Neural Style Transfer

2020-21

Shridatha Hegde	01fe18bec171
Shreyas Dhotrad	01fe18bec170
Vishwajitt Kamat	01fe18bec212
Vishal S Teli	01fe18bec209

ABSTRACT

An embedded intelligent system is a machine that has an embedded, Internet-connected computer which can gather and analyze data and communicate with other systems. Additionally, these systems have the ability to learn from experience, security, connectivity, sophisticated AI-based software systems, and can adapt themselves according to current data.

In this report, MobileNet V3 which is a next generation of MobileNet architecture, is trained on ImageNet dataset for developing an Android application for applying style transfer on given content image. This aims to enhance the execution time running on edge device by making the model light.

CONTENTS

INTRODUCTION

Style transfer is the technique which allows us to apply the style of one image to the content of another image. The principle is to generate an image with the same "content" as a base image, but with the "style" of a different picture (typically artistic). This is achieved through the optimization of a loss function that has 3 components: "style loss", "content loss", and "total variation loss".

LITERATURE SURVEY

Source- *Image Style Transfer Using Convolutional Neural Networks*

This paper produces a Neural Algorithm of Artistic Style that can separate and recombine the image content and style of natural images, which produces a new image of high perceptual quality that combine the content of an arbitrary photograph with the appearance of numerous well known artworks.

Source- *Improving Style Transfer With Calibrated Metrics*

Style transfer methods produce a transferred image which is a rendering of a content image in the manner of a style image. The report seeks to understand how to improve style transfer. To do so requires quantitative evaluation procedures, but the current evaluation is qualitative, mostly involving user studies. This is described by a novel quantitative evaluation procedure. The procedure relies on two statistics: the Effectiveness (E) statistic measures the extent that a given style has been transferred to the target, and the Coherence (C) statistic measures the extent to which the original image's content is preserved.

MODEL ARCHITECTURE

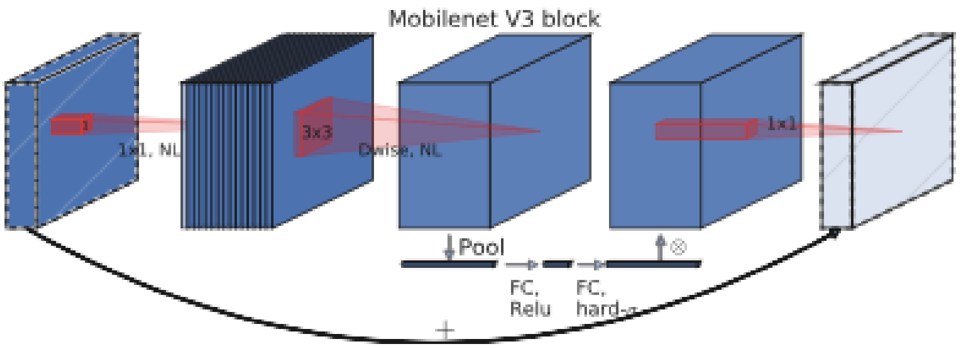
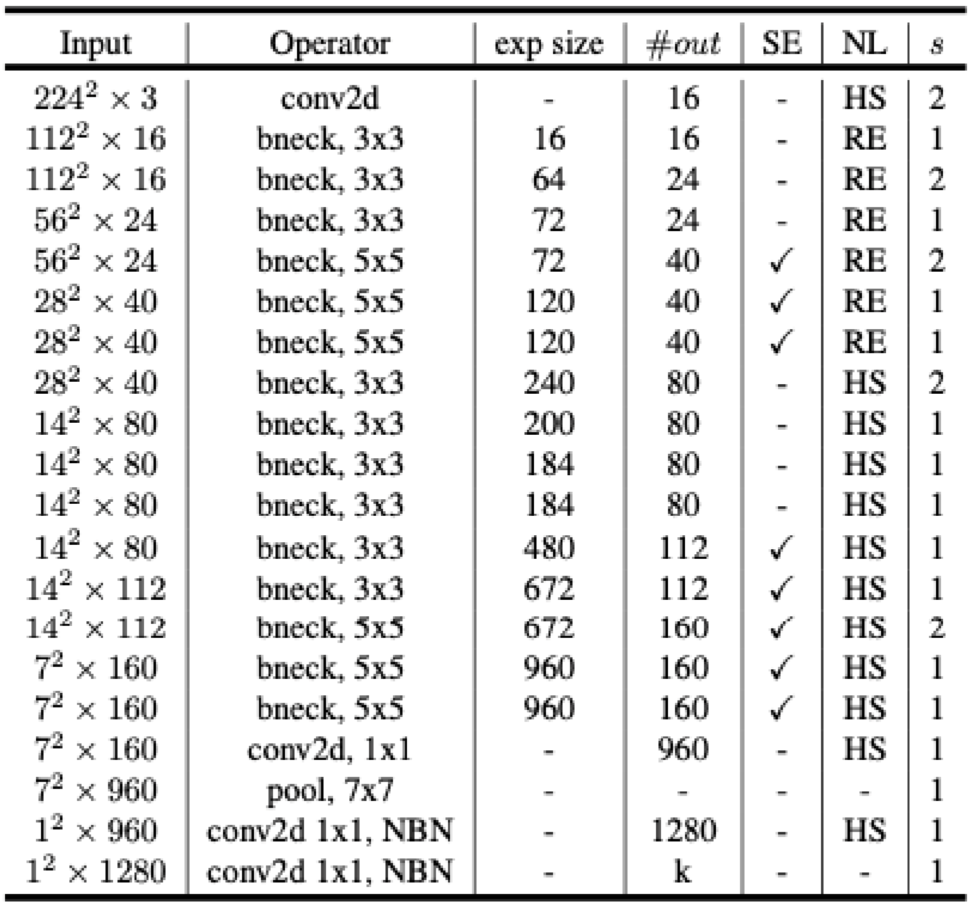


