Task For Position Of Geospatial Machine Learning Engineer

Project Code And Collab Notebook can Be Found Here

Click on the images above or use these links

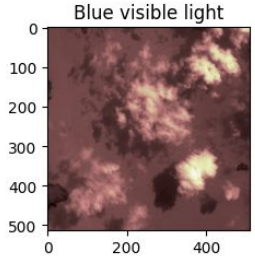
Github - https://github.com/tejas1904/SatelliteCloudCoverDetection

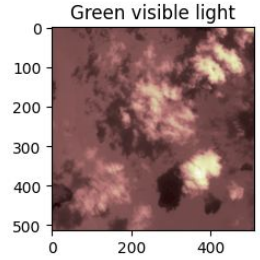
Collab - https://colab.research.google.com/drive/1ox-GmNW0Wv794-_LJLGbUOtwaVrf9kRM?usp=sharing

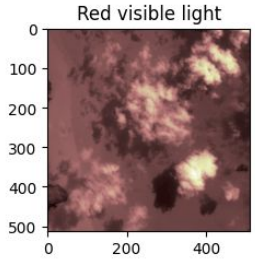
Tejas S
+91 8495998705
tejasboss1904@gmail.com, tejasssravsetty@gmail.com
LinkedIn

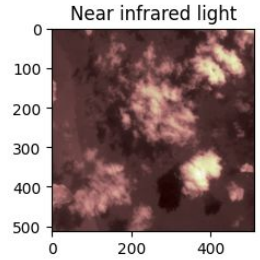
EDA Exploratory data analysis

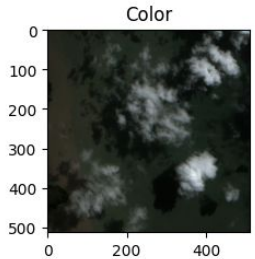
- Train_features:
  - Contains subfolders, each with 4 images.
  - The images in each subfolder are named similarly to the subfolder itself.
  - Images in these subfolders are likely representing different spectral channels.
- Train_labels:
  - Contains images.
  - The names of these images appear to correspond to the subfolders in `Train_features`
  -
- Let's begin by plotting these images along with the corresponding image mask
- The Color image is the red,blue and green channels combined and linear normalized
- True Color is all the channels combined and then sigmoid normalized
- The mask is a binary mask representing the human labelled cloud cover

