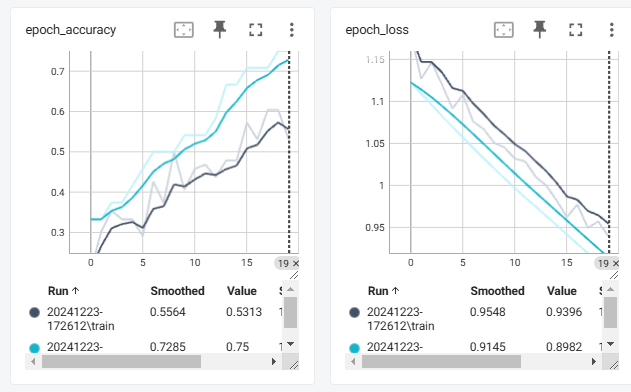
 5-1 keras

# 匯入必要的套件

import os

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from tensorflow.keras.callbacks import TensorBoard

from datetime import datetime  # 用於生成唯一的目錄名稱

# 生成唯一的日誌目錄名稱

current\_time = datetime.now().strftime("%Y%m%d-%H%M%S")

log\_dir = f"tb\_logs/{current\_time}"

os.makedirs(log\_dir, exist\_ok=True)

print(f"已建立日誌目錄: {log\_dir}")

# 加載 Iris 資料集

data = load\_iris()

X, y = data.data, data.target

# 資料標準化

scaler = StandardScaler()

X = scaler.fit\_transform(X)

# 分割訓練集與測試集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# 將目標轉換為 one-hot 編碼

y\_train = tf.keras.utils.to\_categorical(y\_train, num\_classes=3)

y\_test = tf.keras.utils.to\_categorical(y\_test, num\_classes=3)

# 建立模型

model = Sequential([

    Dense(16, activation='relu', input\_shape=(4,)),

    Dropout(0.5),

    Dense(8, activation='relu'),

    Dense(3, activation='softmax')  # 輸出層

])

# 編譯模型

model.compile(

    optimizer=tf.keras.optimizers.Adam(learning\_rate=0.001),

    loss='categorical\_crossentropy',

    metrics=['accuracy']

)

# 設置 TensorBoard 回調

tensorboard\_callback = TensorBoard(log\_dir=log\_dir, histogram\_freq=1)

# 訓練模型

history = model.fit(

    X\_train, y\_train,

    validation\_split=0.2,

    epochs=20,

    batch\_size=32,

    callbacks=[tensorboard\_callback]

)

# 測試模型

test\_loss, test\_acc = model.evaluate(X\_test, y\_test, verbose=2)

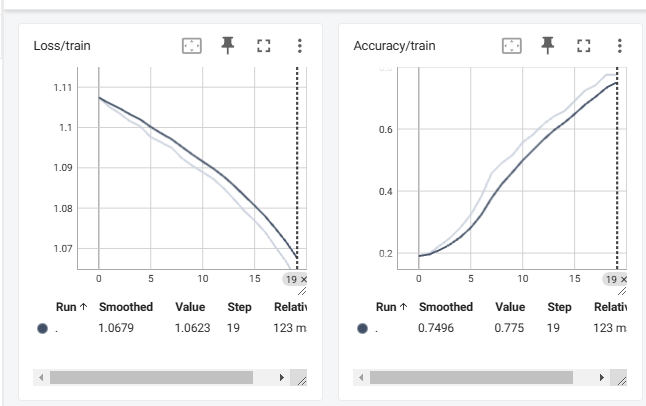
print(f"測試損失: {test\_loss:.4f}, 測試準確率: {test\_acc:.4f}")

# 提示啟動 TensorBoard

print(f"啟動 TensorBoard，執行以下命令：")

print(f"tensorboard --logdir tb\_logs")

5-1 torch

# 匯入必要的套件

import os

import shutil

import torch

import torch.nn as nn

import torch.optim as optim

from torch.utils.data import DataLoader, TensorDataset

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from torch.utils.tensorboard import SummaryWriter

from datetime import datetime

# 動態生成日誌名稱

current\_time = datetime.now().strftime("%Y%m%d-%H%M%S")

log\_dir = f"tb\_logs\_pytorch/{current\_time}"

if os.path.exists(log\_dir):

    shutil.rmtree(log\_dir)  # 刪除舊的日誌目錄

os.makedirs(log\_dir, exist\_ok=True)  # 重建日誌目錄

print(f"已建立日誌目錄: {log\_dir}")

