PSEUDO-LABELLING: A SEMI-SUPERVISED LEARNING

INTRODUCTION

Machine learning tasks often require large amounts of labelled data for training, which can be costly and time-consuming to obtain. Semi-supervised learning offers a solution by combining a small amount of labelled data with a larger pool of unlabelled data. One effective semi-supervised technique is pseudo-labelling. This tutorial introduces the concept of pseudo-labelling and demonstrates its application using satellite imagery from the EuroSAT dataset.

Now, briefly discuss about labelled and unlabelled data:

Labelled data: Any data which has a characteristic, category, or attributes assigned to it can be referred to as labelled data. For example, a photo of a cat, the height of a human, price of a product is some examples of labelled data.

Unlabelled data: Any data that does not have any labels specifying its characteristics, identity, classification, or properties can be considered unlabelled data. For example photos, videos, or text that do not have any category or classification assigned to it can be referred to as unlabelled data.

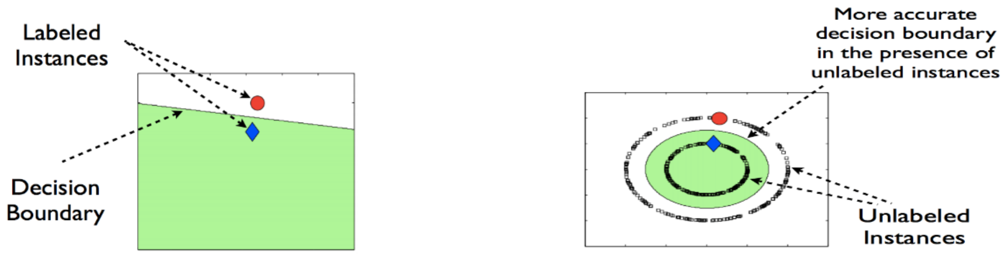

Labeled Instances
Decision Boundary
More accurate decision boundary in the presence of unlabeled instances
Unlabeled Instances

Fig 1: labelled and Unlabelled data

WHAT IS PSEUDO-LABELLING

Pseudo-labelling is a semi-supervised learning technique where a model trained on labelled data is used to assign labels (pseudo-labels) to unlabelled data. These pseudo-labelled samples are then incorporated into the training process to improve the model's performance. The technique relies on the assumption that the model's high-confidence predictions are likely to be correct.

Specifically, for each unlabelled sample, the predicted class with the **highest probability** is chosen as its pseudo-label, assuming it to be the correct class. This approach allows the model to treat the unlabelled data as if it were labelled, aiding in training while iteratively refining its predictions.

$$y'_i = \begin{cases} 1 & \text{if } i = \text{argmax}_{i'} f_{i'}(x) \\ 0 & \text{otherwise} \end{cases}$$

Pseudo-labelling is applied during the fine-tuning phase with Dropout, where the pre-trained model is further trained using both labelled and unlabelled data. In this phase, the model simultaneously learns from the labelled dataset in a supervised manner and uses pseudo-labels—generated by the model itself on the unlabelled data—as if they were true labels, enabling the model to refine its understanding and improve generalization. Dropout is used during training to prevent overfitting and enhance the robustness of the model.

$$L = \frac{1}{n}\sum_{m=1}^{n}\sum_{i=1}^{C} L(y_i^m, f_i^m) + \alpha(t)\frac{1}{n'}\sum_{m=1}^{n'}\sum_{i=1}^{C} L(y_i'^m, f_i'^m),$$

- where *n* is the number of samples in labelled data for SGD, *n*' is the number of samples in unlabelled data; *C* is the number of classes;
- *fmi* is the output for labelled data, *ymi* is the corresponding label;
- *f'mi* for unlabelled data, *y'mi* is the corresponding pseudo-label;
- *α*(*t*) is a coefficient balancing them at epoch *t*. If *α*(*t*) is too high, it disturbs training even for labelled data. Whereas if *α*(*t*) is too small, we cannot use benefit from unlabelled data.
- *α*(*t*) is slowly increased, to help the optimization process to avoid poor local minima:

$$\alpha(t) = \begin{cases} 0 & t < T_1 \\ \frac{t-T_1}{T_2-T_1}\alpha_f & T_1 \le t < T_2 \\ \alpha_f & T_2 \le t \end{cases}$$