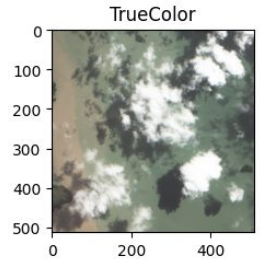


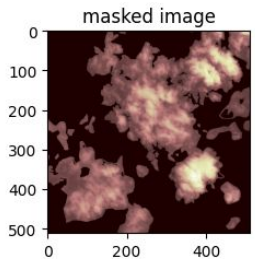


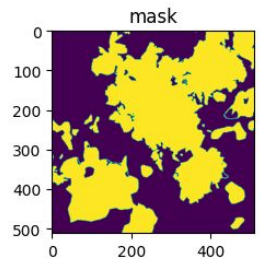












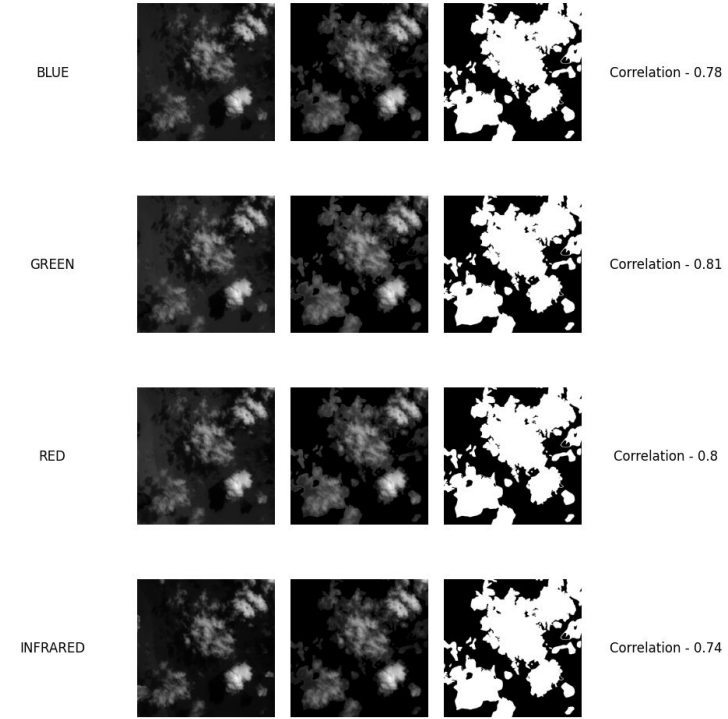
Finding correlations in image data

Since our data is multispectral, let's determine which channels contain the most relevant information. This will help us identify the crucial bands of data. An example correlation between each channel's masked image and the image mask is displayed on the right.

It appears that the highest correlations are found in the green, red, and blue channels, followed by the infrared channel. To validate this pattern across the entire dataset, we can store the correlations for all bands for each target mask in a pandas dataframe. Then, we can identify the top three bands for each case and tally the counts for each of these different top three cases.

On the right, you can see an example for a subset of 10% of the images in our dataset. It shows that band 1, 2, and 3 have the highest correlations, corresponding to the red, blue, and green channels, respectively.

Ideally, we would prefer to use all four image bands, but most large-scale datasets consist of three-channel images. Pretrained models are typically trained on three-channel images, which is why we can consider this constraint.



	Band1	Band2	Band3	Band4	Top3Bands
0	0.58	0.58	0.58	0.67	[Band1, Band2, Band4]
1	0.64	0.62	0.62	0.71	[Band1, Band2, Band4]
2	0.70	0.73	0.73	0.87	[Band2, Band3, Band4]
3	0.56	0.56	0.55	0.64	[Band1, Band2, Band4]

```
band_counts
```

```
[Band1, Band2, Band3]    485
[Band2, Band3, Band4]    439
[Band1, Band2, Band4]    216
[Band1, Band3, Band4]     34
```

Architecture Evaluation and Literature review

Various Methods have been utilized over the years for the process of cloud detection some classical computer vision based and others are machine learning and deep learning based

In classical approaches they use a combined a threshold exponential spectral angle map (TESAM), adaptive Markov random field (aMRF), and dynamic stochastic resonance (DSR). using this revealed that threshold-based algorithms effectively identified cloud presence by determining proper thresholds of apparent reflectance or brightness temperatures through specific channels for various sensors. These thresholds could vary with seasons hence its not that effective

Other research combines multi-scale convolution features, deep learning, decision trees, Bayesian classification, random forest-based methods, and object-based convolutional neural networks. These machine learning methods are adaptable and relatively straightforward because they learn from training data. However, they may not always be consistent because their performance relies on the quality and diversity of the input data used during training.

More latest models use U-net for segmentation mainly includes two parts: the feature extraction and upsampling The scale of feature extraction increases each time in the upsampling image features extracted by U-Net are more advanced than those extracted by FCNs

With the introduction of ViT visual transformers which outperform CNN like resnet on large scale datasets many transformer based segmentation models have been introduced, the types of transformer model and reasoning for selecting a particular architecture is explained later

- Segmenter
  - Uses a transformer-based approach for semantic segmentation which consist of a ViT backbone
  - introduces a mask transformer as the decoder
  - Segmenter especially captures the global context of images, unlike the traditional CNN-based approaches
  - More accurate than a CNN
- SEgmentation TRansformer (SETR)
  - SETR employs a pure ViT Transformer as its encoder, gradually reducing spatial resolution.
  - The SETR encoder transforms an image into a sequence of patches and uses linear projections, patch embedding, and position embedding before passing the information through a series of Transformer layers.
  - The decoder has several variants has a progressive up-sampling design and multi-level feature aggregation
  - As it uses regular ViT in the encoder is it slow and not as accurate as other architectures
- Swin Transformer
  - It uses a shifted window approach rather than a sliding window approach to calculate self attention in the form of hierarchical feature maps
  - The main advantage with this is that has much lower latency than the earlier sliding window based approaches
  - Faster than STER
- SegFormer
  - is a semantic segmentation architecture comprising a hierarchical Transformer encoder
  - Decoder is and a lightweight multilayer perceptron (MLP) decoder
  - It uses a smaller patch size (4x4) compared to ViT's (Vision Transformer) 16x16 for precise segmentation
  - also employs an overlapped patch merging process to preserve local continuity around patches, improving segmentation accuracy
  - Features a positional encoder free design
- For this project we choose **segFormer** as it uses a robust encoder and efficient decoder the implementation is simple on tensorflow and the models come in various sizes B0-B5 as the requirement is to run efficiently on a edge device the smaller B0,B1,B2 can be easily explored to work efficiently on computational low powered devices.

