Self-Supervised Learning using simCLR and Multi-learn model

Report: Self-Supervised learning model using Contrastive and Multi learning with SimCLR and VICReg.

By : Vedant Jain

Git-hub Repo Link

My Introduction

Name - VEDANT JAIN

COLLEGE - IIT(ISM) DHANBAD

SKILLS - Python , C++ , JupyterNotebook , Tableau , Numpy , Pandas , Matplotlib, Seaborn , Docker and Excel

Hobbies - I like to invest my time in Crypto-market analysis and NFT meta on web3 .

"Dear Deepcraft.ai Team,

I hope this message finds you well. I am pleased to submit the completed project as per outlined objectives. This project represents my dedication and effort to align with Deepcraft.ai's vision and goals.

The final deliverables include:

- **[Report for the Chief AI Engineer Project]**
- **[Google Collab Notebook and Git-hub Link]**

I appreciate the opportunity to contribute to your exciting initiatives, and I look forward to your feedback.

Thank you for the opportunity to work on this project."

Content Overview -:

1 Background Study of Self-Supervised Learning (Methods and Challenges)

2 Data set Analysis (Cleaning and Handling Missing values)

3 Technology Overview (Model Study)

4 Evaluation Indicators

5 Verification Results

6 Summary and Future Outlook

"WHAT IS SSL ???"

Self-supervised learning is a type of machine learning where the model learns to predict parts of the input data from other parts, without relying on labeled data

It generates its own labels from the data itself by setting up tasks that allow the model to learn useful representations.

Pre-Task Creation

A pretext task is designed where the model is given an incomplete or corrupted version of the input and is tasked with predicting the missing or altered part. This helps the model learn useful features from the data itself.

Representation Learning

The model learns representations (features) of the data by solving the pretext task. During this stage, the model is not concerned with specific labels but focuses on learning patterns, structures, or characteristics from the raw input data.

Self-supervision (Label Generation)

Self-supervision involves using the structure or parts of the data itself to create pseudo-labels. This is the key part of SSL: the model generates its own supervisory signals.

Fine-Tuning (Transfer Learning)

The learned features from the pretext task are transferred and adapted to improve performance on the downstream task.

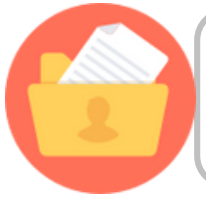
Challenges to be Faced:

Designing Effective Pretext Tasks

The effectiveness of SSL heavily depends on the pretext task used to generate labels.

Scare Unlabelled and Unclean Data:

Data available is less compared to Supervised learning and available data is also not ideally clean so we have to do pre-task i.e is cleaning and classification before going for modelling



Representation Collapse and Overfitting:

SSL models can suffer from representation collapse, where the learned representations do not differentiate well between samples, making the model less effective in downstream tasks.

Optimization and Stability Issues

The absence of well-defined labels makes it more difficult to evaluate model progress and stability.

Transferability to Real-World Applications:

While SSL methods often perform well in research settings, they can struggle to transfer to real-world, production-level applications that involve diverse, noisy, or dynamic data.

REPORT WILL FOCUS ON

SimCLR(Under contrastive Learning) and VICReg(Under Multi Instance-Learning)

SimCLR: (CL)

Batch Pair Funtion

$$l_{i,j} = -\log \frac{\exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{k \neq i} \exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_k)/\tau)} \quad (1)$$

where $\text{sim}(\cdot)$ represents the cosine similarity and $\mathbb{1}_{k \neq i} \in [0, 1]$ corresponds to 1 if $k \neq i$, otherwise 0. The loss is further tunable with an additional temperature parameter τ. The final overall loss is then composed of all positive pairs

Algorithm

Algorithm 1: SimCLR-TS Algorithm

Input: labeled or partially labeled Dataset $\mathcal{D} = \mathcal{D}_{labeled} \cup \mathcal{D}_{unlabeled}$, Set of Augmentations $\mathcal{A}$, models $f_\psi(\cdot)$, $g_\phi(\cdot)$, contrastive loss $l(\cdot)$, cross-entropy loss $XE(\cdot)$

for *epoch in* $epochs_{cl}$ **do**
  for *batch in* $\mathcal{D}$ **do**
    $\tilde{x}_i, \tilde{x}_j = augment(batch)$, $augment(\cdot) \in \mathcal{A}$
    $h_i = f_\psi(\tilde{x}_i)$
    $h_j = f_\psi(\tilde{x}_j)$
    $\psi = \psi + \nabla l(h_i, h_j)$
  end
end
$f(\cdot)$ =freeze_weights($f_\psi(\cdot)$)
for *epoch in* $epochs_{classifier}$ **do**
  for *batch. label in* $\mathcal{D}_{labeled}$ **do**
    $y_i = g_\phi(f(batch))$
    $\phi = \phi + \nabla XE(y_i, label_i)$
  end
end

Illustration

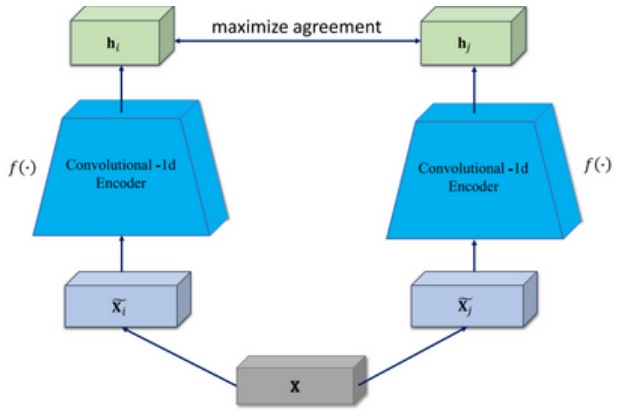


Fig. 2. SimCLR-TS structure.

