Vision Transformer

Multi-Modal Spatial Temporal ViT

Lecture 2

University at Buffalo The State University of New York

Introduction

About Team 1:

Brason Dobson	Bhushan Mahajan	Deep Shahane
Computer Science M.S.	Computer Science M.S.	Computer Science M.S.

Overview

MMST-ViT: Climate Change-aware Crop Yield Prediction via Multi-Modal Spatial-Temporal Vision Transformer

Fudong Lin[1], Summer Crawford[2], Kaleb Guillot[2], Yihe Zhang[2], Yan Chen[3], Xu Yuan[1*], Li Chen[2], Shelby Williams[2], Robert Minvielle[2], Xiangming Xiao[4], Drew Gholson[5], Nicolas Ashwell[5], Tri Setiyono[6], Brenda Tubana[7], Lu Peng[8], Magdy Bayoumi[2], Nian-Feng Tzeng[2]
[1] University of Delaware, [2] University of Louisiana at Lafayette, [3] University of Connecticut, [4] University of Oklahoma, [5] Mississippi State University, [6] Louisiana State University, [7] LSU AgCenter, [8] Tulane University

Recap of Lecture

- Why Vision transformer?
- How vision transformers works
- What is MMST-ViT
- Problem statement of paper
- Key challenges addressed by the paper
- Datasets incorporated
  - Sentinal-2
  - HRRR
  - USDA

Index

- Overview of first lecture
- Multi Modal Transformer (Pyramid Vision Transformer)
- SimCLR Technique
- Spatial Transformer
- Temporal Transformer
- Experiments
- Comparative Performance Evaluation
- Conclusion

MMST-ViT

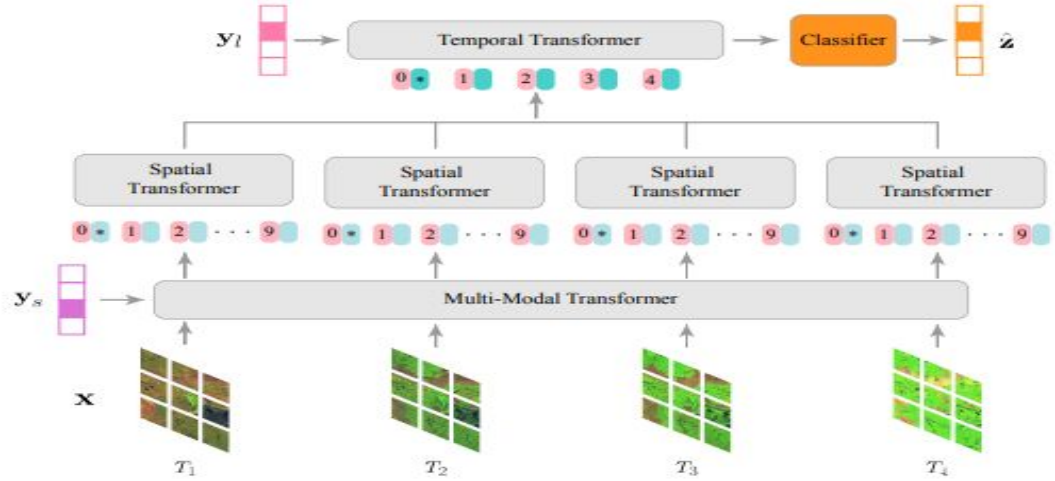


Figure 2: The architecture of our proposed MMST-ViT.

Introduction to Multi-Modal Transformer

- Aim to capture the direct impact of atmospheric weather variations on crop growth effectively.
- Key Components:
  - Visual Backbone Network(Pyramid vision transformer)
  - Multi-Modal Attention Layer
- Transformer Functionality $f_\theta(x, y_s) = v_m$, where
  - $v_m \in R^{TxGxd}$ is the output of MM Transformer.
  - x: Sentinel-2 images
  - Y_s: Short-term meteorological data.
- Introduction of a novel Multi-Modal Multi-Head Attention mechanism to better capture the nuanced effects of weather on crops.
- Implementation of a pioneering multi-modal self-supervised learning strategy using SimCLR for effective pre-training of the MM Transformer without the need for human-labeled data.

