Bring ideas to life
VIA University College

Dimensionality Reduction with Binary Constraint

Autoencoder and Binary Hashing

Project Report

Submitted by:

Constantina Tripon – 253085
Mark Vincze – 224478
Priyanka Shrestha – 299543

Supervisor:

Frederik Thorning Bjørn
Henrik Kronborg Pedersen

Number of characters : 35438

VIA University College

Bachelor of Engineering in Software Technology

Semester 7

1st June 2023

Contents

Abstract

In the realm of big data, the surge in volume, velocity, and variety of information necessitates innovative processing strategies for insightful decision-making. This project proposes a pipeline leveraging autoencoders and binary hashing for dimensionality reduction, optimized storage, and supporting fast lookup functionality. The pipeline is designed for applicability in downstream tasks involving computer vision and can run on standard computers, thereby addressing the high computational resource requirements often associated with deep learning models.

The autoencoder model is a U-Net style architecture, characterized by an encoder, a latent space representation, and a decoder, is used to compress and decompress vector space. This model incorporates a binary hashing constraint as a convolutional layer substitute function in the encoder. The Barlow Twins' model is utilised to validate the efficacy of the autoencoder, and hyperparameter optimization is achieved using the Optuna software.

The pipeline's application is demonstrated on the WorldStrat dataset, comprising 3.928 satellite images. These images, segmented into smaller parts, are subjected to data augmentation techniques such as Gaussian Blur, and the salt and pepper filter to enhance the deep learning model's accuracy.

Finally, Docker is utilized to ensure the consistency of the running environment, enhancing the model's security, flexibility, portability, and scalability through containerization. This approach presents a cost-effective, scalable, and efficient strategy for handling big data, demonstrating potential applicability in various domains requiring large-scale data processing and retrieval.

1 Introduction

Big data is viewed as the amount of data, both structured and unstructured, that is collected, on a day-to-day basis, by public and private organisations, which can contain useful domain specific information. According to Gartner, big data is '*high-volume, -velocity, and –variety information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision making*.' (Sicular, 2013)

With the high technological development in AI algorithms over the recent years, deep learning can be applied to big data analysis and learning tasks for the ability of processing massive amount of information. Many researchers have proven that for a better accuracy of the deep learning model, data augmentation techniques to generate extra data for the training model are necessary. The major downside of deep learning in big data analysis is the large storage capacity an organisation needs to acquire, such that the cost-effective condition of big data is not satisfied. Optimizing storage requirements for deep learning models require not only hardware but also software techniques which provide a scale down design of the learning tasks. (Maryam M Najafabadi, 2015)

Based on the background description, this project report demonstrates a pipeline for dimensionality reduction with binary constraint, meant to reduce storage requirements but also to provide an option for fast lookups to any semantic search functionality used in downstream tasks with computer vision.

An autoencoder model is used to reduce vector space, the binary hashing constraint consisting of a substitute function built as a convolutional layer in the encoder, and the Barlow Twins' model will demonstrate the effectiveness of the autoencoder.

Optimization software is used to find the best hyperparameters which proved to be successful in improving the result of the model.

Containerization is applied to ensure the consistency of the running environment, the system's isolation which enhances security forcing the application to run as an isolated process. Through containerization, the model is easy to be packaged along with its dependencies and deployed to required environments.

A known reality concerning the development and training of deep learning models is that they require high computational resources, especially above the average GPU. This model for dimensionality reduction was built such that it can ran on a standard computer.

In the following sections, the overall process is described, with the final solution being presented in the Result section at the end of this report.

2 Analysis

For dimensionality reduction, autoencoders are commonly used, because of the different capabilities they present. An autoencoder has the advantage of handling unsupervised learning, and because of the feature extraction advantage, which is stored in the latent space learner, can capture and compress the most important information from the original data.

Based on previous experiments, deep learning models are known to display better accuracy if augmentation techniques are used to manipulate the input data.

2.1 Dataset and augmentation techniques

The original dataset used to determine the dimensionality reduction technique contains 3.928 satellite images, each image with a size of 1054x1054 pixels and is called WorldStrat. Because deep learning models rely on the need to use large datasets for training the model, each satellite image has been split into 64 equal parts with a width and height of 128x128 pixels, this resulted with a large dataset of approximately 176.000 unique photos. The dataset contains unique locations such as forest-dense areas, water bodies, cities, snow covered areas, and deserts. Data cleansing did not take place on the dataset as the project is a real-life case, there had to be room for images that were imperfect. When it comes to satellite imagery, there can be occurrences of visual artifacts on the image due to atmospheric distortions or altitude. (Wright, 2020) (Orsolic, 2022)

The technique of augmenting an image refers to manipulation of the images in a dataset, to create similar or different content, mainly used as a pre-processing step. Most common used augmentation filter for image processing known to be applied to deep learning models is the Gaussian Blur, which is used to reduce the amount of noise an image has. For the WorldStrat dataset, multiple augmentation manipulations have been applied alongside Gaussian Blur. Both the Laplacian filter and the Sobel filter are edge detection manipulations of image augmentation. The salt and pepper filter, also known as impulse noise, presents itself as black and white pixels, spread across each image in the dataset.

The dataset used for the project can be found in Appendix 4.

Principal component analysis (PCA), also a form of dimensionality reduction has been considered in image pre-processing because of its positive results in experiments through the years.

2.2 Autoencoder

An autoencoder is a neural network known for learning efficient coding's of unlabeled data. The architecture of an autoencoder consists of three parts: the encoder, a function that transforms the input data, the latent space representation, which compresses the data points that are like each other and are closer together in space, and the decoder, another function that recreates the input representation. A distinct capability of autoencoders is that they provide the same output size as the original input. (Wikipedia, 2023) (Tiu, 2020)

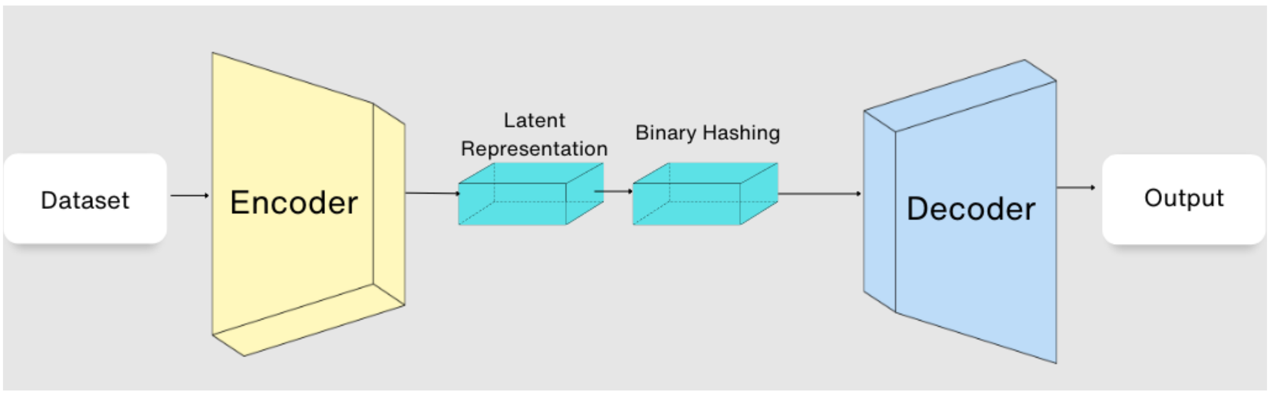


Fig.1 Autoencoder

2.2.1 Hyperparameter optimization

Hyperparameter tunning, in deep learning models, refers to finding the optimal hyperparameter metrics for optimal results. Optuna is an automatic hyperparameter optimization software framework, particularly designed for machine learning and deep learning. It features an imperative, *define-by-run* style user API, optimization is done based on an objective function, by making use of different samplers such as grid search, random, Bayesian, and evolutionary algorithms. (Optuna, 2023) (H, 2020)

2.2.2 Hashing

The problem of binary hashing, where given a high-dimensional vector $x \in R^D$, the matter is to map it to an L-bit vector $z = h(x) \in \{0,1\}^L$ using a hash function h , while preserving the neighbors of in the binary space. Binary hashing is an effective technique for fast search and retrieval of data for usage in different downstream tasks. The objective function of the hash function h or of the binary codes tries to capture some notion of neighborhood preservation. (Perpinan, 2015)

For this project a substitute function is used which proved to have a decisive impact on the output size of the encoder.

2.3 Barlow Twins

The Barlow Twins model is a self-supervised learning algorithm that learns embeddings invariant to distortions of the input sample. The Barlow Twins architecture is based on twin neural networks that share the same weight initialization. (Yann LeCunn, 2021)

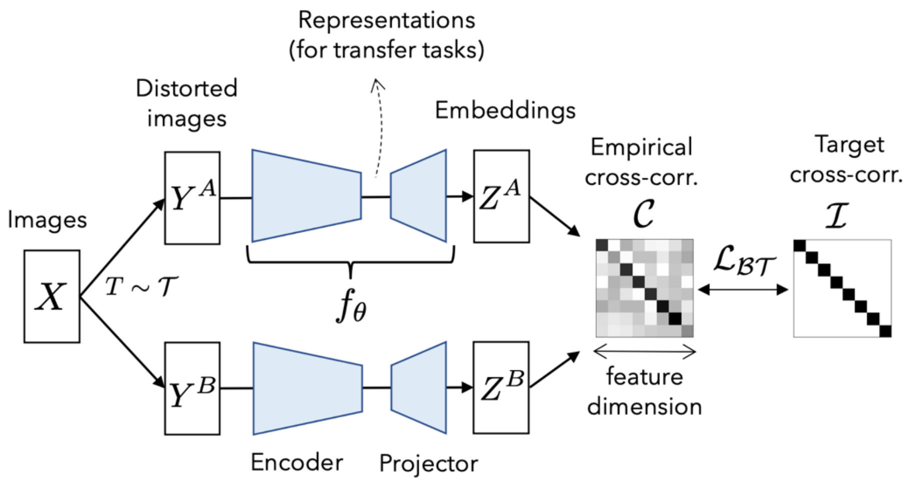


Fig.2 Barlow Twins Representation

The model is used as a test representation of the reconstructed output. One network will have as an input dataset the original image and the second one will have the predicted image from the auto encoder. The covariance between the two images, represented on the diagonal of the correlation matrix will act like a performance metric of the model.

2.4 Docker

Automating the deployment of the pipeline into an isolated environment will provide model flexibility, portability, and scalability. Docker is an open-source platform that allows developers to automate the deployment and management of applications using containerization. Constrainers are isolated environments that encapsulate the model and all its dependencies. Containers also support consistency and replication for running the model's environment, ensuring the recurrency of the output. (Docker, 2023)

3 Design

Under the design section a detailed representation of the proposed framework is presented. The models presented in the previous chapter are decompressed and stated individually, choice of methodologies argued.

3.1 Autoencoder

The decision behind choosing a U-Net style autoencoder for dimensionality reduction was based on the extensive research published with respect to this algorithm and the capabilities that have proven effective in designing this project successfully. Due to the ability of autoencoders to integrate unsupervised data and the effective practice in dimensionality reduction scored high into the decision-making process. The capability of autoencoders to compress meaningful information from the input data into a hidden layer and perform data reconstruction at the output layer proved to be correlated with the purpose of this project, making this another positive argument towards choosing autoencoders. (Ronneberger, 2015)

3.1.1 Encoder

The encoder ‘s input layer size is 128 both width and height and a mode of three (RGB) channels. The mode of the picture was set to three because of the capability of convolutional neural networks in learning separate representations of different color channels, edge detection or different visual pattern detection that the image might contain. The size of the image was set to 128x128 pixels with the intent that this model will not require a supercomputer for the algorithm to be trained on a smaller dataset as well and anyone with a standard CPU and GPU could use this model.