Sources: Contrastive learning based self-supervised time-series analysis [Ref 1]

REPORT WILL FOCUS ON

SimCLR(Under contrastive Learning) and VICReg(Under Multi Instance-Learning)

VIReg: (MIL)

Batch Pair Funtion

$$s\left(Z^A, Z^B\right) = \frac{1}{n}\sum_{b=1}^{n}\left\|z_b^A - z_b^B\right\|_2^2. \quad (14)$$

In addition, the covariance criterion $c(Z)$ in VICReg is defined as

$$c(Z) = \frac{1}{d}\sum_{i\neq j}[C(Z)]_{i,j}^2, \quad (15)$$

where $C(Z)$ represents the covariance matrix of Z. The overall loss of VICReg is a weighted sum of the variance, invariance, and covariance:

$$\mathcal{L} = s\left(Z^A, Z^B\right) + \alpha\left(v\left(Z^A\right) + v\left(Z^B\right)\right) + \beta\left(C\left(Z^A\right) + C\left(Z^B\right)\right) \quad (16)$$

$$v\left(Z^A\right) = \frac{1}{d}\sum_{j=1}^{d}\max(0, \gamma - S\left(z_j^A, \varepsilon\right)). \quad (12)$$

Here, z_j^A represents the vector composed of each value at dimension j in Z^A and S represents the regularized standard deviation, defined as

$$S(y, \varepsilon) = \sqrt{\mathrm{Var}(y) + \varepsilon}. \quad (13)$$

Barlow Twins

BarlowTwins introduced a novel loss function that encourages thesimilarityof embeddingvectors from distorted versions of an examplewhileminimizing re dundancybetweentheircomponents.

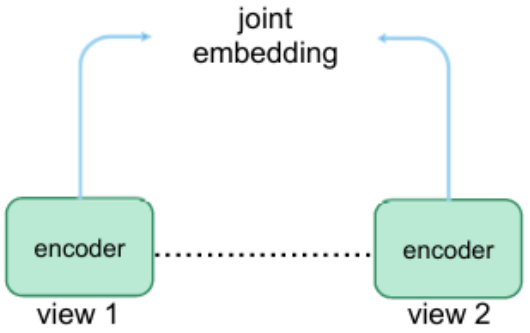
Z^A and Z^B. The loss function of Barlow Twins is defined as

$$\mathcal{L}_{BT} = \sum_i (1 - C_{ii})^2 + \lambda\sum_i\sum_{j\neq i} C_{ij}^2. \quad (10)$$

Here, λ is a hyper-parameter, and C represents the cross-correlation matrix computed between the two batches of embeddings Z^A and Z^B, defined as

$$C_{ij} = \frac{\sum_b z_{b,i}^A z_{b,j}^B}{\sqrt{\sum_b \left(z_{b,i}^A\right)^2}\sqrt{\sum_b \left(z_{b,j}^B\right)^2}}, \quad (11)$$

Illustration



Sources: https://arxiv.org/pdf/2301.05712 [Ref 2]

Pre-Task ??

Context Based Pretask

Context-based methods rely on the inherent contextual relationships among the provided examples, encompassing aspects such as spatial structures and the preservation of both local and global consistency.

We will play with Data to understand how it will play out in training the model.

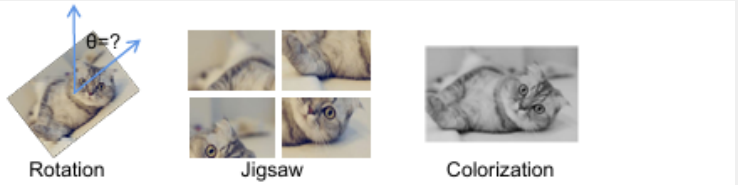
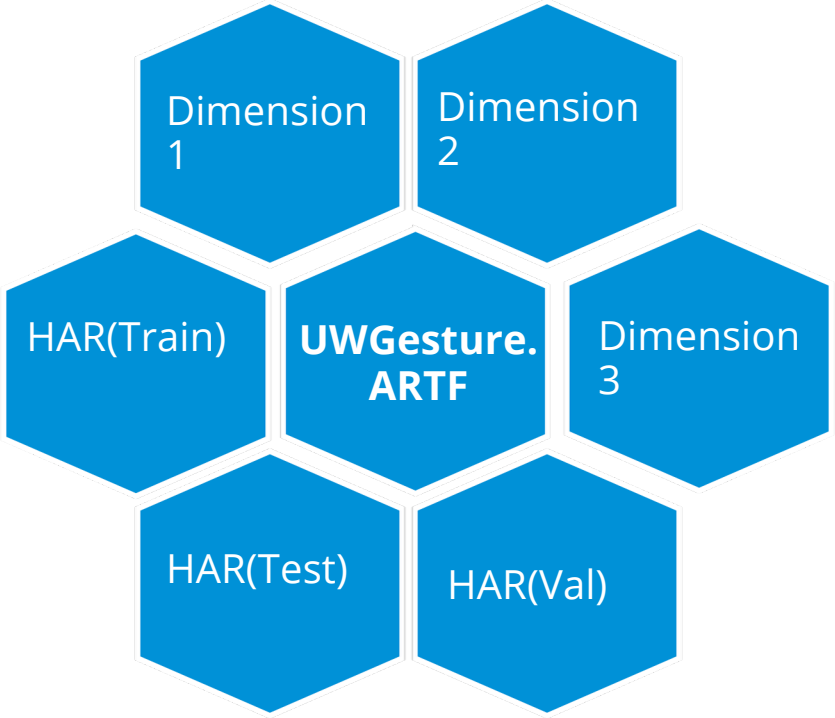


Fig. 4: Illustration of three common context-based methods: rotation, jigsaw, and colorization.