Figure 1 Block Diagram

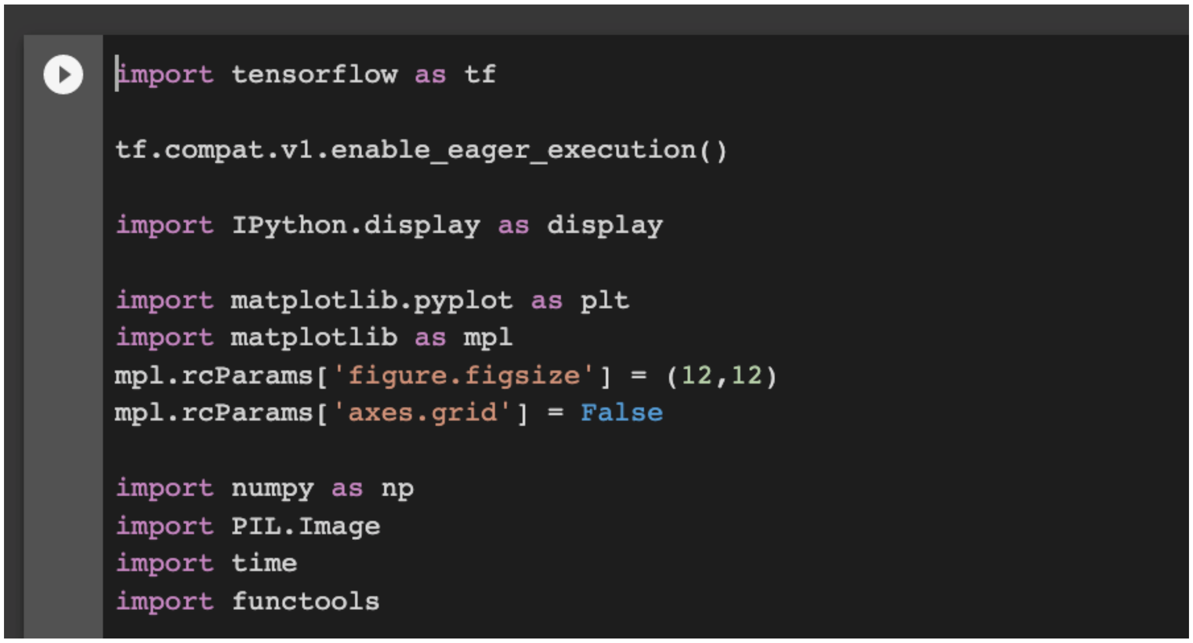


Input	Operator	exp size	$\#out$	SE	NL	s
$224^2 \times 3$	conv2d	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	-	RE	1
$112^2 \times 16$	bneck, 3x3	64	24	-	RE	2
$56^2 \times 24$	bneck, 3x3	72	24	-	RE	1
$56^2 \times 24$	bneck, 5x5	72	40	✓	RE	2
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 3x3	240	80	-	HS	2
$14^2 \times 80$	bneck, 3x3	200	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	480	112	✓	HS	1
$14^2 \times 112$	bneck, 3x3	672	112	✓	HS	1
$14^2 \times 112$	bneck, 5x5	672	160	✓	HS	2
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	conv2d, 1x1	-	960	-	HS	1
$7^2 \times 960$	pool, 7x7	-	-	-	-	1
$1^2 \times 960$	conv2d 1x1, NBN	-	1280	-	HS	1
$1^2 \times 1280$	conv2d 1x1, NBN	-	k	-	-	1

Figure 2 MobileNet V3 Layers

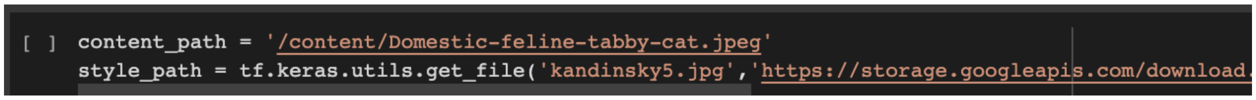
IMPLEMENTATION

I. Python



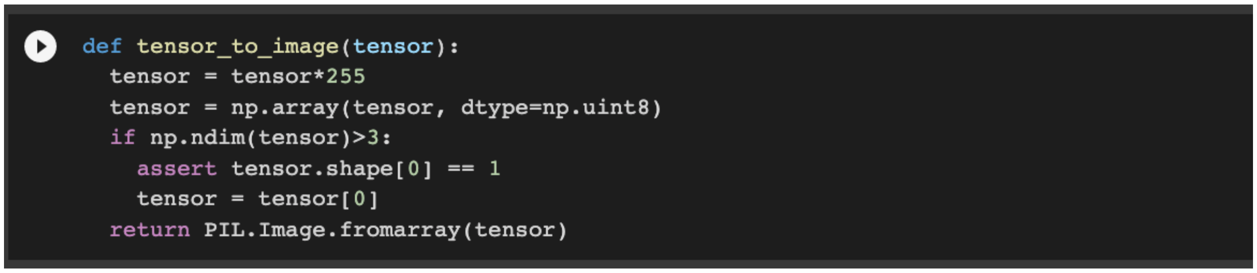
```
import tensorflow as tf

tf.compat.v1.enable_eager_execution()

import IPython.display as display

import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.rcParams['figure.figsize'] = (12,12)
mpl.rcParams['axes.grid'] = False

import numpy as np
import PIL.Image
import time
import functools
```