The API framework used for building the autoencoder is TensorFlow. TensorFlow is designed to be flexible, scalable, and efficient, making it suitable for a wide range of applications. (TensorFlow, 2023)

The architecture of the network can be seen in *Fig.3 Encoder* below. Besides the input layer, the encoder has five hidden layers and an additional layer between layer four and five. The addition layer performs an element wise addition between layers four and five improving the model’s ability to preserve important details and gradients, providing a better information flow. In this layer, the representation of the import features is passed to the lower layers of the model without a loss of representational power.

The same activation function is used in each layer, rectangular linear unit (ReLU) was the activation function that provided the best results after multiple experiments. The latent space representation provided an output shape of 64 by 64 with same number of channels.

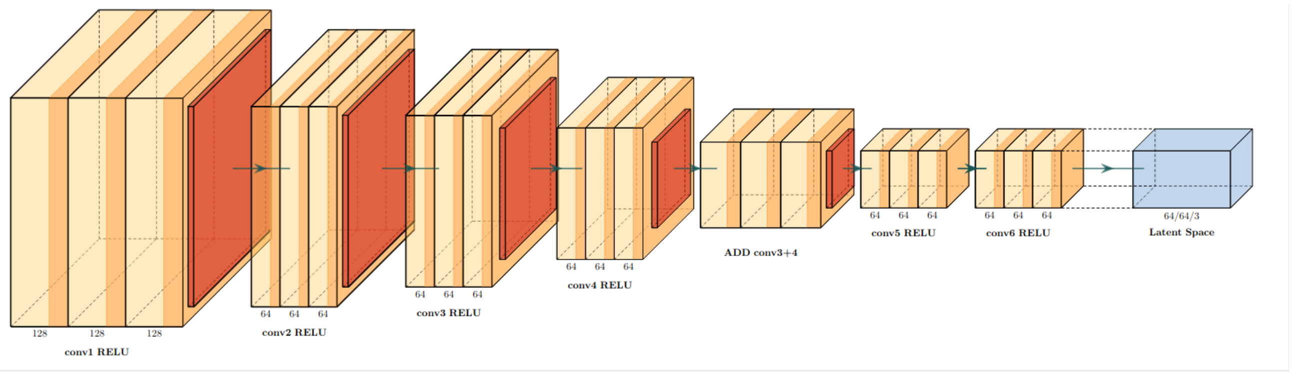


Fig.3 Encoder

3.1.2 Hashing layer

The hashing layer is built using the Conv2D class from Keras, an integrated library from TensorFlow. This layer is integrated in the encoder, the latent space representation being hashed such that it will further reduce the encoder's output. But the layer itself is not applying a hashing function on the output of the encoder.

3.1.3 Decoder

The same as the encoder, the decoder, represented in *Fig.4 Decoder* below, uses the same function, ReLU, for the activations and has skip connection of the layers three and four, to retain information that could otherwise be lost in the convolutions. The input size of the decoder is the same as the size of the latent space representation of the encoder and the output provided is the original size of the input image of the neural network.

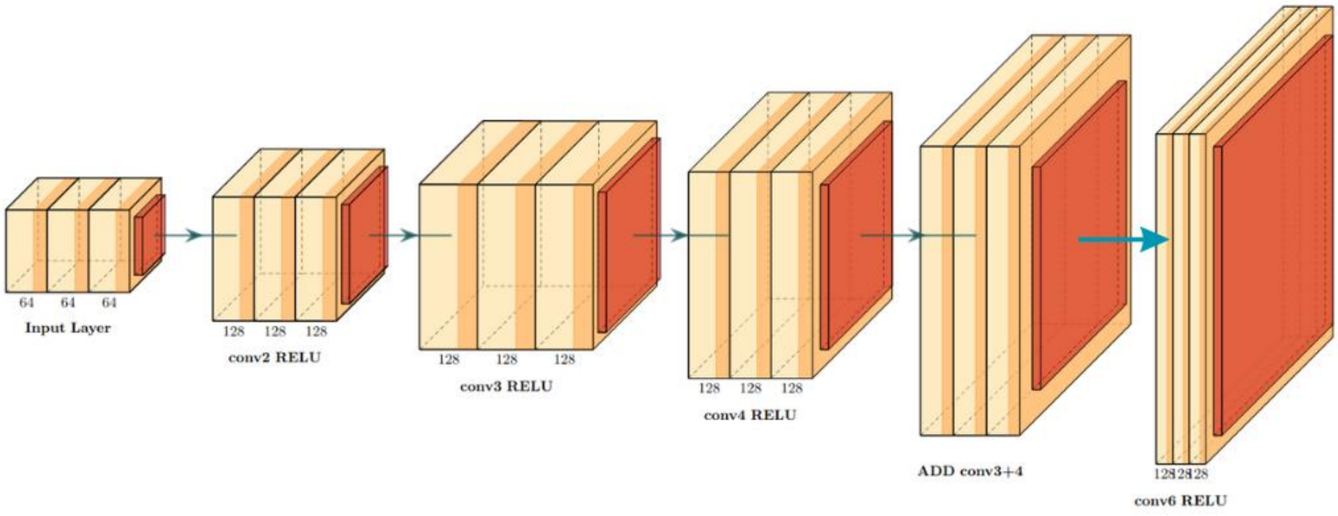


Fig.4 Decoder

3.2 Barlow Twins

As stated under the Analysis section, the Barlow Twins model represents a testing means of the dimensionality reduction using autoencoders. The model uses the same framework as the autoencoder, through the API's provided by TensorFlow and the integrated Keras library.

A detailed representation of the Barlow Twins model as it was implemented to fit the requirements of this project can be visualized in the diagram below. (Fig.5 Barlow Twins)

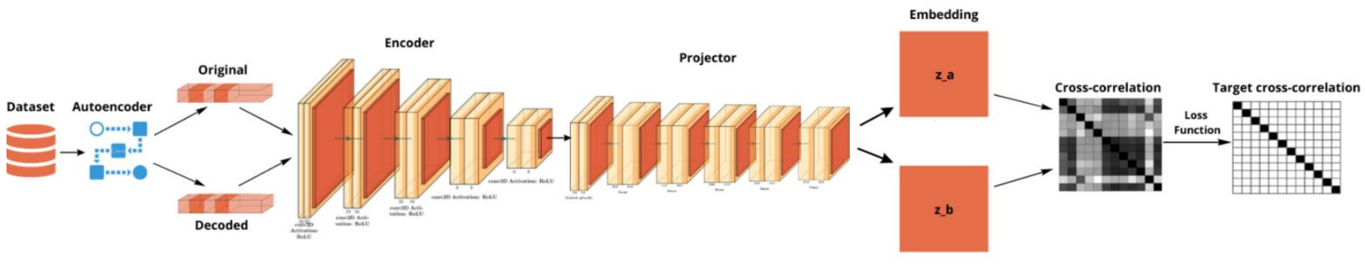


Fig.5 Barlow Twins (Appendix 5)

3.3 Docker

The containerisation of the compression functionality makes it compatible with every computer that has the sufficient resources and can run Docker or similar container management applications. This makes it possible also to swiftly use the compression and decompression without scripting knowledge.

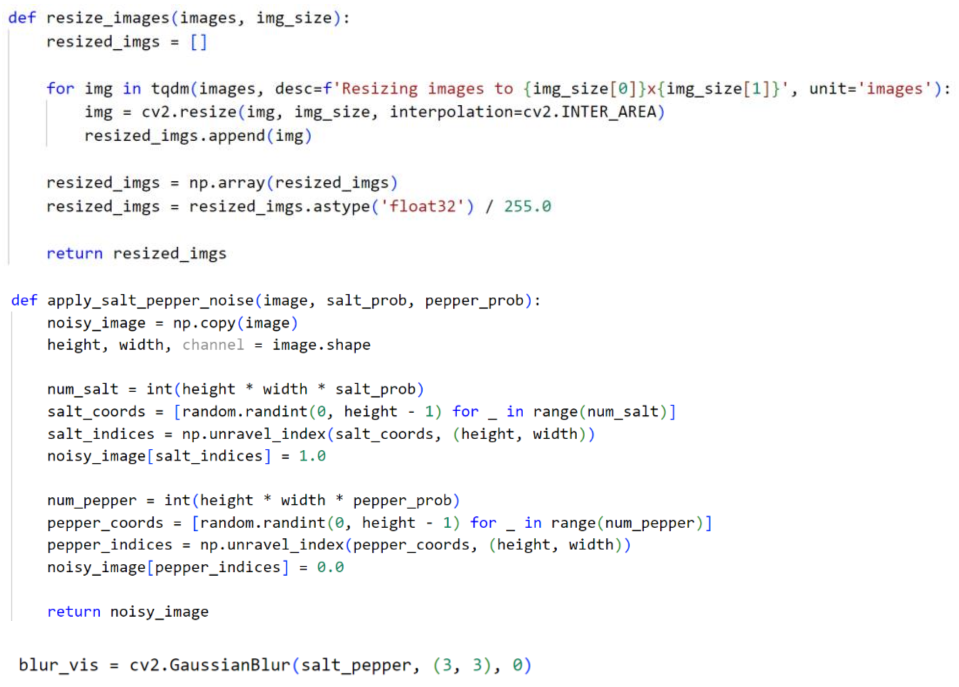
4 Implementation

Under this chapter implementation details will be highlighted describing the development flow of the dimensionality reduction pipeline. For link to the codebase see Appendix 1.

4.1 Pre-processing

The pre-processing applied on the dataset is chosen from the multiple experiments done during analysis and is applied on all the input data used.

In the `resize_image` function takes an array of images and image size to resize on as parameters. Since the model is built on a (128 * 128) size, all the images are resized to the same size. The image size is not customisable, so it is not recommended to use other values other than (128 * 128) unless the autoencoder's architecture is modified to take the specific input shape.



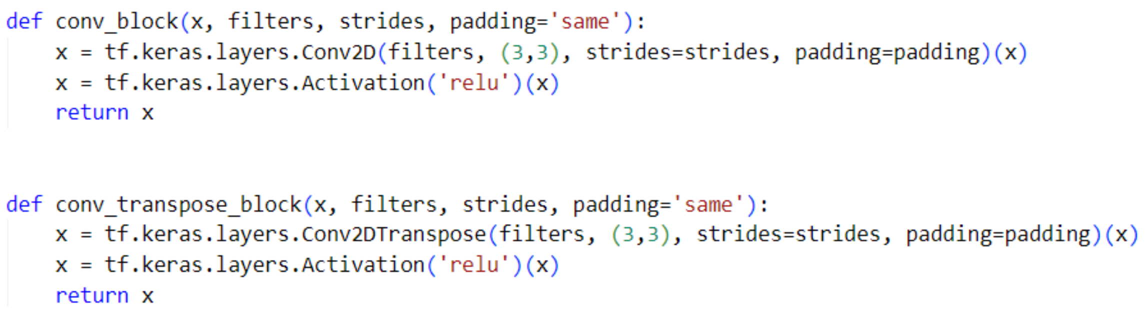
```
def resize_images(images, img_size):
    resized_imgs = []

    for img in tqdm(images, desc=f'Resizing images to {img_size[0]}x{img_size[1]}', unit='images'):
        img = cv2.resize(img, img_size, interpolation=cv2.INTER_AREA)
        resized_imgs.append(img)

    resized_imgs = np.array(resized_imgs)
    resized_imgs = resized_imgs.astype('float32') / 255.0

    return resized_imgs

def apply_salt_pepper_noise(image, salt_prob, pepper_prob):
    noisy_image = np.copy(image)
    height, width, channel = image.shape

    num_salt = int(height * width * salt_prob)
    salt_coords = [random.randint(0, height - 1) for _ in range(num_salt)]
    salt_indices = np.unravel_index(salt_coords, (height, width))
    noisy_image[salt_indices] = 1.0

    num_pepper = int(height * width * pepper_prob)
    pepper_coords = [random.randint(0, height - 1) for _ in range(num_pepper)]
    pepper_indices = np.unravel_index(pepper_coords, (height, width))
    noisy_image[pepper_indices] = 0.0

    return noisy_image

blur_vis = cv2.GaussianBlur(salt_pepper, (3, 3), 0)
```

Fig.6 Code Snip

4.2 Autoencoder

To eliminate code redundancy, individual functions create the layers of the autoencoder. The `conv_block` function is composed of a Conv2D layer with a ReLU activation function which is used in encoder model. The Conv2D layer is used for feature extraction. The layer learns the weights of the filter which is then applied to input data to extract relevant features.

