
        convnext_model,
        val_loader_convnext,
        criterion_convnext,
        device
    )

    scheduler_convnext.step()

    current_lr = optimizer_convnext.param_groups[0]["lr"]

    print(f"Train Loss: {train_loss:.4f} | Train Accuracy: {train_acc:.4f}")
    print(f"Val Loss: {val_loss:.4f} | Val Accuracy: {val_acc:.4f}")
    print(f"Learning Rate: {current_lr:.8f}")

    if val_acc > best_val_acc:
        best_val_acc = val_acc
        patience_counter = 0
        torch.save(convnext_model.state_dict(), best_convnext_model_path)
        print("Best ConvNeXt-Tiny model saved.")
    else:
        patience_counter += 1
        print(f"No improvement. Patience: {patience_counter}/{PATIENCE}")

    if patience_counter >= PATIENCE:

```

```
        print("Early stopping triggered.")
        break

print("Training completed.")
print("Best Validation Accuracy:", best_val_acc)
```

```
Epoch 6/12
100%|██████████| 161/161 [01:30<00:00, 1.78it/s]
100%|██████████| 35/35 [00:19<00:00, 1.81it/s]
Train Loss: 0.4094 | Train Accuracy: 0.8853
Val Loss: 0.8034 | Val Accuracy: 0.7887
Learning Rate: 0.00002550
No improvement. Patience: 1/4
```

```
Epoch 7/12
100%|██████████| 161/161 [01:32<00:00, 1.74it/s]
100%|██████████| 35/35 [00:19<00:00, 1.81it/s]
Train Loss: 0.3160 | Train Accuracy: 0.9110
Val Loss: 0.8146 | Val Accuracy: 0.8270
Learning Rate: 0.00001916
```

training completed.
Best Validation Accuracy: 0.8542805100182149

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import torch
import numpy as np

# Load best ConvNeXt-Tiny model
convnext_model.load_state_dict(torch.load(best_convnext_model_path, map_location=device))
convnext_model = convnext_model.to(device)

test_loss, test_acc_convnext, y_test_convnext, y_pred_convnext, y_prob_convnext = evaluate_convnext(
    convnext_model,
    test_loader_convnext,
    criterion_convnext,
    device
)

print("==== ConvNeXt-Tiny Final Test Evaluation ====")
print("Test Accuracy:", test_acc_convnext)

print("\nClassification Report:")
print(classification_report(y_test_convnext, y_pred_convnext, zero_division=0))

print("\nConfusion Matrix:")
print(confusion_matrix(y_test_convnext, y_pred_convnext))
```

100%|██████████| 35/35 [00:20<00:00, 1.73it/s]==== ConvNeXt-Tiny Final Test Evaluation ====
Test Accuracy: 0.8454545454545455

Classification Report:

	precision	recall	f1-score	support
0	0.98	0.98	0.98	271
1	0.66	0.59	0.62	56
2	0.76	0.89	0.82	150
3	0.54	0.45	0.49	29
4	0.66	0.43	0.52	44
accuracy			0.85	550
macro avg	0.72	0.67	0.69	550
weighted avg	0.84	0.85	0.84	550

Confusion Matrix:

```
[[266  5  0  0  0]
 [ 3 33 19  0  1]
 [ 1 11 134  4  0]
 [ 0  0  7 13  9]
 [ 1  1 16  7 19]]
```

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import os
```

```
SAVE_DIR = "/content/drive/MyDrive/DR_Results_ConvNeXt"
os.makedirs(SAVE_DIR, exist_ok=True)

class_names = [
    "No DR",
    "Mild",
    "Moderate",
    "Severe",
    "Proliferative DR"
]

cm = confusion_matrix(y_test_convnext, y_pred_convnext)

plt.figure(figsize=(9, 7))
disp = ConfusionMatrixDisplay(
    confusion_matrix=cm,
    display_labels=class_names
)

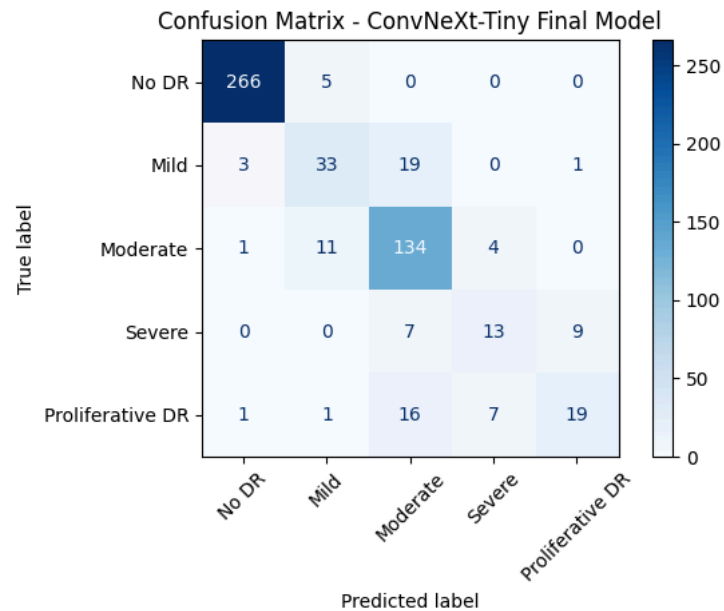
disp.plot(
    cmap="Blues",
    values_format="d",
    xticks_rotation=45
)

plt.title("Confusion Matrix - ConvNeXt-Tiny Final Model")
plt.tight_layout()

cm_path = os.path.join(SAVE_DIR, "convnext_confusion_matrix.png")
plt.savefig(cm_path, dpi=300, bbox_inches="tight")
plt.show()

print("Saved at:", cm_path)
```

<Figure size 900x700 with 0 Axes>



Saved at: /content/drive/MyDrive/DR_Results_ConvNeXt/convnext_confusion_matrix.png

```
cm_normalized = cm.astype("float") / cm.sum(axis=1)[:, np.newaxis]

plt.figure(figsize=(9, 7))
disp = ConfusionMatrixDisplay(
    confusion_matrix=cm_normalized,
    display_labels=class_names
)

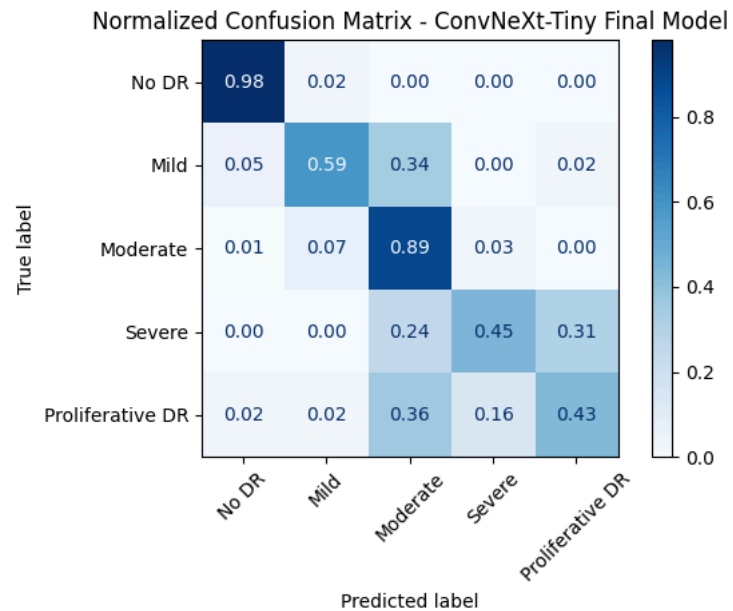
disp.plot(
    cmap="Blues",
    values_format=".2f",
    xticks_rotation=45
)

