

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**

**NATIONAL INSTITUTE OF TECHNOLOGY TIRUCHIRAPPALLI, TAMILNADU - 620015**

**INTERNSHIP REPORT ON**

## Developing Source-Independent Deep Hashing CNNs for Multi-Source Remote Sensing Image Retrieval

## Image – text description model for EUROSAT dataset using CLIP module

Submitted by**: YUVARAJ KRISHNAN B (EC21B1059)**

In partial fulfillment for the award of the

Bachelor of Technology in

## ELECTRONICS AND COMMUNICATION ENGINEERING

**NATIONAL INSTITUTE OF TECHNOLOGY PUDUCHERRY KARAIKAL, PUDUCHERRY – 609609**

UNDER THE GUIDANCE OF

**DR. AVIK HATI**

**Professor**

## DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

## NATIONAL INSTITUTE OF TECHNOLOGY

## TIRUCHIRAPPALLI, TAMILNADU – 620015

**TABLE OF CONTENTS:**

* **Abstract**
* **Overview of the study**
* **Developing Source-Independent Deep Hashing CNNs for Multi-Source Remote Sensing Image Retrieval**

1. **Introduction**
2. **Proposed Method**
3. **Contributions**
4. **Methodology**

* DUAL-SOURCE REMOTE SENSING IMAGE DATA SET
* LARGE-SCALE REMOTE SENSING IMAGE RETRIEVAL
* Architecture of SIDHCNNs:
* Objective Function for Optimizing SIDHCNNs:
* LOSS FUNCTIONS

1. Code overview

* Main Key components and models
* Data loader
* Architecture of Network 1 and 2
* Defining Loss Function
* Objective function:
* Testing and Validation
* Test and Validation output

1. Conclusion
2. Summary

* **Image – text description model for EUROSAT dataset using CLIP module**

1. Introduction
2. Proposed method
3. Methodology
4. Code Overview

* Main Key components and Modules
* Data Loader
* Loss Function
* Training the Model
* Testing the model

1. Outputs of Training and Testing model
2. Conclusion
3. Summary

* **Reference**

**ABSTRACT:**

The paper titled **"Learning Source-Invariant Deep Hashing Convolutional Neural Networks for Cross-Source Remote Sensing Image Retrieval"** by Yansheng Li, Yongjun Zhang, Xin Huang, and Jiayi Ma addresses the challenge of remote sensing image retrieval across different data sources. The authors propose a novel deep hashing framework to learn source-invariant feature representations that can effectively bridge the gap between heterogeneous remote sensing data. This method leverages Convolutional Neural Networks (CNNs) to generate binary hash codes, which are invariant to the source of the remote sensing images. The resulting framework significantly improves retrieval performance and ensures efficient cross-source remote sensing image retrieval.

The paper " introduces CLIP (Contrastive Language–Image Pre-training), a method to leverage natural language descriptions to train visual models. The authors propose a model that learns to connect images and their corresponding textual descriptions, resulting in a system that can understand visual concepts through natural language supervision. CLIP achieves competitive performance on various benchmarks without requiring task-specific fine-tuning, demonstrating the utility of learning from natural language.

Overview of the study:

This report will be detailing a specific area of study undertaken during the internship . We will be recreating this particular paper with our “EuroSAT” dataset .This section involves implementing and testing the **Learning Source-Invariant Deep Hashing Convolutional Neural Networks using the network form the paper which involves images from two different sensors (i.e. Panchromatic sensor and Multi-Spectral sensor) . We build a model with the assigned network and loss function to predict the accuracy of the model .**

In **Learning Transferable Visual Models From Natural Language Supervision .** While standard image models jointly train an image feature extractor and a linear classifier to predict some label, CLIP jointly trains an image encoder and a text encoder to predict the correct pairings of a batch of (image, text) training examples. At test time the learned text encoder synthesizes a zero-shot linear classifier by embedding the names or descriptions of the target dataset’s classes . We will convert image and text to tensors . Feed them into the CLIP model to calculate the accuracy of the model .

**Developing Source-Independent Deep Hashing CNNs for Multi-Source Remote Sensing Image Retrieval**

**Introduction :**

The paper begins by highlighting the importance of remote sensing image retrieval in various applications such as environmental monitoring, disaster management, and urban planning. It points out the challenges posed by the heterogeneity of remote sensing data, which includes variations in spatial resolution, spectral characteristics, and acquisition conditions. Traditional retrieval methods often fail to perform well across different sources due to these inconsistencies.

PROPOSED METHOD:

To address these challenges, the authors propose a deep hashing framework that learns source-invariant features. The framework consists of several key components:

1. **Convolutional Neural Networks (CNNs):** Used to extract high-level feature representations from the input images.
2. **Deep Hashing Layers:** Transform the high-level features into compact binary hash codes.
3. **Source-Invariant Learning:** Ensures that the learned hash codes are invariant to the source of the remote sensing images.

The CNNs are trained with a combination of supervised and unsupervised learning techniques to enhance the robustness and generalizability of the feature representations.

CONTRIBUTIONS:

The main contributions of this paper can be summarized as follows.

1) To the best of our knowledge, this paper, for the first time, reveals the possibility of conducting CS-LSRSIR and shows the potential applications of CS-LSRSIR.

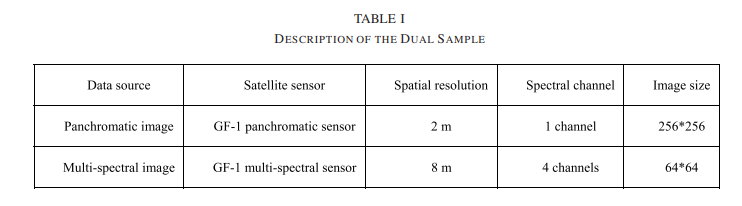
2) This paper proposes SIDHCNNs to cope with CS-LSRSIR where SIDHCNNs can be optimized from scratch in an end-to-end manner. In addition, a series of optimization constraints are advocated to pursue a stable optimization of SIDHCNNs.

3) This paper collects and releases a new DSRSID which is used to evaluate CS-LSRSIR in this paper and benefits promoting the multisource remote sensing image processing technology.

METHODOLOGY:

DUAL-SOURCE REMOTE SENSING IMAGE DATA SET

1. These existing data sets were constructed by only one kind of remote sensing data source and are called uni-source data sets in the following. Intuitively, these uni-source data sets would not be competent for evaluating CS-LSRSIR. To promote the multisource remote sensing image analysis techniques, including the discussed CS-LSRSIR in this paper, it is very urgent to construct a remote sensing image data set containing at least two kinds of remote sensing data sources. To this end, this paper collects a new DSRSID.
2. More specifically, the DSRSID is tiled from two kinds of remote sensing data sources (i.e., panchromatic images and multispectral images) and is manually annotated. The DSRSID is composed of large numbers of dual samples where each dual sample is a combination of one panchromatic image and one multispectral image covering the same ground region. It is noted that the panchromatic image and multispectral image in one dual sample belong to the same land-cover type, but reflect different aspects of the captured ground region because of the spatial and spectral variations.



