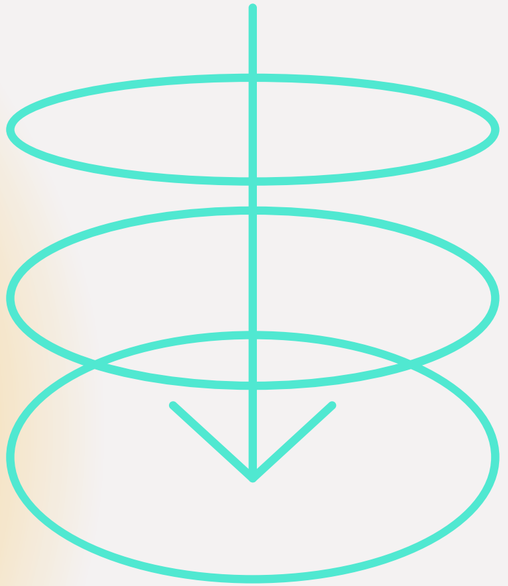
Deep Learning for change detection in remotely sensed images

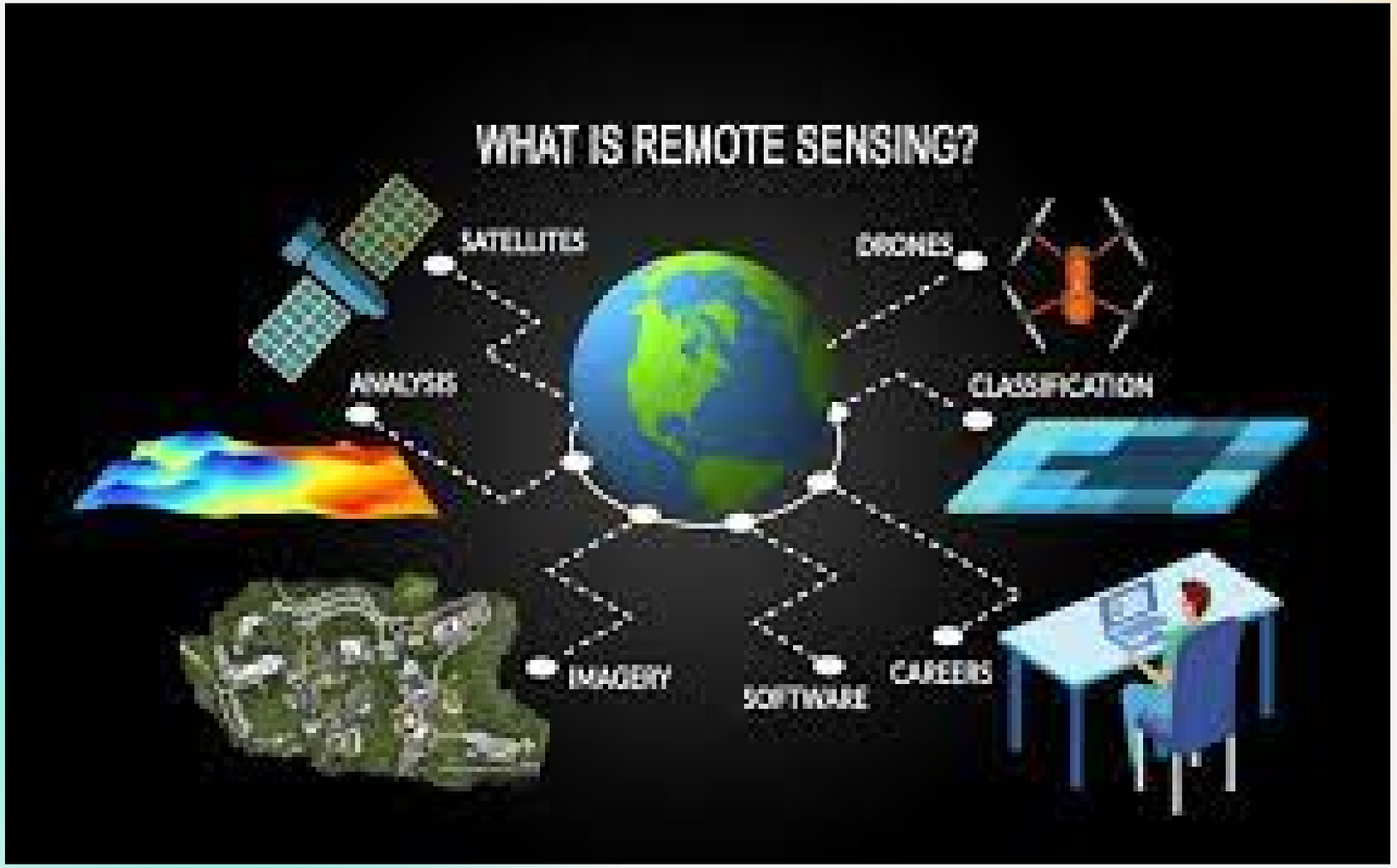


2001095 Uma Kadam

What is Remote Sensing ?

- Remote sensing is the collection of data about the Earth's surface and atmosphere using instruments that are not in direct contact with the Earth's surface.
- Provides valuable information for environmental monitoring, natural resource management, and other applications.
- Remote sensing can be done using satellites, airplanes, drones, and ground-based sensors.
- Remote sensing provides detailed, accurate, and up-to-date information about large areas of the Earth.

REMOTE SENSING



What is change detection?

