SAiDL ASSIGNMENT -2

COMPUTER VISION

SASWATA MUKHERJEE

4th September, 20XX

CONTENTS

INTRODUCTION

The STL-10 dataset is an image recognition dataset that is generally used for the development of unsupervised feature learning, deep learning and self taught learning algorithms for computer vision. It is derived from the CIFAR-10 with some modifications implemented. There are 10 different classes of images, each of which has fewer labelled examples than the CIFAR-10 dataset but lots of unlabelled images. The resolution of the images is 96 x 96 and they are coloured. The image classes present in the dataset are:

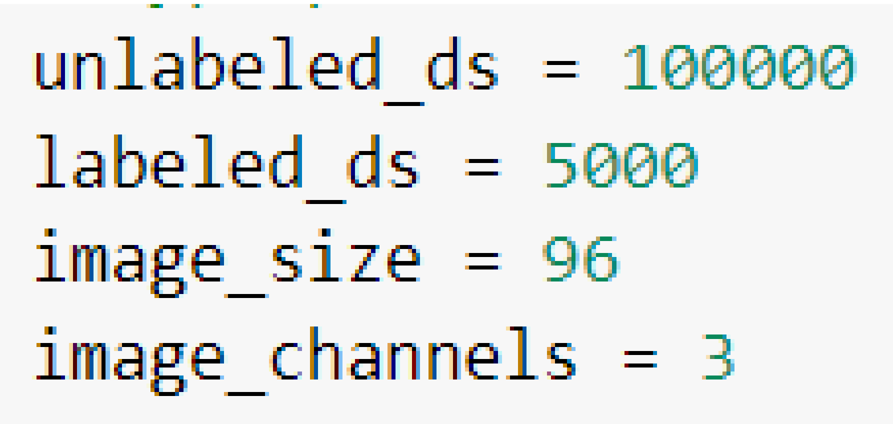
- Airplane
- Bird
- Car
- Cat
- Deer
- Dog
- Horse
- Monkey
- Ship
- Truck

There are **500** training images and **800** test images per class and besides that **100000** unlabelled images for unsupervised learning. In this assignment, the following tasks have been carried out sequentially:

- An image classification model has been trained on the labelled part of the dataset in a **supervised** manner.
- SimCLR is used to learn good image representations from the **unlabelled** data (**self-supervised training**).

SUPERVISED TRAINING

Before carrying out the model training, it is important to preprocess the images in the dataset. For supervised training, the **labeled** part of the dataset has been used.



```
unlabeled_ds = 100000
labeled_ds = 5000
image_size = 96
image_channels = 3
```

The dataset consists of **100000** unlabeled images (unlabeled_ds) and **500** labeled ones (labeled_ds). The size of each image is 96 x 96 pixels (image_size) and the colour of each image is a combination of **three** colours (RGB) having different weights hence image_channels = 3.

The dataset is prepared such that the labeled and unlabeled parts of the dataset are loaded in sync with each other. Both sets are zipped into one variable: **train_ds**

STEPS FOR DATA AUGMENTATION

The most important step of preprocessing in image classification problems is data augmentation. The performance of deep neural networks often improves with the amount of training data available. Data augmentation is a technique to artificially create new data from existing training data. This is done by applying specific techniques to examples from the training data to create new training examples.

Augmentation steps are task specific. The two main steps followed for our specific case has been illustrated below:

1. Cropping and resizing a part of the image to the original resolution.
    a. The class **RandomResizedCrop** is used to carry out this form of image augmentations.
    b. A part of the image is focused on by cropping according to the passed parameters and resized to the original image resolution present in the dataset.
    c. This type of image augmentation will allow the model to learn zoomed in representations of images in which the entire structure may or may not be visible.
    d. Hyperparameters used for this class are:
        i. ***scale****:* tuple of area-range of cropped part.
        ii. ***ratio****:* tuple of aspect-ratio range of the cropped part.