Pyramid Vision Transformer

- Type of architecture designed for computer vision tasks that require handling different scales of visual data, like object detection and semantic segmentation
- Idea of incorporating a pyramid structure to transformer model to generate multi-scale feature maps.
- Key Components:
  - Patch Embedding
  - Position Embedding
  - Transformer Blocks
  - Classifier

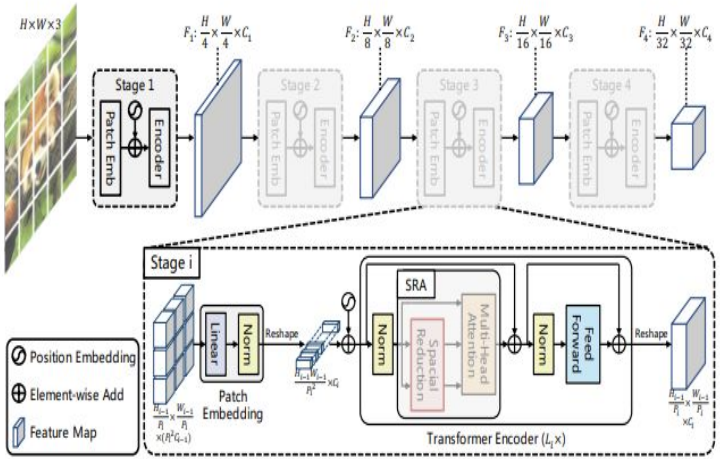


Figure 3: **Overall architecture of Pyramid Vision Transformer (PVT).** The entire model is divided into four stages, each of which is comprised of a patch embedding layer and a L_i-layer Transformer encoder. Following a pyramid structure, the output resolution of the four stages progressively shrinks from high (4-stride) to low (32-stride).

https://arxiv.org/abs/2102.12122

Multi Head Attention vs Spatial Reduction Attention

Comparison:

➢ MHA processes an input sequence by dividing it into multiple heads. Each head computes scaled dot-product attention independently, allowing the model to attend to information from different representation subspaces at different positions.	➢ Designed to reduce the spatial resolution of the 'key' and 'value' matrices before the attention operation, significantly lowering the computational and memory overhead, especially beneficial for high-resolution inputs. ➢ Before computing attention, SRA performs a spatial reduction on K and V, reducing their dimensions.
➢ $Attention(Q,K,V) = Softmax(QK^T/\sqrt{d_{head}})V$	➢ $SRA(Q,K,V) = concat(head_0,...,head_{Ni})W_0$ ➢ $Head_j = Attention(QW_0, SR(K)W_{kj}, SR(V)W_{vj})$ ➢ $SR(x) = Norm(Reshape(x,R_i)W_s)$ ➢ Where R_i is reduction ratio, reducing the spatial dimension by a factor of R_i^2.

Reduced Computational Load: The spatial reduction step lowers the size of the matrices involved in the attention calculations, leading to a reduction in computational complexity from $O(n^2)$ to $O(n^2/R_i^2)$, where n is the original dimension of the input sequence.

Efficiency: By lowering memory usage and computational demands, SRA can handle larger input feature maps or sequences with limited computational resources, making it suitable for high-resolution images.

Working of PVT

- **Input image**:
  - The process starts with an input image of dimensions H×W×C where H is height, W is width, and C is the number of channels .
- **Patch Embedding**:
  - Convert the image into a **sequence of flattened patches** which are then projected into an embedding space.
  - The image is first divided into patches (each of size patch_size×patch_size).
  - Each patch is then flattened and **linearly transformed** into a higher dimensional space.
  - The output at this stage is a sequence of embedded patches of dimension (H/patch_size)× (W/patch_size)×embed_dim.
- **Positional Embedding:**
  - Transformer architecture does not inherently process sequential data, positional embeddings are added to the patch embeddings to provide information position of the patches in the image.

```python
class PatchEmbed(nn.Module):
    """ Image to Patch Embedding
    """

    def __init__(self, img_size=224, patch_size=16, in_chans=3, embed_dim=768):
        super().__init__()
        img_size = to_2tuple(img_size)
        patch_size = to_2tuple(patch_size)

        self.img_size = img_size
        self.patch_size = patch_size
        # assert img_size[0] % patch_size[0] == 0 and img_size[1] % patch_size[1] == 0, \
        #     f"img_size {img_size} should be divided by patch_size {patch_size}."
        self.H, self.W = img_size[0] // patch_size[0], img_size[1] // patch_size[1]
        self.num_patches = self.H * self.W
        self.proj = nn.Conv2d(in_chans, embed_dim, kernel_size=patch_size, stride=patch_size)
        self.norm = nn.LayerNorm(embed_dim)

    def forward(self, x):
        B, C, H, W = x.shape

        x = self.proj(x).flatten(2).transpose(1, 2)
        x = self.norm(x)
        H, W = H // self.patch_size[0], W // self.patch_size[1]

        return x, (H, W)
```

```python
for i in range(num_stages):
    patch_embed = PatchEmbed(img_size=img_size if i == 0 else img_size // (2 ** (i + 1)),
                             patch_size=patch_size if i == 0 else 2,
                             in_chans=in_chans if i == 0 else embed_dims[i - 1],
                             embed_dim=embed_dims[i])
    num_patches = patch_embed.num_patches if i != num_stages - 1 else patch_embed.num_patches + 1
```

```python
pos_embed = nn.Parameter(torch.zeros(1, num_patches, embed_dims[i]))
pos_drop = nn.Dropout(p=drop_rate)
```

