KAN Kolmogorov Arnold Network Vision Transformer

Kolmogorov Arnold Network

相關參考資料

- 原始論文：KAN: Kolmogorov-Arnold Network
- 原始論文網址：https://arxiv.org/abs/2404.19756
- 原始開源代碼：https://github.com/KindXiaoming/pykan

BIBTEX 供論文引用

```
@article{liu2024kan,
  title={KAN: Kolmogorov-Arnold Networks},
  author={Liu, Ziming and Wang, Yixuan and Vaidya, Sachin and Ruehle, Fabian and Halverson, James and
Solja{\v{c}}i{\'c}, Marin and Hou, Thomas Y and Tegmark, Max},
  journal={arXiv preprint arXiv:2404.19756},
  year={2024}
}
```

論文內容

Summary

KAN（Kolmogorov-Arnold Network）論文基本概念。

1. Research Background
   - Kolmogorov-Arnold Representation Theorem（KART）：此定理說明任何多變量的連續函數都可以表示為單變量連續函數和加法操作的有限組合。
   - Limitations of Multi-Layer Perceptron（MLP）：傳統由 MLP 形成的神經網路雖然具有強大的表達能力，但在某些應用中存在固定的 Activation Function，使得其解釋性較差且參數效率低下。
2. Kolmogorov-Arnold Network（KAN）
   - Network Structure：與 MLP 不同，KAN 在 Edge（即權重）上使用可學習的 Activation Function，而不是在 Node（即神經元）上使用固定的 Activation Function。
   - Activation Function：KAN 中的每個權重參數被替換為一個參數化為樣條函數的單變量函數。節點只進行簡單的信號相加操作，不應用任何非線性操作。
3. Advantages of KAN
   - Higher Precision：KAN 在數據擬合和偏微分方程求解方面比 MLP 更準確。例如，在偏微分方程求解中，一個兩層且寬度為十的 KAN 比一個四層且寬度為一百的 MLP 準確度高百倍。
   - Explainability：KAN 可以直觀地可視化，並能與人類用戶進行互動，有助於科學家重新發現數學和物理定律。

Summary

KAN（Kolmogorov-Arnold Network）和 MLP（Multi-Layer Perceptron）之間的主要差異與比較。

1. 激活函數的位置和特性
   - MLP：激活函數固定，位於節點（神經元）上；且激活函數一般是非線性函數，例如 ReLU、Sigmoid 等。
   - KAN：激活函數是可學習的，位於邊（權重）上；且每個權重參數被替換為一個參數化為樣條函數的單變量函數。
2. 網路結構和權重表示
   - MLP：使用線性權重矩陣進行計算，然後應用固定的非線性激活函數；節點進行非線性變換。
   - 結構公式：

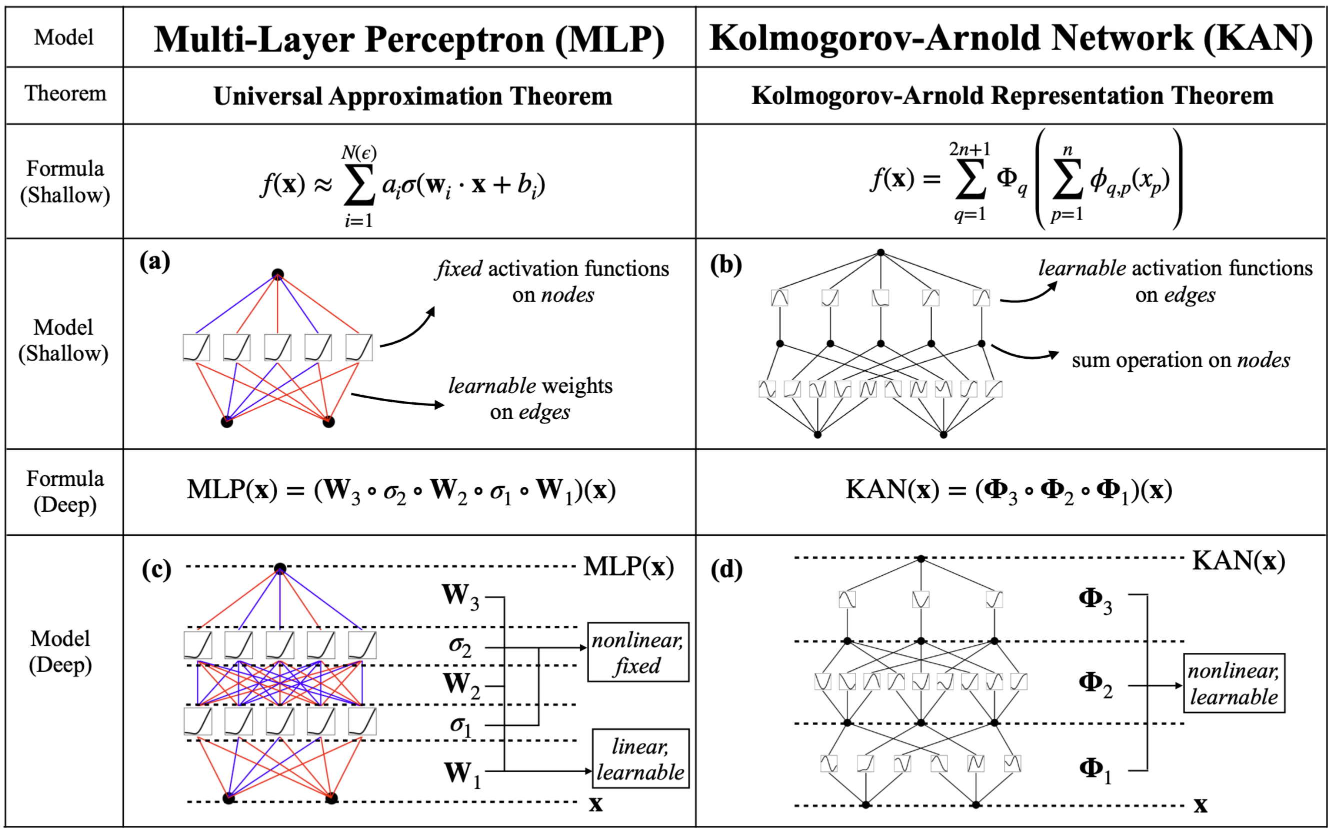
$$MLP(x) = ((W3 \circ \sigma 2 \circ W2 \circ \sigma 1 \circ W1)(x))$$

   - KAN：沒有線性權重矩陣，所有權重都被樣條函數替代；節點僅進行簡單的信號相加操作，不進行非線性變換。

- 結構公式：

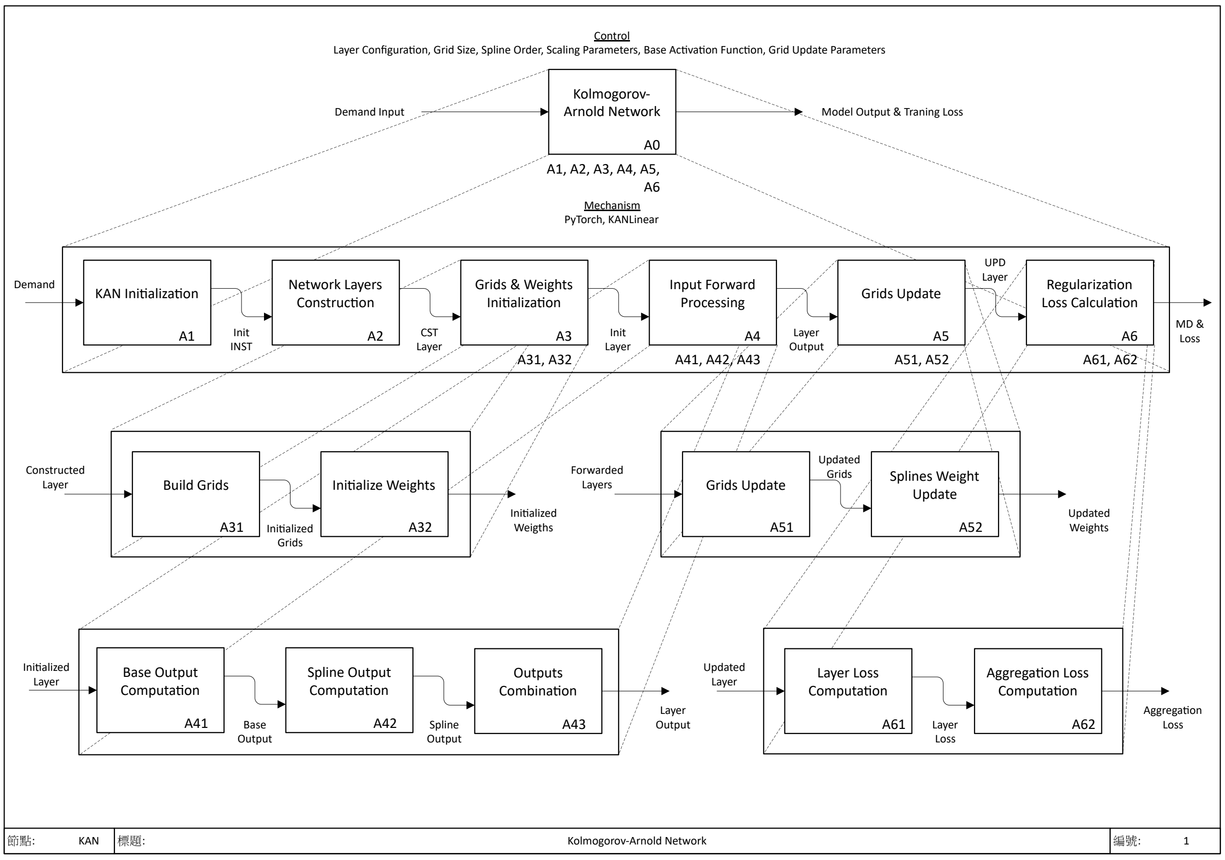
$$KAN(x) = ((\Phi 3 \circ \Phi 2 \circ \Phi 1)(x))$$

3. 訓練方法和參數優化
   - MLP：權重矩陣通過梯度下降法進行訓練；訓練過程需要調整大量的線性權重參數。
   - KAN：核心在於樣條函數的參數化和學習；樣條函數通過調整其參數進行優化。
4. 表達能力和適用範圍
   - MLP：基於普適近似定理，MLP 能夠逼近任意連續函數；常用於各種回歸和分類問題，但在高維數據下可能效率低下。
   - KAN：基於 Kolmogorov-Arnold 表示定理，能夠表達高維數據的組合結構和單變量函數；對於需要高準確度和解釋性的應用，如數學模型和物理模型，有顯著優勢。
5. 可解釋性和直觀性
   - MLP：由於固定的激活函數和複雜的權重矩陣，MLP 在解釋性方面較為薄弱；解釋模型需要額外的工具和方法，如 SHAP、LIME 等。
   - KAN：由於激活函數是可學習的單變量函數，KAN 的結構更易於直觀理解；KAN 的節點僅進行信號相加，使得整體網路更易於可視化和解釋。
6. 計算和資源需求
   - MLP：訓練和推理過程中，計算資源需求較大，特別是在高維數據和大模型情況下。
   - KAN：由於樣條函數的引入，KAN 在同樣準確度下所需的參數和計算資源相對較少；能夠在較小的計算圖上達到與大型 MLP 相同甚至更好的準確度。

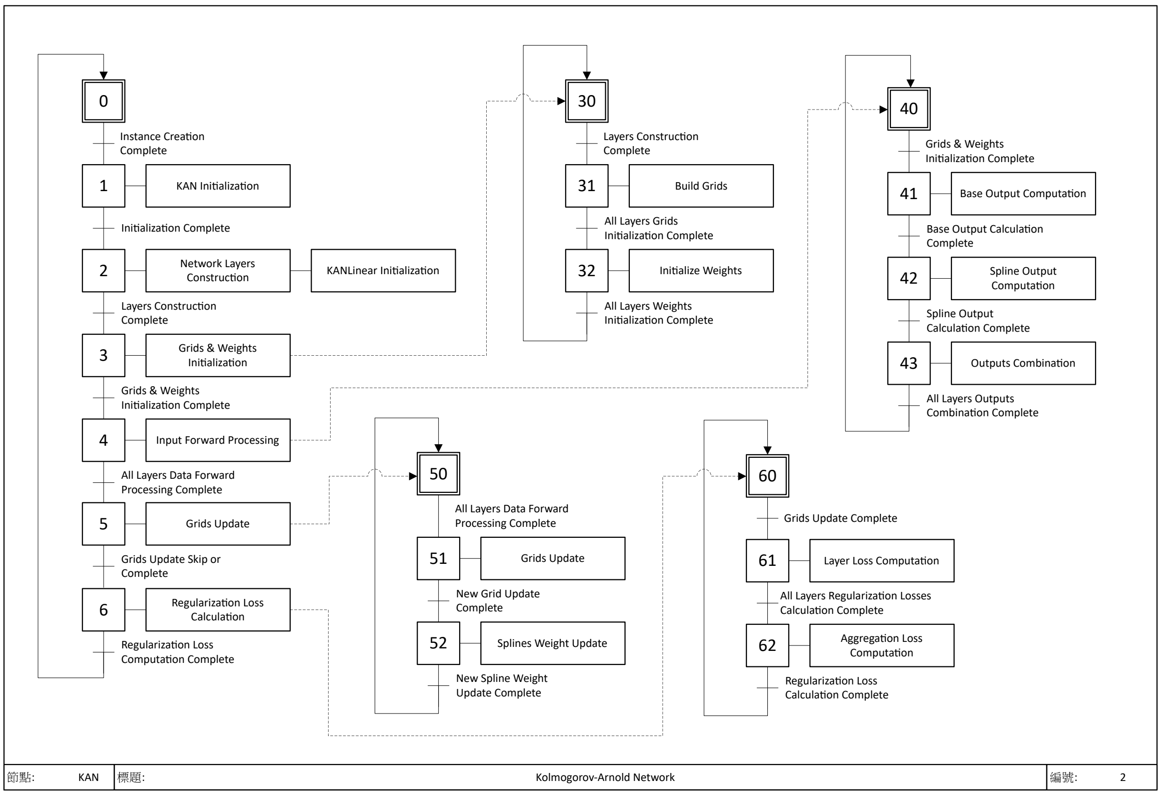


Model	Multi-Layer Perceptron (MLP)	Kolmogorov-Arnold Network (KAN)
Theorem	Universal Approximation Theorem	Kolmogorov-Arnold Representation Theorem
Formula (Shallow)	$f(\mathbf{x}) \approx \sum_{i=1}^{N(\epsilon)} a_i \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$	$f(\mathbf{x}) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right)$
Model (Shallow)	(a) *fixed* activation functions on *nodes*; *learnable* weights on *edges*	(b) *learnable* activation functions on *edges*; sum operation on *nodes*
Formula (Deep)	$\mathrm{MLP}(\mathbf{x}) = (\mathbf{W}_3 \circ \sigma_2 \circ \mathbf{W}_2 \circ \sigma_1 \circ \mathbf{W}_1)(\mathbf{x})$	$\mathrm{KAN}(\mathbf{x}) = (\mathbf{\Phi}_3 \circ \mathbf{\Phi}_2 \circ \mathbf{\Phi}_1)(\mathbf{x})$
Model (Deep)	(c) MLP(**x**); $\mathbf{W}_3$; σ_2; $\mathbf{W}_2$; σ_1; $\mathbf{W}_1$; **x**; *nonlinear, fixed*; *linear, learnable*	(d) KAN(**x**); $\mathbf{\Phi}_3$; $\mathbf{\Phi}_2$; $\mathbf{\Phi}_1$; **x**; *nonlinear, learnable*

IDEF0 設計



GRAFCET 設計



0
Instance Creation Complete
1
KAN Initialization
Initialization Complete
2
Network Layers Construction
KANLinear Initialization
Layers Construction Complete
3
Grids & Weights Initialization
Grids & Weights Initialization Complete
4
Input Forward Processing
All Layers Data Forward Processing Complete
5
Grids Update
Grids Update Skip or Complete
6
Regularization Loss Calculation
Regularization Loss Computation Complete
30
Layers Construction Complete
31
Build Grids
All Layers Grids Initialization Complete
32
Initialize Weights
All Layers Weights Initialization Complete
40
Grids & Weights Initialization Complete
41
Base Output Computation
Base Output Calculation Complete
42
Spline Output Computation
Spline Output Calculation Complete
43
Outputs Combination
All Layers Outputs Combination Complete
50
All Layers Data Forward Processing Complete
51
Grids Update
New Grid Update Complete
52
Splines Weight Update
New Spline Weight Update Complete
60
Grids Update Complete
61
Layer Loss Computation
All Layers Regularization Losses Calculation Complete
62
Aggregation Loss Computation
Regularization Loss Calculation Complete
節點: KAN
標題: Kolmogorov-Arnold Network
編號: 2

