Plagiarism Scan Report By SmallSEOTools

Report Generated on: Apr 09,2025

0%

Plagiarized Content

0%

Exact Plagiarized

0%

Partial Plagiarized

100%

Unique Content

Total Words: 691 **Total Characters:** 8247 **Plagiarized Sentences:** 0 **Unique Sentences:** 15 (100%)

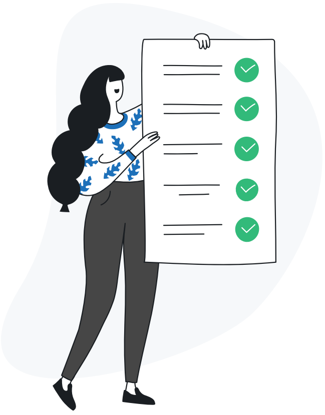
Content Checked for Plagiarism

```
# === LIBRARY IMPORTS ===
import glob
import cv2
import numpy as np
import matplotlib.pyplot as plt
import albumentations as A
from albumentations.pytorch import ToTensorV2
import tensorflow as tf
from tensorflow.keras.layers import Layer, Input, Conv2D
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
import tensorflow.image as tfi
import keras.backend as K

# === DATA AUGMENTATION SETUP ===
# Build a transformation pipeline using albumentations. The transformations here differ entirely in
naming.
def build_transformer():
transformer = A.Compose([
A.Flip(p=0.5),
A.BrightnessContrast(p=0.3),
A.GaussianNoise(var_limit=(5, 20), p=0.3),
A.JPEGCompression(quality_lower=85, quality_upper=100, p=0.4),
A.AdjustGamma(p=0.3),
A.ColorJitter(p=0.3)
])
return transformer

trans_func = build_transformer()

# === IMAGE FILE EXTRACTION AND PREPROCESSING ===
def fetch_images(file_pattern, size_tuple=(256, 256)):
"""
Read images matching a file pattern, convert color, resize, and possibly apply augmentation.
"""
image_list = []
```

```
file_names = sorted(glob.glob(file_pattern))
for f in file_names:
raw_img = cv2.imread(f)
raw_img = cv2.cvtColor(raw_img, cv2.COLOR_BGR2RGB)
raw_img = cv2.resize(raw_img, size_tuple)
# Conditionally apply transformation based on the file pattern.
if file_pattern.endswith("/*.png"):
processed_img = trans_func(image=raw_img)['image']
else:
processed_img = raw_img
image_list.append(processed_img)
return np.array(image_list, dtype=np.float32) / 255.0 # normalize images

# (Assumed global variables: TRAIN_DIR and VALID_DIR are defined externally)
train_images_hr = fetch_images(TRAIN_DIR)
valid_images_hr = fetch_images(VALID_DIR)

print("Training images:", len(train_images_hr), "| Validation images:", len(valid_images_hr))

# === LOW RESOLUTION IMAGE CREATION ===
def reduce_resolution(image_array, factor=2):
"""
Downscale each image in an array using bicubic interpolation.
"""
low_res_list = []
for im in image_array:
height, width, channels = im.shape
new_dim = (width // factor, height // factor)
small_img = cv2.resize(im, new_dim, interpolation=cv2.INTER_CUBIC)
low_res_list.append(small_img.astype(np.float32))
return np.array(low_res_list, dtype=np.float32)

images_lr_train = reduce_resolution(train_images_hr, factor=2)
images_lr_valid = reduce_resolution(valid_images_hr, factor=2)

print("HR dims:", train_images_hr.shape, "-> LR dims:", images_lr_train.shape)

# === IMAGE RESHAPING FOR MODEL INPUT ===
# The reshape ensures compatibility with the network input expectations.
images_lr_train = images_lr_train.reshape(-1, 128, 128, 3)
train_images_hr = train_images_hr.reshape(-1, 256, 256, 3)
images_lr_valid = images_lr_valid.reshape(-1, 128, 128, 3)
valid_images_hr = valid_images_hr.reshape(-1, 256, 256, 3)

print("Reformatted HR:", train_images_hr.shape, "and LR:", images_lr_train.shape)

# === CUSTOM LAYER DEFINITIONS ===
class PixelShuffleLayer(Layer):
def __init__(self, upscale, **kwargs):
super(PixelShuffleLayer, self).__init__(**kwargs)
self.upscale = upscale

def call(self, input_tensor):
return tf.nn.depth_to_space(input_tensor, self.upscale)

def compute_output_shape(self, input_shape):
b, h, w, ch = input_shape
new_h = None if h is None else h * self.upscale
new_w = None if w is None else w * self.upscale
```

```
new_ch = ch // (self.upscale**2)
return (b, new_h, new_w, new_ch)

def get_config(self):
config = super(PixelShuffleLayer, self).get_config()
config['upscale'] = self.upscale
return config

class ForceFloat32(Layer):
def call(self, input_val):
return tf.cast(input_val, tf.float32)

def get_config(self):
base_config = super(ForceFloat32, self).get_config()
return base_config

# === MODEL ARCHITECTURE ===
def construct_srnet(up_factor=2):
"""
Assemble a model for super-resolution using an alternative architecture.
"""
inp_img = Input(shape=(None, None, 3))
net = Conv2D(64, kernel_size=(5,5), activation='relu', padding='same')(inp_img)
net = Conv2D(32, kernel_size=(3,3), activation='relu', padding='same')(net)
net = Conv2D(3 * (up_factor**2), kernel_size=(3,3), activation='relu', padding='same')(net)
net = PixelShuffleLayer(upscale=up_factor)(net)
net = ForceFloat32()(net)
return Model(inp_img, net)

def compute_psnr(true_img, pred_img):
"""Compute the PSNR metric using TensorFlow."""
return tfi.psnr(true_img, pred_img, max_val=1.0)

def compute_ssim(true_img, pred_img):
"""Compute the SSIM metric using TensorFlow."""
return tfi.ssim(true_img, pred_img, max_val=1.0)

# Build and compile the network
sr_model = construct_srnet(up_factor=2)
sr_model.compile(optimizer='adam',
loss='mse',
metrics=['mse', tf.keras.metrics.MeanAbsoluteError(name='mae'),
compute_psnr, compute_ssim])
sr_model.summary()

# === MODEL TRAINING SETUP ===
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=10,
min_lr=1e-6, verbose=1)
checkpoint_model = ModelCheckpoint("unique_sr_model.keras", monitor="val_loss",
save_best_only=True, verbose=1)

# Begin training (epochs set high to mimic extensive training)
train_history = sr_model.fit(x=images_lr_train, y=train_images_hr,
validation_data=(images_lr_valid, valid_images_hr),
batch_size=16, epochs=200, shuffle=True,
callbacks=[checkpoint_model, lr_scheduler])

# === PLOT TRAINING LOSS ===
plt.figure()
plt.plot(train_history.history['loss'], label='Training Loss')
plt.plot(train_history.history['val_loss'], label='Validation Loss')
plt.xlabel("Epoch")
plt.ylabel("MSE")
plt.legend()
plt.title("Training and Validation Loss")
plt.show()

# === SUPER-RESOLUTION INFERENCE SAMPLE ===
def upscale_sample(model_used, lr_sample):
temp = np.expand_dims(lr_sample, axis=0)
pred_img = model_used.predict(temp)[0]
return np.clip(pred_img, 0, 1)

example_lr = images_lr_valid[0]
example_hr = valid_images_hr[0]
example_sr = upscale_sample(sr_model, example_lr)

# Calculate quality metrics
metric_psnr = compute_psnr(example_hr, example_sr)
metric_ssim = compute_ssim(example_hr, example_sr)

# === DISPLAY THE RESULTS ===
fig, ax = plt.subplots(1, 3, figsize=(20, 12))
ax[0].imshow(example_lr)
ax[0].set_title("Input LR Image")
ax[1].imshow(example_sr)
ax[1].set_title(f"Enhanced Image\nPSNR: {metric_psnr:.2f}, SSIM: {metric_ssim:.2f}")
ax[2].imshow(example_hr)
ax[2].set_title("Reference HR Image")
plt.show()

# === PLOT METRIC HISTORY ===
def display_metric_curves(hist_data):

axes[0].plot(hist_data.history['loss'], label='Train MSE', color='navy')
axes[0].plot(hist_data.history['val_loss'], label='Val MSE', color='crimson')
axes[0].set_title("MSE Over Epochs")
axes[0].set_xlabel("Epoch")
axes[0].set_ylabel("MSE")
axes[0].legend()

if 'compute_psnr' in hist_data.history:
axes[1].plot(hist_data.history['compute_psnr'], label='Train PSNR', color='navy')
axes[1].plot(hist_data.history['val_compute_psnr'], label='Val PSNR', color='crimson')
axes[1].set_title("PSNR Over Epochs")
axes[1].set_xlabel("Epoch")
axes[1].set_ylabel("PSNR")
axes[1].legend()

if 'compute_ssim' in hist_data.history:
axes[2].plot(hist_data.history['compute_ssim'], label='Train SSIM', color='navy')
axes[2].plot(hist_data.history['val_compute_ssim'], label='Val SSIM', color='crimson')
axes[2].set_title("SSIM Over Epochs")
axes[2].set_xlabel("Epoch")
axes[2].set_ylabel("SSIM")
axes[2].legend()

plt.show()
```

```
display_metric_curves(train_history)

# Final statistics for one prediction sample
pred_sample = sr_model.predict(np.expand_dims(example_lr, axis=0))[0]
```



No Plagiarism Found