Working of PVT

- **Transformer Encoder stages:**
  - PVT processes the positional embedded patches through multiple stages of transformer blocks.
  - Each block contains:
  - **Layer Normalization:**
  - **Multi-Head Attention (with Spatial Reduction):**
    - This is a key difference from traditional transformers. PVT uses spatial reduction attention where the resolution of key and value tensors is reduced to decrease computational load, especially important for larger images.
    - Queries, keys, and values are generated from the input, but the spatial dimensions of keys and values are reduced, maintaining efficiency.
  - **Feed-Forward Network:** Consists of two linear transformations with a non-linearity in between.
  - **Output:** The transformer blocks output a sequence of feature embeddings for each patch.

```
class Block(nn.Module):

    def __init__(self, dim, num_heads, mlp_ratio=4., qkv_bias=False, qk_scale=None, drop=0., attn_drop=0.,
                 drop_path=0., act_layer=nn.GELU, norm_layer=nn.LayerNorm, sr_ratio=1):
        super().__init__()
        self.norm1 = norm_layer(dim)
        self.attn = Attention(
            dim,
            num_heads=num_heads, qkv_bias=qkv_bias, qk_scale=qk_scale,
            attn_drop=attn_drop, proj_drop=drop, sr_ratio=sr_ratio)
        # NOTE: drop path for stochastic depth, we shall see if this is better than dropout here
        self.drop_path = DropPath(drop_path) if drop_path > 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)

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

        return x
```

```
class Attention(nn.Module):
    def __init__(self, dim, num_heads=8, qkv_bias=False, qk_scale=None, attn_drop=0., proj_drop=0., sr_ratio=1):
        super().__init__()
        assert dim % num_heads == 0, f"dim {dim} should be divided by num_heads {num_heads}."

        self.dim = dim
        self.num_heads = num_heads
        head_dim = dim // num_heads
        self.scale = qk_scale or head_dim ** -0.5

        self.q = nn.Linear(dim, dim, bias=qkv_bias)
        self.kv = nn.Linear(dim, dim * 2, bias=qkv_bias)
        self.attn_drop = nn.Dropout(attn_drop)
        self.proj = nn.Linear(dim, dim)
        self.proj_drop = nn.Dropout(proj_drop)

        self.sr_ratio = sr_ratio
        if sr_ratio > 1:
            self.sr = nn.Conv2d(dim, dim, kernel_size=sr_ratio, stride=sr_ratio)
            self.norm = nn.LayerNorm(dim)
```

Working of PVT

- **Downsampling Between Stages:**
  - As the data passes through each stage, the resolution of the feature map is progressively reduced. This downsampling is achieved not by pooling, as in CNNs, but by adjusting the patch size and the stride in the patch embedding layer of each subsequent stage
- **Final Stage - Classification Head:**
  - After the last transformer stage, a classification token (often learned during training) is appended to the sequence of embeddings if the task is classification.
  - The sequence is passed through a final layer normalization and then to a fully connected layer (classification head) that outputs the class scores.

```
# classification head
self.head = nn.Linear(embed_dims[3], num_classes) if num_classes > 0 else nn.Identity()
```

Multi-Modal Multi-Head Attention (MM-MHA)

- **Purpose:**
  - Designed to enhance crop growth prediction by integrating high-resolution visual data from Pyramid Vision Transformers (PVT) with numerical meteorological data.
- **Core Concept:**
  - MM-MHA captures the dynamic effects of meteorological parameters on crop growth by processing multimodal data, offering a more comprehensive analysis than using visual data alone.