The DSRSID is formulated as **D = {(Pi, Mi, Li)|i = 1, 2,..., N},**

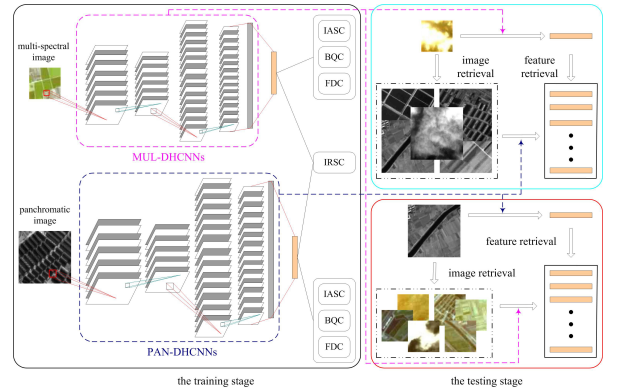
* **D** denotes the set of dual samples,
* **i** denotes the index of the dual sample,
* **N** stands for the volume of the DSRSID (i.e., the number of dual samples),
* **Pi** ∈ R256×256 is the panchromatic image,
* **Mi** ∈ R64×64×4 denotes the multispectral image,
* **Li** denotes the land-cover type.

In this paper, **D = {(Pi, Mi, Li)|i = 1, 2,..., N}** is randomly split into two nonoverlapped parts:

* a training data set **DU Tr = {(Pi, Mi, Li)|i = 1, 2,..., V } ,**
* a testing data set DU Te = {(Pi, Mi, Li)|i = 1, 2,..., Q}, where N = V + Q, V is the volume of the training data set, and Q is the volume of the testing data set.

CROSS-SOURCE LARGE-SCALE REMOTE SENSING IMAGE RETRIEVAL

* CS-LSRSIR approach is composed of two stages: the training stage and the testing stage. More specifically, the training stage is responsible for training SIDHCNNs and the testing stage presents the cross-source remote sensing image retrieval process based on SIDHCNNs. In the following, Section III-A introduces the training stage and Section III-B presents the testing stage.

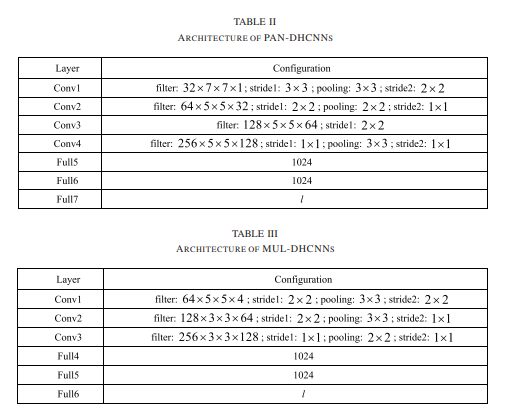


* Workflow of the proposed CS-LSRSIR approach via SIDHCNNs. The proposed CS-LSRSIR method includes two stages: the training stage works on training SIDHCNNs and the testing stage carries out the cross-source remote sensing image retrieval tasks based on the learned SIDHCNNs

Architecture of SIDHCNNs:

Instead of using one unified DHCNNs architecture in US-LSRSIR , we design two different DHCNNs architectures for different remote sensing image sources to fully mine the visual cues in images from different sources. Based on the specific remote sensing image types in the DSRSID, we craft two different DHCNNs architectures for the panchromatic and multispectral images.

* More specifically, the DHCNNs for panchromatic images are called PAN-DHCNNs
* DHCNNs for multispectral images are called MUL-DHCNNs.



Objective Function for Optimizing SIDHCNNs:

This section uses the training data set DU Tr = {(Pi, Mi, Li)|i = 1, 2,..., V } which is introduced in Section II to formulate the objective function to learn SIDHCNNs.

* Let SU ∈ RV×V×2 denotes the intersource pairwise similarity matrix, where SU i,j,1 = 1, SU i,j,2 = 0,i = 1, 2,..., V ; j = 1, 2,..., V if Pi and Mj belong to the same land-cover type and SU i,j,1 = 0, SU i,j,2 = 1 if Pi and Mj come from different land-cover types.
* SP ∈ RV×V ×2 stands for the intrasource pairwise similarity matrix on the panchromatic image data set, where SP i,j,1 = 1, SP i,j,2 = 0 if Pi and Pj belong to the same land-cover category and SP i,j,1 = 0, SP i,j,2 = 1 if Pi and Pj do not belong to the same category.
* Furthermore, SM ∈ RV×V ×2 denotes the intrasource pairwise similarity matrix on the multispectral image data set, where SM can be generated by a similar generation process to that of SP

Let λP and λM stand for the network hyper parameters of PAN-DHCNNs and MUL-DHCNNs, respectively. λ (Pi, λP) ∈ Rl denotes the feature representation of the panchromatic image Pi by PAN-DHCNNs, and ϒ(Mi, λM ) ∈ Rl stands for the feature output of the multi-spectral image Mi by MUL-DHCNNs.

LOSS FUNCTIONS :

**IRSC (Intra-class Reconstruction Similarity Criterion)**:

* **Purpose**: Ensures that the reconstructed images from the same class are similar.
* **Mechanism**: This loss function minimizes the distance between the reconstructed features of images within the same class, promoting intra-class compactness.



**IASC (Inter-Class Similarity and Consistency) Loss**:

* **Purpose**: Ensures that features of images from different classes are distinct in the feature space.
* **Implementation**: Encourages inter-class dissimilarity by maximizing the distance between feature vectors of images from different classes.



**BQC (Balanced Quantization Constraint) Loss**:

* **Purpose**: Ensures that the binary codes generated are balanced, meaning that each bit has an equal probability of being 0 or 1 across the dataset.
* **Implementation**: Enforces a balance in the distribution of binary codes to avoid a biased representation in the hashing process.

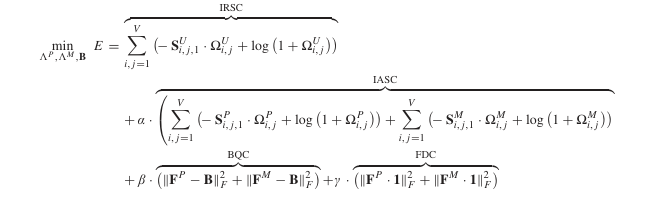


**FDC (Feature Discrepancy Constraint) Loss**:

* **Purpose**: Minimizes the discrepancy between the features learned from different sources (e.g., different satellite sensors) to ensure source invariance.
* **Implementation**: Reduces the feature discrepancy between cross-source images, ensuring that the learned features are robust to variations from different sources.