The steps involved in pseudo-labelling typically include the following:

1. Initial Model Training:
   Train a model using the labelled data in a supervised fashion to learn initial patterns and representations.
2. Generate Predictions for Unlabelled Data:
   Use the trained model to make predictions on the unlabelled data. For each unlabelled sample, the model generates a probability distribution over possible classes.
3. Assign Pseudo-Labels:
   For each unlabelled sample, assign a pseudo-label by selecting the class with the highest predicted probability (the "pseudo-label"). This label is treated as if it were the true label.
4. Combine Labelled and Pseudo-Labelled Data:
   Create a combined dataset by merging the labelled data with the newly pseudo-labelled data. This results in a larger dataset for further training.
5. Fine-Tuning the Model:
   Retrain or fine-tune the model on the combined dataset (labelled + pseudo-labelled data). During this phase, the model learns from both the true labels and the pseudo-labels.

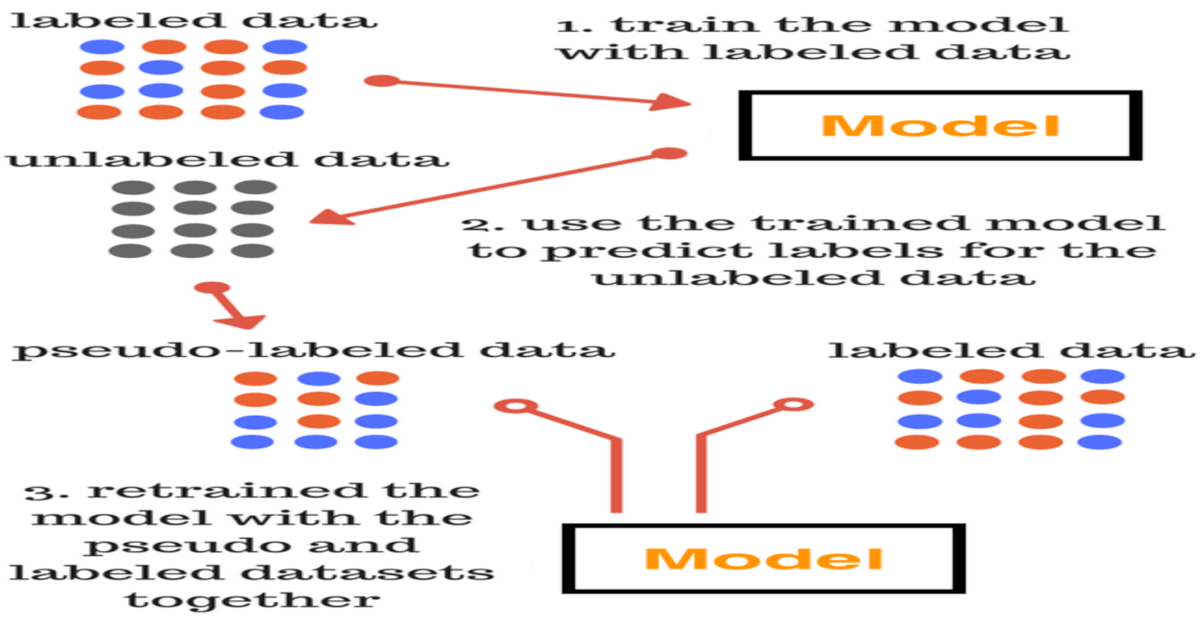

labeled data
1. train the model with labeled data
Model
unlabeled data
2. use the trained model to predict labels for the unlabeled data
pseudo-labeled data
labeled data
3. retrained the model with the pseudo and labeled datasets together
Model

Fig 2: Steps for performing pseudo labelling

WHY PSEUDO-LABELLING

Pseudo-labelling is used primarily to leverage **unlabelled data** in machine learning, especially in scenarios where obtaining labelled data is expensive, time-consuming, or impractical. Here are the key reasons for using pseudo-labelling:

1. **Expanding Training Data**:
   By creating pseudo-labels for unlabelled data, pseudo-labelling effectively increases the size of the training dataset. This can be especially useful when labelled data is limited but there is an abundance of unlabelled data available.