Mathematical Formulation for MM-MHA

- **Attention mechanism:**
  - MM-MHA(Q,K,V) = Softmax($QK^T/\sqrt{d_{head}}$)V
  - Here, $Q=W_Q^m\Phi_m(\chi_m)$, $K = W_K^m\tau_m(y_s)$, $V = W_V^m\tau_m(y_s)$
  - $\Phi_m(\chi_m)$ represents the visual features encoded by PVT.
  - $\tau_m(y_s)$ represents the transformed meteorological data.
- **Parameters:**
  - W_Q^m,W_K^m,W_V^m are learnable projection matrices that transform the inputs into the attention mechanism required format.
  - $\Phi_m(\chi_m)$ and $\tau_m(y_s)$ are the embeddings for visual and meteorological data, respectively.

These transformations ensure that both modalities are compatible for the attention mechanism, allowing the model to effectively learn from and integrate both types of data.

SimCLR

SimCLR (Simple Framework for Contrastive Learning of Visual Representations) is a self-supervised learning method that aims to learn useful representations by maximizing agreement between differently augmented views of the same data example via a contrastive loss in the latent space. The typical components of SimCLR include:

- **Data Augmentation Module:** Generates two augmented versions of the same image, which are treated as a positive pair.
- **Backbone Network:** Used to extract features from these augmented images.
- **Projection Head:** Transforms features to the space where contrastive loss is applied.
- **Contrastive Loss Function:** Typically, a variant of the NT-Xent (normalized temperature-scaled cross-entropy loss) is used to maximize similarity of features from the positive pair while minimizing similarity from negative pairs (different images).

```
class ContrastiveLoss(nn.Module):
    def __init__(self, batch_size, device, temperature=0.5):
        super().__init__()
        self.batch_size = batch_size

        device = torch.device(device)
        self.register_buffer("temperature", torch.tensor(temperature).to(device))
        self.register_buffer("negatives_mask", (~torch.eye(batch_size * 2, batch_size * 2, dtype=bool)).float().

    def forward(self, emb_i, emb_j):
        """
        emb_i and emb_j are batches of embeddings, where corresponding indices are pairs
        z_i, z_j as per SimCLR paper
        """
        z_i = F.normalize(emb_i, dim=1)
        z_j = F.normalize(emb_j, dim=1)

        representations = torch.cat([z_i, z_j], dim=0)
        similarity_matrix = F.cosine_similarity(representations.unsqueeze(1), representations.unsqueeze(0), dim=

        sim_ij = torch.diag(similarity_matrix, self.batch_size)
        sim_ji = torch.diag(similarity_matrix, -self.batch_size)
        positives = torch.cat([sim_ij, sim_ji], dim=0)

        nominator = torch.exp(positives / self.temperature)
        denominator = self.negatives_mask * torch.exp(similarity_matrix / self.temperature)

        loss_partial = -torch.log(nominator / torch.sum(denominator, dim=1))
        loss = torch.sum(loss_partial) / (2 * self.batch_size)
```

SimCLR Mathematics

$$\ell(i,j) = -\log \frac{\exp(\text{sim}(\boldsymbol{v}_m^i, \boldsymbol{v}_m^j)/\tau)}{\sum_{i \neq k, k=1,\ldots,2B} \exp(\text{sim}(\boldsymbol{v}_m^i, \boldsymbol{v}_m^k)/\tau)},$$

$$\mathcal{L}_{\text{cl}} = \frac{1}{2B} \sum_{k=1}^{B} \left[\ell(k, k+B) + \ell(k+B, k)\right], \tag{2}$$

where B = T × G. Here, v^i_m and v^j_m are the positive pair, and τ is the temperature parameter.

```python
class SimCLRDataTransform(object):
    def __init__(self, img_size=224, s=1, kernel_size=9):
        self.img_size = img_size
        self.s = s
        self.kernel_size = kernel_size
        self.transform = self.get_simclr_pipeline_transform()

    def __call__(self, sample):
        xi = self.transform(sample)
        xj = self.transform(sample)
        return xi, xj

    def get_simclr_pipeline_transform(self):
        # get a set of data augmentation transformations as described in the SimCLR paper.
        color_jitter = transforms.ColorJitter(0.8 * self.s, 0.8 * self.s, 0.8 * self.s, 0.2 * self.s)
        data_transforms = transforms.Compose([transforms.RandomResizedCrop(size=self.img_size),
                                              transforms.RandomHorizontalFlip(),
                                              transforms.RandomApply([color_jitter], p=0.8),
                                              transforms.RandomGrayscale(p=0.2),
                                              transforms.GaussianBlur(kernel_size=self.kernel_size),
                                              transforms.ToTensor(),
                                              # transforms.Normalize([0.466, 0.471, 0.380], [0.195, 0.194, 0.192]),
                                              ])
        return data_transforms
```

