Architecture:

```
# example of creating a CNN with an efficient inception module
from keras.models import Model
from keras.layers import Input
from keras.layers import Conv2D
from keras.layers import MaxPooling2D, add
from keras.layers.merge import concatenate
from keras.layers import Dense
from keras.layers import Reshape
import tensorflow as tf
#from keras.utils import plot_model

# function for creating a projected inception module
def inception_module(layer_in, fil):
    # 1x1 conv
    conv1 = Conv2D(fil, (1,1), padding='same', activation='relu')(layer_in)
    # 3x3 conv
    conv3 = Conv2D(fil, (1,1), padding='same', activation='relu')(layer_in)
    conv3 = Conv2D(fil, (3,3), padding='same', activation='relu')(conv3)
    # 5x5 conv
    conv5 = Conv2D(fil, (1,1), padding='same', activation='relu')(layer_in)
    conv5 = Conv2D(fil, (3,3), padding='same', activation='relu')(conv5)
    conv5 = Conv2D(fil, (3,3), padding='same', activation='relu')(conv5)
    # 3x3 max pooling
    #pool = MaxPooling2D((3,3), strides=(1,1), padding='same')(layer_in)
    layer_out = concatenate([conv1, conv3, conv5], axis=-1)
    layer_out = Conv2D(layer_in.shape[3], (1,1), padding='same', activation='relu')(layer_out)
    # concatenate filters, assumes filters/channels last
    layer_out = add([layer_in, layer_out])
    return layer_out

def inception_module2(layer_in, fil, in1, in2):
    # 1x1 conv
    conv1 = Conv2D(fil, (1,1), padding='same', activation='relu')(layer_in)
    # 3x3 conv
    conv3 = Conv2D(fil, (1,1), padding='same', activation='relu')(layer_in)
    conv3 = Conv2D(fil, (3,3), padding='same', activation='relu')(conv3)
    # 5x5 conv
    conv5 = Conv2D(fil, (1,1), padding='same', activation='relu')(layer_in)
    conv5 = Conv2D(fil, (3,3), padding='same', activation='relu')(conv5)
    conv5 = Conv2D(fil, (3,3), padding='same', activation='relu')(conv5)
    # 3x3 max pooling
    #pool = MaxPooling2D((3,3), strides=(1,1), padding='same')(layer_in)
    layer_out_B1 = concatenate([conv1, conv3, conv5], axis=-1, name='B1')
    B1_model= Model(inputs=[in1,in2], outputs=layer_out_B1)
    layer_out_B2 = Conv2D(layer_in.shape[3], (1,1), padding='same', activation='relu',  name='B2')(layer_out_B1)
    B2_model = Model(inputs=[in1,in2], outputs=layer_out_B2)
    # concatenate filters, assumes filters/channels last
    layer_out = add([layer_in, layer_out_B2])
    return layer_out, B1_model, B2_model

def convolution(layer_in, fil, act='relu'):
  return Conv2D(fil, (1,1), padding='same', activation=act)(layer_in)
# define model input
input1 = Input(shape=(32, 32, 1), name='watermarking_image')
# add inception block 1
encoding = inception_module(input1, 32)
encoding = inception_module(encoding, 32)
encoding = convolution(encoding, 24)
encoding = inception_module(encoding, 32)
encoding = inception_module(encoding, 32)
Encoded =  Conv2D(48, (1,1), padding='same', activation='relu')(encoding)
Encoded =  Reshape((128, 128,3))(Encoded)
encoded_model = Model(inputs=input1, outputs=Encoded, name= 'Encoder')

encoded_output = Input(shape=(128, 128,3))
input2 = Input(shape=(128, 128, 3), name='Cover_image')
embedder , B1_model, B2_model= inception_module2(encoded_output, 32, encoded_output, input2)

embedder = concatenate([embedder, input2], axis=-1)
embedder = inception_module(embedder,32)
Embedded = convolution(embedder, 3)
embedded_model = Model(inputs=[encoded_output,input2], outputs=Embedded, name='Embeder')
```

```
N=3
embedded_output = Input(shape=(128, 128, 3))
Invariance = Dense(N,activation='tanh', name = 'invariant')(embedded_output)
invariant_model = Model(inputs=embedded_output, outputs=Invariance, name='Invariance')

invariant_output = Input(shape=(128, 128, N))
extractor = inception_module(invariant_output,32)
extractor = convolution(extractor,3)
extractor_model = Model(inputs=invariant_output, outputs=extractor, name= 'Extractor')

extractor_output = Input(shape=(128, 128, 3))
decoder = Reshape((32, 32,48))(extractor_output)
decoder = inception_module(decoder,32)
decoder = inception_module(decoder,32)
decoder = convolution(decoder, 24)
decoder = inception_module(decoder, 32)
decoder = inception_module(decoder, 32)
output = convolution(decoder, 1, act='sigmoid')
decoder_model = Model(inputs=extractor_output, outputs=output, name='Decoder')

model_output = decoder_model(extractor_model(invariant_model(embedded_model([encoded_model(input1),input2]))))
model = Model(inputs=[input1,input2], outputs=model_output)


# summarize model
model.summary()

dot_img_file = '/tmp/model_1.png'
tf.keras.utils.plot_model(model, to_file=dot_img_file, show_shapes=True)
```

```
Model: "model_2"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
watermarking_image (InputLayer) [(None, 32, 32, 1)]  0
__________________________________________________________________________________________________
Encoder (Functional)            (None, 128, 128, 3)  122258      watermarking_image[0][0]
__________________________________________________________________________________________________
Cover_image (InputLayer)        [(None, 128, 128, 3) 0
```