Data Set Charecterstics



Model Trained on UWGesture Library and Fine-Tuned on HAR data set

PRE-TASK PRE-PROCESSING

```python
# Decode byte strings if needed
for column in df.select_dtypes([object]):
    df[column] = df[column].apply(lambda x: x.decode('utf-8') if isinstance(x, bytes) else x)

# Convert any numpy.ndarray values to lists
for column in df.columns:
    df[column] = df[column].apply(lambda x: x.tolist() if isinstance(x, np.ndarray) else x)

# Fill missing values using forward fill and backward fill
df = df.ffill().bfill()
```

Conversion and Backfilling

Backward and Forward fill techniques -This is used to fill the missing values for both Numeric and Non-numeric data because Median and mean etc are not valid for non-numeric data.

```python
def random_cropping(series, crop_length=None):
    """
    Crops a random sub-section of the time series.
    """
    if crop_length is None or crop_length > len(series):
        crop_length = len(series) // 2  # Default to half the length
    start = np.random.randint(0, len(series) - crop_length + 1)
    cropped_series = np.zeros_like(series)
    cropped_series[start:start+crop_length] = series[start:start+crop_length]
    return cropped_series
```

Random Cropping

Missing values are common in time series datasets to fix this we apply a smoothing filter to reduce noise, emphasizing overall trends.

```python
def time_warp(series, warp_factor=0.2):
    """
    Stretches or compresses the time series by a random factor within the specified range.
    """
    stretch_factor = 1 + (warp_factor * (np.random.rand() * 2 - 1))  # random factor between (1-warp_factor) and (1+warp_factor)
    warped_series = np.interp(np.arange(0, len(series), stretch_factor), np.arange(0, len(series)), series)
    return warped_series[:len(series)]  # Ensure same length after warping
```

Time Wrap

For Time series data we Stretches or compresses parts of the series, introducing temporal variation.

```python
def jitter(series, noise_level=0.05):
    """
    Adds small random noise to each point in the series.
    """
    noise = np.random.normal(0, noise_level, len(series))
    return series + noise
```

Jittering

Jittering helps the model generalize better by learning to ignore irrelevant fluctuations and focus on the core patterns in the data. In SSL pretext tasks, jittered versions of the original time series can be used as positive pairs for contrastive learning, training the model to recognize similar patterns despite minor random noise

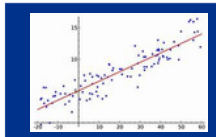
```python
# Ensure numeric columns are of a correct type for scaling
numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns
if len(numeric_columns) > 0:
    scaler = StandardScaler()
    df[numeric_columns] = scaler.fit_transform(df[numeric_columns])
```

Data Scaling and Normalization

Normalization (Min-Max Scaling): Scales the data to a range between 0 and 1.

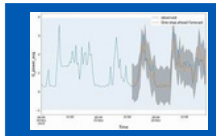
Method Selection

Total 6 Models were used to find best fit for Time Series Dataset



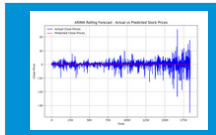
SimCLR (CL)

Uses contrastive learning to maximize agreement between augmented views of the same time series.



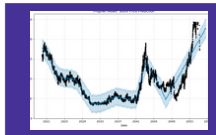
VIReg (MIL)

Minimizes variance, invariance, and covariance losses to enforce diverse and invariant representations.



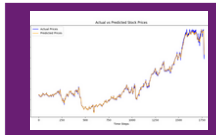
Generative Adversarial Network (GAN)

A Generative Adversarial Network (GAN) creates new, altered views of images to help a contrastive learning model learn view-invariant features



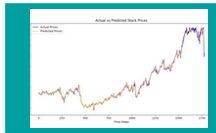
TTT Method

If Test Data is very diffrent from Train dataset works. "combining with MAE it works even better because it predicts the missing info in data first."



CLIP (Multi Modality)

CLIP is a multi-modal model designed to align images and text in a shared representation space, enabling open-vocabulary image recognition and image-text matching.



Masked Image Modelling

MIM is a self-supervised learning technique for visual data that aims to improve representation learning by "reconstructing" parts of an image based on other parts.

MASKED IMAGE MODELLING

Drawback: High computational cost and often disrupts the data as it selects 70% data randomly for masking

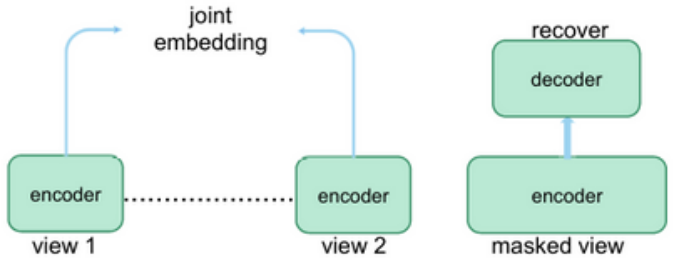


Fig. 7: The broad differences between CL and MIM. Note that the actual differences between their pipelines are not limited to what is shown.

