# Comparison of classification model accuracy with Low-Resolution input images vs Super-Resolution input images obtained using Deep Convolutional Network

**Sanket Khullar, Humayun Akhtar**
**Mechanical Engineering, Texas AM University**
**sanketkhullar@tamu.edu, humayun.akhtar@tamu.edu**

## Abstract

Our method uses two end-to-end mappings between the low/high-resolution images and an image classifier to identify and categorize the 10 class labels. The first mapping is represented as a deep convolutional neural network (CNN) that takes the low-resolution image as the input and outputs the high-resolution one. And the second mapping is represented as CNN with input images and outputs the labels of the images. This report tends to compare the image classifier (second mapping of CNN for the 10-label classification) for both the case with raw input as low-resolution images and our classifier (with the input of high-resolution image) to verify if the higher resolution results in higher accuracy.

## 1 INTRODUCTION

The main idea remains to use the pre-trained convolutional network to obtain high-resolution images (from data set of raw images with 32x32x3 pixels) and then use it to train another convolutional neural network for its classification into 10 labels. The raw image data set will also be trained on the same CNN (with similar hyperparameters). The approach of transfer learning can be used to train the CNN model with the images of the data set to obtain the high-resolution output image. These high-resolution images can be stored and can be used at a later stage. The high-resolution images obtained from a Super-Resolution Deep Convolutional Network can be used to train a Neural network to predict the most appropriate label of the image. Finally, the CNN classifier model trained on high-resolution images can be compared to the CNN trained on raw images to verify if enhancing the resolution of the input images produces a higher accuracy of the classification. The data used in the project is the CIFAR-10 dataset (Canadian Institute for Advanced Research, 10 classes). It is a subset of the Tiny Images dataset and consists of 60000 32x32 color images. The images are labeled with one of 10 mutually exclusive classes: airplane, automobile (but not truck or pickup truck), bird, cat, deer, dog, frog, horse, ship, and truck (but not pickup truck). There are 6000 images per class with 5000 training and 1000 testing images per class.

## 2 Methodology

### 2.1 Preprocessing of data

The CIFAR-10 data set consists of 60000 32x32 color images of 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. For the initial experimentation, the data set is divided into a training batch of 8000 images and a test batch of 2000 images. Between them, the training batches contain exactly 5000 images from each class. As the CIFAR-10 data is available

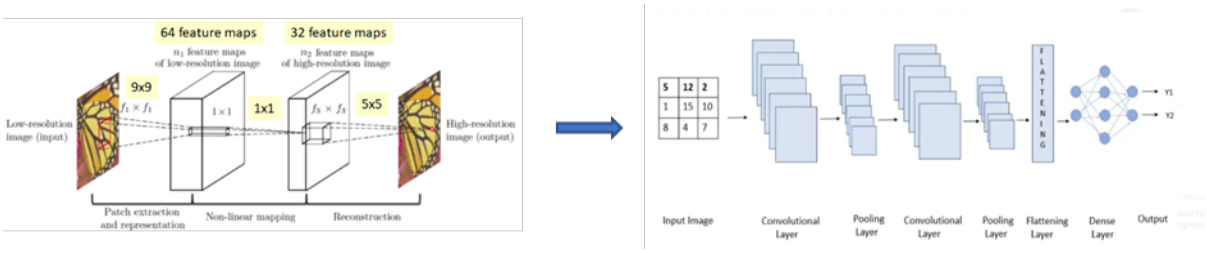


Figure 1: The images from the first CNN outputs the super-resolution images which are used as the input for the second CNN for the classification into 10 categories.