The Final equation of the Loss function can be expressed as :

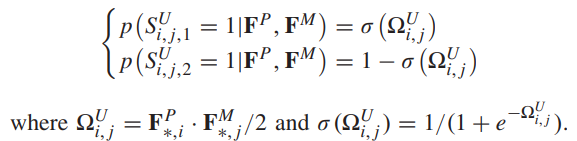


Where the alpha, beta, gamma values can be gives in order that improves the accuracy of the given model .

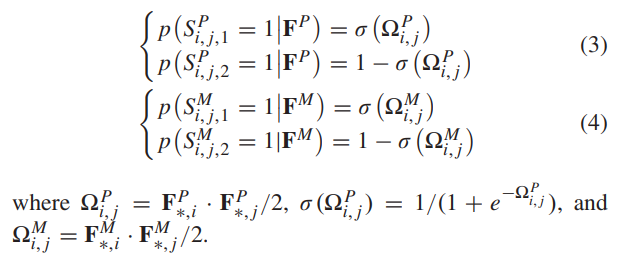
Defining the functions:

Moreover, the objective function for learning SIDHCNNs (i.e., λP and λM ) can be formulated as (1), as FP ∈ Rl×V with FP ∗,i =  ψ  (Pi, λP), FM ∈ Rl×V with FM ∗,i = ϒ(Mi, λM ) and B ∈ {−1, +1} l×V , p(·|FP, FM ) is the intersource-likelihood function, and p(·|FP) and p(·|FM) denote the intrasource-likelihood functions. In addition, α, β, and γ stand for the penalty weights of the constraints.

* More specifically, the intersource-likelihood function is defined by the sigmoid function

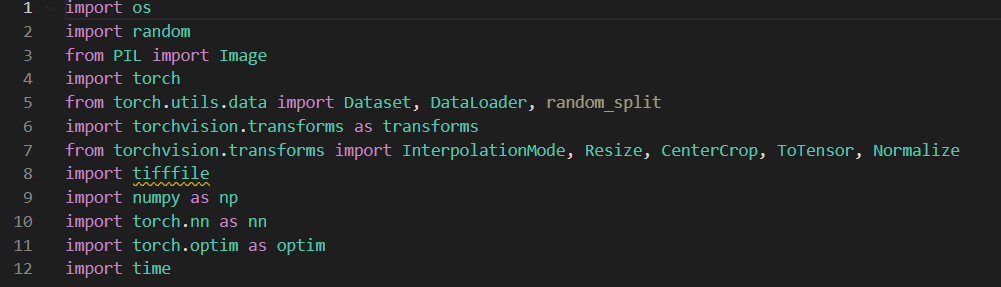


* For the panchromatic image source, the intrasourcelikelihood function can be expressed as. In addition, the intrasource-likelihood function for the multispectral image source can be expressed as



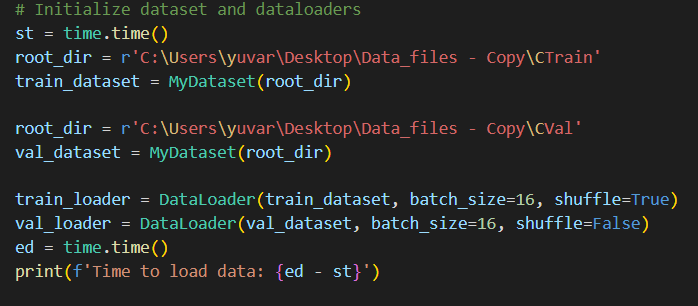
CODE OVERVIEW:

Main Key components and models:



DataLoader:

Construct the Dataloader part such that it fetches the EUROSat dataset from the available folder . We have to load images in pairs (i.e. MS IMAGE AND RGB IMAGE) Then we normalize and resize it , so that they are fed into the network. In this part we use import os.



Architecture of Network 1 and 2:

#### Network1

**Architecture:**

* **Input:** The network takes an input tensor with in\_channels (e.g., 1 for grayscale images) and size 256x256.
* **Layers:**
  + **Conv1:** Convolutional layer with 32 filters, a kernel size of 7x7, stride 3, and no padding.
  + **Pool1:** Average pooling layer with kernel size 3x3 and stride 2.
  + **Conv2:** Convolutional layer with 64 filters, a kernel size of 5x5, stride 2, and no padding.
  + **Pool2:** Average pooling layer with kernel size 2x2 and stride 1.
  + **Conv3:** Convolutional layer with 128 filters, a kernel size of 5x5, stride 2, and no padding.
  + **Conv4:** Convolutional layer with 256 filters, a kernel size of 5x5, stride 1, and no padding.
  + **Pool4:** Average pooling layer with kernel size 3x3 and stride 1.
  + **Linear Layers:** Three fully connected layers with sizes 1024, 1024, and 16 respectively.

**Forward Pass:**

1. Input tensor x passes through Conv1 followed by ReLU activation and Pool1.
2. The resulting tensor passes through Conv2 followed by ReLU activation and Pool2.
3. The tensor then goes through Conv3 and Conv4, each followed by ReLU activation and Pool4.
4. The tensor is reshaped into a 2D tensor and passed through three linear layers with ReLU activation between the first two linear layers.

**Output:**

* The final output is a tensor of shape (batch\_size, 16)

#### Network2

**Architecture:**

* **Input:** The network takes an input tensor with in\_channels (e.g., 1 for grayscale images) and size 64x64.
* **Layers:**
  + **Conv1:** Convolutional layer with 64 filters, a kernel size of 5x5, stride 2, and no padding.
  + **Pool1:** Average pooling layer with kernel size 3x3 and stride 2.
  + **Conv2:** Convolutional layer with 128 filters, a kernel size of 3x3, stride 2, and no padding.
  + **Pool2:** Average pooling layer with kernel size 3x3 and stride 1.
  + **Conv3:** Convolutional layer with 256 filters, a kernel size of 3x3, stride 1, and no padding.
  + **Pool3:** Average pooling layer with kernel size 2x2 and stride 1.
  + **Linear Layers:** Three fully connected layers with sizes 1024, 1024, and 16 respectively.

**Forward Pass:**

1. Input tensor y passes through Conv1 followed by ReLU activation and Pool1.
2. The resulting tensor passes through Conv2 followed by ReLU activation and Pool2.
3. The tensor then goes through Conv3, followed by ReLU activation and Pool3.
4. The tensor is reshaped into a 2D tensor and passed through three linear layers with ReLU activation between the first two linear layers.