No Augmentation i.e. no division of data

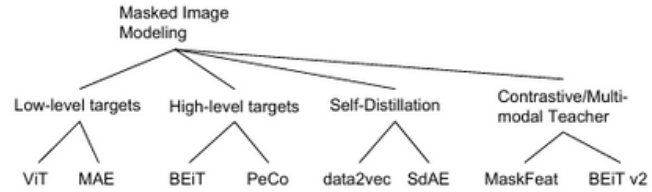
40-70% of data is masked and rest is trained to encode it which helps in learning about charecterstics of data

BEiT: Like BERT in NLP, BEiT splits the image into smaller parts (tokens) and hides some. Then, it tries to predict these missing parts using visual tokens. they use pixel target tokens are target tokens to train model. they usually work on small portion of image gone like 25%.used on larger datasets. generally efficienct and accurate for complex and more avability of data tasks

MAE - This is a simpler model that hides large parts of an image (up to 75%) and uses a decoder to reconstruct the missing areas directly from the pixels.They use image directly as pixels. In this major portion of image is not there 75%. Used on small dtatsets generally. generally efficienct and accurate for less complex and less avability of data tasks

2 Major ALGOs (BEit and MAE)

BEiT used for Larger Dataset whereas MAE used for Small Datasets



VideoMAE and OmniMAE are used in video and concurrent model training .

Illustration of Branched MIM

```
# Initialize masks and targets
mask = torch.zeros(batch_size, seq_length, dtype=torch.bool)
masked_data = data.clone()
target = data.clone()

for i in range(batch_size):
    # Randomly select indices to mask
    mask_indices = np.random.choice(seq_length, num_masked, replace=False)
    mask[i, mask_indices] = True
    masked_data[i, :, mask_indices] = 0  # Masking by setting the segment to zero
```

MAE does not utilize image tokens; instead, it approaches the problem from the perspective of image signal sparsity.

CLIP METHOD

Drawback: Does not capture Temporal relationships and often dependent on natural language descriptions

Multimodal - used diffrent data source like text,image and video etc to train the model (used in timeseries). used in premium quality task because have to deal with noises in all data sources. techniques involve - Fusion, Cross-modal learning and attention mechnaism.(CLIP's advancements have significantly propelled multimodal learning)

Multi-Modality Learning

Multi-modal(ality) learning converge naturally in infants' learning mechanisms

```python
class CLIPModel(nn.Module):
    def __init__(self, ts_input_dim, ts_hidden_dim, ts_output_dim, text_input_dim, text_output_dim
        super(CLIPModel, self).__init__()
        self.ts_encoder = TimeSeriesEncoder(ts_input_dim, ts_hidden_dim, ts_output_dim)
        self.text_encoder = TextEncoder(text_input_dim, text_output_dim)

    def forward(self, ts_data, text_data):
        ts_embedding = self.ts_encoder(ts_data)
        text_embedding = self.text_encoder(text_data)
        return ts_embedding, text_embedding

    def compute_loss(self, ts_embedding, text_embedding, temperature=0.07):
        # Normalize embeddings
        ts_embedding = F.normalize(ts_embedding, dim=1)
        text_embedding = F.normalize(text_embedding, dim=1)

        # Compute similarity matrix
        similarity_matrix = torch.matmul(ts_embedding, text_embedding.T) / temperature

        # Labels for contrastive loss
        batch_size = ts_embedding.size(0)
        labels = torch.arange(batch_size).to(ts_embedding.device)
```

CLIP leverages a CL-style pre-training task to predict the correspondence between captions and images. Benefiting from the CL paradigm, CLIP is capable of training models from scratch on an extensive dataset comprising 400 million image-text pairs collected from the internet. Consequently, CLIP's advancements have significantly propelled multi modal learning to the forefront of research attention.

As of 2024 There is still study being conducting on this so this is not very implementable as of now and also this is the least accurate in terms of Time series data

GAN (Generative Adversarial Network)

objective function [35], [143] is given as

$$\min_G \max_D V(G,D) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1 - D(G(z)))]. \quad (18)$$

The SS-GAN [144] is defined by combining the objective functions of GANs with the concept of rotation [7]:

$$L_G(G,D) = -V(G,D) - \alpha \mathbb{E}_{x\sim p_G}\mathbb{E}_{r\sim R}[\log Q_D(R=r|x^r)], \quad (19)$$

$$L_D(G,D) = V(G,D) - \beta \mathbb{E}_{x\sim p_{data}}\mathbb{E}_{r\sim R}[\log Q_D(R=r|x^r)], \quad (20)$$

Objective Funtion of SSL GAN Called SS-GAN

Generator Loss and Discriminator Loss is depicted . Generator tries to generate as similar image as possible to positive pair and Discriminator tries to distinguish the generated image as hard as possible

Drawback: Insensitive to Hyperparameters and Model Collapsing

TABLE 3: Different losses of SSL.

Category		Method	Loss	Equation
Pretext	Context-Based	Rotation [7]	Rotation Prediction	(3)
	CL	MoCo v1 [50]	InfoNCE	(5)
		SimCLR v1 [52]	InfoNCE	(6)
		SimSiam [69]	Cosine Similarity	(9)
		Barlow Twins [55]	Invariance, and Covariance	(10)
		VICReg [56]	Variance, Invariance, and Covariance	(16)
Combinations with Other Learning Paradigms		SS-GAN [144]	GAN loss + Rotation Prediction	(19 & 20)
		S^4L [145]	Supervised and Unsupervised Loss	(21)
		SSL improving robustness [146]	Supervised and Self-supervised Adversarial Training Loss	(22)
		unsupervised multi-view learning [64]	Self-supervised Loss on Multiple Views	(25)