Fine Tune A ViT model

- Initialized a segformer B0-model with input size 224x224 for fine tuning,
- Used a custom implementation of segformer in tensorflow with modified layers so that it supports model pruning and optimization
- Dataset and Input Pipeline Creation
    - Created a custom **data loader** class that maps a data set of type `tf.data.dataset`
    - Data loader necessary as dataset size is 17 GB which will not fit on computer primary memory (RAM)
    - Data loader does the following
        - For each folder in train_features read the 3 images corresponding to red blue and green channel.
        - Combines the 3 images into a single 3 dimensional input feature and normalize the image
        - Read the binary cloud mask , this is the output feature
        - Apply augmentation random horizontal and vertical flips to both image and cloud mask
        - Apply additional random brightness and contrast to the input images
    - Applied `tf.data.AUTOTUNE` to the dataset so that execution is parallelized and interleaved and GPU never remains idle.

- 3 datasets created train test and validation with a 70,20,10 split and mapped to data loader
- Model parameters
  - Model is compiled with
    - Optimizer - adam
    - Learning rate - 0.001
    - Loss - Categorical_Crossentropy
- Training
  - Model is trained on the train-dataset with shuffling on each batch
  - Batch-size-32
  - epoch -15
- After 15 epochs model achieves 90%accuracy on train set and 89% on val set

