Jay Nakrani 18057042

How Neural Networks learn from both Labeled and Unlabeled data

Introduction

Welcome to the exciting tutorial of learning Neural Networks with Pseudo-Labels!

Let’s start the tutorial with the imagination. Have you ever imagine how we can train the mighty neural networks when we actually don’t have enough labeled data? Well in this tutorial I am going to present you a smart trick to do that which is called ‘pseudo-labeling’, which gives power to machines to learn from both labeled and unlabeled data. Just like us humans, how we learn from books and real-world experiences!

What is Labeled data and Unlabeled data? What is the difference between them?

Let’s quickly understand labeled and unlabeled data before focusing on pseudo-labeling.

What is labeled data?

"Cat"

All the data with the labels are the labeled data that is the simplest way to explain it! In this, each data comes with a clear label. They are like text book with answers. For example, an image of a cat with the label ‘cat’. This type of data makes it easy to train a neural network, but they are more time consuming.

What is unlabeled data?

" "

Simple, all the data without labels are unlabeled data. They are like a book without answers. The image besides is a bull dog saying hi! Well, that we can see but machines are not able to see it. They can read it but they don’t know what the correct response are. In the real world we have tons of unlabeled data, but it is hard for machines to learn from it without labels.

Imagine if you have huge pile of data, but only a small part of it is labeled. That makes the process worst but what if we could make our neural network guess the missing labels and use them for learning? That's exactly what pseudo-labeling does! In this tutorial we will train a Multilayer Neural Network (MLNN) using this method and we will compare its performance with the performance of labeled data alone.

By the end of this learning, you are able to understand how pseudo-labeling works and also able to write your own code to implement it. Whether you are someone looking to sharpen your skills or beginner, this tutorial will make semi-supervised learning easy and fun to grasp. Let's start the tutorial with understanding the core idea.

What is Supervised and Unsupervised learning?

Supervised learning and Unsupervised learning are the main ways that machines learn. These are like how we humans learn with the guidance (Supervised) and through self-discovery (Unsupervised).

Supervised Learning

This is like learning with a teacher. Let's get yourself to the math class! Every time you solve a problem, your teacher will tell you whether it is wrong or right. If you make a mistake, they are there to correct it. And over time you will improve your problem solving skill because you're learning with the guidance. That is exactly how supervised learning works in machines.

In this scenario, a model is trained on labeled data and all inputs has correct answers. For instance, if you want your model to recognize cats and dogs and you train your model with thousands of labeled images of cats and dogs. The model will learn patterns from that and it will be able to classify new images of dogs and cats correctly.

Supervised learning works well for machines but what about you? Imagine labeling millions of images by hand, it seems slow and impractical!

Unsupervised learning

Unsupervised learning is like learning without a teacher. Let's say you are seeing different animals and notice that some has fur, some have feathers and some have scales. Even though you don't know their names, you can still sort them into groups based on how they look, by self-discovery. This is how unsupervised learning works.

In unsupervised learning, data has no labels means there is nothing to tell the model what things are. So model finds patterns on its own. One way it does this is by grouping similar things together, this way is called as clustering.

For example, if we provide model with no labels, it might still divide all the dog pictures in one group and all the cat pictures in another, even though it doesn't know their names.

Unsupervised learning is useful when we don't have labeled data, but the model might not always group things correctly as there is no teacher.

Semi-supervised learning – the best of both!

Now imagine what if we combine both methods together? Sounds great, isn't it. We can use a small portion of labeled data to guide the learning process and let the model learn on itself from the large amount of unlabeled data. That is exactly what semi-supervised learning does.