Diffrent Losses Table in SSL

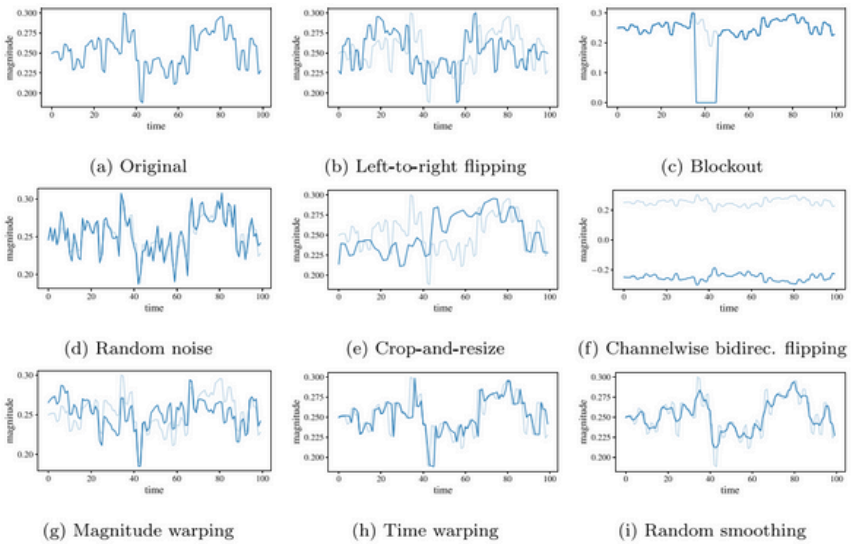
```python
# Generator Network
class Generator(nn.Module):
    def __init__(self, noise_dim, hidden_dim, output_dim, seq_length):
        super(Generator, self).__init__()
        self.fc1 = nn.Linear(noise_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, hidden_dim * seq_length // 2)
        self.fc3 = nn.Linear(hidden_dim * seq_length // 2, output_dim * seq_length)
        self.output_dim = output_dim
        self.seq_length = seq_length

    def forward(self, z):
        x = torch.relu(self.fc1(z))
        x = torch.relu(self.fc2(x))
        x = torch.tanh(self.fc3(x))
        return x.view(-1, self.output_dim, self.seq_length)

# Discriminator Network
class Discriminator(nn.Module):
    def __init__(self, input_dim, hidden_dim, seq_length):
        super(Discriminator, self).__init__()
        self.fc1 = nn.Linear(input_dim * seq_length, hidden_dim * seq_length // 2)
        self.fc2 = nn.Linear(hidden_dim * seq_length // 2, hidden_dim)
        self.fc3 = nn.Linear(hidden_dim, 1)

    def forward(self, x):
        x = x.view(x.size(0), -1)  # Flatten time-series data
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        return torch.sigmoid(self.fc3(x))
```

CODE SNIPPET

Contrastive Methods leads to Overfitting and Generative methods leads to scalability issues so we devised new paradigm called Contrastive Generative Algorithm and GAN is part of that

SimCLR(Under CL)

Drawback: High Augmentation sensitivity and Cost



Augmentation Methods

1. **Left-to-Right Flipping:** Reverses the time sequence, adding robustness to sequence order.
2. **Blockout:** Randomly zeros out a section of the series to simulate missing data.
3. **Random Noise:** Adds noise scaled to each channel's variance, enhancing tolerance to natural variations.
4. **Crop and Resize:** Crops a random segment and interpolates to original length, helping focus on different subsequences.
5. **Magnitude Warping:** Varies amplitude using a sine wave pattern, simulating natural intensity fluctuations.
6. **Time Warping**: Stretches or compresses parts of the series, introducing temporal variation.
7. **Random Smoothing:** Applies a smoothing filter to reduce noise, emphasizing overall trends.

Results of Processes

MoCov2 builds upon **MoCo v1 and SimCLR v1** , incorporating a multilayer perceptron (MLP) projection head and more data augmentations.

Augmentation done in simCLR are good for categorization tasks but not for obejct recognization tasks. so strong aug. is must but relying on that only is not good as well. The distortions introduced by strong data augmentation can alter the image structure, resulting in a distribution that differs from that of weakly augmented images

```python
# Left-to-right flipping
def left_to_right_flip(series):
    return np.flip(series, axis=1)  # Flip across time axis (axis=1)

# Bidirectional flipping (mirroring across the time axis)
def bidirectional_flip(series):
    return -1 * series
```

$$\tilde{\mathbf{X}} = \mathbf{X}\Upsilon \text{ where } \quad \Upsilon_{i,j} = \begin{cases} 1, & \text{if}(j = T - i + 1) \\ 0, & \text{if}(j \neq T - i + 1) \end{cases}$$

$$\tilde{\mathbf{X}} = \Gamma\mathbf{X} \text{ where } \quad \Gamma_{i,j} = \begin{cases} 1, & \text{if}(j = m - i + 1) \\ 0, & \text{if}(j \neq m - i + 1) \end{cases}$$

```python
# Blockout
def blockout(series, block_size_ratio=0.1):
    T = series.shape[1]
    block_size = int(block_size_ratio * T)
    start = np.random.randint(0, T - block_size)
    blocked_series = series.copy()
    blocked_series[:, start:start + block_size] = 0  # Block out along the time axis
    return blocked_series
```

