DSA526 Project Report
Super Image Resolution Using RNN

Team 16 : Hari Krishna J 224101064
Pavan Kumar S 224101067
Jaya Teja Reddy P 224101065
Venkata Sai Sri Ram 224101068

1

1. Problem Statement:

The Problem is given an image with low pixel resolution, and the target is to produce a high pixel resolution image of the given input image. The objective here is to develop an RNN which uses CNN as a base neural network that takes the low pixel resolution image and outputs high pixel resolution image

2. Data Set:

- The Data set used in our project is collected from Kaggle, which contains low-resolution and its high-resolution images. It contains two folders and one CSV file.
- Image_data.csv contains indexing of images of the folders ”low res” and ”high res.”
- low res: This folder contains low-resolution images of respective images of high-resolution images, and there are three types of depixelated images, namely 1/2,1/4, and 1/6th of the original image.
- Naming in ”low res” is done accordingly example: 245_2.jpg means depixelated 1/2th of the original image similarly X_4.jpg, X_6.jpg for ¼ and 1/6 depixalated images respectively.
- Dataset:`https://www.kaggle.com/datasets/quadeer15sh/image-super-resolution-from-unsplash`

3. Related Work:

Image Super Resolution refers to the task of enhancing the resolution of an image from low resolution (LR) to high (HR). It is popularly used in the following applications:

- **Surveillance:** to detect, identify, and perform facial recognition on low-resolution images obtained from security cameras.
- **Medical:** Capturing high-resolution MRI images can be tricky when it comes to scanning time, spatial coverage, and signal-to-noise ratio (SNR). Super-resolution helps resolve this by generating high-resolution MRI from otherwise low-resolution MRI images.
- **Media:** super-resolution can be used to reduce server costs, as media can be sent at a lower resolution and upscaled on the fly.

Deep learning techniques have been fairly successful in solving the problem of image and video super-resolution. There are many methods used to solve this task. Some of them are following:

- Pre-Upsampling Super Resolution
- Post-Upsampling Super Resolution
- Residual Networks
- Multi-Stage Residual Networks
- Recursive Networks
- Progressive Reconstruction Networks
- Multi-Branch Networks
- Attention-Based Networks
- Generative Models

In the project, we used a Recurrent neural network to develop a model which used CNN as base network for image super-resolution

4. Methods:

The project is divided into small modules which carry out a part of the work in the project, and the modules are namely

- Data Collection & Cleaning
- Making Train, validation and Test data
- Model Description
- Model Training
- Validation and testing

4.1. Dataset Split:

- Entire data set split into two parts train=85% and validation=15%
- **High resolution images :** train_hiresimage_generator(85%) and val_hiresimage_generator(15%)
- **Low resolution images:** train_lowresimage_generator(85%) and val_lowresimage_generator(15%)

4.2. Model Description:

- corresponding capture contains model description
- It contains 12 layers and four recurrent between layers
- Activation function used: LeakyReLu
- Normalization : Batch Normalization
- Stride : 2 and kernel: 3×3
- Total params: 9,602,898
- Trainable params: 9,600,338
- Non-trainable params: 2,560

The block diagram given below describes the Model architecture of the model used in our project.