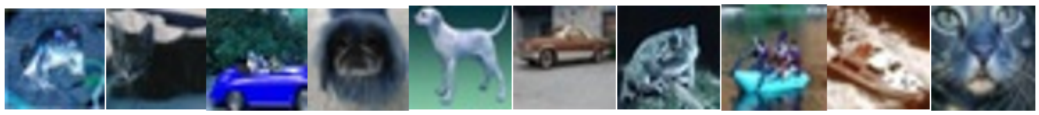

Figure 2: Few of the images obtained form the conversion of the RGB values from the CIFAR-10 dataset

in form of an array of 5000x32x32x3 and the CNN model is used for generating super-resolution images – ESRGAN; takes images as input. Finally, the CNN classification model will take input images in the form of an RGB values array.

Steps for processing the data:

a) To import the data, we used TensorFlow and matplotlib library. Since the dataset is used globally, one can directly import the dataset from the Keras module of the TensorFlow library.

b) The imported data was in the form of RGB value arrays. So the RGB values were converted to 32x32 pixel images to be fed into the CNN network. The output of the CNN is the super Resolution images. (Super Resolution CNN - ESRGAN

c) The super-resolution images were then converted to RGB values array.

Neural Networks are the programmable patterns that helps to solve complex problems and bring the best achievable output. Deep Learning as we all know is a step ahead of Machine Learning, and it helps to train the Neural Networks for getting the solution of questions unanswered and or improving the solution!

## 3 Initial Experimentation

### 3.1 Setup

CIFAR-10 data set has 50000 images for training the CNN model, but for the nascent stage, 11220 images have been used. 70 percent of the images were used for training the model and 30 percent were used to evaluate the performance of the model. To generate the super-resolution images, the transfer learning approach from the Enhanced SRGAN (ESRGAN) model from Xinntao, Researcher at Tencent ARC Lab, (Applied Research Center) is used.

### 3.2 CNN Classifier

The convolutional network has 2 convolutional layers with two pooling layers. Each convolutional layer has 32 filters. From each such filter, the convolutional layer learns something about the image, like hue, boundary, shape/feature. The value of the parameters should be in the power of 2. We have used SAME padding. In the SAME padding, there is a layer of zeros padded on all the boundaries of the image, so there is no loss of data. Moreover, the dimension of the output of the image after convolution is the same as the input of the image. RELU activation functinon is used in the dense layer and the output. While compiling the model, we need to take into account the loss function. There are two loss functions used generally, Sparse Categorical Cross-Entropy(scce) and Categorical


Figure 3: The output obtained after running all the images of the SR CIFAR-10 through the ESRGAN-CNN to get super resolution images

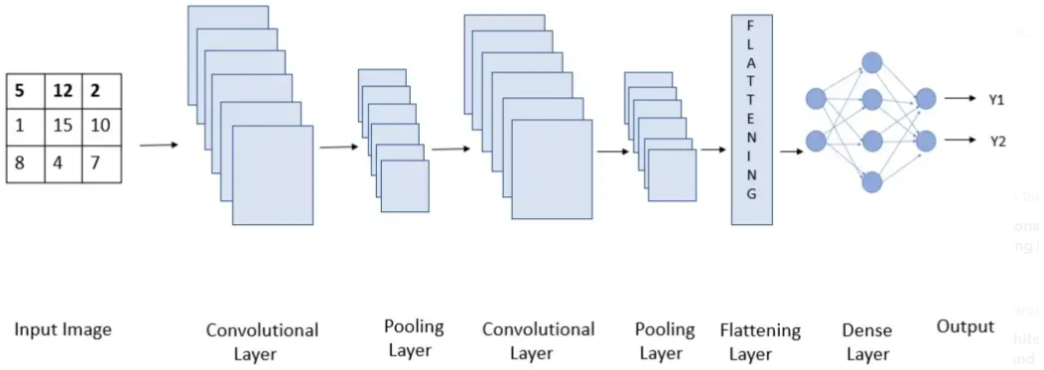


Figure 4: The architecture of the convolutional neural network used for the classification into 10 labels

Cross-Entropy(cce). Sparse Categorical Cross-Entropy(scce) is used when the classes are mutually exclusive, the classes are totally distinct then this is used. Categorical Cross-Entropy is used when a label or part can have multiple classes. In our scenario, the classes are totally distinctive so we are using Sparse Categorical Cross-Entropy.