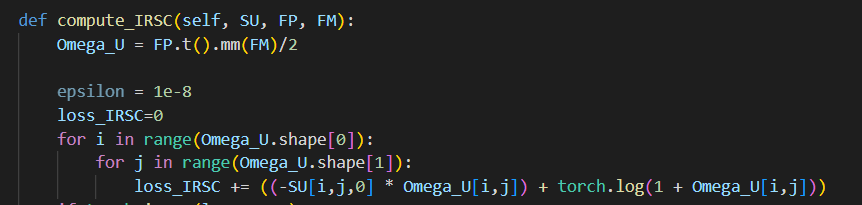
**Output:**

* The final output is a tensor of shape (batch\_size, 16).

We define the combination of Network 1 and Network 2 as a class called CombinedNetwork with constructor .

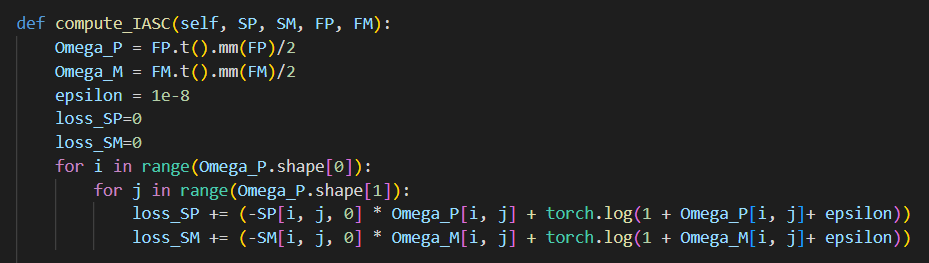
Defining Loss Function:

We define all the loss functions with alpha , beta and gamma as constants



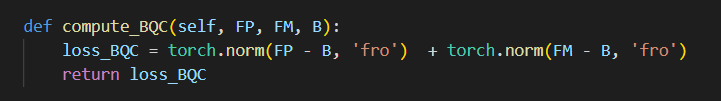
IRSC is being computed here using for loop , it goes to every iteration to compute the loss.

0,1 defines the dimentions of the matrix (rows and columns)

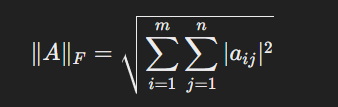


IASC is being computed here using for loop , it goes to every iteration to compute the loss.

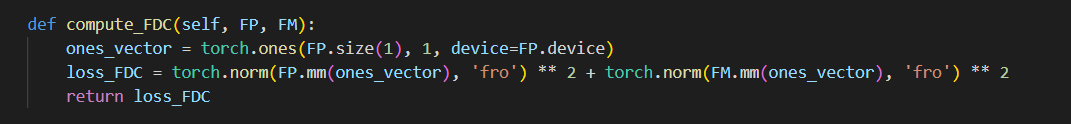
0,1 defines the dimentions of the matrix (rows and columns)



The Frobenius norm is a matrix norm of an m×nm \times nm×n matrix AAA defined as the square root of the sum of the absolute squares of its elements.

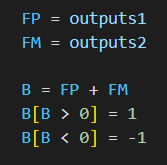


We compute FDC with the following code:

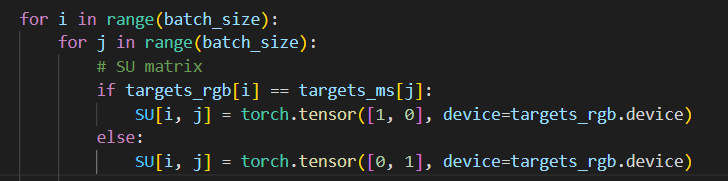


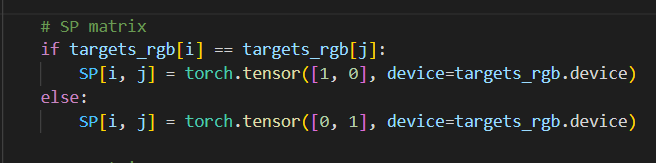
Objective function:

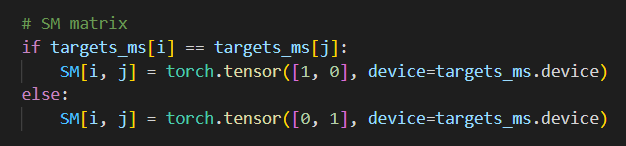
We initialize the objective function SU, SP, SM, FP, FM and B



FP and FM are the outputs of Network 1 and Network 2. SU, SP and SM are the intersource pairwise similarity matrix and intrasource pairwise similarity matrix







TESTING AND VALIDATION:

**Batch Processing:**

* Iterates over batches of data from train\_loader.
* inputs\_rgb and inputs\_ms are moved to the specified device (cuda or cpu).
* optimizer.zero\_grad(): Resets the gradients to zero before the backward pass.
* outputs1, outputs2: Forward pass through the combined model with the inputs.

**Similarity Matrices:**

* compute\_similarity\_matrices(targets, targets): Computes the similarity matrices SU, SP, SM using target labels. This needs to be replaced with the actual computation logic.
* FP and FM are the outputs from the model.
* B is a binary matrix based on the sum of FP and FM.

**Loss Calculation:**

* loss = loss\_fn(SU, SP, SM, FP.t(), FM.t(), B.t()): Calculates the loss using the specified loss function.
* loss.backward(): Performs backpropagation to compute the gradients.

**Gradient Clipping:**

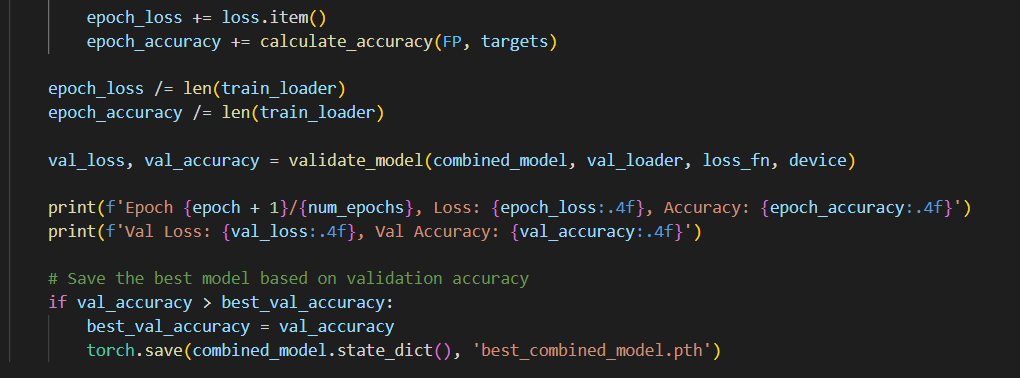
* torch.nn.utils.clip\_grad\_norm\_(combined\_model.parameters(), max\_norm=1.0): Clips gradients to avoid exploding gradients.