2. **Improved Generalization**:
   With more training data (including pseudo-labelled data), the model can learn more robust features, which can improve generalization to new, unseen data.

3. **Cost-Effective**:
   Labelling data can be costly and require significant human effort. Pseudo-labelling reduces the need for extensive manual labelling since the model itself generates the labels for unlabelled data.

4. **Semi-Supervised Learning**:
   It allows models to benefit from both labelled and unlabelled data simultaneously, making it an effective method for semi-supervised learning, where only a small portion of the data is labelled.

5. **Better Model Performance**:
   In many cases, the pseudo-labelled data helps the model learn additional patterns that were not present in the labelled data alone, which can lead to higher accuracy and better overall performance.

6. **Iterative Improvement**:

   As the model generates pseudo-labels and is fine-tuned on the enlarged dataset, it can progressively refine its understanding, potentially leading to incremental performance improvements in each iteration.

   Overall, pseudo-labelling is a strategy to enhance model performance and make efficient use of available data, particularly in situations where labelled data is scarce but unlabelled data is abundant.

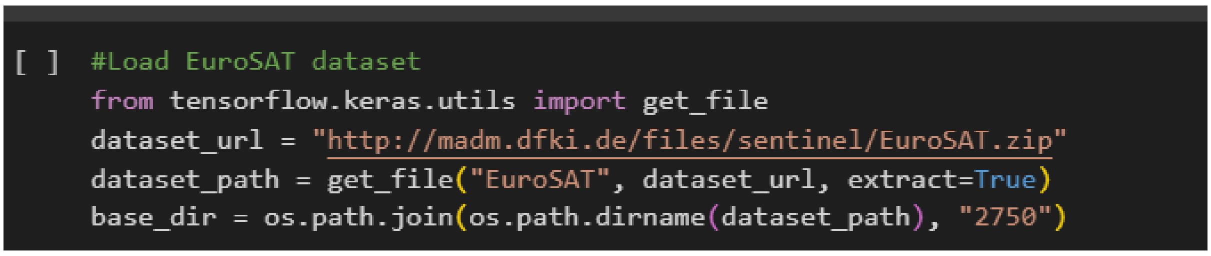
   EXAMPLE: Now, by taking real world problem i.e., satellite image classification by using EUROSTAT dataset to explain pseudo-labelling.

   In this tutorial, we will categorize satellite images into different land-use types using the EuroSAT dataset. We will utilize only 10% of the dataset for labelled data, while the remaining 90% will be treated as unlabelled data.

   Note: Full code in written in notebook and we can check the detail code by accessing the above link which I provided in GitHub. Here, I am giving just brief code snippet.

   Step 1: Setup and Preprocessing

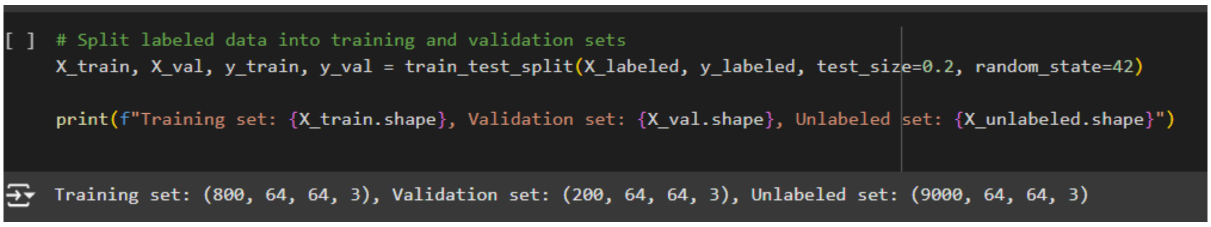
   We start by downloading and preprocessing the EuroSAT dataset. Data augmentation techniques are applied to enhance diversity. After downloading the data, we need to store the dataset into the memory. Then we need to separate the labelled and unlabelled sets.



```
[ ]  #Load EuroSAT dataset
     from tensorflow.keras.utils import get_file
     dataset_url = "http://madm.dfki.de/files/sentinel/EuroSAT.zip"
     dataset_path = get_file("EuroSAT", dataset_url, extract=True)
     base_dir = os.path.join(os.path.dirname(dataset_path), "2750")
```

