2025,04,25

TimeSFormerのアーキテクチャ(ChatGPT-o4作成)

この実装では、空間的注意を統合するために、時間的注意と空間的注意が組み合わさるように工夫しています。

```python

import torch

import torch.nn as nn

import torch.nn.functional as F

class PatchEmbedding(nn.Module):o4

def \_\_init\_\_(self, in\_channels, patch\_size, embed\_dim):

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

self.patch\_size = patch\_size

self.proj = nn.Conv3d(in\_channels, embed\_dim, kernel\_size=(1, patch\_size, patch\_size),

stride=(1, patch\_size, patch\_size))

def forward(self, x):

x = self.proj(x) # (B, C, T, H', W')

x = x.flatten(2) # (B, C, T\*H'\*W')

x = x.transpose(1, 2) # (B, T\*H'\*W', C)

return x

class TemporalAttention(nn.Module):

def \_\_init\_\_(self, embed\_dim, num\_heads):

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

self.num\_heads = num\_heads

self.head\_dim = embed\_dim // num\_heads

assert self.head\_dim \* num\_heads == embed\_dim, "Embedding dimension must be divisible by number of heads"

self.values = nn.Linear(embed\_dim, embed\_dim, bias=False)

self.keys = nn.Linear(embed\_dim, embed\_dim, bias=False)

self.queries = nn.Linear(embed\_dim, embed\_dim, bias=False)

self.fc\_out = nn.Linear(embed\_dim, embed\_dim)

def forward(self, x):

N, seq\_length, embed\_dim = x.shape

values = self.values(x)

keys = self.keys(x)

queries = self.queries(x)

queries = queries.view(N, seq\_length, self.num\_heads, self.head\_dim).transpose(1, 2) # (N, heads, seq\_length, head\_dim)

keys = keys.view(N, seq\_length, self.num\_heads, self.head\_dim).transpose(1, 2) # (N, heads, seq\_length, head\_dim)

values = values.view(N, seq\_length, self.num\_heads, self.head\_dim).transpose(1, 2) # (N, heads, seq\_length, head\_dim)

energy = queries @ keys.transpose(-2, -1) # (N, heads, seq\_length, seq\_length)

attention = F.softmax(energy / (self.head\_dim \*\* (1 / 2)), dim=-1)

out = attention @ values # (N, heads, seq\_length, head\_dim)

out = out.transpose(1, 2).contiguous().view(N, seq\_length, embed\_dim) # (N, seq\_length, embed\_dim)

return self.fc\_out(out)

class SpatialAttention(nn.Module):

def \_\_init\_\_(self, embed\_dim):

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

self.conv = nn.Conv2d(embed\_dim, embed\_dim, kernel\_size=1)

self.sigmoid = nn.Sigmoid()

def forward(self, x):

# x: (B, N\_patches, embed\_dim)

B, N\_patches, embed\_dim = x.shape

# N\_patchesをH', W'に再形成 (仮定：N\_patchesはT, H', W'の形になる)

H\_W = int(N\_patches \*\* 0.5)

x = x.transpose(1, 2).reshape(B, embed\_dim, H\_W, H\_W) # (B, embed\_dim, H', W')

# 空間的注意を計算

attention = self.sigmoid(self.conv(x)) # (B, embed\_dim, H', W')

x = x \* attention # 空間的なスケーリング

x = x.view(B, embed\_dim, -1).transpose(1, 2) # (B, H'\*W', embed\_dim)

return x

class TimeSFormer(nn.Module):

def \_\_init\_\_(self, in\_channels, embed\_dim, num\_heads, num\_classes, patch\_size=16):

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

self.patch\_embedding = PatchEmbedding(in\_channels, patch\_size, embed\_dim)

self.temporal\_attention = TemporalAttention(embed\_dim, num\_heads)

self.spatial\_attention = SpatialAttention(embed\_dim)

self.fc = nn.Linear(embed\_dim, num\_classes)

def forward(self, x):

x = self.patch\_embedding(x) # (B, N\_patches, embed\_dim)

x = self.temporal\_attention(x) # (B, N\_patches, embed\_dim)

x = self.spatial\_attention(x) # (B, H'\*W', embed\_dim)

x = x.mean(dim=1) # Global average pooling

x = self.fc(x) # (B, num\_classes)

return x

# 使用例

if \_\_name\_\_ == "\_\_main\_\_":

model = TimeSFormer(in\_channels=3, embed\_dim=128, num\_heads=8, num\_classes=10)

x = torch.randn(32, 3, 16, 224, 224) # (バッチサイズ, チャンネル, 時間, 高さ, 幅)

output = model(x)

print(output.shape) # (32, 10)

```

### 改良点の説明

1. \*\*SpatialAttentionクラス\*\*:

- 空間的な注意を計算するためのクラスです。

- `nn.Conv2d`を使って、埋め込み次元に対する1x1の畳み込みを行い、空間的注意を生成します。

- Sigmoid関数を適用して、0から1の範囲の注意重みを得ます。

2. \*\*TimeSFormerクラス\*\*:

- `temporal\_attention`の後に`spatial\_attention`を適用し、時間と空間の両方の側面から注意を払うことができるようにします。