Python PyTorch 模擬驗證

- 基於 IDEF0 和 Grafcet 重構後之 Kolmogorov-Arnold Network

Info

基於 Grafcet 設計後的一些調整

- Original Implementation 的 Performance 問題主要在於需要展開所有中間變數來執行不同的 Activation Function。對於具有 `in_features` 輸入和 `out_features` 輸出的層，原始實現需要將輸入展開為形狀為 `(batch_size, out_features, in_features)` 的 Tensor 來執行 Activation Function。然而，所有的 Activation Function 都是一組固定 Base Function（B-Spline）的線性組合，也就可以將計算重新整理為使用不同的 Base Function 激活輸入，然後將它們線性組合。

```python
import math

import torch
import torch.nn.functional as F


class KANLinear(torch.nn.Module):
    def __init__(
            self,
            in_features,
            out_features,
            grid_size=5,
            spline_order=3,
            scale_base=1.0,
            scale_spline=1.0,
            enable_standalone_scale_spline=True,
            base_activation=torch.nn.SiLU,
            grid_eps=0.02,
            grid_range=[-1, 1],
    ):
        super(KANLinear, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.grid_size = grid_size
        self.spline_order = spline_order

        # 構建網格點
        self.grid = self.build_grid(grid_range, grid_size, spline_order)

        # 初始化基礎權重和樣條權重
        self.base_weight, self.spline_weight, self.spline_scaler = self.initialize_weights(
            out_features, in_features, grid_size, spline_order, scale_base, scale_spline,
enable_standalone_scale_spline
        )

        self.scale_base = scale_base
        self.scale_spline = scale_spline
        self.enable_standalone_scale_spline = enable_standalone_scale_spline
        self.base_activation = base_activation()
        self.grid_eps = grid_eps

    def build_grid(self, grid_range, grid_size, spline_order):
        h = (grid_range[1] - grid_range[0]) / grid_size
        grid = (
            (
                    torch.arange(-spline_order, grid_size + spline_order + 1) * h
                    + grid_range[0]
            )
            .expand(self.in_features, -1)
            .contiguous()
        )
        return grid

    def initialize_weights(self, out_features, in_features, grid_size, spline_order, scale_base, scale_spline,
                           enable_standalone_scale_spline):
        base_weight = torch.nn.Parameter(torch.Tensor(out_features, in_features))
        spline_weight = torch.nn.Parameter(
            torch.Tensor(out_features, in_features, grid_size + spline_order)
        )
        if enable_standalone_scale_spline:
            spline_scaler = torch.nn.Parameter(
                torch.Tensor(out_features, in_features)
            )
        else:
            spline_scaler = None
        torch.nn.init.kaiming_uniform_(base_weight, a=math.sqrt(5) * scale_base)
        torch.nn.init.kaiming_uniform_(spline_weight, a=math.sqrt(5) * scale_spline)
        if enable_standalone_scale_spline:
            torch.nn.init.kaiming_uniform_(spline_scaler, a=math.sqrt(5) * scale_spline)
        return base_weight, spline_weight, spline_scaler

    def b_splines(self, x: torch.Tensor):
        bases = self.calculate_b_spline_bases(x)
        return bases.contiguous()

    def calculate_b_spline_bases(self, x: torch.Tensor):
        grid: torch.Tensor = (
            self.grid
        )  # (in_features, grid_size + 2 * spline_order + 1)
        x = x.unsqueeze(-1)
        bases = ((x >= grid[:, :-1]) & (x < grid[:, 1:])).to(x.dtype)
        for k in range(1, self.spline_order + 1):
            bases = (
                            (x - grid[:, : -(k + 1)])
                            / (grid[:, k:-1] - grid[:, : -(k + 1)])
                            * bases[:, :, :-1]
                    ) + (
                            (grid[:, k + 1:] - x)
                            / (grid[:, k + 1:] - grid[:, 1:(-k)])
                            * bases[:, :, 1:]
                    )
        return bases

    def curve2coeff(self, x: torch.Tensor, y: torch.Tensor):
        A = self.b_splines(x).transpose(
            0, 1
        )  # (in_features, batch_size, grid_size + spline_order)
        B = y.transpose(0, 1)  # (in_features, batch_size, out_features)
        solution = torch.linalg.lstsq(
            A, B
        ).solution  # (in_features, grid_size + spline_order, out_features)
        result = solution.permute(
            2, 0, 1
        )  # (out_features, in_features, grid_size + spline_order)
        return result.contiguous()

    @property
    def scaled_spline_weight(self):
        if self.enable_standalone_scale_spline:
            return self.spline_weight * self.spline_scaler.unsqueeze(-1)
        else:
            return self.spline_weight

    def forward(self, x: torch.Tensor):
        base_output = self.compute_base_output(x)
        spline_output = self.compute_spline_output(x)
        return base_output + spline_output

    def compute_base_output(self, x: torch.Tensor):
        return F.linear(self.base_activation(x), self.base_weight)

    def compute_spline_output(self, x: torch.Tensor):
        return F.linear(
            self.b_splines(x).view(x.size(0), -1),
            self.scaled_spline_weight.view(self.out_features, -1),
        )

    @torch.no_grad()
    def update_grid(self, x: torch.Tensor, margin=0.01):
        batch = x.size(0)
        splines = self.b_splines(x)  # (batch, in, coeff)
        splines = splines.permute(1, 0, 2)  # (in, batch, coeff)
        orig_coeff = self.scaled_spline_weight  # (out, in, coeff)
        orig_coeff = orig_coeff.permute(1, 2, 0)  # (in, coeff, out)
        unreduced_spline_output = torch.bmm(splines, orig_coeff)  # (in, batch, out)
        unreduced_spline_output = unreduced_spline_output.permute(
            1, 0, 2
        )  # (batch, in, out)

        x_sorted = torch.sort(x, dim=0)[0]
        grid_adaptive = x_sorted[
            torch.linspace(
                0, batch - 1, self.grid_size + 1, dtype=torch.int64, device=x.device
            )
        ]

        uniform_step = (x_sorted[-1] - x_sorted[0] + 2 * margin) / self.grid_size
        grid_uniform = (
                torch.arange(
                    self.grid_size + 1, dtype=torch.float32, device=x.device
                ).unsqueeze(1)
                * uniform_step
                + x_sorted[0]
                - margin
        )

        grid = self.grid_eps * grid_uniform + (1 - self.grid_eps) * grid_adaptive
        grid = torch.concatenate(
            [
                grid[:1]
                - uniform_step
                * torch.arange(self.spline_order, 0, -1, device=x.device).unsqueeze(1),
                grid,
                grid[-1:]
                + uniform_step
                * torch.arange(1, self.spline_order + 1, device=x.device).unsqueeze(1),
            ],
            dim=0,
        )

        self.grid.copy_(grid.T)
        self.spline_weight.data.copy_(self.curve2coeff(x, unreduced_spline_output))

    def regularization_loss(self, regularize_activation=1.0, regularize_entropy=1.0):
        l1_fake = self.spline_weight.abs().mean(-1)
        regularization_loss_activation = l1_fake.sum()
        p = l1_fake / regularization_loss_activation
        regularization_loss_entropy = -torch.sum(p * p.log())
        return (
                regularize_activation * regularization_loss_activation
                + regularize_entropy * regularization_loss_entropy
        )


class KAN(torch.nn.Module):
    def __init__(
            self,
            layers_hidden,
            grid_size=5,
            spline_order=3,
            scale_base=1.0,
            scale_spline=1.0,
            base_activation=torch.nn.SiLU,
            grid_eps=0.02,
            grid_range=[-1, 1],
    ):
        super(KAN, self).__init__()
        self.grid_size = grid_size
        self.spline_order = spline_order
        # 構建 KAN 的層
        self.layers = self.build_layers(
            layers_hidden, grid_size, spline_order, scale_base, scale_spline, base_activation, grid_eps,
grid_range
        )

    def build_layers(self, layers_hidden, grid_size, spline_order, scale_base, scale_spline, base_activation,
grid_eps,
                     grid_range):
        layers = torch.nn.ModuleList()
        for in_features, out_features in zip(layers_hidden, layers_hidden[1:]):
            layers.append(
                KANLinear(
                    in_features,
                    out_features,
                    grid_size=grid_size,
                    spline_order=spline_order,
                    scale_base=scale_base,
                    scale_spline=scale_spline,
                    base_activation=base_activation,
                    grid_eps=grid_eps,
                    grid_range=grid_range,
                )
            )
        return layers

    def forward(self, x: torch.Tensor, update_grid=False):
        for layer in self.layers:
            if update_grid:
                layer.update_grid(x)
            x = layer(x)
        return x

    def regularization_loss(self, regularize_activation=1.0, regularize_entropy=1.0):
        return sum(
            layer.regularization_loss(regularize_activation, regularize_entropy)
            for layer in self.layers
        )
```

Transformer

相關參考資料

- 原始論文：**Attention is All You Need**
- 原始論文網址：https://arxiv.org/abs/1706.03762
- 原始開源代碼：https://github.com/huggingface/transformers

BIBTEX 供論文引用（HuggingFace Library）

```
@inproceedings{wolf-etal-2020-transformers,
    title = "Transformers: State-of-the-Art Natural Language Processing",
    author = "Thomas Wolf and Lysandre Debut and Victor Sanh and Julien Chaumond and Clement Delangue and Anthony
Moi and Pierric Cistac and Tim Rault and Rémi Louf and Morgan Funtowicz and Joe Davison and Sam Shleifer and
Patrick von Platen and Clara Ma and Yacine Jernite and Julien Plu and Canwen Xu and Teven Le Scao and Sylvain
Gugger and Mariama Drame and Quentin Lhoest and Alexander M. Rush",
    booktitle = "Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System
Demonstrations",
    month = oct,
    year = "2020",
    address = "Online",
    publisher = "Association for Computational Linguistics",
    url = "https://www.aclweb.org/anthology/2020.emnlp-demos.6",
    pages = "38--45"
}
```

論文內容

- 全局分析
- 位置編碼
- 多頭注意力機制
- 殘差
- Batch Normal
- Layer Normal
- Decoder

Vision Transformer (ViT)

相關參考資料

- 原始論文：An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale
- 原始論文網址：https://arxiv.org/abs/2010.11929
- 原始開源代碼：https://github.com/google-research/vision_transformer

BIBTEX 供論文引用

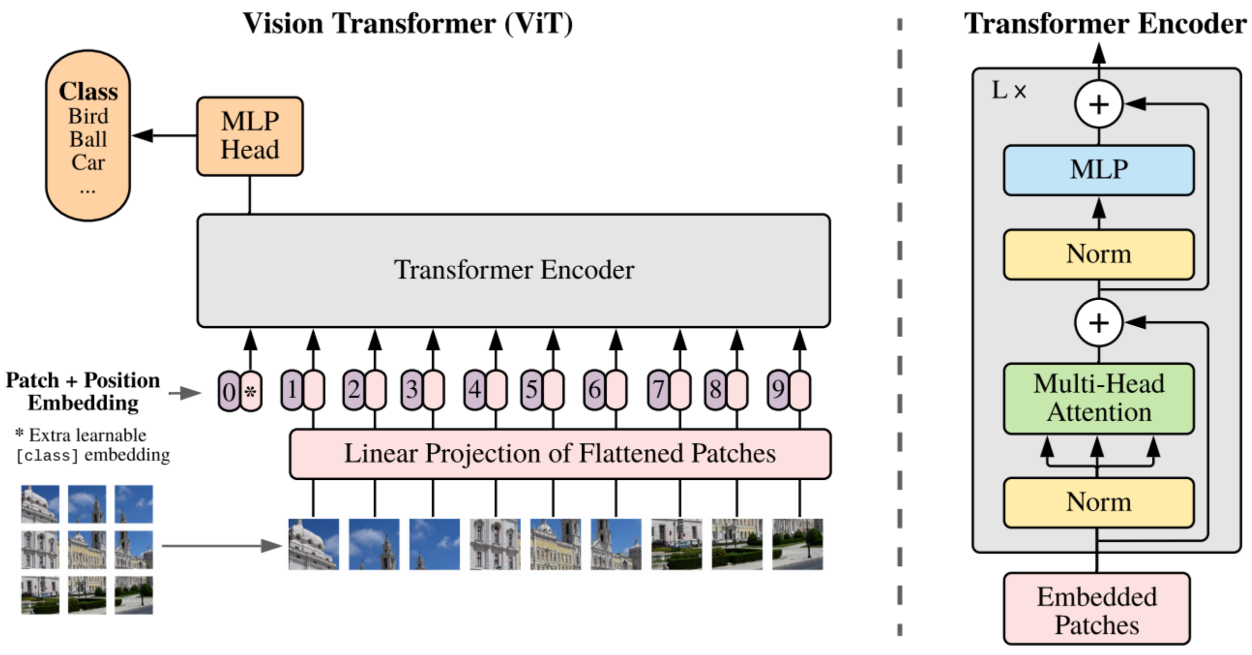
```
@article{dosovitskiy2020vit,
  title={An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale},
  author={Dosovitskiy, Alexey and Beyer, Lucas and Kolesnikov, Alexander and Weissenborn, Dirk and Zhai, Xiaohua and Unterthiner, Thomas and  Dehghani, Mostafa and Minderer, Matthias and Heigold, Georg and Gelly, Sylvain and Uszkoreit, Jakob and Houlsby, Neil},
  journal={ICLR},
  year={2021}
}
```

論文內容

Important

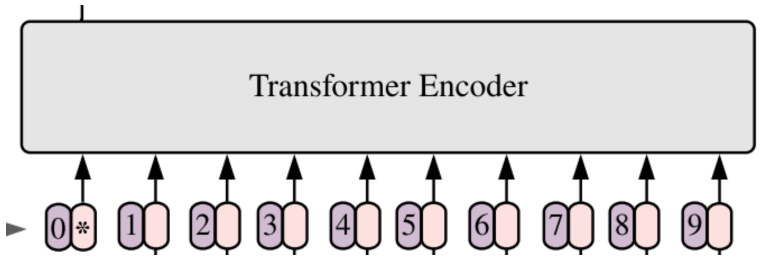
簡要的來說，Vision Transformer (ViT) 的模型由三個 Module 組成，分別是：

- Linear Projection of Flattened Patches（Embedding 層）
- Transformer Encoder
- MLP Head（最終用於分類的層結構）



1. Embedding 層結構詳解

對於標準的 Transformer 模塊，要求輸入的是 Token（向量）序列，即二維矩陣 [num_token, token_dim]，參考下圖，Token 0 - 9 對應的都是向量，就最基本的 ViT-B/16 來說，每個 Token 向量的長度為 768。



對於影像數據而言，其數據格式為 [H, W, C]，是一個三維的矩陣，很明顯不是 Transformer 想要的，所以我們會需要先通過一個 Embedding 層來對數據進行變換。如下圖所示，首先將一張圖片依據給定的大小分成一堆 Patches，同樣以 ViT-B/16 舉例，將輸入圖片（224X224）按照 16X16 大小的 Patch 進行劃分，劃分後會得到 (224/16)^2 也就是 196 個 Patches。接著通過線性映射把每個 Patch 映射到一維向量中，就 ViT-B/16 的情況來看，每個 Patch 的 Data Shape 維 [16, 16, 3]，通過映射得到一個長度維 768 的向量（後續統稱為 Token）

$$[16,16,3] \rightarrow [768]$$

在代碼的實現中，直接通過一個卷積層來進行實現。以 ViT-B/16 為例，直接使用一個卷積核大小為 16X16，步距為 16，卷稽核個數為 768 的卷積來實現。通過卷積

$$[224,224,3] \rightarrow [14,14,768]$$

然後把 H 以及 W 兩個維度展平即可

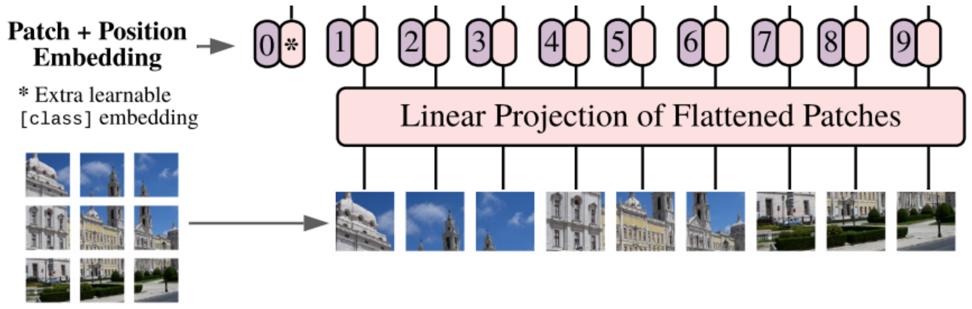
$$[14,14,768] \rightarrow [196,768]$$

此時正好變成了一個二維矩陣，而這就是 Transformer 希望也想要的。

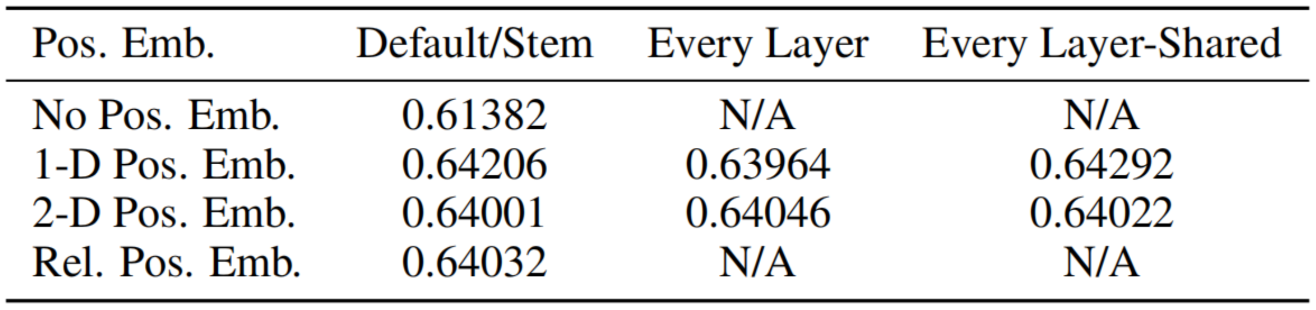
在輸入 Transformer Encoder 之前需要注意加上 [class]Token 以及 Position Embedding。論文中作者有提到 ViT 參考了 BERT，在剛剛得到的一對 Tokens 中插入一個專門用於分類的 [class]Token，這個 [class]Token 是一個可訓練的參數，數據格式和其他 Token 一樣都是一個向量，以 ViT-B/16 為例，就是一個長度為 768 的向量，與之前從圖片中生成的 Tokens 拼接再一起

$$Cat([1,768],[196,768]) \rightarrow [197,768]$$

然後關於 Position Embedding 就是原先 Transformer 所提到的 Positional Encoding，這裡的 Position Embedding 採用的是一個可訓練的參數（1-D Pos. Emb.），是直接疊加在 Tokens 上面的（add），所以 Shape 要一樣。以 ViT-B/16 為例，剛剛拼接 [class]Token 後 Shape 是 [197, 768]，那麼這裡的 Position Embedding 的 Shape 也要是 [197, 768]。


Patch + Position Embedding
* Extra learnable [class] embedding
0
*
1
2
3
4
5
6
7
8
9
Linear Projection of Flattened Patches

對於 Position Embedding 原文也有進行一系列嘗試，原始碼中 Default 是使用 1D Pos. Emb.，對比不使用 Position Embedding 準確率大約提升了百分之三，不過 2D Pos. Emb. 沒有甚麼明顯的差異。



Pos. Emb.	Default/Stem	Every Layer	Every Layer-Shared
No Pos. Emb.	0.61382	N/A	N/A
1-D Pos. Emb.	0.64206	0.63964	0.64292
2-D Pos. Emb.	0.64001	0.64046	0.64022
Rel. Pos. Emb.	0.64032	N/A	N/A

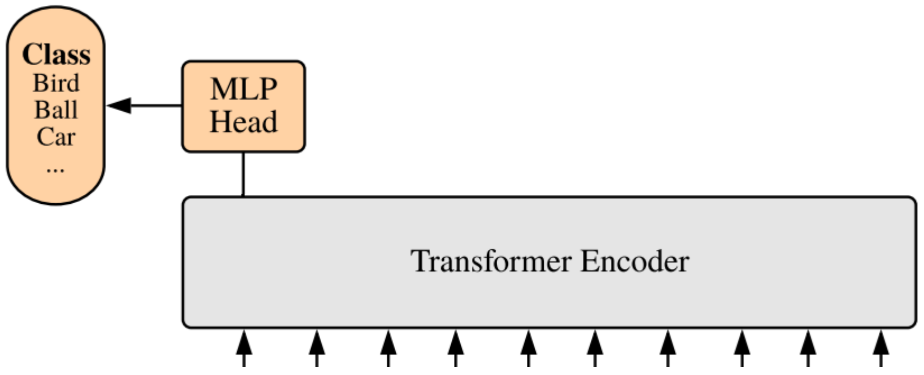
2. Transformer Encoder 結構詳解

Transformer Encoder 事實上就是重複堆疊 Encoder Block L 次，下圖是我個人基於理解所畫出的 Encoder Block，主要由下列幾個部分組成