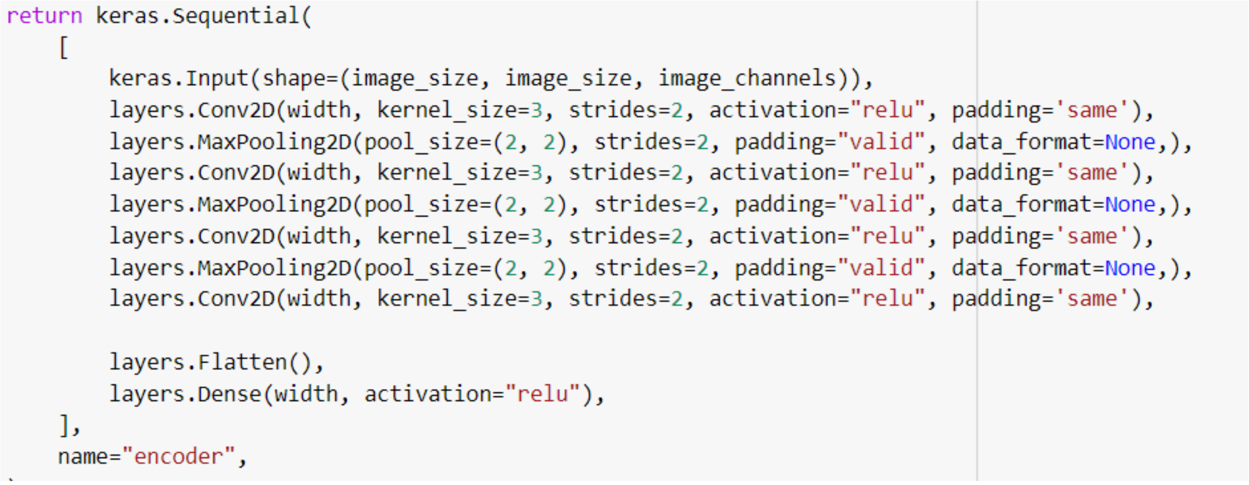
2. Distortion of the colour distribution of the original images.
    a. The class **RandomColorJitter** is used to carry out this form of image augmentations.
    b. Training images are passed through a pipeline that would distort the colour composition of the images.
    c. This enhances the learning of images that might not be present in the conventionally coloured format.
    d. Hyperparameters used for this class are:
        i. ***brightness****:* random extrapolation/interpolation between image and darkness.
        ii. ***contrast****:* random extrapolation/interpolation between image and its mean intensity value.
        iii. ***saturation****:* random interpolation/extrapolation between the image and its grayscale counterpart
        iv. ***hue****:* random shift in hue in hsv colorspace.

3. In addition to these classes, the **RandomGaussianNoise** class is used to add some *'noise'* to the images so that the model can learn better and be able to predict images that might not be very clearly visible.
    a. Hyperparameters used for this class are:
        i. ***stddev****:* standard deviation; introduces gaussian blur in the images.

4. The images are also **flipped horizontally** to enhance the model's ability to identify laterally inverted images of the same classes of objects. Vertical flipping is not done since it is highly unlikely that real images passed would be vertically inverted.

Encoder Architecture

The encoder architecture for supervised training is a basic convolutional neural network consisting of **4 convolutional hidden layers** and **3 max-pooling layers** between each convolutional layer. Each convolutional layer has **128** nodes and the **relu** activation function is implemented on each such layer. The max-pooling layers have been added to summarise the output of each convolutional layer and thus reduce the training time.



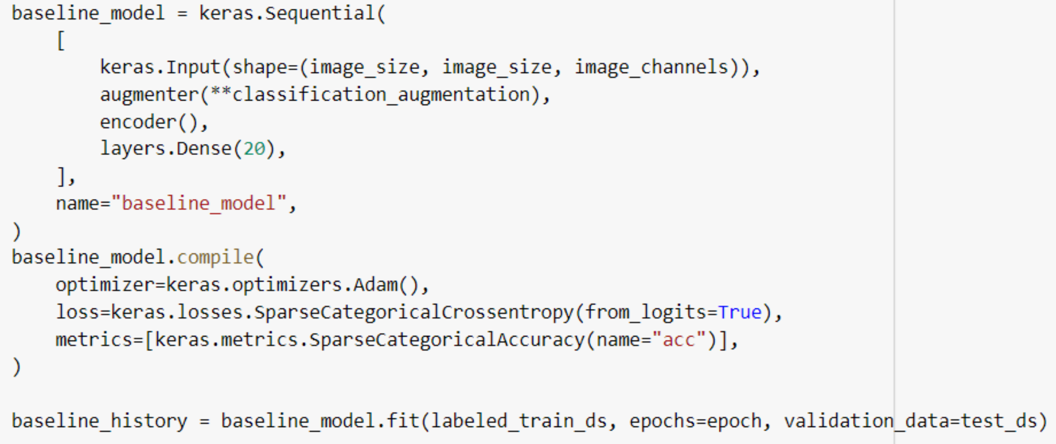
```
return keras.Sequential(
    [
        keras.Input(shape=(image_size, image_size, image_channels)),
        layers.Conv2D(width, kernel_size=3, strides=2, activation="relu", padding='same'),
        layers.MaxPooling2D(pool_size=(2, 2), strides=2, padding="valid", data_format=None,),
        layers.Conv2D(width, kernel_size=3, strides=2, activation="relu", padding='same'),
        layers.MaxPooling2D(pool_size=(2, 2), strides=2, padding="valid", data_format=None,),
        layers.Conv2D(width, kernel_size=3, strides=2, activation="relu", padding='same'),
        layers.MaxPooling2D(pool_size=(2, 2), strides=2, padding="valid", data_format=None,),
        layers.Conv2D(width, kernel_size=3, strides=2, activation="relu", padding='same'),

        layers.Flatten(),
        layers.Dense(width, activation="relu"),
    ],
    name="encoder",
```