# 加載 Iris 資料集

data = load\_iris()

X, y = data.data, data.target

# 資料標準化

scaler = StandardScaler()

X = scaler.fit\_transform(X)

# 分割訓練集與測試集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# 轉換為 TensorDataset

train\_dataset = TensorDataset(

    torch.tensor(X\_train, dtype=torch.float32),

    torch.tensor(y\_train, dtype=torch.long)

)

test\_dataset = TensorDataset(

    torch.tensor(X\_test, dtype=torch.float32),

    torch.tensor(y\_test, dtype=torch.long)

)

# 建立 DataLoader

train\_loader = DataLoader(train\_dataset, batch\_size=32, shuffle=True)

test\_loader = DataLoader(test\_dataset, batch\_size=32, shuffle=False)

# 定義模型

class IrisClassifier(nn.Module):

    def \_\_init\_\_(self):

        super(IrisClassifier, self).\_\_init\_\_()

        self.fc1 = nn.Linear(4, 16)

        self.fc2 = nn.Linear(16, 8)

        self.fc3 = nn.Linear(8, 3)

        self.relu = nn.ReLU()

        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):

        x = self.relu(self.fc1(x))

        x = self.relu(self.fc2(x))

        x = self.softmax(self.fc3(x))

        return x

# 初始化模型

model = IrisClassifier()

# 定義損失函數和優化器

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=0.001)

# 初始化 TensorBoard 記錄器

writer = SummaryWriter(log\_dir)

# 訓練模型

num\_epochs = 20

for epoch in range(num\_epochs):

    model.train()

    total\_loss = 0

    correct = 0

    total = 0

    for X\_batch, y\_batch in train\_loader:

        # 前向傳播

        outputs = model(X\_batch)

        loss = criterion(outputs, y\_batch)

        # 反向傳播與優化

        optimizer.zero\_grad()

        loss.backward()

        optimizer.step()

        # 計算損失和準確率

        total\_loss += loss.item()

        \_, predicted = torch.max(outputs, 1)

        correct += (predicted == y\_batch).sum().item()

        total += y\_batch.size(0)

    train\_loss = total\_loss / len(train\_loader)

    train\_acc = correct / total

    print(f"Epoch [{epoch + 1}/{num\_epochs}], Loss: {train\_loss:.4f}, Accuracy: {train\_acc:.4f}")

    # 記錄到 TensorBoard

    writer.add\_scalar('Loss/train', train\_loss, epoch)

    writer.add\_scalar('Accuracy/train', train\_acc, epoch)

# 測試模型

model.eval()

correct = 0

total = 0

with torch.no\_grad():

    for X\_batch, y\_batch in test\_loader:

        outputs = model(X\_batch)

        \_, predicted = torch.max(outputs, 1)

        correct += (predicted == y\_batch).sum().item()

        total += y\_batch.size(0)

test\_acc = correct / total

print(f"測試準確率: {test\_acc:.4f}")

writer.add\_scalar('Accuracy/test', test\_acc)