- Layer Norm：
- Multi-Head Attention：
- Dropout/DropPath：
- MLP Block：

自繪圖片

3. MLP Head 結構詳解



4. 為了方便理解自行繪製的 Vision Transformer 架構（以 ViT-B/16 為例）

自繪圖片

ViT PyTorch 重構

相關說明

Info

代碼使用流程介紹

1. 下載好資料集（Dataset），目前代碼預設（Default）用的是花的分類（tf_flowers）：https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz
2. 在 `train.py` 中把 `--data-path` 設定為解壓縮後的 `flower_photos` 文件夾的絕對路徑。
3. 下載預訓練權重（Pre-Trained Weight），在 `vit_model.py` 中，每個模型都有提供預訓練權重的下載地址，根據選擇的模型下載對應的預訓練權重。
4. 在 `train.py` 腳本中將 `--weights` 參數設定為下載好的預訓練權重路徑。
5. 設定好資料集的路徑 `--data-path` 以及預訓練權重的路徑 `--weights`，即可使用 `train.py` 開始訓練（訓練過程中會自動生成 `class_indices.json` 文件）。
6. 在 `predict.py` 中導入與訓練代碼中相同的模型，並將 `model_weight_path` 設定為訓練好的模型權重路徑（預設保存在 `weights` 文件夾下）。
7. 在 `predict.py` 中將 `img_path` 設定為你需要預測的圖片的絕對路徑。
8. 設定好權重路徑 `model_weight_path` 和預測圖片路徑 `img_path`，即可使用 `predict.py` 進行預測。
9. 如果要使用其他資料集，就比照預設採用的「花的分類」之文件結構進行擺放（即一個類別對應一個文件夾），並且將訓練及預測代碼中的 `num_classes` 設定為你自己的數據的類別數。

> Quote
>
> 額外優化參考論文：Bag of Tricks for Image Classification with Convolutional Neural Networks
> 額外優化參考論文網址：https://arxiv.org/abs/1812.01187

```
@misc{1812.01187,
Author = {Tong He and Zhi Zhang and Hang Zhang and Zhongyue Zhang and Junyuan Xie and Mu Li},
Title = {Bag of Tricks for Image Classification with Convolutional Neural Networks},
Year = {2018},
Eprint = {arXiv:1812.01187},
}
```

代碼實現

> Important
>
> PyTorch 重構後之所有 Vision Transformer 代碼
>
> - 代碼倉庫：https://github.com/toby0622/Kolmogorov-Arnold-Network-Vision-Transformer

- vit_model.py

```python
import torch
import torch.nn as nn
from functools import partial
from collections import OrderedDict


def drop_path(x, drop_prob: float = 0.0, training: bool = False):
    if drop_prob == 0.0 or not training:
        return x

    keep_prob = 1 - drop_prob
    shape = (x.shape[0],) + (1,) * (
        x.ndim - 1
    )  # 適用於不同維度的張量，而不僅僅是 2D 卷積網絡
    random_tensor = keep_prob + torch.rand(shape, dtype=x.dtype, device=x.device)
    random_tensor.floor_()  # 二值化
    output = x.div(keep_prob) * random_tensor

    return output


# Drop paths (Stochastic Depth) per sample (when applied in main path of residual blocks)
class DropPath(nn.Module):
    def __init__(self, drop_prob=None):
        super(DropPath, self).__init__()
        self.drop_prob = drop_prob
    def forward(self, x):
        return drop_path(x, self.drop_prob, self.training)


# 2D Image to Patch Embedding
class PatchEmbed(nn.Module):
    def __init__(
        self, img_size=224, patch_size=16, in_c=3, embed_dim=768, norm_layer=None
    ):
        super().__init__()
        img_size = (img_size, img_size)
        patch_size = (patch_size, patch_size)
        self.img_size = img_size
        self.patch_size = patch_size
        self.grid_size = (img_size[0] // patch_size[0], img_size[1] // patch_size[1])
        self.num_patches = self.grid_size[0] * self.grid_size[1]

        self.proj = nn.Conv2d(
            in_c, embed_dim, kernel_size=patch_size, stride=patch_size
        )
        self.norm = norm_layer(embed_dim) if norm_layer else nn.Identity()

    def forward(self, x):
        B, C, H, W = x.shape
        assert (
            H == self.img_size[0] and W == self.img_size[1]
        ), f"輸入圖像大小 ({H}*{W}) 與模型大小 ({self.img_size[0]}*{self.img_size[1]}) 不匹配。"

        # flatten: [B, C, H, W] -> [B, C, HW]
        # transpose: [B, C, HW] -> [B, HW, C]
        x = self.proj(x).flatten(2).transpose(1, 2)
        x = self.norm(x)
        return x


class Attention(nn.Module):
    def __init__(
        self,
        dim,  # 輸入 token 的維度
        num_heads=8,
        qkv_bias=False,
        qk_scale=None,
        attn_drop_ratio=0.0,
        proj_drop_ratio=0.0,
    ):
        super(Attention, self).__init__()
        self.num_heads = num_heads
        head_dim = dim // num_heads
        self.scale = qk_scale or head_dim**-0.5
        self.qkv = nn.Linear(dim, dim * 3, bias=qkv_bias)
        self.attn_drop = nn.Dropout(attn_drop_ratio)
        self.proj = nn.Linear(dim, dim)
        self.proj_drop = nn.Dropout(proj_drop_ratio)

    def forward(self, x):
        # [batch_size, num_patches + 1, total_embed_dim]
        B, N, C = x.shape

        # qkv(): -> [batch_size, num_patches + 1, 3 * total_embed_dim]
        # reshape: -> [batch_size, num_patches + 1, 3, num_heads, embed_dim_per_head]
        # permute: -> [3, batch_size, num_heads, num_patches + 1, embed_dim_per_head]
        qkv = (
            self.qkv(x)
            .reshape(B, N, 3, self.num_heads, C // self.num_heads)
            .permute(2, 0, 3, 1, 4)
        )
        # [batch_size, num_heads, num_patches + 1, embed_dim_per_head]
        q, k, v = (
            qkv[0],
            qkv[1],
            qkv[2],
        )  # make torchscript happy (cannot use tensor as tuple)

        # transpose: -> [batch_size, num_heads, embed_dim_per_head, num_patches + 1]
        # @: multiply -> [batch_size, num_heads, num_patches + 1, num_patches + 1]
        attn = (q @ k.transpose(-2, -1)) * self.scale
        attn = attn.softmax(dim=-1)
        attn = self.attn_drop(attn)

        # @: multiply -> [batch_size, num_heads, num_patches + 1, embed_dim_per_head]
        # transpose: -> [batch_size, num_patches + 1, num_heads, embed_dim_per_head]
        # reshape: -> [batch_size, num_patches + 1, total_embed_dim]
        x = (attn @ v).transpose(1, 2).reshape(B, N, C)
        x = self.proj(x)
        x = self.proj_drop(x)
        return x


# Vision Transformer 使用的 MLP
class Mlp(nn.Module):
    def __init__(
        self,
        in_features,
        hidden_features=None,
        out_features=None,
        act_layer=nn.GELU,
        drop=0.0,
    ):
        super().__init__()
        out_features = out_features or in_features
        hidden_features = hidden_features or in_features
        self.fc1 = nn.Linear(in_features, hidden_features)
        self.act = act_layer()
        self.fc2 = nn.Linear(hidden_features, out_features)
        self.drop = nn.Dropout(drop)

    def forward(self, x):
        x = self.fc1(x)
        x = self.act(x)
        x = self.drop(x)
        x = self.fc2(x)
        x = self.drop(x)
        return x


class Block(nn.Module):
    def __init__(
        self,
        dim,
        num_heads,
        mlp_ratio=4.0,
        qkv_bias=False,
        qk_scale=None,
        drop_ratio=0.0,
        attn_drop_ratio=0.0,
        drop_path_ratio=0.0,
        act_layer=nn.GELU,
        norm_layer=nn.LayerNorm,
    ):
        super(Block, self).__init__()
        self.norm1 = norm_layer(dim)
        self.attn = Attention(
            dim,
            num_heads=num_heads,
            qkv_bias=qkv_bias,
            qk_scale=qk_scale,
            attn_drop_ratio=attn_drop_ratio,
            proj_drop_ratio=drop_ratio,
        )
        # 注意：隨機深度的丟棄路徑，我們將看看這是否比 dropout 更好
        self.drop_path = (
            DropPath(drop_path_ratio) if drop_path_ratio > 0.0 else nn.Identity()
        )
        self.norm2 = norm_layer(dim)
        mlp_hidden_dim = int(dim * mlp_ratio)
        self.mlp = Mlp(
            in_features=dim,
            hidden_features=mlp_hidden_dim,
            act_layer=act_layer,
            drop=drop_ratio,
        )

    def forward(self, x):
        x = x + self.drop_path(self.attn(self.norm1(x)))
        x = x + self.drop_path(self.mlp(self.norm2(x)))
        return x


class VisionTransformer(nn.Module):
    def __init__(
        self,
        img_size=224,
        patch_size=16,
        in_c=3,
        num_classes=1000,
        embed_dim=768,
        depth=12,
        num_heads=12,
        mlp_ratio=4.0,
        qkv_bias=True,
        qk_scale=None,
        representation_size=None,
        distilled=False,
        drop_ratio=0.0,
        attn_drop_ratio=0.0,
        drop_path_ratio=0.0,
        embed_layer=PatchEmbed,
        norm_layer=None,
        act_layer=None,
    ):
        """
        Args:
            img_size (int, tuple): input image size
            patch_size (int, tuple): patch size
            in_c (int): number of input channels
            num_classes (int): number of classes for classification head
            embed_dim (int): embedding dimension
            depth (int): depth of transformer
            num_heads (int): number of attention heads
            mlp_ratio (int): ratio of mlp hidden dim to embedding dim
            qkv_bias (bool): enable bias for qkv if True
            qk_scale (float): override default qk scale of head_dim ** -0.5 if set
            representation_size (Optional[int]): enable and set representation layer (pre-logits) to this value
if set
            distilled (bool): model includes a distillation token and head as in DeiT models
            drop_ratio (float): dropout rate
            attn_drop_ratio (float): attention dropout rate
            drop_path_ratio (float): stochastic depth rate
            embed_layer (nn.Module): patch embedding layer
            norm_layer: (nn.Module): normalization layer
        """
        super(VisionTransformer, self).__init__()
        self.num_classes = num_classes
        self.num_features = self.embed_dim = (
            embed_dim  # 為了與其他模型一致，使用 num_features
        )
        self.num_tokens = 2 if distilled else 1
        norm_layer = norm_layer or partial(nn.LayerNorm, eps=1e-6)
        act_layer = act_layer or nn.GELU

        self.patch_embed = embed_layer(
            img_size=img_size, patch_size=patch_size, in_c=in_c, embed_dim=embed_dim
```

```python
        )
        num_patches = self.patch_embed.num_patches

        self.cls_token = nn.Parameter(torch.zeros(1, 1, embed_dim))
        self.dist_token = (
            nn.Parameter(torch.zeros(1, 1, embed_dim)) if distilled else None
        )
        self.pos_embed = nn.Parameter(
            torch.zeros(1, num_patches + self.num_tokens, embed_dim)
        )
        self.pos_drop = nn.Dropout(p=drop_ratio)

        dpr = [
            x.item() for x in torch.linspace(0, drop_path_ratio, depth)
        ]  # 隨機深度衰減規則
        self.blocks = nn.Sequential(
            *[
                Block(
                    dim=embed_dim,
                    num_heads=num_heads,
                    mlp_ratio=mlp_ratio,
                    qkv_bias=qkv_bias,
                    qk_scale=qk_scale,
                    drop_ratio=drop_ratio,
                    attn_drop_ratio=attn_drop_ratio,
                    drop_path_ratio=dpr[i],
                    norm_layer=norm_layer,
                    act_layer=act_layer,
                )
                for i in range(depth)
            ]
        )
        self.norm = norm_layer(embed_dim)

        # 表示層
        if representation_size and not distilled:
            self.has_logits = True
            self.num_features = representation_size
            self.pre_logits = nn.Sequential(
                OrderedDict(
                    [
                        ("fc", nn.Linear(embed_dim, representation_size)),
                        ("act", nn.Tanh()),
                    ]
                )
            )
        else:
            self.has_logits = False
            self.pre_logits = nn.Identity()

        # 分類頭
        self.head = (
            nn.Linear(self.num_features, num_classes)
            if num_classes > 0
            else nn.Identity()
        )
        self.head_dist = None
        if distilled:
            self.head_dist = (
                nn.Linear(self.embed_dim, self.num_classes)
                if num_classes > 0
                else nn.Identity()
            )

        # 權重初始化
        nn.init.trunc_normal_(self.pos_embed, std=0.02)
        if self.dist_token is not None:
            nn.init.trunc_normal_(self.dist_token, std=0.02)

        nn.init.trunc_normal_(self.cls_token, std=0.02)
        self.apply(_init_vit_weights)
```

```python
    def forward_features(self, x):
        # [B, C, H, W] → [B, num_patches, embed_dim]
        x = self.patch_embed(x)  # [B, 196, 768]
        # [1, 1, 768] → [B, 1, 768]
        cls_token = self.cls_token.expand(x.shape[0], -1, -1)
        if self.dist_token is None:
            x = torch.cat((cls_token, x), dim=1)  # [B, 197, 768]
        else:
            x = torch.cat(
                (cls_token, self.dist_token.expand(x.shape[0], -1, -1), x), dim=1
            )

        x = self.pos_drop(x + self.pos_embed)
        x = self.blocks(x)
        x = self.norm(x)
        if self.dist_token is None:
            return self.pre_logits(x[:, 0])
        else:
            return x[:, 0], x[:, 1]

    def forward(self, x):
        x = self.forward_features(x)
        if self.head_dist is not None:
            x, x_dist = self.head(x[0]), self.head_dist(x[1])
            if self.training and not torch.jit.is_scripting():
                # 推理期間，返回兩個分類器預測的平均值
                return x, x_dist
            else:
                return (x + x_dist) / 2
        else:
            x = self.head(x)
        return x


# ViT 權重初始化
def _init_vit_weights(m):
    # parameter m = module
    if isinstance(m, nn.Linear):
        nn.init.trunc_normal_(m.weight, std=0.01)
        if m.bias is not None:
            nn.init.zeros_(m.bias)
    elif isinstance(m, nn.Conv2d):
        nn.init.kaiming_normal_(m.weight, mode="fan_out")
        if m.bias is not None:
            nn.init.zeros_(m.bias)
    elif isinstance(m, nn.LayerNorm):
        nn.init.zeros_(m.bias)
        nn.init.ones_(m.weight)


def vit_base_patch16_224(num_classes: int = 1000):
    # ViT-Base model (ViT-B/16) ImageNet-1k weights @ 224x224
    model = VisionTransformer(
        img_size=224,
        patch_size=16,
        embed_dim=768,
        depth=12,
        num_heads=12,
        representation_size=None,
        num_classes=num_classes,
    )

    return model


def vit_base_patch16_224_in21k(num_classes: int = 21843, has_logits: bool = True):
    # ViT-Base model (ViT-B/16) ImageNet-21k weights @ 224x224
    model = VisionTransformer(
        img_size=224,
        patch_size=16,
        embed_dim=768,
        depth=12,
        num_heads=12,
        representation_size=768 if has_logits else None,
        num_classes=num_classes,
    )

    return model


def vit_base_patch32_224(num_classes: int = 1000):
    # ViT-Base model (ViT-B/32) ImageNet-1k weights @ 224x224
    model = VisionTransformer(
        img_size=224,
        patch_size=32,
        embed_dim=768,
        depth=12,
        num_heads=12,
        representation_size=None,
        num_classes=num_classes,
    )

    return model


def vit_base_patch32_224_in21k(num_classes: int = 21843, has_logits: bool = True):
    # ViT-Base model (ViT-B/32) ImageNet-21k weights @ 224x224
    model = VisionTransformer(
        img_size=224,
        patch_size=32,
        embed_dim=768,
        depth=12,
        num_heads=12,
        representation_size=768 if has_logits else None,
        num_classes=num_classes,
    )

    return model


def vit_large_patch16_224(num_classes: int = 1000):
    # ViT-Large model (ViT-L/16) ImageNet-1k weights @ 224x224
    model = VisionTransformer(
        img_size=224,
        patch_size=16,
        embed_dim=1024,
        depth=24,
        num_heads=16,
        representation_size=None,
        num_classes=num_classes,
    )

    return model


def vit_large_patch16_224_in21k(num_classes: int = 21843, has_logits: bool = True):
    # ViT-Large model (ViT-L/16) ImageNet-21k weights @ 224x224
    model = VisionTransformer(
        img_size=224,
        patch_size=16,
        embed_dim=1024,
        depth=24,
        num_heads=16,
        representation_size=1024 if has_logits else None,
        num_classes=num_classes,
    )

    return model


def vit_large_patch32_224_in21k(num_classes: int = 21843, has_logits: bool = True):
```

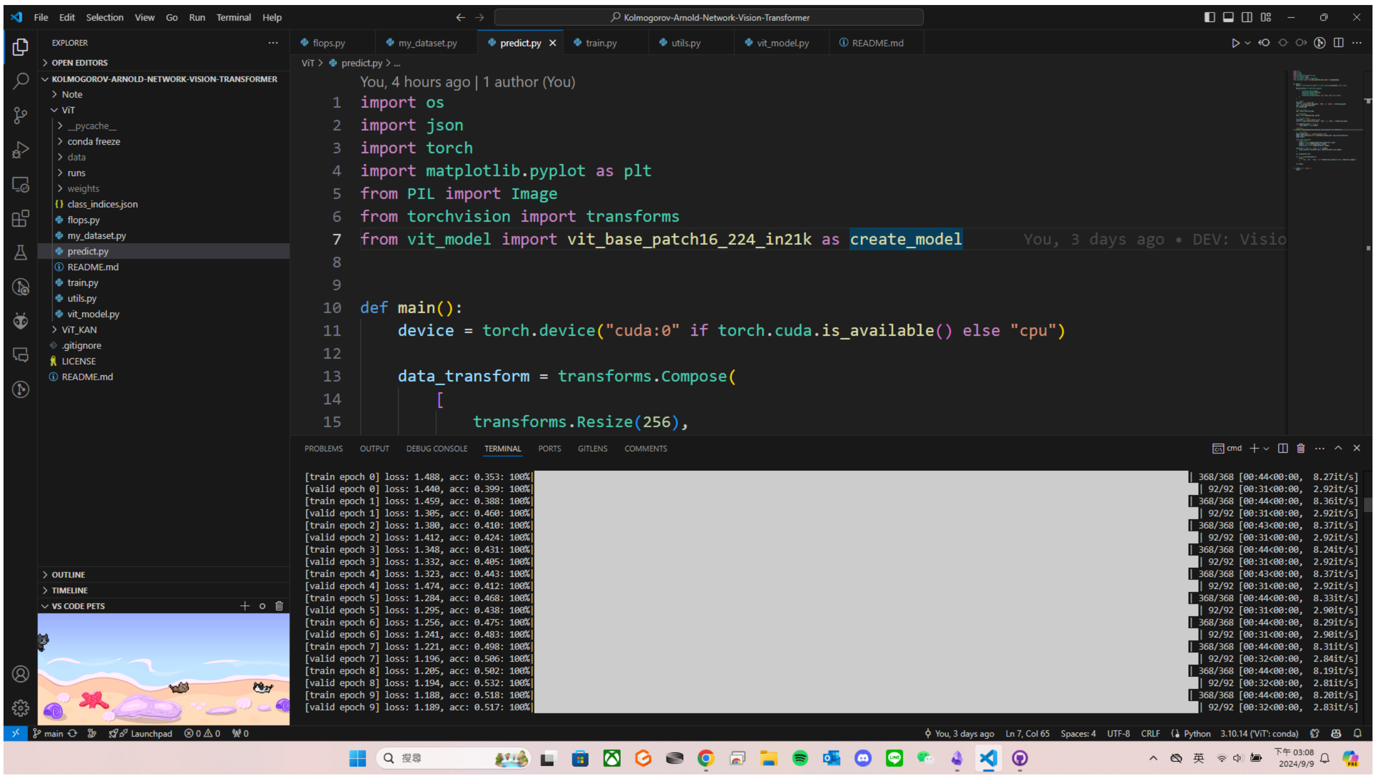
```python
    # ViT-Large model (ViT-L/32) ImageNet-21k weights @ 224x224
    model = VisionTransformer(
        img_size=224,
        patch_size=32,
        embed_dim=1024,
        depth=24,
        num_heads=16,
        representation_size=1024 if has_logits else None,
        num_classes=num_classes,
    )

    return model


def vit_huge_patch14_224_in21k(num_classes: int = 21843, has_logits: bool = True):
    # ViT-Huge model (ViT-H/14) ImageNet-21k weights @ 224x224
    model = VisionTransformer(
        img_size=224,
        patch_size=14,
        embed_dim=1280,
        depth=32,
        num_heads=16,
        representation_size=1280 if has_logits else None,
        num_classes=num_classes,
    )

    return model
```