The hyperparameter **width** is preset to take the value **128**. The **kernel_size** is the dimension of the filter of that particular convolutional layer; **strides** control the length of traversal in both horizontal and vertical directions. Padding is provided to the convolutional layers to ensure that the dimensions are preserved after each such layer.

Our baseline model consists of the augmentor and the encoder functions. Two metrics have been used to gauge the performance of the model during supervised training : **loss** and **accuracy**. The optimization goal is to maximize accuracy and minimize loss for both training and test datasets.

The optimizer used for this purpose is **Adam**, which is a slight variation of the stochastic gradient descent algorithm based on the adaptive learning of the first order and second order moments. Since our dataset has multiple classes with integer labels, I have used the **SparseCategoricalCrossentropy** loss function offered by keras. The model is trained for **40** epochs.



```
baseline_model = keras.Sequential(
    [
        keras.Input(shape=(image_size, image_size, image_channels)),
        augmenter(**classification_augmentation),
        encoder(),
        layers.Dense(20),
    ],
    name="baseline_model",
)
baseline_model.compile(
    optimizer=keras.optimizers.Adam(),
    loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=[keras.metrics.SparseCategoricalAccuracy(name="acc")],
)

baseline_history = baseline_model.fit(labeled_train_ds, epochs=epoch, validation_data=test_ds)
```

The evaluation metrics obtained after training for 40 epochs are as follows:

MAXIMUM VALIDATION ACCURACY: **58.63%**

MINIMUM VALIDATION LOSS : **1.15**

NOTABLE OBSERVATIONS:

- After 40 epochs, loss reduces from 2.30 to 1.15.
- There is not a great difference between the training and validation accuracies.

SELF SUPERVISED LEARNING USING SimCLR

After supervised training, an attempt is made to implement a self-supervised training based on the same encoder architecture with the help of the SimCLR framework. High quality labeled datasets remain one of the biggest obstacles for the mainstream adoption of deep learning solutions. While we are seeing unprecedented upgrades in deep learning research and technology, many of those methods can't be widely adopted due to limitations in the creation of datasets. This obstacle has propelled research in alternative methods such as semi-supervised and self-supervised learning which are able to operate by pretraining with unlabeled datasets. To address this challenge, Google recently unveiled SimCLR, a framework for working with self-supervised and semi-supervised models for image analysis.

Contrastive Learning

One of the features of a human's vision cognition is to rapidly determine similarities and differences in sets of images. When applied to deep learning models, this ability to detect differences is what we know as contrastive learning. Conceptually, contrastive learning can be defined as the ability to learn representations by enforcing similar elements to be equal and dissimilar elements to be different.

The key thought behind contrastive learning is to expand learning beyond high level features. If we are trying to determine differences in the tail of two images of whales, sometimes high level features such as color and size are not sufficient because they are going to be too similar in the different pictures. Contrastive learning, combines those high level features with local and global features that can give a context to the analysis in order to determine similarities and differences between images.

Two questions to be answered with regard to semi-supervised and self-supervised learning approaches are:

- *When would we expect semi-supervised learning approaches to fail? What can we do to avoid this?*
- *Apart from achieving a high test set accuracy, what other metrics are important while comparing and contrasting different learning approaches?*

Failure scenarios of semi-supervised learning approaches

There might arise scenarios where the approaches of semi-supervised learning fail to generate desirable results. To understand what could be these scenarios, it is important to understand the underlying working of these approaches first. In semi-supervised approaches, the learning algorithm is trained on a combination of labeled and unlabeled data. The amount of labeled data available is usually very small compared to the amount of unlabelled data. A typical workflow to implement this type of task would be to cluster the similar data points using an unsupervised learning algorithm and then use the available labeled data to label the rest of the unlabeled data based on the clustering. For this entire process to work out successfully, some **assumptions** are made which are:

- **Continuity of Datapoints:** Points lying close to each other have an increasingly high likelihood of having the same output label.
- **Creation of Discrete Clusters:** The unlabeled data points could be divided into a discrete number of clusters and points present in the same cluster are more likely to have the same output label.
- **Balanced Dataset of Labeled Data:** The part of the dataset that has labeled data should consist of a more or less even distribution of the different classes of objects present in the dataset.