**Optimizer Step:**

* optimizer.step(): Updates the model parameters based on the computed gradients.

**Metrics Calculation:**

* epoch\_loss += loss.item(): Accumulates the loss for the current epoch.
* epoch\_accuracy += calculate\_accuracy(FP, targets): Accumulates the accuracy for the current epoch.



**Epoch Metrics:**

* epoch\_loss /= len(train\_loader): Averages the loss over all batches.
* epoch\_accuracy /= len(train\_loader): Averages the accuracy over all batches.

**Validation:**

* val\_loss, val\_accuracy = validate\_model(combined\_model, val\_loader, loss\_fn, device): Validates the model on the validation set.

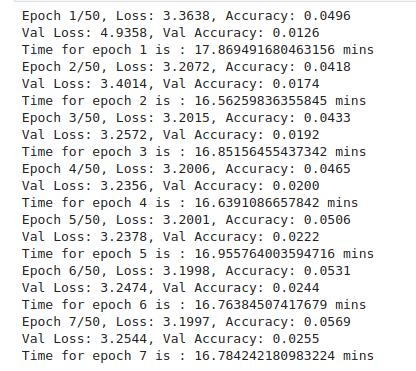
**Logging:**

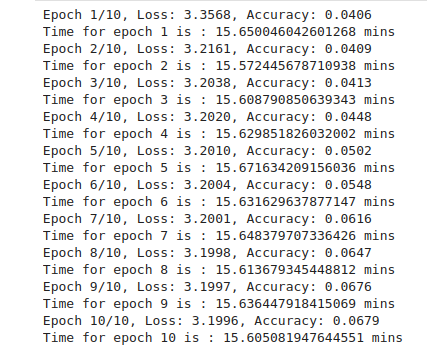
* Prints the training and validation loss and accuracy for each epoch.

**Model Saving:**

* If the current validation accuracy is the best so far, saves the model's state.

Training and Validation output with loss:





OUTPUT OBSERVATION:

We have made the code to run till 100 epochs and the loss has been decreasing significantly by using SGD optimizer and the accuracy increases gradually, But the accuracy is not so close as given in the research paper. This is due to the dataset which we are using.

We use EUROSAT dataset that has RGB and MS images with a size of 60,000 images.

The dataset used in the paper "Learning Source-Invariant Deep Hashing Convolutional Neural Networks for Cross-Source Remote Sensing Image Retrieval" is called the Dual-Source Remote Sensing Image Data Set (DSRSID). This dataset includes:

* Eight typical land-cover categories
* 10,000 dual samples in each category
* Each dual sample consists of one panchromatic image and one multispectral image covering the same ground region, but reflecting different aspects due to spatial and spectral variations​

CONCLUSION

The paper presented a novel deep hashing framework designed to address the challenges of cross-source remote sensing image retrieval. The proposed approach incorporates several key innovations:

1. **Source-Invariant Representation Learning**: The method focuses on learning representations that are invariant to the source domain, effectively handling variations between different imaging sensors and conditions.
2. **Deep Hashing with Convolutional Neural Networks**: By leveraging CNNs, the framework learns compact binary hash codes that are efficient for large-scale image retrieval tasks.
3. **Comprehensive Experimental Evaluation**: Extensive experiments demonstrated the effectiveness of the proposed method, showcasing significant improvements in retrieval performance compared to state-of-the-art techniques.
4. **Potential Applications**: The approach has broad applicability in various remote sensing scenarios, particularly in tasks requiring efficient and accurate image retrieval across different sensor types and imaging conditions.

SUMMARY

The proposed deep hashing framework successfully addresses the challenges of cross-source image retrieval, providing a robust solution with promising performance improvements. The research opens up new possibilities for enhancing remote sensing applications through advanced deep learning techniques.

## Image – text description model for EUROSAT dataset using CLIP module

INTRODUCTION:

The introduction of the paper "Learning Transferable Visual Models From Natural Language Supervision" by Alec Radford et al. outlines the revolutionary impact of pre-training methods that learn directly from raw text in natural language processing (NLP) over recent years. This success in NLP, attributed to task-agnostic objectives like autoregressive and masked language modeling, inspires the exploration of similar methodologies in computer vision. The paper investigates whether scalable pre-training methods that leverage web-scale text collections can bring about a comparable breakthrough in the field of computer vision .

### PROPOSED METHOD:

The proposed method centres on using natural language supervision to learn visual models, a concept that is not entirely new but has gained fresh relevance with recent advancements in representation learning. The approach involves creating a large dataset of 400 million (image, text) pairs sourced from the internet. The core of the method is a contrastive learning framework where the model is trained to predict which text is paired with which image in a batch of data. This is done by maximizing the cosine similarity of the correct (image, text) pairs and minimizing it for incorrect pairings.

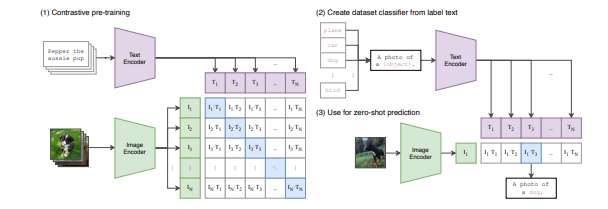
The method uses a contrastive objective, inspired by findings that such objectives can yield better representations than predictive ones. Specifically, the model employs a symmetric cross-entropy loss over cosine similarity scores, optimizing a multi-modal embedding space shared between an image encoder and a text encoder .