The `conv_transpose_block` is another function that is composed of Conv2DTranspose layer with a ReLU activation function. This function is used for decoder model. The use of Conv2DTranspose layer is used for upscaling and reconstruction of the input data. The layer is typically used for generating high-resolution images from a lower-resolution representation.



```
def conv_block(x, filters, strides, padding='same'):
    x = tf.keras.layers.Conv2D(filters, (3,3), strides=strides, padding=padding)(x)
    x = tf.keras.layers.Activation('relu')(x)
    return x


def conv_transpose_block(x, filters, strides, padding='same'):
    x = tf.keras.layers.Conv2DTranspose(filters, (3,3), strides=strides, padding=padding)(x)
    x = tf.keras.layers.Activation('relu')(x)
    return x
```

Fig.7 Code Snip

4.2.1 Encoder

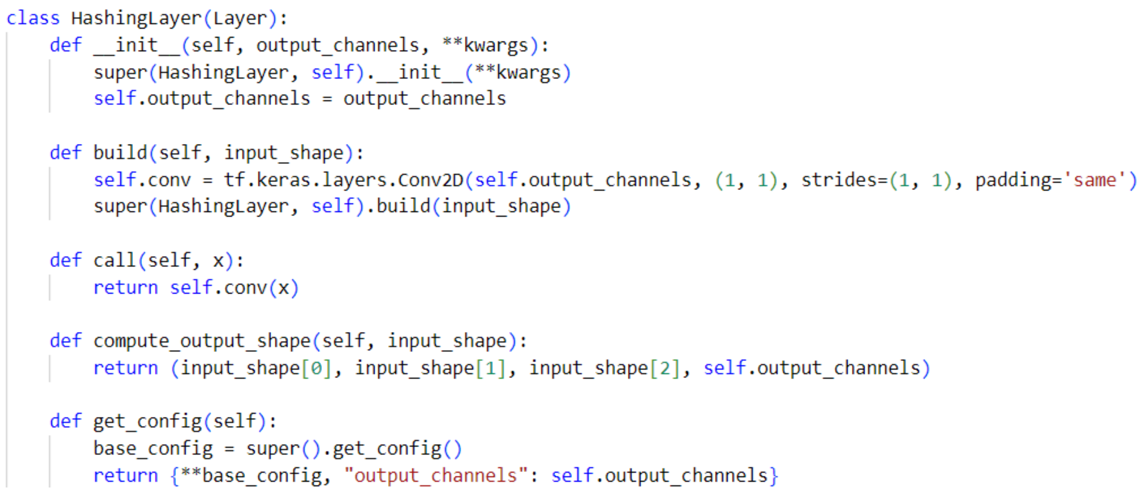
The `create_encoder` function constructs an encoder model using a series of `conv_block` function. The function takes an input of shape (128, 128, 3) and outputs a latent representation of size (64, 64, 2). A custom `HashingLayer` is applied to the last layer of the model ($a6$) which is a placeholder for a layer with binary hashing function applied to the output of the encoder.

```
def create_encoder(input_shape=(128, 128, 3)):
    input_img = Input(shape=input_shape)
    a1 = conv_block(input_img, 32, 1)
    a2 = conv_block(a1, 64, 2)
    a3 = conv_block(a2, 128, 1)
    a4 = conv_block(a3, 128, 1)
    skip_1 = Add()([a4, a3])
    a5 = conv_block(skip_1, 64, 1)
    a6 = conv_block(a5, 3, 1)
    a7 = HashingLayer(2)(Model(input_img, a6).output)

    return tf.keras.Model(input_img, a7)
```

Fig.8 Code Snip

4.2.1.1 Hashing layer

The class `HashingLayer` inherits the `Layer` class from the Keras library and is created to be used as the last convolutional layer of the encoder. The function `compute_output_shape` returns the shape of the output sensor based on the shape of the input sensor. The returned output shape will have the same special dimensions as the input sensor but with a different number of channels. The `get_config` method returns a dictionary containing the configuration of the layer for when the model has to be loaded, including the `output_channels` parameter. The `call` function has to be implemented from the parent class, this function is executing the hashing in the layer itself, in this case it is replaced with a Conv2D layer that takes the `output_channels` as arguments.

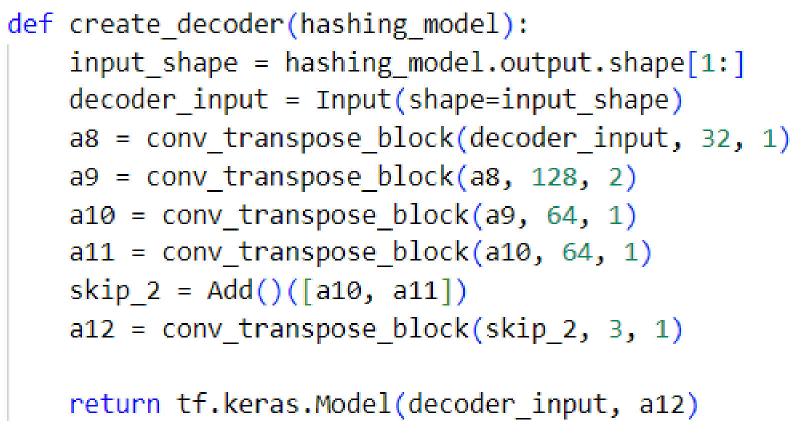


```
class HashingLayer(Layer):
    def __init__(self, output_channels, **kwargs):
        super(HashingLayer, self).__init__(**kwargs)
        self.output_channels = output_channels

    def build(self, input_shape):
        self.conv = tf.keras.layers.Conv2D(self.output_channels, (1, 1), strides=(1, 1), padding='same')
        super(HashingLayer, self).build(input_shape)

    def call(self, x):
        return self.conv(x)

    def compute_output_shape(self, input_shape):
        return (input_shape[0], input_shape[1], input_shape[2], self.output_channels)

    def get_config(self):
        base_config = super().get_config()
        return {**base_config, "output_channels": self.output_channels}
```

Fig.9 Code Snip

4.2.2 Decoder

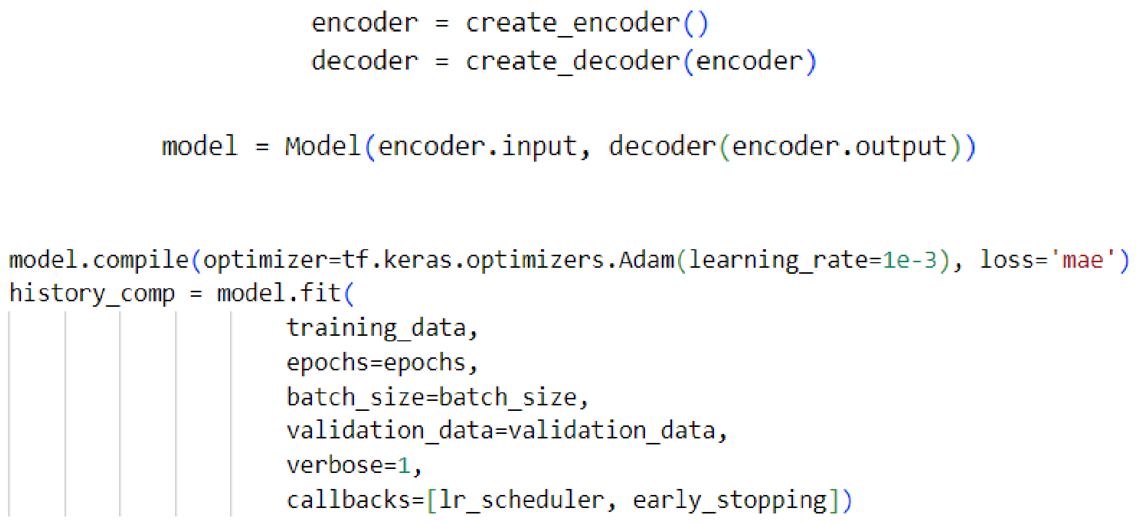
The `create_decoder` function represents the decoder model using a series of `conv_transpose_block` functions. The function takes the output of encoder as input and outputs a reconstructed image representation of size (128, 128, 3).



```
def create_decoder(hashing_model):
    input_shape = hashing_model.output.shape[1:]
    decoder_input = Input(shape=input_shape)
    a8 = conv_transpose_block(decoder_input, 32, 1)
    a9 = conv_transpose_block(a8, 128, 2)
    a10 = conv_transpose_block(a9, 64, 1)
    a11 = conv_transpose_block(a10, 64, 1)
    skip_2 = Add()([a10, a11])
    a12 = conv_transpose_block(skip_2, 3, 1)

    return tf.keras.Model(decoder_input, a12)
```

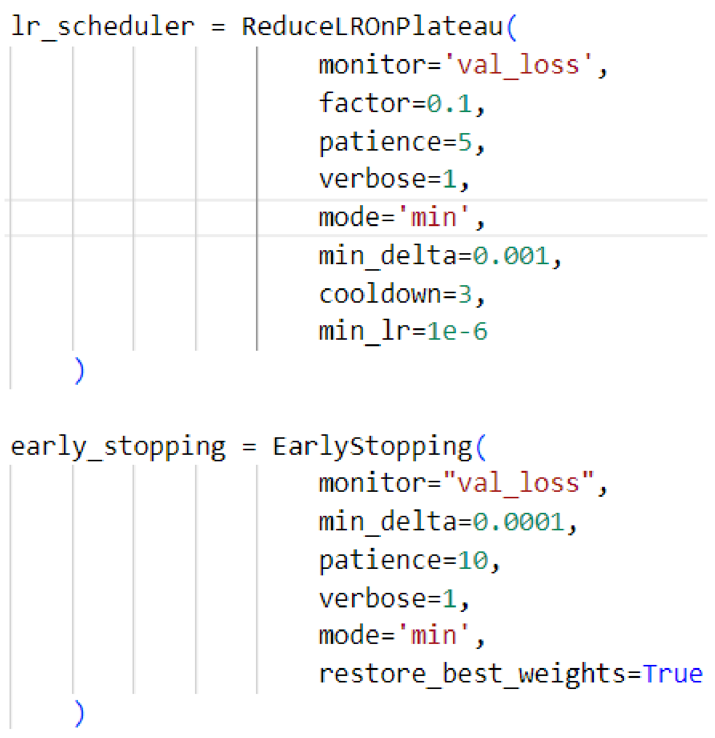
Fig.10 Code Snip

The model uses mean absolute error (`mae`) as loss function and Adam as optimizer, starting with a learning rate of 1e-3, that is a common value to use to initialise Adam. The learning rate of the model is crucial attribute as if it is too high the model can converge too fast or diverge completely, if the value is too low then the model can be stuck in a non-optimal local minimum or not converge quick enough. The `lr_scheduler` and `early_stopping` is used as callback functions to control the training process and reduce the chances of overfitting. (Keras, 2023)



```
encoder = create_encoder()
decoder = create_decoder(encoder)

model = Model(encoder.input, decoder(encoder.output))


model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3), loss='mae')
history_comp = model.fit(
                    training_data,
                    epochs=epochs,
                    batch_size=batch_size,
                    validation_data=validation_data,
                    verbose=1,
                    callbacks=[lr_scheduler, early_stopping])
```