For the blockout augmentation, we randomly select an area of $\lambda \cdot m \in \mathbb{N}$ neighboring elements inside the time-series signal and set all values to zero. Starting in row k and column l, then

$$\tilde{x}_{i,j} = \begin{cases} 0 & \text{if } k \leq i \leq k + \lambda m \text{ and } l \leq j \leq l + \lambda m \\ x_{i,j} & \text{otherwise} \end{cases} \tag{8}$$

```python
# Random noise
def random_noise(series, noise_factor=0.1):
    sigma = np.std(series, axis=1, keepdims=True)  # Compute std for each channel
    noise = np.random.uniform(-1, 1, series.shape) * sigma * noise_factor
    return series + noise
```

$$\tilde{\mathbf{X}} = \mathbf{X} + \mathbf{R} \circ \lambda\Sigma, \quad \Sigma = \{\Sigma, \ldots, \Sigma\} \in \mathbb{R}^{m \times T} \tag{9}$$

where $\mathbf{X}$ denotes the original data and $\tilde{\mathbf{X}}$ the augmented data, Σ a column-vector composed of the per-channel standard deviations σ_i and $\circ$ and element-wise multiplication.

```python
def magnitude_warping(series, magnitude=0.1):
    """Warp the magnitude of the time series"""
    series = np.array(series, dtype=np.float32)  # Ensure it's a float32 array
    factor = 1 + np.random.uniform(-magnitude, magnitude)
    return series * factor


def time_warping(series, warp_factor=0.2):
    indices = np.arange(series.shape[1])
    warp_indices = np.random.uniform(0, warp_factor, series.shape[1])
    warped_series = np.interp(indices + warp_indices, indices, series)
    return warped_series

def random_smoothing(series, window_size=3):
    return np.convolve(series, np.ones(window_size)/window_size, mode='same')
```

$$H(z) = \frac{\lambda}{2}z^1 + (1 - \lambda) + \frac{\lambda}{2}z^{-1}.$$

$$\mathbf{X} = \mathbf{X} \circ \lambda\Psi, \quad \Psi = 1^{m \times T} + \sin\left(\begin{bmatrix} \frac{2\pi nt}{T} + \theta_1 \\ \vdots \\ \frac{2\pi nt}{T} + \theta_m \end{bmatrix}\right) \text{ where}$$

$$t \in [0, \ldots, T], \ \theta_i \in [0, \ldots, 2\pi].$$

Source : Ref1

VIReg (MIL)

Z^A and Z^B. The loss function of Barlow Twins is defined as

$$\mathcal{L}_{BT} = \sum_i (1 - C_{ii})^2 + \lambda \sum_i \sum_{j \neq i} C_{ij}^2. \quad (10)$$

Here, λ is a hyper-parameter, and C represents the cross-correlation matrix computed between the two batches of embeddings Z^A and Z^B, defined as

$$C_{ij} = \frac{\sum_b z_{b,i}^A z_{b,j}^B}{\sqrt{\sum_b \left(z_{b,i}^A\right)^2} \sqrt{\sum_b \left(z_{b,j}^B\right)^2}}, \quad (11)$$

where b indexes batch samples and i, j index the vector dimension of the networks' outputs. C is a square matrix that measures the correlation between the two batches of

BARLOW TWINS

Barlow Twins is a self-supervised learning (SSL) method that uses Reduce Reductancy approach Barlow Twins' objective is to make each dimension of the embedding vector carry unique information, reducing redundancy.

Drawback: Too many regulation terms and prone to collapse

VICReg builds on the Barlow Twins approach by introducing variance regularization to prevent representation collapse. It balances three key objectives:

1. **Variance:** Ensures that the standard deviation of each dimension in the embeddings stays above a threshold y, penalizing low variance to maintain diversity.
2. **Invariance:** Uses the mean squared Euclidean distance between embeddings to enforce similarity between augmented views, minimizing the distance for similar samples.
3. **Covariance**: Computes the covariance matrix of the embeddings and penalizes off-diagonal elements (correlations between dimensions), encouraging independent features.

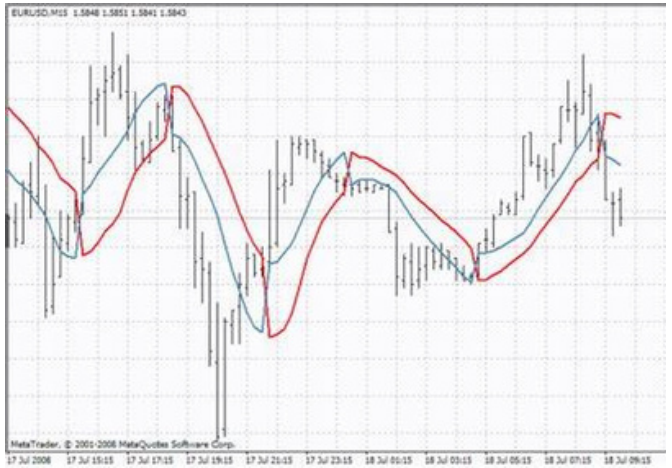
Training And Fine Tuning

- For each batch, two augmented views are generated.
- The embeddings of these views are passed through the VICReg loss.
- The model's parameters are updated to minimize this loss.

```python
# Fine-tuning loop
def fine_tune(model, dataloader, criterion, optimizer, epochs=10):
    model.train()

    for epoch in range(epochs):
        epoch_loss = 0.0
        correct_preds = 0
        total_preds = 0

        for inputs, labels in dataloader:
            inputs, labels = inputs.to(device), labels.to(device)

            # Forward pass
            outputs = model(inputs)
            loss = criterion(outputs, labels)

            # Backpropagation
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            # Track loss and accuracy
            epoch_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            correct_preds += (predicted == labels).sum().item()
            total_preds += labels.size(0)

        accuracy = 100 * correct_preds / total_preds
        print(f"Epoch [{epoch + 1}/{epochs}], Loss: {epoch_loss / len(dataloader):.4f}, Accuracy: {acc
```

Evaluation Indicators

While all methods have their strengths, SimCLR and VIReg stands out due to its ability to model complex relationships and temporal dynamics effectively.