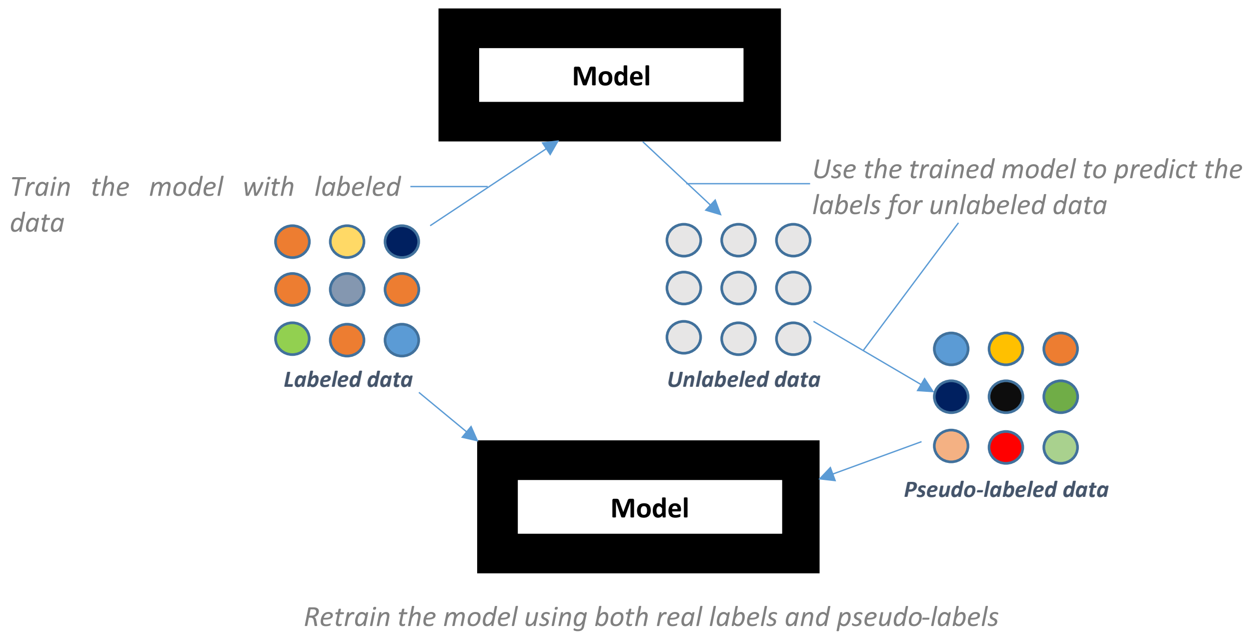
In semi-supervised learning, instead of labeling everything, we train the model on a small portion of labeled dataset and let the model improve itself through unlabeled data. Just like humans! Semi-supervised learning is faster, cheaper and often effective as fully supervised learning.

Now the exciting part comes - pseudo-labeling, the most powerful techniques in semi-supervised learning! Whoo… let's dive in!

How pseudo-labeling works in semi-supervised learning

Pseudo-labeling is a simple and powerful technique which helps to turn an unsupervised problem into a supervised one. Through unlabeled data, we let the model make its best guesses and use those guesses as labels for further training.

Here is how it works:


Model
Train the model with labeled data
Use the trained model to predict the labels for unlabeled data
Labeled data
Unlabeled data
Pseudo-labeled data
Model
Retrain the model using both real labels and pseudo-labels

First we give our model a small amount of labeled data to train the neural network. This means we are giving model some examples where we already know the right answers so it can start learning patterns.

Next, we tell model to predict labels for the unlabeled data with the help of labelled data. We can call this process pseudo-labeling as they are not assigned by humans but they were guessed by the model!

Finally, with the help of both labeled data and newly generated pseudo-labeled data, we retrain our model. These means that model is getting more training data, this process helps model to learn better and make more accurate predictions over time. This process keeps going and the accuracy of the model gradually increases! Interesting!

Just like humans, when we want to learn a new language, first we need a teacher to explain things (labeled data). But with time we start getting better, we are able to start guessing the meanings of new words (pseudo-labels) and learning from those guesses. The more we practice, the better we become!

Many researchers have found that this method works really well. In 2013, Dong-Hyun Lee was one of the first researchers to show that pseudo-labeling can improve the model accuracy with the help of small portion of labeled data (Lee, 2013).

Later, in 2020, Xie et al. at Google used pseudo-labels in a method called Noisy Student Training, which helped AI models to learn from their guesses and reach top-level accuracy in image recognition (Xie, 2020).

And now we will learn how to implement pseudo-labeling into our model and will compare the accuracy of semi-supervised learning with pseudo-labels and accuracy of supervised learning with labeled data only.