Fig.11 Code Snip

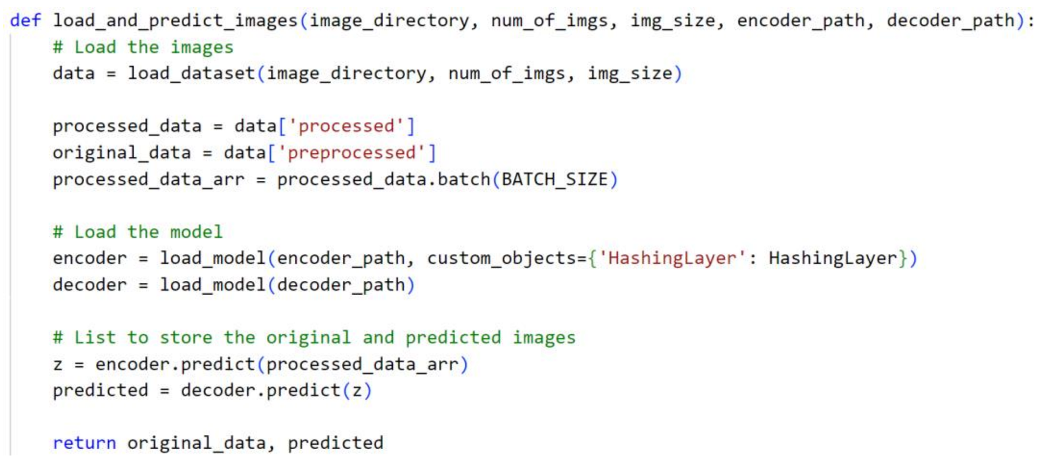


```
lr_scheduler = ReduceLROnPlateau(
                    monitor='val_loss',
                    factor=0.1,
                    patience=5,
                    verbose=1,
                    mode='min',
                    min_delta=0.001,
                    cooldown=3,
                    min_lr=1e-6
    )

early_stopping = EarlyStopping(
                    monitor="val_loss",
                    min_delta=0.0001,
                    patience=10,
                    verbose=1,
                    mode='min',
                    restore_best_weights=True
    )
```

Fig.12 Code Snip

4.3 Barlow Twins

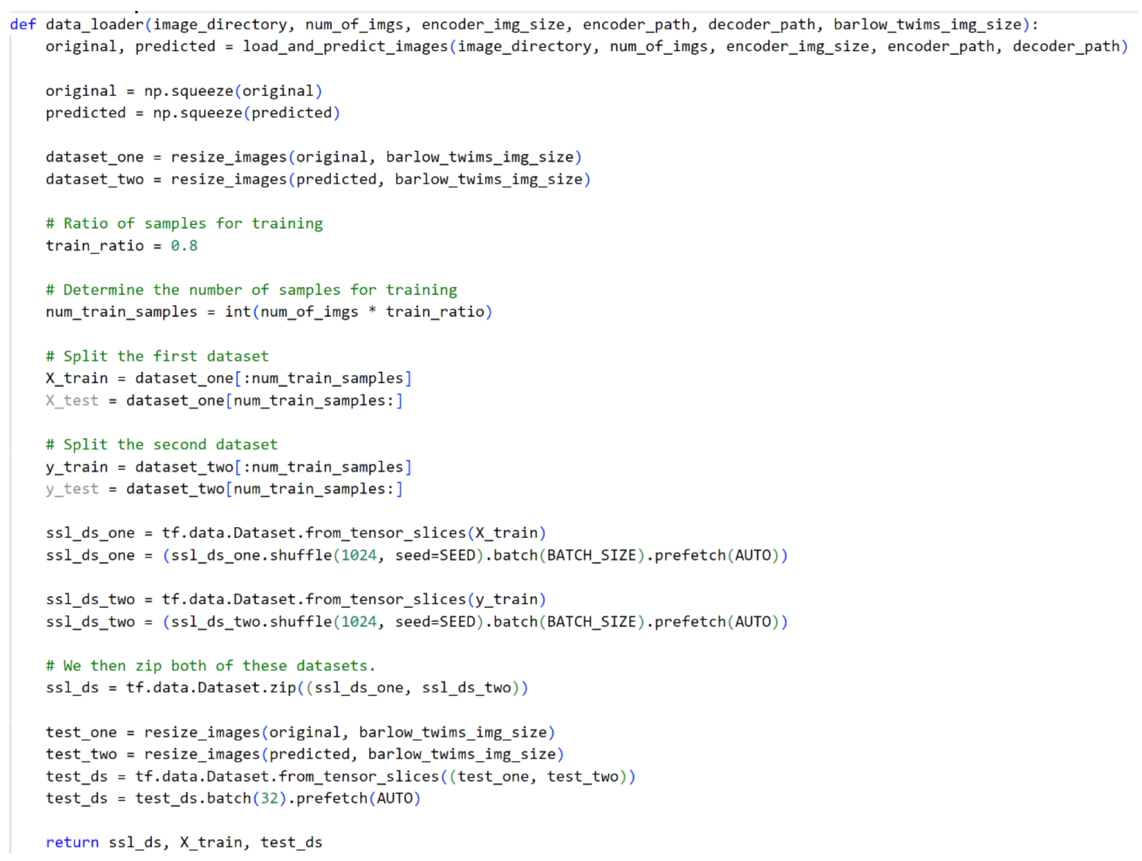
The `load_and_predict_images` function in load_for_bt.py file loads the data, applies pre-processing and uses the encoder and decoder models to retrieve the reconstructed image.



```
def load_and_predict_images(image_directory, num_of_imgs, img_size, encoder_path, decoder_path):
    # Load the images
    data = load_dataset(image_directory, num_of_imgs, img_size)

    processed_data = data['processed']
    original_data = data['preprocessed']
    processed_data_arr = processed_data.batch(BATCH_SIZE)

    # Load the model
    encoder = load_model(encoder_path, custom_objects={'HashingLayer': HashingLayer})
    decoder = load_model(decoder_path)

    # List to store the original and predicted images
    z = encoder.predict(processed_data_arr)
    predicted = decoder.predict(z)

    return original_data, predicted
```

Fig.13 Code Snip

The `data_loader` function takes the original and decoded images from the load_for_bt.py file. The original Barlow Twins model applies augmentation on the same image and builds the model on the augmented dataset. However, the model proposed in this report is used for testing objectives therefore the pre-processed and predicted image sets are used as two augmentations of the original image (dataset_one and dataset_two). The two datasets are then converted into `tensor_slices` (ssl_ds_one and ssl_ds_two) and zipped into one dataset (ssl_ds) to be used in the Barlow Twins model.

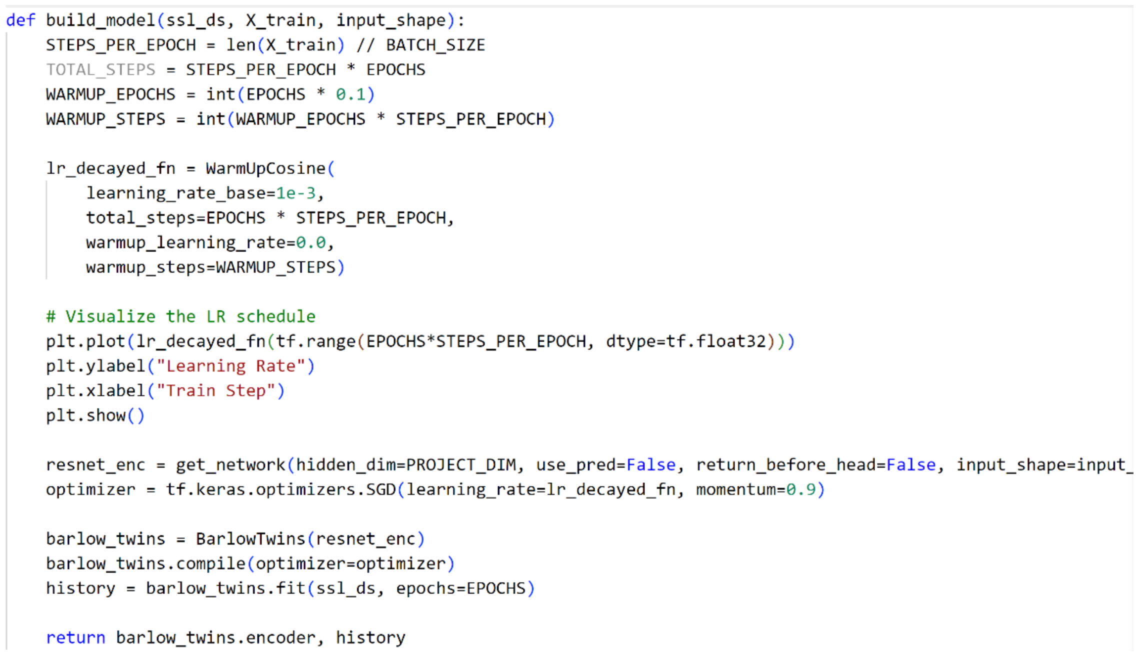


```
def data_loader(image_directory, num_of_imgs, encoder_img_size, encoder_path, decoder_path, barlow_twims_img_size):
    original, predicted = load_and_predict_images(image_directory, num_of_imgs, encoder_img_size, encoder_path, decoder_path)

    original = np.squeeze(original)
    predicted = np.squeeze(predicted)

    dataset_one = resize_images(original, barlow_twims_img_size)
    dataset_two = resize_images(predicted, barlow_twims_img_size)

    # Ratio of samples for training
    train_ratio = 0.8

    # Determine the number of samples for training
    num_train_samples = int(num_of_imgs * train_ratio)

    # Split the first dataset
    X_train = dataset_one[:num_train_samples]
    X_test = dataset_one[num_train_samples:]

    # Split the second dataset
    y_train = dataset_two[:num_train_samples]
    y_test = dataset_two[num_train_samples:]

    ssl_ds_one = tf.data.Dataset.from_tensor_slices(X_train)
    ssl_ds_one = (ssl_ds_one.shuffle(1024, seed=SEED).batch(BATCH_SIZE).prefetch(AUTO))

    ssl_ds_two = tf.data.Dataset.from_tensor_slices(y_train)
    ssl_ds_two = (ssl_ds_two.shuffle(1024, seed=SEED).batch(BATCH_SIZE).prefetch(AUTO))

    # We then zip both of these datasets.
    ssl_ds = tf.data.Dataset.zip((ssl_ds_one, ssl_ds_two))

    test_one = resize_images(original, barlow_twims_img_size)
    test_two = resize_images(predicted, barlow_twims_img_size)
    test_ds = tf.data.Dataset.from_tensor_slices((test_one, test_two))
    test_ds = test_ds.batch(32).prefetch(AUTO)

    return ssl_ds, X_train, test_ds
```

Fig.14 Code Snip

The `build_model` function is responsible for building and training the Barlow Twins model. The constants are configured at the beginning of model and are specific to the number of steps per epoch (100), the total steps for training and the warm-up steps.