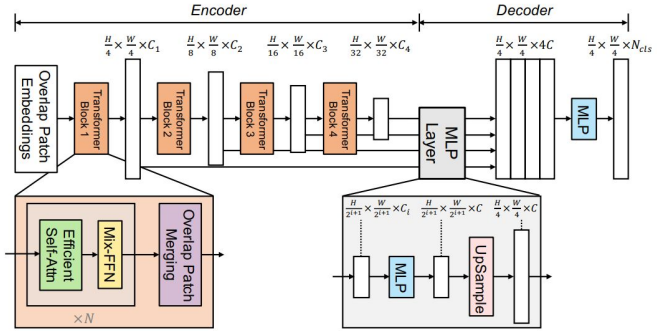
Model Optimization

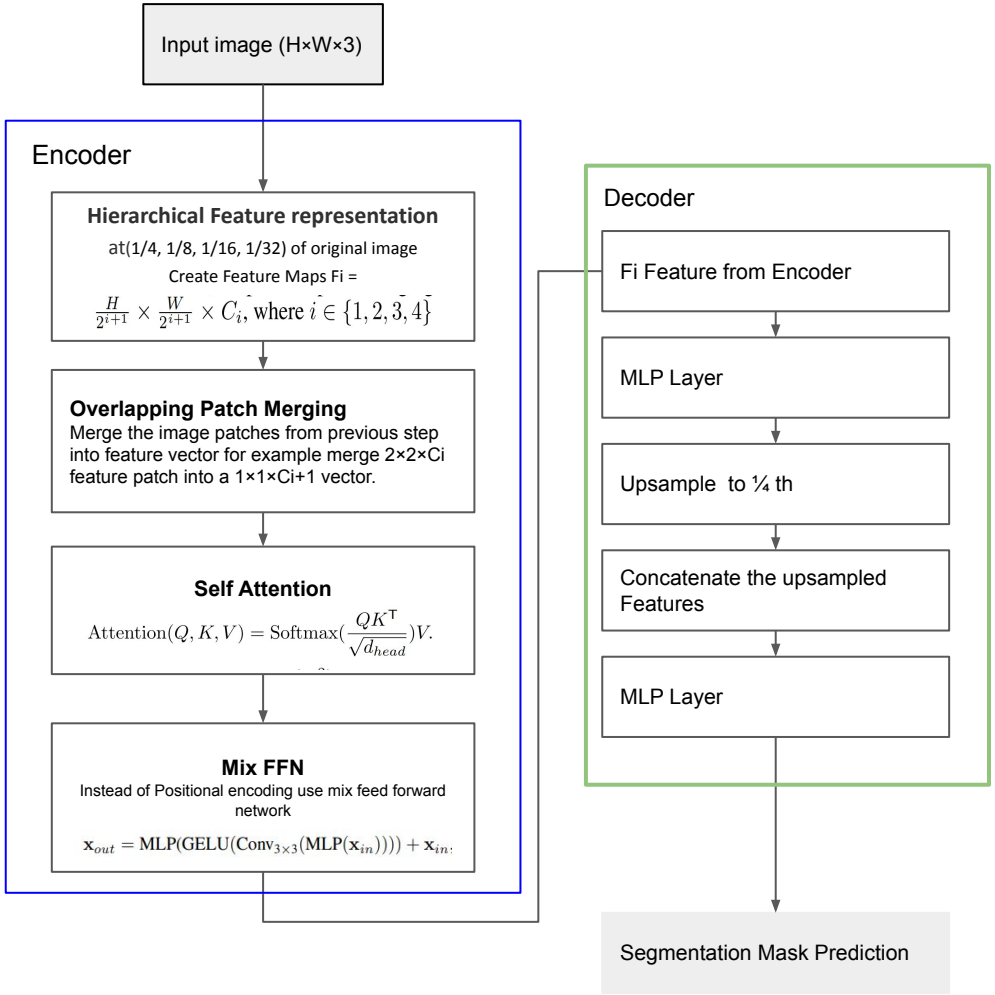
- Explored both **model pruning** and **quantization**
- **Pruning**
  - Model pruning removes less important or redundant weights to create a more optimized network to run efficiently on computationally limited devices
  - For this project lets explore sparsity based pruning which sets n% of weights to zero
  - In our case we perform by using tensorflow inbuilt Tf optimization
  - To perform pruning our custom layers in the architecture must must import `tfmot.sparsity.keras.PrunableLayer` along with this it must return the weights to be considered for pruning
  - In our case we optimize the weights of the conv2d and Dense layers in the Mlp blocks.
  - Pruning Process
    - Being by creating a copy of the original trained non-pruned model
    - Use `tfmot.prune_low_magnitude` and apply `tfmot.sparsity.keras.ConstantSparsity` with 80% sparsity
    - Copy the weights of the trained model over to the new model
    - Finetune this new pruned model on the train dataset for a few epochs
  - This makes the model with 80% sparse weighs

- Once the fine-tuning is complete apply to tfmot.sparsity.keras.strip_pruning this remove the variables reducing the model size
- Accuracy change
  - Before pruning `37/37 [==============] - loss: 0.2885 - accuracy: 0.8932`
  - After Pruning `74/74 [==============] - loss: 0.3571 - accuracy: 0.8696`
  - No significant loss in pruned model
- Model Size change after zipping
  - Original model - `Size of gzipped original model without stripping: 13.42 mb`
  - Pruned Model - `Size of gzipped pruned model with stripping: 8.35 mb`
  - Reduction of `5.2Mb!`
- **Quantization**
  - Quantization is process of converting the weights of a model to a lower precision
  - 16 bit floats or 8 bit integers. 16-bit floats for GPU acceleration and 8-bit integer for CPU execution.Dynamic quantization also reduces to 8-bit integer
  - For this project we can apply Dynamic quantization
  - Model Size change after zipping
    - Original pruned model - `Size of gzipped original model without stripping: 13.42 mb`
    - Quantized Model - `size of the tf lite quantized model in Mb 4.02`
    - Reduction of `9.3Mb!`

Architecture Explanation and Flowchart

- The segformer model consists of an Transformer based encoder and and an fully mlp decoder
  - The encoder generates both high level coarse and low level dense features
  - The decoder fuses these multi level features to predict the segmentation mask
- Encoder process
  - Input Image of size H × W × 3, is divided into patches of size 4 × 4. Contrary to ViT that uses patches of size 16 × 16, using smaller patches favors the dense prediction task.
  - The goal of the hierarchical Transformer encoder is to generate multi-level features at different resolutions (1/4, 1/8, 1/16, 1/32 of the original image resolution). These features are obtained through patch merging, resulting in hierarchical feature maps.
  - A salient Feature of Segformer is Unlike ViT, which uses positional encoding (PE) to introduce location information, SegFormer introduces Mix-FFN directly uses a 3 × 3 convolution layer in the feed-forward network, which considers the effect of zero padding to encode location information. This eliminates the need for explicit positional encoding. Depth-wise convolutions are used for efficiency.
- Decoder Process
  - The decoder is a fully mlp based
  - Unifying Layer unify the feature maps
  - A upsampling layer is followed by concentration of maps
  - The final layer predicts the segmentation map