II. i)Dependencies

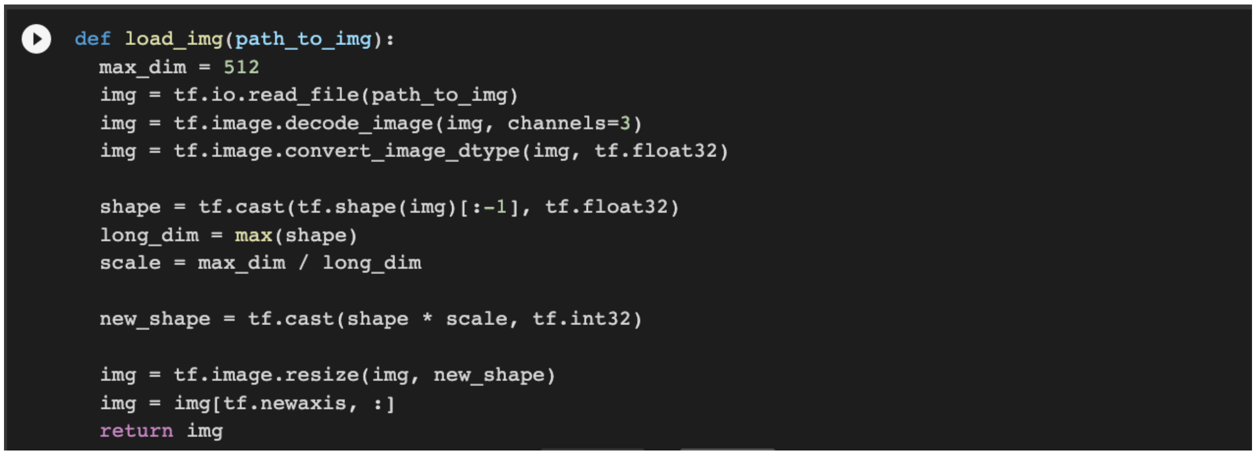


```
[ ] content_path = '/content/Domestic-feline-tabby-cat.jpeg'
    style_path = tf.keras.utils.get_file('kandinsky5.jpg','https://storage.googleapis.com/download.
```

ii) Image extraction and Pre-Processing

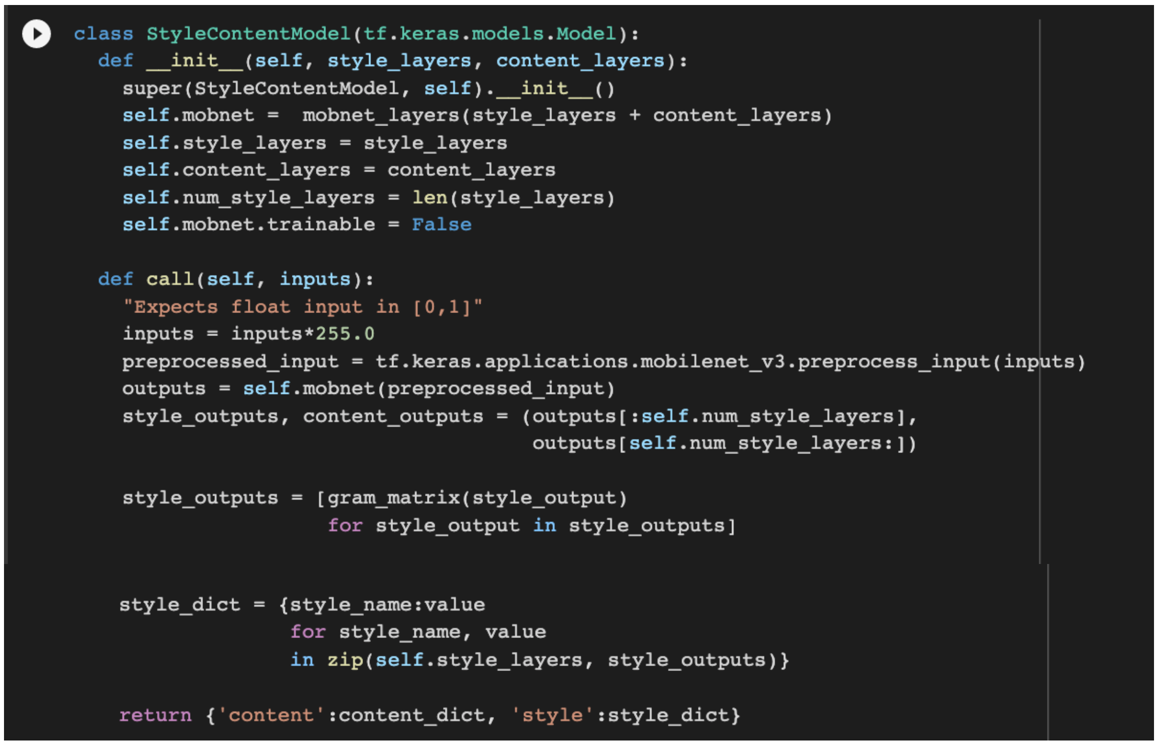


```python
def tensor_to_image(tensor):
  tensor = tensor*255
  tensor = np.array(tensor, dtype=np.uint8)
  if np.ndim(tensor)>3:
    assert tensor.shape[0] == 1
    tensor = tensor[0]
  return PIL.Image.fromarray(tensor)
```



