Detailed Explanation of the Code

1. Imports and Setup

Your code begins with importing various libraries necessary for the image processing and GUI components of the project:

```
import numpy as np
import cv2
import os
from tkinter import filedialog, Tk, Button, Label, messagebox, DoubleVar
from tkinter.ttk import Progressbar, Style
from matplotlib import pyplot as plt
from threading import Thread
from tkinter import Tk, Text, END
```

- **NumPy**
    - **Purpose**: NumPy is a fundamental package for scientific computing in Python.
    - **Functionality**: It provides support for arrays, matrices, and many mathematical functions to operate on these data structures efficiently.
    - **Usage in Project**: NumPy is used for numerical operations and array manipulations, such as calculating the mean and standard deviation of the color channels.
- **OpenCV**
    - **Purpose**: OpenCV (Open Source Computer Vision Library) is an open-source library that provides tools for computer vision and image processing.
    - **Functionality**: It includes functions for reading, writing, and manipulating images, as well as various image processing algorithms.
    - **Usage in Project**: OpenCV is used to read images, convert them between color spaces (BGR to LAB and vice versa), and perform the core color transfer operations.
- **OS**
    - **Purpose**: The OS module in Python provides a way of using operating system-dependent functionality like reading or writing to the file system.

- **Functionality**: It includes functions to interact with the file system, such as file and directory manipulation.
- **Usage in Project**: The OS module is used to construct file paths and handle file operations such as saving the processed images.

- **Tkinter**

  - **Purpose**: Tkinter is the standard GUI (Graphical User Interface) library for Python.
  - **Functionality**: It provides a toolkit to create and manage GUI elements like windows, buttons, labels, and dialogs.
  - **Usage in Project**: Tkinter is used to create the main application window, buttons for selecting files and folders, labels for displaying messages, and a progress bar to indicate the status of the conversion process.

- **Matplotlib**

  - **Purpose**: Matplotlib is a plotting library for the Python programming language.
  - **Functionality**: It provides an API for embedding plots into applications and creating static, animated, and interactive visualizations.
  - **Usage in Project**: Matplotlib is used to display the source, target, and result images side
  - **Threading**

  - **Purpose**: The threading module in Python is used to run multiple threads (tasks, function calls) concurrently.
  - **Functionality**: It allows the program to perform tasks in the background, keeping the main application responsive.
  - **Usage in Project**: Threading is used to run the color transfer process in a separate thread, preventing the GUI from freezing during the operation.

2. Function Definitions

Reading and Converting Images:

The `read_file` function reads a source and target image, then converts them from BGR color space (used by OpenCV) to LAB color space.

```
11  def read_file(sn, tn):
12      s = cv2.imread(sn)
13      s = cv2.cvtColor(s, cv2.COLOR_BGR2LAB)
14      t = cv2.imread(tn)
15      t = cv2.cvtColor(t, cv2.COLOR_BGR2LAB)
16      return s, t
17
```

LAB Color Space:

LAB color space separates lightness (L) from the color components (A and B). L represents lightness, while A and B represent the color opponents green–red and blue–yellow. This separation allows for more precise color adjustments.

Calculating Mean and Standard Deviation:

The `get_mean_and_std` function calculates the mean and standard deviation of the color values in the LAB channels of the image.

```
18  def get_mean_and_std(x):
19      x_mean, x_std = cv2.meanStdDev(x)
20      x_mean = np.hstack(np.around(x_mean, 2))
21      x_std = np.hstack(np.around(x_std, 2))
22      return x_mean, x_std
23
```

Image Processing Theory:

- Mean and Standard Deviation: The mean measures the average color value in each channel, and the standard deviation measures the spread of the color values around the mean, indicating contrast.