Workflow of MultiModal Transformer

- Model takes two types of data:Satellite images denoted by X" & short-term meteorological data.
- The augmented satellite images are processed through a Pyramid Vision Transformer. PVT is adept at handling high-resolution images and extracts complex features due to its capability to maintain a global receptive field.
- Outputs from the PVT (visual features of the satellite images) are then combined with the processed short-term meteorological data in a Multi-Modal Attention mechanism.
- This Multi-Modal Attention module is crucial for merging information from different modalities (visual and meteorological), enhancing the model's ability to interpret the combined effect of visual appearance and environmental factors on the observed phenomena.

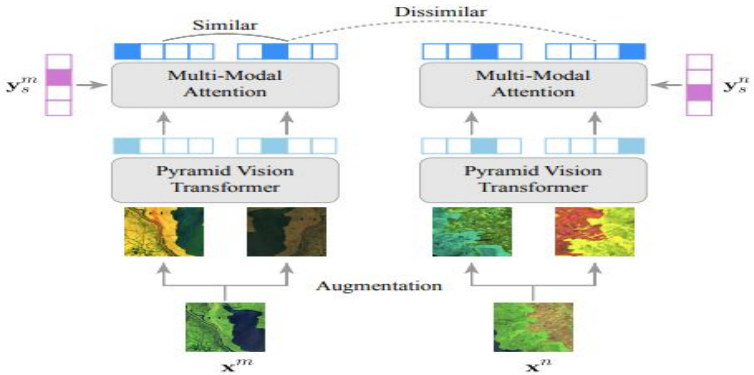


Figure 3: Multi-modal self-supervised learning.

Spatial Transformer

Modeled after vanilla ViT

County size (grid count) varies ->

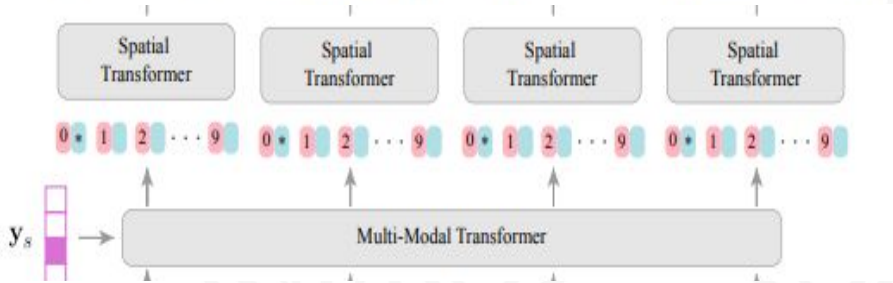
Flexible number of positional embeddings

Utilizes MultiModal output to learn the global spatial information of a county.

Learns high-resolution spatial dependency among counties for accurate agricultural tracking.

$$\text{S-MHA}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{Softmax}(\boldsymbol{Q}\boldsymbol{K}^T/\sqrt{d})\boldsymbol{V},$$
$$\boldsymbol{Q} = \boldsymbol{W}_s^Q \cdot \varphi_s(\boldsymbol{v}_m),\ \boldsymbol{K} = \boldsymbol{W}_s^K \cdot \varphi_s(\boldsymbol{v}_m), \quad (4)$$
$$\boldsymbol{V} = \boldsymbol{W}_s^V \cdot \varphi_s(\boldsymbol{v}_m).$$

$$\varphi_s(\boldsymbol{v}_m) = [\boldsymbol{v}_m^{\text{cls}};\ \boldsymbol{v}_m^1;\ \boldsymbol{v}_m^2;\ \ldots;\ \boldsymbol{v}_m^G] + \mathbf{E}_{\text{pos}},$$



Spatial Transformer

Overview:

- Captures spatial relationships
- Uses multi-head attention mechanism
- Treats input as having spatial dimensions