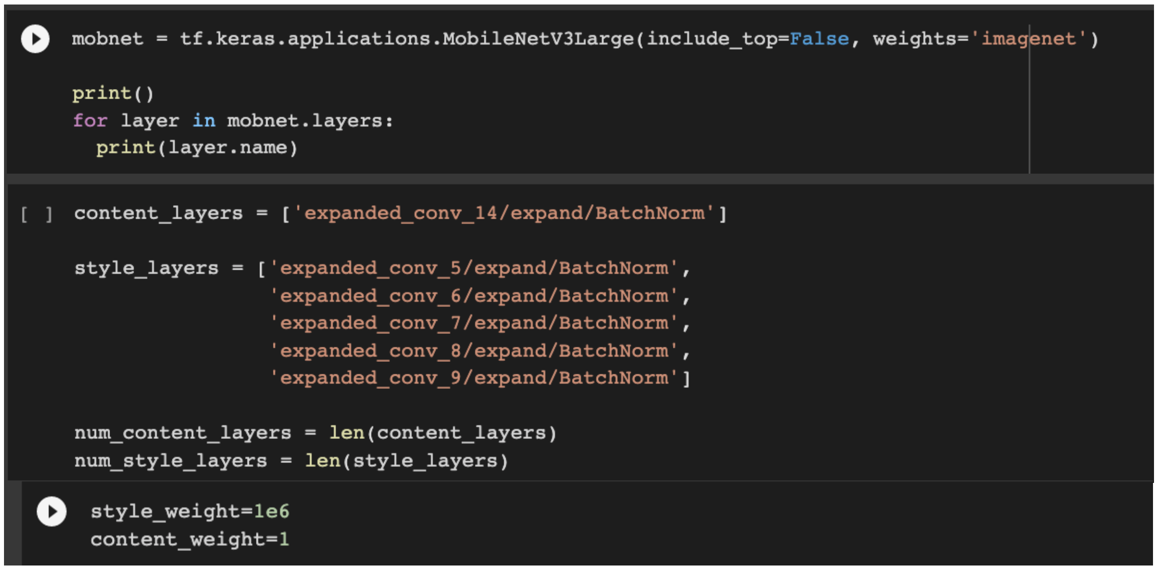
```python
def load_img(path_to_img):
  max_dim = 512
  img = tf.io.read_file(path_to_img)
  img = tf.image.decode_image(img, channels=3)
  img = tf.image.convert_image_dtype(img, tf.float32)

  shape = tf.cast(tf.shape(img)[:-1], tf.float32)
  long_dim = max(shape)
  scale = max_dim / long_dim

  new_shape = tf.cast(shape * scale, tf.int32)

  img = tf.image.resize(img, new_shape)
  img = img[tf.newaxis, :]
  return img
```

iii) Model



```python
class StyleContentModel(tf.keras.models.Model):
  def __init__(self, style_layers, content_layers):
    super(StyleContentModel, self).__init__()
    self.mobnet =  mobnet_layers(style_layers + content_layers)
    self.style_layers = style_layers
    self.content_layers = content_layers
    self.num_style_layers = len(style_layers)
    self.mobnet.trainable = False

  def call(self, inputs):
    "Expects float input in [0,1]"
    inputs = inputs*255.0
    preprocessed_input = tf.keras.applications.mobilenet_v3.preprocess_input(inputs)
    outputs = self.mobnet(preprocessed_input)
    style_outputs, content_outputs = (outputs[:self.num_style_layers],
                                      outputs[self.num_style_layers:])

    style_outputs = [gram_matrix(style_output)
                     for style_output in style_outputs]


    style_dict = {style_name:value
                  for style_name, value
                  in zip(self.style_layers, style_outputs)}

    return {'content':content_dict, 'style':style_dict}
```