# 關閉 TensorBoard 記錄器

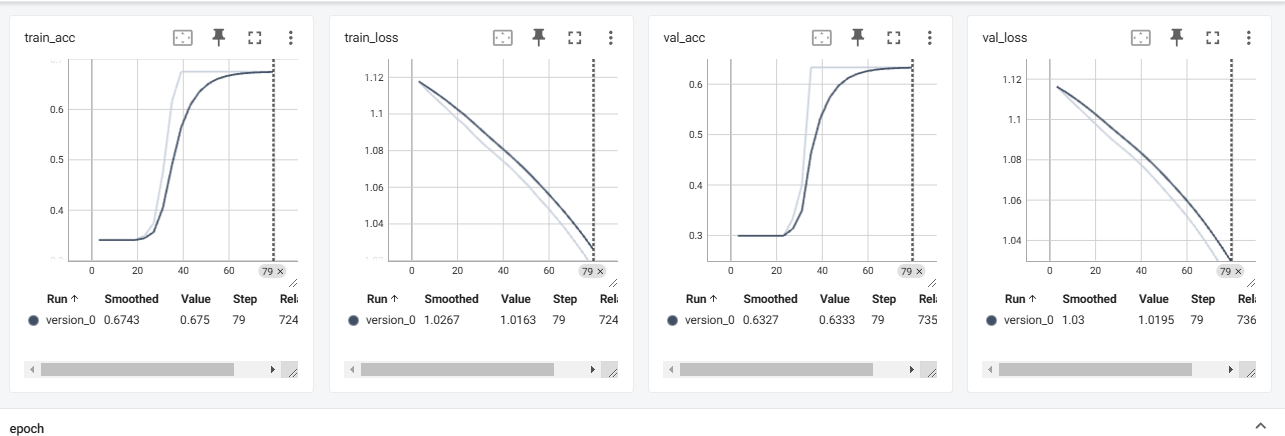
writer.close()

# 提示啟動 TensorBoard

print(f"啟動 TensorBoard，執行以下命令：")

print(f"tensorboard --logdir {log\_dir}")

5-1 torch lightning

# 匯入必要的套件

import os

import shutil

import torch

from torch.utils.data import DataLoader, TensorDataset

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from pytorch\_lightning import LightningModule, Trainer

from pytorch\_lightning.callbacks import ModelCheckpoint, EarlyStopping

from pytorch\_lightning.loggers import TensorBoardLogger

from datetime import datetime

# 動態生成日誌名稱

current\_time = datetime.now().strftime("%Y%m%d-%H%M%S")

log\_dir = f"tb\_logs\_pytorch\_lightning/{current\_time}"

if os.path.exists(log\_dir):

    shutil.rmtree(log\_dir)  # 刪除舊的日誌目錄

os.makedirs(log\_dir, exist\_ok=True)  # 重建日誌目錄

print(f"已建立日誌目錄: {log\_dir}")

# 加載 Iris 資料集

data = load\_iris()

X, y = data.data, data.target

# 資料標準化

scaler = StandardScaler()

X = scaler.fit\_transform(X)

# 分割訓練集與測試集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# 轉換為 TensorDataset

train\_dataset = TensorDataset(

    torch.tensor(X\_train, dtype=torch.float32),

    torch.tensor(y\_train, dtype=torch.long)

)

test\_dataset = TensorDataset(

    torch.tensor(X\_test, dtype=torch.float32),

    torch.tensor(y\_test, dtype=torch.long)

)

# 建立 DataLoader

train\_loader = DataLoader(train\_dataset, batch\_size=32, shuffle=True)

test\_loader = DataLoader(test\_dataset, batch\_size=32, shuffle=False)

# 定義模型

class IrisClassifier(LightningModule):

    def \_\_init\_\_(self):

        super(IrisClassifier, self).\_\_init\_\_()

        self.fc1 = torch.nn.Linear(4, 16)

        self.fc2 = torch.nn.Linear(16, 8)

        self.fc3 = torch.nn.Linear(8, 3)

        self.relu = torch.nn.ReLU()

        self.softmax = torch.nn.Softmax(dim=1)

        self.criterion = torch.nn.CrossEntropyLoss()

    def forward(self, x):

        x = self.relu(self.fc1(x))

        x = self.relu(self.fc2(x))

        x = self.softmax(self.fc3(x))

        return x

    def training\_step(self, batch, batch\_idx):

        X\_batch, y\_batch = batch

        outputs = self(X\_batch)

        loss = self.criterion(outputs, y\_batch)

        acc = (outputs.argmax(dim=1) == y\_batch).float().mean()

        self.log("train\_loss", loss, on\_step=False, on\_epoch=True, prog\_bar=True)

        self.log("train\_acc", acc, on\_step=False, on\_epoch=True, prog\_bar=True)

        return loss

    def validation\_step(self, batch, batch\_idx):