plt.title("Normalized Confusion Matrix - ConvNeXt-Tiny Final Model")
plt.tight_layout()

norm_cm_path = os.path.join(SAVE_DIR, "convnext_normalized_confusion_matrix.png")
plt.savefig(norm_cm_path, dpi=300, bbox_inches="tight")
plt.show()

print("Saved at:", norm_cm_path)
```

<Figure size 900x700 with 0 Axes>



Saved at: /content/drive/MyDrive/DR_Results_ConvNeXt/convnext_normalized_confusion_matrix.png

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.preprocessing import label_binarize
import numpy as np
import os

# Binarize true labels
y_test_bin = label_binarize(y_test_convnext, classes=[0, 1, 2, 3, 4])

plt.figure(figsize=(9, 7))

auc_scores = []

for i in range(5):
    fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_prob_convnext[:, i])
    roc_auc = auc(fpr, tpr)
    auc_scores.append(roc_auc)

    plt.plot(
        fpr,
        tpr,
        linewidth=2,
        label=f"{class_names[i]} AUC = {roc_auc:.3f}"
    )

plt.plot(
    [0, 1],
    [0, 1],
```

```
        linestyle="--",
        linewidth=2,
        label="Random Classifier"
    )

    macro_auc = roc_auc_score(
        y_test_bin,
        y_prob_convnext,
        average="macro",
        multi_class="ovr"
    )

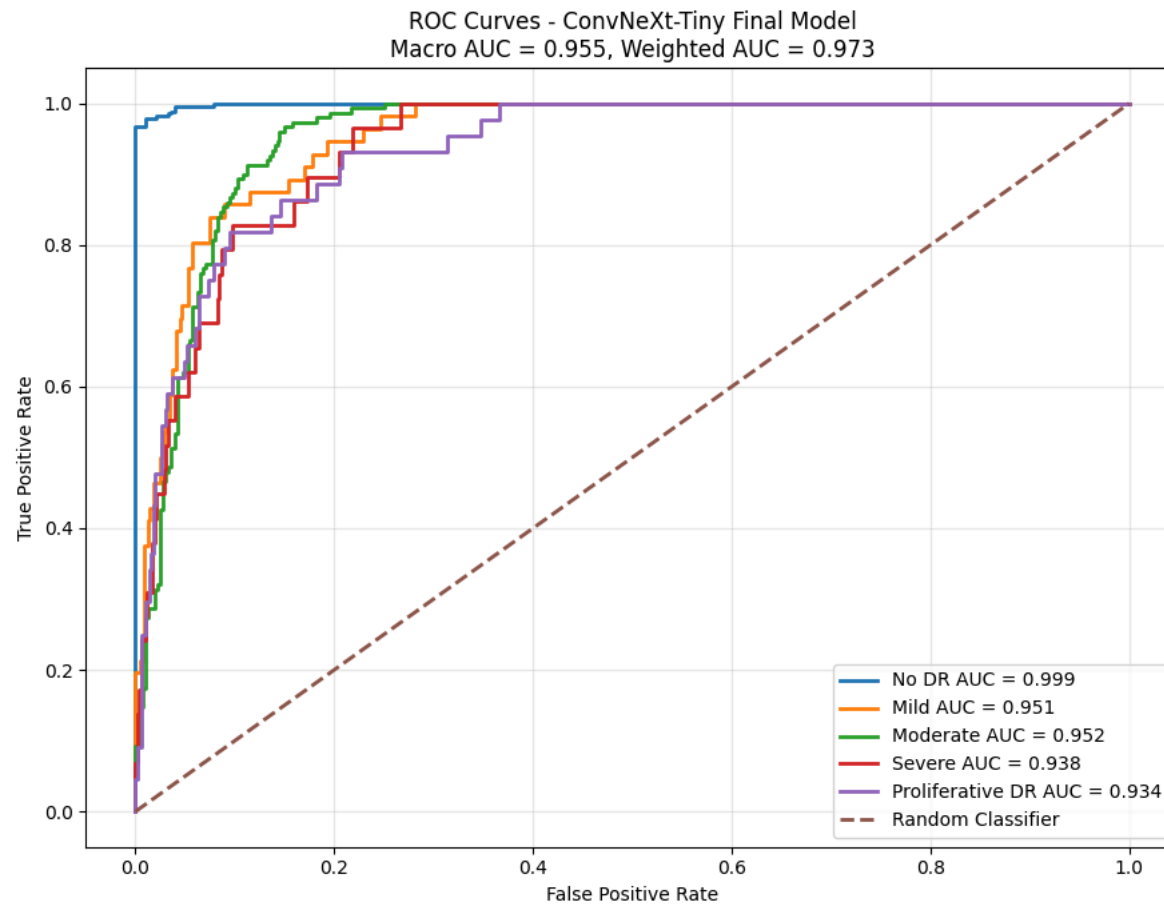
    weighted_auc = roc_auc_score(
        y_test_bin,
        y_prob_convnext,
        average="weighted",
        multi_class="ovr"
    )

    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title(
        f"ROC Curves - ConvNeXt-Tiny Final Model\n"
        f"Macro AUC = {macro_auc:.3f}, Weighted AUC = {weighted_auc:.3f}"
    )

    plt.legend(loc="lower right")
    plt.grid(alpha=0.3)
    plt.tight_layout()

    roc_path = os.path.join(SAVE_DIR, "convnext_roc_curves.png")
    plt.savefig(roc_path, dpi=300, bbox_inches="tight")
    plt.show()

    print("Saved at:", roc_path)
    print("Macro AUC:", macro_auc)
    print("Weighted AUC:", weighted_auc)
```



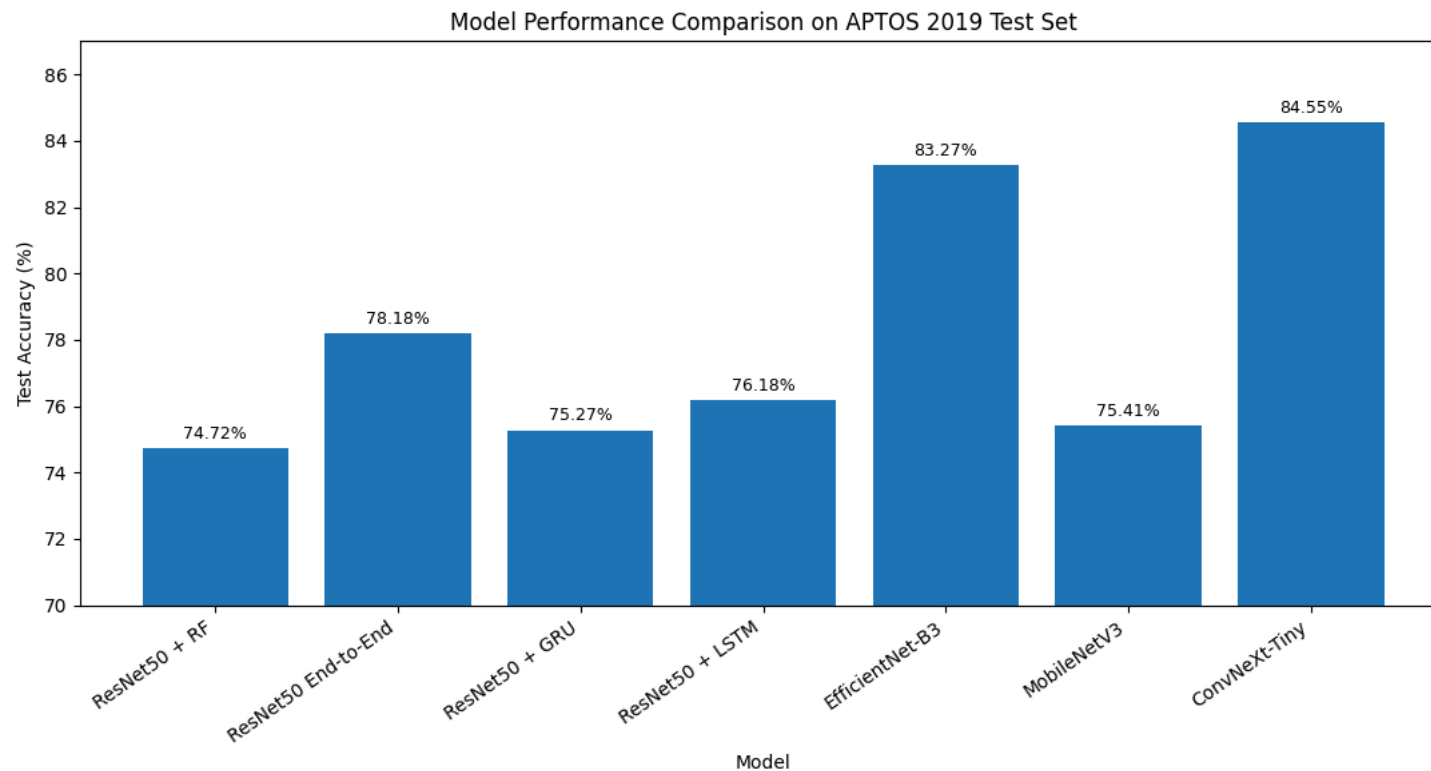
Saved at: /content/drive/MyDrive/DR_Results_ConvNeXt/convnext_roc_curves.png
 Macro AUC: 0.9548831577297339
 Weighted AUC: 0.972909505610836

```
import matplotlib.pyplot as plt
import os
```

```
models_names = [
    "ResNet50 + RF",
    "ResNet50 End-to-End",
    "ResNet50 + GRU",
    "ResNet50 + LSTM",
    "EfficientNet-B3",
    "MobileNetV3",
    "ConvNeXt-Tiny"
]
```

```
test accuracies = [
    74.72,
```

```
78.18,  
75.27,  
76.18,  
83.27,  
75.41,  
84.55  
]  
  
plt.figure(figsize=(11, 6))  
bars = plt.bar(models_names, test accuracies)  
  
plt.ylabel("Test Accuracy (%)")  
plt.xlabel("Model")  
plt.title("Model Performance Comparison on APTOS 2019 Test Set")  
plt.xticks(rotation=35, ha="right")  
plt.ylim(70, 87)  
  
for bar, acc in zip(bars, test accuracies):  
    plt.text(  
        bar.get_x() + bar.get_width() / 2,  
        bar.get_height() + 0.3,  
        f"{acc:.2f}%",  
        ha="center",  
        fontsize=9  
    )  
  
plt.tight_layout()  
  
comparison_path = os.path.join(SAVE_DIR, "model_accuracy_comparison.png")  
plt.savefig(comparison_path, dpi=300, bbox_inches="tight")  
plt.show()  
  
print("Saved at:", comparison_path)
```



Saved at: /content/drive/MyDrive/DR_Results_ConvNeXt/model_accuracy_comparison.png

```
import matplotlib.pyplot as plt
import os

SAVE_DIR = "/content/drive/MyDrive/DR_Results_ConvNeXt"
os.makedirs(SAVE_DIR, exist_ok=True)

models_names = [
    "EfficientNet-B3",
    "ConvNeXt-Tiny"
]

test_accuracies = [
    83.27,
    84.55
]

plt.figure(figsize=(7, 5))

bars = plt.bar(models_names, test_accuracies)

plt.ylabel("Test Accuracy (%)")
plt.title("Accuracy Improvement from EfficientNet-B3 to ConvNeXt-Tiny")
```

```
plt.ylim(80, 86)

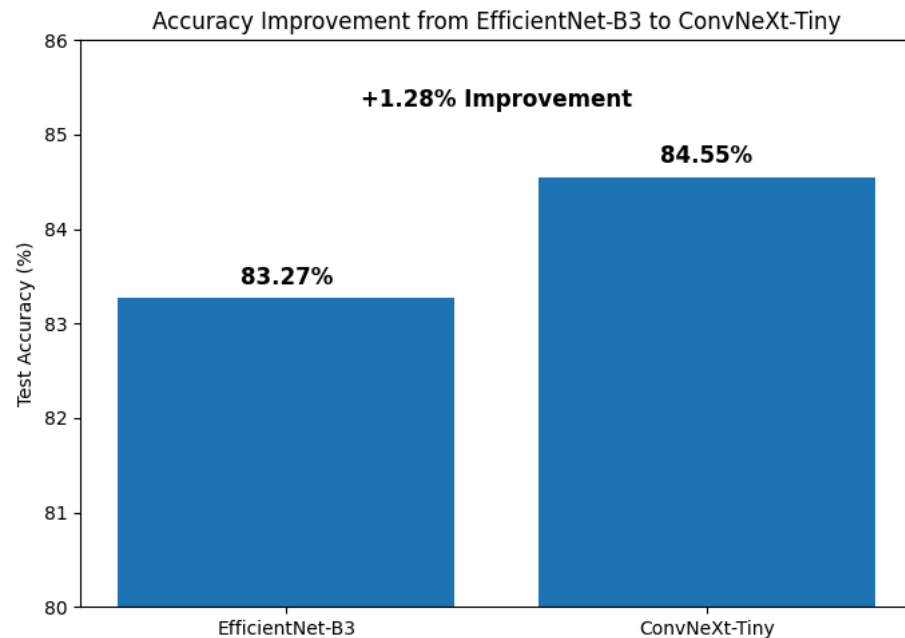
for bar, acc in zip(bars, test_accuracies):
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        bar.get_height() + 0.15,
        f"{acc:.2f}%",
        ha="center",
        fontsize=12,
        fontweight="bold"
    )

plt.text(
    0.5,
    85.3,
    "+1.28% Improvement",
    ha="center",
    fontsize=12,
    fontweight="bold"
)

plt.tight_layout()

improvement_chart_path = os.path.join(SAVE_DIR, "accuracy_improvement_convnext_vs_efficientnet.png")
plt.savefig(improvement_chart_path, dpi=300, bbox_inches="tight")
plt.show()

print("Saved at:", improvement_chart_path)
```



Saved at: /content/drive/MyDrive/DR_Results_ConvNeXt/accuracy_improvement_convnext_vs_efficientnet.png

```

import matplotlib.pyplot as plt
import numpy as np
import os

SAVE_DIR = "/content/drive/MyDrive/DR_Results_ConvNeXt"
os.makedirs(SAVE_DIR, exist_ok=True)

# ConvNeXt-Tiny training history from the training log
epochs = np.arange(1, 13)

train_loss = [
    1.1148, 0.7889, 0.6856, 0.5565,
    0.4933, 0.4094, 0.3160, 0.2531,
    0.2225, 0.1935, 0.1769, 0.1689
]

train_acc = [
    0.6348, 0.7741, 0.8057, 0.8416,
    0.8638, 0.8853, 0.9110, 0.9372,
    0.9442, 0.9559, 0.9586, 0.9618
]

val_loss = [
    0.9593, 0.8066, 0.9000, 0.7507,
    0.8154, 0.8034, 0.8146, 0.8694,
    0.8402, 0.9236, 0.9385, 0.9063
]

val_acc = [
    0.7432, 0.7523, 0.7905, 0.7905,
    0.8015, 0.7887, 0.8270, 0.8342,
    0.8452, 0.8452, 0.8543, 0.8397
]

```

```

plt.figure(figsize=(8, 5))

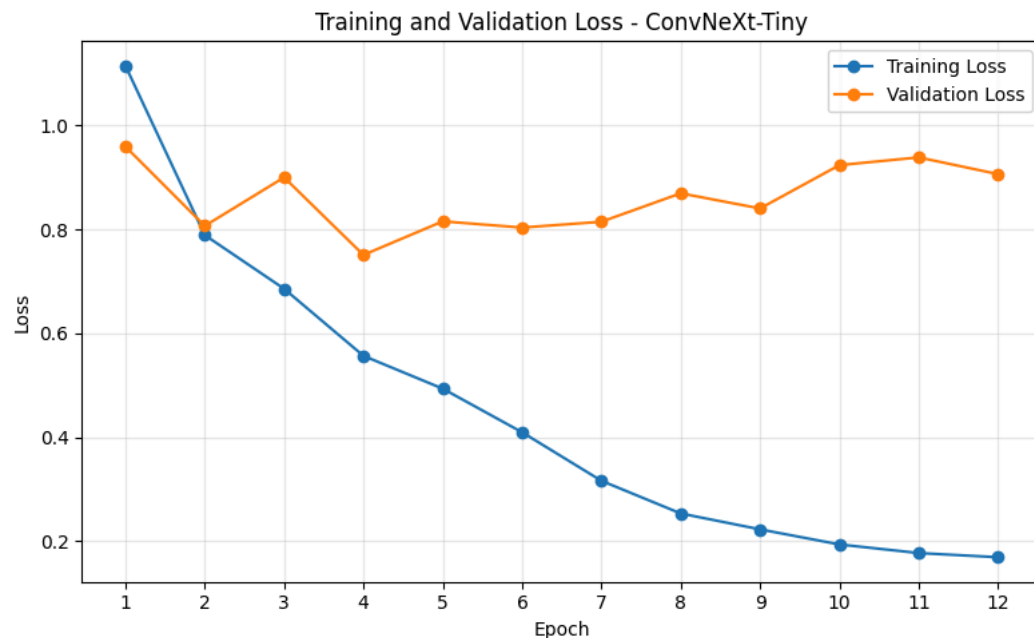
plt.plot(epochs, train_loss, marker="o", label="Training Loss")
plt.plot(epochs, val_loss, marker="o", label="Validation Loss")

plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Training and Validation Loss - ConvNeXt-Tiny")
plt.xticks(epochs)
plt.legend()
plt.grid(alpha=0.3)
plt.tight_layout()

loss_path = os.path.join(SAVE_DIR, "convnext_training_validation_loss.png")
plt.savefig(loss_path, dpi=300, bbox_inches="tight")
plt.show()

print("Saved at:", loss_path)

```



Saved at: /content/drive/MyDrive/DR_Results_ConvNeXt/convnext_training_validation_loss.png

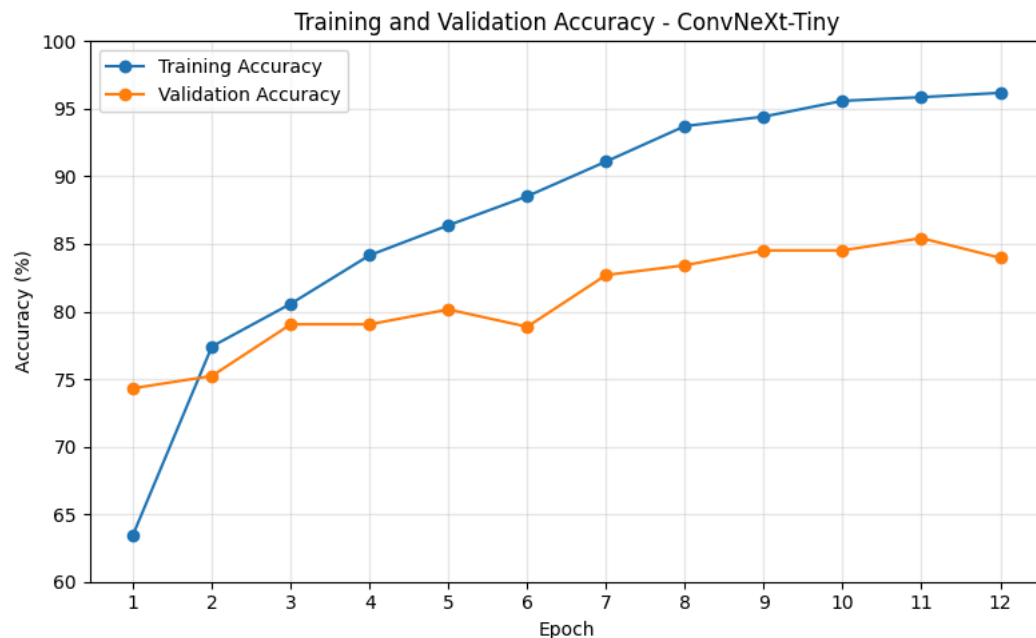
```
plt.figure(figsize=(8, 5))

plt.plot(epochs, np.array(train_acc) * 100, marker="o", label="Training Accuracy")
plt.plot(epochs, np.array(val_acc) * 100, marker="o", label="Validation Accuracy")

plt.xlabel("Epoch")
plt.ylabel("Accuracy (%)")
plt.title("Training and Validation Accuracy - ConvNeXt-Tiny")
plt.xticks(epochs)
plt.ylim(60, 100)
plt.legend()
plt.grid(alpha=0.3)
plt.tight_layout()

acc_path = os.path.join(SAVE_DIR, "convnext_training_validation_accuracy.png")
plt.savefig(acc_path, dpi=300, bbox_inches="tight")
plt.show()

print("Saved at:", acc_path)
```



Saved at: /content/drive/MyDrive/DR_Results_ConvNeXt/convnext_training_validation_accuracy.png

```
overfitting_gap = (np.array(train_acc) - np.array(val_acc)) * 100

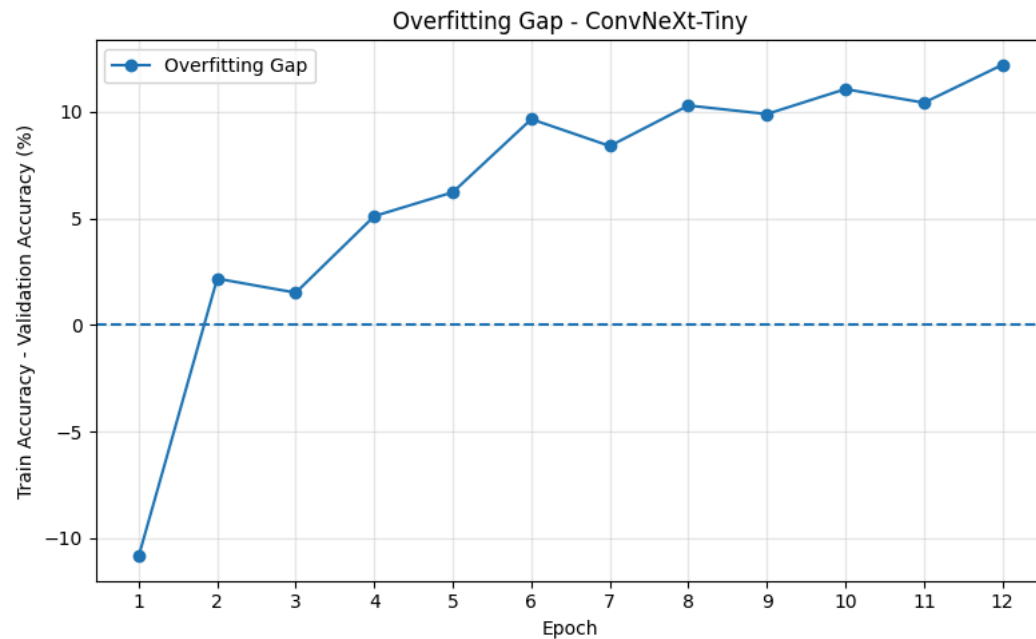
plt.figure(figsize=(8, 5))

plt.plot(epochs, overfitting_gap, marker="o", label="Overfitting Gap")

plt.axhline(y=0, linestyle="--")
plt.xlabel("Epoch")
plt.ylabel("Train Accuracy - Validation Accuracy (%)")
plt.title("Overfitting Gap - ConvNeXt-Tiny")
plt.xticks(epochs)
plt.legend()
plt.grid(alpha=0.3)
plt.tight_layout()

gap_path = os.path.join(SAVE_DIR, "convnext_overfitting_gap.png")
plt.savefig(gap_path, dpi=300, bbox_inches="tight")
plt.show()

print("Saved at:", gap_path)
```



Saved at: /content/drive/MyDrive/DR_Results_ConvNeXt/convnext_overfitting_gap.png

```
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

# Loss
axes[0].plot(epochs, train_loss, marker="o", label="Training Loss")
axes[0].plot(epochs, val_loss, marker="o", label="Validation Loss")
axes[0].set_title("Loss Curve")
axes[0].set_xlabel("Epoch")
axes[0].set_ylabel("Loss")
axes[0].set_xticks(epochs)
axes[0].legend()
axes[0].grid(alpha=0.3)

# Accuracy
axes[1].plot(epochs, np.array(train_acc) * 100, marker="o", label="Training Accuracy")
axes[1].plot(epochs, np.array(val_acc) * 100, marker="o", label="Validation Accuracy")
axes[1].set_title("Accuracy Curve")
axes[1].set_xlabel("Epoch")
axes[1].set_ylabel("Accuracy (%)")
axes[1].set_xticks(epochs)
axes[1].legend()
axes[1].grid(alpha=0.3)

# Overfitting Gap
axes[2].plot(epochs, overfitting_gap, marker="o", label="Overfitting Gap")
axes[2].axhline(y=0, linestyle="--")
axes[2].set_title("Overfitting Gap")
axes[2].set_xlabel("Epoch")
```

```

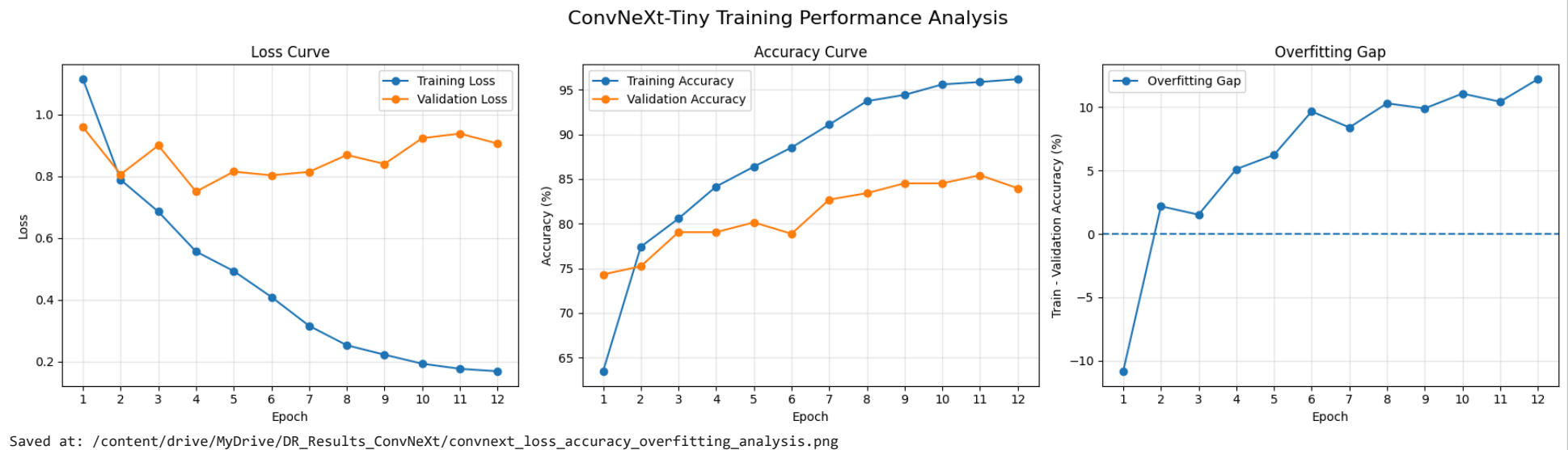
axes[2].set_ylabel("Train - Validation Accuracy (%)")
axes[2].set_xticks(epochs)
axes[2].legend()
axes[2].grid(alpha=0.3)

plt.suptitle("ConvNeXt-Tiny Training Performance Analysis", fontsize=16)
plt.tight_layout()

combined_path = os.path.join(SAVE_DIR, "convnext_loss_accuracy_overfitting_analysis.png")
plt.savefig(combined_path, dpi=300, bbox_inches="tight")
plt.show()

print("Saved at:", combined_path)

```



```

import matplotlib.pyplot as plt
import os

SAVE_DIR = "/content/drive/MyDrive/DR_Results_ConvNeXt"
os.makedirs(SAVE_DIR, exist_ok=True)

models = [
    "Reference Paper\nFramework 2 + Augmentation",
    "Proposed Model\nConvNeXt-Tiny"
]

accuracies = [
    83.78,
    84.55
]

```

```
]

plt.figure(figsize=(8, 5))

bars = plt.bar(models, accuracies)

plt.ylabel("Test Accuracy (%)")
plt.title("Comparison with Reference Paper")
plt.ylim(80, 86)

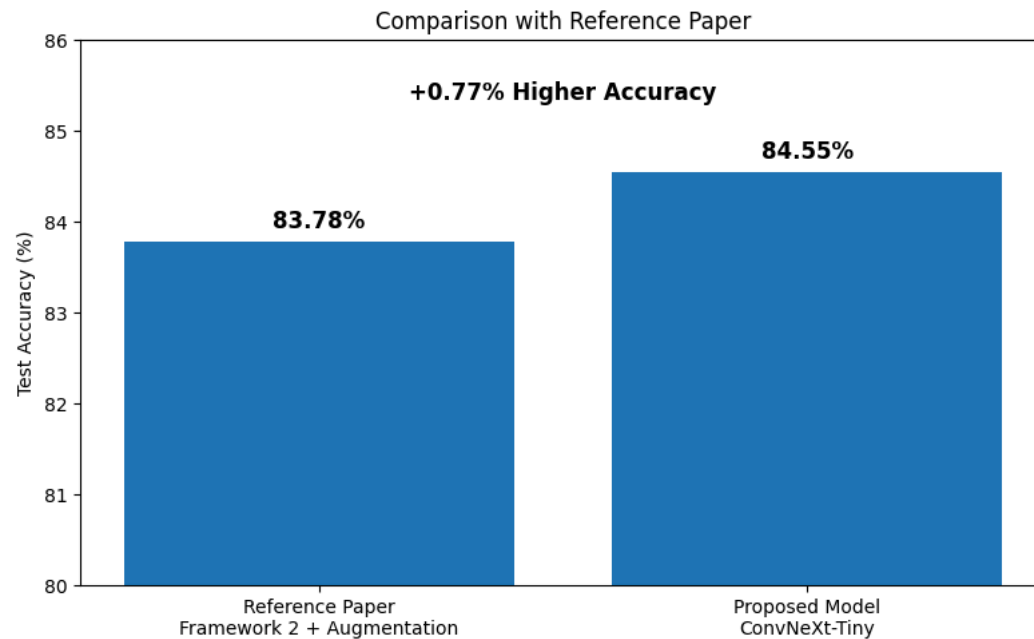
for bar, acc in zip(bars, accuracies):
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        bar.get_height() + 0.15,
        f"{acc:.2f}%",
        ha="center",
        fontsize=12,
        fontweight="bold"
    )

plt.text(
    0.5,
    85.35,
    "+0.77% Higher Accuracy",
    ha="center",
    fontsize=12,
    fontweight="bold"
)

plt.tight_layout()

paper_comparison_path = os.path.join(SAVE_DIR, "proposed_model_vs_reference_paper.png")
plt.savefig(paper_comparison_path, dpi=300, bbox_inches="tight")
plt.show()

print("Saved at:", paper_comparison_path)
```



Saved at: /content/drive/MyDrive/DR_Results_ConvNeXt/proposed_model_vs_reference_paper.png

```
import matplotlib.pyplot as plt
import os

SAVE_DIR = "/content/drive/MyDrive/DR_Results_ConvNeXt"
os.makedirs(SAVE_DIR, exist_ok=True)

models = [
    "EfficientNet-B3\nPrevious Best",
    "Reference Paper\nBest Result",
    "ConvNeXt-Tiny\nProposed Model"
]

accuracies = [
    83.27,
    83.78,
    84.55
]

plt.figure(figsize=(9, 5.5))

bars = plt.bar(models, accuracies)

plt.ylabel("Test Accuracy (%)")
plt.title("Performance Comparison of Top Models")
plt.ylim(80, 86)

for bar, acc in zip(bars, accuracies):
```

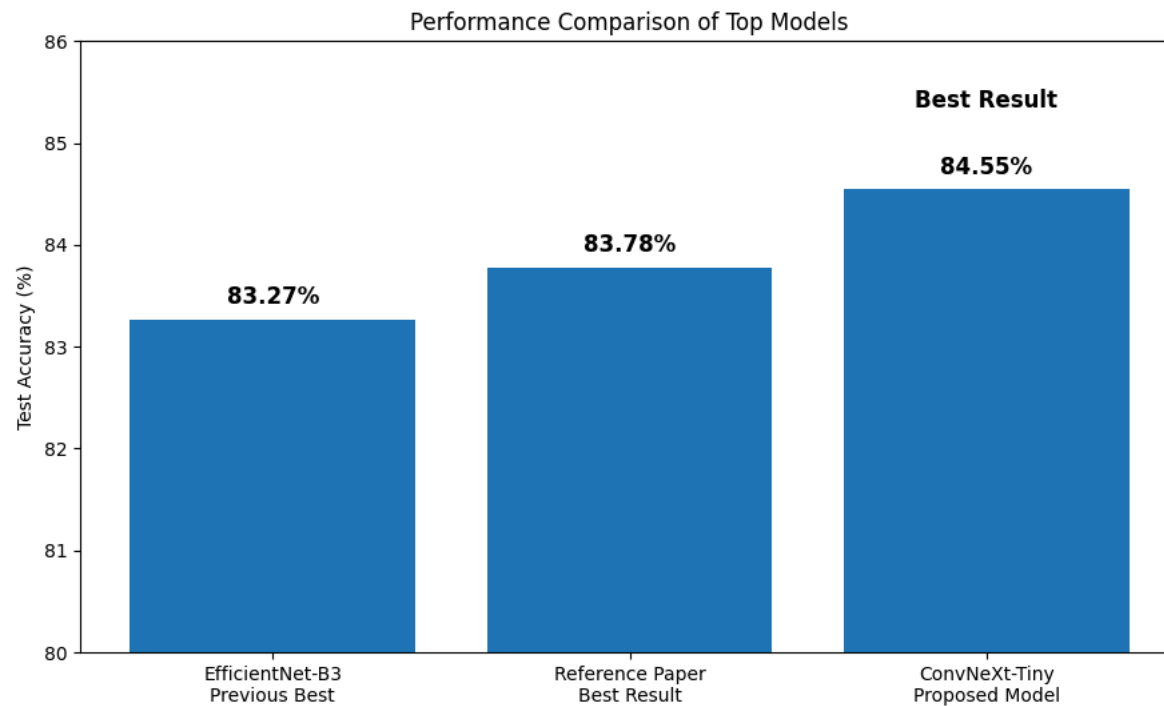
```
plt.text(
    bar.get_x() + bar.get_width() / 2,
    bar.get_height() + 0.15,
    f"{acc:.2f}%",
    ha="center",
    fontsize=12,
    fontweight="bold"
)

plt.text(
    2,
    85.35,
    "Best Result",
    ha="center",
    fontsize=12,
    fontweight="bold"
)

plt.tight_layout()

final_comparison_path = os.path.join(SAVE_DIR, "top_models_and_paper_comparison.png")
plt.savefig(final_comparison_path, dpi=300, bbox_inches="tight")
plt.show()

print("Saved at:", final_comparison_path)
```



Saved at: /content/drive/MyDrive/DR_Results_ConvNeXt/top_models_and_paper_comparison.png

```
import matplotlib.pyplot as plt
import numpy as np
import os

SAVE_DIR = "/content/drive/MyDrive/DR_Results_ConvNeXt"
os.makedirs(SAVE_DIR, exist_ok=True)

# ConvNeXt-Tiny training log
epochs = np.arange(1, 13)

train_loss = [
    1.1148, 0.7889, 0.6856, 0.5565,
    0.4933, 0.4094, 0.3160, 0.2531,
    0.2225, 0.1935, 0.1769, 0.1689
]

val_loss = [
    0.9593, 0.8066, 0.9000, 0.7507,
    0.8154, 0.8034, 0.8146, 0.8694,
    0.8402, 0.9236, 0.9385, 0.9063
]

# Create figure
fig, ax = plt.subplots(figsize=(10, 6))

ax.plot(
    epochs,
    train_loss,
    marker="o",
    linewidth=2,
    label="ConvNeXt-Tiny Training Loss"
)

ax.plot(
    epochs,
    val_loss,
    marker="o",
    linewidth=2,
    label="ConvNeXt-Tiny Validation Loss"
)

ax.set_title(
    "ConvNeXt-Tiny, Training Loss vs. Validation Loss per Epoch",
    fontsize=14,
    fontweight="bold"
)

ax.set_xlabel("Epoch", fontsize=12)
ax.set_ylabel("Loss", fontsize=12)

ax.set_xticks(epochs)
ax.grid(True, alpha=0.4)
ax.legend(loc="upper right")

for spine in ax.spines.values():
```

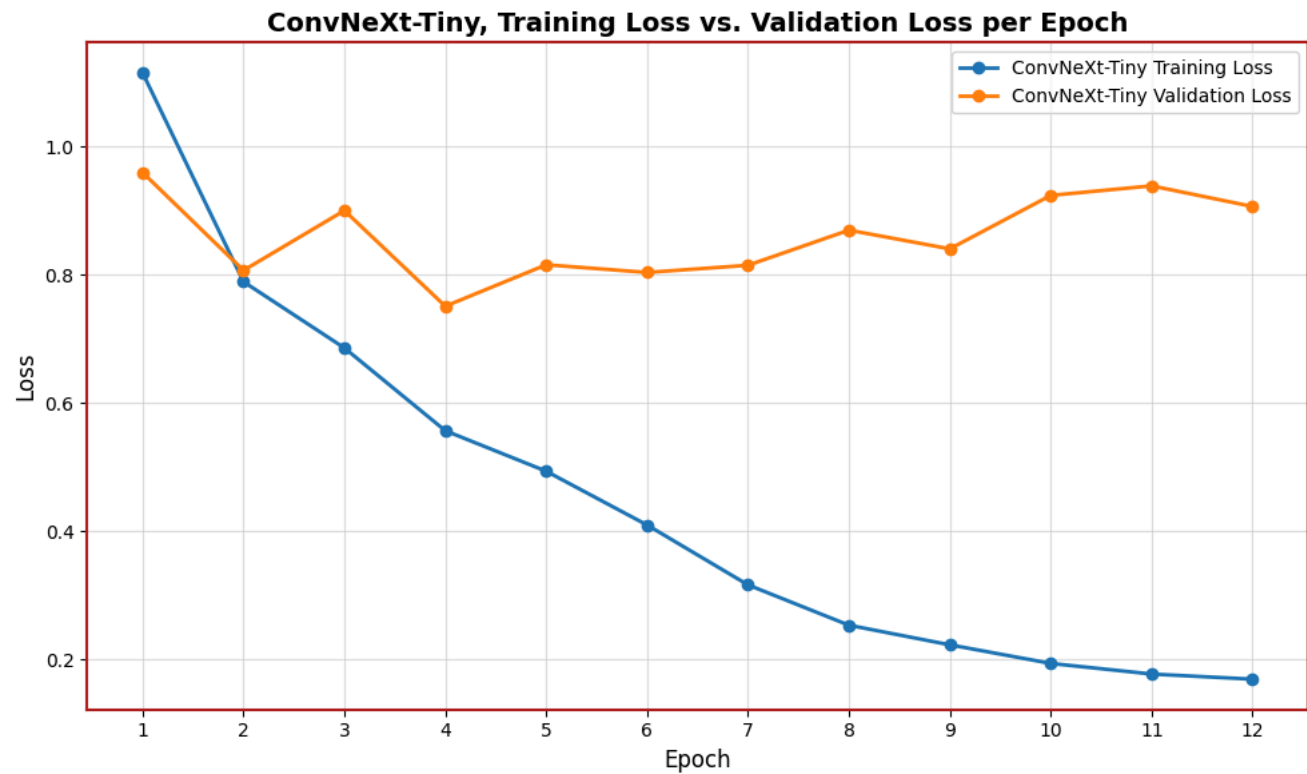
```
spine.set_edgecolor("firebrick")
spine.set_linewidth(1.5)

plt.tight_layout()

loss_comparison_path = os.path.join(
    SAVE_DIR,
    "convnext_training_vs_validation_loss_per_epoch.png"
)

plt.savefig(loss_comparison_path, dpi=300, bbox_inches="tight")
plt.show()

print("Saved at:", loss_comparison_path)
```



Saved at: /content/drive/MyDrive/DR_Results_ConvNeXt/convnext_training_vs_validation_loss_per_epoch.png

```
import matplotlib.pyplot as plt
import numpy as np
import os

SAVE_DIR = "/content/drive/MyDrive/DR_Results_ConvNeXt"
os.makedirs(SAVE_DIR, exist_ok=True)

# Epochs
```

```

epochs = np.arange(1, 13)

# EfficientNet-B3 V2 Validation Accuracy
effnet_val_acc = [
    0.6976, 0.7577, 0.7869, 0.7814,
    0.7978, 0.8015, 0.8106, 0.8197,
    0.8106, 0.8087, 0.8142, 0.8142
]

# ConvNeXt-Tiny Validation Accuracy
convnext_val_acc = [
    0.7432, 0.7523, 0.7905, 0.7905,
    0.8015, 0.7887, 0.8270, 0.8342,
    0.8452, 0.8452, 0.8543, 0.8397
]

plt.figure(figsize=(10, 6))

plt.plot(
    epochs,
    np.array(effnet_val_acc) * 100,
    marker="o",
    linewidth=2,
    label="EfficientNet-B3 Validation Accuracy"
)

plt.plot(
    epochs,
    np.array(convnext_val_acc) * 100,
    marker="o",
    linewidth=2,
    label="ConvNeXt-Tiny Validation Accuracy"
)

# Mark best points
best_eff_epoch = epochs[np.argmax(effnet_val_acc)]
best_eff_acc = max(effnet_val_acc) * 100

best_conv_epoch = epochs[np.argmax(convnext_val_acc)]
best_conv_acc = max(convnext_val_acc) * 100

plt.scatter(best_eff_epoch, best_eff_acc, s=120)
plt.text(
    best_eff_epoch,
    best_eff_acc + 0.7,
    f"Best EfficientNet-B3\n{best_eff_acc:.2f}%",
    ha="center",
    fontsize=10,
    fontweight="bold"
)

plt.scatter(best_conv_epoch, best_conv_acc, s=120)
plt.text(
    best_conv_epoch,
    best_conv_acc + 0.7,
    f"Best ConvNeXt-Tiny\n{best_conv_acc:.2f}%",