iv)Fine-Tuning Hyper Parameter

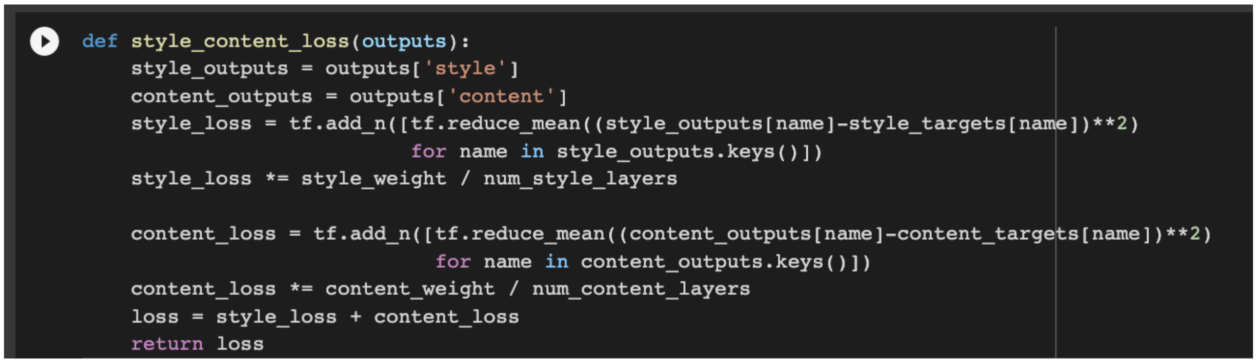


```
mobnet = tf.keras.applications.MobileNetV3Large(include_top=False, weights='imagenet')

print()
for layer in mobnet.layers:
  print(layer.name)
```

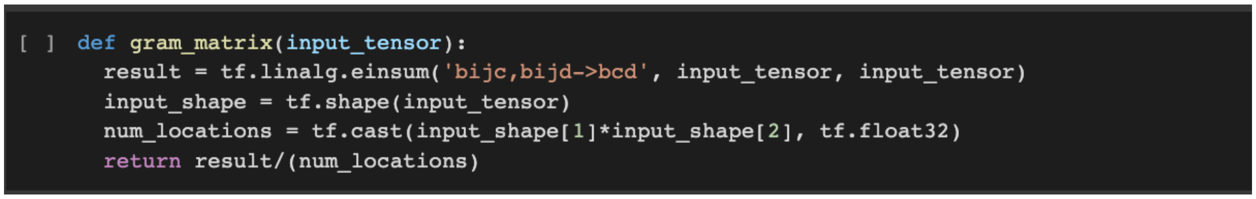
```
[ ] content_layers = ['expanded_conv_14/expand/BatchNorm']

    style_layers = ['expanded_conv_5/expand/BatchNorm',
                    'expanded_conv_6/expand/BatchNorm',
                    'expanded_conv_7/expand/BatchNorm',
                    'expanded_conv_8/expand/BatchNorm',
                    'expanded_conv_9/expand/BatchNorm']

    num_content_layers = len(content_layers)
    num_style_layers = len(style_layers)
```

```
style_weight=1e6
content_weight=1
```