Change Detection in Remotely Sensed Images

- Change detection is the process of identifying and analyzing differences between two or more satellite images of the same area taken at different times.
- Remotely sensed satellite images are used to detect and monitor changes in land use, land cover, vegetation, etc over time.
- Change detection can be used for a variety of applications, including disaster response, urban planning, natural resource management, and environmental monitoring.
- Change detection can reveal important information about the rate and extent of change in a given area, and can help decision-makers better understand and respond to environmental challenges.

Traditional Change Detection Techniques

There are 2 types of traditional Change tection Methods:

- *Pixel Based Change Detection :*

Detect changes between two or more remotely sensed images by comparing individual pixels

- *Object Based Change Detection:*

Segmenting the information based on spectral and spatial information

Traditional CD Methods

Algebra Based techniques:

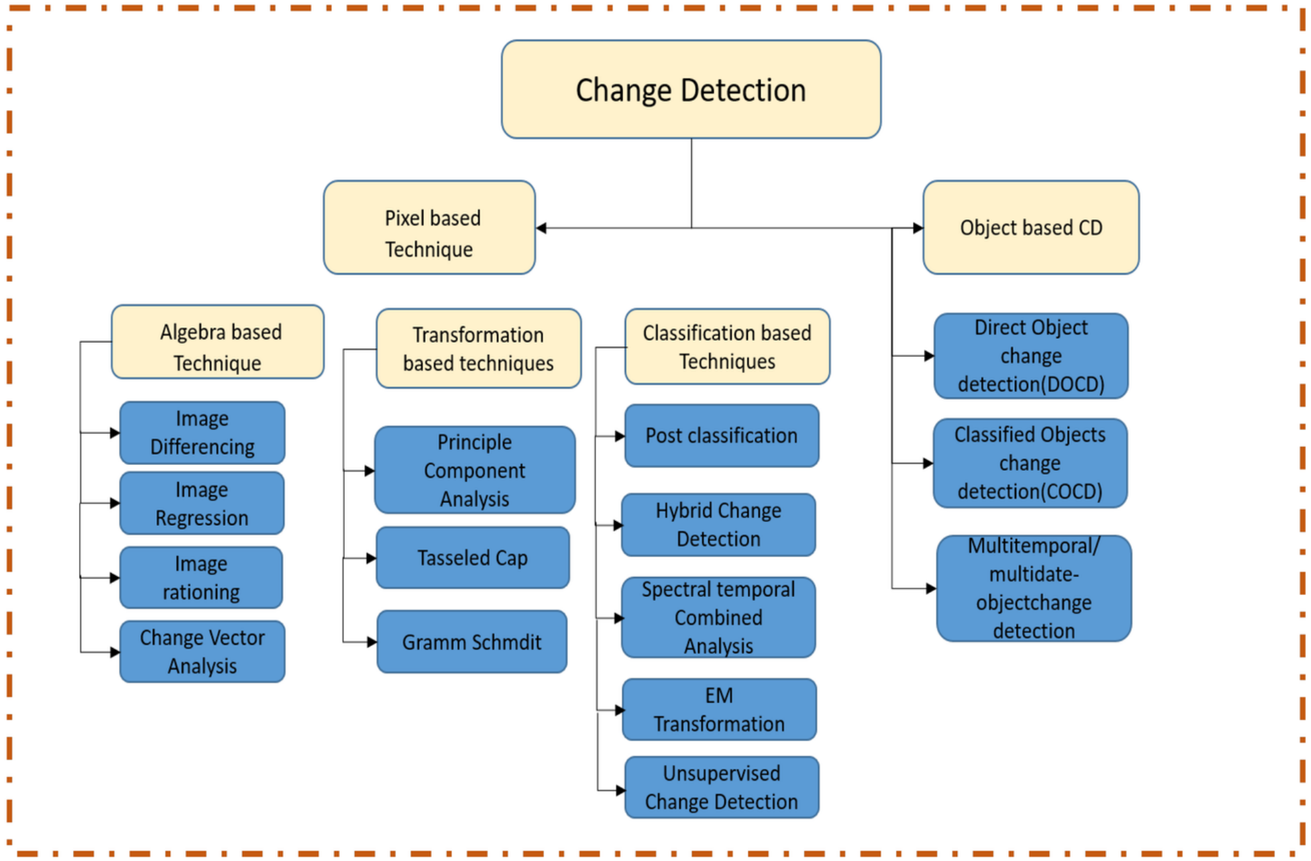
Use mathematical operations to detect changes between two or more remotely sensed images.

Transformation Based Methods:

Transform remotely sensed images into a different domain or representation to facilitate the detection of changes

Classification Based Methods:

Detect changes between two or more satellite images by comparing the classifications of the images



Deep Learning for Change Detection

What is DLCD ? Why is it needed?

- DLCD uses advanced deep learning algorithms, to automatically detect changes between remotely sensed images.
- It is able to learn complex features and patterns from large datasets, allowing it to identify subtle changes in the landscape that may be missed by traditional change detection methods.
- Involves training a neural network on large dataset of labelled images and can detect changes in unlabeled images of new dataset