### 3.3 Method

The dataset has 11220 color images comprising 10 different classes. We are giving two different inputs to the same CNN model (classification rule). As the first input to the model the image size is 32x32 and the dataset has around 7500 training images and 3740 test images, whereas for the second input the image size is 128x128 obtained using the transfer learning approach from the Enhanced

SRGAN (ESRGAN) model from Xinntao, Researcher at Tencent ARC Lab, (Applied Research Center). The dataset again has 7500 high-resolution training images and 3750 test images.

CNN classification model was trained separately for both data sets using the same hyper-parameters. After multiple iterations, the number of epochs was kept equal to 10. Other hyper-parameters such as the number of convolutional layers, max-pooling layers, number of filters per layer, flattening layer, and the dense layer were also kept the same to precisely compare the accuracy of the classifier with raw input images vs super-resolution input images (obtained from ERSGAN-CNN)

## 4 Results

For the final result, the entire data set of the 50,000 images was used. The entire dataset was divided into two sets, a training batch of 40,000 images and a test batch of 10,000 images. Two different CNN classification models were made, first for the model with input of raw images and second for the model with super-resolution images input.

Around 40,000 images are used for training the model while 10,000 images are used as the test set for both the classification model. The results were obtained after training the CNN classification model with 10 labels for 6.5 hours.

Various different iterations were made to decide the number of epochs. With a higher number of epochs (more than 10), the CNN classification model tends to overfit the data (the super-resolution images as input) with more than 92 percent accuracy on the training batch and less than 60 percent accuracy on the test batch. So after a few iterations, the number of epochs was obtained to eliminate the chances of overfitting the given data set of Super-resolution images.

CNN classification model was trained separately for both data sets using the same hyper-parameters. After multiple iterations, the number of epochs was kept equal to 4. Other hyper-parameters such as the number of convolutional layers, max-pooling layers, number of filters per layer, flattening layer, and the dense layer were also kept the same to precisely compare the accuracy of the classifier with raw input images vs super-resolution input images (obtained from ERSGAN-CNN)

| Content | Classifier on Raw Images | Classifier on SR Images |
|---|---|---|
| Training size | 40,000 | 40,000 |
| Test Size | 10,000 | 10,000 |
| Accuracy on Training Data | 63.6% | 76.27% |
| Accuracy on Test Data | 65.07% | 66.24% |

## 5 Conclusion

After comparing the results of both the CNN classification model (one with the input of Raw images and one with the input of super-resolution images obtained from ESRGAN), it can be seen that with a good amount of data, and training the CNN classification models while keeping the hyperparameters same, the CNN classification model with input with super-resolution images tend to have higher accuracy on the test batch (while keeping a number of epochs to 4, to avoid the overfitting of the training batch) as compared to the CNN classification model with raw images input. Although the difference in accuracy is 1.17 percent, it totally depends on the number of epochs used to train the data set.

## 6 References

[1] Eirikur Agustsson and Radu Timofte. Ntire 2017 challenge on single image super-resolution: Dataset and study. In CVPRW, 2017. 6

[2] Sefi Bell-Kligler, Assaf Shocher, and Michal Irani. Blind super-resolution kernel estimation using an internal-gan. In NeurIPS, 2019. 1, 2

[3] Yochai Blau, Roey Mechrez, Radu Timofte, TomerMichaeli, and Lihi Zelnik-Manor. The 2018 pirm challenge on perceptual image super-resolution. In ECCVW, 2018. 6, 12

[4] Yochai Blau and Tomer Michaeli. The perception-distortion tradeoff. In CVPR, 2018. 6

[5]. Chao Dong, Chen Change Loy, Member, IEEE, Kaiming He, Member, IEEE, and Xiaoou Tang: 'Image Super-Resolution Using Deep Convolutional Networks'

[6] Xin, M., Wang, Y. Research on image classification model based on deep convolution neural network. J Image Video Proc. 2019, 40 (2019). https://doi.org/10.1186/s13640-019-0417-8

[7] Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.