この修正版では、時間的な特徴と空間的な特徴を分離して捉えることができるため、`TimeSFormer`は時系列データに対してより効果的に学習することが可能です。

この実装では、空間的注意を統合するために、時間的注意と空間的注意が組み合わさるように工夫しています。

```python

import torch

import torch.nn as nn

import torch.nn.functional as F

class PatchEmbedding(nn.Module):

def \_\_init\_\_(self, in\_channels, patch\_size, embed\_dim):

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

self.patch\_size = patch\_size

self.proj = nn.Conv3d(in\_channels, embed\_dim, kernel\_size=(1, patch\_size, patch\_size), stride=(1, patch\_size, patch\_size))

def forward(self, x):

x = self.proj(x) # (B, C, T, H', W')

x = x.flatten(2) # (B, C, T\*H'\*W')

x = x.transpose(1, 2) # (B, T\*H'\*W', C)

return x

class TemporalAttention(nn.Module):

def \_\_init\_\_(self, embed\_dim, num\_heads):

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

self.num\_heads = num\_heads

self.head\_dim = embed\_dim // num\_heads

assert self.head\_dim \* num\_heads == embed\_dim, "Embedding dimension must be divisible by number of heads"

self.values = nn.Linear(embed\_dim, embed\_dim, bias=False)

self.keys = nn.Linear(embed\_dim, embed\_dim, bias=False)

self.queries = nn.Linear(embed\_dim, embed\_dim, bias=False)

self.fc\_out = nn.Linear(embed\_dim, embed\_dim)

def forward(self, x):

N, seq\_length, embed\_dim = x.shape

values = self.values(x)

keys = self.keys(x)

queries = self.queries(x)

queries = queries.view(N, seq\_length, self.num\_heads, self.head\_dim).transpose(1, 2) # (N, heads, seq\_length, head\_dim)

keys = keys.view(N, seq\_length, self.num\_heads, self.head\_dim).transpose(1, 2) # (N, heads, seq\_length, head\_dim)

values = values.view(N, seq\_length, self.num\_heads, self.head\_dim).transpose(1, 2) # (N, heads, seq\_length, head\_dim)

energy = queries @ keys.transpose(-2, -1) # (N, heads, seq\_length, seq\_length)

attention = F.softmax(energy / (self.head\_dim \*\* (1 / 2)), dim=-1)

out = attention @ values # (N, heads, seq\_length, head\_dim)

out = out.transpose(1, 2).contiguous().view(N, seq\_length, embed\_dim) # (N, seq\_length, embed\_dim)

return self.fc\_out(out)

class SpatialAttention(nn.Module):

def \_\_init\_\_(self, embed\_dim):

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

self.conv = nn.Conv2d(embed\_dim, embed\_dim, kernel\_size=1)

self.sigmoid = nn.Sigmoid()

def forward(self, x):

# x: (B, N\_patches, embed\_dim)

B, N\_patches, embed\_dim = x.shape

# N\_patchesをH', W'に再形成 (仮定：N\_patchesはT, H', W'の形になる)

H\_W = int(N\_patches \*\* 0.5)

x = x.transpose(1, 2).reshape(B, embed\_dim, H\_W, H\_W) # (B, embed\_dim, H', W')

# 空間的注意を計算

attention = self.sigmoid(self.conv(x)) # (B, embed\_dim, H', W')

x = x \* attention # 空間的なスケーリング

x = x.view(B, embed\_dim, -1).transpose(1, 2) # (B, H'\*W', embed\_dim)

return x

class TimeSFormer(nn.Module):

def \_\_init\_\_(self, in\_channels, embed\_dim, num\_heads, num\_classes, patch\_size=16):

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

self.patch\_embedding = PatchEmbedding(in\_channels, patch\_size, embed\_dim)

self.temporal\_attention = TemporalAttention(embed\_dim, num\_heads)

self.spatial\_attention = SpatialAttention(embed\_dim)

self.fc = nn.Linear(embed\_dim, num\_classes)

def forward(self, x):

x = self.patch\_embedding(x) # (B, N\_patches, embed\_dim)

x = self.temporal\_attention(x) # (B, N\_patches, embed\_dim)

x = self.spatial\_attention(x) # (B, H'\*W', embed\_dim)

x = x.mean(dim=1) # Global average pooling

x = self.fc(x) # (B, num\_classes)

return x

# 使用例

if \_\_name\_\_ == "\_\_main\_\_":

model = TimeSFormer(in\_channels=3, embed\_dim=128, num\_heads=8, num\_classes=10)

x = torch.randn(32, 3, 16, 224, 224) # (バッチサイズ, チャンネル, 時間, 高さ, 幅)

output = model(x)

print(output.shape) # (32, 10)

```

### 改良点の説明

1. \*\*SpatialAttentionクラス\*\*:

- 空間的な注意を計算するためのクラスです。

- `nn.Conv2d`を使って、埋め込み次元に対する1x1の畳み込みを行い、空間的注意を生成します。

- Sigmoid関数を適用して、0から1の範囲の注意重みを得ます。

2. \*\*TimeSFormerクラス\*\*:

- `temporal\_attention`の後に`spatial\_attention`を適用し、時間と空間の両方の側面から注意を払うことができるようにします。

この修正版では、時間的な特徴と空間的な特徴を分離して捉えることができるため、`TimeSFormer`は時系列データに対してより効果的に学習することが可能です。