v) Content and Style Loss

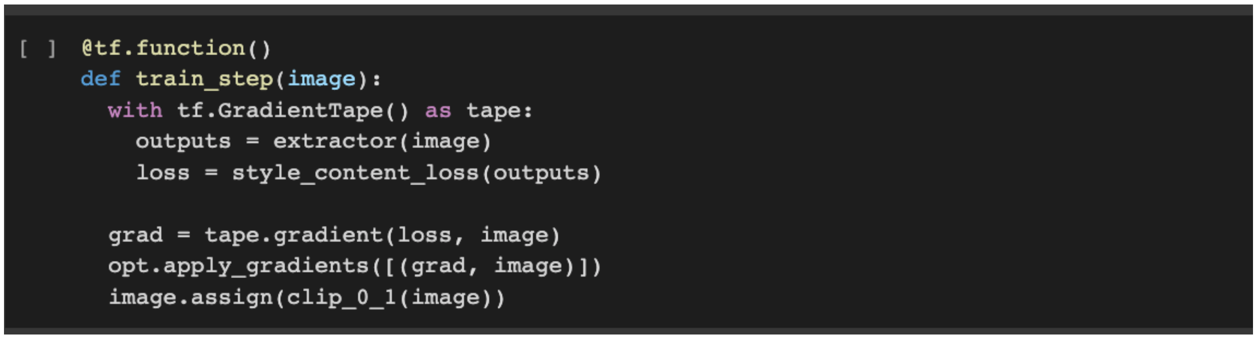


```
def style_content_loss(outputs):
    style_outputs = outputs['style']
    content_outputs = outputs['content']
    style_loss = tf.add_n([tf.reduce_mean((style_outputs[name]-style_targets[name])**2)
                           for name in style_outputs.keys()])
    style_loss *= style_weight / num_style_layers

    content_loss = tf.add_n([tf.reduce_mean((content_outputs[name]-content_targets[name])**2)
                             for name in content_outputs.keys()])
    content_loss *= content_weight / num_content_layers
    loss = style_loss + content_loss
    return loss
```

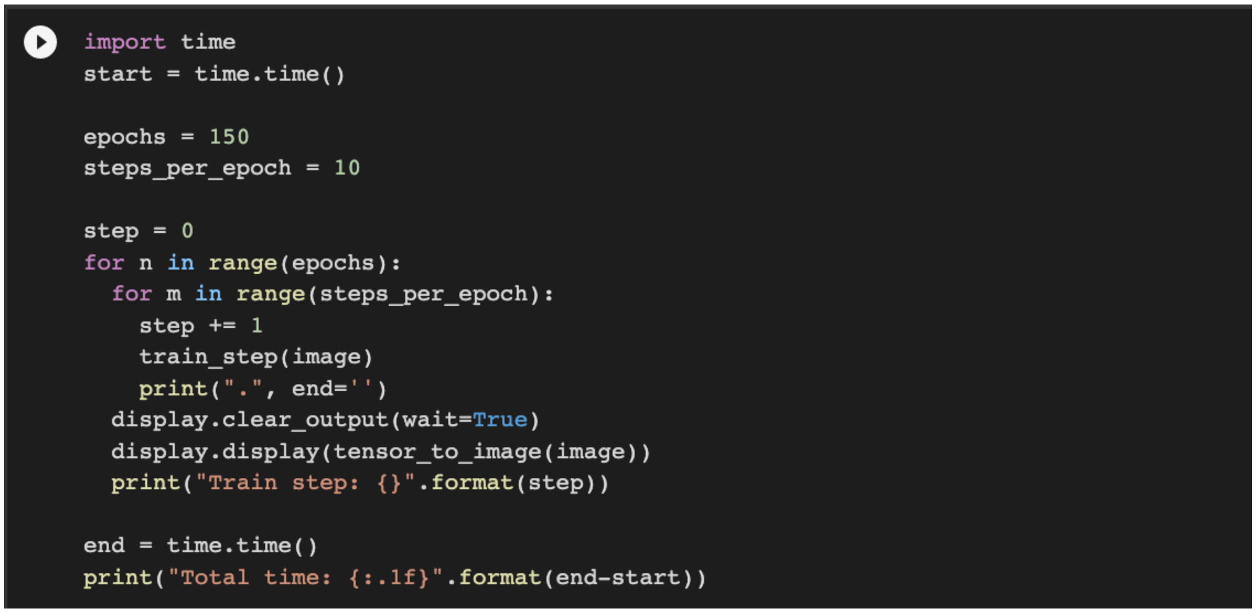


```
[ ] def gram_matrix(input_tensor):
      result = tf.linalg.einsum('bijc,bijd->bcd', input_tensor, input_tensor)
      input_shape = tf.shape(input_tensor)
      num_locations = tf.cast(input_shape[1]*input_shape[2], tf.float32)
      return result/(num_locations)
```