Color Transfer:

The `color_transfer` function applies the color transfer technique to each pair of source and target images.

```python
def color_transfer(sources, targets, output_folder):
    results = []  # List to store the source, target, and result images for display

    for n in range(len(sources)):
        print("Converting picture " + str(n+1) + "...")
        s, t = read_file(sources[n], targets[n])
        s_mean, s_std = get_mean_and_std(s)
        t_mean, t_std = get_mean_and_std(t)

        height, width, channel = s.shape
        for i in range(height):
            for j in range(width):
                for k in range(channel):
                    x = s[i, j, k]
                    x = ((x - s_mean[k]) * (t_std[k] / s_std[k])) + t_mean[k]
                    x = round(x)
                    x = 0 if x < 0 else x
                    x = 255 if x > 255 else x
                    s[i, j, k] = x
        result = cv2.cvtColor(s, cv2.COLOR_LAB2BGR)
        result_path = os.path.join(output_folder, 'r' + str(n+1) + '.bmp')
        cv2.imwrite(result_path, result)

        # Store images for later display
        results.append((cv2.cvtColor(cv2.imread(sources[n]), cv2.COLOR_BGR2RGB),
                        cv2.cvtColor(cv2.imread(targets[n]), cv2.COLOR_BGR2RGB),
                        cv2.cvtColor(result, cv2.COLOR_BGR2RGB)))

        # Update the progress bar
        progress_var.set((n + 1) / len(sources) * 100)
        root.update_idletasks()

    return results
```

Color Transfer Algorithm:

The algorithm normalizes the color values of the source image by subtracting the mean and dividing by the standard deviation, then scales and shifts these values to match the target image's color distribution. This process adjusts the source image’s color characteristics to those of the target image.

-Displaying Images:

The `show_images` function displays the source, target, and result images side by side using Matplotlib.

```
58  def show_images(results):
59      for idx, (source, target, result) in enumerate(results):
60          plt.figure(figsize=(15, 5))
61
62          plt.subplot(1, 3, 1)
63          plt.imshow(source)
64          plt.title(f'Source Image {idx+1}')
65          plt.axis('off')
66
67          plt.subplot(1, 3, 2)
68          plt.imshow(target)
69          plt.title(f'Target Image {idx+1}')
70          plt.axis('off')
71
72          plt.subplot(1, 3, 3)
73          plt.imshow(result)
74          plt.title(f'Result Image {idx+1}')
75          plt.axis('off')
76
77          plt.tight_layout()
78          plt.show()
79
```

3. GUI Components

File and Folder Selection:

The functions `select_source_files`, `select_target_files`, and `select_output_folder` allow users to select the source images, target images, and the output folder, respectively.

```
def select_source_files():
    global source_files
    source_files = filedialog.askopenfilenames(title='Select Source Images', filetypes=[("Image files", "*.bmp;*.jpg;*.jpeg;*.png")])

    if not source_files:
        messagebox.showerror("Error", "No source files selected.")
    else:
        source_label.config(text=f"Selected {len(source_files)} source files")

def select_target_files():
    global target_files
    target_files = filedialog.askopenfilenames(title='Select Target Images', filetypes=[("Image files", "*.bmp;*.jpg;*.jpeg;*.png")])

    if not target_files:
        messagebox.showerror("Error", "No target files selected.")
    else:
        target_label.config(text=f"Selected {len(target_files)} target files")

def select_output_folder():
    global output_folder
    output_folder = filedialog.askdirectory(title='Select Output Folder')

    if not output_folder:
        messagebox.showerror("Error", "No output folder selected.")
    else:
        output_label.config(text=f"Output folder selected: {output_folder}")

```

Starting Conversion:

The `start_conversion` function starts the color transfer process after validating the user inputs.

```
def start_conversion():
    if not source_files or not target_files:
        messagebox.showerror("Error", "Please select source and target files first.")
        return

    if len(source_files) != len(target_files):
        messagebox.showerror("Error", "The number of source and target files must be the same.")
        return

    if not output_folder:
        messagebox.showerror("Error", "Please select an output folder.")
        return

    # Start the conversion in a separate thread to keep the UI responsive
    conversion_thread = Thread(target=run_conversion)
    conversion_thread.start()

def run_conversion():
    results = color_transfer(source_files, target_files, output_folder)
    show_images(results)
```

Running Conversion:

The `run_conversion` function executes the color transfer and displays the results.

```
def run_conversion():
    results = color_transfer(source_files, target_files, output_folder)
    show_images(results)