Deep Learning Based Change Detection Methods

Siamese Neural Network

CNN

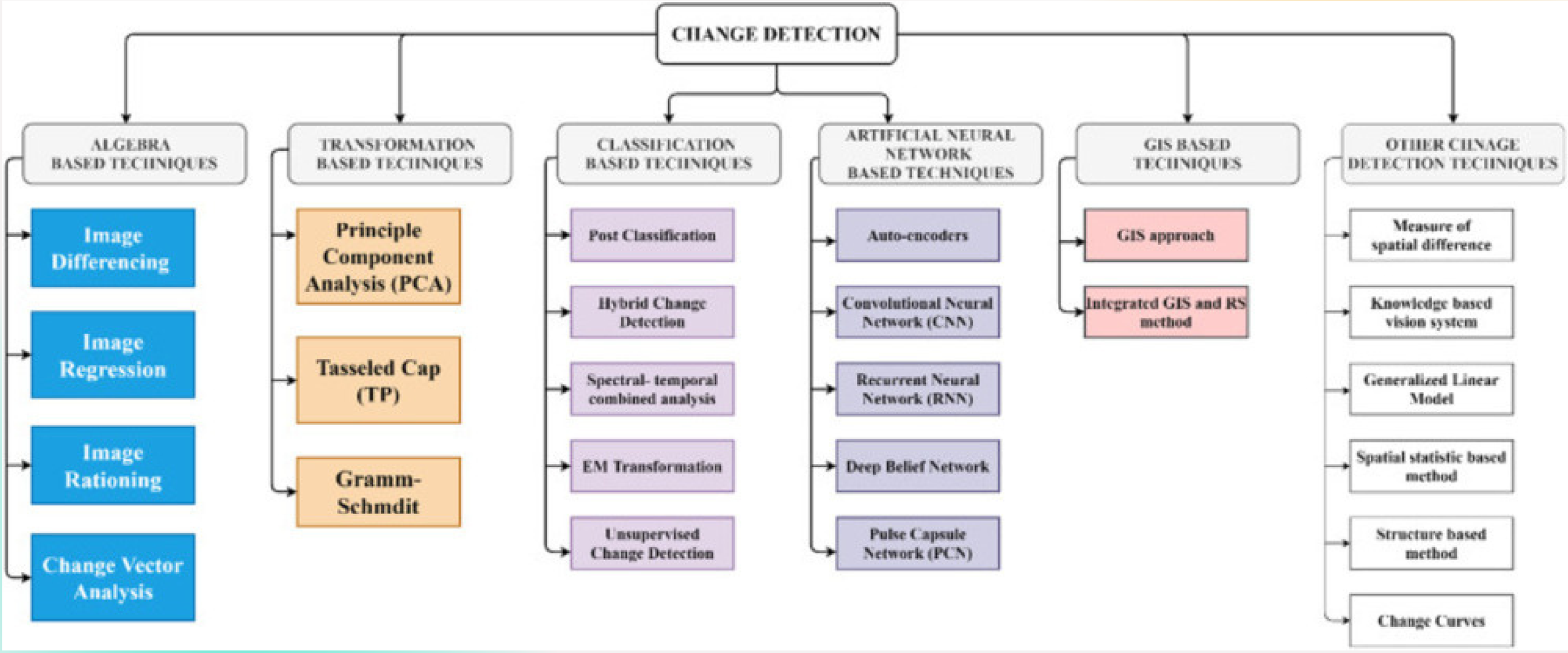
RNN

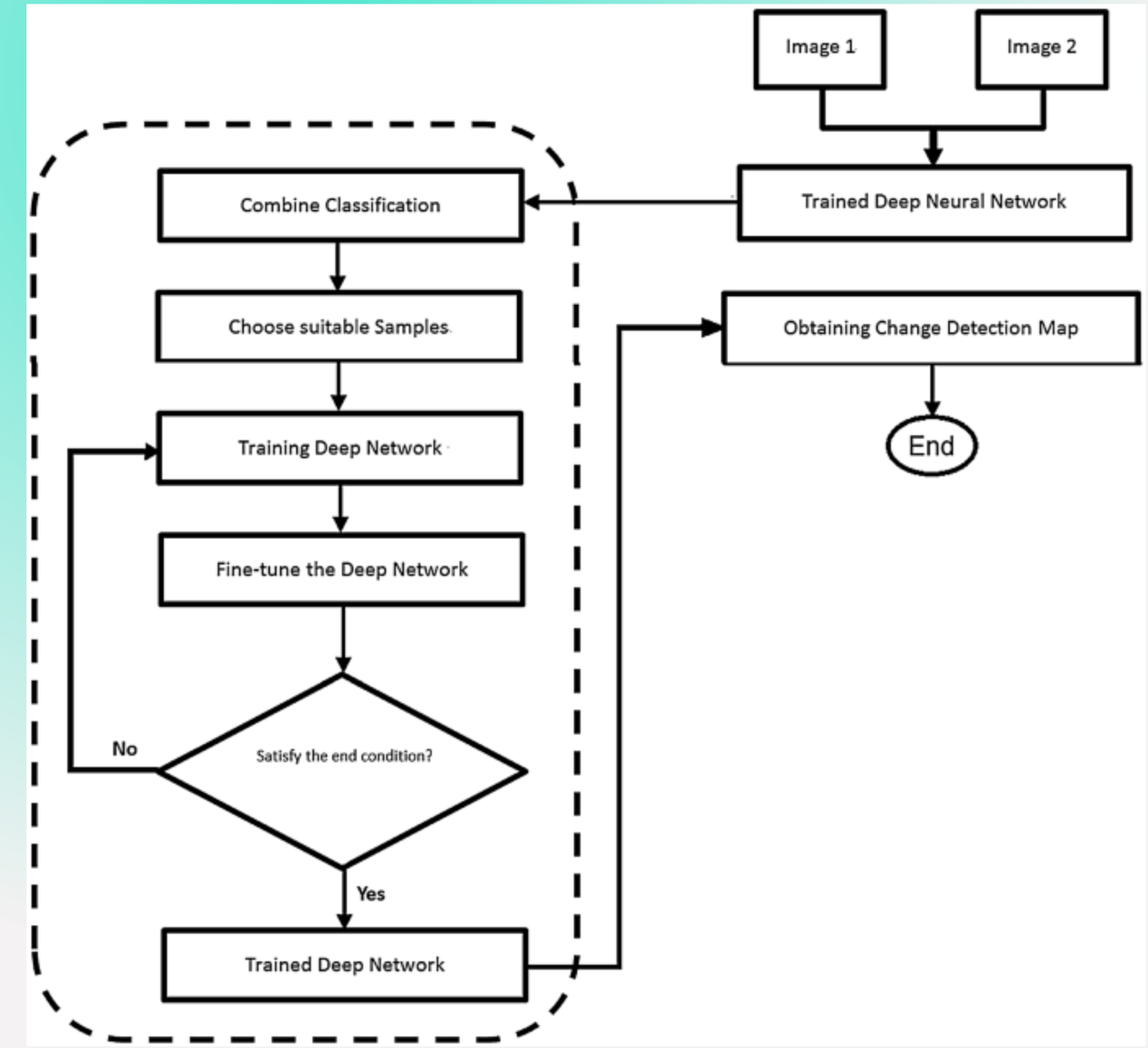
Deep Belief Network

GAN

Autoencoders

Deep Belief Networks


CHANGE DETECTION
ALGEBRA BASED TECHNIQUES
Image Differencing
Image Regression
Image Rationing
Change Vector Analysis
TRANSFORMATION BASED TECHNIQUES
Principle Component Analysis (PCA)
Tasseled Cap (TP)
Gramm-Schmdit
CLASSIFICATION BASED TECHNIQUES
Post Classification
Hybrid Change Detection
Spectral- temporal combined analysis
EM Transformation
Unsupervised Change Detection
ARTIFICIAL NEURAL NETWORK BASED TECHNIQUES
Auto-encoders
Convolutional Neural Network (CNN)
Recurrent Neural Network (RNN)
Deep Belief Network
Pulse Capsule Network (PCN)
GIS BASED TECHNIQUES
GIS approach
Integrated GIS and RS method
OTHER CHNAGE DETECTION TECHNIQUES
