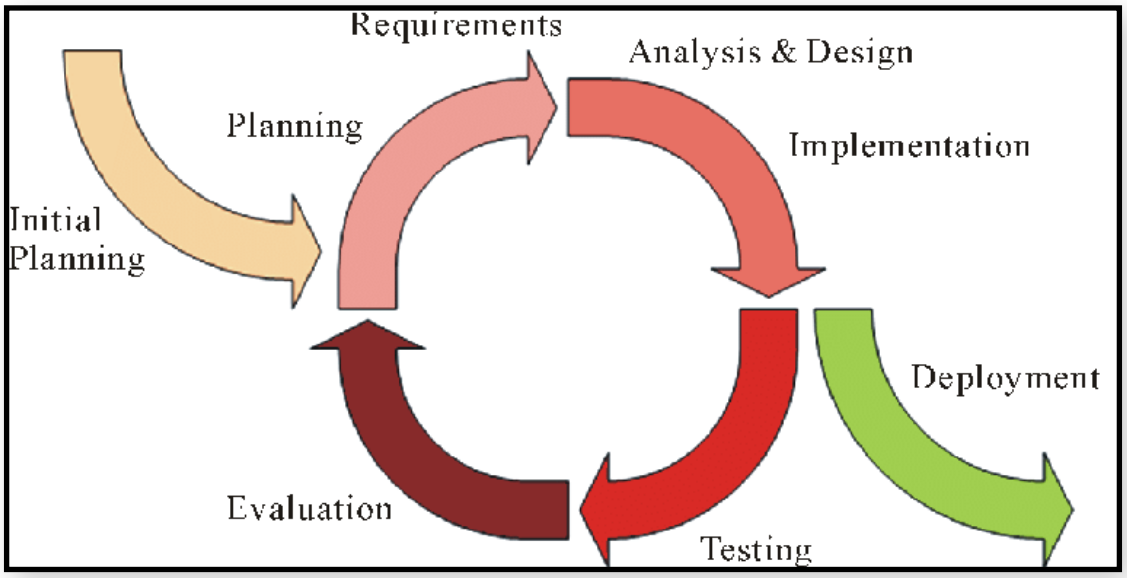
compress.io

SOFTWARE DEVELOPMENT

The software development life cycle (SDLC) is a framework defining tasks performed at each step in the software development process. SDLC is a structure followed by a development team within the software organisation. It consists of a detailed plan describing how to develop, maintain and replace specific software.



Our proposed Deep Learning model follows the Iterative Waterfall model of Software development life cycle.

- Initial Planning and Requirement Gatherings :

Initial planning and requirement gathering step includes the prior analysis of the past and current technologies in market. Based on which our model meets the current benchmarks set by the technologies in use.
We gathered information on the current compression standards i.e JPEG, JPEG2000, RAW, TIFF,etc.

- Implementation :

Tools and Technologies used:

- Python :

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasises readability and therefore reduces the cost of program maintenance.

- TensorFlow :

TensorFlow is a free and open-source software library for data flow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google.

- OpenCV :

OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real-time computer vision. In simple language it is library used for Image Processing. It is mainly used to do all the operation related to Images.

- Keras :

Keras is an API designed for human beings, not machines. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimises the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.

Implementation steps:

- Loading Data set
- Splitting the Dataset in training and testing parts
- Building a Neural Network

- Training the Neural Network

    -Input the data
    -Encoding the input

-

```
[ ] input_img= Input(shape=(784,))
    encoded = Dense(units=128, activation='relu')(input_img)
    encoded = Dense(units=64, activation='relu')(encoded)
    decoded = Dense(units=64, activation='relu')(encoded)
    decoded = Dense(units=128, activation='relu')(decoded)
    decoded = Dense(units=784, activation='sigmoid')(decoded)
    autoencoder=Model(input_img, decoded)
    encoder = Model(input_img, encoded)
    encoder.summary()
    autoencoder.compile(optimizer='adam', loss='binary_crossentropy', metrics=['a
    autoencoder.fit(X_train, X_train,
                    epochs=50,
                    batch_size=256,
                    shuffle=True,
                    validation_data=(X_test, X_test))
```

    -Decoding the encoded part
    -Making predictions

Evaluation :

Evaluating the final predictions made by the Neural Network and comparing the accuracy and loss with the expected values and try to achieve it using Optimizer and Loss function.

Optimizer : adam
Loss Function : binary_cross_entropy

Testing and final Deployment :

Testing the final model and deploying it on a web server with proper front end finishes the software development cycle of the proposed model.

```
60000/60000 [==============================] - 6s 103us/step - loss: 0.2282 -
Epoch 2/50
60000/60000 [==============================] - 5s 86us/step - loss: 0.1331 - a
Epoch 3/50
60000/60000 [==============================] - 5s 85us/step - loss: 0.1161 - a
Epoch 4/50
60000/60000 [==============================] - 5s 85us/step - loss: 0.1075 - a
Epoch 5/50
60000/60000 [==============================] - 5s 87us/step - loss: 0.1027 - a
Epoch 6/50
60000/60000 [==============================] - 5s 86us/step - loss: 0.0995 - a
Epoch 7/50
60000/60000 [==============================] - 5s 86us/step - loss: 0.0967 - a
Epoch 8/50
60000/60000 [==============================] - 5s 85us/step - loss: 0.0944 - a
Epoch 9/50
60000/60000 [==============================] - 5s 85us/step - loss: 0.0924 - a
Epoch 10/50
60000/60000 [==============================] - 5s 87us/step - loss: 0.0912 - a
Epoch 11/50
60000/60000 [==============================] - 5s 90us/step - loss: 0.0898 - a
Epoch 12/50
60000/60000 [==============================] - 5s 88us/step - loss: 0.0889 - a
Epoch 13/50
60000/60000 [==============================] - 5s 85us/step - loss: 0.0879 - a
Epoch 14/50
60000/60000 [==============================] - 5s 86us/step - loss: 0.0872 - a
Epoch 15/50
60000/60000 [==============================] - 5s 86us/step - loss: 0.0864 - a
Epoch 16/50
```