Done

10 個 Epoch 的訓練結果（當前基準），無預訓練權重

Attention

沒有預訓練權重載入的情況下，基於論文所優化的 Learning Rate Scheduler 會導致學習率過小，需要嘗試把一開始給定的 lr 以及 lrf 調高進行驗證看看結果



```python
import os
import json
import torch
import matplotlib.pyplot as plt
from PIL import Image
from torchvision import transforms
from vit_model import vit_base_patch16_224_in21k as create_model


def main():
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

    data_transform = transforms.Compose(
        [
            transforms.Resize(256),
```

```
[train epoch 0] loss: 1.488, acc: 0.353: 100%|██████████| 368/368 [00:44<00:00,  8.27it/s]
[valid epoch 0] loss: 1.440, acc: 0.399: 100%|██████████| 92/92 [00:31<00:00,  2.92it/s]
[train epoch 1] loss: 1.459, acc: 0.388: 100%|██████████| 368/368 [00:44<00:00,  8.36it/s]
[valid epoch 1] loss: 1.305, acc: 0.460: 100%|██████████| 92/92 [00:31<00:00,  2.92it/s]
[train epoch 2] loss: 1.380, acc: 0.410: 100%|██████████| 368/368 [00:43<00:00,  8.37it/s]
[valid epoch 2] loss: 1.412, acc: 0.424: 100%|██████████| 92/92 [00:31<00:00,  2.92it/s]
[train epoch 3] loss: 1.348, acc: 0.431: 100%|██████████| 368/368 [00:44<00:00,  8.24it/s]
[valid epoch 3] loss: 1.332, acc: 0.405: 100%|██████████| 92/92 [00:31<00:00,  2.92it/s]
[train epoch 4] loss: 1.323, acc: 0.443: 100%|██████████| 368/368 [00:43<00:00,  8.37it/s]
[valid epoch 4] loss: 1.474, acc: 0.412: 100%|██████████| 92/92 [00:31<00:00,  2.92it/s]
[train epoch 5] loss: 1.284, acc: 0.468: 100%|██████████| 368/368 [00:44<00:00,  8.33it/s]
[valid epoch 5] loss: 1.295, acc: 0.438: 100%|██████████| 92/92 [00:31<00:00,  2.90it/s]
[train epoch 6] loss: 1.256, acc: 0.475: 100%|██████████| 368/368 [00:44<00:00,  8.29it/s]
[valid epoch 6] loss: 1.241, acc: 0.483: 100%|██████████| 92/92 [00:31<00:00,  2.90it/s]
[train epoch 7] loss: 1.221, acc: 0.498: 100%|██████████| 368/368 [00:44<00:00,  8.31it/s]
[valid epoch 7] loss: 1.196, acc: 0.506: 100%|██████████| 92/92 [00:32<00:00,  2.84it/s]
[train epoch 8] loss: 1.205, acc: 0.502: 100%|██████████| 368/368 [00:44<00:00,  8.19it/s]
[valid epoch 8] loss: 1.194, acc: 0.532: 100%|██████████| 92/92 [00:32<00:00,  2.81it/s]
[train epoch 9] loss: 1.188, acc: 0.518: 100%|██████████| 368/368 [00:44<00:00,  8.20it/s]
[valid epoch 9] loss: 1.189, acc: 0.517: 100%|██████████| 92/92 [00:32<00:00,  2.83it/s]
```

ViT KAN 替換

相關說明

Info

代碼使用流程介紹

Help

基本上沒有區別，就是把原本的使用 PyTorch 重構完的代碼把 MLP 的部分轉換為 KAN，詳細的差異會於下列說明

1. 下載好資料集（Dataset），目前代碼預設（Default）用的是花的分類（tf_flowers）：https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz
2. 在 `train.py` 中把 `--data-path` 設定為解壓縮後的 `flower_photos` 文件夾的絕對路徑。
3. 下載預訓練權重（Pre-Trained Weight），在 `vit_model.py` 中，每個模型都有提供預訓練權重的下載地址，根據選擇的模型下載對應的預訓練權重。
4. 在 `train.py` 腳本中將 `--weights` 參數設定為下載好的預訓練權重路徑。
5. 設定好資料集的路徑 `--data-path` 以及預訓練權重的路徑 `--weights`，即可使用 `train.py` 開始訓練（訓練過程中會自動生成 `class_indices.json` 文件）。
6. 在 `predict.py` 中導入與訓練代碼中相同的模型，並將 `model_weight_path` 設定為訓練好的模型權重路徑（預設保存在 `weights` 文件夾下）。
7. 在 `predict.py` 中將 `img_path` 設定為你需要預測的圖片的絕對路徑。
8. 設定好權重路徑 `model_weight_path` 和預測圖片路徑 `img_path`，即可使用 `predict.py` 進行預測。
9. 如果要使用其他資料集，就比照預設採用的「花的分類」之文件結構進行擺放（即一個類別對應一個文件夾），並且將訓練及預測代碼中的 `num_classes` 設定為你自己的數據的類別數。

Quote

原先設計的 Kolmogorov-Arnold Network 雖然有進行部分優化，但在訓練速度上還是令人捉急，為了比較方便地進行比對，改為使用「柴比雪夫多項式」（Chebyshev Polynomials） 取代原先的「B 樣條曲線」（B-Splines）進行函數擬合

- 應用於代碼中的 ChebyKANLayer Class

```python
class ChebyKANLayer(nn.Module):
    def __init__(self, input_dim, output_dim, degree):
        super(ChebyKANLayer, self).__init__()
        self.inputdim = input_dim
        self.outdim = output_dim
        self.degree = degree

        self.cheby_coeffs = nn.Parameter(torch.empty(input_dim, output_dim, degree + 1))
        nn.init.normal_(self.cheby_coeffs, mean=0.0, std=1 / (input_dim * (degree + 1)))
        self.register_buffer("arange", torch.arange(0, degree + 1, 1))

    def forward(self, x):
        # x 形狀: (batch_size, seq_len, inputdim)
        batch_size, seq_len, _ = x.shape

        # 將 x 展平以便與切比雪夫多項式一起使用: (batch_size * seq_len, inputdim)
        x = x.view(-1, self.inputdim)

        # 使用 tanh 將輸入歸一化到 [-1, 1]
        x = torch.tanh(x)

        # 重新塑形並重複輸入 degree + 1 次
        x = x.view((-1, self.inputdim, 1)).expand(-1, -1, self.degree + 1)

        # 應用 acos 並乘以 arange [0 .. degree]
        x = x.acos() * self.arange

        # 應用 cos 以獲取切比雪夫多項式值
        x = x.cos()

        # 計算切比雪夫插值
        y = torch.einsum("bid,iod->bo", x, self.cheby_coeffs)

        # 重新塑形回原始序列格式
        y = y.view(batch_size, seq_len, self.outdim)

        return y
```

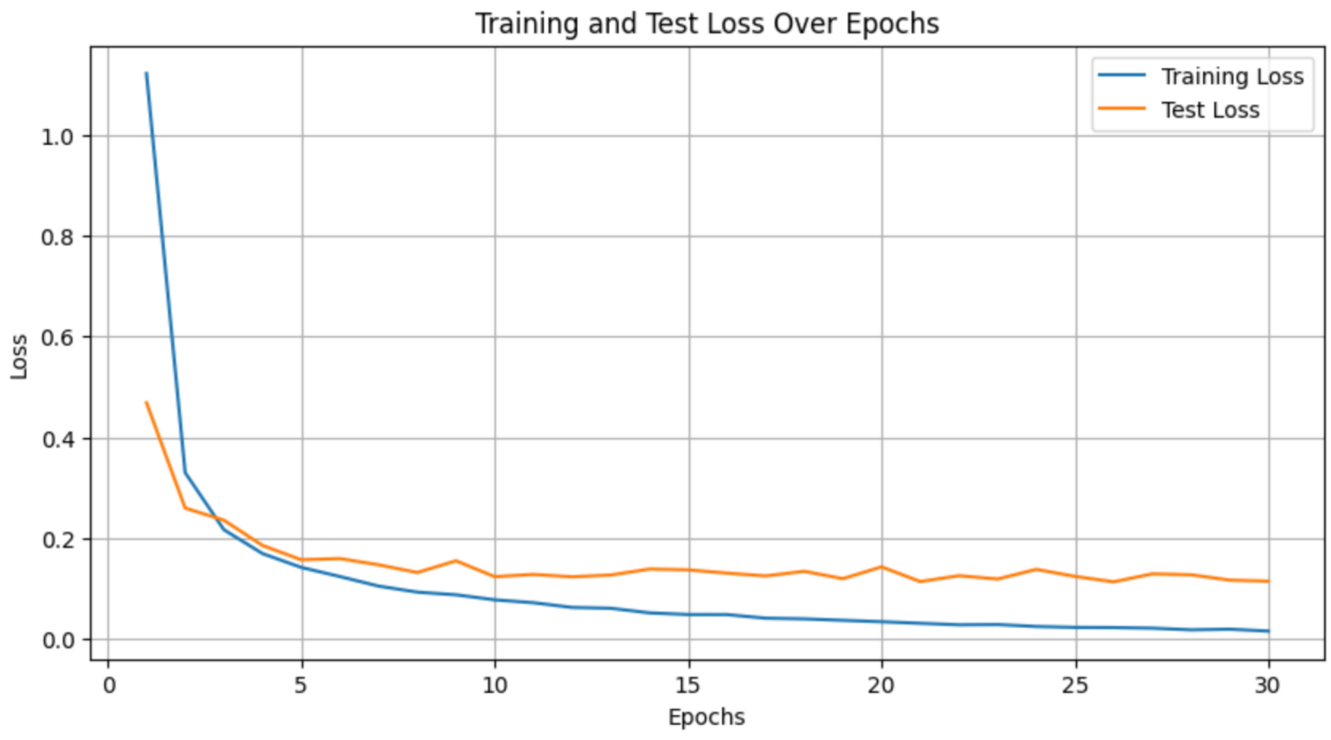
- 修改後的 Block 設計

```
class Block(nn.Module):
    def __init__(
        self,
        dim,
        num_heads,
        mlp_ratio=4.0,
        qkv_bias=False,
        qk_scale=None,
        drop_ratio=0.0,
        attn_drop_ratio=0.0,
        drop_path_ratio=0.0,
        act_layer=nn.GELU,
        norm_layer=nn.LayerNorm,
        degree=3,  # 添加 degree 作為 ChebyKAN 的參數
    ):
        super(Block, self).__init__()
        self.norm1 = norm_layer(dim)
        self.attn = Attention(
            dim,
            num_heads=num_heads,
            qkv_bias=qkv_bias,
            qk_scale=qk_scale,
            attn_drop_ratio=attn_drop_ratio,
            proj_drop_ratio=drop_ratio,
        )
        self.drop_path = (
            DropPath(drop_path_ratio) if drop_path_ratio > 0.0 else nn.Identity()
        )
        self.norm2 = norm_layer(dim)
        mlp_hidden_dim = int(dim * mlp_ratio)

        # 使用兩層 ChebyKANLayer 模擬 MLP 結構
        self.chebykan1 = ChebyKANLayer(
            input_dim=dim, output_dim=mlp_hidden_dim, degree=degree
        )
        self.chebykan2 = ChebyKANLayer(
            input_dim=mlp_hidden_dim, output_dim=dim, degree=degree
        )

    def forward(self, x):
        x = x + self.drop_path(self.attn(self.norm1(x)))
        x = x + self.drop_path(self.chebykan2(self.chebykan1(self.norm2(x))))
        return x
```

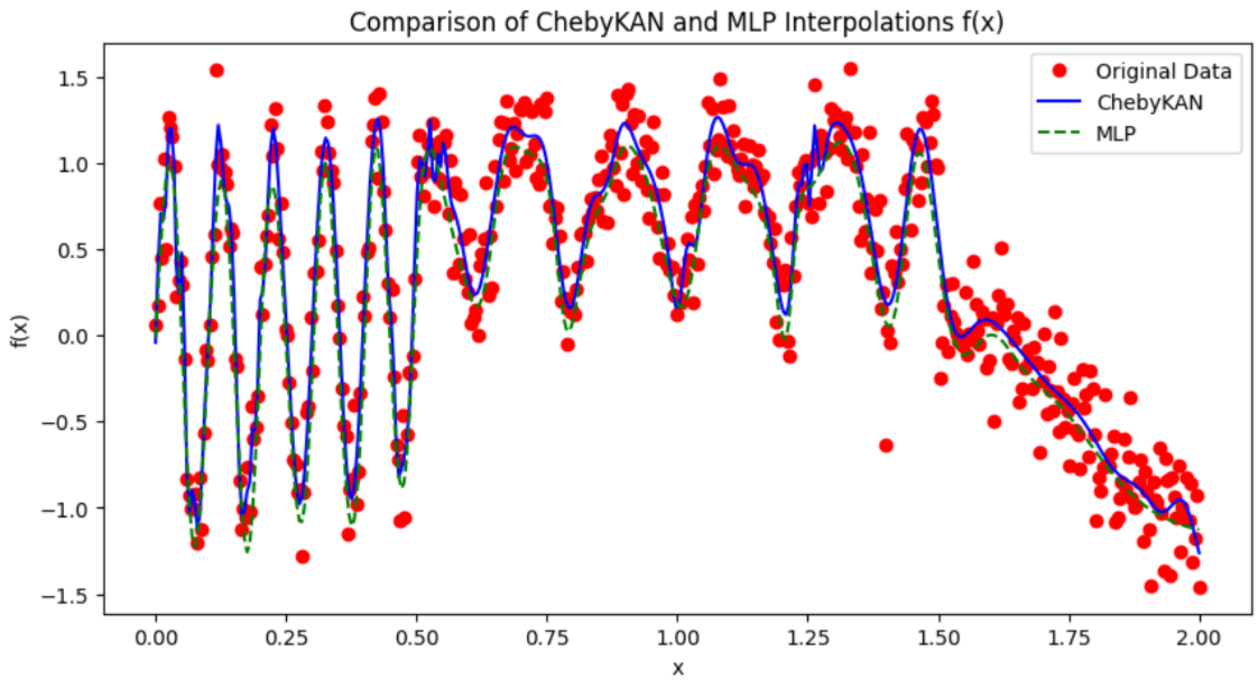
Note

下列為可行性驗證的各式檢驗圖表

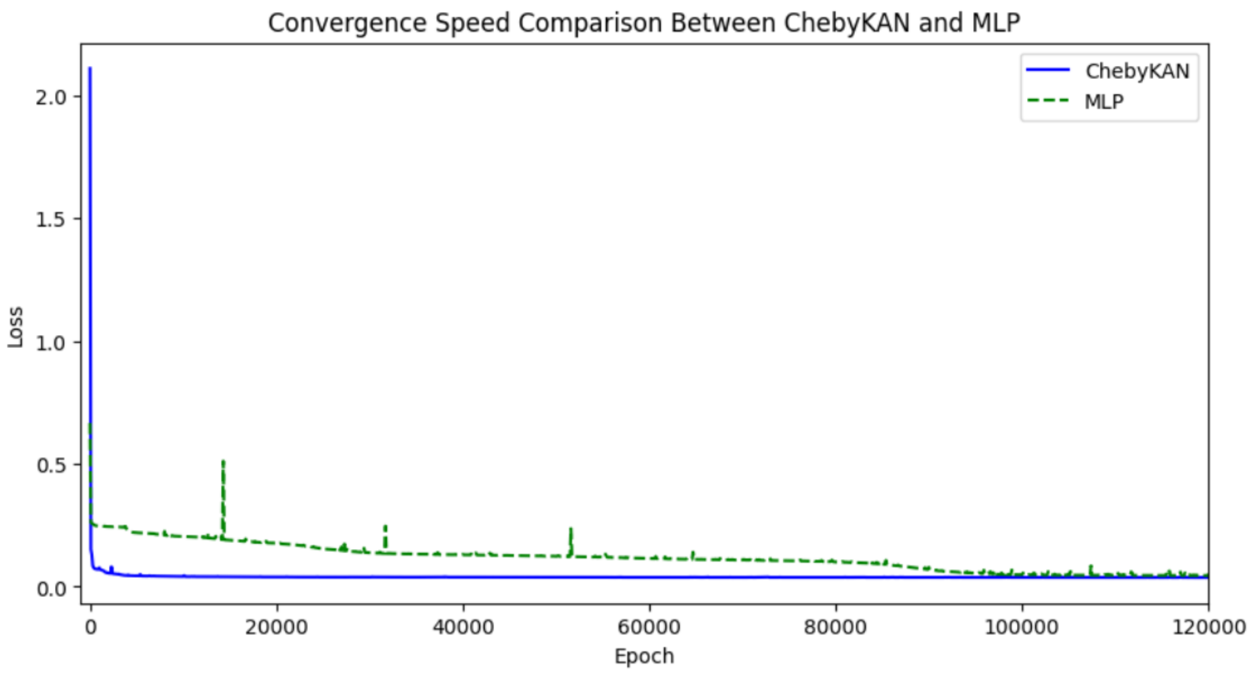
- ChebyKAN Training & Test Loss Over Epochs



- ChebyKAN Comparison with MLP Interpolations



- Covergence Speed Comparison Between ChebyKAN & MLP



代碼實現

Important

經過調整後的 `vit_model.py`

```python
import torch
import torch.nn as nn
from functools import partial
from collections import OrderedDict


class ChebyKANLayer(nn.Module):
    def __init__(self, input_dim, output_dim, degree):
        super(ChebyKANLayer, self).__init__()
        self.inputdim = input_dim
        self.outdim = output_dim
        self.degree = degree

        self.cheby_coeffs = nn.Parameter(torch.empty(input_dim, output_dim, degree + 1))
        nn.init.normal_(self.cheby_coeffs, mean=0.0, std=1 / (input_dim * (degree + 1)))
        self.register_buffer("arange", torch.arange(0, degree + 1, 1))

    def forward(self, x):
        # x 形狀: (batch_size, seq_len, inputdim)
        batch_size, seq_len, _ = x.shape

        # 將 x 展平以便與切比雪夫多項式一起使用: (batch_size * seq_len, inputdim)
        x = x.view(-1, self.inputdim)

        # 使用 tanh 將輸入歸一化到 [-1, 1]
        x = torch.tanh(x)

        # 重新塑形並重複輸入 degree + 1 次
        x = x.view((-1, self.inputdim, 1)).expand(-1, -1, self.degree + 1)

        # 應用 acos 並乘以 arange [0 .. degree]
        x = x.acos() * self.arange

        # 應用 cos 以獲取切比雪夫多項式值
        x = x.cos()

        # 計算切比雪夫插值
        y = torch.einsum("bid,iod->bo", x, self.cheby_coeffs)
        # 重新塑形回原始序列格式
        y = y.view(batch_size, seq_len, self.outdim)

        return y


def drop_path(x, drop_prob: float = 0.0, training: bool = False):
    if drop_prob == 0.0 or not training:
        return x

    keep_prob = 1 - drop_prob
    shape = (x.shape[0],) + (1,) * (
        x.ndim - 1
    )  # 適用於不同維度的張量，而不僅僅是 2D 卷積網絡
    random_tensor = keep_prob + torch.rand(shape, dtype=x.dtype, device=x.device)
    random_tensor.floor_()  # 二值化
    output = x.div(keep_prob) * random_tensor

    return output


# 每個樣本的 drop path (隨機深度，應用於殘差塊的主路徑時)
class DropPath(nn.Module):
    def __init__(self, drop_prob=None):
        super(DropPath, self).__init__()
        self.drop_prob = drop_prob

    def forward(self, x):
        return drop_path(x, self.drop_prob, self.training)


# 2D 影像 Patch Embedding
class PatchEmbed(nn.Module):
    def __init__(
        self, img_size=224, patch_size=16, in_c=3, embed_dim=768, norm_layer=None
    ):
        super().__init__()
        img_size = (img_size, img_size)
        patch_size = (patch_size, patch_size)
        self.img_size = img_size
        self.patch_size = patch_size
        self.grid_size = (img_size[0] // patch_size[0], img_size[1] // patch_size[1])
        self.num_patches = self.grid_size[0] * self.grid_size[1]

        self.proj = nn.Conv2d(
            in_c, embed_dim, kernel_size=patch_size, stride=patch_size
        )
        self.norm = norm_layer(embed_dim) if norm_layer else nn.Identity()

    def forward(self, x):
        B, C, H, W = x.shape
        assert (
            H == self.img_size[0] and W == self.img_size[1]
        ), f"輸入圖像大小 ({H}*{W}) 與模型大小 ({self.img_size[0]}*{self.img_size[1]}) 不匹配。"

        # flatten: [B, C, H, W] -> [B, C, HW]
        # transpose: [B, C, HW] -> [B, HW, C]
        x = self.proj(x).flatten(2).transpose(1, 2)
        x = self.norm(x)
        return x


class Attention(nn.Module):
    def __init__(
        self,
        dim,  # 輸入 token 的維度
        num_heads=8,
        qkv_bias=False,
        qk_scale=None,
        attn_drop_ratio=0.0,
        proj_drop_ratio=0.0,
    ):
        super(Attention, self).__init__()
        self.num_heads = num_heads
        head_dim = dim // num_heads
        self.scale = qk_scale or head_dim**-0.5
        self.qkv = nn.Linear(dim, dim * 3, bias=qkv_bias)
        self.attn_drop = nn.Dropout(attn_drop_ratio)
        self.proj = nn.Linear(dim, dim)
        self.proj_drop = nn.Dropout(proj_drop_ratio)

    def forward(self, x):
        # [batch_size, num_patches + 1, total_embed_dim]
        B, N, C = x.shape

        # qkv(): -> [batch_size, num_patches + 1, 3 * total_embed_dim]
        # reshape: -> [batch_size, num_patches + 1, 3, num_heads, embed_dim_per_head]
        # permute: -> [3, batch_size, num_heads, num_patches + 1, embed_dim_per_head]
        qkv = (
            self.qkv(x)
            .reshape(B, N, 3, self.num_heads, C // self.num_heads)
            .permute(2, 0, 3, 1, 4)
        )
        # [batch_size, num_heads, num_patches + 1, embed_dim_per_head]
        q, k, v = (
            qkv[0],
            qkv[1],
            qkv[2],
        )  # make torchscript happy (cannot use tensor as tuple)

        # transpose: -> [batch_size, num_heads, embed_dim_per_head, num_patches + 1]
        # @: multiply -> [batch_size, num_heads, num_patches + 1, num_patches + 1]
        attn = (q @ k.transpose(-2, -1)) * self.scale
        attn = attn.softmax(dim=-1)
        attn = self.attn_drop(attn)

        # @: multiply -> [batch_size, num_heads, num_patches + 1, embed_dim_per_head]
        # transpose: -> [batch_size, num_patches + 1, num_heads, embed_dim_per_head]
        # reshape: -> [batch_size, num_patches + 1, total_embed_dim]
        x = (attn @ v).transpose(1, 2).reshape(B, N, C)
        x = self.proj(x)
        x = self.proj_drop(x)
        return x


# Vision Transformer 使用的 MLP
class Mlp(nn.Module):
    def __init__(
        self,
        in_features,
        hidden_features=None,
        out_features=None,
        act_layer=nn.GELU,
        drop=0.0,
    ):
        super().__init__()
        out_features = out_features or in_features
        hidden_features = hidden_features or in_features
        self.fc1 = nn.Linear(in_features, hidden_features)
        self.act = act_layer()
        self.fc2 = nn.Linear(hidden_features, out_features)
        self.drop = nn.Dropout(drop)

    def forward(self, x):
        x = self.fc1(x)
        x = self.act(x)
        x = self.drop(x)
        x = self.fc2(x)
        x = self.drop(x)
        return x
```

```python
class Block(nn.Module):
    def __init__(
        self,
        dim,
        num_heads,
        mlp_ratio=4.0,
        qkv_bias=False,
        qk_scale=None,
        drop_ratio=0.0,
        attn_drop_ratio=0.0,
        drop_path_ratio=0.0,
        act_layer=nn.GELU,
        norm_layer=nn.LayerNorm,
        degree=3,  # 添加 degree 作為 ChebyKAN 的參數
    ):
        super(Block, self).__init__()
        self.norm1 = norm_layer(dim)
        self.attn = Attention(
            dim,
            num_heads=num_heads,
            qkv_bias=qkv_bias,
            qk_scale=qk_scale,
            attn_drop_ratio=attn_drop_ratio,
            proj_drop_ratio=drop_ratio,
        )
        self.drop_path = (
            DropPath(drop_path_ratio) if drop_path_ratio > 0.0 else nn.Identity()
        )
        self.norm2 = norm_layer(dim)
        mlp_hidden_dim = int(dim * mlp_ratio)

        # 使用兩層 ChebyKANLayer 模擬 MLP 結構
        self.chebykan1 = ChebyKANLayer(
            input_dim=dim, output_dim=mlp_hidden_dim, degree=degree
        )
        self.chebykan2 = ChebyKANLayer(
            input_dim=mlp_hidden_dim, output_dim=dim, degree=degree
        )

    def forward(self, x):
        x = x + self.drop_path(self.attn(self.norm1(x)))
        x = x + self.drop_path(self.chebykan2(self.chebykan1(self.norm2(x))))
        return x


class VisionTransformer(nn.Module):
    def __init__(
        self,
        img_size=224,
        patch_size=16,
        in_c=3,
        num_classes=1000,
        embed_dim=768,
        depth=12,
        num_heads=12,
        mlp_ratio=4.0,
        qkv_bias=True,
        qk_scale=None,
        representation_size=None,
        distilled=False,
        drop_ratio=0.0,
        attn_drop_ratio=0.0,
        drop_path_ratio=0.0,
        embed_layer=PatchEmbed,
        norm_layer=None,
        act_layer=None,
    ):
        """
        Args:
            img_size (int, tuple): input image size
            patch_size (int, tuple): patch size
            in_c (int): number of input channels
            num_classes (int): number of classes for classification head
            embed_dim (int): embedding dimension
            depth (int): depth of transformer
            num_heads (int): number of attention heads
            mlp_ratio (int): ratio of mlp hidden dim to embedding dim
            qkv_bias (bool): enable bias for qkv if True
            qk_scale (float): override default qk scale of head_dim ** -0.5 if set
            representation_size (Optional[int]): enable and set representation layer (pre-logits) to this value
 if set
            distilled (bool): model includes a distillation token and head as in DeiT models
            drop_ratio (float): dropout rate
            attn_drop_ratio (float): attention dropout rate
            drop_path_ratio (float): stochastic depth rate
            embed_layer (nn.Module): patch embedding layer
            norm_layer: (nn.Module): normalization layer
        """
        super(VisionTransformer, self).__init__()
        self.num_classes = num_classes
        self.num_features = self.embed_dim = (
            embed_dim  # 為了與其他模型一致，使用 num_features
        )
        self.num_tokens = 2 if distilled else 1
        norm_layer = norm_layer or partial(nn.LayerNorm, eps=1e-6)
        act_layer = act_layer or nn.GELU

        self.patch_embed = embed_layer(
            img_size=img_size, patch_size=patch_size, in_c=in_c, embed_dim=embed_dim
        )
        num_patches = self.patch_embed.num_patches

        self.cls_token = nn.Parameter(torch.zeros(1, 1, embed_dim))
        self.dist_token = (
            nn.Parameter(torch.zeros(1, 1, embed_dim)) if distilled else None
        )
        self.pos_embed = nn.Parameter(
            torch.zeros(1, num_patches + self.num_tokens, embed_dim)
        )
        self.pos_drop = nn.Dropout(p=drop_ratio)

        dpr = [
            x.item() for x in torch.linspace(0, drop_path_ratio, depth)
        ]  # 隨機深度衰減規則
        self.blocks = nn.Sequential(
            *[
                Block(
                    dim=embed_dim,
                    num_heads=num_heads,
                    mlp_ratio=mlp_ratio,
                    qkv_bias=qkv_bias,
                    qk_scale=qk_scale,
                    drop_ratio=drop_ratio,
                    attn_drop_ratio=attn_drop_ratio,
                    drop_path_ratio=dpr[i],
                    norm_layer=norm_layer,
                    act_layer=act_layer,
                )
                for i in range(depth)
            ]
        )
        self.norm = norm_layer(embed_dim)

        # 表示層
        if representation_size and not distilled:
            self.has_logits = True
            self.num_features = representation_size
            self.pre_logits = nn.Sequential(
                OrderedDict(
                    [
                        ("fc", nn.Linear(embed_dim, representation_size)),
```

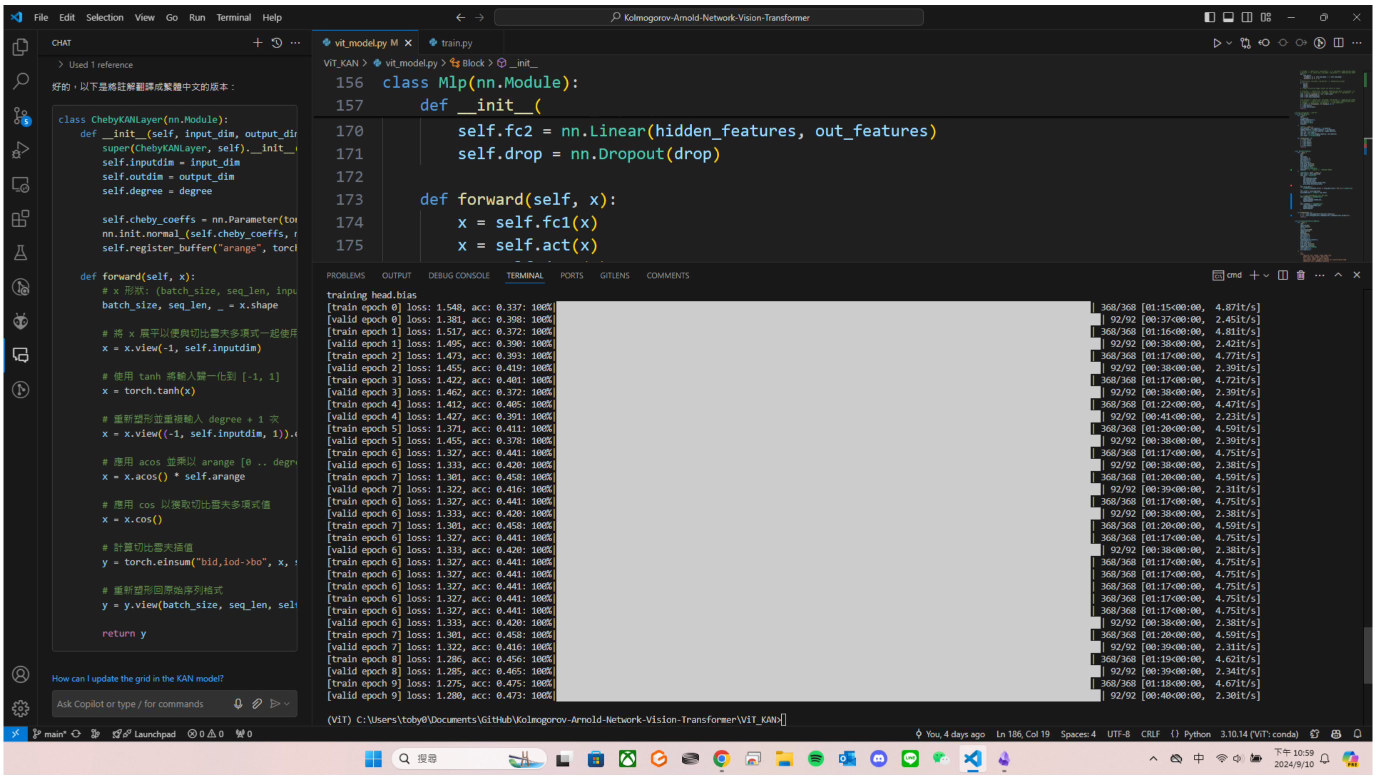
```python
                        ("act", nn.Tanh()),
                    ]
                )
            )
        else:
            self.has_logits = False
            self.pre_logits = nn.Identity()

        # 分類頭
        self.head = (
            nn.Linear(self.num_features, num_classes)
            if num_classes > 0
            else nn.Identity()
        )
        self.head_dist = None
        if distilled:
            self.head_dist = (
                nn.Linear(self.embed_dim, self.num_classes)
                if num_classes > 0
                else nn.Identity()
            )

        # 權重初始化
        nn.init.trunc_normal_(self.pos_embed, std=0.02)
        if self.dist_token is not None:
            nn.init.trunc_normal_(self.dist_token, std=0.02)

        nn.init.trunc_normal_(self.cls_token, std=0.02)
        self.apply(_init_vit_weights)

    def forward_features(self, x):
        # [B, C, H, W] → [B, num_patches, embed_dim]
        x = self.patch_embed(x)  # [B, 196, 768]
        # [1, 1, 768] → [B, 1, 768]
        cls_token = self.cls_token.expand(x.shape[0], -1, -1)
        if self.dist_token is None:
            x = torch.cat((cls_token, x), dim=1)  # [B, 197, 768]
        else:
            x = torch.cat(
                (cls_token, self.dist_token.expand(x.shape[0], -1, -1), x), dim=1
            )

        x = self.pos_drop(x + self.pos_embed)
        x = self.blocks(x)
        x = self.norm(x)
        if self.dist_token is None:
            return self.pre_logits(x[:, 0])
        else:
            return x[:, 0], x[:, 1]

    def forward(self, x):
        x = self.forward_features(x)
        if self.head_dist is not None:
            x, x_dist = self.head(x[0]), self.head_dist(x[1])
            if self.training and not torch.jit.is_scripting():
                # 推理期間，返回兩個分類器預測的平均值
                return x, x_dist
            else:
                return (x + x_dist) / 2
        else:
            x = self.head(x)
        return x


# ViT 權重初始化
def _init_vit_weights(m):
    # parameter m = module
    if isinstance(m, nn.Linear):
        nn.init.trunc_normal_(m.weight, std=0.01)
        if m.bias is not None:
            nn.init.zeros_(m.bias)
    elif isinstance(m, nn.Conv2d):
        nn.init.kaiming_normal_(m.weight, mode="fan_out")
        if m.bias is not None:
            nn.init.zeros_(m.bias)
    elif isinstance(m, nn.LayerNorm):
        nn.init.zeros_(m.bias)
        nn.init.ones_(m.weight)


def vit_base_patch16_224(num_classes: int = 1000):
    # ViT-Base model (ViT-B/16) ImageNet-1k weights @ 224x224
    model = VisionTransformer(
        img_size=224,
        patch_size=16,
        embed_dim=768,
        depth=12,
        num_heads=12,
        representation_size=None,
        num_classes=num_classes,
    )

    return model


def vit_base_patch16_224_in21k(num_classes: int = 21843, has_logits: bool = True):
    # ViT-Base model (ViT-B/16) ImageNet-21k weights @ 224x224
    model = VisionTransformer(
        img_size=224,
        patch_size=16,
        embed_dim=768,
        depth=12,
        num_heads=12,
        representation_size=768 if has_logits else None,
        num_classes=num_classes,
    )

    return model


def vit_base_patch32_224(num_classes: int = 1000):
    # ViT-Base model (ViT-B/32) ImageNet-1k weights @ 224x224
    model = VisionTransformer(
        img_size=224,
        patch_size=32,
        embed_dim=768,
        depth=12,
        num_heads=12,
        representation_size=None,
        num_classes=num_classes,
    )

    return model


def vit_base_patch32_224_in21k(num_classes: int = 21843, has_logits: bool = True):
    # ViT-Base model (ViT-B/32) ImageNet-21k weights @ 224x224
    model = VisionTransformer(
        img_size=224,
        patch_size=32,
        embed_dim=768,
        depth=12,
        num_heads=12,
        representation_size=768 if has_logits else None,
        num_classes=num_classes,
    )

    return model


def vit_large_patch16_224(num_classes: int = 1000):
    # ViT-Large model (ViT-L/16) ImageNet-1k weights @ 224x224
    model = VisionTransformer(
        img_size=224,
        patch_size=16,
        embed_dim=1024,
        depth=24,
        num_heads=16,
        representation_size=None,
        num_classes=num_classes,
    )

    return model


def vit_large_patch16_224_in21k(num_classes: int = 21843, has_logits: bool = True):
    # ViT-Large model (ViT-L/16) ImageNet-21k weights @ 224x224
    model = VisionTransformer(
        img_size=224,
        patch_size=16,
        embed_dim=1024,
        depth=24,
        num_heads=16,
        representation_size=1024 if has_logits else None,
        num_classes=num_classes,
    )

    return model


def vit_large_patch32_224_in21k(num_classes: int = 21843, has_logits: bool = True):
    # ViT-Large model (ViT-L/32) ImageNet-21k weights @ 224x224
    model = VisionTransformer(
        img_size=224,
        patch_size=32,
        embed_dim=1024,
        depth=24,
        num_heads=16,
        representation_size=1024 if has_logits else None,
        num_classes=num_classes,
    )

    return model


def vit_huge_patch14_224_in21k(num_classes: int = 21843, has_logits: bool = True):
    # ViT-Huge model (ViT-H/14) ImageNet-21k weights @ 224x224
    model = VisionTransformer(
        img_size=224,
        patch_size=14,
        embed_dim=1280,
        depth=32,
        num_heads=16,
        representation_size=1280 if has_logits else None,
        num_classes=num_classes,
    )

    return model
```

✓ Done

10 個 Epoch 的訓練結果（當前基準），無預訓練權重

⚠ Attention

Learning Rate 過小的問題在應用 Kolmogorov-Arnold Network 後更為明顯，因為沒有足夠的學習量能讓函數進行擬合，如純 PyTorch 實現一樣需要重新調整



Used 1 reference

好的，以下是將註解翻譯成繁體中文的版本：

```
class ChebyKANLayer(nn.Module):
    def __init__(self, input_dim, output_dir
        super(ChebyKANLayer, self).__init__(
        self.inputdim = input_dim
        self.outdim = output_dim
        self.degree = degree

        self.cheby_coeffs = nn.Parameter(tor
        nn.init.normal_(self.cheby_coeffs, r
        self.register_buffer("arange", torch

    def forward(self, x):
        # x 形狀: (batch_size, seq_len, inpu
        batch_size, seq_len, _ = x.shape

        # 將 x 展平以便與切比雪夫多項式一起使用
        x = x.view(-1, self.inputdim)

        # 使用 tanh 將輸入歸一化到 [-1, 1]
        x = torch.tanh(x)

        # 重新塑形並重複輸入 degree + 1 次
        x = x.view((-1, self.inputdim, 1)).e

        # 應用 acos 並乘以 arange [0 .. degr
        x = x.acos() * self.arange

        # 應用 cos 以獲取切比雪夫多項式值
        x = x.cos()

        # 計算切比雪夫插值
        y = torch.einsum("bid,iod->bo", x, s

        # 重新塑形回原始序列格式
        y = y.view(batch_size, seq_len, self

        return y
```

How can I update the grid in the KAN model?

Ask Copilot or type / for commands

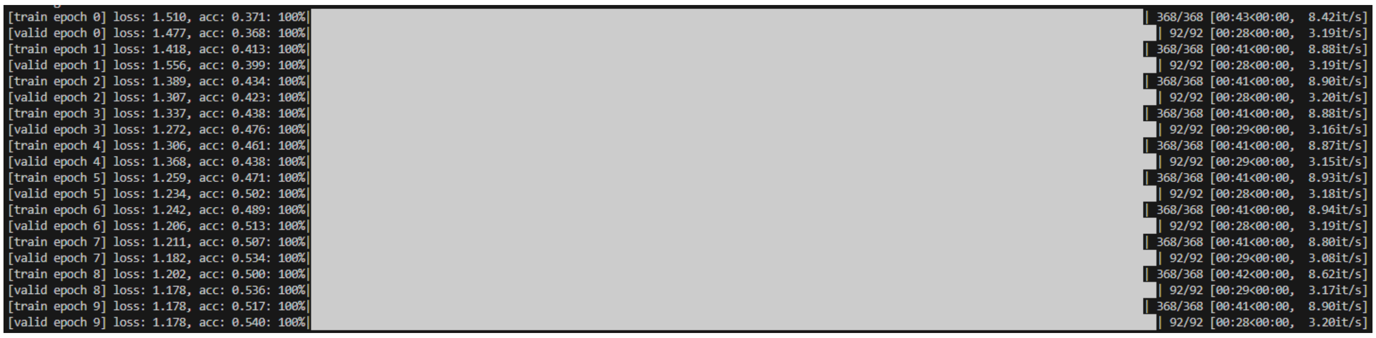
```
class Mlp(nn.Module):
    def __init__(
        self.fc2 = nn.Linear(hidden_features, out_features)
        self.drop = nn.Dropout(drop)

    def forward(self, x):
        x = self.fc1(x)
        x = self.act(x)
```

```
training head.bias
[train epoch 0] loss: 1.548, acc: 0.337: 100%|██████████| 368/368 [01:15<00:00,  4.87it/s]
[valid epoch 0] loss: 1.381, acc: 0.398: 100%|██████████| 92/92 [00:37<00:00,  2.45it/s]
[train epoch 1] loss: 1.517, acc: 0.372: 100%|██████████| 368/368 [01:16<00:00,  4.81it/s]
[valid epoch 1] loss: 1.495, acc: 0.390: 100%|██████████| 92/92 [00:38<00:00,  2.42it/s]
[train epoch 2] loss: 1.473, acc: 0.393: 100%|██████████| 368/368 [01:17<00:00,  4.77it/s]
[valid epoch 2] loss: 1.455, acc: 0.419: 100%|██████████| 92/92 [00:38<00:00,  2.39it/s]
[train epoch 3] loss: 1.422, acc: 0.401: 100%|██████████| 368/368 [01:17<00:00,  4.72it/s]
[valid epoch 3] loss: 1.462, acc: 0.372: 100%|██████████| 92/92 [00:38<00:00,  2.39it/s]
[train epoch 4] loss: 1.412, acc: 0.405: 100%|██████████| 368/368 [01:22<00:00,  4.47it/s]
[valid epoch 4] loss: 1.427, acc: 0.391: 100%|██████████| 92/92 [00:41<00:00,  2.23it/s]
[train epoch 5] loss: 1.371, acc: 0.411: 100%|██████████| 368/368 [01:20<00:00,  4.59it/s]
[valid epoch 5] loss: 1.455, acc: 0.378: 100%|██████████| 92/92 [00:38<00:00,  2.39it/s]
[train epoch 6] loss: 1.327, acc: 0.441: 100%|██████████| 368/368 [01:17<00:00,  4.75it/s]
[valid epoch 6] loss: 1.333, acc: 0.420: 100%|██████████| 92/92 [00:38<00:00,  2.38it/s]
[train epoch 7] loss: 1.301, acc: 0.458: 100%|██████████| 368/368 [01:20<00:00,  4.59it/s]
[valid epoch 7] loss: 1.322, acc: 0.416: 100%|██████████| 92/92 [00:39<00:00,  2.31it/s]
[train epoch 6] loss: 1.327, acc: 0.441: 100%|██████████| 368/368 [01:17<00:00,  4.75it/s]
[valid epoch 6] loss: 1.333, acc: 0.420: 100%|██████████| 92/92 [00:38<00:00,  2.38it/s]
[train epoch 7] loss: 1.301, acc: 0.458: 100%|██████████| 368/368 [01:20<00:00,  4.59it/s]
[train epoch 6] loss: 1.327, acc: 0.441: 100%|██████████| 368/368 [01:17<00:00,  4.75it/s]
[valid epoch 6] loss: 1.333, acc: 0.420: 100%|██████████| 92/92 [00:38<00:00,  2.38it/s]
[train epoch 6] loss: 1.327, acc: 0.441: 100%|██████████| 368/368 [01:17<00:00,  4.75it/s]
[train epoch 6] loss: 1.327, acc: 0.441: 100%|██████████| 368/368 [01:17<00:00,  4.75it/s]
[train epoch 6] loss: 1.327, acc: 0.441: 100%|██████████| 368/368 [01:17<00:00,  4.75it/s]
[train epoch 6] loss: 1.327, acc: 0.441: 100%|██████████| 368/368 [01:17<00:00,  4.75it/s]
[train epoch 6] loss: 1.327, acc: 0.441: 100%|██████████| 368/368 [01:17<00:00,  4.75it/s]
[valid epoch 6] loss: 1.333, acc: 0.420: 100%|██████████| 92/92 [00:38<00:00,  2.38it/s]
[train epoch 7] loss: 1.301, acc: 0.458: 100%|██████████| 368/368 [01:20<00:00,  4.59it/s]
[valid epoch 7] loss: 1.322, acc: 0.416: 100%|██████████| 92/92 [00:39<00:00,  2.31it/s]
[train epoch 8] loss: 1.286, acc: 0.456: 100%|██████████| 368/368 [01:19<00:00,  4.62it/s]
[valid epoch 8] loss: 1.285, acc: 0.465: 100%|██████████| 92/92 [00:39<00:00,  2.34it/s]
[train epoch 9] loss: 1.275, acc: 0.475: 100%|██████████| 368/368 [01:18<00:00,  4.67it/s]
[valid epoch 9] loss: 1.280, acc: 0.473: 100%|██████████| 92/92 [00:40<00:00,  2.30it/s]

(ViT) C:\Users\toby0\Documents\GitHub\Kolmogorov-Arnold-Network-Vision-Transformer\ViT_KAN>
```

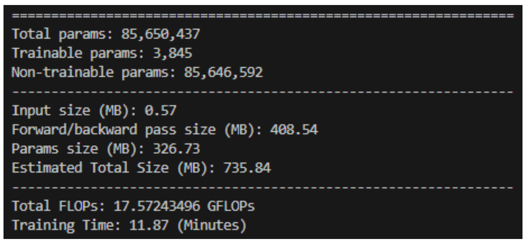
實驗結果

資料集 tf_flowers

- 原始 ViT-Base model (ViT-B/16)

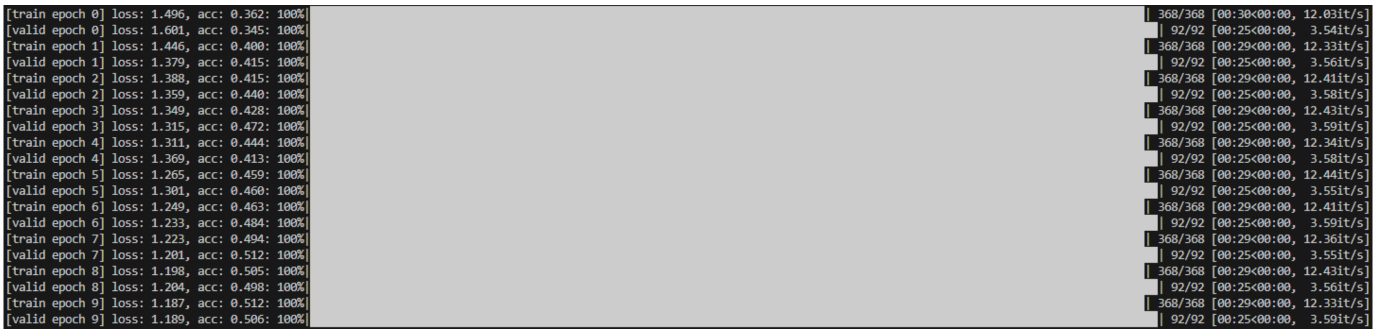


```
[train epoch 0] loss: 1.510, acc: 0.371: 100%|██████████| 368/368 [00:43<00:00,  8.42it/s]
[valid epoch 0] loss: 1.477, acc: 0.368: 100%|██████████| 92/92 [00:28<00:00,  3.19it/s]
[train epoch 1] loss: 1.418, acc: 0.413: 100%|██████████| 368/368 [00:41<00:00,  8.88it/s]
[valid epoch 1] loss: 1.556, acc: 0.399: 100%|██████████| 92/92 [00:28<00:00,  3.19it/s]
[train epoch 2] loss: 1.389, acc: 0.434: 100%|██████████| 368/368 [00:41<00:00,  8.90it/s]
[valid epoch 2] loss: 1.307, acc: 0.423: 100%|██████████| 92/92 [00:28<00:00,  3.20it/s]
[train epoch 3] loss: 1.337, acc: 0.438: 100%|██████████| 368/368 [00:41<00:00,  8.88it/s]
[valid epoch 3] loss: 1.272, acc: 0.476: 100%|██████████| 92/92 [00:29<00:00,  3.16it/s]
[train epoch 4] loss: 1.306, acc: 0.461: 100%|██████████| 368/368 [00:41<00:00,  8.87it/s]
[valid epoch 4] loss: 1.368, acc: 0.438: 100%|██████████| 92/92 [00:29<00:00,  3.15it/s]
[train epoch 5] loss: 1.259, acc: 0.471: 100%|██████████| 368/368 [00:41<00:00,  8.93it/s]
[valid epoch 5] loss: 1.234, acc: 0.502: 100%|██████████| 92/92 [00:28<00:00,  3.18it/s]
[train epoch 6] loss: 1.242, acc: 0.489: 100%|██████████| 368/368 [00:41<00:00,  8.94it/s]
[valid epoch 6] loss: 1.206, acc: 0.513: 100%|██████████| 92/92 [00:28<00:00,  3.19it/s]
[train epoch 7] loss: 1.211, acc: 0.507: 100%|██████████| 368/368 [00:41<00:00,  8.80it/s]
[valid epoch 7] loss: 1.182, acc: 0.534: 100%|██████████| 92/92 [00:29<00:00,  3.08it/s]
[train epoch 8] loss: 1.202, acc: 0.500: 100%|██████████| 368/368 [00:42<00:00,  8.62it/s]
[valid epoch 8] loss: 1.178, acc: 0.536: 100%|██████████| 92/92 [00:29<00:00,  3.17it/s]
[train epoch 9] loss: 1.178, acc: 0.517: 100%|██████████| 368/368 [00:41<00:00,  8.90it/s]
[valid epoch 9] loss: 1.178, acc: 0.540: 100%|██████████| 92/92 [00:28<00:00,  3.20it/s]
```

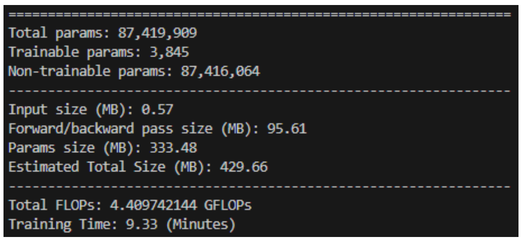


```
================================================================
Total params: 85,650,437
Trainable params: 3,845
Non-trainable params: 85,646,592
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 408.54
Params size (MB): 326.73
Estimated Total Size (MB): 735.84
----------------------------------------------------------------
Total FLOPs: 17.57243496 GFLOPs
Training Time: 11.87 (Minutes)
```

- 原始 ViT-Base model (ViT-B/32)

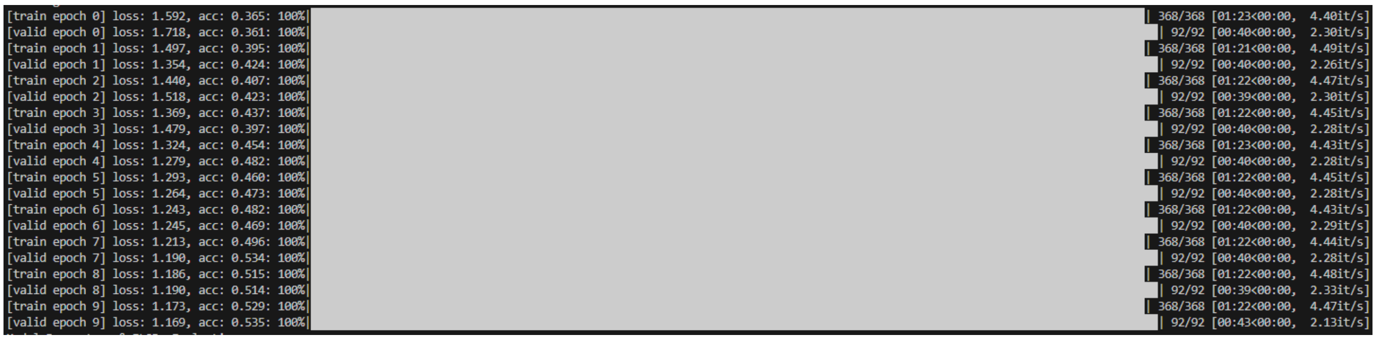


```
[train epoch 0] loss: 1.496, acc: 0.362: 100%|██████████| 368/368 [00:30<00:00, 12.03it/s]
[valid epoch 0] loss: 1.601, acc: 0.345: 100%|██████████| 92/92 [00:25<00:00,  3.54it/s]
[train epoch 1] loss: 1.446, acc: 0.400: 100%|██████████| 368/368 [00:29<00:00, 12.33it/s]
[valid epoch 1] loss: 1.379, acc: 0.415: 100%|██████████| 92/92 [00:25<00:00,  3.56it/s]
[train epoch 2] loss: 1.388, acc: 0.415: 100%|██████████| 368/368 [00:29<00:00, 12.41it/s]
[valid epoch 2] loss: 1.359, acc: 0.440: 100%|██████████| 92/92 [00:25<00:00,  3.58it/s]
[train epoch 3] loss: 1.349, acc: 0.428: 100%|██████████| 368/368 [00:29<00:00, 12.43it/s]
[valid epoch 3] loss: 1.315, acc: 0.472: 100%|██████████| 92/92 [00:25<00:00,  3.59it/s]
[train epoch 4] loss: 1.311, acc: 0.444: 100%|██████████| 368/368 [00:29<00:00, 12.34it/s]
[valid epoch 4] loss: 1.369, acc: 0.413: 100%|██████████| 92/92 [00:25<00:00,  3.58it/s]
[train epoch 5] loss: 1.265, acc: 0.459: 100%|██████████| 368/368 [00:29<00:00, 12.44it/s]
[valid epoch 5] loss: 1.301, acc: 0.460: 100%|██████████| 92/92 [00:25<00:00,  3.55it/s]
[train epoch 6] loss: 1.249, acc: 0.463: 100%|██████████| 368/368 [00:29<00:00, 12.41it/s]
[valid epoch 6] loss: 1.233, acc: 0.484: 100%|██████████| 92/92 [00:25<00:00,  3.59it/s]
[train epoch 7] loss: 1.223, acc: 0.494: 100%|██████████| 368/368 [00:29<00:00, 12.36it/s]
[valid epoch 7] loss: 1.201, acc: 0.512: 100%|██████████| 92/92 [00:25<00:00,  3.55it/s]
[train epoch 8] loss: 1.198, acc: 0.505: 100%|██████████| 368/368 [00:29<00:00, 12.43it/s]
[valid epoch 8] loss: 1.204, acc: 0.498: 100%|██████████| 92/92 [00:25<00:00,  3.56it/s]
[train epoch 9] loss: 1.187, acc: 0.512: 100%|██████████| 368/368 [00:29<00:00, 12.33it/s]
[valid epoch 9] loss: 1.189, acc: 0.506: 100%|██████████| 92/92 [00:25<00:00,  3.59it/s]
```

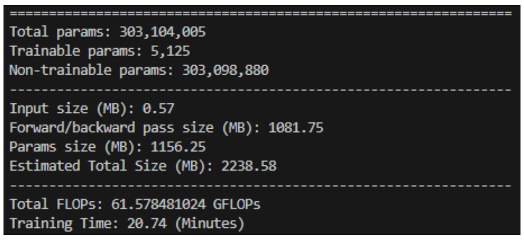


```
=================================================================
Total params: 87,419,909
Trainable params: 3,845
Non-trainable params: 87,416,064
-----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 95.61
Params size (MB): 333.48
Estimated Total Size (MB): 429.66
-----------------------------------------------------------------
Total FLOPs: 4.409742144 GFLOPs
Training Time: 9.33 (Minutes)
```

- 原始 ViT-Large model (ViT-L/16)

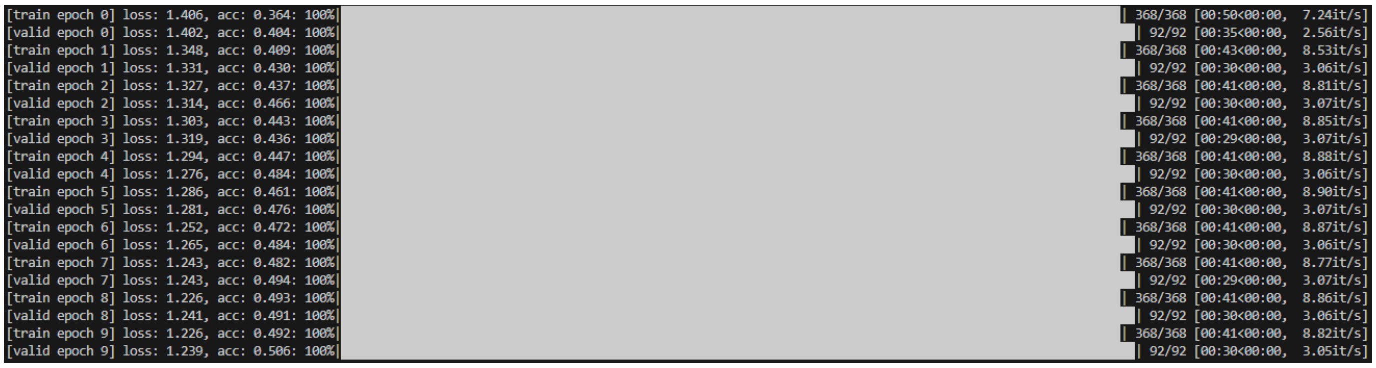


```
[train epoch 0] loss: 1.592, acc: 0.365: 100%|██████████| 368/368 [01:23<00:00,  4.40it/s]
[valid epoch 0] loss: 1.718, acc: 0.361: 100%|██████████| 92/92 [00:40<00:00,  2.30it/s]
[train epoch 1] loss: 1.497, acc: 0.395: 100%|██████████| 368/368 [01:21<00:00,  4.49it/s]
[valid epoch 1] loss: 1.354, acc: 0.424: 100%|██████████| 92/92 [00:40<00:00,  2.26it/s]
[train epoch 2] loss: 1.440, acc: 0.407: 100%|██████████| 368/368 [01:22<00:00,  4.47it/s]
[valid epoch 2] loss: 1.518, acc: 0.423: 100%|██████████| 92/92 [00:39<00:00,  2.30it/s]
[train epoch 3] loss: 1.369, acc: 0.437: 100%|██████████| 368/368 [01:22<00:00,  4.45it/s]
[valid epoch 3] loss: 1.479, acc: 0.397: 100%|██████████| 92/92 [00:40<00:00,  2.28it/s]
[train epoch 4] loss: 1.324, acc: 0.454: 100%|██████████| 368/368 [01:23<00:00,  4.43it/s]
[valid epoch 4] loss: 1.279, acc: 0.482: 100%|██████████| 92/92 [00:40<00:00,  2.28it/s]
[train epoch 5] loss: 1.293, acc: 0.460: 100%|██████████| 368/368 [01:22<00:00,  4.45it/s]
[valid epoch 5] loss: 1.264, acc: 0.473: 100%|██████████| 92/92 [00:40<00:00,  2.28it/s]
[train epoch 6] loss: 1.243, acc: 0.482: 100%|██████████| 368/368 [01:22<00:00,  4.43it/s]
[valid epoch 6] loss: 1.245, acc: 0.469: 100%|██████████| 92/92 [00:40<00:00,  2.29it/s]
[train epoch 7] loss: 1.213, acc: 0.496: 100%|██████████| 368/368 [01:22<00:00,  4.44it/s]
[valid epoch 7] loss: 1.190, acc: 0.534: 100%|██████████| 92/92 [00:40<00:00,  2.28it/s]
[train epoch 8] loss: 1.186, acc: 0.515: 100%|██████████| 368/368 [01:22<00:00,  4.48it/s]
[valid epoch 8] loss: 1.190, acc: 0.514: 100%|██████████| 92/92 [00:39<00:00,  2.33it/s]
[train epoch 9] loss: 1.173, acc: 0.529: 100%|██████████| 368/368 [01:22<00:00,  4.47it/s]
[valid epoch 9] loss: 1.169, acc: 0.535: 100%|██████████| 92/92 [00:43<00:00,  2.13it/s]
```

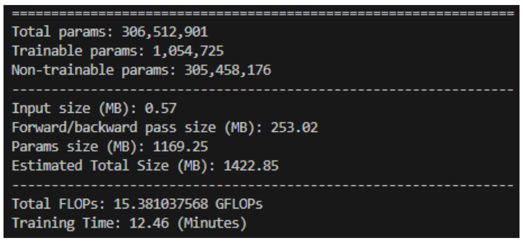


```
=================================================================
Total params: 303,104,005
Trainable params: 5,125
Non-trainable params: 303,098,880
-----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 1081.75
Params size (MB): 1156.25
Estimated Total Size (MB): 2238.58
-----------------------------------------------------------------
Total FLOPs: 61.578481024 GFLOPs
Training Time: 20.74 (Minutes)
```

- 原始 ViT-Large model (ViT-L/32)

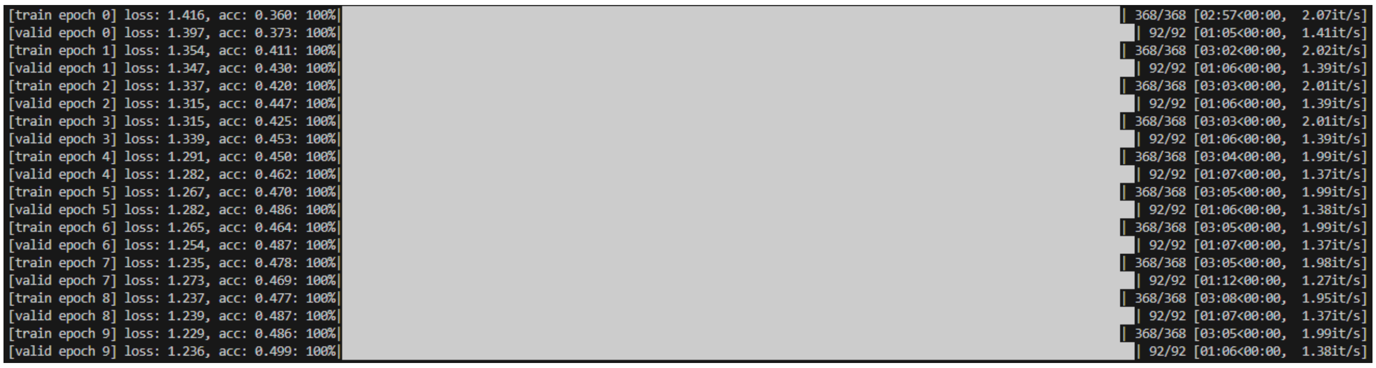


```
[train epoch 0] loss: 1.406, acc: 0.364: 100%|██████████| 368/368 [00:50<00:00,  7.24it/s]
[valid epoch 0] loss: 1.402, acc: 0.404: 100%|██████████| 92/92 [00:35<00:00,  2.56it/s]
[train epoch 1] loss: 1.348, acc: 0.409: 100%|██████████| 368/368 [00:43<00:00,  8.53it/s]
[valid epoch 1] loss: 1.331, acc: 0.430: 100%|██████████| 92/92 [00:30<00:00,  3.06it/s]
[train epoch 2] loss: 1.327, acc: 0.437: 100%|██████████| 368/368 [00:41<00:00,  8.81it/s]
[valid epoch 2] loss: 1.314, acc: 0.466: 100%|██████████| 92/92 [00:30<00:00,  3.07it/s]
[train epoch 3] loss: 1.303, acc: 0.443: 100%|██████████| 368/368 [00:41<00:00,  8.85it/s]
[valid epoch 3] loss: 1.319, acc: 0.436: 100%|██████████| 92/92 [00:29<00:00,  3.07it/s]
[train epoch 4] loss: 1.294, acc: 0.447: 100%|██████████| 368/368 [00:41<00:00,  8.88it/s]
[valid epoch 4] loss: 1.276, acc: 0.484: 100%|██████████| 92/92 [00:30<00:00,  3.06it/s]
[train epoch 5] loss: 1.286, acc: 0.461: 100%|██████████| 368/368 [00:41<00:00,  8.90it/s]
[valid epoch 5] loss: 1.281, acc: 0.476: 100%|██████████| 92/92 [00:30<00:00,  3.07it/s]
[train epoch 6] loss: 1.252, acc: 0.472: 100%|██████████| 368/368 [00:41<00:00,  8.87it/s]
[valid epoch 6] loss: 1.265, acc: 0.484: 100%|██████████| 92/92 [00:30<00:00,  3.06it/s]
[train epoch 7] loss: 1.243, acc: 0.482: 100%|██████████| 368/368 [00:41<00:00,  8.77it/s]
[valid epoch 7] loss: 1.243, acc: 0.494: 100%|██████████| 92/92 [00:29<00:00,  3.07it/s]
[train epoch 8] loss: 1.226, acc: 0.493: 100%|██████████| 368/368 [00:41<00:00,  8.86it/s]
[valid epoch 8] loss: 1.241, acc: 0.491: 100%|██████████| 92/92 [00:30<00:00,  3.06it/s]
[train epoch 9] loss: 1.226, acc: 0.492: 100%|██████████| 368/368 [00:41<00:00,  8.82it/s]
[valid epoch 9] loss: 1.239, acc: 0.506: 100%|██████████| 92/92 [00:30<00:00,  3.05it/s]
```

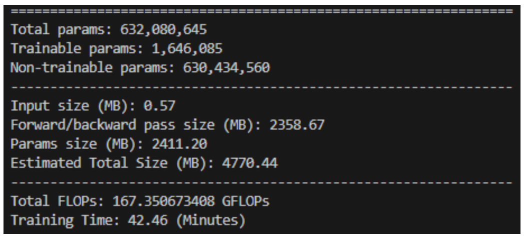


```
=================================================================
Total params: 306,512,901
Trainable params: 1,054,725
Non-trainable params: 305,458,176
-----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 253.02
Params size (MB): 1169.25
Estimated Total Size (MB): 1422.85
-----------------------------------------------------------------
Total FLOPs: 15.381037568 GFLOPs
Training Time: 12.46 (Minutes)
```

- 原始 ViT-Huge model (ViT-H/14)

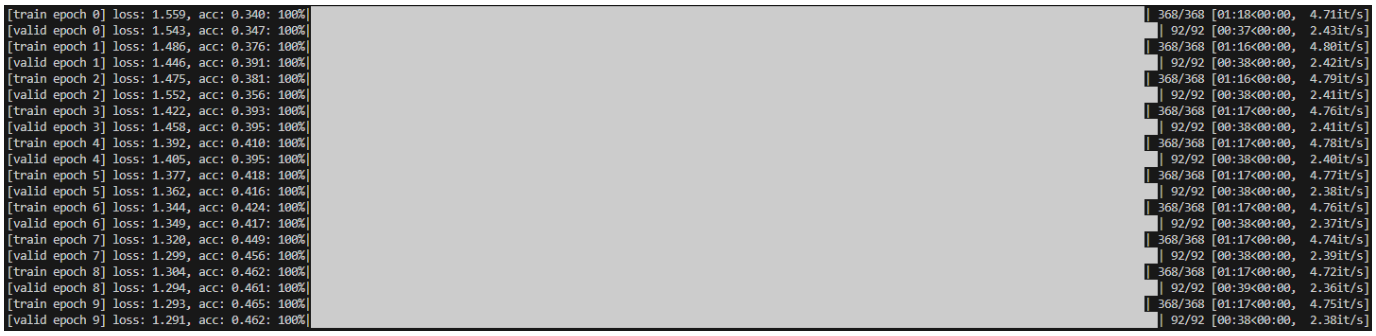


```
[train epoch 0] loss: 1.416, acc: 0.360: 100%|██████████| 368/368 [02:57<00:00,  2.07it/s]
[valid epoch 0] loss: 1.397, acc: 0.373: 100%|██████████| 92/92 [01:05<00:00,  1.41it/s]
[train epoch 1] loss: 1.354, acc: 0.411: 100%|██████████| 368/368 [03:02<00:00,  2.02it/s]
[valid epoch 1] loss: 1.347, acc: 0.430: 100%|██████████| 92/92 [01:06<00:00,  1.39it/s]
[train epoch 2] loss: 1.337, acc: 0.420: 100%|██████████| 368/368 [03:03<00:00,  2.01it/s]
[valid epoch 2] loss: 1.315, acc: 0.447: 100%|██████████| 92/92 [01:06<00:00,  1.39it/s]
[train epoch 3] loss: 1.315, acc: 0.425: 100%|██████████| 368/368 [03:03<00:00,  2.01it/s]
[valid epoch 3] loss: 1.339, acc: 0.453: 100%|██████████| 92/92 [01:06<00:00,  1.39it/s]
[train epoch 4] loss: 1.291, acc: 0.450: 100%|██████████| 368/368 [03:04<00:00,  1.99it/s]
[valid epoch 4] loss: 1.282, acc: 0.462: 100%|██████████| 92/92 [01:07<00:00,  1.37it/s]
[train epoch 5] loss: 1.267, acc: 0.470: 100%|██████████| 368/368 [03:05<00:00,  1.99it/s]
[valid epoch 5] loss: 1.282, acc: 0.486: 100%|██████████| 92/92 [01:06<00:00,  1.38it/s]
[train epoch 6] loss: 1.265, acc: 0.464: 100%|██████████| 368/368 [03:05<00:00,  1.99it/s]
[valid epoch 6] loss: 1.254, acc: 0.487: 100%|██████████| 92/92 [01:07<00:00,  1.37it/s]
[train epoch 7] loss: 1.235, acc: 0.478: 100%|██████████| 368/368 [03:05<00:00,  1.98it/s]
[valid epoch 7] loss: 1.273, acc: 0.469: 100%|██████████| 92/92 [01:12<00:00,  1.27it/s]
[train epoch 8] loss: 1.237, acc: 0.477: 100%|██████████| 368/368 [03:08<00:00,  1.95it/s]
[valid epoch 8] loss: 1.239, acc: 0.487: 100%|██████████| 92/92 [01:07<00:00,  1.37it/s]
[train epoch 9] loss: 1.229, acc: 0.486: 100%|██████████| 368/368 [03:05<00:00,  1.99it/s]
[valid epoch 9] loss: 1.236, acc: 0.499: 100%|██████████| 92/92 [01:06<00:00,  1.38it/s]
```

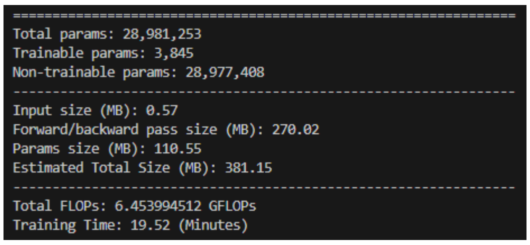


```
==========================================================================
Total params: 632,080,645
Trainable params: 1,646,085
Non-trainable params: 630,434,560
--------------------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 2358.67
Params size (MB): 2411.20
Estimated Total Size (MB): 4770.44
--------------------------------------------------------------------------
Total FLOPs: 167.350673408 GFLOPs
Training Time: 42.46 (Minutes)
```

- KAN 重構 ViT-Base model (ViT-B/16)

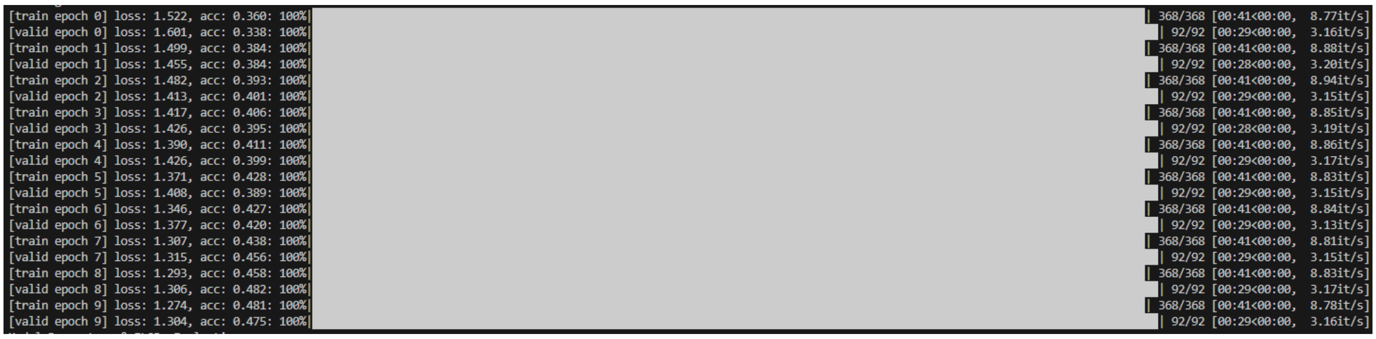


```
[train epoch 0] loss: 1.559, acc: 0.340: 100%|██████████| 368/368 [01:18<00:00,  4.71it/s]
[valid epoch 0] loss: 1.543, acc: 0.347: 100%|██████████| 92/92 [00:37<00:00,  2.43it/s]
[train epoch 1] loss: 1.486, acc: 0.376: 100%|██████████| 368/368 [01:16<00:00,  4.80it/s]
[valid epoch 1] loss: 1.446, acc: 0.391: 100%|██████████| 92/92 [00:38<00:00,  2.42it/s]
[train epoch 2] loss: 1.475, acc: 0.381: 100%|██████████| 368/368 [01:16<00:00,  4.79it/s]
[valid epoch 2] loss: 1.552, acc: 0.356: 100%|██████████| 92/92 [00:38<00:00,  2.41it/s]
[train epoch 3] loss: 1.422, acc: 0.393: 100%|██████████| 368/368 [01:17<00:00,  4.76it/s]
[valid epoch 3] loss: 1.458, acc: 0.395: 100%|██████████| 92/92 [00:38<00:00,  2.41it/s]
[train epoch 4] loss: 1.392, acc: 0.410: 100%|██████████| 368/368 [01:17<00:00,  4.78it/s]
[valid epoch 4] loss: 1.405, acc: 0.395: 100%|██████████| 92/92 [00:38<00:00,  2.40it/s]
[train epoch 5] loss: 1.377, acc: 0.418: 100%|██████████| 368/368 [01:17<00:00,  4.77it/s]
[valid epoch 5] loss: 1.362, acc: 0.416: 100%|██████████| 92/92 [00:38<00:00,  2.38it/s]
[train epoch 6] loss: 1.344, acc: 0.424: 100%|██████████| 368/368 [01:17<00:00,  4.76it/s]
[valid epoch 6] loss: 1.349, acc: 0.417: 100%|██████████| 92/92 [00:38<00:00,  2.37it/s]
[train epoch 7] loss: 1.320, acc: 0.449: 100%|██████████| 368/368 [01:17<00:00,  4.74it/s]
[valid epoch 7] loss: 1.299, acc: 0.456: 100%|██████████| 92/92 [00:38<00:00,  2.39it/s]
[train epoch 8] loss: 1.304, acc: 0.462: 100%|██████████| 368/368 [01:17<00:00,  4.72it/s]
[valid epoch 8] loss: 1.294, acc: 0.461: 100%|██████████| 92/92 [00:39<00:00,  2.36it/s]
[train epoch 9] loss: 1.293, acc: 0.465: 100%|██████████| 368/368 [01:17<00:00,  4.75it/s]
[valid epoch 9] loss: 1.291, acc: 0.462: 100%|██████████| 92/92 [00:38<00:00,  2.38it/s]
```

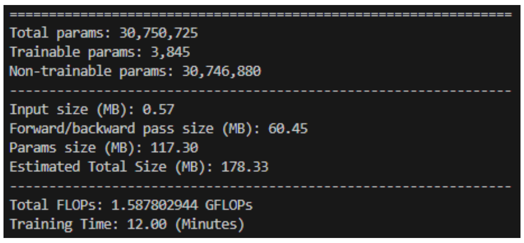


```
==========================================================================
Total params: 28,981,253
Trainable params: 3,845
Non-trainable params: 28,977,408
--------------------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 270.02
Params size (MB): 110.55
Estimated Total Size (MB): 381.15
--------------------------------------------------------------------------
Total FLOPs: 6.453994512 GFLOPs
Training Time: 19.52 (Minutes)
```

- KAN 重構 ViT-Base model (ViT-B/32)

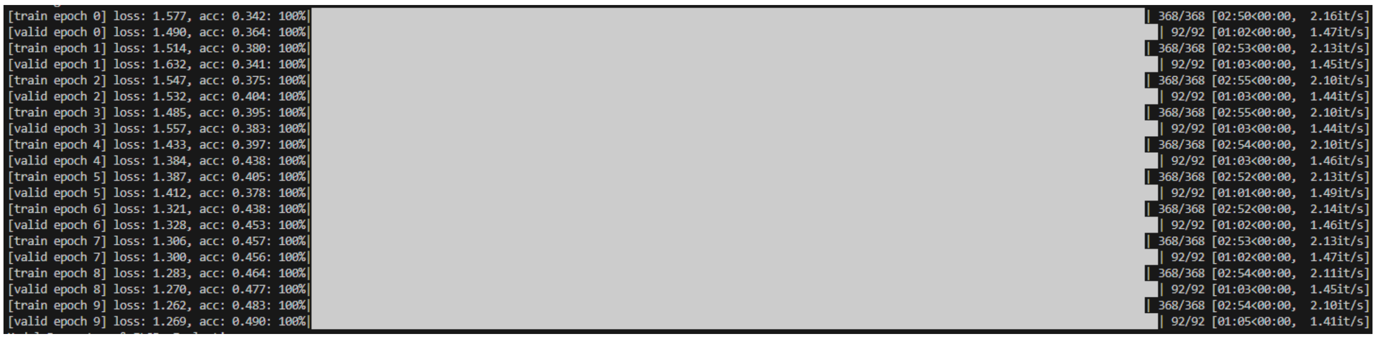


```
[train epoch 0] loss: 1.522, acc: 0.360: 100%|██████████| 368/368 [00:41<00:00,  8.77it/s]
[valid epoch 0] loss: 1.601, acc: 0.338: 100%|██████████| 92/92 [00:29<00:00,  3.16it/s]
[train epoch 1] loss: 1.499, acc: 0.384: 100%|██████████| 368/368 [00:41<00:00,  8.88it/s]
[valid epoch 1] loss: 1.455, acc: 0.384: 100%|██████████| 92/92 [00:28<00:00,  3.20it/s]
[train epoch 2] loss: 1.482, acc: 0.393: 100%|██████████| 368/368 [00:41<00:00,  8.94it/s]
[valid epoch 2] loss: 1.413, acc: 0.401: 100%|██████████| 92/92 [00:29<00:00,  3.15it/s]
[train epoch 3] loss: 1.417, acc: 0.406: 100%|██████████| 368/368 [00:41<00:00,  8.85it/s]
[valid epoch 3] loss: 1.426, acc: 0.395: 100%|██████████| 92/92 [00:28<00:00,  3.19it/s]
[train epoch 4] loss: 1.390, acc: 0.411: 100%|██████████| 368/368 [00:41<00:00,  8.86it/s]
[valid epoch 4] loss: 1.426, acc: 0.399: 100%|██████████| 92/92 [00:29<00:00,  3.17it/s]
[train epoch 5] loss: 1.371, acc: 0.428: 100%|██████████| 368/368 [00:41<00:00,  8.83it/s]
[valid epoch 5] loss: 1.408, acc: 0.389: 100%|██████████| 92/92 [00:29<00:00,  3.15it/s]
[train epoch 6] loss: 1.346, acc: 0.427: 100%|██████████| 368/368 [00:41<00:00,  8.84it/s]
[valid epoch 6] loss: 1.377, acc: 0.420: 100%|██████████| 92/92 [00:29<00:00,  3.13it/s]
[train epoch 7] loss: 1.307, acc: 0.438: 100%|██████████| 368/368 [00:41<00:00,  8.81it/s]
[valid epoch 7] loss: 1.315, acc: 0.456: 100%|██████████| 92/92 [00:29<00:00,  3.15it/s]
[train epoch 8] loss: 1.293, acc: 0.458: 100%|██████████| 368/368 [00:41<00:00,  8.83it/s]
[valid epoch 8] loss: 1.306, acc: 0.482: 100%|██████████| 92/92 [00:29<00:00,  3.17it/s]
[train epoch 9] loss: 1.274, acc: 0.481: 100%|██████████| 368/368 [00:41<00:00,  8.78it/s]
[valid epoch 9] loss: 1.304, acc: 0.475: 100%|██████████| 92/92 [00:29<00:00,  3.16it/s]
```

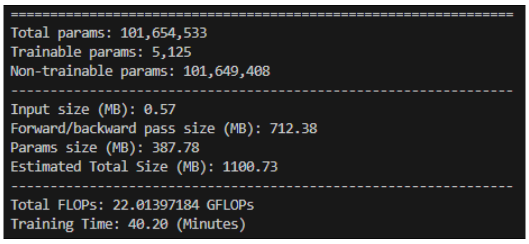


```
==========================================================================
Total params: 30,750,725
Trainable params: 3,845
Non-trainable params: 30,746,880
--------------------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 60.45
Params size (MB): 117.30
Estimated Total Size (MB): 178.33
--------------------------------------------------------------------------
Total FLOPs: 1.587802944 GFLOPs
Training Time: 12.00 (Minutes)
```

- KAN 重構 ViT-Large model (ViT-L/16)

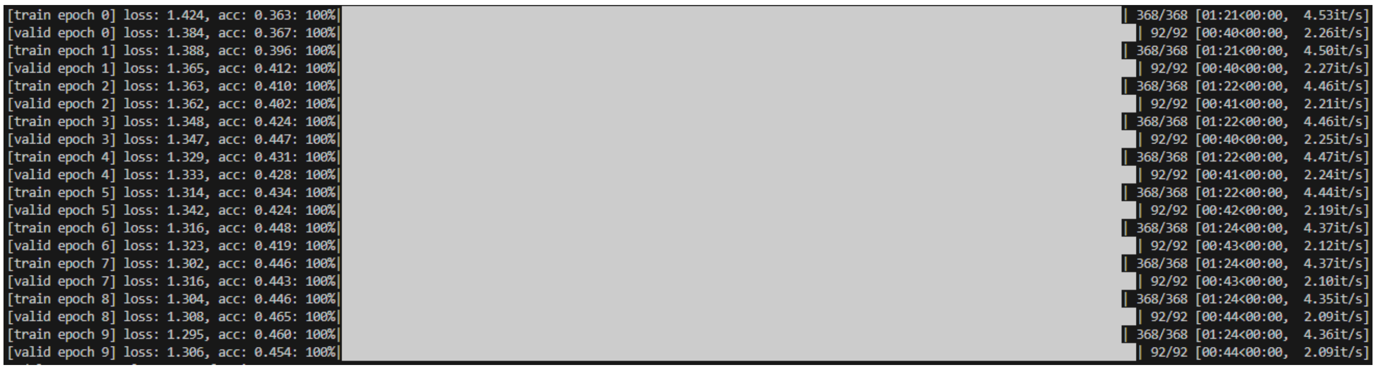


```
[train epoch 0] loss: 1.577, acc: 0.342: 100%|| 368/368 [02:50<00:00,  2.16it/s]
[valid epoch 0] loss: 1.490, acc: 0.364: 100%||  92/92 [01:02<00:00,  1.47it/s]
[train epoch 1] loss: 1.514, acc: 0.380: 100%|| 368/368 [02:53<00:00,  2.13it/s]
[valid epoch 1] loss: 1.632, acc: 0.341: 100%||  92/92 [01:03<00:00,  1.45it/s]
[train epoch 2] loss: 1.547, acc: 0.375: 100%|| 368/368 [02:55<00:00,  2.10it/s]
[valid epoch 2] loss: 1.532, acc: 0.404: 100%||  92/92 [01:03<00:00,  1.44it/s]
[train epoch 3] loss: 1.485, acc: 0.395: 100%|| 368/368 [02:55<00:00,  2.10it/s]
[valid epoch 3] loss: 1.557, acc: 0.383: 100%||  92/92 [01:03<00:00,  1.44it/s]
[train epoch 4] loss: 1.433, acc: 0.397: 100%|| 368/368 [02:54<00:00,  2.10it/s]
[valid epoch 4] loss: 1.384, acc: 0.438: 100%||  92/92 [01:03<00:00,  1.46it/s]
[train epoch 5] loss: 1.387, acc: 0.405: 100%|| 368/368 [02:52<00:00,  2.13it/s]
[valid epoch 5] loss: 1.412, acc: 0.378: 100%||  92/92 [01:01<00:00,  1.49it/s]
[train epoch 6] loss: 1.321, acc: 0.438: 100%|| 368/368 [02:52<00:00,  2.14it/s]
[valid epoch 6] loss: 1.328, acc: 0.453: 100%||  92/92 [01:02<00:00,  1.46it/s]
[train epoch 7] loss: 1.306, acc: 0.457: 100%|| 368/368 [02:53<00:00,  2.13it/s]
[valid epoch 7] loss: 1.300, acc: 0.456: 100%||  92/92 [01:02<00:00,  1.47it/s]
[train epoch 8] loss: 1.283, acc: 0.464: 100%|| 368/368 [02:54<00:00,  2.11it/s]
[valid epoch 8] loss: 1.270, acc: 0.477: 100%||  92/92 [01:03<00:00,  1.45it/s]
[train epoch 9] loss: 1.262, acc: 0.483: 100%|| 368/368 [02:54<00:00,  2.10it/s]
[valid epoch 9] loss: 1.269, acc: 0.490: 100%||  92/92 [01:05<00:00,  1.41it/s]
```

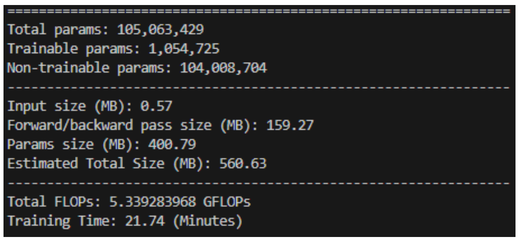


```
==============================================================
Total params: 101,654,533
Trainable params: 5,125
Non-trainable params: 101,649,408
--------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 712.38
Params size (MB): 387.78
Estimated Total Size (MB): 1100.73
--------------------------------------------------------------
Total FLOPs: 22.01397184 GFLOPs
Training Time: 40.20 (Minutes)
```

- KAN 重構 ViT-Large model (ViT-L/32)

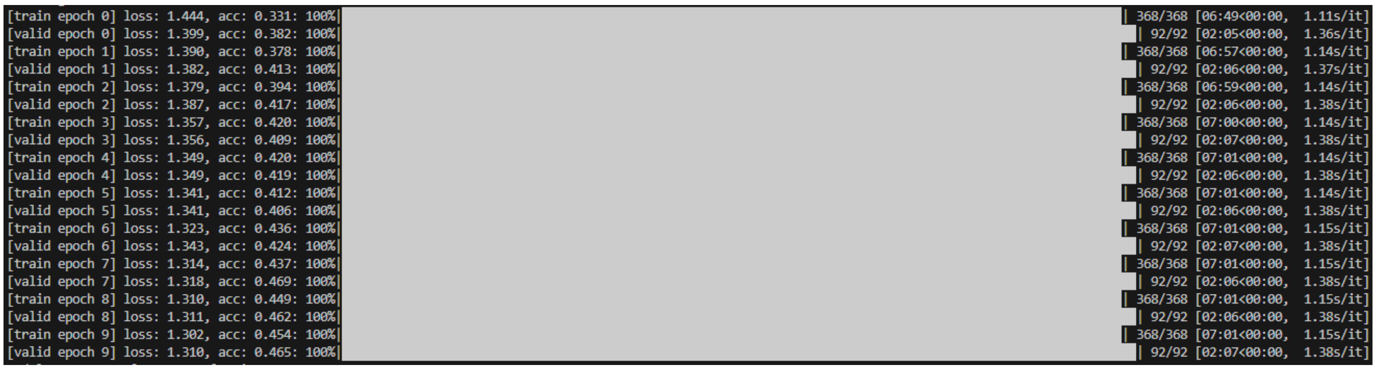


```
[train epoch 0] loss: 1.424, acc: 0.363: 100%|| 368/368 [01:21<00:00,  4.53it/s]
[valid epoch 0] loss: 1.384, acc: 0.367: 100%||  92/92 [00:40<00:00,  2.26it/s]
[train epoch 1] loss: 1.388, acc: 0.396: 100%|| 368/368 [01:21<00:00,  4.50it/s]
[valid epoch 1] loss: 1.365, acc: 0.412: 100%||  92/92 [00:40<00:00,  2.27it/s]
[train epoch 2] loss: 1.363, acc: 0.410: 100%|| 368/368 [01:22<00:00,  4.46it/s]
[valid epoch 2] loss: 1.362, acc: 0.402: 100%||  92/92 [00:41<00:00,  2.21it/s]
[train epoch 3] loss: 1.348, acc: 0.424: 100%|| 368/368 [01:22<00:00,  4.46it/s]
[valid epoch 3] loss: 1.347, acc: 0.447: 100%||  92/92 [00:40<00:00,  2.25it/s]
[train epoch 4] loss: 1.329, acc: 0.431: 100%|| 368/368 [01:22<00:00,  4.47it/s]
[valid epoch 4] loss: 1.333, acc: 0.428: 100%||  92/92 [00:41<00:00,  2.24it/s]
[train epoch 5] loss: 1.314, acc: 0.434: 100%|| 368/368 [01:22<00:00,  4.44it/s]
[valid epoch 5] loss: 1.342, acc: 0.424: 100%||  92/92 [00:42<00:00,  2.19it/s]
[train epoch 6] loss: 1.316, acc: 0.448: 100%|| 368/368 [01:24<00:00,  4.37it/s]
[valid epoch 6] loss: 1.323, acc: 0.419: 100%||  92/92 [00:43<00:00,  2.12it/s]
[train epoch 7] loss: 1.302, acc: 0.446: 100%|| 368/368 [01:24<00:00,  4.37it/s]
[valid epoch 7] loss: 1.316, acc: 0.443: 100%||  92/92 [00:43<00:00,  2.10it/s]
[train epoch 8] loss: 1.304, acc: 0.446: 100%|| 368/368 [01:24<00:00,  4.35it/s]
[valid epoch 8] loss: 1.308, acc: 0.465: 100%||  92/92 [00:44<00:00,  2.09it/s]
[train epoch 9] loss: 1.295, acc: 0.460: 100%|| 368/368 [01:24<00:00,  4.36it/s]
[valid epoch 9] loss: 1.306, acc: 0.454: 100%||  92/92 [00:44<00:00,  2.09it/s]
```

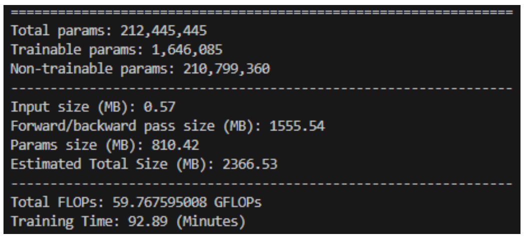


```
==============================================================
Total params: 105,063,429
Trainable params: 1,054,725
Non-trainable params: 104,008,704
--------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 159.27
Params size (MB): 400.79
Estimated Total Size (MB): 560.63
--------------------------------------------------------------
Total FLOPs: 5.339283968 GFLOPs
Training Time: 21.74 (Minutes)
```

- KAN 重構 ViT-Huge model (ViT-H/14)



```
[train epoch 0] loss: 1.444, acc: 0.331: 100%|| 368/368 [06:49<00:00,  1.11s/it]
[valid epoch 0] loss: 1.399, acc: 0.382: 100%||  92/92 [02:05<00:00,  1.36s/it]
[train epoch 1] loss: 1.390, acc: 0.378: 100%|| 368/368 [06:57<00:00,  1.14s/it]
[valid epoch 1] loss: 1.382, acc: 0.413: 100%||  92/92 [02:06<00:00,  1.37s/it]
[train epoch 2] loss: 1.379, acc: 0.394: 100%|| 368/368 [06:59<00:00,  1.14s/it]
[valid epoch 2] loss: 1.387, acc: 0.417: 100%||  92/92 [02:06<00:00,  1.38s/it]
[train epoch 3] loss: 1.357, acc: 0.420: 100%|| 368/368 [07:00<00:00,  1.14s/it]
[valid epoch 3] loss: 1.356, acc: 0.409: 100%||  92/92 [02:07<00:00,  1.38s/it]
[train epoch 4] loss: 1.349, acc: 0.420: 100%|| 368/368 [07:01<00:00,  1.14s/it]
[valid epoch 4] loss: 1.349, acc: 0.419: 100%||  92/92 [02:06<00:00,  1.38s/it]
[train epoch 5] loss: 1.341, acc: 0.412: 100%|| 368/368 [07:01<00:00,  1.14s/it]
[valid epoch 5] loss: 1.341, acc: 0.406: 100%||  92/92 [02:06<00:00,  1.38s/it]
[train epoch 6] loss: 1.323, acc: 0.436: 100%|| 368/368 [07:01<00:00,  1.15s/it]
[valid epoch 6] loss: 1.343, acc: 0.424: 100%||  92/92 [02:07<00:00,  1.38s/it]
[train epoch 7] loss: 1.314, acc: 0.437: 100%|| 368/368 [07:01<00:00,  1.15s/it]
[valid epoch 7] loss: 1.318, acc: 0.469: 100%||  92/92 [02:06<00:00,  1.38s/it]
[train epoch 8] loss: 1.310, acc: 0.449: 100%|| 368/368 [07:01<00:00,  1.15s/it]
[valid epoch 8] loss: 1.311, acc: 0.462: 100%||  92/92 [02:06<00:00,  1.38s/it]
[train epoch 9] loss: 1.302, acc: 0.454: 100%|| 368/368 [07:01<00:00,  1.15s/it]
[valid epoch 9] loss: 1.310, acc: 0.465: 100%||  92/92 [02:07<00:00,  1.38s/it]
```



```
==============================================================
Total params: 212,445,445
Trainable params: 1,646,085
Non-trainable params: 210,799,360
--------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 1555.54
Params size (MB): 810.42
Estimated Total Size (MB): 2366.53
--------------------------------------------------------------
Total FLOPs: 59.767595008 GFLOPs
Training Time: 92.89 (Minutes)
```

- 比對的表格

ViT Model	Transformer Design	Training Time (Minutes)	Accuracy (Valid Epoch)	Total Params (M)	Total FLOPs (GFLOPs)	Estimated Size (MB)
ViT-Base (ViT-B/16)	Original	11.87	0.540	85.65	17.57	735.84
ViT-Base (ViT-B/16)	KAN	19.52	0.462	28.98	6.45	381.15
ViT-Base (ViT-B/32)	Original	9.33	0.506	87.42	4.41	429.66
ViT-Base (ViT-B/32)	KAN	12.00	0.475	30.75	1.59	178.33
ViT-Large (ViT-L/16)	Original	20.74	0.535	303.10	64.58	2238.58
ViT-Large (ViT-L/16)	KAN	40.20	0.490	101.65	22.01	1100.73
ViT-Large (ViT-L/32)	Original	12.46	0.506	306.51	15.38	1422.85
ViT-Large (ViT-L/32)	KAN	21.74	0.454	105.06	5.34	560.63
ViT-Huge (ViT-H/14)	Original	42.46	0.499	632.08	167.35	4770.44
ViT-Huge (ViT-H/14)	KAN	92.89	0.465	212.45	59.77	2366.53

訓練優化後

- 原始 Vision Transformer
- KAN 重構後 Vision Transformer
- 比對的表格