### METHODOLOGY

The methodology of the paper "Learning Transferable Visual Models From Natural Language Supervision" by Alec Radford et al. revolves around training a model called CLIP (Contrastive Language-Image Pre-Training) using a large-scale dataset of image and text pairs. The methodology can be broken down into several key components:

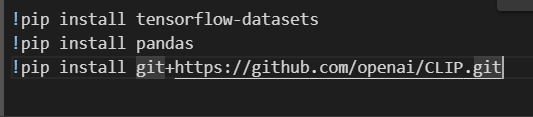
1. **Dataset Construction**:
   * The dataset consists of 400 million (image, text) pairs collected from various sources on the internet. The images are paired with their corresponding textual descriptions.
2. **Model Architecture**:
   * **Image Encoder**: Two architectures are considered for the image encoder: a modified ResNet-50 and the Vision Transformer (ViT). The ResNet-50 is chosen for its widespread use and proven performance, with several modifications to improve performance. The Vision Transformer uses a different approach by treating images as sequences of patches, which are processed similarly to how words are processed in NLP transformers.
   * **Text Encoder**: The text encoder is a Transformer model that processes the text descriptions. It uses a byte pair encoding (BPE) with a vocabulary size of 49,152. The text is processed into a sequence of tokens, and the activations at the end of the sequence are used as the feature representation of the text.
3. **Contrastive Learning Objective**:
   * The core of the training methodology is a contrastive learning objective. The model is trained to predict which text corresponds to which image in a batch of data. This is achieved by maximizing the cosine similarity between the correct (image, text) pairs and minimizing it for incorrect pairs.
   * A symmetric cross-entropy loss is used over cosine similarity scores. The image and text representations are projected into a shared embedding space where the similarities are computed.
4. **Training Details**:
   * A series of five ResNets and three Vision Transformers were trained. ResNets include ResNet-50, ResNet-101, and scaled versions (RN50x4, RN50x16, RN50x64) with increasing compute. Vision Transformers include ViT-B/32, ViT-B/16, and ViT-L/14.
   * The models are trained for 32 epochs using the Adam optimizer with decoupled weight decay. The learning rate is decayed using a cosine schedule.
   * Data augmentation includes random square cropping from resized images. The temperature parameter controlling the range of logits in the softmax is optimized as a log-parameterized multiplicative scalar.

This comprehensive approach allows the CLIP model to learn a shared multi-modal embedding space that generalizes well across various tasks and datasets without requiring task-specific training. The large-scale dataset and contrastive learning objective are crucial in enabling zero-shot transfer capabilities

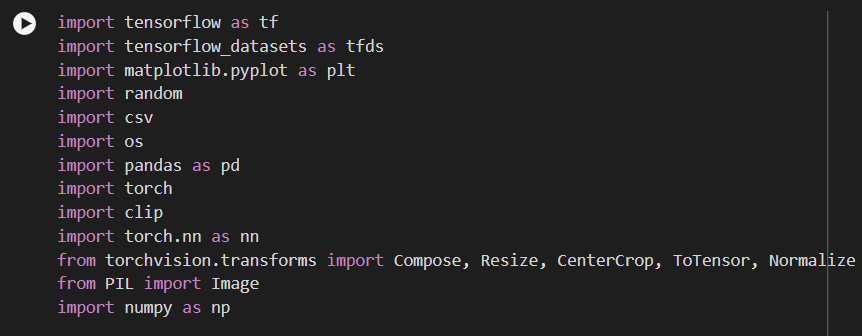


CODE OVERVIEW:

Main Key components and Modules



* Set up an environment for using CLIP, a model from OpenAI.

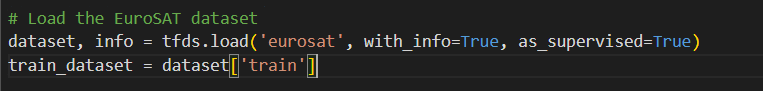


Import all the libraries in order to run the code

Data Loader:

 Loads the EuroSAT dataset using tfds.load(), which returns the dataset and some additional information about it.

 Assigns the training split of the dataset to train\_dataset.



class\_names = ["AnnualCrop", "Forest", "HerbaceousVegetation", "Highway", "Industrial", "Pasture", "PermanentCrop", "Residential", "River", "SeaLake"]

* We define the class names and each class name have a set of class description . And out of the description , we get a selected description for the given class name.
* Then we assign class name to the given label.
* Instead of this dataset, we use a dataset that has 100 images per class. In order to reduce the time taken to obtain the output

i.e.  class\_names = {

        0: "Annual Crop",

        1: "Forest",

        2: "Herbaceous Vegetation",

        3: "Highway",

        4: "Industrial",

        5: "Pasture",

        6: "Permanent Crop",

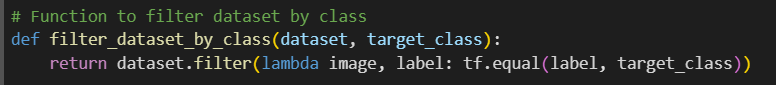
        7: "Residential",

        8: "River",

        9: "Sea/Lake"

    }

* Then with the selected description , we save all the images of the particular model in a csv file . With image name , class name , description as columns.



* Write a function to filter the dataset by classes.
* Later save the output in the csv file

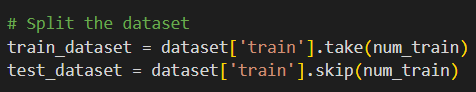
# Specify the class label and number of images to describe

class\_label\_to\_describe =3   # For example, 1 for Forest

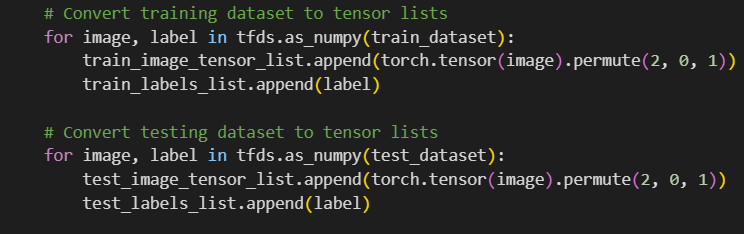
num\_images\_to\_describe = 3000  # Specify the number of images you want to describe

output\_csv\_file =  f"image\_descriptions\_{class\_names[class\_label\_to\_describe].replace(' ', '\_')}.csv"

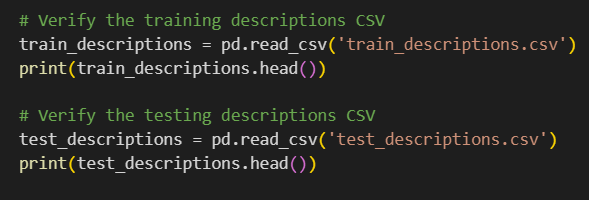
* Split the dataset into training and testing set . Later convert the images into Tensor , so that they can be fed into the clip model



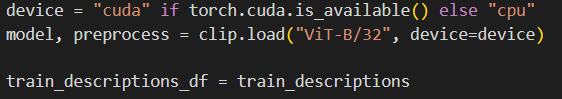
* Preprocess the image to convert image into Tensor



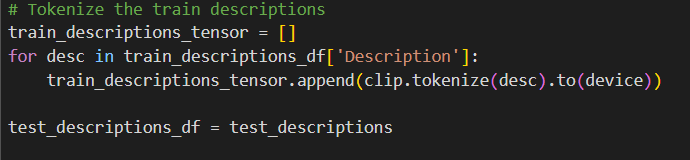
* Save the training descriptions with images as train\_descriptions.csv and test descriptions with images as test\_descriptions.csv



* Load the CLIP model with the preprocessing abilities



* Tokenize the train and test descriptions to convert them into tensors . So that we can load the text tensor and image tensor into the model . Train the model



Dataloader:

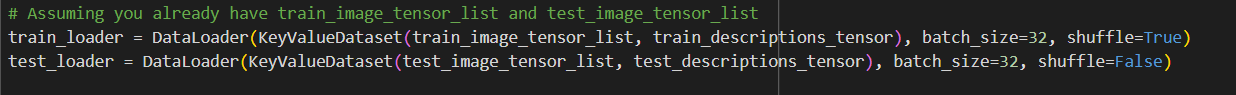
 The KeyValueDataset class takes two lists: image\_tensor\_list and description\_tensor\_list.

 The \_\_len\_\_ method returns the length of the dataset.

 The \_\_getitem\_\_ method retrieves an image and description by index, resizes the image, and pads the description to a fixed length (77 in this case).

 train\_image\_tensor\_list and test\_image\_tensor\_list are lists (or tensors) of images.

 train\_descriptions\_tensor and test\_descriptions\_tensor are lists (or tensors) of text descriptions, each text tensor having a shape like [1, text\_length].

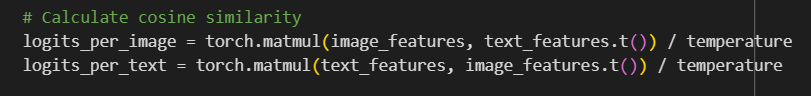


LOSS FUNCTION:

**Normalization**:

* The image and text features are normalized to unit vectors. This is done using the F.normalize function from PyTorch, which normalizes the features along the specified dimension (dim=-1 means the last dimension).
* Normalization helps to ensure that the cosine similarity (dot product of normalized vectors) is within the range [-1, 1].

**Cosine Similarity Calculation**:



* The cosine similarity between image and text features is calculated using matrix multiplication (torch.matmul).
* logits\_per\_image represents the similarity scores for each image against all text descriptions.
* logits\_per\_text represents the similarity scores for each text description against all images.
* The scores are scaled by dividing by a temperature parameter, which controls the range of the logits. A lower temperature makes the model more confident, leading to larger differences between logits.

**Cross Entropy Loss**:

* Cross-entropy loss is computed between the logits and the labels for both image-to-text and text-to-image similarity scores.
* loss\_image measures how well each image matches its correct text description.
* loss\_text measures how well each text description matches its correct image.

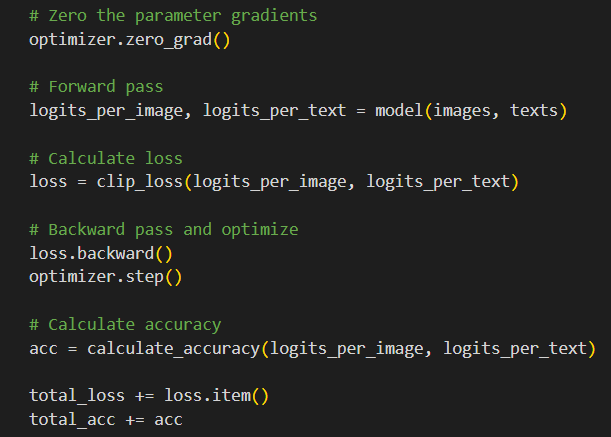
Functions of CLIP loss:

The clip\_loss function implements a contrastive loss for training a model to align image and text representations. The key steps involve:

* Normalizing the feature vectors.
* Calculating cosine similarities (logits) between images and texts.
* Creating labels for the pairs.
* Computing cross-entropy loss for both directions (image-to-text and text-to-image).
* Averaging the losses to get the final loss value used for training.

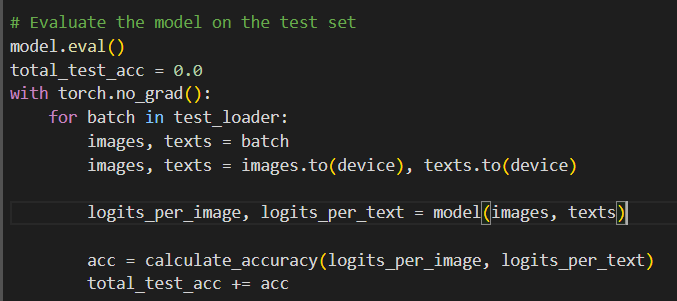
Training and Testing:

Training the model:



* The Adam optimizer from PyTorch is initialized with the model's parameters and a learning rate of 1e-4.
* The number of epochs for training is set to 10.
* The training loop iterates over the number of epochs.
* model.train() sets the model to training mode (important for certain layers like dropout and batch normalization).
* total\_loss and total\_acc are initialized to accumulate the loss and accuracy over the entire epoch.
* The loop iterates over batches of data from train\_loader, transferring the images and texts tensors to the specified device (e.g., GPU).
* The clip\_loss function calculates the contrastive loss between the image and text logits.
* The loss's gradients with respect to the model parameters are computed via backpropagation.
* The optimizer updates the model parameters based on the computed gradients.
* The batch loss and accuracy are added to the total loss and accuracy for the epoch.

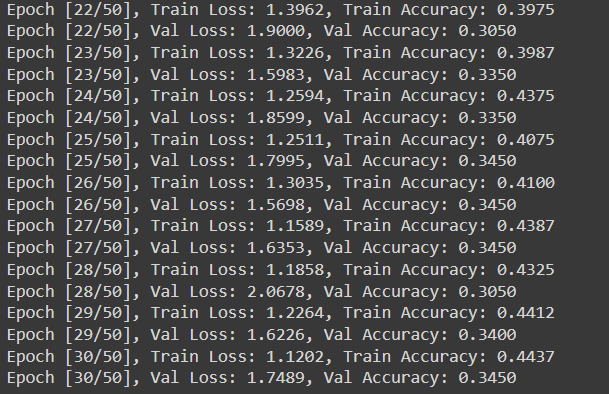
TESTING THE MODEL:



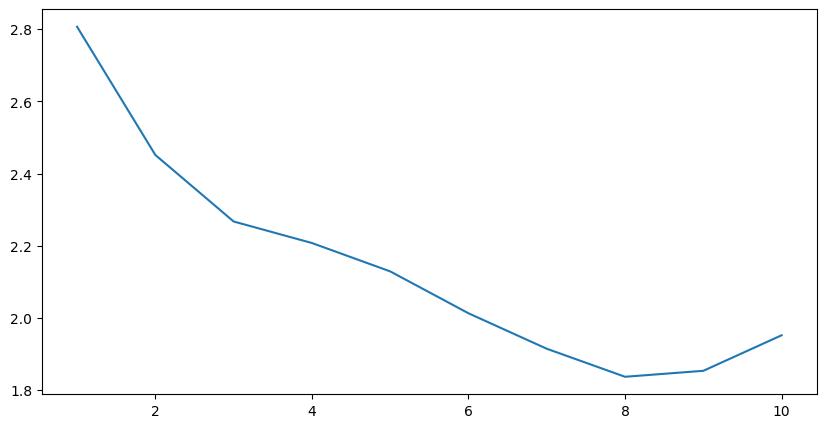
* model.eval() sets the model to evaluation mode. This is important because it ensures that layers like dropout and batch normalization behave correctly during evaluation (e.g., not applying dropout and using running statistics for batch normalization).
* total\_test\_acc is initialized to accumulate the accuracy over the entire test set.
* with torch.no\_grad(): is used to disable gradient calculation, which reduces memory consumption and speeds up computation during evaluation since gradients are not needed.
* The loop iterates over batches of data from test\_loader.
* images and texts tensors are transferred to the specified device (e.g., GPU).
* The calculate\_accuracy function computes the accuracy of the model's predictions for the current batch. This function should compare the logits to the ground truth labels to determine how many predictions are correct.

OUTPUTS OF THE TRAINING AND VALIDATION MODEL:

TRAINING:



LOSS PLOT:

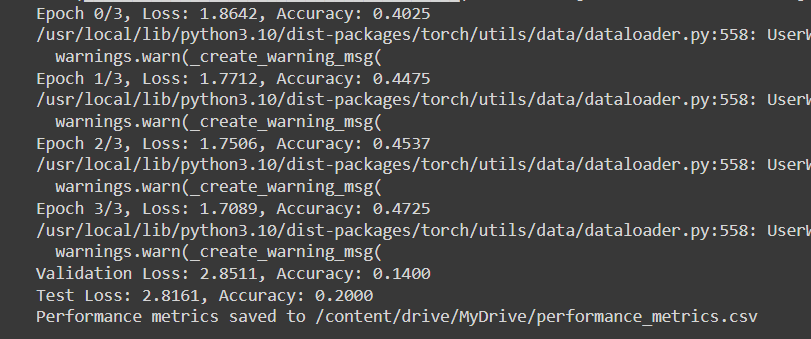


OUTPUT OBSERVATION:

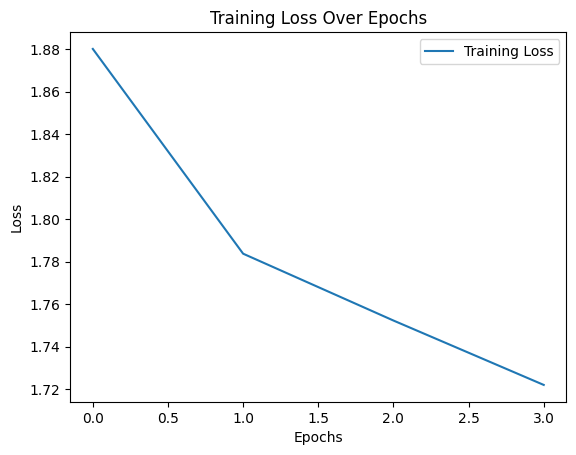
The above code has been run for 50 epochs and the loss has been decreasing gradually. We use SGD optimizer, so that the loss decreases gradually in every step and the accuracy also increases. This particular code has 10 descriptions for a particular class. The output of the particular code has been mentioned above

OUTPUT OF CODE HAVING ONE DESCRIPTION PER CLASS:

OUTPUTS OF THE TRAINING AND VALIDATION MODEL:



LOSS PLOT:



OUTPUT OBSERVATION:

The above code has been run for 3 epochs and the loss has been decreasing gradually. We use SGD optimizer, so that the loss decreases gradually in every step and the accuracy also increases. This particular code has 1 description for a particular class. The output of the particular code has been mentioned above. Here we have introduced test model too. We have used the same dataset for both the codes.

### **SUMMARY**

The paper by Alec Radford et al. introduces a novel approach to training visual models using natural language supervision. Traditional methods in computer vision rely heavily on labelled image data, which can be costly and time-consuming to acquire. In contrast, this paper proposes leveraging large-scale natural language datasets, which are more readily available, to train visual models. By formulating pre-training objectives derived from natural language tasks such as predicting masked words or sentences, the authors demonstrate that these models can effectively learn to extract and utilize semantic information from text to improve visual recognition tasks.

Key contributions and findings include:

* **Transferability of Features**: The learned representations from natural language pre-training tasks transfer well to various visual tasks, improving performance and generalization.
* **Efficiency and Scalability**: Utilizing natural language supervision allows for efficient training on large-scale datasets without requiring extensive manual annotation of images.
* **Enhanced Semantic Understanding**: Models trained with natural language supervision exhibit enhanced semantic understanding, capturing nuanced relationships and attributes in visual data.

### **CONCLUSION:**

In conclusion, Alec Radford et al. present a compelling framework for enhancing visual model training through natural language supervision. By leveraging the abundance and richness of natural language data, they address challenges related to data scarcity and annotation costs in computer vision. The results underscore the effectiveness of their approach in improving transfer learning capabilities across diverse visual recognition tasks. Future research directions could explore refining the pre-training objectives further, investigating additional synergies between natural language understanding and computer vision, and applying these techniques to real-world applications such as image captioning, visual question answering, and autonomous systems.

REFERENCE:

1. **"Learning Source-Invariant Deep Hashing Convolutional Neural Networks for Cross-Source Remote Sensing Image Retrieval"** by Yansheng Li, Yongjun Zhang, Xin Huang, and Jiayi Ma

Link: <https://ieeexplore.ieee.org/document/8385104>

1. **“Learning Transferable Visual Models From Natural Language Supervision**" by Alec Radford et al.

**Link:** [**https://arxiv.org/abs/2103.00020**](https://arxiv.org/abs/2103.00020)

1. [**https://www.udemy.com/course/machinelearning/?couponCode=LETSLEARNNOWPP**](https://www.udemy.com/course/machinelearning/?couponCode=LETSLEARNNOWPP)
2. Y. Ma et al., “Remote sensing big data computing: Challenges and opportunities,” Future Gener. Comput. Syst., vol. 51, pp. 47–60, Oct. 2015.
3. M. Chi, A. Plaza, J. A. Benediktsson, Z. Sun, J. Shen, and Y. Zhu, “Big data for remote sensing: Challenges and opportunities,” Proc. IEEE, vol. 104, no. 11, pp. 2207–2219, Nov. 2016.