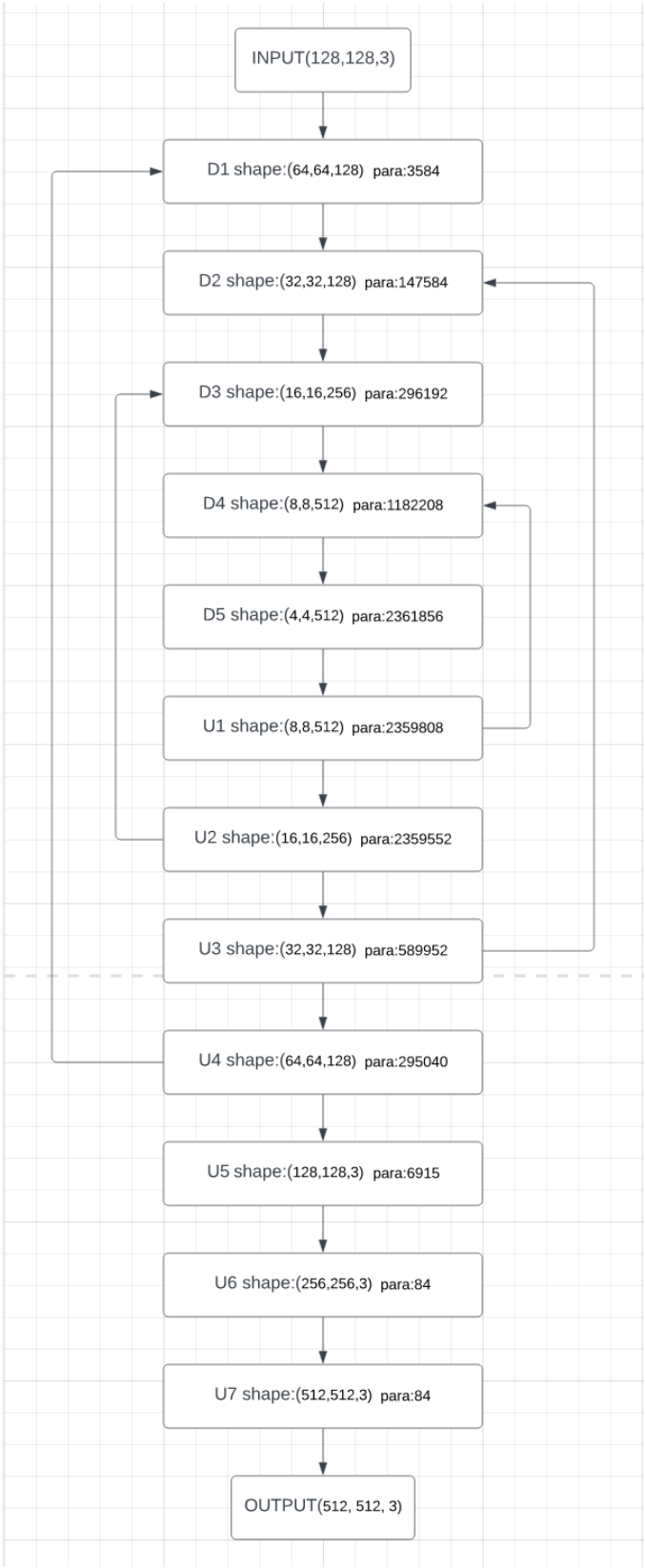

Figure 1. Model Architecture

5. Experiment:

```
import tensorflow as tf
import tensorflow.keras.layers as tfl
import os
import pandas as pd
from tqdm import tqdm
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
import keras
from keras.models import Sequential
import keras
import os
from tqdm import tqdm
import re
import matplotlib.pyplot as plt
from tensorflow.keras.utils import img_to_array
import numpy as np
import pandas as pd
import os
import re
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense, Conv2D, MaxPooling2D, Dropout, Conv2DTranspose, UpSampling2D, add
from tensorflow.keras.models import Model
from tensorflow.keras import regularizers
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.python.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
import matplotlib.pyplot as plt
```

Figure 2. these are the libraries used in the project

5.1. Dataset reading:

```
base_directory = '/home/jtreddy/Downloads/archive/Image Super Resolution - Unsplash'
hires_folder = os.path.join(base_directory, 'high res')
lowres_folder = os.path.join(base_directory, 'low res')
```

```
data = pd.read_csv("/home/jtreddy/Downloads/archive/Image Super Resolution - Unsplash/image_data.csv")
data['low_res'] = data['low_res'].apply(lambda x: os.path.join(lowres_folder,x))
data['high_res'] = data['high_res'].apply(lambda x: os.path.join(hires_folder,x))
data.head()
```

Figure 3. Data Reading

5.2. Data Split:

```
batch_size = 4

image_datagen = ImageDataGenerator(rescale=1./255,validation_split=0.15)
mask_datagen = ImageDataGenerator(rescale=1./255,validation_split=0.15)

train_hiresimage_generator = image_datagen.flow_from_dataframe(
        data,
        x_col='high_res',
        target_size=(512, 512),
        class_mode = None,
        batch_size = batch_size,
        seed=42,
        subset='training')

train_lowresimage_generator = image_datagen.flow_from_dataframe(
        data,
        x_col='low_res',
        target_size=(128, 128),
        class_mode = None,
        batch_size = batch_size,
        seed=42,
        subset='training')

val_hiresimage_generator = image_datagen.flow_from_dataframe(
        data,
        x_col='high_res',
        target_size=(512, 512),
        class_mode = None,
        batch_size = batch_size,
        seed=42,
        subset='validation')

val_lowresimage_generator = image_datagen.flow_from_dataframe(
        data,
        x_col='low_res',
        target_size=(128, 128),
        class_mode = None,
        batch_size = batch_size,
        seed=42,
        subset='validation')

train_generator = zip(train_lowresimage_generator, train_hiresimage_generator)
val_generator = zip(val_lowresimage_generator, val_hiresimage_generator)

def imageGenerator(train_generator):
    for (low_res, hi_res) in train_generator:
            yield (low_res, hi_res)
```