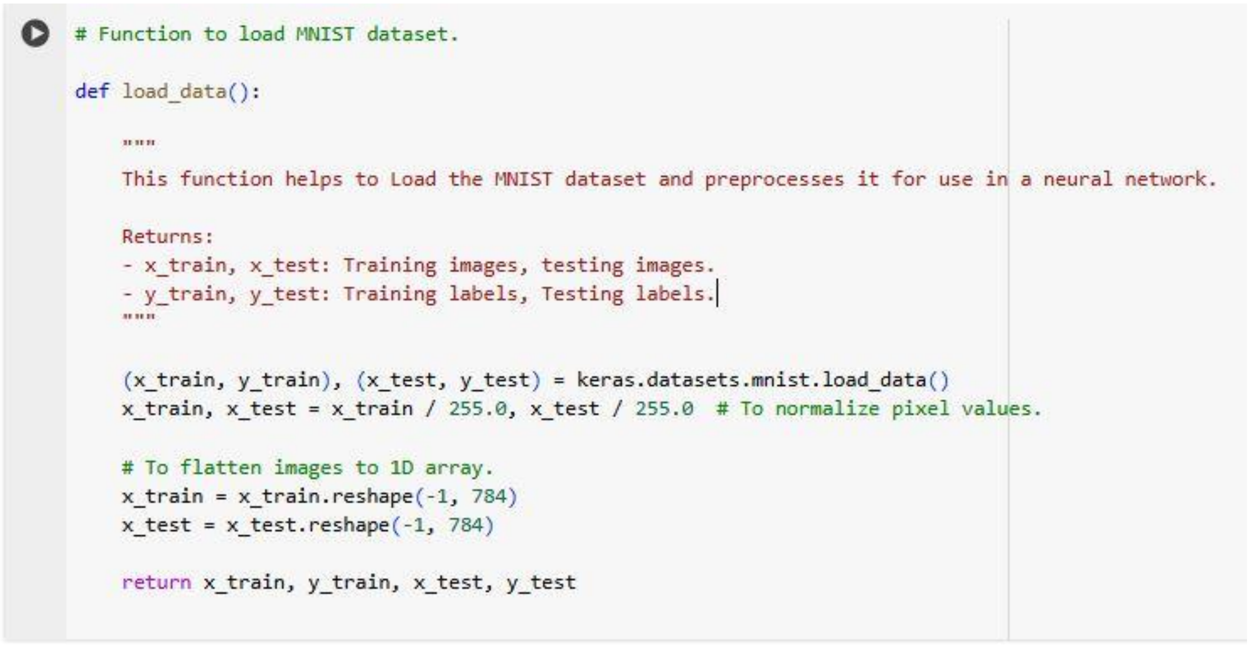
A Neural Network that predicts handwritten digits

Here, we are going to create a neural network model that can recognize handwritten digits, this can be also called as digit classification. For this we are going to use MNIST dataset (LeCun, 1998), this dataset contains 70,000 images of handwritten numbers from 0 to 9. All the images are labelled correctly and they are in 28x28 pixel grid.

The goal here is to teach the neural network to look at this images and predict the right digit. We will use a small set of labeled images to train the network, and a bigger set of unlabeled images for semi-supervised learning. This way, model can improve accuracy overtime. It will recognize patterns in pixels and connect them to the right digit.

Let’s start coding!

After installing all the necessary libraries we can define a helper function to load and prepare the dataset for the neural network. This can be seen in below image.

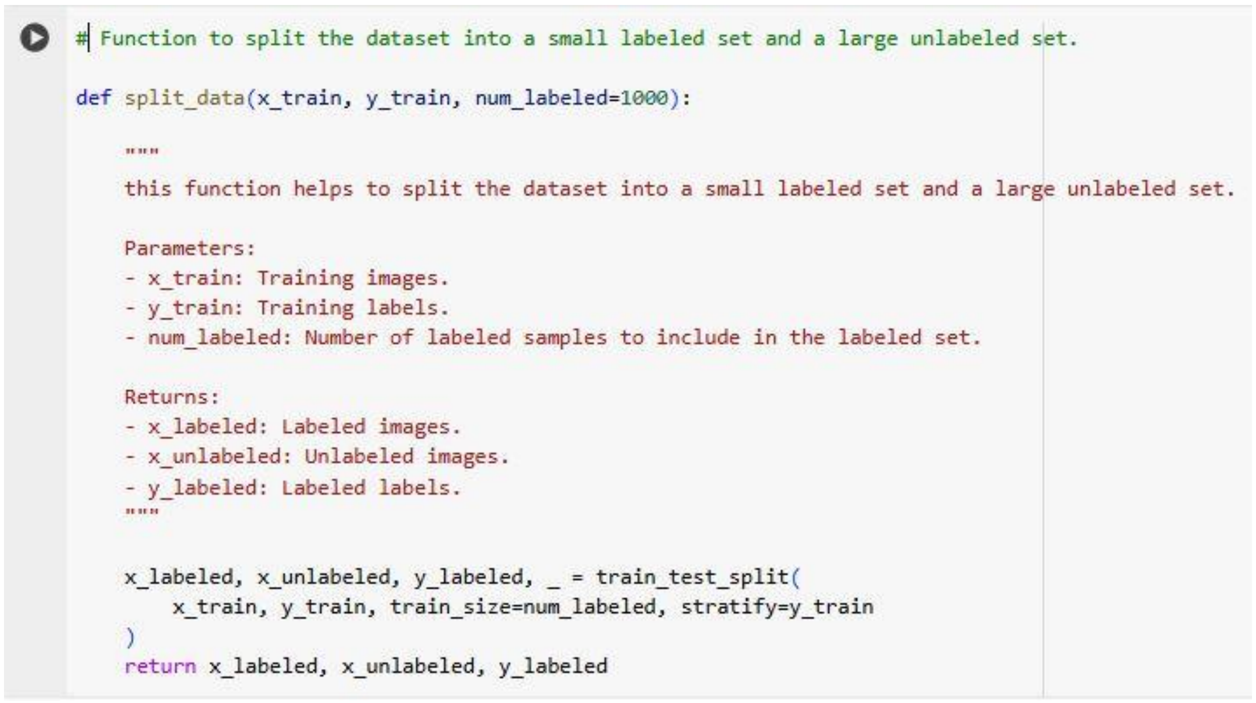


```python
# Function to load MNIST dataset.

def load_data():

    """
    This function helps to Load the MNIST dataset and preprocesses it for use in a neural network.

    Returns:
    - x_train, x_test: Training images, testing images.
    - y_train, y_test: Training labels, Testing labels.
    """

    (x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
    x_train, x_test = x_train / 255.0, x_test / 255.0  # To normalize pixel values.

    # To flatten images to 1D array.
    x_train = x_train.reshape(-1, 784)
    x_test = x_test.reshape(-1, 784)

    return x_train, y_train, x_test, y_test
```