Function for calculating loss:

```
def gram_matrix(x):
    x = tf.transpose(x)
    features = tf.reshape(x, (tf.shape(x)[0], -1))
    gram = tf.matmul(features, tf.transpose(features))
    return gram


def dis(x,y):
    channels = 3
    size = 128 * 128
    return tf.reduce_sum(tf.square(x - y)) / (4.0 * (channels ** 2) * (size ** 2))


def style_loss(b1_w,b1_m,b2_w,b2_m):
    x1 = dis(gram_matrix(b1_w),gram_matrix(b1_m))
    x2 = dis(gram_matrix(b2_w),gram_matrix(b2_m))
    return 0.5*(x1+x2)
```

| output: | (None, 128, 128, 3) | | | output: | [(None, 128, 128, 3)] |

Gradient calculation and loss calcutation:

```
import keras.backend as K
loss_fn = tf.keras.losses.MeanAbsoluteError()
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-4)
encoder_optimizer = tf.keras.optimizers.Adam(learning_rate=1e-4)
embedder_optimizer = tf.keras.optimizers.Adam(learning_rate=1e-4)
decoder_optimizer = tf.keras.optimizers.Adam(learning_rate=1e-4)
def training_step(w,c):
    with tf.GradientTape() as encoder_tape, tf.GradientTape() as embedder_tape, tf.GradientTape() as tape, tf.GradientTape() as decoder_tape
        wfi = encoded_model(w)
        mi = embedded_model([wfi,c])
        ti = invariant_model(mi)
        wfo = extractor_model(ti)
        wo = decoder_model(wfo)
        WO = model([w,c])

        fidelity_loss = loss_fn(c,mi)
        print("fidelity_loss:", fidelity_loss)

        extraction_loss = loss_fn(w,WO)
        print("extraction_loss:", extraction_loss)
        #loss += sum(vae.losses)

        wf_B1_output= B1_model([wfi,c])
        wf_B2_output= B2_model([wfi,c])
        mi_B1_output= B1_model([mi,c])
        mi_B2_output= B2_model([mi,c])
        information_loss = style_loss(wf_B1_output,mi_B1_output,wf_B2_output,mi_B2_output)
        print("information_loss:", information_loss)

        w = invariant_model.get_layer('invariant').get_weights()[0]
        w_sum_over_input_dim = tf.reduce_sum(tf.square(w), axis=0)
        w_ = tf.expand_dims(w_sum_over_input_dim, 1)
        h_ = tf.square(1-ti*ti)
        h_times_w_ = tf.matmul(h_, w_)
        jacobian = 0.01*tf.reduce_mean(h_times_w_)

        print("invariant_loss:", jacobian)
        #print("invariant_loss:", invariant_loss)

        total_loss = fidelity_loss  + information_loss
        model_loss = extraction_loss + jacobian +total_loss

    grads = tape.gradient(model_loss, model.trainable_weights)
```

```
    optimizer.apply_gradients(zip(grads, model.trainable_weights))

    #grads = decoder_tape.gradient(extraction_loss, decoder_model.trainable_weights)
    #decoder_optimizer.apply_gradients(zip(grads, decoder_model.trainable_weights))


    grads = encoder_tape.gradient(total_loss, encoded_model.trainable_weights)
    encoder_optimizer.apply_gradients(zip(grads, encoded_model.trainable_weights))



    grads = embedder_tape.gradient(total_loss, embedded_model.trainable_weights)
    embedder_optimizer.apply_gradients(zip(grads, embedded_model.trainable_weights))





    return total_loss
```