Modification in In Attention Mechanism

- In original Vit the attention mechanism used is

$$\text{Attention}(Q, K, V) = \text{Softmax}(\frac{QK^T}{\sqrt{d_{head}}})V.$$

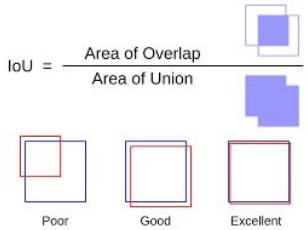
- Where three matrices involved: Q (Query), K (Key), and V (Value),
- Q, K, V have the same dimensions N × C
- where N = H × W is the length of the sequence
- As this has computational complexity of $O(N^2)$
- The authors describe a new reduction of the sequence length of n to k given by

$$\hat{K} = \text{Reshape}(\frac{N}{R}, C \cdot R)(K)$$
$$K = \text{Linear}(C \cdot R, C)(\hat{K}),$$

- R is the reduction ratio
- Therefore, the new K has dimensions N/R × C
- Reducing the complexity to O(N2/R)

Results and Discussion

- There are various metrics for segmentation task
  - **Accuracy** measures the ratio of correctly predicted pixels to the total number of pixels in the image.
    Formula: ACC = (TP + TN) / (TP + TN + FP + FN)
  - Precision (P):
    Precision measures the ratio of correctly predicted positive pixels to the total number of pixels predicted as positive.
    Formula: P = TP / (TP + FP)
  - **Recall** measures the ratio of correctly predicted positive pixels to the total number of actual positive pixels.
    Formula: R = TP / (TP + FN)
  - **F1-Score** is the harmonic mean of precision and recall. It provides a balance between precision and recall.
    Formula: F1 = 2 * (P * R) / (P + R)
  - **Intersection over Union (IoU)**also known as the Jaccard Index, measures the ratio of the intersection of the predicted and true positive pixels to their union. It's often used to evaluate the overlap between the predicted and ground truth masks.
- Accuracy is not a very good measure for Segmentation purposes as if the final segmentation mask contains many negative samples compared to positive samples a simple model that always predicts negative will achieve high accuracy
- Iou is a much more apt measure as is considers the overlap of prediction and ground truth

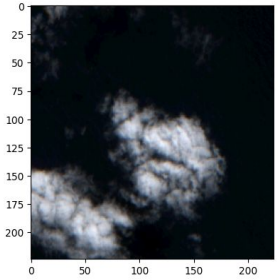


$$J(A,B) := \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

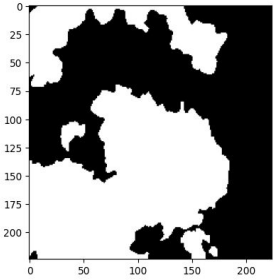
Comparison of results of Segformer To Unet

Metric	Segformer	Unet
Accuray	90.18	88.60
Precision	88.37	86.67
Recall	88.30	86.68
F1-Score	83.98	82.30
IOU	85.30	82.27

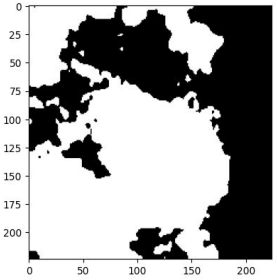
Segformer



Image

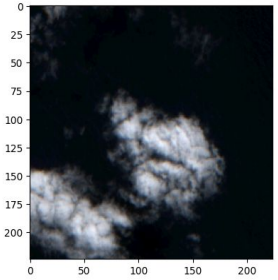


Mask

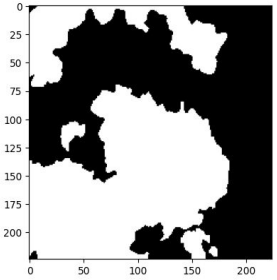


prediction

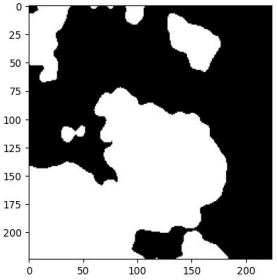
Unet



Image



Mask



prediction

Discussion