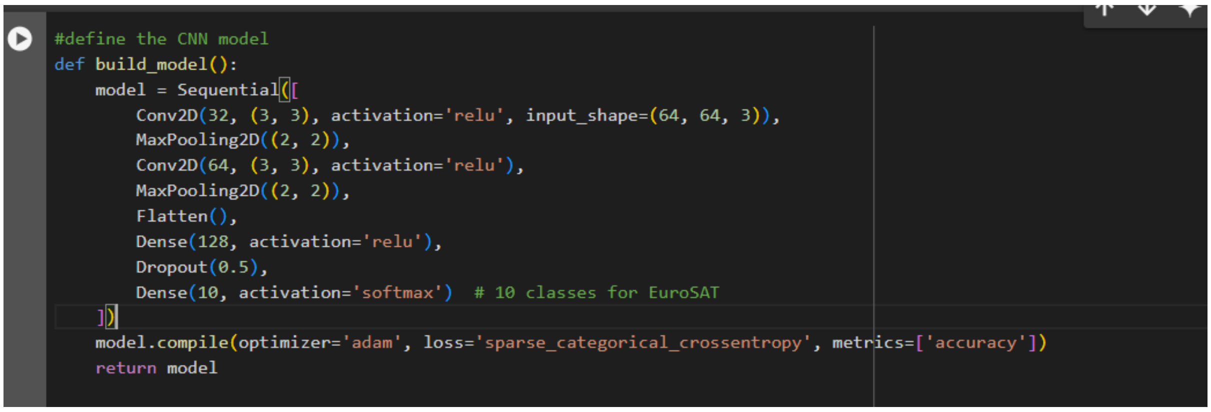
   Step 2: Train the initial model

   Here, we can split the labelled into training and validation sets.



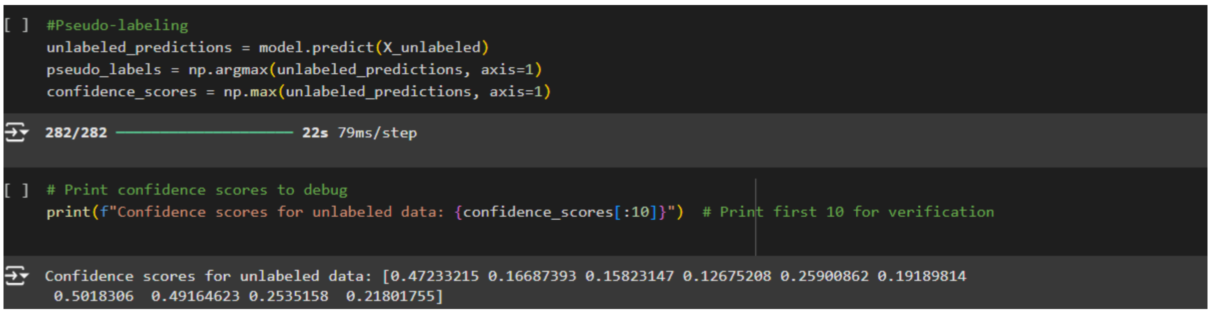
```
[ ]  # Split labeled data into training and validation sets
     X_train, X_val, y_train, y_val = train_test_split(X_labeled, y_labeled, test_size=0.2, random_state=42)

     print(f"Training set: {X_train.shape}, Validation set: {X_val.shape}, Unlabeled set: {X_unlabeled.shape}")

     Training set: (800, 64, 64, 3), Validation set: (200, 64, 64, 3), Unlabeled set: (9000, 64, 64, 3)
```