Key features:

- Content-based attention across spatial extent
- No additional positional or spatial bias
- Relies solely on provided content (input data) ->
- Generalized learning

Functionality:

- Can project output back to input dimension
- Includes dropout for regularization

```python
class SpatialAttention(nn.Module):
    def __init__(self, dim, heads=8, dim_head=64, dropout=0.):
        super().__init__()
        inner_dim = dim_head * heads
        project_out = not (heads == 1 and dim_head == dim)

        self.heads = heads
        self.scale = dim_head ** -0.5

        self.to_qkv = nn.Linear(dim, inner_dim * 3, bias=False)

        self.to_out = nn.Sequential(
            nn.Linear(inner_dim, dim),
            nn.Dropout(dropout)
        ) if project_out else nn.Identity()

    def forward(self, x):
        b, n, _, h = *x.shape, self.heads
        qkv = self.to_qkv(x).chunk(3, dim=-1)
        q, k, v = map(lambda t: rearrange(t, 'b n (h d) -> b h n d', h=h), qkv)

        dots = einsum('b h i d, b h j d -> b h i j', q, k) * self.scale
        attn = dots.softmax(dim=-1)

        out = einsum('b h i j, b h j d -> b h i d', attn, v)
        out = rearrange(out, 'b h n d -> b n (h d)')
        out = self.to_out(out)
        return out
```

Temporal Transformer

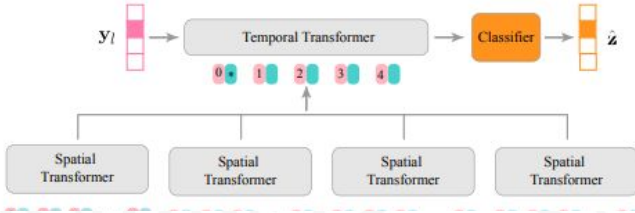
Combines the outputs of the Spatial Transformer and y_l to capture global temporal information AND the impact of long-term climate change on crop yields.

Incorporates a relative meteorological bias into each head for similarity computation.

Captures long-range temporal dependency for learning the impact of long-term climate change on crops.

$$\text{T-MHA}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{Softmax}(\boldsymbol{Q}\boldsymbol{K}^{\boldsymbol{T}}/\sqrt{d} + \pi_t(\boldsymbol{y}_l))\boldsymbol{V},$$
$$\boldsymbol{Q} = \boldsymbol{W}_t^Q \cdot \varphi_t(\boldsymbol{v}_s),\ \boldsymbol{K} = \boldsymbol{W}_t^K \cdot \varphi_t(\boldsymbol{v}_s),$$
$$\boldsymbol{V} = \boldsymbol{W}_t^V \cdot \varphi_t(\boldsymbol{v}_s). \quad (6)$$

$$\varphi_t(\boldsymbol{v}_s) = [\boldsymbol{v}_s^{\text{cls}};\ \boldsymbol{v}_s^1;\ \boldsymbol{v}_s^2;\ \ldots\ ;\ \boldsymbol{v}_s^T] + \mathbf{E}_{\text{tmp}}$$



Temporal Transformer

Overview:

- Captures temporal relationships in input data
- Uses temporal multi-head attention mechanism
- Designed for inputs with a temporal dimension

Key feature:

- Optional bias term capability
- Can inject temporal or positional information
- Suitable for tasks where input order/timing matters

Additional functionality:

- Includes output projection
- Incorporates dropout for regularization

```python
class TemporalAttention(nn.Module):
    def __init__(self, dim, heads=8, dim_head=64, dropout=0.):
        super().__init__()
        inner_dim = dim_head * heads
        project_out = not (heads == 1 and dim_head == dim)

        self.heads = heads
        self.scale = dim_head ** -0.5

        self.to_qkv = nn.Linear(dim, inner_dim * 3, bias=False)

        self.to_out = nn.Sequential(
            nn.Linear(inner_dim, dim),
            nn.Dropout(dropout)
        ) if project_out else nn.Identity()
```

Output

The output of the Temporal Transformer is ultimately used by a linear classifier for predicting the annual crop yields.

$$\hat{z} = W^T v_t + b,$$

Key Differences

Standard ViT Attention

- Uses fixed-size image patches as tokens
- Incorporates learned positional embeddings
- Retains spatial information but not spatial dependencies

Spatial-Temporal / PVT

Window-based attention starts at the local/momentary level and then contextualizes extracted representations with global information (e.g., notices cars before noticing traffic patterns over a highway's 24-hour period).