[8] ESRGAN (ESRGAN) model from Xinntao, Researcher at Tencent ARC Lab, (Applied Research Center)

[9] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Learning a deep convolutional network for image super-resolution. In ECCV, 2014. 1, 2

[10] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. IEEE TPAMI, 38(2):295–307, 2016. 1, 2

[11] Michael Elad and Arie Feuer. Restoration of a single superresolution image from several blurred, noisy, and undersampled measured images. IEEE transactions on image processing, 6(12):1646–1658, 1997. 1, 2, 3

## 7 Appendix

The python script of the entire CNN classification model is as follows:

```
# %%
import os
from matplotlib.image import imread
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import cifar10
from keras.utils import np_utils
from matplotlib import pyplot as plt
import numpy as np
from PIL import Image
import keras
import sys

# %% [markdown]
# #### Converting the SR Image Data to RGB values for the CNN Classification model

# %%
file_list=os.listdir(r"D:/TAMU/ECEN 649, Pattern Recognition/Final Project/Super Resolution
len(file_list)
file_list[2]
SR_data = np.full((len(file_list),128,128,3), 0.0)
for i in range(len(file_list)):
    SR_data[i] = imread('''D:/TAMU/ECEN 649, Pattern Recognition/Final Project/Super Resolut
SR_data.shape

# %% [markdown]
# Reading the Cifar-10 data from Keras
#

# %%
(X_train, y_train), (X_test,y_test) = cifar10.load_data()
sub1 = ""
sub2 = "_rlt.png"
index = []
# getting index of substrings
for i in range(len(file_list)):
    idx1 = file_list[i].index(sub1)
    idx2 = file_list[i].index(sub2)
    res = ''
    # getting elements in between
    for idx in range(idx1 + len(sub1), idx2):
        res = res + file_list[i][idx]
    index.append(res)
int_index = [int(index[i]) for i in range(len(index))]
y = np.zeros((50000,1))
for i,val in enumerate(int_index):
    y[i] = y_train[val-1][0]
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import cifar10

# %% [markdown]
#
# #### Loading the dataset and seperating the training and test batch
#

# %%
# setting class names
class_names=['airplane','automobile','bird','cat','deer','dog','frog','horse','ship','truck'
```



Figure 5:

```
import gc
gc.collect()
X_test_SR = SR_data[40000:50000]

X_train_SR = SR_data[0:40000]
y_test_SR = y[40000:50000]
y_train_SR = y[0:40000]

# %% [markdown]
# #### Building a Convolutional Neural Network

# %%
cifar10_model=tf.keras.models.Sequential()
# First Layer
cifar10_model.add(tf.keras.layers.Conv2D(filters=32,kernel_size=3,padding="same", activatior
# Second Layer
cifar10_model.add(tf.keras.layers.Conv2D(filters=32,kernel_size=3,padding="same", activatior
# Max Pooling Layer
cifar10_model.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2,padding='valid'))
# Third Layer
cifar10_model.add(tf.keras.layers.Conv2D(filters=64,kernel_size=3,padding="same", activatior
# Fourth Layer
cifar10_model.add(tf.keras.layers.Conv2D(filters=64,kernel_size=3,padding="same", activatior
# Max Pooling Layer
cifar10_model.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2,padding='valid'))
# Flattening Layer
cifar10_model.add(tf.keras.layers.Flatten())
# Droput Layer
cifar10_model.add(tf.keras.layers.Dropout(0.5,noise_shape=None,seed=None))
# Adding the first fully connected layer
cifar10_model.add(tf.keras.layers.Dense(units=128,activation='relu'))
# Output Layer
cifar10_model.add(tf.keras.layers.Dense(units=10,activation='softmax'))
cifar10_model.summary()

# %% [markdown]
# #### Compiling the Model

# %%
count_0 = 0
count_1 = 0
count_2 = 0

count_3 = 0
count_4 = 0

count_5 = 0
count_6 = 0

count_7 = 0

count_8 = 0
count_9 = 0



for i in  range(y.shape[0]):
    if y[i][0] == 0:
        count_0 += 1
    elif y[i][0] == 1:
```