Image loading: watermarking data = mnist; cover image = flower dataset

```
# Model / data parameters
from tensorflow import keras
import numpy as np
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255

x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)

x_train = tf.image.resize(x_train, [32,32])
x_test = tf.image.resize(x_test, [32,32])

import pathlib
dataset_url = "https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz"
data_dir = tf.keras.utils.get_file(origin=dataset_url,
                                   fname='flower_photos',
                                   untar=True)
data_dir = pathlib.Path(data_dir)



train_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(128, 128),
  batch_size=32)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz
228818944/228813984 [==============================] - 2s 0us/step
Found 3670 files belonging to 5 classes.
Using 2936 files for training.
```

```
from tensorflow.keras.layers.experimental.preprocessing import Rescaling
rescale = Rescaling(scale=1.0/255)
train_ds = train_ds.map(lambda image,label:(rescale(image),label))
```

Training steps:

```
for epoch in range(100):
  losses = []  # Keep track of the losses over time.
  start=0
  step=32
  end=step

  for i,c in enumerate(train_ds):
          w=x_train[start:end]
```

```
        start=end
        end+=step
        loss = training_step(w,c[0])
        losses.append(loss)
        #print("Step:", i, "Loss:", sum(losses) / len(losses))
        if i== 90:
          break
```

```
fidelity_loss: tf.Tensor(0.3437144, shape=(), dtype=float32)
extraction_loss: tf.Tensor(0.48208773, shape=(), dtype=float32)
information_loss: tf.Tensor(0.180002, shape=(), dtype=float32)
invariant_loss: tf.Tensor(0.024093155, shape=(), dtype=float32)
fidelity_loss: tf.Tensor(0.3410124, shape=(), dtype=float32)
extraction_loss: tf.Tensor(0.4751768, shape=(), dtype=float32)
information_loss: tf.Tensor(0.13131961, shape=(), dtype=float32)
invariant_loss: tf.Tensor(0.024162557, shape=(), dtype=float32)
fidelity_loss: tf.Tensor(0.3598534, shape=(), dtype=float32)
extraction_loss: tf.Tensor(0.47245753, shape=(), dtype=float32)
information_loss: tf.Tensor(0.1456432, shape=(), dtype=float32)
invariant_loss: tf.Tensor(0.024085531, shape=(), dtype=float32)
fidelity_loss: tf.Tensor(0.33269504, shape=(), dtype=float32)
extraction_loss: tf.Tensor(0.46971136, shape=(), dtype=float32)
information_loss: tf.Tensor(0.08670299, shape=(), dtype=float32)
invariant_loss: tf.Tensor(0.024108898, shape=(), dtype=float32)
fidelity_loss: tf.Tensor(0.3469181, shape=(), dtype=float32)
extraction_loss: tf.Tensor(0.46385002, shape=(), dtype=float32)
information_loss: tf.Tensor(0.05263534, shape=(), dtype=float32)
invariant_loss: tf.Tensor(0.024239292, shape=(), dtype=float32)
fidelity_loss: tf.Tensor(0.36179316, shape=(), dtype=float32)
extraction_loss: tf.Tensor(0.46134263, shape=(), dtype=float32)
information_loss: tf.Tensor(0.06756106, shape=(), dtype=float32)
invariant_loss: tf.Tensor(0.024104046, shape=(), dtype=float32)
fidelity_loss: tf.Tensor(0.3646993, shape=(), dtype=float32)
extraction_loss: tf.Tensor(0.45599893, shape=(), dtype=float32)
information_loss: tf.Tensor(0.031849567, shape=(), dtype=float32)
invariant_loss: tf.Tensor(0.024227552, shape=(), dtype=float32)
fidelity_loss: tf.Tensor(0.3753105, shape=(), dtype=float32)
extraction_loss: tf.Tensor(0.45092973, shape=(), dtype=float32)
information_loss: tf.Tensor(0.042343102, shape=(), dtype=float32)
invariant_loss: tf.Tensor(0.02417113, shape=(), dtype=float32)
fidelity_loss: tf.Tensor(0.3451001, shape=(), dtype=float32)
extraction_loss: tf.Tensor(0.45070636, shape=(), dtype=float32)
information_loss: tf.Tensor(0.014630157, shape=(), dtype=float32)
invariant_loss: tf.Tensor(0.024338555, shape=(), dtype=float32)
fidelity_loss: tf.Tensor(0.37387598, shape=(), dtype=float32)
extraction_loss: tf.Tensor(0.44792917, shape=(), dtype=float32)
information_loss: tf.Tensor(0.030508615, shape=(), dtype=float32)
invariant_loss: tf.Tensor(0.024200099, shape=(), dtype=float32)
fidelity_loss: tf.Tensor(0.35631132, shape=(), dtype=float32)
extraction_loss: tf.Tensor(0.44266248, shape=(), dtype=float32)
information_loss: tf.Tensor(0.019415494, shape=(), dtype=float32)
invariant_loss: tf.Tensor(0.024217108, shape=(), dtype=float32)
fidelity_loss: tf.Tensor(0.38884723, shape=(), dtype=float32)
extraction_loss: tf.Tensor(0.44434792, shape=(), dtype=float32)
information_loss: tf.Tensor(0.02065253, shape=(), dtype=float32)
invariant_loss: tf.Tensor(0.024200145, shape=(), dtype=float32)
fidelity_loss: tf.Tensor(0.35614526, shape=(), dtype=float32)
extraction_loss: tf.Tensor(0.43864214, shape=(), dtype=float32)
information_loss: tf.Tensor(0.018545428, shape=(), dtype=float32)
invariant_loss: tf.Tensor(0.024157748, shape=(), dtype=float32)
fidelity_loss: tf.Tensor(0.34880304, shape=(), dtype=float32)
extraction_loss: tf.Tensor(0.4400568, shape=(), dtype=float32)
information_loss: tf.Tensor(0.013204791, shape=(), dtype=float32)
invariant_loss: tf.Tensor(0.024413466, shape=(), dtype=float32)
fidelity_loss: tf.Tensor(0.38504684, shape=(), dtype=float32)
extraction_loss: tf.Tensor(0.43370116, shape=(), dtype=float32)
```

Generate output image:

Double-click (or enter) to edit

```
from matplotlib import pyplot as plt
def generate_images(in1, in2, mod1, mod2, mod3):
  extracted = mod1([in1,in2])
  encoded = mod2(in1)
  embedded = mod3([encoded,in2])
  plt.figure(figsize=(15, 15))
  xxx=np.squeeze(in1[0],axis=-1)
  yyy=np.squeeze(extracted[0],axis=-1)
  print(xxx.shape)
```

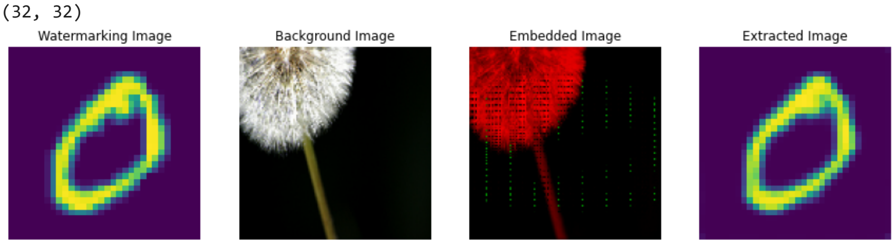
```
    display_list = [xxx, in2[0], embedded[0], yyy]
    title = ['Watermarking Image', 'Background Image', 'Embedded Image','Extracted Image']

    for i in range(4):
      plt.subplot(1, 4, i+1)
      plt.title(title[i])
      # getting the pixel values between [0, 1] to plot it.
      plt.imshow(display_list[i] )
      plt.axis('off')
    plt.show()


  for i,c in enumerate(train_ds):
    generate_images(x_train[i:i+1], c[0][0:1], model, encoded_model, embedded_model)
    if i>=2:
      break
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0
(32, 32)
```


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0
(32, 32)
```




```
(32, 32)
```