```

4. Main Application

Setting up the Main Window:

The main window is set up with various GUI elements like buttons, labels, and a progress bar.

```
if __name__ == "__main__":
    source_files = []
    target_files = []
    output_folder = ""

    root = Tk()
    root.title("Color Transfer Application")

    style = Style()
    style.configure("TButton", font=("Arial", 12), width=20)
    style.configure("TLabel", font=("Arial", 12), foreground="blue")
    style.configure("TProgressbar", thickness=30)  # Increase the height of the progress bar

    # Create a Text widget
    text_widget = Text(root, height=2, width=40, font=("Arial", 16, "bold"), bd=0,
bg=root.cget("bg"), wrap="word")
    text_widget.pack(pady=10)

    # Insert text with colored word "color"
    text_widget.insert(END, "Welcome to ", "color_black")
    text_widget.insert(END, "C", "color_red")
    text_widget.insert(END, "O", "color_green")
    text_widget.insert(END, "L", "color_blue")
    text_widget.insert(END, "O", "color_orange")
    text_widget.insert(END, "R", "color_purple")
    text_widget.insert(END, " transfer app", "color_black")

    # Add tags for colors
    #different colors to make it nice
    text_widget.tag_configure("color_black", foreground="black", font=("Arial", 16, "bold"))
    text_widget.tag_configure("color_red", foreground="red", font=("Arial", 20, "bold"))
    text_widget.tag_configure("color_green", foreground="green", font=("Arial", 20, "bold"))
    text_widget.tag_configure("color_blue", foreground="blue", font=("Arial", 20, "bold"))
```

```
    text_widget.tag_configure("color_orange", foreground="orange", font=("Arial", 20, "bold"))
    text_widget.tag_configure("color_purple", foreground="purple", font=("Arial", 20, "bold"))

    # Make the Text widget read-only
    text_widget.configure(state="disabled")

    # Center the text using padding
    text_widget.tag_add("center", "1.0", END)  # Add a tag to center the text
    text_widget.tag_configure("center", justify="center")

    #Source file selection
    Label(root, text="Select source and target images", font=("Arial", 12, "bold"),
fg="green").pack(pady=10)
    Button(root, text="Select Source Files", command=select_source_files).pack(pady=10)
    source_label = Label(root, text="", font=("Arial", 12), fg="red")
    source_label.pack(pady=5)

    #Target file selection
    Button(root, text="Select Target Files", command=select_target_files).pack(pady=10)
    target_label = Label(root, text="", font=("Arial", 12), fg="red")
    target_label.pack(pady=5)
    Label(root, text="", height=2).pack()

    #Output folder selection
    Label(root, text="Select where to save result images", font=("Arial", 12, "bold"),
fg="green").pack(pady=10)
    Button(root, text="Select Output Folder", command=select_output_folder).pack(pady=10)
    output_label = Label(root, text="", font=("Arial", 12), fg="red")
    output_label.pack(pady=5)
    Label(root, text="", height=2).pack()

    #Button to Start the operation
    Button(root, text="Start", command=start_conversion, width=20, height=2,
       bg="red", fg="white", font=("Arial", 14, "bold")).pack(pady=10)

    #progress bar
    progress_var = DoubleVar()
    progress_bar = Progressbar(root, variable=progress_var, maximum=100, style="TProgressbar")
    progress_bar.pack(pady=10, padx=20, fill="x")

    # Add credentials at the bottom
    Label(root, text="Yashmika Senadheera\n21/ENG/128", font=("Arial", 10),
fg="purple").pack(pady=20)

    root.mainloop()
```

Image Processing Theory Integration

1. LAB Color Space: The choice of LAB color space is crucial as it allows for independent manipulation of luminance and chromatic components, facilitating more accurate and visually pleasing color adjustments.

2. Mean and Standard Deviation: These statistical measures help in normalizing and matching the color distributions of source and target images, ensuring a consistent and balanced color transfer.

3. Color Transfer Algorithm: By normalizing the source image's color values and then scaling them to match the target image's color distribution, the algorithm achieves the desired color transfer, effectively blending the aesthetic elements of the target image into the source image.