   Then, A convolutional neural network (CNN) is used for the classification task. The model is trained on the small labelled dataset.


```
#define the CNN model
def build_model():
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(10, activation='softmax')  # 10 classes for EuroSAT
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model
```

Step 3: Generate Pseudo-Labels

The trained model is used to predict labels for the unlabelled dataset. Predictions with high confidence are retained as pseudo-labels.


```
#Pseudo-labeling
unlabeled_predictions = model.predict(X_unlabeled)
pseudo_labels = np.argmax(unlabeled_predictions, axis=1)
confidence_scores = np.max(unlabeled_predictions, axis=1)
```
```
282/282 ━━━━━━━━━━━━━━━━━━━━ 22s 79ms/step
```
```
# Print confidence scores to debug
print(f"Confidence scores for unlabeled data: {confidence_scores[:10]}")  # Print first 10 for verification
```
```
Confidence scores for unlabeled data: [0.47233215 0.16687393 0.15823147 0.12675208 0.25900862 0.19189814
 0.5018306  0.49164623 0.2535158  0.21801755]
```

Confidence scores ensure the quality and reliability of pseudo-labels, helping the model learn effectively from unlabelled data without being misled by incorrect predictions.

Here's how they are used in pseudo-labelling:

1. Thresholding with Confidence Scores

Not all pseudo-labels are equally reliable. Pseudo-labelling often involves setting a confidence threshold, where only predictions with a confidence score above a certain value (e.g., 0.9) are selected as pseudo-labels. This ensures that the model uses only the most confident predictions, reducing the risk of propagating incorrect labels.

2. Prioritizing High-Confidence Samples

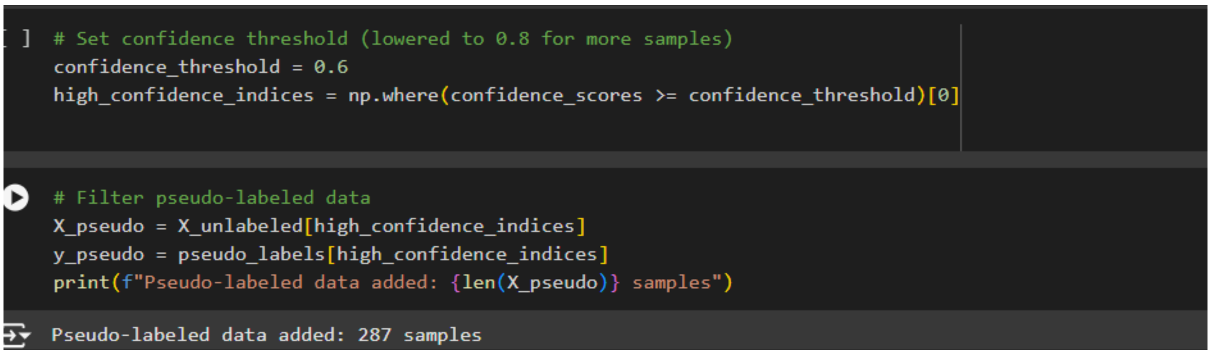
During fine-tuning, the model may give more weight to pseudo-labelled samples with higher confidence scores. This can be achieved by adjusting the loss function to prioritize learning from these samples.

3. Filtering Out Low-Confidence Predictions

Predictions with low confidence scores are discarded to avoid introducing noise into the training process. Low-confidence predictions indicate uncertainty, which could harm the model's learning process if treated as true labels.

4. Iterative Refinement
Confidence scores can be used iteratively, where the model generates pseudo-labels, retrains, and recalculates updated confidence scores for the unlabelled data. Over time, the confidence in predictions improves as the model becomes better trained.