Figure 4. Data Split - Train: 85% Validation: 15%

5.3. Model Description:

```
LSIZE = 128
HSIZE = 512
from keras import layers
def down(filters , kernel_low_size, apply_batch_normalization = True):
    downsample = tf.keras.models.Sequential()
    downsample.add(layers.Conv2D(filters,kernel_low_size,padding = 'same', strides = 2))
    if apply_batch_normalization:
        downsample.add(layers.BatchNormalization())
    downsample.add(keras.layers.LeakyReLU())
    return downsample
def up(filters, kernel_low_size, dropout = False):
    upsample = tf.keras.models.Sequential()
    upsample.add(layers.Conv2DTranspose(filters, kernel_low_size,padding = 'same', strides = 2))
    if dropout:
        upsample.dropout(0.2)
    upsample.add(keras.layers.LeakyReLU())
    return upsample
def model():
    inputs = layers.Input(shape= [LSIZE,LSIZE,3])
    print(inputs.shape)
    d1 = down(128,(3,3),False)(inputs)
    print(d1.shape)
    d2 = down(128,(3,3),False)(d1)
    print(d2.shape)
    d3 = down(256,(3,3),True)(d2)
    print(d3.shape)
    d4 = down(512,(3,3),True)(d3)
    print(d4.shape)
    d5 = down(512,(3,3),True)(d4)
    print(d5.shape)
    #upsampling
    u1 = up(512,(3,3),False)(d5)
    print(u1.shape)
    u1 = layers.concatenate([u1,d4])
    u2 = up(256,(3,3),False)(u1)
    print(u2.shape)
    u2 = layers.concatenate([u2,d3])
    u3 = up(128,(3,3),False)(u2)
    print(u3.shape)
    u3 = layers.concatenate([u3,d2])
    u4 = up(128,(3,3),False)(u3)
    u4 = layers.concatenate([u4,d1])
    u5 = up(3,(3,3),False)(u4)
    u6 = up(3,(3,3),False)(u5)
    u7 = up(3,(3,3),False)(u6)
    output = layers.Conv2D(3,(2,2),strides = 1, padding = 'same')(u7)
    return tf.keras.Model(inputs=inputs, outputs=output)
model = model()
model.summary()
```

Figure 5. Model Architecture Description

5.4. Model Training:

```
train_samples = train_hiresimage_generator.samples
val_samples = val_hiresimage_generator.samples

train_img_gen = imageGenerator(train_generator)
val_image_gen = imageGenerator(val_generator)
```

```
batch_size=1
model.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = 0.001), loss = 'mean_absolute_error',
              metrics = ['acc'])
H = model.fit(train_img_gen,
                    steps_per_epoch=train_samples//batch_size,
                    validation_data=val_image_gen,
                    validation_steps=val_samples//batch_size,
                    epochs=20)
```

Figure 6. Sampling and Training

6. Results:

6.1. Validation:

```
n = 0
for i,m in val_generator:
    img,mask = i,m
    sr1 = model.predict(img)
    if n < 20:
        fig, axs = plt.subplots(1 , 3, figsize=(20,4))
        axs[0].imshow(img[0])
        axs[0].set_title('Low Resolution Image')
        axs[1].imshow(mask[0])
        axs[1].set_title('High Resolution Image')
        axs[2].imshow(sr1[0])
        axs[2].set_title('Predicted High Resolution Image')
        plt.show()
        n+=1
    else:
        break
```

Figure 7. Model validation

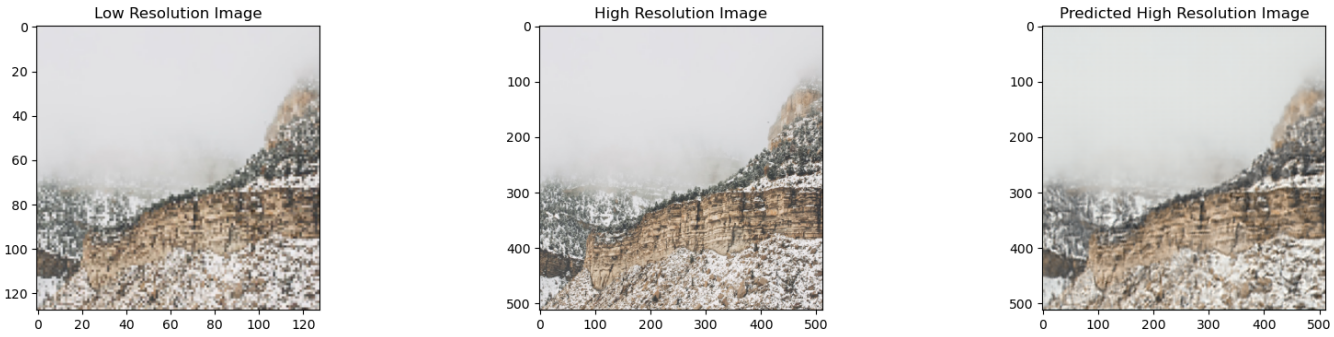

Figure 8. Output 1

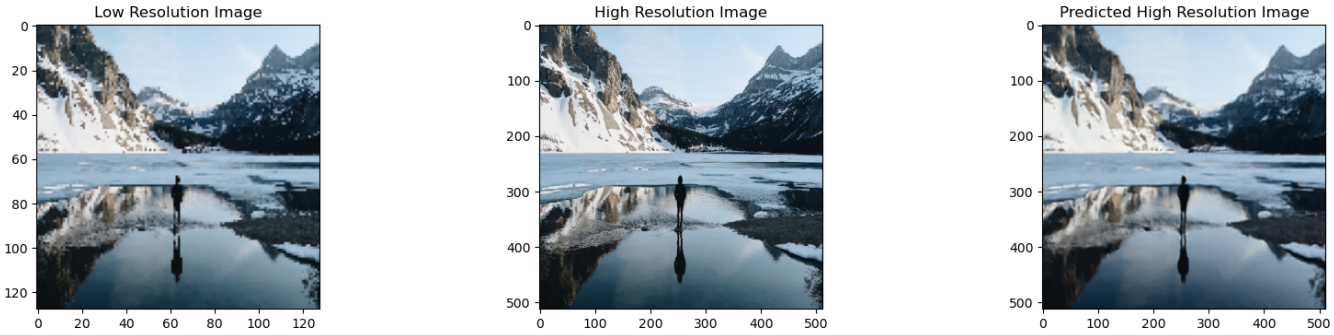

Figure 9. Output 2

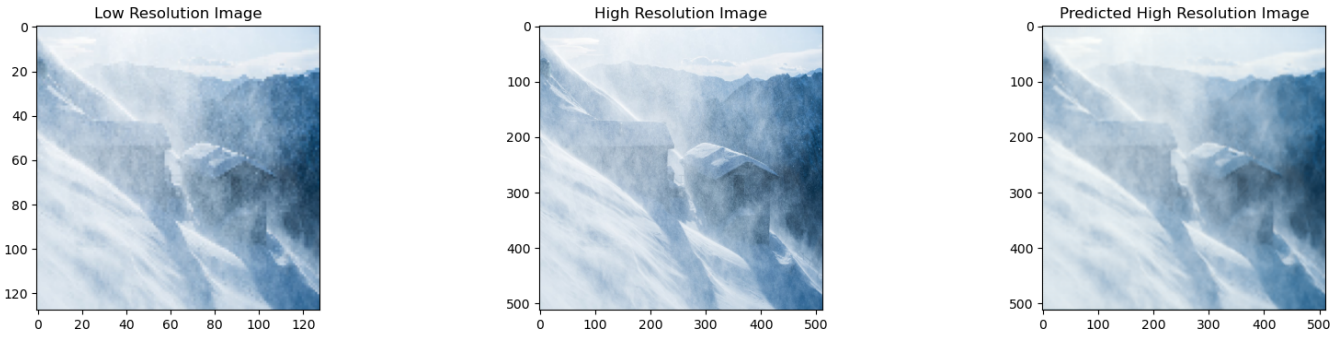

Figure 10. Output 3

6.2. Model Performance:

```
plt.style.use("ggplot")
plt.figure()
N=20
plt.plot(np.arange(0,N),H.history["loss"],label="train_loss")
plt.plot(np.arange(0,N),H.history["val_loss"],label="val_loss")
plt.plot(np.arange(0,N),H.history["acc"],label="train_acc")
plt.plot(np.arange(0,N),H.history["val_acc"],label="val_acc")
plt.title("accuracy and loss 256 *4")
plt.xlabel("Epoch #")
plt.ylabel("loss/acc")
plt.legend(loc="upper right")
```