The violation of either of these assumptions would lead to a failure of the semi-supervised approaches for image classification tasks. However if the assumptions are maintained, it is possible to obtain a classification model with an accuracy even higher than its supervised counterpart.

Metrics for comparing different learning methods

Among supervised learning approaches, **accuracy** is generally the most common evaluation metric that is used to gauge the performance of a classification model. Besides accuracy, **precision**, **recall** and **f1-scores** are other evaluation metrics which give us further insight into the performance of the model.

PRECISION = **#TRUE POSITIVES** / **#PREDICTED POSITIVES**

RECALL = **#TRUE POSITIVES** / **#POSITIVES**

F1-SCORE IS LIKE AN AVERAGE MEASURE OF PRECISION AND RECALL

These evaluation metrics are relevant only to supervised learning. In approaches involving unlabeled data, these metrics cannot be measured as there would not be any collection of true labels to set a benchmark for comparison. For unsupervised learning algorithms that implement clustering, **silhouette coefficient** and **Dunn's index** are principal evaluation metrics.

The silhouette coefficient is defined for each sample and is given by:

s = (**b**-**a**) / *max*(**a**,**b**) where

a = mean distance between a sample and other points in the same cluster.

b = mean distance between a sample and all other points in the next nearest cluster.

Dunn's index is equal to the minimum inter-cluster distance divided by the maximum cluster size.

For image segmentation procedures, another evaluation metric that is very commonly used is **IoU** or **intersection over union.** IoU is the area of overlap between the predicted segmentation and the ground truth divided by the area of union between the predicted segmentation and the ground truth.

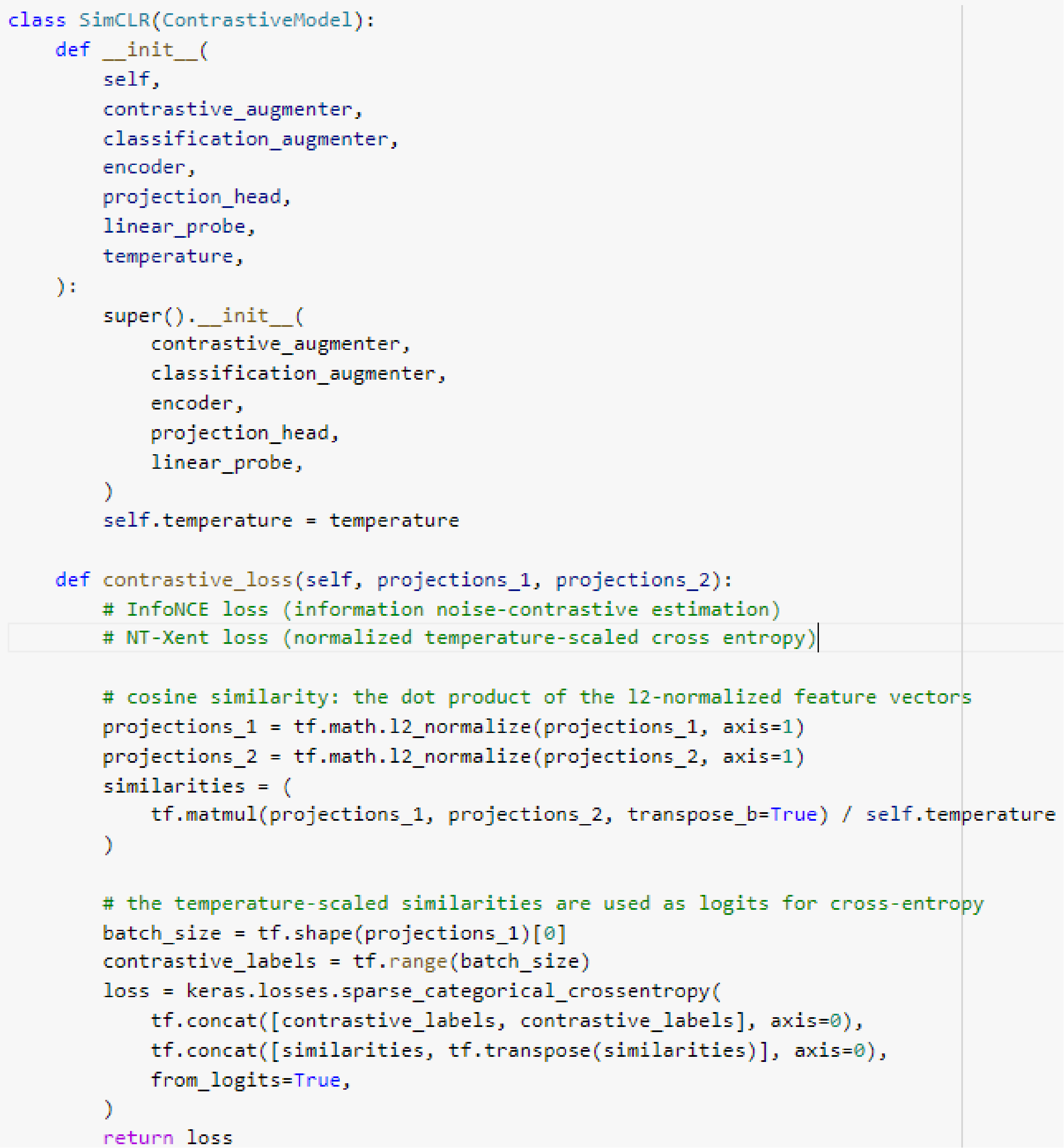
In our case of contrastive learning, however the metrics used are:

Contrastive Loss: Takes the output of the network for a positive example and calculates its distance to an example of the same class and contrasts that with the distance to negative examples. The loss is low if positive samples are encoded to similar representations and negative examples are encoded to different representations.

Probe Accuracy: Accuracy of the linear classifier fitted on top of the learned representations.

Results of training with SimCLR

The self-supervised training was done using the SimCLR framework.



```
class SimCLR(ContrastiveModel):
    def __init__(
        self,
        contrastive_augmenter,
        classification_augmenter,
        encoder,
        projection_head,
        linear_probe,
        temperature,
    ):
        super().__init__(
            contrastive_augmenter,
            classification_augmenter,
            encoder,
            projection_head,
            linear_probe,
        )
        self.temperature = temperature

    def contrastive_loss(self, projections_1, projections_2):
        # InfoNCE loss (information noise-contrastive estimation)
        # NT-Xent loss (normalized temperature-scaled cross entropy)

        # cosine similarity: the dot product of the l2-normalized feature vectors
        projections_1 = tf.math.l2_normalize(projections_1, axis=1)
        projections_2 = tf.math.l2_normalize(projections_2, axis=1)
        similarities = (
            tf.matmul(projections_1, projections_2, transpose_b=True) / self.temperature
        )

        # the temperature-scaled similarities are used as logits for cross-entropy
        batch_size = tf.shape(projections_1)[0]
        contrastive_labels = tf.range(batch_size)
        loss = keras.losses.sparse_categorical_crossentropy(
            tf.concat([contrastive_labels, contrastive_labels], axis=0),
            tf.concat([similarities, tf.transpose(similarities)], axis=0),
            from_logits=True,
        )
        return loss
```

SimCLR was used to perform self-supervised image classification using the same encoder architecture as that of the baseline model. The same image augmentation steps were followed, the only difference being that this time the augmentations were much stronger than the ones used for supervised training. Results show that the linear classifier **was not able** to capture the learned representations effectively.

CONTRASTIVE ACCURACY:	**99.76%**
TRAINING PROBE ACCURACY:	**35.34%**
VALIDATION PROBE ACCURACY:	**33.65%**

Ways to bring about Improvement

There can be two reasons due to which I could not obtain good results using SimCLR:

- The image representations obtained might not be good enough. It might be the case that training data requires better augmentation steps to be implemented on it.
- The linear probe might be unsatisfactory. I have used a very basic linear probe to fit on the learned image representations. A better way could be to use a properly tuned logistic regression classifier.

REFERENCES

- STL-10 dataset. https://ai.stanford.edu/~acoates/stl10/

- A modern data augmentation library – Resource KT. https://resourcekt.co.uk/a-modern-data-augmentation-library/

- Google Open Sources SimCLR, A Framework for Self https://www.kdnuggets.com/2020/04/google-open-sources-simclr-self-supervised-semi-supervised-image-training.html

- Metrics to Evaluate your Semantic Segmentation Model | by https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2

- Contrastive Loss Explained. Contrastive loss has been used https://towardsdatascience.com/contrastive-loss-explaned-159f2d4a87ec