Figure 6:

```
        count_1 += 1
    elif y[i][0] == 2:
        count_2 += 1
    elif y[i][0] == 3:
        count_3 += 1
    elif y[i][0] == 4:
        count_4 += 1
    elif y[i][0] == 5:
        count_5 += 1
    elif y[i][0] == 6:
        count_6 += 1
    elif y[i][0] == 7:
        count_7 += 1
    elif y[i][0] == 8:
        count_8 += 1
    elif y[i][0] == 9:
        count_9 += 1

cifar10_model.compile(loss="sparse_categorical_crossentropy", optimizer="Adam", metrics=["sp

# %% [markdown]
# #### Training the Model

# %%
cifar10_model.fit(X_train_SR,y_train_SR,epochs= 4)

# %%
test_loss, test_accuracy = cifar10_model.evaluate(X_test_SR, y_test_SR)

# %% [markdown]
#
.....................................................................................

# %%
print("Test accuracy SR Model: {}".format(test_accuracy))

# %% [markdown]
# ### Creating and training the same model with raw images data

# %%
CIFAR_Raw_data = np.full((50000,32,32,3), 0)
int_index[1]
for i,val in enumerate(int_index):
    CIFAR_Raw_data[i] = X_train[val-1]
 plt.imshow(CIFAR_Raw_data[39999])


# %% [markdown]
# #### Building a Convolutional Neural Network for Raw CIFAR 10 Dataset

# %%
cifar10_raw_model=tf.keras.models.Sequential()
# First Layer
cifar10_raw_model.add(tf.keras.layers.Conv2D(filters=32,kernel_size=3,padding="same", activa
# Second Layer
cifar10_raw_model.add(tf.keras.layers.Conv2D(filters=32,kernel_size=3,padding="same", activa
# Max Pooling Layer
cifar10_raw_model.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2,padding='valid'))
# Third Layer
cifar10_raw_model.add(tf.keras.layers.Conv2D(filters=64,kernel_size=3,padding="same", activa
```



Figure 7:

```
# Fourth Layer
cifar10_raw_model.add(tf.keras.layers.Conv2D(filters=64,kernel_size=3,padding="same", activa
# Max Pooling Layer
cifar10_raw_model.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2,padding='valid'))
# Flattening Layer
cifar10_raw_model.add(tf.keras.layers.Flatten())
# Droput Layer
cifar10_raw_model.add(tf.keras.layers.Dropout(0.5,noise_shape=None,seed=None))
# Adding the first fully connected layer
cifar10_raw_model.add(tf.keras.layers.Dense(units=128,activation='relu'))
# Output Layer
cifar10_raw_model.add(tf.keras.layers.Dense(units=10,activation='softmax'))
cifar10_raw_model.summary()

# %% [markdown]
# #### Compiling the Model

# %%
cifar10_raw_model.compile(loss="sparse_categorical_crossentropy", optimizer="Adam", metrics=
#### Training the Model
X_test_raw = CIFAR_Raw_data[40000:50000]


X_train_raw = CIFAR_Raw_data[0:40000]

y_test_raw = y[40000:50000]

y_train_raw = y[0:40000]

# %%
cifar10_raw_model.fit(X_train_raw,y_train_raw,epochs= 4)

# %%
model.load_weights(checkpoint_path)

# %%
test_loss, test_accuracy = cifar10_raw_model.evaluate(X_test_raw, y_test_raw)

# %%
print("Test accuracy CIFAR raw Model: {}".format(test_accuracy))
```



Figure 8: