Positional Convolution Experts (PCE) Network Architecture

Mirkzx

July 2, 2025

Contents

1 Introduction **2**

2 Problem Definition **2**

3 PCE Solution **2**

4 Network Architecture and Pipeline **2**

4.1 Architecture Overview 2

4.1.1 PatchExtractor 3

4.1.2 Router 3

4.1.3 Convolutional Expert 4

4.2 Pipeline 4

5 Training Methodology **4**

5.1 Phase 1: Stabilization (Epochs 0-150) 4

5.2 Phase 2: Fine Tuning (Epochs 150+) 5

6 Loss Function **5**

6.1 Classification Loss 5

6.2 Router Regularization 5

6.2.1 Confidence Loss 5

6.2.2 Anti-Collapse Loss 5

6.3 Total Loss 5

7 Implementation Example **6**

8 Key Features **6**

1 Introduction

This repository implements the **Positional Convolution Experts (PCE)** network. The PCE architecture addresses limitations of traditional convolutional neural networks by implementing specialized processing through experts for different image regions.

2 Problem Definition

Traditional CNNs, even modern ones, apply the same filter across the entire image, which can cause the loss of deep semantic details. Consider an image containing:

- Different people wearing various colored clothes
- A street with specific texture
- Trees with detailed foliage
- Urban panorama (houses, buildings, etc.)

Semantically, this is a rich image where each element has well-defined semantics. Applying the same filter across the entire image could lose details about the semantics of individual objects, such as:

- The specific color of leaves
- The precise texture of the sidewalk
- The architectural details of buildings
- The chromatic variations of clothing

3 PCE Solution

PCE addresses this problem by applying *convolutional experts* to image portions that become progressively smaller as network depth increases. This means that in deeper layers, experts will specialize on particularly small patches, enabling them to learn semantic relationships of tiny image portions containing rich information about specific objects or object parts within the entire image.

To decide which expert to delegate each patch to, we use a **router** that takes as input the patch enriched with two spatial information types:

1. The absolute position of the patch within the image
2. The relative position of pixels within the individual patch

4 Network Architecture and Pipeline

4.1 Architecture Overview

For simplicity, we analyze a single layer. All other layers repeat the same structure. We have 3 main components:

4.1.1 PatchExtractor

The `PatchExtractor` takes the feature map as input and splits it into P patches of size $N \times N$, enriching the channel dimension with absolute patch position and relative pixel position:

$$\mathbf{X}_{\text{patches}} = [B, P, C+4, H, W] \tag{1}$$

where:

- B = batch size
- P = number of patches
- $C + 4$ = original channels + 4 positional channels
- H, W = patch height and width

4.1.2 Router

The router takes the matrix $\mathbf{X}_{\text{patches}} \in \mathbb{R}^{B \times P \times 8 \times H \times W}$ and applies *Spatial Pyramid Pooling* (SSP) to obtain patch embeddings. These embeddings are used to find similarity with router keys through cosine similarity:

Similarity Calculation:

$$\text{similarity} = \frac{\mathbf{e}_{\text{patch}} \cdot \mathbf{K}^T}{||\mathbf{e}_{\text{patch}}|| \cdot ||\mathbf{K}||} \tag{2}$$

$$\mathbf{w} = \text{softmax}(\text{similarity}) \tag{3}$$

where $\mathbf{e}_{\text{patch}}$ is the patch embedding and $\mathbf{K}$ are the router keys.

Adaptive Threshold: For adaptive threshold calculation, various statistics are used:

$$w_{\max} = \max_{i \in \{1,\ldots,E\}} \{w_i\} \tag{4}$$

$$\text{max\_component} = 1 - w_{\max} \tag{5}$$

$$H(\mathbf{w}) = -\sum_{i=1}^{E} w_i \log(w_i) \tag{6}$$

$$H_{\max} = \log(E) \tag{7}$$

$$\text{entropy\_component} = \frac{H(\mathbf{w})}{H_{\max}} \tag{8}$$

$$\text{gap\_component} = \frac{w_{(1)} - \bar{w}_{\text{rest}}}{w_{(1)} + \epsilon} \tag{9}$$

where:

- $w_{(1)}$ is the weight of the expert with highest probability
- $\bar{w}_{\text{rest}} = \frac{1}{k} \sum_{i=2}^{k+1} w_{(i)}$ with $k = \min(4, E-1)$
- $\epsilon = 10^{-8}$ for numerical stability

The final adaptive threshold is:

$$\theta_{\text{adaptive}} = \lambda_1 \cdot \text{max\_component} + \lambda_2 \cdot \text{entropy\_component} + \lambda_3 \cdot \text{gap\_component} \tag{10}$$

Weights are filtered to respect the threshold:

$$\mathbf{w}_{\text{filtered}} = \mathbf{w} \odot \sigma(\beta(\mathbf{w} - \theta_{\text{adaptive}})) \tag{11}$$

where σ is the sigmoid function and β is a scaling parameter.

4.1.3 Convolutional Expert

Each expert takes its dedicated patch and applies the following operations twice (forming a ResNet block):

`Conv2d` → `BatchNorm2d` → `ReLU` → `Dropout`

4.2 Pipeline

The layer receives the feature map, which is decomposed into patches enriched with positional information. These are passed to the router, which chooses which expert (through a weight) to delegate each patch to.

Patches are processed by experts through Conv2D, BatchNorm, ReLU, and Dropout (applied twice). Once all expert feature maps are obtained, they are reaggregated through weighted sum:

$$\mathbf{F}_{\text{output}} = \sum_{i=1}^{E} w_i \cdot \mathbf{F}_i \tag{12}$$

where $\mathbf{F}_i$ is the feature map of expert i and w_i is its weight.

This final feature map is reconstructed by reinserting patches into their original positions, obtaining a new single feature map to which a final convolution is applied.

In the output layer, the feature map is transformed into a vector with SSP and passed to a fully connected layer to obtain final logits.

5 Training Methodology

The network is trained in **two distinct phases**:

5.1 Phase 1: Stabilization (Epochs 0-150)

In this phase, the network is trained with backpropagation on:

- Expert convolutions
- Projection convolution
- Final post-reassembly convolution

Router keys are updated with *Exponential Moving Average* (EMA):

$$\mathbf{k}_i^{(t+1)} = \alpha \cdot \mathbf{k}_i^{(t)} + (1 - \alpha) \cdot \frac{\sum_j w_{ij} \cdot \mathbf{e}_j}{\sum_j w_{ij}} \tag{13}$$

where w_{ij} is the weight of expert i for patch j and $\mathbf{e}_j$ is the embedding of patch j.

5.2 Phase 2: Fine Tuning (Epochs 150+)

Router keys are inserted into the computational graph and backpropagation is also performed on them:

```
if epoch >= 150:
    router.set_keys_trainable(True)
```

Listing 1: Transition to Phase 2

6 Loss Function

The PCE network uses a composite loss function that combines classification objectives with router regularization.

6.1 Classification Loss

$$\mathcal{L}_{\text{cls}} = \text{CrossEntropy}(\text{logits}, \text{labels}) \tag{14}$$

6.2 Router Regularization

6.2.1 Confidence Loss

Encourages confident expert selection by minimizing entropy:

$$\mathcal{L}_{\text{confidence}} = \frac{1}{|B||P|} \sum_{b=1}^{|B|} \sum_{p=1}^{|P|} H(\mathbf{w}_{bp}) \tag{15}$$

where $H(\mathbf{w}_{bp}) = -\sum_{e=1}^{E} w_{bpe} \log(w_{bpe})$.

6.2.2 Anti-Collapse Loss

Prevents expert collapse by penalizing unused experts:

$$\mathcal{L}_{\text{collapse}} = \sum_{e=1}^{E} \mathbf{1}_{[\text{usage}_e < \tau]} \tag{16}$$

where:

$$\text{usage}_e = \frac{1}{|B||P|} \sum_{b,p} \mathbf{1}_{[w_{bpe} > 0]} \tag{17}$$

and $\tau = 0.01$ is the minimum usage threshold.

6.3 Total Loss

$$\mathcal{L}_{\text{router}} = \alpha \cdot \mathcal{L}_{\text{confidence}} + \beta \cdot \mathcal{L}_{\text{collapse}} \tag{18}$$

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{router}} \tag{19}$$

where α and β are regularization hyperparameters.

7 Implementation Example

```python
from Model.PCE import PCENetwork
from Model.Components.Router import Router

# Initialize router and model
router = Router(num_experts=4, temperature=1.0)
model = PCENetwork(
    input_channels=7,  # 3 RGB + 4 positional
    num_experts=4,
    patch_size=16,
    router=router,
    num_classes=10
)

# Training setup
optimizer = Adam(model.parameters(), lr=0.001)
scheduler = PCEScheduler(
    optimizer,
    phase_epochs=[150, 100],
    base_lr=0.001,
    phase_multipliers=[1.0, 0.3]
)
```

Listing 2: Basic PCE Network Usage

8 Key Features

- **Spatial Specialization**: Each expert learns region-specific features
- **Computational Efficiency**: Only selected experts process patches
- **Interpretability**: Routing decisions provide insights into network behavior
- **Robustness**: Adaptive threshold prevents expert overfitting
- **Scalability**: Architecture can be extended with more layers and experts