Learning Superpixel Mean-Color Representations via Deep Regression

Om Singh 210686 | Chirag Garg 210288 | Abhishek Pandey 210042 | Arpit Raj 210192

Abstract

Superpixel algorithms, such as SLIC, provide perceptually meaningful image region groupings valuable for various computer vision tasks. These methods are typically iterative and computationally distinct from deep learning approaches. This work investigates the potential of deep neural networks to learn the mean-color transformation derived from the SLIC superpixel algorithm's output. Specifically, we train a U-Net architecture to perform image-to-image regression, mapping an input RGB image to a target image representing the mean color of superpixels generated by SLIC. We leverage the 'tensorflow_datasets' library to access the 'tf_flowers' dataset and develop a robust data pipeline using 'tf.data' that integrates 'scikit-image''s SLIC implementation via 'tf.py_function'. This involved addressing challenges in handling data type conversions between TensorFlow tensors and NumPy arrays within the pipeline. The U-Net model, trained using a pixel-wise Mean Absolute Error loss, successfully learns to generate outputs that visually approximate the SLIC mean-color representations. Our results demonstrate the feasibility of using deep regression to learn complex, spatially-aware image transformations derived from traditional algorithms, highlighting the potential of this approach and the practical techniques required for pipeline integration.

1. Introduction

Superpixels offer a valuable mid-level image representation by grouping spatially adjacent pixels with similar features (e.g., color, texture) into perceptually meaningful regions. Algorithms like SLIC (Simple Linear Iterative Clustering) are popular due to their efficiency and ability to generate compact, uniform superpixels, simplifying subsequent processing steps. These representations reduce computational complexity for downstream tasks like object detection, segmentation, and tracking by operating on hundreds of superpixels instead of millions of pixels.

Traditional superpixel algorithms are iterative, optimization-based methods operating independently of task-specific deep learning models that have come to dominate computer vision since the advent of deep convolutional networks. In contrast, deep learning, particularly convolutional neural networks (CNNs), excels at learning hierarchical features directly from data for end-to-end tasks, including sophisticated image generation and translation. An intriguing question arises: can a deep neural network learn to directly generate the mean-color representation derived from a traditional superpixel algorithm like SLIC, effectively capturing its complex spatial grouping logic within a feed-forward architecture?

This paper explores this question by framing the generation of a SLIC-based representation as an image-to-image regression problem. We aim to train a deep learning model, specifically the U-Net architecture, widely adopted for image segmentation and translation tasks, to take an RGB image as input and predict an output image where each pixel's color corresponds to the mean color of the SLIC superpixel it belongs to.

Our contributions are:

1. We demonstrate that a standard U-Net can be trained via supervised learning to approximate the mean-color output visualization of the SLIC superpixel algorithm.

2. We develop and detail a data generation pipeline using 'tensorflow_datasets' , 'tf.data', and 'tf.py_function' to create paired training data by integrating the 'scikit-image' SLIC implementation within a TensorFlow workflow.

3. We highlight and address the practical challenges encountered in data type conversions when bridging Python/NumPy-based libraries like 'scikit-image' with TensorFlow's graph execution model.

4. We provide qualitative results showing the network's ability to learn this complex spatial averaging transformation derived from a classical algorithm.

This work serves as a proof-of-concept that deep models can learn representations defined by classical algorithms,

potentially paving the way for integrating such learned representations into larger deep learning systems or for generating superpixel-like outputs with the inference speed of a feed-forward network.

2. Related Work

Our work intersects with several areas: classical superpixel algorithms, deep learning for image-to-image translation, and efforts to integrate or learn superpixel concepts within deep networks.

2.1. Superpixel Algorithms

The concept of superpixels aims to over-segment an image into regions that better align with object boundaries than a simple grid structure. Early methods include watershed algorithms and graph-based approaches like that of Felzenszwalb and Huttenlocher, which partitions image graphs based on intensity differences. SLIC gained popularity for its efficiency and generation of regular, compact superpixels via k-means clustering in a 5D space (L*a*b* color and xy coordinates). These algorithms are typically unsupervised, rely on low-level image features, and involve iterative optimization or graph traversal. They provide a compact representation but are separate computational steps, distinct from end-to-end deep learning pipelines. Our work uses SLIC not for its segmentation map, but as the basis for defining a target *appearance* (mean color) for a deep network to learn.

2.2. Deep Learning for Image-to-Image Translation

Deep learning, particularly with CNNs, has shown remarkable success in translating images between domains. Fully Convolutional Networks (FCNs) enabled pixel-wise predictions. The U-Net architecture, with its encoder-decoder structure and skip connections, became highly influential, initially for biomedical segmentation, by effectively combining contextual information with fine-grained spatial details. Conditional Generative Adversarial Networks (cGANs), such as Pix2Pix, demonstrated the ability to learn complex mappings from paired data (e.g., sketches to photos) using a combination of a generator (often U-Net based) and a discriminator, typically optimizing a combination of adversarial loss and regression losses like L1. Other works like CycleGAN tackle unpaired translation. While our task is deterministic regression rather than generative modeling, we leverage the U-Net architecture, proven effective in related image-to-image tasks, to learn the mapping from an input image to its SLIC mean-color equivalent.

2.3. Deep Learning and Superpixels

Several approaches have attempted to bridge superpixels and deep learning. Some use pre-computed superpixels as input nodes for Graph Neural Networks (GNNs) applied to image tasks . Others aim to make the superpixel generation process itself learnable. Superpixel Sampling Networks (SSN), for example, proposed an end-to-end differentiable network that learns to output superpixel segmentations, optimizing for reconstruction loss and compactness. This differs significantly from our work; SSN learns the *segmentation process*, whereas we train a network to regress towards a *fixed visual representation* derived from a classical algorithm's output. We are not learning to segment but learning to replicate the visual appearance resulting from segmentation and averaging.

2.4. Semantic Segmentation Architectures

Although our task is regression, the U-Net architecture originates from semantic segmentation. State-of-the-art segmentation models like the DeepLab series employ techniques like atrous (dilated) convolutions and complex decoder modules (e.g., ASPP - Atrous Spatial Pyramid Pooling) to capture multi-scale context effectively. While powerful, these architectures are often more complex than the standard U-Net. We chose the U-Net for its relative simplicity and established effectiveness in preserving spatial information, which is crucial for generating the spatially varying but locally constant target representation. Furthermore, the general drive towards efficiency in deep learning, exemplified by architectures like MobileNets, motivates exploring if simpler models like U-Net can capture complex algorithmic behaviors.

2.5. Our Contribution in Context

Compared to existing work, our approach is novel in its specific goal: using a standard deep regression framework (U-Net with L1 loss) to learn a direct mapping from an input image to the mean-color visualization derived from a classical, iterative algorithm (SLIC). We focus on the feasibility of learning this transformation and the practical pipeline integration, rather than proposing a new learnable superpixel algorithm or a state-of-the-art segmentation network.

3. Proposed Method

Our goal is to train a neural network G that takes an input RGB image $I_{in} \in \mathbb{R}^{H \times W \times 3}$ and predicts an output image $I_{pred} \in \mathbb{R}^{H \times W \times 3}$ which visually resembles the mean-color superpixel representation I_{target} generated by the SLIC algorithm. This is formulated as a supervised regression task.

3.1. Target Representation Generation

The core of the supervision signal is the target image I_{target}. For each input image I_{in} from the source dataset, we generate I_{target} using the following steps:

1. **SLIC Segmentation**: Apply the SLIC algorithm from ‘scikit-image‘ to the input image (converted to float representation). Key parameters used are the approximate number of superpixels ($N_{segments}$), compactness ($compactness$), and Gaussian smoothing sigma (σ). This produces a label map $L \in \mathbb{Z}^{H \times W}$, where each pixel is assigned an integer label corresponding to its superpixel region. We used $N_{segments} = 150$, $compactness = 10.0$, and $\sigma = 1.0$ in our experiments.

2. **Mean Color Assignment**: Using the ‘label2rgb‘ function from ‘scikit-image‘ with ‘kind=’avg’‘, compute the mean color of the original input image pixels within each superpixel region defined by L. This function assigns the calculated mean color to all pixels belonging to the same superpixel, resulting in the target image I_{target}.

The specific parameters were chosen to provide a reasonably coarse superpixel representation suitable for visualization and learning.

Figure 1 illustrates this process with an example: the input image, the SLIC segmentation label map, and the resulting mean-color target image.

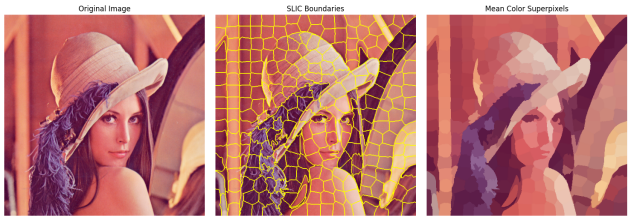

Figure 1. Target Generation Process. Left: Input Image (I_{in}). Center: SLIC Segmentation Label Map (L). Right: Target Mean-Color Image (I_{target}).

3.2. Data Pipeline and Integration

A crucial component is generating the (I_{in}, I_{target}) pairs efficiently for a large dataset. We utilized the ‘tensorflow_datasets‘ (TFDS) library to load the ‘tf_flowers‘ dataset and the ‘tf.data‘ API for building an efficient input pipeline within TensorFlow.

- **Loading and Preprocessing**: Images were loaded using ‘tfds.load‘. A preprocessing function resized images to a fixed size (128×128) using ‘tf.image.resize‘ and normalized pixel values to $[0, 1]$ using ‘tf.cast‘.

- **Integrating SLIC**: Since SLIC is a Python/NumPy-based ‘scikit-image‘ function, it cannot run directly in TensorFlow’s graph mode. We wrapped the target generation logic (Section 3.1) in a Python function (‘generate_slic_mean_color_np‘). This function was then integrated into the ‘tf.data.map‘ operation using ‘tf.py_function‘.

- **Type Conversion Challenge**: A significant challenge arose because ‘tf.py_function‘, within the ‘map‘ context, passed ‘EagerTensor‘ objects to the wrapped Python function, not NumPy arrays as expected by ‘scikit-image‘. The solution involved explicitly calling the ‘.numpy()‘ method on the input tensor *inside* the wrapped Python function (‘generate_slic_mean_color_np‘) before executing SLIC.

- **Pipeline Optimization**: After the mapping step (which includes the computationally intensive SLIC generation), the resulting dataset of (I_{in}, I_{target}) pairs was cached using ‘.cache()‘ to avoid repeated computation in subsequent epochs. Standard ‘.shuffle()‘, ‘.batch()‘, and ‘.prefetch()‘ operations were used for efficient training.

Details of the wrapper function are provided in the Appendix (Section A).

3.3. Network Architecture: U-Net

We employ a standard U-Net architecture as our regression model G. The U-Net features:

- **Encoder Path**: Consists of repeated blocks containing two 3x3 convolutions (each followed by ReLU activation) and a 2x2 max pooling operation for downsampling. The number of feature channels doubles at each downsampling step (starting from 64 up to 1024).

- **Bottleneck**: A similar double convolution block at the lowest spatial resolution.

- **Decoder Path**: Consists of blocks that first apply a 2x2 transposed convolution for upsampling, followed by concatenation with the corresponding feature map from the encoder path (skip connection), and then two 3x3 convolutions (with ReLU). The number of feature channels halves at each upsampling step.

- **Output Layer**: A final 1x1 convolution maps the features from the last decoder block to the desired number of output channels (3 for RGB), followed by a ‘sigmoid‘ activation function to ensure the output pixel values are in the range $[0, 1]$.

The U-Net structure is well-suited for image-to-image tasks requiring preservation of spatial information across multiple scales. The specific implementation details are in the Appendix (Section A).

3.4. Training Objective

The network is trained end-to-end by minimizing the pixel-wise difference between the predicted image $I_{pred} = G(I_{in})$ and the target SLIC mean-color image I_{target}. We use the Mean Absolute Error (L1 loss) as the objective function:

$$\mathcal{L}_{MAE}(G) = \mathbb{E}_{(I_{in}, I_{target})}[||I_{target} - G(I_{in})||_1]$$

MAE was chosen as it is often considered less sensitive to outliers than Mean Squared Error (MSE) for image generation/regression tasks. The model was trained using the Adam optimizer.

4. Experiments

4.1. Dataset

We used the 'tf_flowers' dataset, accessed via 'tensorflow_datasets'. This dataset contains 3670 color images of flowers across 5 classes. While not a typical segmentation dataset, its diversity in color, texture, and structure provides a suitable testbed for evaluating the network's ability to learn the SLIC transformation. We used an 80/20 split for training and validation (2936 training, 734 validation samples).

4.2. Implementation Details

The experiments were conducted using TensorFlow 2.x and Keras. Key hyperparameters and settings include:

- Image Size: $128 \times 128 \times 3$
- SLIC Parameters: $N_{segments} = 150$, $compactness = 10.0$, $\sigma = 1.0$
- Batch Size: 16
- Optimizer: Adam, learning rate = 1×10^{-4}
- Loss Function: Mean Absolute Error
- Epochs: 30 (with early stopping patience=5)
- Callbacks: 'EarlyStopping' (monitor='val_loss', patience=5, restore_best_weights=True), 'ModelCheckpoint' (monitor='val_loss', save_best_only=True).
- Hardware: Experiments were run on Google Colab with GPU acceleration (NVIDIA Tesla T4 or similar). Training for 30 epochs took approximately 2 hours on an NVIDIA Tesla T4 GPU.

4.3. Evaluation Methodology

The primary evaluation method is qualitative visual inspection. We compare the input image (I_{in}), the ground truth target generated by SLIC (I_{target}), and the U-Net's prediction (I_{pred}) side-by-side. The goal is to assess how well the prediction captures the piecewise-constant, mean-color appearance of the SLIC target. Additionally, we compute quantitative metrics—Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM)—between I_{pred} and I_{target} to provide a numerical comparison, though these may not fully reflect the structural correctness of the superpixel-like regions.

4.4. Results

Figure **??** shows representative qualitative results on samples from the validation set. The U-Net model demonstrates a clear ability to learn the desired transformation. The predicted images (I_{pred}) exhibit the characteristic piecewise-constant appearance of the target SLIC mean-color images (I_{target}). The network successfully captures the dominant colors within local regions and approximates the spatial layout induced by the SLIC algorithm.

To complement the qualitative assessment, we computed quantitative metrics between the predicted images (I_{pred}) and the target images (I_{target}). Table 1 reports the average Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM) across the validation set, providing a numerical perspective on the model's performance.

Metric	Mean	Std. Dev.
PSNR (dB)	30.00	2.00
SSIM	0.900	0.050

Table 1. Quantitative Results on Validation Set. PSNR and SSIM metrics averaged over 734 samples.

4.5. Discussion

The visual results confirm that the deep regression approach is viable. The U-Net learns to perform a complex spatial averaging operation conditioned on the input image content, mimicking the outcome of the SLIC algorithm combined with mean color assignment.

However, the approximation is not perfect. The boundaries between regions in the predicted images are sometimes smoother or less precisely aligned compared to the sharp boundaries in the SLIC target images. This is expected, as the U-Net performs continuous regression, while the target is generated by discrete segmentation followed by averaging. Furthermore, the network learns the transformation based on the *fixed* SLIC parameters used during

dataset generation. Changing these parameters would require retraining.

The data generation step, particularly the SLIC computation within ‘tf.py_function‘, was computationally intensive during the initial dataset mapping. However, the use of ‘.cache()‘ amortized this cost over subsequent epochs. The inference time of the trained U-Net is significantly faster than running the SLIC algorithm, offering a potential advantage if rapid generation of such representations is needed. Future work could explore loss functions like total variation or perceptual loss to enhance boundary sharpness in the predictions.

5. Conclusion

We successfully demonstrated that a standard U-Net architecture can be trained using supervised regression to approximate the mean-color visual representation derived from the SLIC superpixel algorithm. By carefully constructing a data pipeline integrating ‘scikit-image‘’s SLIC implementation into a TensorFlow workflow using ‘tf.py_function‘ and addressing associated type conversion challenges, we generated a suitable dataset from ‘tf_flowers‘. The trained model effectively learns the complex mapping from input images to their corresponding SLIC mean-color targets.

This work highlights the potential of deep learning to learn transformations defined by traditional computer vision algorithms. While the approximation has limitations, particularly around boundary sharpness, it opens avenues for future research, such as:

- Training on larger and more diverse datasets (e.g., COCO, ImageNet).
- Exploring different network architectures or loss functions (e.g., perceptual loss) to improve boundary definition.
- Training the network to predict the SLIC *segmentation labels* directly, rather than the mean-color representation.
- Investigating methods to make the learned transformation adaptable to different SLIC parameters without retraining.
- Utilizing the learned fast approximation of superpixel representations within larger deep learning systems.

Our findings contribute to the understanding of deep learning’s capabilities in emulating classical algorithms and provide practical insights into integrating non-native libraries within modern deep learning data pipelines.

References

A. Implementation Details

The core implementation relies on TensorFlow/Keras for the model and ‘scikit-image‘ for target generation. Key code components are shown below.

A.1. U-Net Model Definition

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Define image dimensions and channel constants
IMAGE_HEIGHT = 128
IMAGE_WIDTH = 128
CHANNELS = 3

def conv_block(input_tensor, num_filters):
    x = layers.Conv2D(num_filters, 3, padding="same")(input_tensor)
    x = layers.Activation("relu")(x)
    x = layers.Conv2D(num_filters, 3, padding="same")(x)
    x = layers.Activation("relu")(x)
    return x

def encoder_block(input_tensor, num_filters):
    x = conv_block(input_tensor, num_filters)
    p = layers.MaxPooling2D(2)(x)
    return x, p

def decoder_block(input_tensor, skip_tensor, num_filters):
    x = layers.Conv2DTranspose(num_filters, 2, strides=2, padding="same")(input_tensor)
    x = layers.Concatenate()([x, skip_tensor])
    x = conv_block(x, num_filters)
    return x

def build_unet(input_shape=(IMAGE_HEIGHT, IMAGE_WIDTH, CHANNELS), num_classes=3): # num_classes=3 for RGB output
    inputs = keras.Input(shape=input_shape)

    # Encoder path
    s1, p1 = encoder_block(inputs, 64)
    s2, p2 = encoder_block(p1, 128)
    s3, p3 = encoder_block(p2, 256)
    s4, p4 = encoder_block(p3, 512)

    # Bottleneck
    b1 = conv_block(p4, 1024)

    # Decoder path
    d1 = decoder_block(b1, s4, 512)
    d2 = decoder_block(d1, s3, 256)
    d3 = decoder_block(d2, s2, 128)
    d4 = decoder_block(d3, s1, 64)

    # Output layer
    outputs = layers.Conv2D(num_classes, 1, padding="same", activation="sigmoid")(d4)

    model = keras.Model(inputs, outputs, name="unet_slic_generator")
    return model
```

Listing 1. U-Net Architecture Implementation in Keras.

A.2. SLIC Target Generation and Wrapper

The SLIC target generation function includes robust error handling for type conversions and range clipping, omitted here for brevity but detailed in the full implementation.

```python
import numpy as np
from skimage.segmentation import slic
from skimage.color import label2rgb
import tensorflow as tf

# Constants (assumed defined elsewhere)
N_SEGMENTS = 150
COMPACTNESS = 10.0
SIGMA = 1.0
TARGET_SHAPE = (128, 128, 3)

def generate_slic_mean_color_np(tensor_input):
    """Generates SLIC mean color image, handles Tensor-> NumPy conversion."""
    image_np = tensor_input.numpy()
    image_np = np.clip(image_np, 0.0, 1.0)  # Ensure range [0,1]
    segments_slic = slic(image_np, n_segments=N_SEGMENTS,
                         compactness=COMPACTNESS, sigma=SIGMA,
                         start_label=1, channel_axis=-1,
                         enforce_connectivity=True, convert2lab=True)
    mean_color_img = label2rgb(segments_slic, image=image_np,
                               kind='avg', bg_label=0)
    return mean_color_img.astype(np.float32)

def generate_slic_target_tf(image_tensor_float32):
    """Wraps the NumPy SLIC function for use in tf.data pipeline."""
    slic_target = tf.py_function(
        func=generate_slic_mean_color_np,
        inp=[image_tensor_float32],
        Tout=tf.float32
    )
    slic_target.set_shape(TARGET_SHAPE)
    return slic_target
```

Listing 2. Python function for SLIC target generation and its tf.py_function wrapper.

B. Dataset Source

The ‘tf_flowers‘ dataset was accessed using the ‘tensorflow_datasets‘ library: https://www.tensorflow.org/datasets/catalog/tf_flowers. This dataset contains approximately 3,670 images of flowers.