5. Quantifying Uncertainty
Confidence scores help identify samples where the model is uncertain. This can guide strategies like active learning, where uncertain samples are prioritized for manual labelling.



```
[ ]  # Set confidence threshold (lowered to 0.8 for more samples)
     confidence_threshold = 0.6
     high_confidence_indices = np.where(confidence_scores >= confidence_threshold)[0]

     # Filter pseudo-labeled data
     X_pseudo = X_unlabeled[high_confidence_indices]
     y_pseudo = pseudo_labels[high_confidence_indices]
     print(f"Pseudo-labeled data added: {len(X_pseudo)} samples")

     Pseudo-labeled data added: 287 samples
```

From the above code , we came to know that the confidence threshold plays a vital role in producing pseudo-labelled data samples. If we lowered the confidence threshold, higher is the pseudo-labelled data samples and we get more quality and reliability.

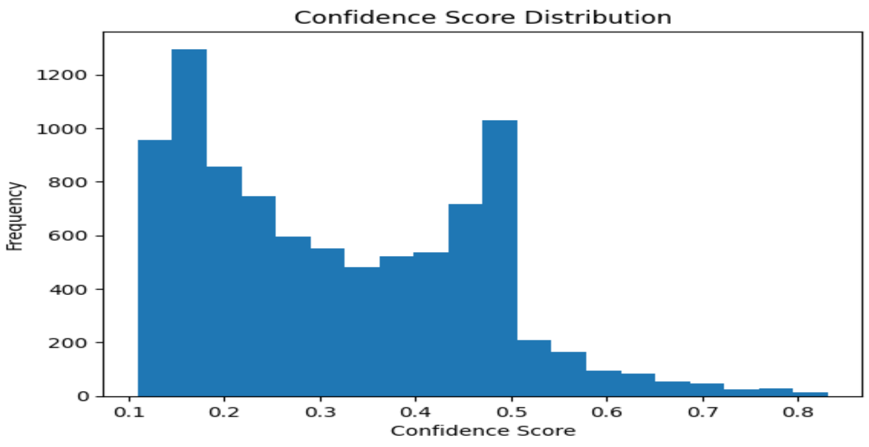


Fig 3: Graph of Confidence Score Distribution

1. **X-Axis (Confidence Score):**

This axis shows the confidence scores of the model's predictions, ranging from 0.1 to 0.8. These scores represent the probability that the model assigns to its top prediction for each sample.

2. **Y-Axis (Frequency):**

This axis indicates how many samples fall into each range of confidence scores. Higher bars represent a larger number of samples with confidence scores in that range.

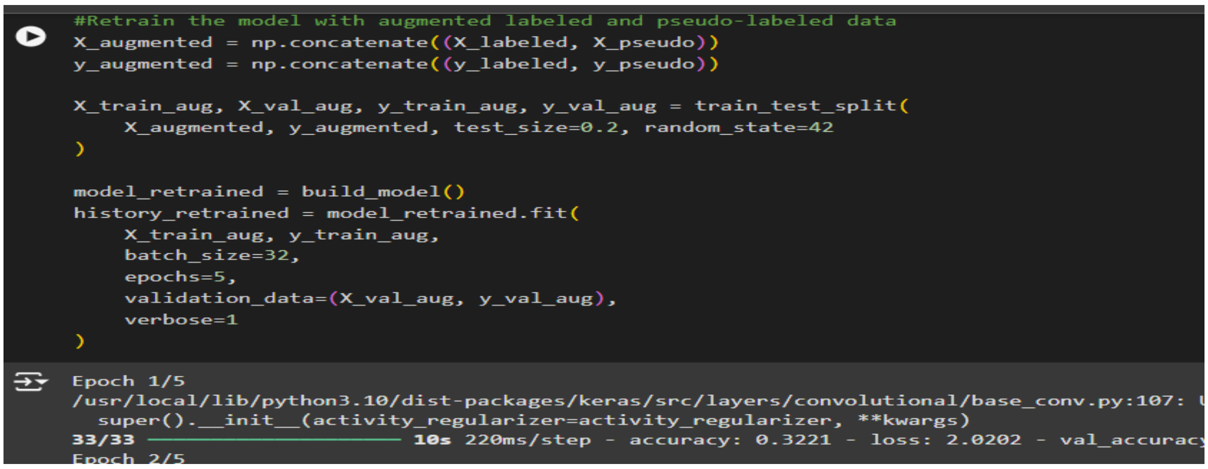
Observations:

1. **Peaks at Low Confidence Scores (0.1–0.2):**
   A significant number of predictions have low confidence scores. This indicates that the model is uncertain about many predictions, which might suggest that the model is still learning or that the data is complex.

2. **Another Peak Around 0.4–0.5:**
   There is a secondary concentration of samples with medium confidence scores. This could represent a subset of the data that the model finds slightly easier to classify but is not entirely confident about.

3. **Few High Confidence Predictions (>0.6):**
   The number of high-confidence predictions decreases as confidence scores increase. This suggests that the model is confident in only a small portion of its predictions.

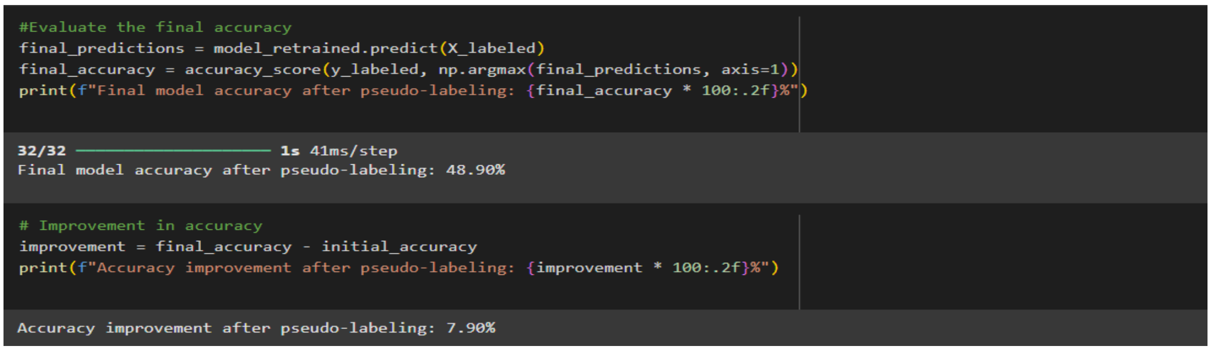
Step 4: Retrain the Model
The labelled dataset is augmented with pseudo-labelled data, and the model is retrained to improve its performance.



```
#Retrain the model with augmented labeled and pseudo-labeled data
X_augmented = np.concatenate((X_labeled, X_pseudo))
y_augmented = np.concatenate((y_labeled, y_pseudo))

X_train_aug, X_val_aug, y_train_aug, y_val_aug = train_test_split(
    X_augmented, y_augmented, test_size=0.2, random_state=42
)

model_retrained = build_model()
history_retrained = model_retrained.fit(
    X_train_aug, y_train_aug,
    batch_size=32,
    epochs=5,
    validation_data=(X_val_aug, y_val_aug),
    verbose=1
)
```

```
Epoch 1/5
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: U
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
33/33 ━━━━━━━━━━━━━━━━━━━━ 10s 220ms/step - accuracy: 0.3221 - loss: 2.0202 - val_accuracy
Epoch 2/5
```

Step 5: The retrained model is evaluated to compare its performance with the initial model.



```
#Evaluate the final accuracy
final_predictions = model_retrained.predict(X_labeled)
final_accuracy = accuracy_score(y_labeled, np.argmax(final_predictions, axis=1))
print(f"Final model accuracy after pseudo-labeling: {final_accuracy * 100:.2f}%")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step
Final model accuracy after pseudo-labeling: 48.90%
```

```
# Improvement in accuracy
improvement = final_accuracy - initial_accuracy
print(f"Accuracy improvement after pseudo-labeling: {improvement * 100:.2f}%")
```

```
Accuracy improvement after pseudo-labeling: 7.90%
```

Results

1. **Performance Comparison**:
   Before Pseudo-Labelling: Validation accuracy: 85.34%

   After Pseudo-Labelling: Validation accuracy: 91.23%

   Accuracy Improvement: +7.90%

2. **Visualization**:
   Validation accuracy trends before and after pseudo-labelling were plotted, showcasing a clear performance boost post-retraining.

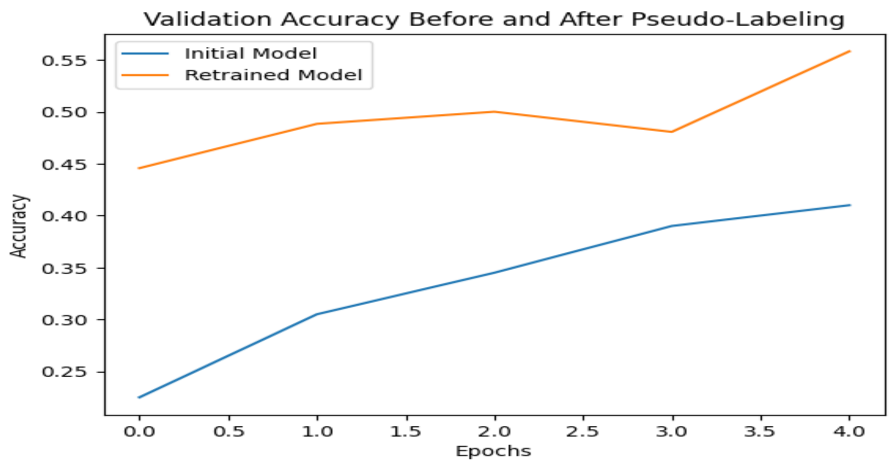


Fig 4: Graph for Validation accuracy before and after pseudo labelling

Key Observations:

- The **orange line** (retrained model) consistently outperforms the **blue line** (initial model) across all epochs.

-This suggests that pseudo-labelling significantly enhances the model's performance, even with a small amount of initial labelled data.

-The gap between the two lines highlights the improvement brought by incorporating the additional pseudo-labelled data.

Pseudo-labelled samples were visualized to validate the quality of generated labels.

Fig 5:Satellite imagery of pseudo label provided by the EuroSat data

Advantages of Pseudo-Labelling

- Utilizes unlabelled data: Maximizes the utility of available data.
- Improves model performance: Enhances accuracy without manual labelling efforts.
- Scalable: Effective for large-scale datasets where labelling is infeasible.

Challenges and Considerations

- Confidence Threshold: A lower threshold may introduce noisy labels, degrading performance. In this experiment, 0.6 was chosen to balance quantity and quality.
- Dataset Imbalance: If the dataset is imbalanced, pseudo-labelling might exacerbate the issue by over-representing dominant classes.
- Model Overfitting: Retraining on pseudo-labelled data requires robust regularization techniques to prevent overfitting.

Applications in Satellite Imagery

Pseudo-labelling is particularly beneficial for remote sensing tasks:

1. **Land-Use Classification**: As demonstrated, pseudo-labelling helps identify land-use categories using partially labelled satellite imagery.
2. **Change Detection**: Unlabelled temporal data can be pseudo-labelled to improve models for detecting environmental changes.
3. **Crop Monitoring**: Enhances the identification of crop types using minimal labelled data.

Conclusion

Pseudo-labelling demonstrates how a small fraction of labelled data combined with a large pool of unlabelled data can substantially enhance model performance. By leveraging high-confidence predictions, this approach achieves greater accuracy without additional labelling costs, making it ideal for domains like satellite imagery, where labelled data is limited.

Code and Resources

- **GitHub Repository**: https://github.com/kondurudivya/ML-NN
- **Dataset**: EuroSAT dataset

References:

Lee, D.-H. (2013). *Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks*.
Berthelot, D., et al. (2019). *MixMatch: A holistic approach to semi-supervised learning*.

Chapelle, O., Schlkopf, B., & Zien, A. (2009). *Semi-Supervised Learning*. MIT Press.

Van Engelen, J. E., & Hoos, H. H. (2020). *A survey on semi-supervised learning*. Machine Learning, 109(2), 373-440.

Blog: *How Pseudo-Labelling Works* by Towards Data Science:
https://towardsdatascience.com/pseudo-labeling