The `lr_decayed_fn` function is used to configure the learning rate for the optimizer. The `WarmUpCosine` class combines the concepts of warm-up learning rate and cosine decay, that works by gradually increases the learning rate during the warm-up phase and then decays steadily using the cosine decay schedule. This class helps stabilize the training process, prevent large fluctuations of the learning rate and enables the model to converge to a better solution.



```
def build_model(ssl_ds, X_train, input_shape):
    STEPS_PER_EPOCH = len(X_train) // BATCH_SIZE
    TOTAL_STEPS = STEPS_PER_EPOCH * EPOCHS
    WARMUP_EPOCHS = int(EPOCHS * 0.1)
    WARMUP_STEPS = int(WARMUP_EPOCHS * STEPS_PER_EPOCH)

    lr_decayed_fn = WarmUpCosine(
        learning_rate_base=1e-3,
        total_steps=EPOCHS * STEPS_PER_EPOCH,
        warmup_learning_rate=0.0,
        warmup_steps=WARMUP_STEPS)

    # Visualize the LR schedule
    plt.plot(lr_decayed_fn(tf.range(EPOCHS*STEPS_PER_EPOCH, dtype=tf.float32)))
    plt.ylabel("Learning Rate")
    plt.xlabel("Train Step")
    plt.show()

    resnet_enc = get_network(hidden_dim=PROJECT_DIM, use_pred=False, return_before_head=False, input_shape=input_
    optimizer = tf.keras.optimizers.SGD(learning_rate=lr_decayed_fn, momentum=0.9)

    barlow_twins = BarlowTwins(resnet_enc)
    barlow_twins.compile(optimizer=optimizer)
    history = barlow_twins.fit(ssl_ds, epochs=EPOCHS)

    return barlow_twins.encoder, history
```

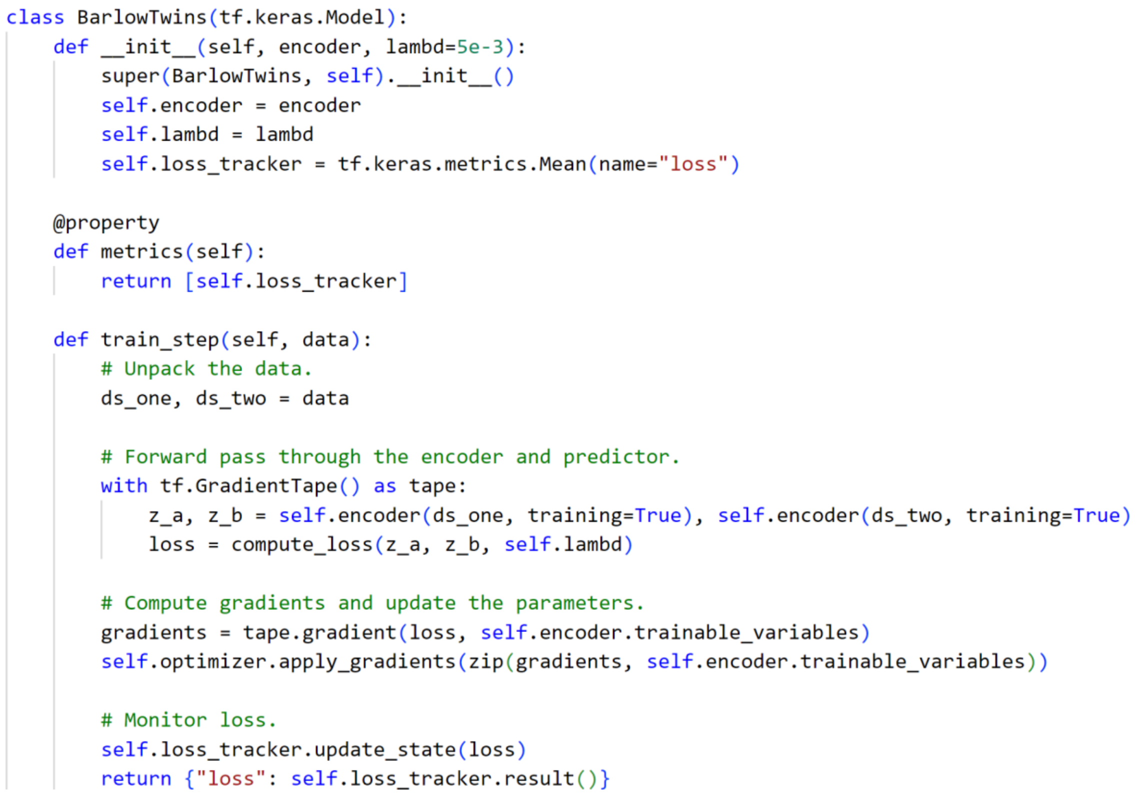
Fig.15 Code Snip

The `get_network` function is a prebuild ResNet-20 encoder model which is built on several conv2D and Dense layers. The model is built upon the input set of (32, 32, 3). Resnet20 is a customizable model, and it has proved to have a higher probability to perform better on larger input shape.

```
def get_network(n=2, hidden_dim=128, use_pred=False, return_before_head=True, input_shape=INPUT_SHAPE):
    depth = n * 9 + 2
    n_blocks = ((depth - 2) // 9) - 1

    # The input tensor
    inputs = Input(shape=input_shape)
    x = Rescaling(scale=1.0 / 127.5, offset=-1)(inputs)

    # The Stem Convolution Group
    x = stem(x)

    # The learner
    x = learner(x, n_blocks)

    # Projections
    trunk_output = GlobalAvgPool2D()(x)
    projection_outputs = projection_head(trunk_output, hidden_dim=hidden_dim)

    if return_before_head:
        model = Model(inputs, [trunk_output, projection_outputs])
    else:
        model = Model(inputs, projection_outputs)

    # Predictions
    if use_pred:
        prediction_outputs = prediction_head(projection_outputs)
        if return_before_head:
            model = Model(inputs, [projection_outputs, prediction_outputs])
        else:
            model = Model(inputs, prediction_outputs)

    return model
```

Fig.16 Code Snip

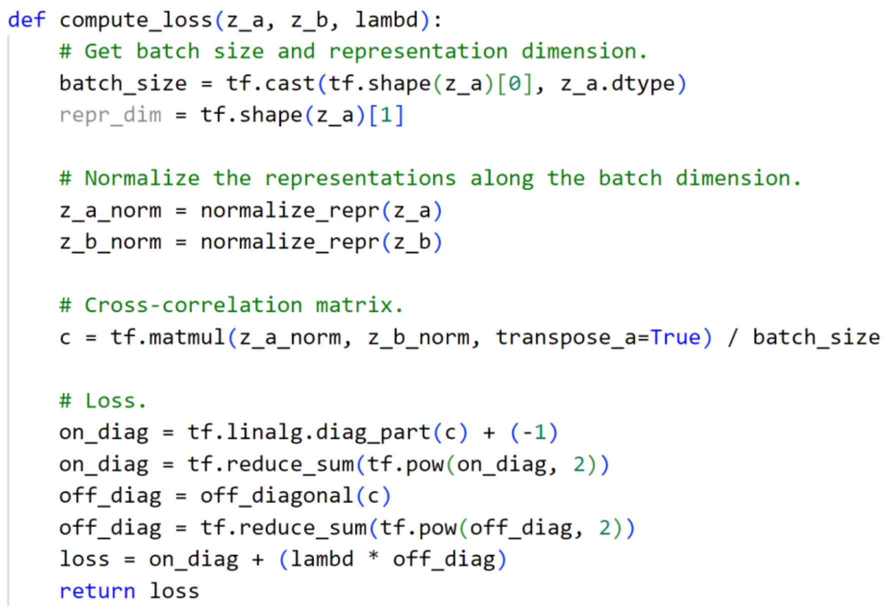
The Barlow Twins model takes the input data and unpacks it into two variables (ds_one and ds_two). The model then computes two encoded representations (z_a and z_b). The encoded representations are passed through a custom loss function, compute_loss and measures the gradients of the loss with respect to the trainable variable.



```python
class BarlowTwins(tf.keras.Model):
    def __init__(self, encoder, lambd=5e-3):
        super(BarlowTwins, self).__init__()
        self.encoder = encoder
        self.lambd = lambd
        self.loss_tracker = tf.keras.metrics.Mean(name="loss")

    @property
    def metrics(self):
        return [self.loss_tracker]

    def train_step(self, data):
        # Unpack the data.
        ds_one, ds_two = data

        # Forward pass through the encoder and predictor.
        with tf.GradientTape() as tape:
            z_a, z_b = self.encoder(ds_one, training=True), self.encoder(ds_two, training=True)
            loss = compute_loss(z_a, z_b, self.lambd)

        # Compute gradients and update the parameters.
        gradients = tape.gradient(loss, self.encoder.trainable_variables)
        self.optimizer.apply_gradients(zip(gradients, self.encoder.trainable_variables))

        # Monitor loss.
        self.loss_tracker.update_state(loss)
        return {"loss": self.loss_tracker.result()}
```

Fig.17 Code Snip

The `compute_loss` function is the original loss function computed in the Barlow Twins model. The loss is calculated based on the cross-correlation matrix; the final loss is determined as the sum of the squared elements of the diagonal with the scaled sum of the squared off-diagonal elements.



```python
def compute_loss(z_a, z_b, lambd):
    # Get batch size and representation dimension.
    batch_size = tf.cast(tf.shape(z_a)[0], z_a.dtype)
    repr_dim = tf.shape(z_a)[1]

    # Normalize the representations along the batch dimension.
    z_a_norm = normalize_repr(z_a)
    z_b_norm = normalize_repr(z_b)

    # Cross-correlation matrix.
    c = tf.matmul(z_a_norm, z_b_norm, transpose_a=True) / batch_size

    # Loss.
    on_diag = tf.linalg.diag_part(c) + (-1)
    on_diag = tf.reduce_sum(tf.pow(on_diag, 2))
    off_diag = off_diagonal(c)
    off_diag = tf.reduce_sum(tf.pow(off_diag, 2))
    loss = on_diag + (lambd * off_diag)
    return loss
```

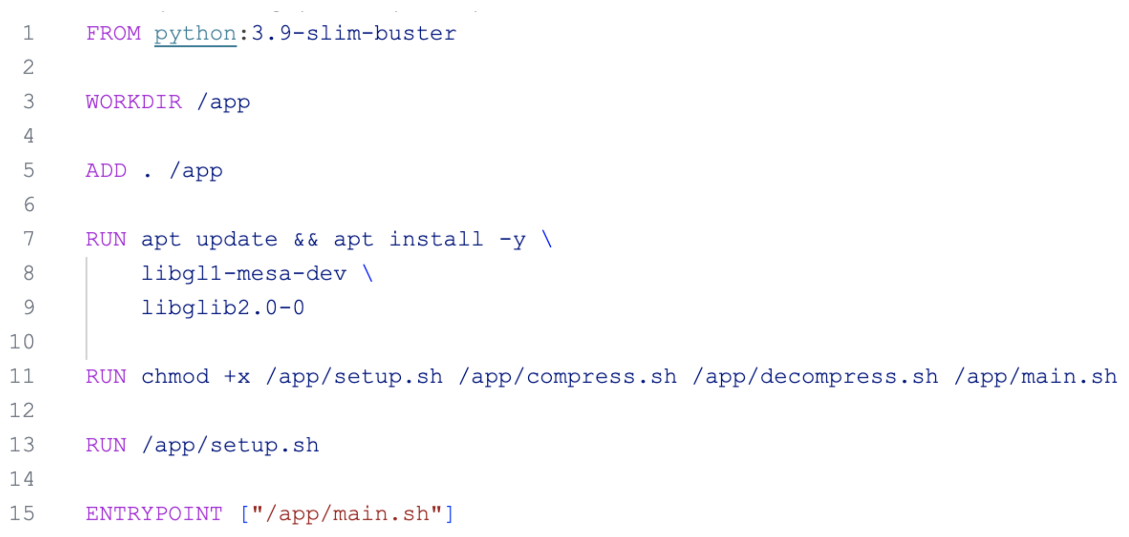
Fig.18 Code Snip

4.4 Docker

The encoder and decoder model are loaded with a Python script that takes care of the command line arguments, then it is executed with a shell script. The container is built after with the shell scripts inside the mounted folder.

The container's configuration and behaviour is controlled with a Dockerfile, it is built with a Linux image, installs Python as first steps, then sets the current directory as the workspace and mounts it to the container's filesystem.

After that the necessary Debian packages are installed, the shell scripts are made to be executable and runs the `setup.sh`, that will – according to the underlying operating system's needs - install the pip dependencies. The `main.sh` script is selected as starting point of the container, that script decides on what functionality (compress, decompress) should be triggered with the appropriate flags.



```
FROM python:3.9-slim-buster

WORKDIR /app

ADD . /app

RUN apt update && apt install -y \
    libgl1-mesa-dev \
    libglib2.0-0

RUN chmod +x /app/setup.sh /app/compress.sh /app/decompress.sh /app/main.sh

RUN /app/setup.sh

ENTRYPOINT ["/app/main.sh"]
```

Fig.19 Code Snip

5 Test

This chapter represents the overall result of the proposed framework. The tests include the loss function of the autoencoder, the structure similarity index measure which represents the percentage of how much the reconstructed image embodies the original and the correlation matrix of the Barlow Twins model that showcases the relationship between the variables.

5.1 Train-Validation loss graph

The train-validation loss graph shows the training and validation loss for each epoch during training phase. The figure below shows the train-validation loss graph obtained with the final autoencoder model. As seen in the graph the training and validation loss reached down from 0.025 to 0.005 when it reached to epoch 30 which indicates the performance and generalization ability of the model during the training phase.

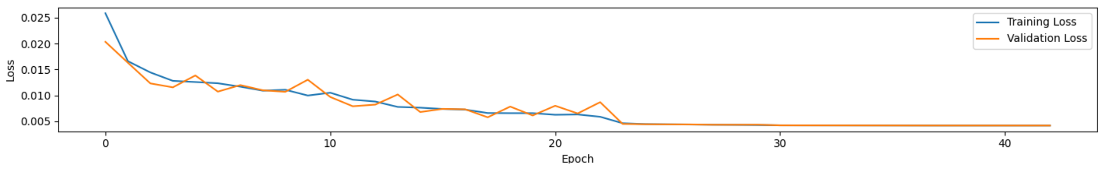


Fig.20 Autoencoder Loss

5.2 Visualization

Another method used to test the autoencoder model was to plot the original and predicted images and visually inspecting the similarities between the images. As seen in the figure, the decoded images capture almost all of the components of the original image which shows that the model was able to achieve a high-quality reconstruction of the original image.

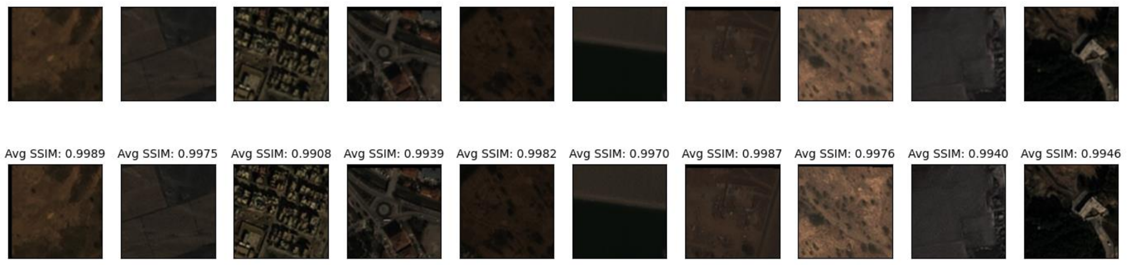


Fig.21 Structural Similarity Index Measure Visualisation

5.3 Barlow Twins

A correlation matrix was used as a measurement for testing the success of the Barlow Twins model. The cross-correlation matrix obtained from Barlow twins shows how similar the input images are. The matrix obtained by training with 25.000 images for 500 epochs can be seen in the figure below. The figure below shows a faint diagonal line which could be improved with the use of higher computational resources and a lengthy training process that can last up to a week.

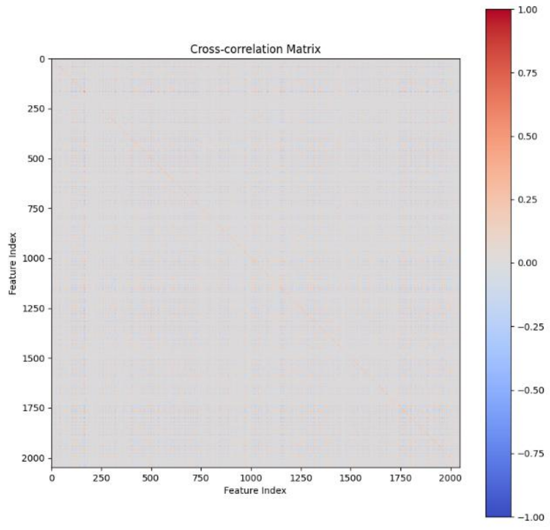


Fig.22 Barlow Twins Correlation Matrix

6 Experiments

The result of this framework was the consequence of multiple experiments. For a determined period, the focus of this project was highly directed towards finding the best methods that will guarantee the success of this framework.

Starting with the different augmentation and pre-processing methods applied, up to defining the best hyperparameters for the autoencoder model and the multiple optimization changes exploited for the Barlow Twins model, all have been documented and briefly introduced under this section, and the various results generated by the experiments have been catalogued.

6.1 ML-BPR Repository

The ML-BPR repository was created to have a sandbox and records of all the scripts that were used for the experiments executed. The file was named based on the versions in Jira. The experiments were carried out based on the research already published regarding similar projects and inputs from the supervisors. The repository can be found in Appendix 2.

6.2 Optuna

The framework called, Optuna was used for finding the best and most significant hyperparameters for the autoencoder model under training. With Optuna the model was trained with different combinations of hyperparameters with specified number of studies. All the models found by Optuna was migrated to the Optuna Dashboard to visualize the trials and the most important hyperparameters, to show which one has the most effect on the model's performance.

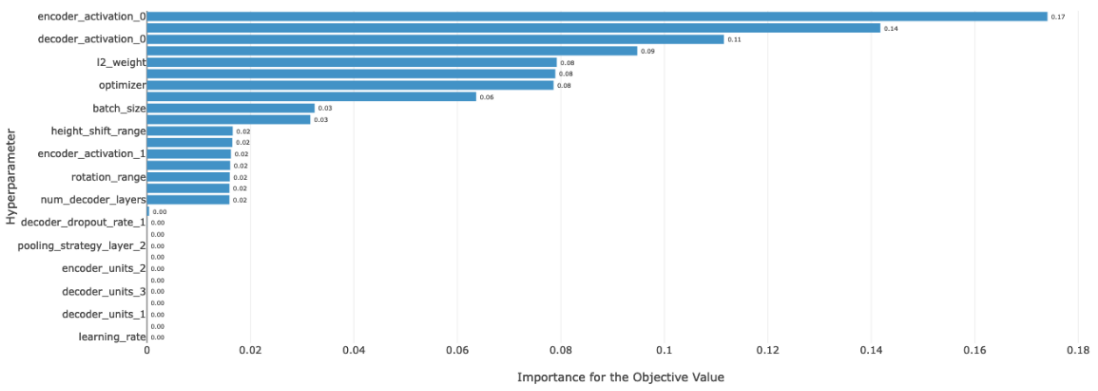


Fig.23 Importance of hyperparameters

The plot below shows one study's selections from the combinations of hyperparameters and their respective ranges. This way it is possible to follow what values were already trialled, and the framework can reset the parameters after each run by evaluating which combinations did better.

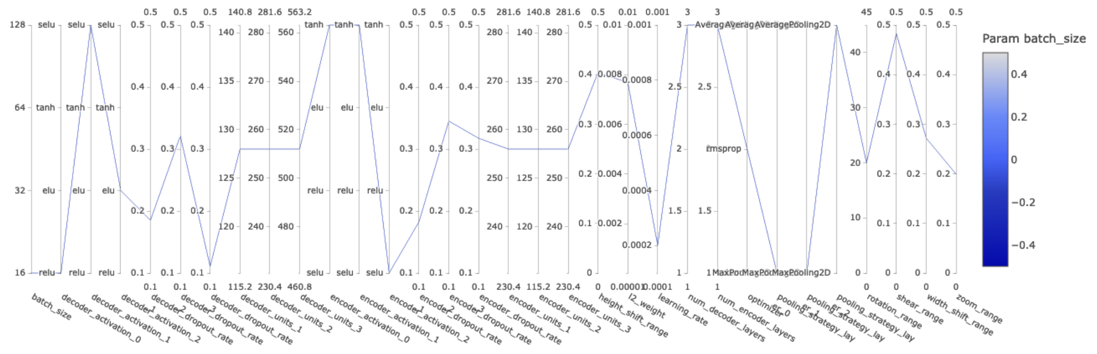


Fig.24 Hyperparameter trial

6.3 Results of Experiments

The results of the experiments were visualized and saved in a pdf file which can be found in Appendix 3.

7 Results and Discussion

This chapter summarise the overall results generated from the proposed framework. The purpose of this project explicitly presented the development of a pipeline for dimensionality reduction and imposing a binary constraint to further reduce the feature vector.

The autoencoder model gives a good result with the structural similarity of 95-99% and a good representation of the decoded image. The model was able to compress the original image of size (128, 128, 3) into an encoded representation of (64, 64, 2) which is a 75% reduction in size of the image. The rough calculation of the bytes used for storing original and encoded image is *196.608 bytes* and *32.768 bytes*.

The hashing layer reduces the dimension of the latent representation by one third while keeping the important information which is little compression compared to an actual hashing algorithm. In the traditional sense, a hashing function is applied on the output of the previous layer and a hashed output is returned, which still can significantly decrease the size of the encoder's output data.

For this project, the implementation of the Barlow Twins model came as a form of requirement, but simpler machine learning models could have provided a successful result. Algorithms like KNN would have been a better choice, the result from the KNN model allowing the validation whether the reduced dataset still contains enough information and evaluate the impact of the dimensionality reduction technique.

The Barlow Twins algorithm was published two years ago, in 2021 and the limited experimentation done based on the model does not provide much documentation. Used as a model for self-supervised learning, the model outperforms the other models Tin computer vision tasks. The original model is well suited for increased dimensionality and took seven days to train on a supercomputer. This framework was designed for any standard computer available making it difficult to have an acceptable result on a small dimensional dataset. (Yann LeCunn, 2021)

8 Conclusions

The framework proposed for creating the pipeline for reducing the dimension of the feature vectors proved to generalize well on the tests done. The experimentation done at different stages of the project development improved the result.

Augmentation is an essential component in deep learning models. After multiple unsuccessful augmentation methods applied to the dataset, the filtering technique which proved to increase the structure similarity index measure for the autoencoder model was selected.

The autoencoder model illustrates the dimensionality reduction methodology and the binary hashing constraint that needed to be a constituent part of this framework is represented by the convolutional layer named hashing, which is the latent space representation of the encoder. Because of the successful result of the autoencoder, the binary hashing constraint was eliminated as an implementation requirement and can be considered as a future improvement of this project.

Although the Barlow Twins algorithm is used as a testing model for the dimensionality reduction, its implementation is noteworthy because of the positive advantages it provided in regard to SSL models in computer vision. The original implementation its proven to be positive, however, for this project, the output provided was not sufficient. Details concerning improvements for the Barlow Twins model can be found in the next section.

As a general conclusion, the framework has achieved the initial purpose of creating a dimensionality reduction pipeline and imposing a binary constraint such that it will reduce the feature vectors, to be used later for downstream tasks.

9 Project future

During the experimental stage of the project various improvements have surfaced, however they could not be included in the scope or timeframe of this project.

The pre-processing applied to the dataset is based solely on data augmentation methods tested during the initial phase of the project period, however, techniques like outlier detection or dealing with data inconsistencies could create an increase in performance.

The current implementation of the autoencoder model is based on 25.000 images from the Kaggle dataset. With a higher data volume and a more variational dataset the model has the potential to perform better. The model's architecture could also be scaled up to handle different input image sizes.

The hashing layer is a Conv2D layer which reduces the latent space representation by 75 percent. As a future improvement, a discrete hash used to efficiently convert input data into a smaller representation which is a binary code. With the binary hashing algorithm implemented, the space of the latent representation could be further reduced, and the binary hashing will make the search of the image in larger database faster.

The Barlow Twins model is trained on 17.000 images of size (64 * 64). In the official research paper, most improves the authors propose relate to the exploration of higher dimensional embeddings but admit that it will require alternative hardware. It is a proven fact, that increasing the dimensionality of the representation also increases the accuracy in downstream tasks. As stated in previous chapters, the Barlow Twins model has been trained on a machine equipped with high GPU memory and ran for seven days. If the standard computer would be replaced by one with high GPU memory, the algorithm will perform better.

For storing the images' encoded formats, there could be a more sophisticated pipeline that implements a database connection, for uploading the compressed images as vector formats and for also retrieving the vectors, such as Weaviate. (Weaviate, 2023)

10 References

AI, M., 2021. *Meta AI.* [Online]
Available at: https://ai.facebook.com/blog/self-supervised-learning-the-dark-matter-of-intelligence/
[Accessed 6 January 2023].

Docker, 2023. *Docker.* [Online]
Available at: https://www.docker.com/
[Accessed 9 May 2023].

Do, T.-T., 2016. *Springer Link.* [Online]
Available at: https://link.springer.com/chapter/10.1007/978-3-319-46454-1_14
[Accessed 9 January 2023].

H, S., 2020. *Analytics Vidha.* [Online]
Available at: https://www.analyticsvidhya.com/blog/2020/11/hyperparameter-tuning-using-optuna/
[Accessed 7 April 2023].

Jha, A. H., 2023. *PyTorch Lightning.* [Online]
Available at:
https://lightning.ai/docs/pytorch/stable/notebooks/lightning_examples/barlow-twins.html
[Accessed 8 March 2023].

Keras, 2023. *Keras.* [Online]
Available at: https://keras.io/api/callbacks/
[Accessed 9 May 2023].

Keras, 2023. *Keras.* [Online]
Available at: https://keras.io/examples/vision/barlow_twins/#pseudocode-of-loss-and-model
[Accessed 23 May 2023].

Kubler, R., 2022. *Medium.* [Online]
Available at: https://towardsdatascience.com/outlier-detection-with-autoencoders-6c7ac3e2aa90
[Accessed 3 April 2023].

Kumar, G., 2014. *ResearchGate.* [Online]
Available at:
https://www.researchgate.net/publication/260952140_A_Detailed_Review_of_Feature_Extraction_in_Image_Processing_Systems
[Accessed 9 January 2023].
Maryam M Najafabadi, F. V. T. M. K. N. S. E. M., 2015. *Journal of Big Data.* [Online]
Available at: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-014-0007-7
[Accessed 5 March 2023].
Mena, F., 2019. *ResearchGate.* [Online]
Available at:
https://www.researchgate.net/publication/336823794_A_Binary_Variational_Autoencoder_for_Hashing
[Accessed 9 January 2023].
NISO, 2010. *Scientific and Technical Reports -,* Baltimore: National Information Standards Oganization.
Optuna, 2023. *Optuna.* [Online]
Available at: https://optuna.readthedocs.io/en/stable/index.html
[Accessed 2 February 2023].
Orsolic, J. &. I., 2022. *Kaggle.* [Online]
Available at: https://www.kaggle.com/datasets/jucor1/worldstrat
[Accessed 5 February 2023].
Perpinan, M. C., 2015. *arvix.* [Online]
Available at: https://arxiv.org/abs/1501.00756
[Accessed 1 February 2023].
Ronneberger, O., 2015. *arXiv.* [Online]
Available at: https://arxiv.org/abs/1505.04597
[Accessed 1 Fabruary 2023].
Sadr, A. V., 2019. *Journal of Big Data.* [Online]
Available at: https://link.springer.com/article/10.1007/s00521-021-05839-5
[Accessed 9 January 2023].

Sicular, S., 2013. *Forbes.* [Online]
Available at: https://www.forbes.com/sites/gartnergroup/2013/03/27/gartners-big-data-definition-consists-of-three-parts-not-to-be-confused-with-three-vs/?sh=13c685ed42f6
[Accessed 4 March 2023].
Sikorsky, I., 2020. *arXiv.* [Online]
Available at: https://arxiv.org/pdf/2004.14717.pdf
[Accessed 9 January 2023].
TensorFlow, 2023. *TensorFlow.* [Online]
Available at: https://www.tensorflow.org/
[Accessed 1 February 2023].
Thudumu, S., 2020. *Journal of Big Data.* [Online]
Available at: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-020-00320-x
[Accessed 7 January 2023].
Tiu, E., 2020. *Medium.* [Online]
Available at: https://towardsdatascience.com/understanding-latent-space-in-machine-learning-de5a7c687d8d
[Accessed 2 April 2023].
Vaaras, E., 2022. *arXiv.* [Online]
Available at: https://arxiv.org/pdf/2206.10188.pdf
[Accessed 9 January 2023].
VIA Engineering, in preparation. *Confidential Student Reports,* s.l.: s.n.
Weaviate, 2023. *Weaviate.* [Online]
Available at: https://weaviate.io/developers/weaviate
[Accessed 9 May 2023].
Wikipedia, 2023. *Wikipedia.* [Online]
Available at: https://en.wikipedia.org/wiki/Autoencoder
[Accessed 22 March 2023].
Wright, A., 2020. *Mapware.* [Online]
Available at: https://mapware.com/blog/understanding-errors-and-distortion-in-remote-

sensing/

[Accessed 5 March 2023].

Yann LeCunn, J. Z. L. J., 2021. *arXiv.* [Online]

Available at: https://arxiv.org/abs/2103.03230

[Accessed 1 Ferbruary 2023].

11 Appendices

Appendix 1: Source Code Base

https://github.com/Purple-Wizard/BPR

Appendix 2: Experimented Code Base

https://github.com/Purple-Wizard/ML-BPR-trials

Bring ideas to life
VIA University College

Dimensionality Reduction with Binary Constraint

Autoencoder and Binary Hashing
Process Report

Submitted by:

Constantina Tripon – 253085
Mark Vincze – 224478
Priyanka Shrestha – 299543

Supervisor:

Frederik Thorning Bjørn
Henrik Kronborg Pedersen

Number of characters: 20779
VIA University College
Bachelor of Engineering in Software Technology
Semester 7
1st June 2023

Table of content

1 Introduction

The group that stands behind this project and process reports is named Purple Wizard (PW) and is represented by Priyanka, Mark and Constantina. This projects topic was provided by Systematic Aps and developed in accordance with the company requirements.

PW's way of working adopted an agile methodology designed by means of the Kanban framework, the preliminary meetings being held remotely, fact that has changed during the project period.

This report will describe the stages of the development process, focusing transiently on cultural backgrounds and verbosely on the agile process, will also include the personal reflections of each member of PW in regard to the individual participation and outcome of the project.

The discussion, supervision and conclusion sections will complete the process report.

2 Group Description

The group has three members, coming from different nationalities, who have worked together on previous projects. All together have been working on SEP4, and every school project since, and two of them have been on the same team and school projects since SEP3.

The team has a diversified cultural background: Nepal, Hungary and Romania. To have a better understanding of why the team is well integrated and functioning successfully, an analysis into the Lewis Model of Cross Culture is conducted.

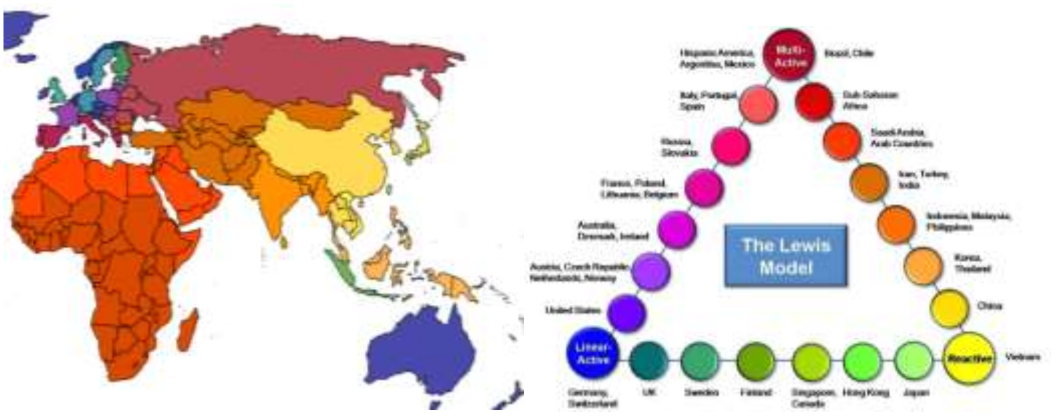


Figure 1. Lewis Model of Cross-Cultural Competence (Tangerine, 2023)

Based on the model, Hungary and Romania have common cultural values. Both countries are classified as being multi-active, which means that they are driven by emotions, and are presented as being impulsive, impatient individuals. However, both of the group members have resided in Denmark for quite a while now hence they are also characterized as being very linear active, which means they are task oriented, organized planners and analytical thinkers.

Whereas Nepal lies right in between a multi-active and reactive section which represents a balance between professional and social life. The model also characterizes Nepal as an emotional and polite group who is good at maintaining a social relationship and enjoy harmony in the society and are individuals who are people oriented.

Established by the Lewis Model and the previous experience of working together, the member from Hungary is responsible for maintaining the structure of the entire project and quality assurance, the member from Romania is responsible for meeting deadlines and critical thinker and the member from Nepal is responsible for the group stability and has the role of mediator.

3 Project Initiation

3.1 Project Idea and Group Contract

PW had the opportunity to work jointly with *Systematic ApS* for searching for a relevant idea to research and develop as a Bachelor Project. The company offered some project ideas which was suitable to fulfil the requirements and increase the knowledge PW wanted to accumulate while working on the Bachelor Project. Out of four options that *Systematic* provided, the team decided to pursue the project titled *Dimensionality Reduction with Binary Hashing*. The decision was made based on the individual preference and passion of each member to ensure a project selection that aligned with personal motivation.

Communication, meetings, and agreements which sustained the initial rules and regulation each member of PW agreed to respect during the development of this project, are stipulated in the Group Contract, available in Appendix 1.

3.2 Agile Methodology

As mentioned in the introduction, the Kanban framework provided the project management process needed for increased efficiency in project development. Configured under the Atlassian umbrella, Jira was set up, building the roadmap of the project using Epics.

Together with Kanban, epics allow a structured approach to bigger tasks, helping teams gain a better understanding of the task at hand.

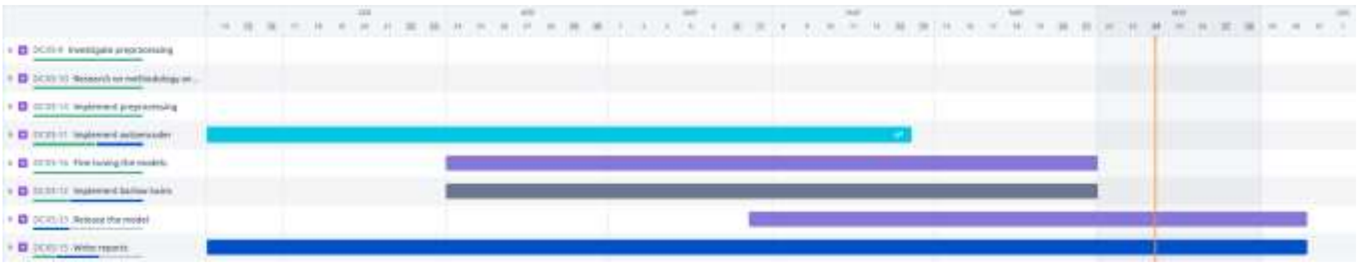


Fig.2 Kanban Roadmap

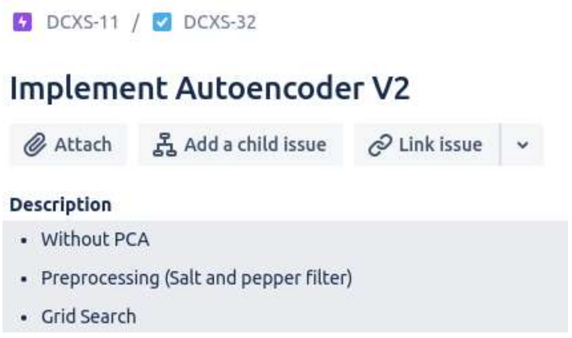
The Jira board structure includes six individual boards, including:

1. TO DO
2. IN PROGRESS
3. IN TEST
4. IN REVIEW
5. DONE
6. DROPPED

Fig.3 Kanban Board

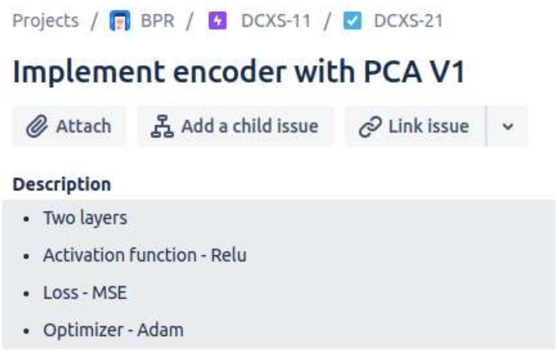
The first four cards in the board do not need further clarification. In terms of interpreting the definition of DONE understood by PW, the unilateral decision stopped at individual task presentation. When one of the group members starts working on a selected task, and that task passes forward through the TEST phase and once it reaches the REVIEW card, the task initiator presents one's work development in an unformal presentation. During this assessment session, under the REVIEW phase of the working process, each member gives out one's feedback. Once the whole group is satisfied with the end result, that specific task is then moved on to the DONE card.