Indicators	Description
Downstreamtask Performance	Downstream Task Evaluation: You test the model on an animal classification task (like identifying cats, dogs, and birds). If it performs well, you know it learned useful features.
k-NN classification	Accuracy of K-NN classification can indicate how well-separated the learned feature space is, without requiring an additional classifier to be trained.
Network Dissection	You examine the network and discover that certain parts of the model have learned to recognize animal fur, feathers, or even specific patterns like spots or stripes. This shows that SSL helped the model understand animal-specific features. this includes disintegrating each neuron to see what model learned

These indices help evaluate the effectiveness of SSL models in learning meaningful, transferable features that work well across different tasks, even when no labeled data was used during training.
Contrastive Loss is used in our Code snipet

Co-Relation Analysis

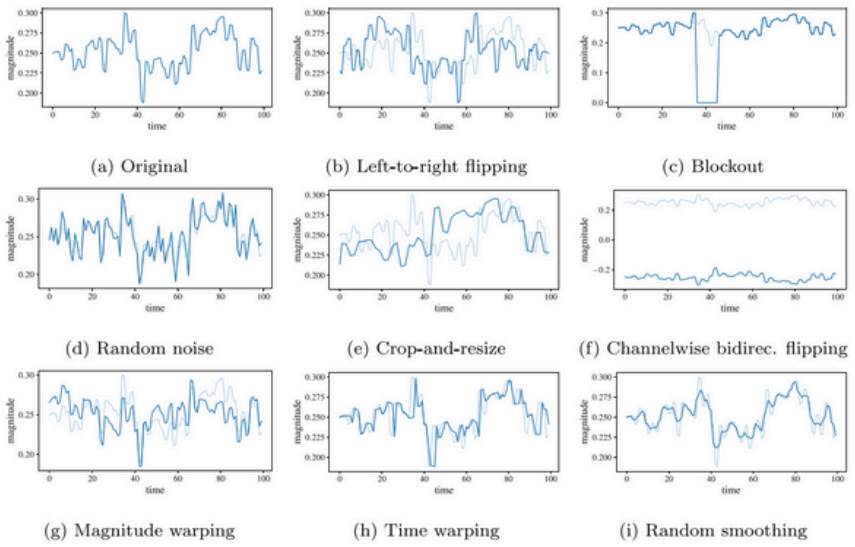
Use correlation coefficients to identify highly correlated features and retain only one feature from each highly correlated pair.

Expected Outcome:

Increased Predictive Accuracy: The model is expected to demonstrate improved accuracy metrics (e.g., lower MAE, MSE, RMSE) after applying feature selection.

Feature Importance

In fine-tuning, feature and weight importance help adapt a pre-trained model to a new task by focusing on relevant patterns and adjusting weights accordingly. Feature importance enables the model to prioritize patterns specific to the new data, such as trends in time series classification, while weight importance allows selective adjustment of the model's parameters. Lower layers often retain general patterns from pre-training, needing minimal change, while higher layers adapt more to task-specific nuances. This process optimizes the model's performance on the new dataset without completely retraining from scratch.



Expected Outcome:

Faster Training Time: The training process is anticipated to be quicker due to a reduced feature set, leading to more efficient experimentation.

Handling missing Values

Null Values: Missing data can lead to inaccurate predictions and biased results. Proper imputation methods (mean, median, mode, or advanced techniques like KNN imputation) can help retain valuable information.

Expected Outcome:

Models trained on datasets with appropriately handled missing values will demonstrate higher accuracy compared to those with unaddressed missing values.

Expected Outcome:

Models trained on datasets with removed outliers will yield lower error rates and higher predictive performance.

Outlier Removal

Impact of Outliers: Outliers can skew the results and mislead the model's learning process. Identifying and removing outliers using techniques like Z-score or IQR can stabilize the data distribution.

Normalization and Scaling

Normalizing or standardizing features (e.g., Min-Max scaling or Z-score normalization) ensures that all features contribute equally to model training, especially in algorithms sensitive to feature scales .

Expected Outcome:

- Models using normalized datasets will demonstrate better convergence and performance metrics compared to those using raw, unscaled features.

Expected Outcome:

Models trained on a carefully selected subset of features will have higher accuracy and lower complexity compared to models using all available features.

Summary

This project aims to enhance time series classification accuracy, specifically for the UWaveGestureLibrary dataset, using self-supervised learning (SSL). SSL leverages large-scale unlabeled data to learn useful feature representations, which can improve downstream classification performance. In this project, we pre-trained on a separate dataset using self-supervised methods VICReg and SimCLR. VICReg, with its variance, invariance, and covariance regularization, ensures robust and diverse feature extraction, while SimCLR, a contrastive learning approach, encourages similar representations for augmented views of the same instance. After pre-training, we fine-tuned the model on the UWaveGestureLibrary data to maximize classification accuracy, effectively adapting the learned features to the gesture recognition task. This approach helps improve model accuracy by using SSL-extracted representations that capture essential time series characteristics.

Pre-Processing Techniques and Outliers Removal

These techniques not only enhance the model's predictive accuracy but also ensure more reliable and robust forecasts by minimizing noise and irrelevant variations in the data. Here's how they shape the future outlook of:

```
# 1. **Smoothing using Rolling Window (moving average)**
df['Close_smoothed'] = df['Close'].rolling(window=5).mean()

# 2. **Outlier Removal using Z-Score**
z_scores = np.abs(stats.zscore(df['Close_smoothed'].fillna(df['Close'])))
df = df[(z_scores < 3)]  # Removing outliers where z-score > 3

# 3. **Scaling (MinMaxScaler) for features**
scaler = MinMaxScaler()
df[features] = scaler.fit_transform(df[features])
df[[target]] = scaler.fit_transform(df[[target]])
```

- **Smoothing methods (e.g., moving averages, exponential smoothing) help remove short-term fluctuations and highlight long-term trends in time series data**
- **Future Outlook: More sophisticated smoothing techniques like seasonal decomposition or wavelet transformations may be employed to decompose time series into trend, seasonal, and residual components before feeding them, enhancing trend prediction accuracy.**

Future Outlook

GAN and CLIP can definately be the Future Outlook

Methods	Linear Probe	Fine-Tuning	VOC_det	VOC_seg	COCO_det	COCO_seg	ADE20K_seg	DB
Random:	17.1_A [8]	-	60.2^e_R [69]	19.8_A [8]	36.7_R [50]	33.7_R [50]	-	-
R50 Sup	76.5 [68]	76.5 [68]	81.3^e [69]	74.4 [67]	40.6 [50]	36.8 [50]	-	-
ViT-B Sup	82.3 [70]	82.3 [70]	-	-	47.9 [70]	42.9 [70]	47.4 [70]	-
Context-Based:								
Jigsaw [8]	45.7_R [68]	54.7	61.4_R [42]	37.6	-	-	-	256
Colorization [38]	39.6_R [68]	40.7 [7]	46.9	35.6	-	-	-	-
Rotation [7]	38.7	50.0	54.4	39.1	-	-	-	128
CL Based on Negative Examples:								
Examplar [132]	31.5 [48]	-	-	-	-	-	-	-
Instdisc [48]	54.0	-	65.4	-	-	-	-	256
MoCo v1 [50]	60.6	-	74.9	-	40.8	36.9	-	256
SimCLR [52]	73.9_V [82]	-	81.8^e [69]	-	37.9 [69]	33.3 [69]	-	4096
MoCo v2 [51]	72.2 [69]	-	82.5^e	-	39.8 [56]	36.1 [56]	-	256
MoCo v3 [82]	76.7	83.2	-	-	47.9 [70]	42.7 [70]	47.3 [70]	4096
CL Based on Clustering:								
SwAV [68]	75.3	-	82.6^e [56]	-	41.6	37.8 [56]	-	4096
CL Based on Self-distillation:								
BYOL [67]	74.3	-	81.4^e [69]	76.3	40.4 [56]	37.0 [56]	-	4096
SimSiam [69]	71.3	-	82.4^e [69]	-	39.2	34.4	-	512
DINO [83]	78.2	83.6 [98]	-	-	46.8 [100]	41.5 [100]	44.1 [99]	1024
CL Based on Feature Decorrelation:								
Barlow Twins [55]	73.2	-	82.6^e [56]	-	39.2	34.3	-	2048
VICReg [56]	73.2	-	82.4^e	-	39.4	36.4	-	2048
Masked Image Modeling (ViT-B by default):								
Context Encoder [104]	21.0_A [7]	-	44.5_A [7]	30.0_A	-	-	-	-
BEiT v1 [99]	56.7 [111]	83.4 [98]	-	-	49.8 [70]	44.4 [70]	47.1 [70]	2000
MAE [70]	67.8	83.6	-	-	50.3	44.9	48.1	4096
SimMIM [101]	56.7	83.8	-	-	52.3_{Swin-B} [244]	-	52.8_{Swin-B} [244]	2048
PeCo [107]	-	84.5	-	-	43.9	39.8	46.7	2048
iBOT [98]	79.5	84.0	-	-	51.2	44.2	50.0	1024
MimCo [110]	-	83.9	-	-	44.9	40.7	48.91	2048
CAE [100]	70.4	83.9	-	-	50	44	50.2	2048
data2vec [108]	-	84.2	-	-	-	-	-	2048
SdAE [109]	64.9	84.1	-	-	48.9	43.0	48.6	768
BEiT v2 [111]	80.1	85.5	-	-	-	-	53.1	2048

By leveraging a self-supervised GAN (SS-GAN) approach, we could also use task-specific transformations—such as temporal shifts or rotations—to further enrich the model's feature learning. Integrating CLIP (Contrastive Language-Image Pretraining) in a multimodal setup, where textual or categorical data is paired with time series data, could enhance the model's contextual understanding. Although CLIP is typically used for vision-language tasks, adapting its principles to align time series signals with textual descriptions (like event labels or metadata) could improve the interpretability and classification accuracy of time-sensitive events. This combination of GAN-generated data and CLIP-inspired multi-modal learning could significantly enhance the model's ability to generalize across various time series classification tasks.

Here are two potential hypotheses based on your use of VICReg and SimCLR for training and fine-tuning on time series data:

Hypothesis 1: VICReg's Redundancy Reduction Enhances Feature Diversity for Time Series Data By leveraging VICReg’s focus on variance, invariance, and covariance, the model will produce a more robust set of diverse features that capture different aspects of the time series data. This should lead to improved classification accuracy when fine-tuned on the UWaveGestureLibrary dataset, as it helps avoid redundancy and collapse in feature representation.

Hypothesis 2: SimCLR's Augmentation and Contrastive Learning Will Improve Distinction of Gesture Classes The contrastive learning approach in SimCLR, which relies on strong data augmentations, should encourage the model to differentiate between gesture classes more effectively. Through augmentations like jittering or time-based shifts, SimCLR should help the model learn invariant representations that improve its ability to classify similar yet distinct time series patterns.