Spatial Attention

- Operates on pre-processed spatial data
- Does not use explicit positional encoding
- Relies on inherent spatial structure of input

Temporal Attention

- Similar to spatial attention
- Includes an optional bias term
- Incorporates temporal or positional information explicitly
- More suitable for sequences or time-series data

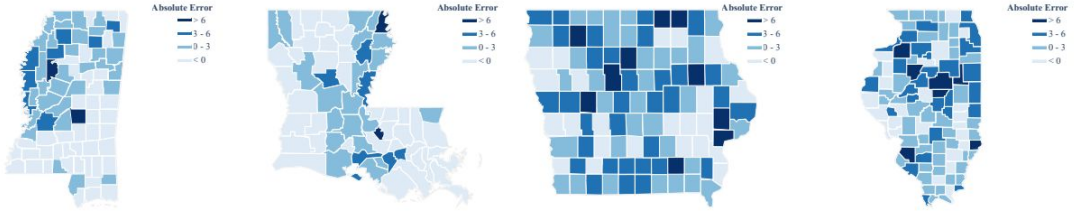
Experiments

Predict 2021 crop yields at the county level across four U.S. states, with prediction performance measured by three metrics.

RMSE: Measures how far predicted values are from actual values. Lower is better.

R^2: Shows how much of the outcome can be explained by the model. Ranges from 0 (no explanation) to 1 (perfect explanation).

Correlation: Indicates how closely two variables are related. Ranges from -1 (perfect negative) to 1 (perfect positive).



(a) Mississippi (MS) (b) Louisiana (LA) (c) Iowa (IA) (d) Illinois (IL)

Figure 4: Illustration of absolute prediction errors for soybean across four U.S. states. Note that a county/parish with "< 0" indicates its soybean yield data is unavailable.

Model Settings (Hyperparameters)

Similarly to SimCLR:

Random cropping, random horizontal flipping, random Gaussian blur, and color jittering allow data augmentation for pre-training and generalize unseen data (variation in lighting, contrast, luminosity. etc.)

After pre-training, we fine-tune MMST-ViT for 100 epochs using AdamW with $\beta 1 = 0.9$, $\beta 2 = 0.999$, a cosine decay schedule with an initial learning rate of $1e-3$, and warmup epochs of 5.

Table 2: Model details used in our study

Model		Layer	Hidden Size	Head	MLP Size	Others
Multi-Modal Transformer	PVT-T/4	{2, 2, 2, 2}	{64, 128, 320, 512}	{1, 2, 5, 8}	{512, 1024, 1280, 2048}	SR Ratios = {8, 4, 2, 1}
	MM-MHA	2	512	8	2048	Context Size = 9
Spatial Transformer	S-MHA	4	512	3	2048	-
Temporal Transformer	T-MHA	4	512	3	2048	Context Size = 9

Comparative Performance Evaluation

Table 3: Overall comparative crop yield predictions for 2021, with the best results shown in bold. Cotton yields are measured in pounds per acre (LB/AC), whereas other crop yields are measured in bushels per acre (BU/AC)

Method	Corn			Cotton			Soybean			Winter Wheat		
	RMSE (↓)	R^2 (↑)	Corr (↑)	RMSE (↓)	R^2 (↑)	Corr (↑)	RMSE (↓)	R^2 (↑)	Corr (↑)	RMSE (↓)	R^2 (↑)	Corr (↑)
ConvLSTM	18.6	0.611	0.782	65.4	0.715	0.846	7.2	0.616	0.785	7.4	0.511	0.715
CNN-RNN	14.6	0.705	0.839	69.5	0.653	0.808	5.8	0.703	0.839	7.5	0.614	0.783
GNN-RNN	14.2	0.730	0.854	58.5	0.647	0.804	5.4	0.748	0.865	6.0	0.621	0.788
Ours	**10.5**	**0.811**	**0.900**	**42.4**	**0.790**	**0.889**	**3.9**	**0.843**	**0.918**	**4.6**	**0.785**	**0.886**

Comparative Performance Evaluation

ConvLSTM overlooks the impact of meteorological parameters on crop growth.