During the first experiments and based on the supervisor feedback, the DROPPED card was added to represent the tasks which will be missing from development but decided to be kept for a summary of the complete development process.



Implement Autoencoder V2

Description
- Without PCA
- Preprocessing (Salt and pepper filter)
- Grid Search



Implement encoder with PCA V1

Description
- Two layers
- Activation function - Relu
- Loss - MSE
- Optimizer - Adam

Fig.3 Kanban Task Card

PW decided that the individual tasks must have a comprehensive name and short description whenever necessary. Although the methodology adopted for the development of this project is agile, the work process does not allow for iterative progress. Versioning replaced the iterative process, must be included in the task name, and is used after the feedback session during the REVIEW process. If throughout the review process, improvements are discussed, a new task is created with the same name but different version.

For more insight on the work process see Appendix 2 where snippets of Jira during the development process are provides.

4 Project Description

The project depicts a dimensionality reduction technique which is needed to be binarized with the intention of minimizing storage requirements, and at the same time being capable of vector restoration whenever needed in downstream tasks. For performance metrics, a deep learning model built from two Siamese networks and invariant to distortions, helped concretize the pipeline of the release model.

The main problem for the project was dimensionality reduction while preserving semantic relation of the original data and being restored. With the time limitation in mind, a delimitation was set to only handle images.

PW decided to use Scrum as a methodology for the project as the team had experience with it from previous semesters. However, during the Inception phase the team came to realization that the user stories required for the project could not be generated. The realization stemmed from the project's heavy focus on the machine learning rather than traditional software components. Which is why the team decided to shift towards using Kanban. Kanban is an agile project management framework that emphasizes optimizing workflow. Especially because of the nature of the project where experimentation and iteration play a crucial role, using Kanban has been fruitful.

Since the project shifted from Scrum to Kanban, the time schedule which was initially planned based on Sprints also moved towards roadmap which was created in Jira based on the Epics.

For more details see Appendix 3.

5 Project Execution

The communication of PW has been mostly through physical meetings and couple of times through Zoom. The formal Scrum meetings such as planning and retrospective meetings were dropped as soon as the methodology was shifted towards Kanban. Periodically, informal meetings were conducted within the team for creating tasks and planning the workflow. Working together has been advantageous as it was easier to communicate and review each other's tasks.

The shift to Kanban has helped the team to focus more on tasks and delivery rather than the time-bound iterations. The continuous delivery emphasis of Kanban aligned well with the need for continuous improvement and feedback loops.

The entire project consisted of iteration of the same tasks centred around modifying and testing the model. To address the iterative nature of the tasks, the team decided to version the tasks in Jira. The versioning allowed the team to keep track of the changes made to the model in each iteration. This helped the team to maintain the structured and organized workflow, facilitating effective collaboration and ensuring clear understanding of the project's evolution over time.

Midway through the project, the team introduced the "DROPPED" column in Jira board. The column was added to accommodate the tasks that were no longer deemed as relevant to pursue. As the team progressed the team encountered tasks that became obsolete due to the new insights in the project. The addition of the column allowed team to clearly separate these tasks from the active ones.

6 Personal Reflections

6.1 Constantina Tripon

The motivation behind selecting this project topic roused from my interest in learning more about machine learning and deep learning models. Due to the limited literature the program provided by the school offered, the chosen topic proved to be satisfying in terms of its scope and added knowledge.

While I was familiar with the subject of dimensionality reduction, the implementation of binary hashing was unknown, as well as models like Barlow Twins with redundancy reduction. To gain a comprehensive understanding, thorough research was needed, lots a published research read, plenty of trial and error in implementation to finally reach a standard level of understanding of the concepts. By dedicating time to study and comprehend these advanced techniques, I aimed to expand my knowledge and expertise in these specific areas.

Implementing the pipeline for this project presented significant challenges and required substantial research efforts. However, the process proved to be highly beneficial as it facilitated the acquisition of a wealth of knowledge. The comprehensive research done contributed to a deeper understanding of the deep learning models, the architecture, and the powerful application in real-world utilization.

Throughout the development phase, a valuable skill I acquired was working with the Kanban methodology. This approach to project management provided an effective framework for organizing tasks and workflow, ensuring efficient coordination and collaboration within the team.

Pair programming played a major role in enhancing the learning outcomes of the project. By collaborating closely, we were able to leverage each other's expertise and perspectives, leading to a deeper understanding of the subject. Through this collaborative approach, we engaged in active discussions, exchanged ideas, and challenged each other's assumptions. The process of working together in real-time allowed for immediate feedback and continuous learning. It not only improved our technical skills but also fostered effective communication, problem-solving, and critical thinking. Pair programming proved to be a valuable practice that enhanced the overall learning experience and contributed to the success of the project.

This project brought a significant outcome in terms of my deep learning expertise. It served as a platform for in-depth exploration into the complexities of deep learning models, raising a high comprehension of the architectural details, algorithms, and practical implementations. The hands-on experience gained through this experience played a significant role in consolidating and increasing my overall understanding of deep learning principles.

When it comes to teamwork, we actively collaborated on several projects, the majority of which yielded successful outcomes. Overall, I am highly content with both the team dynamics and the final result achieved in this particular project. Moreover, the experience collected during the development process has been substantially valuable, as it has contributed significantly to the increase of my knowledge curve.

6.2 Mark Vincze

As a software engineering student, I have always been eager to learn, embrace new challenges, and push boundaries. Starting a new project with my familiar team, the same individuals I have worked with on previous semester projects, felt both comfortable and exciting. It was in the middle of our sixth semester when we kicked off the bachelor project course, and we had just started our first machine learning course in the same semester, so we were nowhere close to imagining ourselves being able to take on a project like this. Working with Python, started in my internship semester and been using it since, so it was really helpful developing this project and being able to quickly prototype and create meaningful applications that can be executed on most of the machines.

About deep learning, with its immense complexity and high-level abstractions, presented a substantial leap from our recently acquired knowledge in machine learning. Despite the steep learning curve, we decided to go into an ambitious project: satellite image compression using deep learning, more specifically, autoencoders with binary constraints.

The initial stage of our project begun with intensive research. Which led us to explore autoencoders and the preprocessing of datasets. For preprocessing, there was a lot of back and forth on using or not using any filtering on the images, as in the beginning it was very difficult to validate if the enhanced images made any difference on the end result. But after, we encountered yet another challenge: our computers' resource management. Moreover, it was not just about being able to code a model structure together. The effective utilisation of resources, particularly utilising the GPU for training, emerged as an equally challenging terrain to navigate. However, these challenges only encouraged us on, and we took it upon ourselves to turn these constraints into opportunities as a learning path. We also dove into the technical depths of the autoencoders, attempting to dissect its mechanisms, understanding its relevance to image compression, and visualising potential applications. Simultaneously, we were grappling with Optuna, the tool for hyper-parameter tuning. Comprehending and utilising this tool was a great help for automating some of the lengthy processes of finding meaningful improvements in our architecture changes and evaluating what hyperparameters were giving the most sense to explore our final model with. As for the hashing functionality part of our project, we have experimented with different ways of implementing it in the model or separately but was not successful unfortunately, we need to learn more about binary hashing in the context of machine learning.

In this pursuit of the right model, the challenges of machine learning quickly became apparent. To bring out the true potential of each model, the appropriate set of hyperparameters – the 'perfect combo' – had to be found. There are no universal solutions in this field, as every model and every problem carry its own distinctive traits and peculiarities. The journey for the 'right' model, thus, was a demanding and rigorous one.

As our journey neared its conclusion, our focus turned to the deplorability of our models. Docker became our ally, providing us with the means to encapsulate our applications into deployable containers. The challenge here was to ensure the smooth operation of our hard-wrought models across diverse environments, a task we tackled with unravelled determination.

Guidance from our university supervisors was instrumental to our project's success. Their expertise, constructive feedback, and relentless support helped us a lot on how to navigate the complex landscape of deep learning, changes in our project's methodology and process. The demanding work from the project gave us an even deeper insight into machine and deep learning, I can take on a new problem with more confidence and know where to look for answers. I am certain that this project has fortified all of our technical skills and gathered valuable lessons in resilience, collaboration, and problem-solving, lessons that will be instrumental in our forthcoming software engineering pursuits.

6.3 Priyanka Shrestha

The bachelor's project stood out than other projects due to the heavy focus on machine learning rather than software components. The topic of the project was something I was interested in since sixth semester. During the initial phase, the group faced a challenge of having little knowledge of what we were supposed to do. However, with continuous research and dedication we were able to achieve commendable results.

In terms of the process and methodology used, the project took a significant change transitioning from Scrum to Kanban. I was nervous at first as it was a new experience for me, been accustomed to a more structured approach with Scrum. However, the adoption of

Kanban proved out to be a better fit for the project allowing us the flexibility and adaptability needed to make this project successful.

The project provided an opportunity for personal growth particularly in terms of research skills and technical knowledge such as working with GPUs. The process of trying out different models throughout the project was an invaluable learning curve for me. By exploring different models, I was able to gain insight into the strengths, weaknesses and applicability to our problem domain. Every failed model that I tried in the project served as a valuable learning process and contributed to my understanding and growth.

Throughout the semester, I have delved into various advanced concepts in machine learning, including autoencoders, hashing, and Barlow Twins models. Autoencoders have fascinated me with their ability to learn efficient representations of data by reconstructing inputs. Exploring their architectures and training methodologies has provided valuable insights into unsupervised learning and dimensionality reduction techniques. Additionally, studying Barlow Twins models with redundancy reduction has been an enlightening experience. Even though the Barlow Twins model could not be trained as expected we had to accept the fact that we did not have enough resources for it. Overall, this exploration of autoencoders, and Barlow Twins models has significantly expanded my understanding of cutting-edge techniques in the field of machine learning.

Being a part of the project and collaborating with the team members was an enjoyable experience. Our interest and motivation have been aligned since third semester. We have found common ground and shared enthusiasm for the subject matter and project at hand. I enjoyed and learned a lot from my team members and am grateful towards them and supervisor for the support and valuable feedback throughout the process.

7 Supervision

Supervision sessions were scheduled to guarantee the desired result PW wished to achieve with this project. The supervision was divided between processes and implementation details, PW taking advantage of the knowledge provided by the supervisors. The meetings were casual, productive, the supervisors gave feedback on the work presented, offering input and feedback based on the work provided, but also proposed personal suggestions and different approaches that the PW members did not consider.

PW members are grateful for the support, advice and time, the supervisors offered throughout the project development period and for the fast response whenever contacted through online channels.

8 Conclusions

Overall, the project demonstrated the team's commitment to continuous improvement and adaptation. By embracing the iterative approach, the team was able to make informed decisions to enhance the performance. The utilization of tools such as Jira facilitated the team with task management, tracking process and improve efficiency.

Through diligent efforts, collaboration and utilization of effective project management tool, the team successfully overcome the challenges and achieved valuable outcome. The team's dedication and hard work played a crucial role in the success of the project.

9 References

Tangerine, R., 2023. *Red Tangerine.* [Online]
Available at: https://www.redtangerine.org/agile-around-the-world/the-lewis-model/
[Accessed 9 January 2023].