vi) Training



```
[ ]  @tf.function()
     def train_step(image):
       with tf.GradientTape() as tape:
         outputs = extractor(image)
         loss = style_content_loss(outputs)

       grad = tape.gradient(loss, image)
       opt.apply_gradients([(grad, image)])
       image.assign(clip_0_1(image))
```



```
import time
start = time.time()

epochs = 150
steps_per_epoch = 10

step = 0
for n in range(epochs):
  for m in range(steps_per_epoch):
    step += 1
    train_step(image)
    print(".", end='')
  display.clear_output(wait=True)
  display.display(tensor_to_image(image))
  print("Train step: {}".format(step))

end = time.time()
print("Total time: {:.1f}".format(end-start))
```

III. Android app development

TensorFlow Lite is a popular open-source deep learning framework, which can be used on-device mobile inference. It enables the bulk of ML processing to take place on the device, utilizing less intensive models. Such models run faster, give potential privacy enhancement, consume less power. On Android, TensorFlow Lite leverages specialist mobile accelerators through the Neural Network API, providing better performance while minimizing power usage that is expected when training datasets.

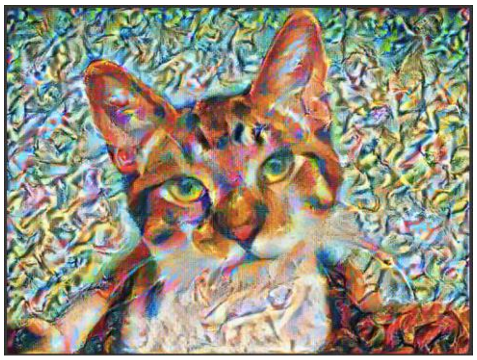
After the model is trained, we converted the Model to the TensorFlow Lite version. TF lite model is a special format model efficient in terms of accuracy and also is a light-weight version that will occupy less space, these features make TF Lite models the right fit to work on Mobile and Embedded Devices

Models at Edge should also have low latency to run inferences. Light-weight and low latency model can be achieved by reducing the amount of computation required to predict. Optimization reduces the size of the model or improves the latency.

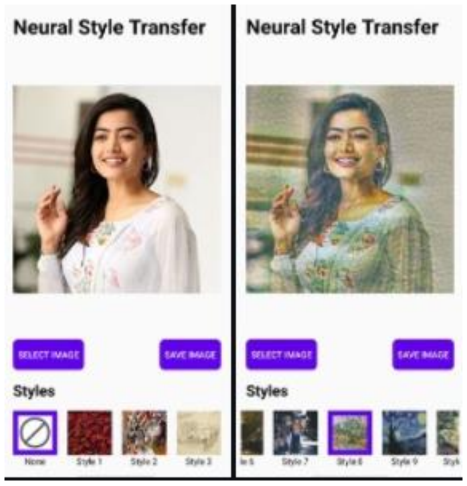
Quantization can be applied to weight and activations. Both weights and activations can be quantized by converting to an integer**,** and this will give low latency, smaller size, and reduced power consumption.

RESULTS AND CONCLUSION

I. Python output



II. Android Application



The Neural Style Transfer application using MobileNet V3 architecture is implemented on edge device with the user interface developed through Android Studio, and results are shown by building and running APK file on android device.

REFERENCES

[1] N. Ashikhmin. *Fast texture transfer*. *IEEE Computer Graphics and Applications*, 23(4):38–43, July 2003

[2] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. *ImageNet Large Scale Visual Recognition Challenge*. arXiv:1409.0575 [cs], Sept. 2014. arXiv: 1409.0575.

[3] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Vijay Vasudevan. *Searching for MobileNetV3*.arXiv:1905.02244[cs],May.2019.arXiv: 1905.02244