Measure of spatial difference
Knowledge based vision system
Generalized Linear Model
Spatial statistic based method
Structure based method
Change Curves



- Collect and preprocess satellite images
- Create training and testing datasets
- Choose a deep learning architecture
- Train the deep learning model:
- Evaluate the deep learning model:
- Apply the deep learning model to new data

Methodology For DLCD Implementation in Change Detection

Solution Methods

01 - Convolutional Neural Network

02 - Fully Convolutional Siamese Network

03 - SNUNetCD

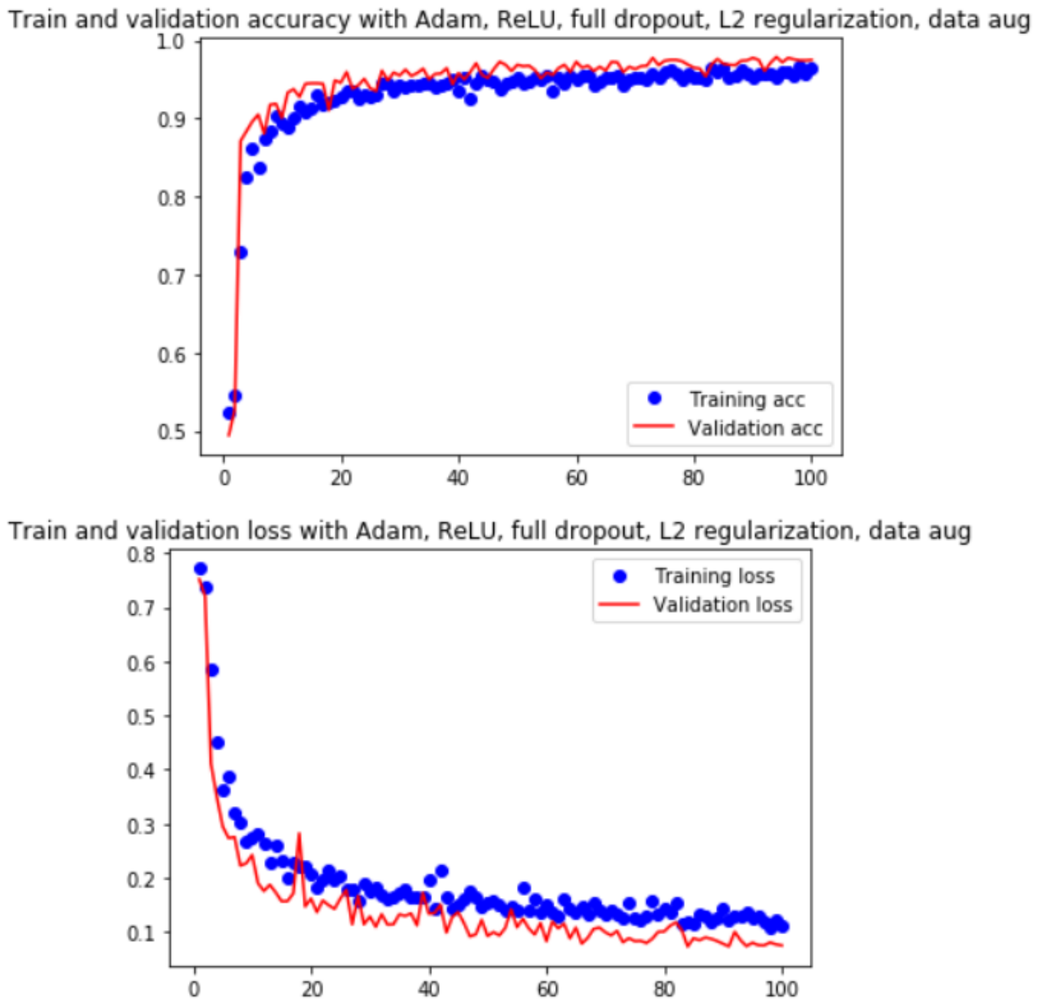
Issues in existing methods

- Incase of disasters the traditional methods prove to be time-consuming and inefficient.
- Loss of localization information in deep layers of neural networks
- Uncertainty of pixels at the edge of changed targets

Convolutional Neural Network

CNN

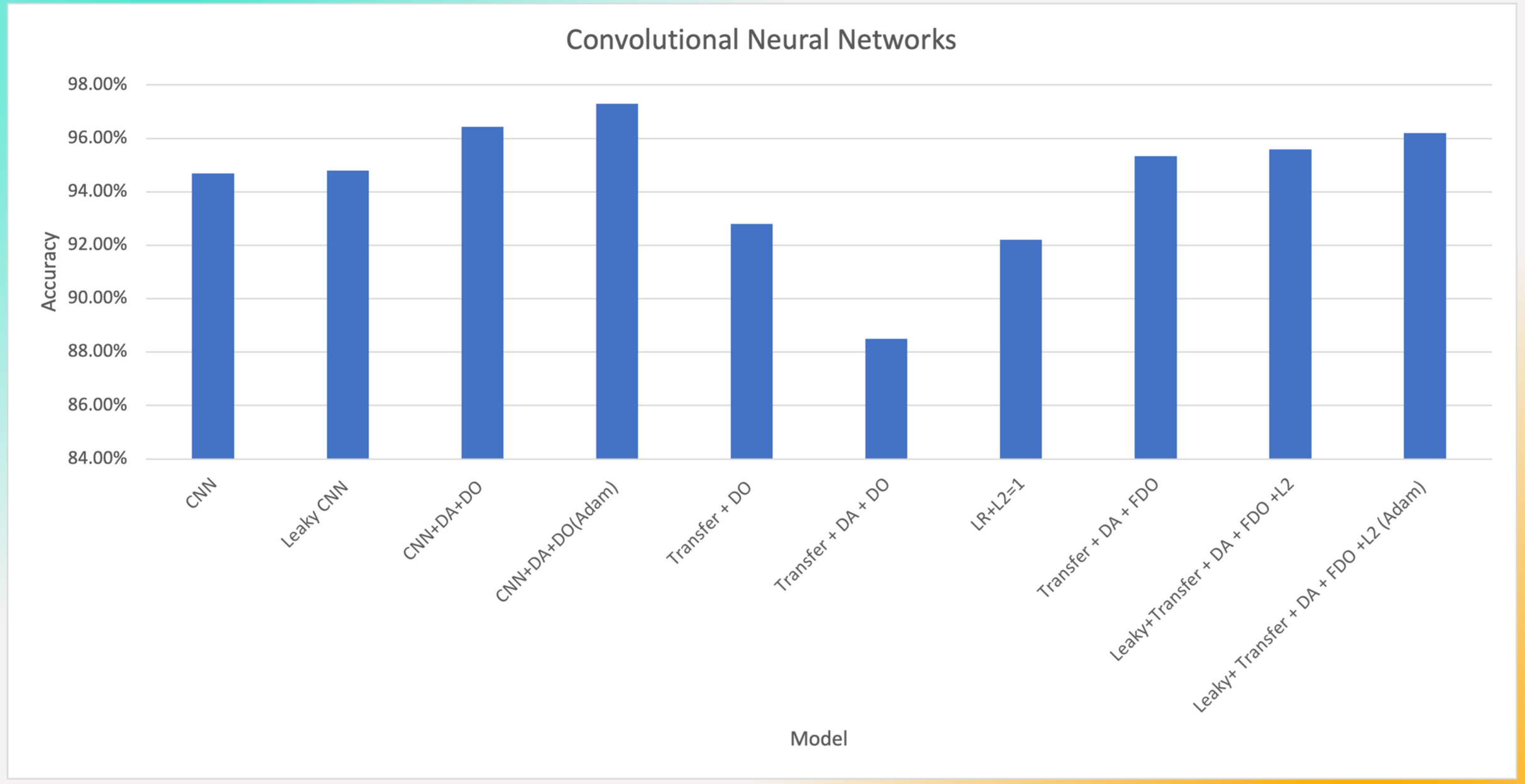
- A Convolutional Neural Network (CNN) is a deep learning algorithm that is widely used for image and video processing tasks.
- It consists of multiple layers of filters that extract important features from the input image.
- These extracted features are then processed by fully connected layers to classify the image.
- This has solved the time consuming process of feature engineering and extraction



Methodology

- Data was processed to eliminate images that were partially black or covered by clouds to ensure high quality images
- The images were projected into a consistent feature dimension using a process called featurization. Eg (128x128) into (150x150)
- Usage of VGG-16 to stop when there are too many “dead” filters.
- The featurized images were fed through a CNN to extract features such as edge detection and other relevant information for damage detection.
- The model was trained using a binary cross-entropy loss function and optimized using the Adam optimizer.
- A logistic regression on the featurized data to see how it compares to a densely connected layer and find that it performs better than a dense layer

```
#another grand model with just relu
from keras.layers import LeakyReLU
from keras.regularizers import l2
model = models.Sequential()
model.add(layers.Conv2D(32,(3,3), activation = 'relu', input_shape = (150,150,3)))
# model.add(LeakyReLU(alpha=0.1))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Dropout(0.25))

model.add(layers.Conv2D(64,(3,3), activation = 'relu'))
#model.add(LeakyReLU(alpha=0.1))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Dropout(0.25))

model.add(layers.Conv2D(128,(3,3), activation = 'relu'))
#model.add(LeakyReLU(alpha=0.1))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Dropout(0.25))

model.add(layers.Conv2D(128,(3,3), activation = 'relu'))
#model.add(LeakyReLU(alpha=0.1))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Dropout(0.25))

model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(512, activation = 'relu',kernel_regularizer = l2(1e-4)))
#model.add(LeakyReLU(alpha=0.1))
model.add(layers.Dense(1, activation = 'sigmoid'))
#compile the model with RMSprob with learning rate
from keras import optimizers
model.compile(loss = 'binary_crossentropy', optimizer = optimizers.RMSprop(lr=1e-4), metrics = ['acc'])

#process the jpeg image
#create an image generator
from keras.preprocessing.image import ImageDataGenerator
```


Convolutional Neural Networks
98.00%
96.00%
94.00%
92.00%
90.00%
88.00%
86.00%
84.00%
Accuracy
CNN
Leaky CNN
CNN+DA+DO
CNN+DA+DO(Adam)
Transfer + DO
Transfer + DA + DO
LR+L2=1
Transfer + DA + FDO
Leaky+Transfer + DA + FDO +L2
Leaky+ Transfer + DA + FDO +L2 (Adam)
Model

Fully Convolutional Siamese Networks

01 - What is it?

- A Fully Convolutional Network (FCN) is a type of neural netwo
  used for image processing
- Unlike traditional convolutional networks, which are designed for classification tasks, FCNs use only convolutional and pooling layers to transform the input image into an output map of the same size.

Necessity?

02 - Benefits

- Spatial Information Preservation
- Multi-scale Processing
- Transfer Learning: Pretrained FCN models
- Real-time Performance:

Methodology

FC-SIAM-DIFF && FC-SIAM-CONC

- The network takes an input image and outputs a pixel-wise mask indicating the class of each pixel.
- The network consists of several convolutional layers, each followed by batch normalization, dropout, and a Rectified Linear Unit (ReLU) activation function. The architecture has a contracting path (encoder) and an expanding path (decoder), which are connected by skip connections.
- Skip connections are concatenated during the decoding phase in FC-SIAM-CONC whereas the absolute value of their difference is concatenated in FC-SIAM-DIFF
-

- Usage of transformers can improve FCNN.

They provide the following benefits:

- Improved long-range dependency modeling
- Better handling of variable-length inputs
- Fine-grained feature extraction
- Improved transfer learning

Improvements to Fully Convolutional Siamese Models

SNUNet-CD

Issue

- Loss of localization information in deep layers of neural networks
- Uncertainty of pixels at the edge of changed targets

SNUNet-CD addresses these issues by using dense skip connections between encoder and decoder, and between decoder and decoder to maintain high-resolution, fine-grained representations. This alleviates the uncertainty of pixels at the edge of changed targets and determination miss of small targets.

New future

Methodology

Concatenation of Siamese Network and Nested Unet

- The network architecture is a combination of Siamese network and NestedUNet, which leverages both deep and shallow-layer information for acccurate change detection

- The Siamese network is used to extract features from two input images (reference and test) simultaneously, which are then concatenated and fed into the NestedUNet.

- The NestedUNet consists of an encoder-decoder structure with dense skip connections between encoder and decoder, and between decoder and decoder. This allows for compact information transmission between layers, maintaining high-resolution, fine-grained representations.

Improvements to the SNUNET-CD model

Ensemble Learning

- .In the context of siamese nested U-Net models, one way to use ensemble learning is to train multiple models with different hyperparameters or on different subsets of the data.

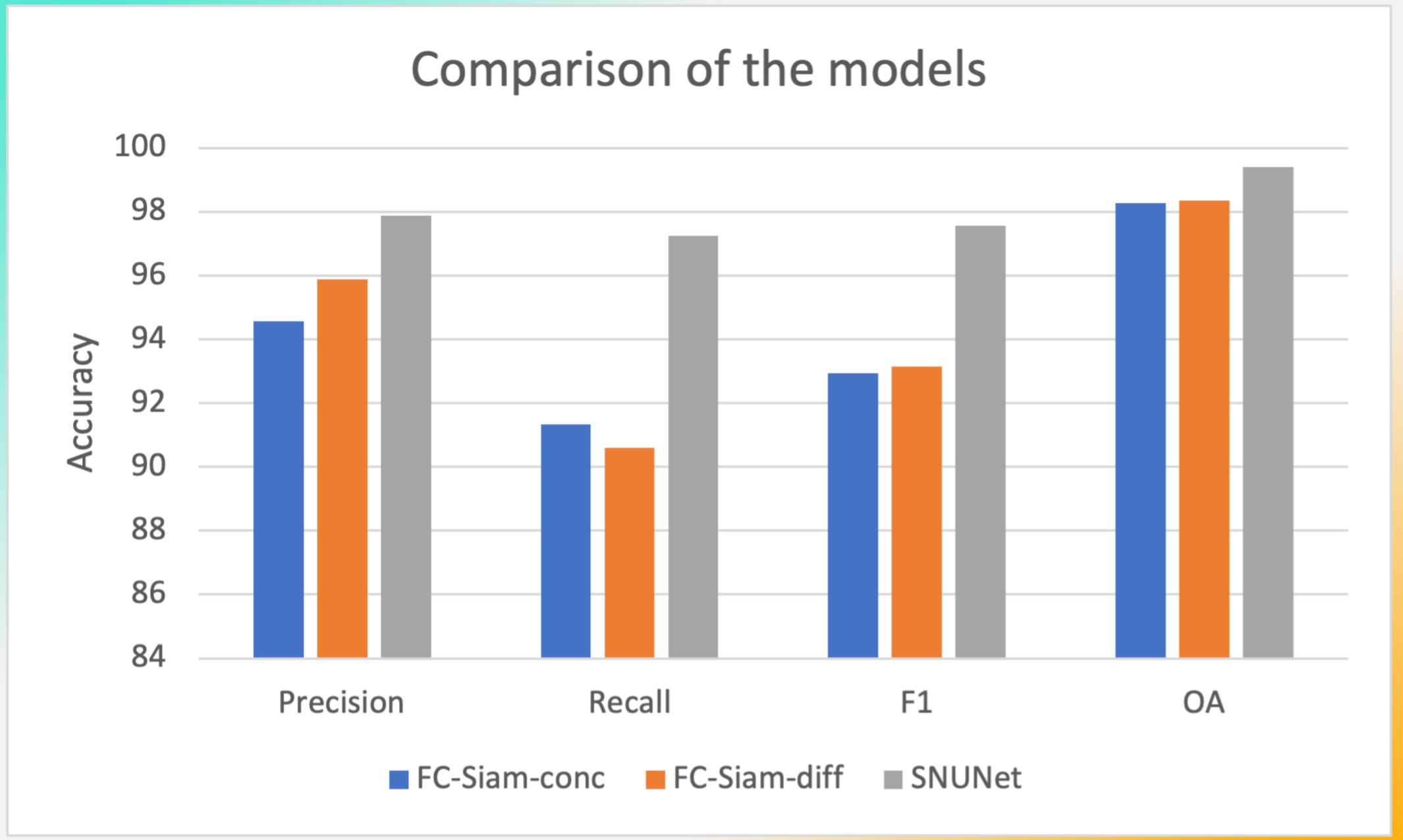
- These models can then be combined using various techniques such as averaging or voting to make a final prediction.

- Thus we can improve the overall performance of the model, increase its robustness to different hyperparameters and data subsets, and reduce the risk of overfitting.

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.nn.modules.padding import ReplicationPad2d

class SiamUnet_conc(nn.Module):
    """SiamUnet_conc segmentation network."""

    def __init__(self, input_nbr, label_nbr):
        super(SiamUnet_conc, self).__init__()

        self.input_nbr = input_nbr

        self.conv11 = nn.Conv2d(input_nbr, 16, kernel_size=3, padding=1)
        self.bn11 = nn.BatchNorm2d(16)
        self.do11 = nn.Dropout2d(p=0.2)
        self.conv12 = nn.Conv2d(16, 16, kernel_size=3, padding=1)
        self.bn12 = nn.BatchNorm2d(16)
        self.do12 = nn.Dropout2d(p=0.2)

        self.conv21 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
        self.bn21 = nn.BatchNorm2d(32)
        self.do21 = nn.Dropout2d(p=0.2)
        self.conv22 = nn.Conv2d(32, 32, kernel_size=3, padding=1)
        self.bn22 = nn.BatchNorm2d(32)
        self.do22 = nn.Dropout2d(p=0.2)
```

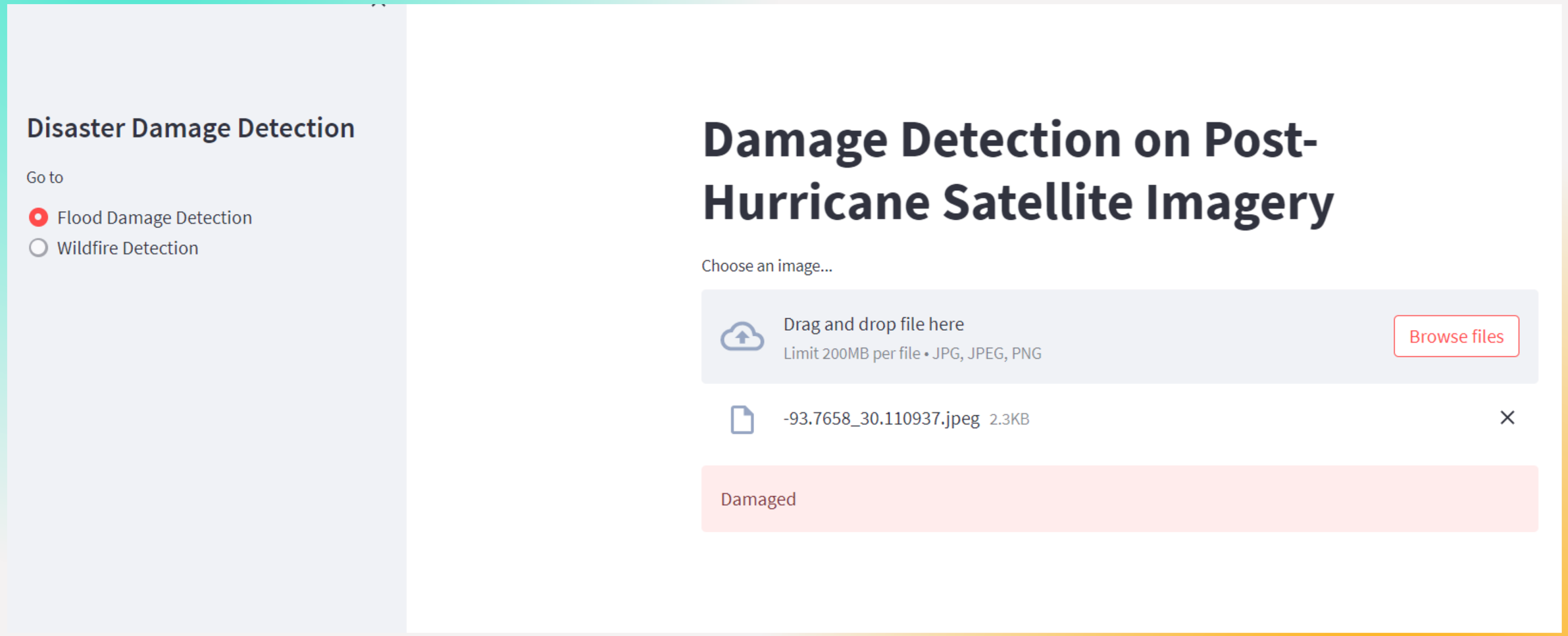
```python
def forward(self, x1, x2):

    """Forward method."""
    # Stage 1
    x11 = self.do11(F.relu(self.bn11(self.conv11(x1))))
    x12_1 = self.do12(F.relu(self.bn12(self.conv12(x11))))
    x1p = F.max_pool2d(x12_1, kernel_size=2, stride=2)


    # Stage 2
    x21 = self.do21(F.relu(self.bn21(self.conv21(x1p))))
    x22_1 = self.do22(F.relu(self.bn22(self.conv22(x21))))
    x2p = F.max_pool2d(x22_1, kernel_size=2, stride=2)

    # Stage 3
    x31 = self.do31(F.relu(self.bn31(self.conv31(x2p))))
    x32 = self.do32(F.relu(self.bn32(self.conv32(x31))))
    x33_1 = self.do33(F.relu(self.bn33(self.conv33(x32))))
    x3p = F.max_pool2d(x33_1, kernel_size=2, stride=2)

    # Stage 4
    x41 = self.do41(F.relu(self.bn41(self.conv41(x3p))))
    x42 = self.do42(F.relu(self.bn42(self.conv42(x41))))
    x43_1 = self.do43(F.relu(self.bn43(self.conv43(x42))))
    x4p = F.max_pool2d(x43_1, kernel_size=2, stride=2)
```

```
class SiamUnet_diff(nn.Module):
    def forward(self, x1, x2):
        x41d = self.do41d(F.relu(self.bn41d(self.conv41d(x42d))))

        # Stage 3d
        x3d = self.upconv3(x41d)
        pad3 = ReplicationPad2d((0, x33_1.size(3) - x3d.size(3), 0, x33_1.size(2) - x3d.size(2)))
        x3d = torch.cat((pad3(x3d), torch.abs(x33_1 - x33_2)), 1)
        x33d = self.do33d(F.relu(self.bn33d(self.conv33d(x3d))))
        x32d = self.do32d(F.relu(self.bn32d(self.conv32d(x33d))))
        x31d = self.do31d(F.relu(self.bn31d(self.conv31d(x32d))))

        # Stage 2d
        x2d = self.upconv2(x31d)
        pad2 = ReplicationPad2d((0, x22_1.size(3) - x2d.size(3), 0, x22_1.size(2) - x2d.size(2)))
        x2d = torch.cat((pad2(x2d), torch.abs(x22_1 - x22_2)), 1)
        x22d = self.do22d(F.relu(self.bn22d(self.conv22d(x2d))))
        x21d = self.do21d(F.relu(self.bn21d(self.conv21d(x22d))))

        # Stage 1d
        x1d = self.upconv1(x21d)
        pad1 = ReplicationPad2d((0, x12_1.size(3) - x1d.size(3), 0, x12_1.size(2) - x1d.size(2)))
        x1d = torch.cat((pad1(x1d), torch.abs(x12_1 - x12_2)), 1)
        x12d = self.do12d(F.relu(self.bn12d(self.conv12d(x1d))))
        x11d = self.conv11d(x12d)

        return self.sm(x11d)
```


Comparison of the models
100
98
96
94
92
90
88
86
84
Accuracy
Precision
Recall
F1
OA
FC-Siam-conc
FC-Siam-diff
SNUNet

GUI

WEBSITE


Disaster Damage Detection
Go to
Flood Damage Detection
Wildfire Detection
Damage Detection on Post-Hurricane Satellite Imagery
Choose an image...
Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG
Browse files
-93.7658_30.110937.jpeg 2.3KB
Damaged

Share

image2.png
0.8MB

Pre-processing has been done.

Prediction has been done.

** CO2 Emission Calculator **

Please select the type of forest:

Tropical Forest

Please enter the image resolution value:

10

The total burnt area is:

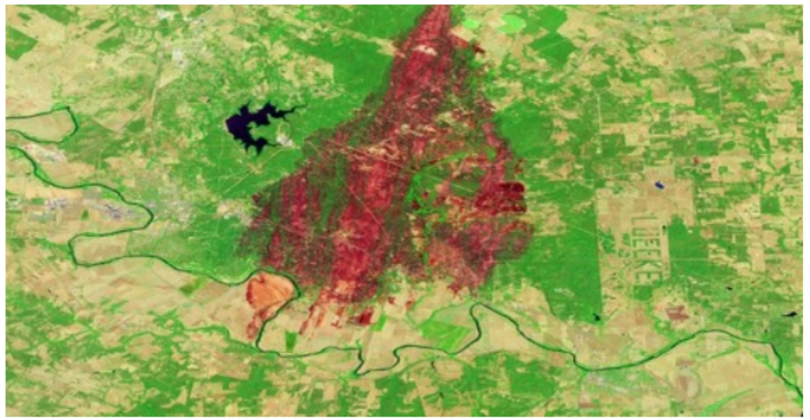
`0.91 km^2`

The total CO2 emitted is:

`41532.87 tons`

This is equivalent to:

`0.04 days of daily electricity power`



** The Predicted Mask is **:

Manage app

Thanks