        X\_batch, y\_batch = batch

        outputs = self(X\_batch)

        loss = self.criterion(outputs, y\_batch)

        acc = (outputs.argmax(dim=1) == y\_batch).float().mean()

        self.log("val\_loss", loss, on\_step=False, on\_epoch=True, prog\_bar=True)

        self.log("val\_acc", acc, on\_step=False, on\_epoch=True, prog\_bar=True)

        return loss

    def test\_step(self, batch, batch\_idx):

        X\_batch, y\_batch = batch

        outputs = self(X\_batch)

        acc = (outputs.argmax(dim=1) == y\_batch).float().mean()

        self.log("test\_acc", acc, on\_step=False, on\_epoch=True, prog\_bar=True)

        return acc

    def configure\_optimizers(self):

        return torch.optim.Adam(self.parameters(), lr=0.001)

# 初始化模型

model = IrisClassifier()

# 初始化 TensorBoard 記錄器

logger = TensorBoardLogger("tb\_logs\_pytorch\_lightning", name=current\_time)

# 設置回調函數

checkpoint\_callback = ModelCheckpoint(

    monitor="val\_loss",

    dirpath=log\_dir,

    filename="best-checkpoint",

    save\_top\_k=1,

    mode="min"

)

early\_stopping\_callback = EarlyStopping(monitor="val\_loss", patience=5, mode="min")

# 訓練模型

trainer = Trainer(

    max\_epochs=20,

    logger=logger,

    callbacks=[checkpoint\_callback, early\_stopping\_callback]

)

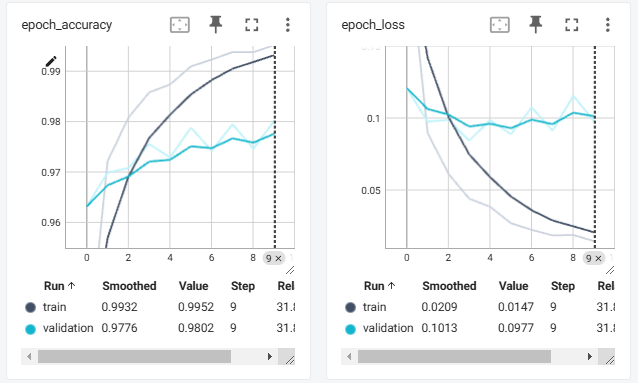
trainer.fit(model, train\_dataloaders=train\_loader, val\_dataloaders=test\_loader)

# 測試模型

trainer.test(model, dataloaders=test\_loader)

print(f"測試完成！日誌儲存在: {log\_dir}")

5-2 Dense NN

import os

from datetime import datetime

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten

from tensorflow.keras.callbacks import TensorBoard

from tensorflow.keras.datasets import mnist

# 加載 MNIST 資料集

(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()

# 資料標準化

X\_train = X\_train / 255.0  # 將像素值縮放到 [0, 1] 範圍

X\_test = X\_test / 255.0

# 動態生成 TensorBoard 日誌目錄

log\_dir = os.path.join("tb\_logs\_dense\_nn", datetime.now().strftime("%Y%m%d-%H%M%S"))

if not os.path.exists(log\_dir):

    os.makedirs(log\_dir)

print(f"TensorBoard 日誌目錄: {log\_dir}")

# 定義 Dense NN 模型

model = Sequential([

    Flatten(input\_shape=(28, 28)),  # 展平成 784 維向量

    Dense(256, activation='relu'),  # 第一個全連接層

    Dense(128, activation='relu'),  # 第二個全連接層

    Dense(10, activation='softmax')  # 輸出層 (10 個類別)

])

# 編譯模型

model.compile(

    optimizer='adam',

    loss='sparse\_categorical\_crossentropy',

    metrics=['accuracy']

)

# 初始化 TensorBoard 回調

tensorboard\_callback = TensorBoard(log\_dir=log\_dir, histogram\_freq=1)

# 訓練模型

model.fit(

    X\_train, y\_train,

    epochs=10,  # 訓練 10 個 Epoch

    batch\_size=32,  # 每次使用 32 筆資料進行訓練

    validation\_split=0.2,  # 使用 20% 的資料作為驗證集

    callbacks=[tensorboard\_callback]  # 加入 TensorBoard 回調

)

# 評估模型

test\_loss, test\_acc = model.evaluate(X\_test, y\_test, verbose=2)