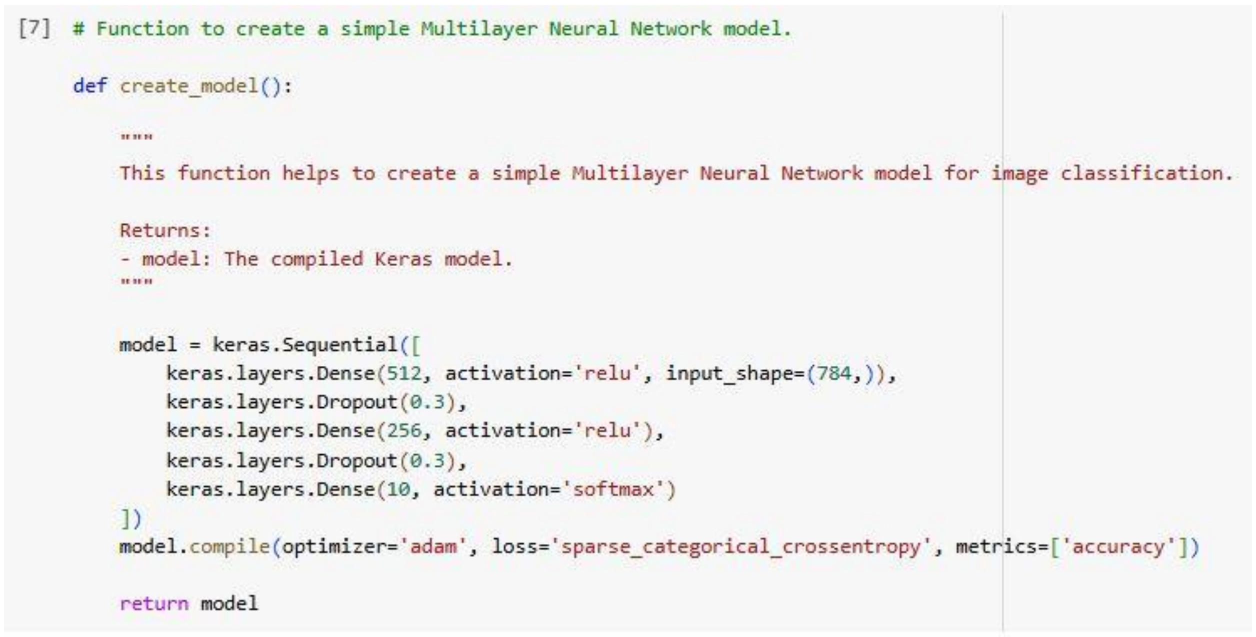
In this function, we load the MNIST data first with their correct labels. Where x_train and x_test will get the image data and y_train and y_test will get the number data (0-9). The images are grayscale with pixel values from 0 to 255. So, we scale the pixel values by dividing it with 255. Now pixel values are ranged from 0 to 1 instead of 0 to 255, this helps the neural network to learn better. And we also need to flatten images into 1D array, as images are originally two dimensional with 28x28 grids and neural network process data in linear format. So images are reshaped into a one-dimensional array with 784 pixels. This way model can treat each pixel as an individual feature.

Then split_data and create_model functions help to split the dataset and build a simple neural network. The split_data function divides dataset into two sets: a small labeled set and a large unlabeled set. We will use stratified sampling so that both sets have a balanced mix of digits. We get labeled images, unlabeled images and the labels for the labeled set from this function.

The create_model function creates two hidden layers, one with 512 neurons and second with 256 neurons. These numbers are chosen to give the model enough power to learn from images but not be too complicated. Relu activation helps the model to recognize complex patterns. After each hidden layer, we will add 30% dropout to prevent the model from getting too stuck on the training data and overfitting. The output layer has 10 neurons (one for each digit) with softmax activation which helps the model to pick the most likely digit. Adam optimizer and categorical cross-entropy loss helps to train model efficiently and to check how close the model’s guesses are to the actual digits.

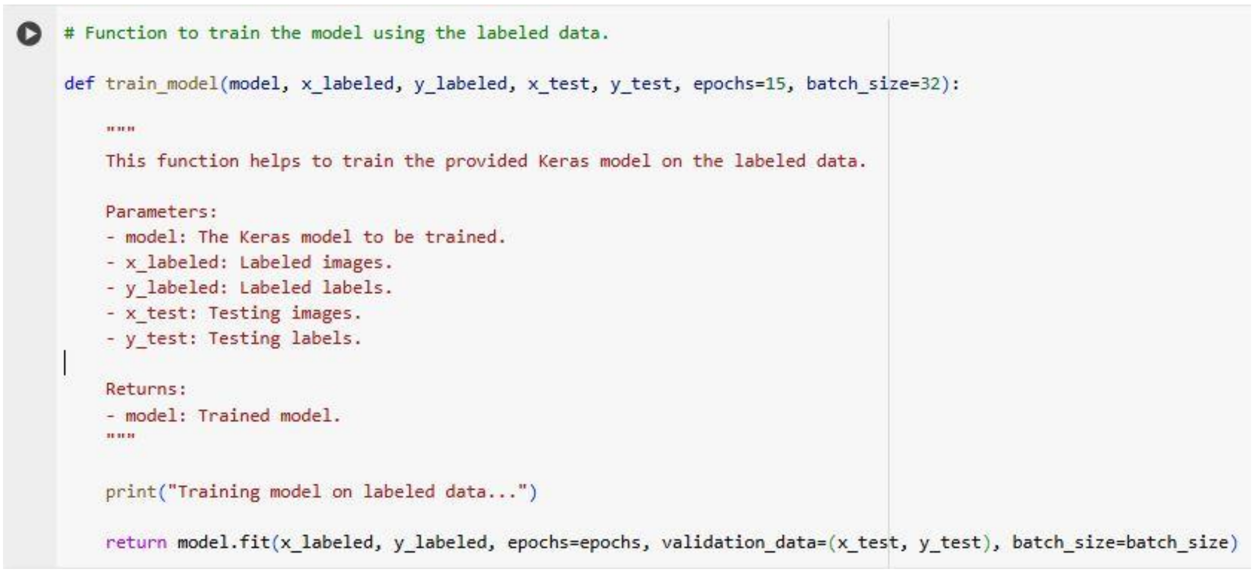


```python
# Function to split the dataset into a small labeled set and a large unlabeled set.

def split_data(x_train, y_train, num_labeled=1000):

    """
    this function helps to split the dataset into a small labeled set and a large unlabeled set.

    Parameters:
    - x_train: Training images.
    - y_train: Training labels.
    - num_labeled: Number of labeled samples to include in the labeled set.

    Returns:
    - x_labeled: Labeled images.
    - x_unlabeled: Unlabeled images.
    - y_labeled: Labeled labels.
    """

    x_labeled, x_unlabeled, y_labeled, _ = train_test_split(
        x_train, y_train, train_size=num_labeled, stratify=y_train
    )
    return x_labeled, x_unlabeled, y_labeled
```



```python
[7] # Function to create a simple Multilayer Neural Network model.

def create_model():

    """
    This function helps to create a simple Multilayer Neural Network model for image classification.

    Returns:
    - model: The compiled Keras model.
    """

    model = keras.Sequential([
        keras.layers.Dense(512, activation='relu', input_shape=(784,)),
        keras.layers.Dropout(0.3),
        keras.layers.Dense(256, activation='relu'),
        keras.layers.Dropout(0.3),
        keras.layers.Dense(10, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

    return model
```

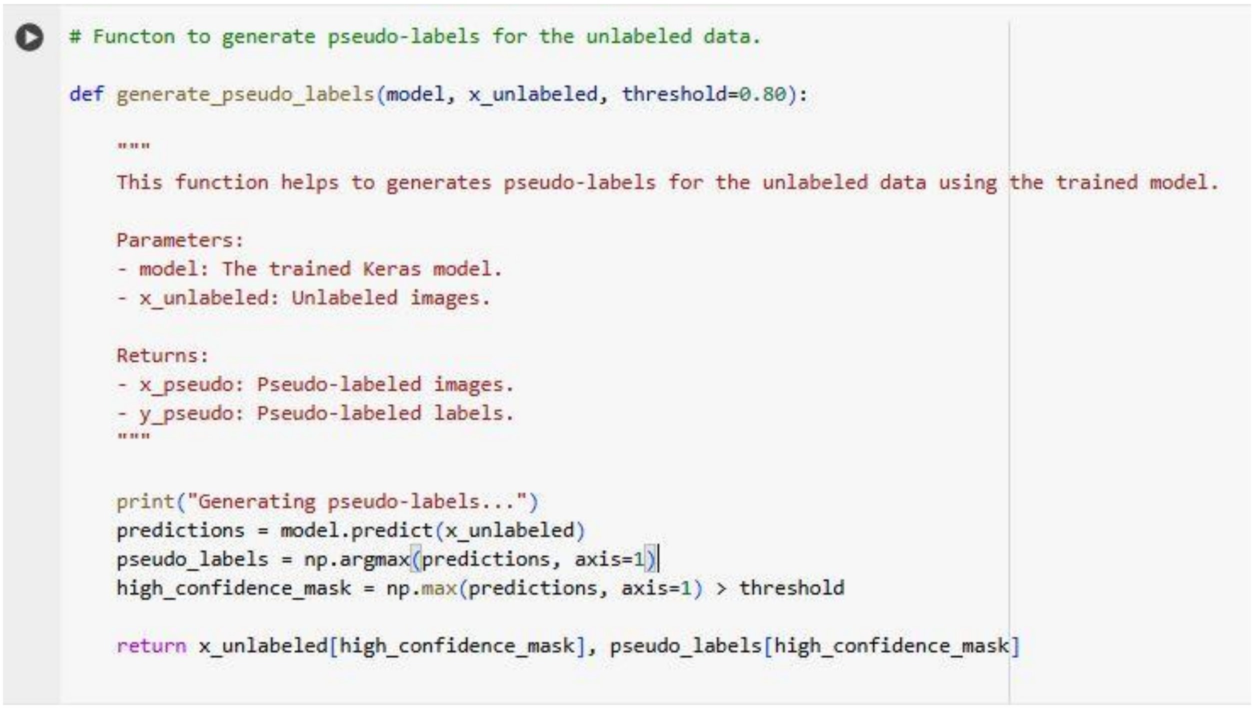
The train_model function trains the model on labeled data. The epochs=15 means the model will go through the entire training data 15 times and batch_size=32 means the model will process 32 images at a time before updating its weights.



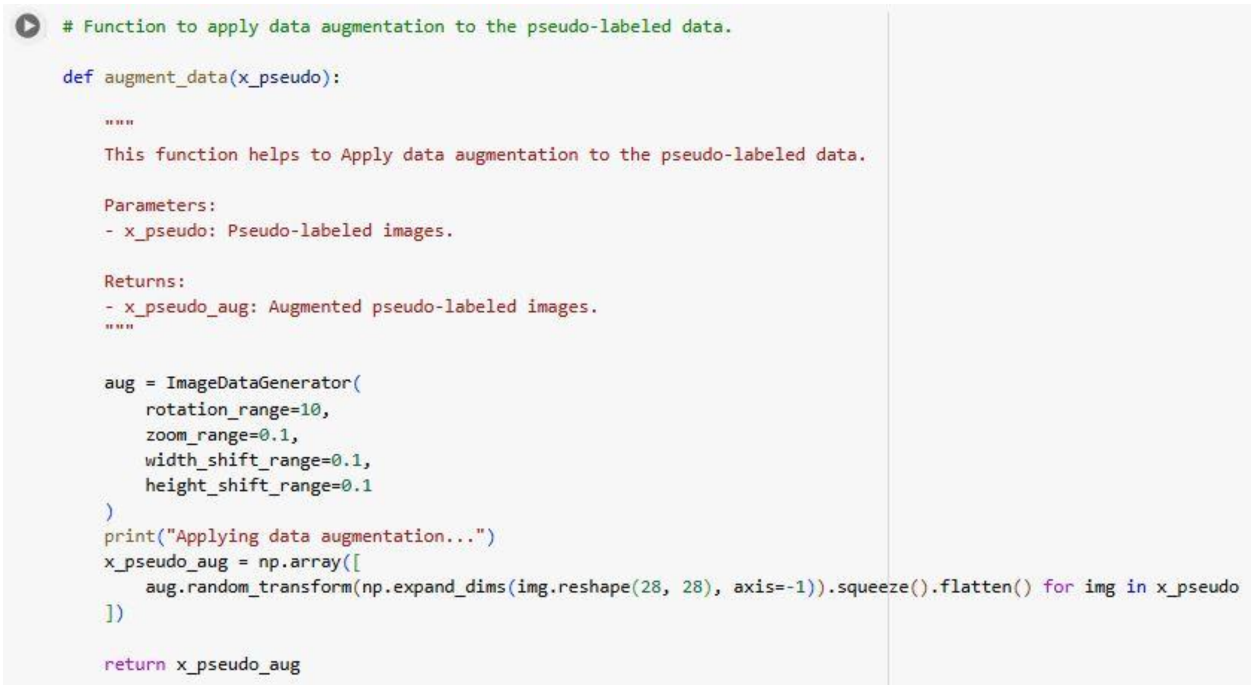
```
# Function to train the model using the labeled data.

def train_model(model, x_labeled, y_labeled, x_test, y_test, epochs=15, batch_size=32):

    """
    This function helps to train the provided Keras model on the labeled data.

    Parameters:
    - model: The Keras model to be trained.
    - x_labeled: Labeled images.
    - y_labeled: Labeled labels.
    - x_test: Testing images.
    - y_test: Testing labels.

    Returns:
    - model: Trained model.
    """

    print("Training model on labeled data...")

    return model.fit(x_labeled, y_labeled, epochs=epochs, validation_data=(x_test, y_test), batch_size=batch_size)
```

The generate_pseudo_labels function makes model to predict the labels for unlabeled data. The np.argmax () helps to pick the digit with the highest predicted probability for each image. The axis=1 is used as we want to look at each image’s prediction across the different digits (0-9). The np.max () helps to find the highest confidence predictions from all, and here we set the threshold to 80%. This way it will select only predictions with confidence above 80% to make the labels more reliable.

The augment_data function helps to make slight changes to pseudo-labeled images. We can use ImageDataGenerator from Keras to apply transformations to images like rotation, zoom, and shifting. Then with the help of NumPy libraries we can reshape the images to 28x28 and add random transformations. After transformations we can flatten them back into 1D arrays. These helps model to learn better by seeing variations of the same image.



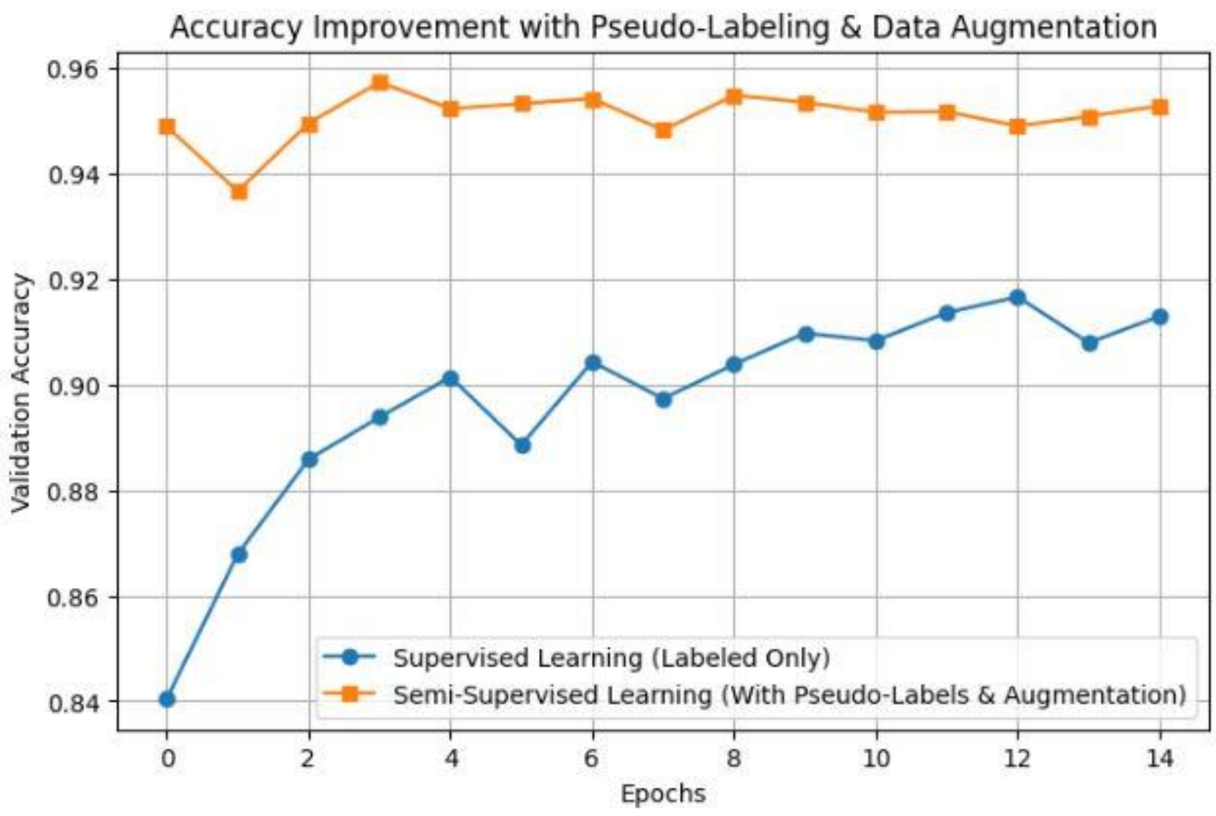
```
# Functon to generate pseudo-labels for the unlabeled data.

def generate_pseudo_labels(model, x_unlabeled, threshold=0.80):

    """
    This function helps to generates pseudo-labels for the unlabeled data using the trained model.

    Parameters:
    - model: The trained Keras model.
    - x_unlabeled: Unlabeled images.

    Returns:
    - x_pseudo: Pseudo-labeled images.
    - y_pseudo: Pseudo-labeled labels.
    """

    print("Generating pseudo-labels...")
    predictions = model.predict(x_unlabeled)
    pseudo_labels = np.argmax(predictions, axis=1)
    high_confidence_mask = np.max(predictions, axis=1) > threshold

    return x_unlabeled[high_confidence_mask], pseudo_labels[high_confidence_mask]
```



```
# Function to apply data augmentation to the pseudo-labeled data.

def augment_data(x_pseudo):

    """
    This function helps to Apply data augmentation to the pseudo-labeled data.

    Parameters:
    - x_pseudo: Pseudo-labeled images.

    Returns:
    - x_pseudo_aug: Augmented pseudo-labeled images.
    """

    aug = ImageDataGenerator(
        rotation_range=10,
        zoom_range=0.1,
        width_shift_range=0.1,
        height_shift_range=0.1
    )
    print("Applying data augmentation...")
    x_pseudo_aug = np.array([
        aug.random_transform(np.expand_dims(img.reshape(28, 28), axis=-1)).squeeze().flatten() for img in x_pseudo
    ])

    return x_pseudo_aug
```

Then we can combine both labeled data and augmented pseudo-data with the help of combine_data function. And we can retrain our model with this combined data through retrain_model function. The evaluate_model function helps to evaluate the final model's performance on the test data.

And with the help of all these functions, we can create main function to compare the accuracy of supervised learning and semi-supervised learning. First, we train the model using a small set of labeled images (supervised-learning). Then, we use pseudo-labeling, where the trained model guesses labels for unlabeled images (semi-supervised learning). After applying data augmentation and combining all the data, we retrain the model to improve its learning. Finally, using the plot_accuracy function, we can create a graph to compare the accuracy of both methods. After following all these process, we finally get the accuracy plot as below. Hurray!



This graph is like a learning race! The blue line (supervised learning) starts slow, improving gradually with labeled data. Meanwhile, the orange line (semi-supervised learning) gets a big boost from pseudo-labeling and data augmentation, staying ahead with better accuracy.

More smartly labeled data = a smarter model!

For full code check my GitHub repository, click here.

References

Lee, D.-H. (2013) ‘Pseudo-Label: The simple and efficient semi-supervised learning method for deep neural networks’. *Proceedings of the ICML 2013 Workshop on Challenges in Representation Learning*. Available at: https://citeseerx.ist.psu.edu/document?doi=798d9840d2439a0e5d47bcf5d164aa46d5e7dc26

Xie, Q., Luong, M.-T., Hovy, E. and Le, Q. V. (2020) ‘Self-training with Noisy Student improves ImageNet classification’. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Available at: https://arxiv.org/abs/1911.04252

LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P., 1998. *Gradient-based learning applied to document recognition*. Proceedings of the IEEE, 86(11), pp.2278–2324. Available at: http://yann.lecun.com/exdb/mnist/