Figure 11. Model Performance plot

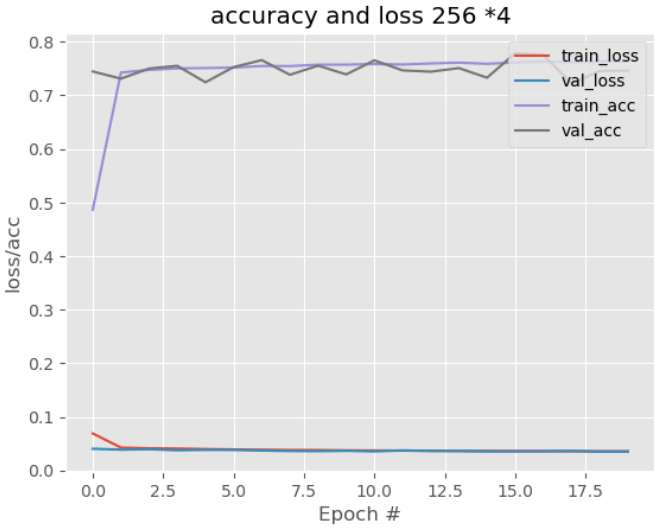

Figure 12. Accuracy Plot

7. Conclusion:

In this project, we developed RNN based super-resolution model, which can be applied to practically any kind of low-quality image, such as face images, vehicle license plates, satellite imagery, and old photos, with numerous applications including in important domains such as security and law enforcement.

However, there remain many challenges to be tackled. One such problem is our model for some images, and it loses its color composition. For example, refer Figures 13,14. The main cause for this loss can be several reasons which are :

- Not having enough data set.
- Small data but has wide range of landscapes, textures, etc. This imbalance causes improper training of the model.
- The internal layer weights and gradients can be a reason.
- The trained model biased towards few type of images

The current model accuracy is 76%. There is scope for further development in this project which can be developed further, along with remaining challenges and directions for future work.

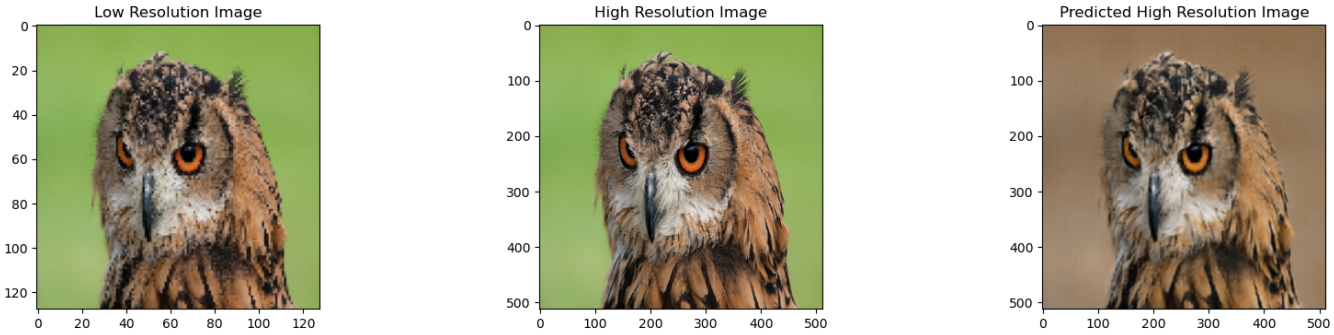

Figure 13. Failed Output 1

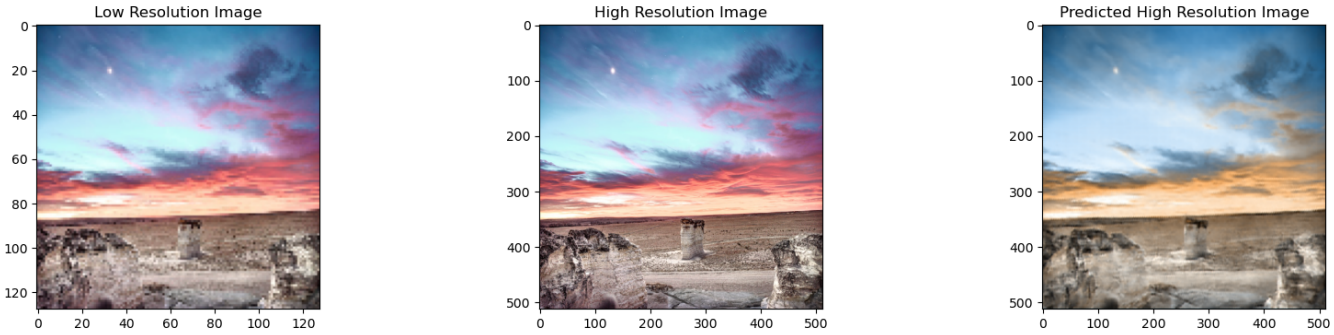

Figure 14. Failed Output 2

Github Link to Sourcecode: https://github.com/jayatejareddy/Image-Super-Resolution-IPML-Project/tree/main