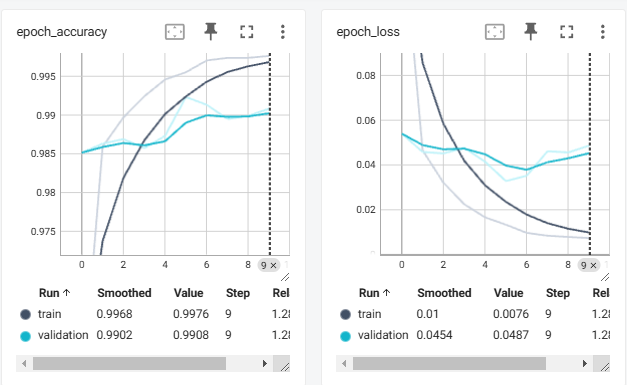
print(f"測試損失: {test\_loss:.4f}, 測試準確率: {test\_acc:.4f}")

# 提示啟動 TensorBoard

print(f"啟動 TensorBoard，執行以下命令：")

print(f"tensorboard --logdir {log\_dir}")

5-2 CNN

import os

from datetime import datetime

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

from tensorflow.keras.callbacks import TensorBoard

from tensorflow.keras.datasets import mnist

# 加載 MNIST 資料集

(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()

# 調整資料形狀並標準化

X\_train = X\_train.reshape(-1, 28, 28, 1) / 255.0  # 新增通道維度 (28, 28, 1)

X\_test = X\_test.reshape(-1, 28, 28, 1) / 255.0

# 動態生成 TensorBoard 日誌目錄

log\_dir = os.path.join("tb\_logs\_cnn", datetime.now().strftime("%Y%m%d-%H%M%S"))

if not os.path.exists(log\_dir):

    os.makedirs(log\_dir)

print(f"TensorBoard 日誌目錄: {log\_dir}")

# 定義 CNN 模型

model = Sequential([

    Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),  # 第一個卷積層

    MaxPooling2D((2, 2)),  # 第一個池化層

    Conv2D(64, (3, 3), activation='relu'),  # 第二個卷積層

    MaxPooling2D((2, 2)),  # 第二個池化層

    Flatten(),  # 展平層

    Dense(128, activation='relu'),  # 全連接層

    Dense(10, activation='softmax')  # 輸出層 (10 個類別)

])

# 編譯模型

model.compile(

    optimizer='adam',

    loss='sparse\_categorical\_crossentropy',

    metrics=['accuracy']

)

# 初始化 TensorBoard 回調

tensorboard\_callback = TensorBoard(log\_dir=log\_dir, histogram\_freq=1)

# 訓練模型

model.fit(

    X\_train, y\_train,

    epochs=10,  # 訓練 10 個 Epoch

    batch\_size=32,  # 每次使用 32 筆資料進行訓練

    validation\_split=0.2,  # 使用 20% 的資料作為驗證集

    callbacks=[tensorboard\_callback]  # 加入 TensorBoard 回調

)

# 評估模型

test\_loss, test\_acc = model.evaluate(X\_test, y\_test, verbose=2)

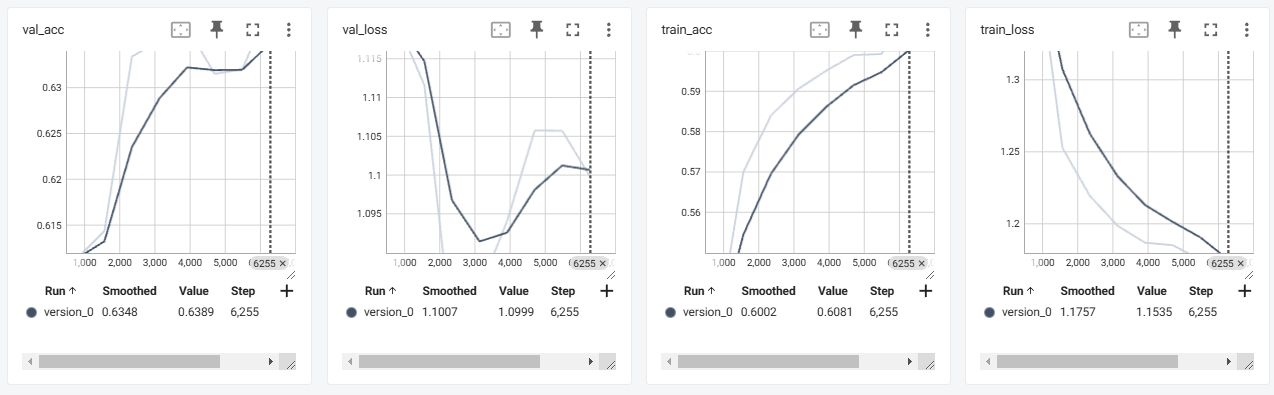
print(f"測試損失: {test\_loss:.4f}, 測試準確率: {test\_acc:.4f}")