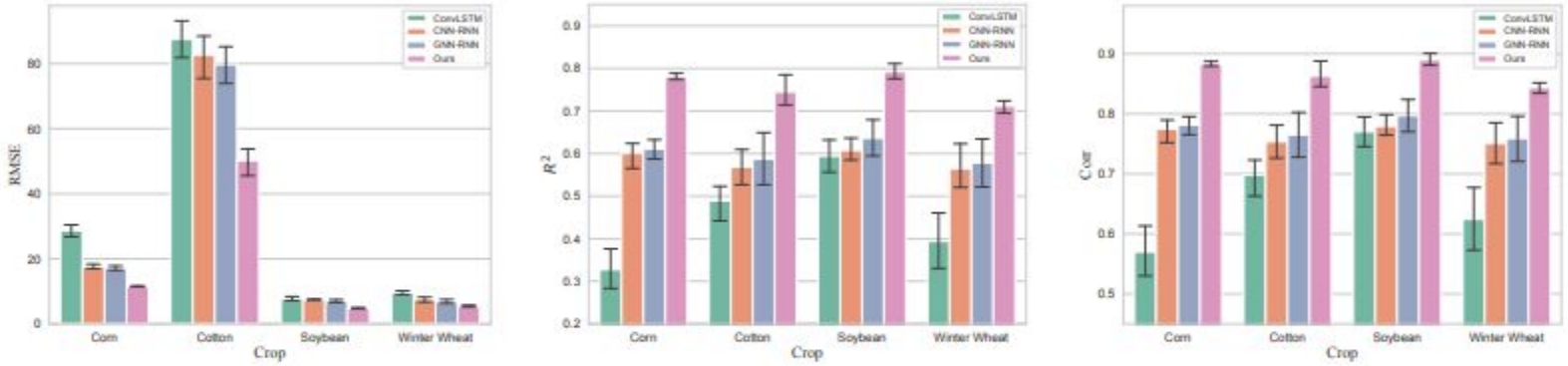
CNN-RNN cannot model the spatial dependency among neighborhood regions.

GNN-RNN only considers the meteorological data.

MMST-ViT outperforms the most recent state-of-the-art (i.e., GNN-RNN) measurably.

Visual remote sensing and numerical meteorological data. (Multimodal!)

Comparative Performance Evaluation



(a) RMSE (b) R^2 (c) Corr

Figure 5: Illustration of the performance for predictions of one year ahead under (a) RMSE, (b) R^2, and (c) Corr, with the cotton yield measured by LB/AC and other crop yields measured by BU/AC.

Conclusion

- The paper introduces the Multi-Modal Spatial-Temporal Vision Transformer (MMST-ViT), which integrates remote sensing and meteorological data for more accurate crop yield prediction.
- MMST-ViT includes three core components: **Multi-Modal Transformer**, **Spatial Transformer**, and **Temporal Transformer**, each utilizing a unique Multi-Head Attention (MHA) mechanism to process both visual and numerical data.
- The model effectively captures both short-term weather variations and long-term climate change, contributing to more reliable crop predictions.
- Extensive experiments on over 200 U.S. counties demonstrate that MMST-ViT consistently outperforms state-of-the-art models in terms of RMSE, R-squared (R^2), and Pearson Correlation Coefficient.

Conclusion

- The introduction of a multi-modal contrastive learning technique allows MMST-ViT to pre-train without requiring extensive human supervision, addressing overfitting concerns.
- The model's application can help in improving agricultural decision-making processes by providing more accurate and timely crop yield forecasts

New Ideas

1. Transfer Learning Across Geographies
2. Meta-Learning for Fast Adaptation
3. Few-shot Learning

References:

- MMST-ViT: Climate Change-aware Crop Yield Prediction via Multi-Modal Spatial-Temporal Vision Transformer
  https://arxiv.org/pdf/2309.09067
- An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale
  https://arxiv.org/abs/2010.11929
- A GNN-RNN Approach for Harnessing Geospatial and Temporal Information: Application to Crop Yield Prediction
  https://arxiv.org/pdf/2111.08900
- An Open and Large-Scale Dataset for Multi-Modal Climate Change-aware Crop Yield Predictions
  https://arxiv.org/html/2406.06081v1
- Pyramid Vision Transformer: A Versatile Backbone for Dense Prediction without Convolutions
  https://arxiv.org/pdf/2102.12122
- A Simple Framework for Contrastive Learning of Visual Representations https://arxiv.org/pdf/2002.05709

Q&A