```

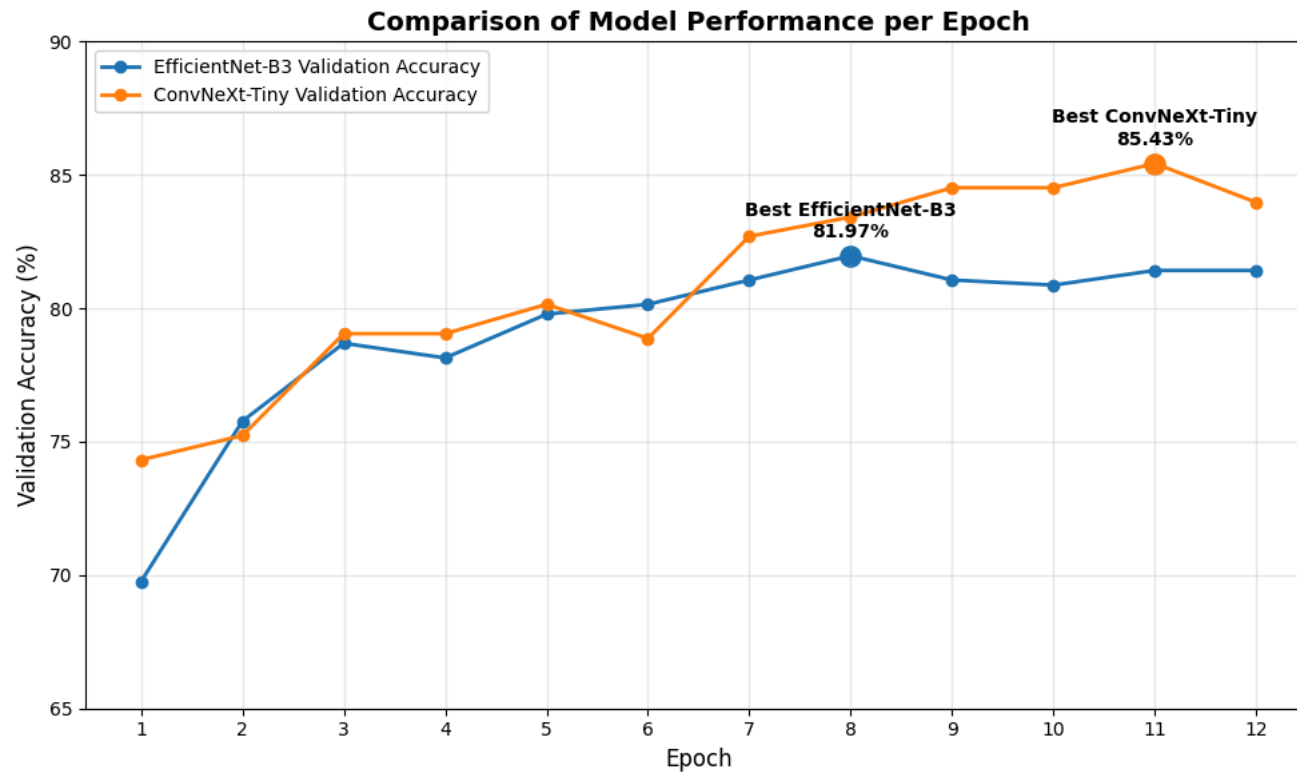
```
ha="center",
fontsize=10,
fontweight="bold"
)

plt.xlabel("Epoch", fontsize=12)
plt.ylabel("Validation Accuracy (%)", fontsize=12)
plt.title("Comparison of Model Performance per Epoch", fontsize=14, fontweight="bold")
plt.xticks(epochs)
plt.ylim(65, 90)
plt.grid(alpha=0.3)
plt.legend()
plt.tight_layout()

comparison_path = os.path.join(
    SAVE_DIR,
    "comparison_model_performance_per_epoch_top2.png"
)

plt.savefig(comparison_path, dpi=300, bbox_inches="tight")
plt.show()

print("Saved at:", comparison_path)
```



Saved at: /content/drive/MyDrive/DR_Results_ConvNeXt/comparison_model_performance_per_epoch_top2.png

```

import matplotlib.pyplot as plt
import numpy as np
import os

SAVE_DIR = "/content/drive/MyDrive/DR_Results_ConvNeXt"
os.makedirs(SAVE_DIR, exist_ok=True)

epochs = np.arange(1, 13)

# EfficientNet-B3 V2 Loss
effnet_train_loss = [
    1.2659, 0.8784, 0.7365, 0.6681,
    0.5651, 0.4916, 0.4004, 0.3670,
    0.3081, 0.3102, 0.2648, 0.2874
]

effnet_val_loss = [
    0.9851, 0.8657, 0.8394, 0.8377,
    0.8460, 0.8870, 0.9061, 0.9213,
    0.9570, 0.9363, 0.9742, 0.9926
]

# ConvNeXt-Tiny Loss
convnext_train_loss = [
    1.1148, 0.7889, 0.6856, 0.5565,
    0.4933, 0.4094, 0.3160, 0.2531,
    0.2225, 0.1935, 0.1769, 0.1689
]

convnext_val_loss = [
    0.9593, 0.8066, 0.9000, 0.7507,
    0.8154, 0.8034, 0.8146, 0.8694,
    0.8402, 0.9236, 0.9385, 0.9063
]

plt.figure(figsize=(11, 6))

plt.plot(
    epochs,
    effnet_train_loss,
    marker="o",
    linewidth=2,
    label="EfficientNet-B3 Training Loss"
)

plt.plot(
    epochs,
    effnet_val_loss,
    marker="o",
    linewidth=2,
    label="EfficientNet-B3 Validation Loss"
)

plt.plot(
    epochs,

```

```
convnext_train_loss,
marker="o",
linewidth=2,
label="ConvNeXt-Tiny Training Loss"
)

plt.plot(
    epochs,
    convnext_val_loss,
    marker="o",
    linewidth=2,
    label="ConvNeXt-Tiny Validation Loss"
)

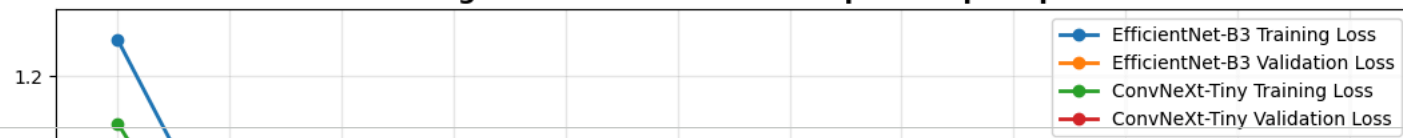
plt.xlabel("Epoch", fontsize=12)
plt.ylabel("Loss", fontsize=12)
plt.title("Training and Validation Loss Comparison per Epoch", fontsize=14, fontweight="bold")
plt.xticks(epochs)
plt.grid(alpha=0.3)
plt.legend()
plt.tight_layout()

loss_comparison_path = os.path.join(
    SAVE_DIR,
    "top_two_models_loss_comparison_per_epoch.png"
)

plt.savefig(loss_comparison_path, dpi=300, bbox_inches="tight")
plt.show()

print("Saved at:", loss_comparison_path)
```

Training and Validation Loss Comparison per Epoch



```
import matplotlib.pyplot as plt
import numpy as np
import os

SAVE_DIR = "/content/drive/MyDrive/DR_Results_ConvNeXt"
os.makedirs(SAVE_DIR, exist_ok=True)

epochs = np.arange(1, 13)

# Accuracy
effnet_val_acc = [
    0.6976, 0.7577, 0.7869, 0.7814,
    0.7978, 0.8015, 0.8106, 0.8197,
    0.8106, 0.8087, 0.8142, 0.8142
]

convnext_val_acc = [
    0.7432, 0.7523, 0.7905, 0.7905,
    0.8015, 0.7887, 0.8270, 0.8342,
    0.8452, 0.8452, 0.8543, 0.8397
]

# Loss
effnet_train_loss = [
    1.2659, 0.8784, 0.7365, 0.6681,
    0.5651, 0.4916, 0.4004, 0.3670,
    0.3081, 0.3102, 0.2648, 0.2874
]

effnet_val_loss = [
    0.9851, 0.8657, 0.8394, 0.8377,
    0.8460, 0.8870, 0.9061, 0.9213,
    0.9570, 0.9363, 0.9742, 0.9926
]

convnext_train_loss = [
    1.1148, 0.7889, 0.6856, 0.5565,
    0.4933, 0.4094, 0.3160, 0.2531,
    0.2225, 0.1935, 0.1769, 0.1689
]

convnext_val_loss = [
    0.9593, 0.8066, 0.9000, 0.7507,
```