# 提示啟動 TensorBoard

print(f"啟動 TensorBoard，執行以下命令：")

print(f"tensorboard --logdir {log\_dir}")

5-3 vgg16

import os

from datetime import datetime

import torch

import torch.nn as nn

import torchvision.transforms as transforms

from torchvision.datasets import CIFAR10

from torchvision.models import vgg16, vgg19

from torch.utils.data import DataLoader

import pytorch\_lightning as pl

from pytorch\_lightning.loggers import TensorBoardLogger

from pytorch\_lightning.callbacks import ModelCheckpoint, EarlyStopping

def main():

    # 動態生成 TensorBoard 日誌目錄

    current\_time = datetime.now().strftime("%Y%m%d-%H%M%S")

    log\_dir = f"tb\_logs\_cifar/{current\_time}"

    os.makedirs(log\_dir, exist\_ok=True)

    print(f"TensorBoard 日誌目錄: {log\_dir}")

    # 定義模型

    class VGGClassifier(pl.LightningModule):

        def \_\_init\_\_(self, use\_vgg16=True, num\_classes=10, lr=0.001):

            super(VGGClassifier, self).\_\_init\_\_()

            self.save\_hyperparameters()  # 保存超參數

            self.lr = lr

            # 選擇 VGG16 或 VGG19 預訓練模型

            self.feature\_extractor = vgg16(pretrained=True) if use\_vgg16 else vgg19(pretrained=True)

            for param in self.feature\_extractor.parameters():

                param.requires\_grad = False  # 凍結預訓練權重

            # 修改分類器部分

            in\_features = self.feature\_extractor.classifier[0].in\_features

            self.feature\_extractor.classifier = nn.Sequential(

                nn.Flatten(),

                nn.Linear(in\_features, 256),

                nn.ReLU(),

                nn.Dropout(0.5),

                nn.Linear(256, num\_classes)

            )

            self.criterion = nn.CrossEntropyLoss()

        def forward(self, x):

            return self.feature\_extractor(x)

        def training\_step(self, batch, batch\_idx):

            images, labels = batch

            outputs = self(images)

            loss = self.criterion(outputs, labels)

            acc = (outputs.argmax(dim=1) == labels).float().mean()

            self.log("train\_loss", loss, on\_step=False, on\_epoch=True, prog\_bar=True)

            self.log("train\_acc", acc, on\_step=False, on\_epoch=True, prog\_bar=True)

            return loss

        def validation\_step(self, batch, batch\_idx):

            images, labels = batch

            outputs = self(images)

            loss = self.criterion(outputs, labels)

            acc = (outputs.argmax(dim=1) == labels).float().mean()

            self.log("val\_loss", loss, on\_step=False, on\_epoch=True, prog\_bar=True)

            self.log("val\_acc", acc, on\_step=False, on\_epoch=True, prog\_bar=True)

            return loss

        def test\_step(self, batch, batch\_idx):

            images, labels = batch

            outputs = self(images)

            acc = (outputs.argmax(dim=1) == labels).float().mean()

            self.log("test\_acc", acc, on\_step=False, on\_epoch=True, prog\_bar=True)

            return acc

        def configure\_optimizers(self):

            return torch.optim.Adam(self.parameters(), lr=self.lr)

    # 資料預處理

    transform = transforms.Compose([

        transforms.Resize((32, 32)),

        transforms.ToTensor(),

        transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])

    ])

    # 加載 CIFAR-10 資料集

    train\_dataset = CIFAR10(root="data", train=True, download=True, transform=transform)

    test\_dataset = CIFAR10(root="data", train=False, download=True, transform=transform)

    train\_loader = DataLoader(train\_dataset, batch\_size=64, shuffle=True, num\_workers=4)

    val\_loader = DataLoader(test\_dataset, batch\_size=64, shuffle=False, num\_workers=4)

    # 初始化模型

    model = VGGClassifier(use\_vgg16=True)  # 設為 False 使用 VGG19

    # 初始化 TensorBoard 記錄器

    logger = TensorBoardLogger("tb\_logs\_cifar", name=current\_time)

    # 設置回調函數

    checkpoint\_callback = ModelCheckpoint(

        monitor="val\_loss",

        dirpath=log\_dir,

        filename="best-checkpoint",

        save\_top\_k=1,

        mode="min"

    )

    early\_stopping\_callback = EarlyStopping(monitor="val\_loss", patience=5, mode="min")

    # 訓練器

    trainer = pl.Trainer(

        max\_epochs=10,

        logger=logger,

        callbacks=[checkpoint\_callback, early\_stopping\_callback],

        accelerator="gpu" if torch.cuda.is\_available() else "cpu",

        devices=1

    )

    # 訓練模型

    trainer.fit(model, train\_dataloaders=train\_loader, val\_dataloaders=val\_loader)

    # 測試模型

    trainer.test(model, dataloaders=val\_loader)

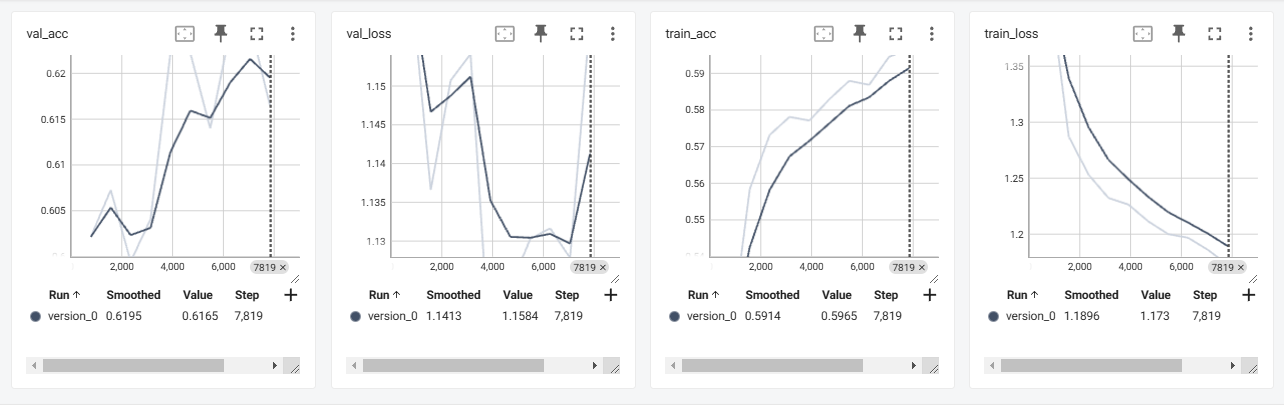
    print(f"測試完成！日誌儲存在: {log\_dir}")

# 確保程式在主模組中執行

if \_\_name\_\_ == '\_\_main\_\_':

    main()

5-3 vgg19

import os

from datetime import datetime

import torch

import torch.nn as nn

import torchvision.transforms as transforms

from torchvision.datasets import CIFAR10

from torchvision.models import vgg16, vgg19

from torch.utils.data import DataLoader

import pytorch\_lightning as pl

from pytorch\_lightning.loggers import TensorBoardLogger

from pytorch\_lightning.callbacks import ModelCheckpoint, EarlyStopping

def main():

    # 動態生成 TensorBoard 日誌目錄

    current\_time = datetime.now().strftime("%Y%m%d-%H%M%S")

    log\_dir = f"tb\_logs\_cifar/{current\_time}"

    os.makedirs(log\_dir, exist\_ok=True)

    print(f"TensorBoard 日誌目錄: {log\_dir}")

    # 定義模型

    class VGGClassifier(pl.LightningModule):

        def \_\_init\_\_(self, use\_vgg16=True, num\_classes=10, lr=0.001):

            super(VGGClassifier, self).\_\_init\_\_()

            self.save\_hyperparameters()  # 保存超參數

            self.lr = lr

            # 選擇 VGG16 或 VGG19 預訓練模型

            self.feature\_extractor = vgg16(pretrained=True) if use\_vgg16 else vgg19(pretrained=True)

            for param in self.feature\_extractor.parameters():

                param.requires\_grad = False  # 凍結預訓練權重

            # 修改分類器部分

            in\_features = self.feature\_extractor.classifier[0].in\_features

            self.feature\_extractor.classifier = nn.Sequential(

                nn.Flatten(),

                nn.Linear(in\_features, 256),

                nn.ReLU(),

                nn.Dropout(0.5),

                nn.Linear(256, num\_classes)

            )

            self.criterion = nn.CrossEntropyLoss()

        def forward(self, x):

            return self.feature\_extractor(x)

        def training\_step(self, batch, batch\_idx):

            images, labels = batch

            outputs = self(images)

            loss = self.criterion(outputs, labels)

            acc = (outputs.argmax(dim=1) == labels).float().mean()

            self.log("train\_loss", loss, on\_step=False, on\_epoch=True, prog\_bar=True)

            self.log("train\_acc", acc, on\_step=False, on\_epoch=True, prog\_bar=True)

            return loss

        def validation\_step(self, batch, batch\_idx):

            images, labels = batch

            outputs = self(images)

            loss = self.criterion(outputs, labels)

            acc = (outputs.argmax(dim=1) == labels).float().mean()

            self.log("val\_loss", loss, on\_step=False, on\_epoch=True, prog\_bar=True)

            self.log("val\_acc", acc, on\_step=False, on\_epoch=True, prog\_bar=True)

            return loss

        def test\_step(self, batch, batch\_idx):

            images, labels = batch

            outputs = self(images)

            acc = (outputs.argmax(dim=1) == labels).float().mean()

            self.log("test\_acc", acc, on\_step=False, on\_epoch=True, prog\_bar=True)

            return acc

        def configure\_optimizers(self):

            return torch.optim.Adam(self.parameters(), lr=self.lr)

    # 資料預處理

    transform = transforms.Compose([

        transforms.Resize((32, 32)),

        transforms.ToTensor(),

        transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])

    ])

    # 加載 CIFAR-10 資料集

    train\_dataset = CIFAR10(root="data", train=True, download=True, transform=transform)

    test\_dataset = CIFAR10(root="data", train=False, download=True, transform=transform)

    train\_loader = DataLoader(train\_dataset, batch\_size=64, shuffle=True, num\_workers=4)

    val\_loader = DataLoader(test\_dataset, batch\_size=64, shuffle=False, num\_workers=4)

    # 初始化模型

    model = VGGClassifier(use\_vgg16=False)  # 使用 VGG19

    # 初始化 TensorBoard 記錄器

    logger = TensorBoardLogger("tb\_logs\_cifar", name=current\_time)

    # 設置回調函數

    checkpoint\_callback = ModelCheckpoint(

        monitor="val\_loss",

        dirpath=log\_dir,

        filename="best-checkpoint",

        save\_top\_k=1,

        mode="min"

    )

    early\_stopping\_callback = EarlyStopping(monitor="val\_loss", patience=5, mode="min")

    # 訓練器

    trainer = pl.Trainer(

        max\_epochs=10,

        logger=logger,

        callbacks=[checkpoint\_callback, early\_stopping\_callback],

        accelerator="gpu" if torch.cuda.is\_available() else "cpu",

        devices=1

    )

    # 訓練模型

    trainer.fit(model, train\_dataloaders=train\_loader, val\_dataloaders=val\_loader)

    # 測試模型

    trainer.test(model, dataloaders=val\_loader)

    print(f"測試完成！日誌儲存在: {log\_dir}")

# 確保程式在主模組中執行

if \_\_name\_\_ == '\_\_main\_\_':

    main()