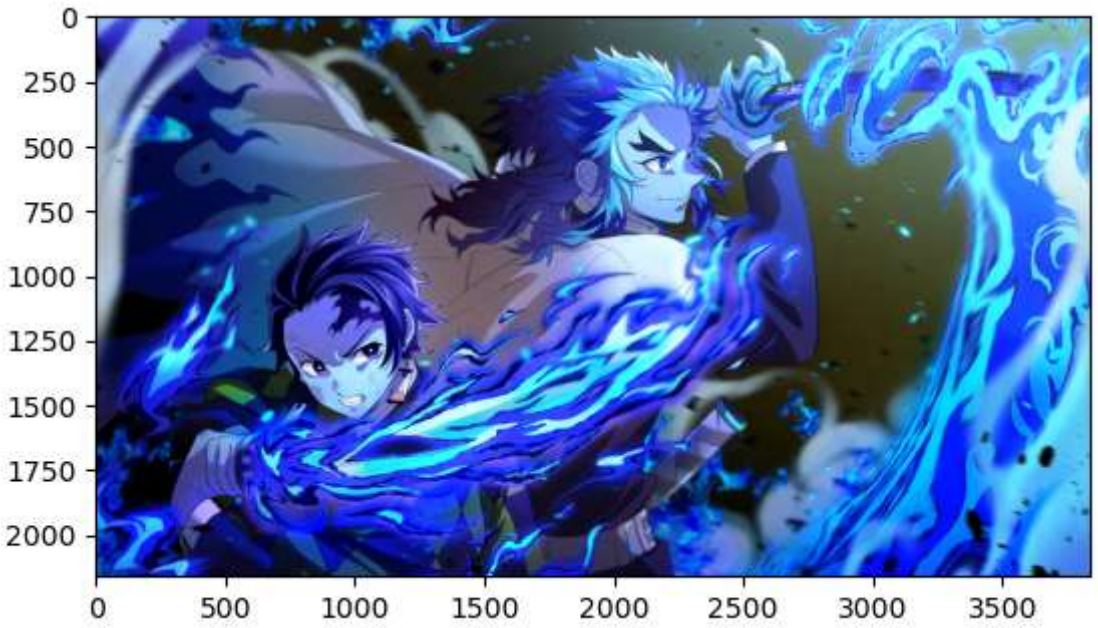
In [1]:
```
import cv2
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```
image_path ='kyojiro.jpg'

image = cv2.imread(image_path)
plt.imshow(image)
```

Out[2]:
```
<matplotlib.image.AxesImage at 0x1783b692af0>
```



In [14]:
```
# Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Initialize an empty list to store the compressed image
compressed_image = []

# get the shape of the image
height,width = gray_image.shape


# iterate over the rows of the image
for row in range(height):

    # Initialize a variable to store the current pixel value
    cur_pixel = -1

    # Initialize a variable to store the current run length
    cur_run = 0


    # iterate over the columns of the image
    for col in range(width):

        # Get the pixel value
        pixel = gray_image[row, col]

        # Check if the pixel value is equal to the current pixel value
        if pixel == cur_pixel:

            # increment the current run length
            cur_run += 1

        else:
            # If the pixel value is different, append the current run to the c
            compressed_image.append((cur_pixel, cur_run))

            # Update the current pixel value and reset the current run length
            cur_pixel = pixel
            cur_run = 1

    # Append the last run to the compressed image
    compressed_image.append((cur_pixel, cur_run))
```

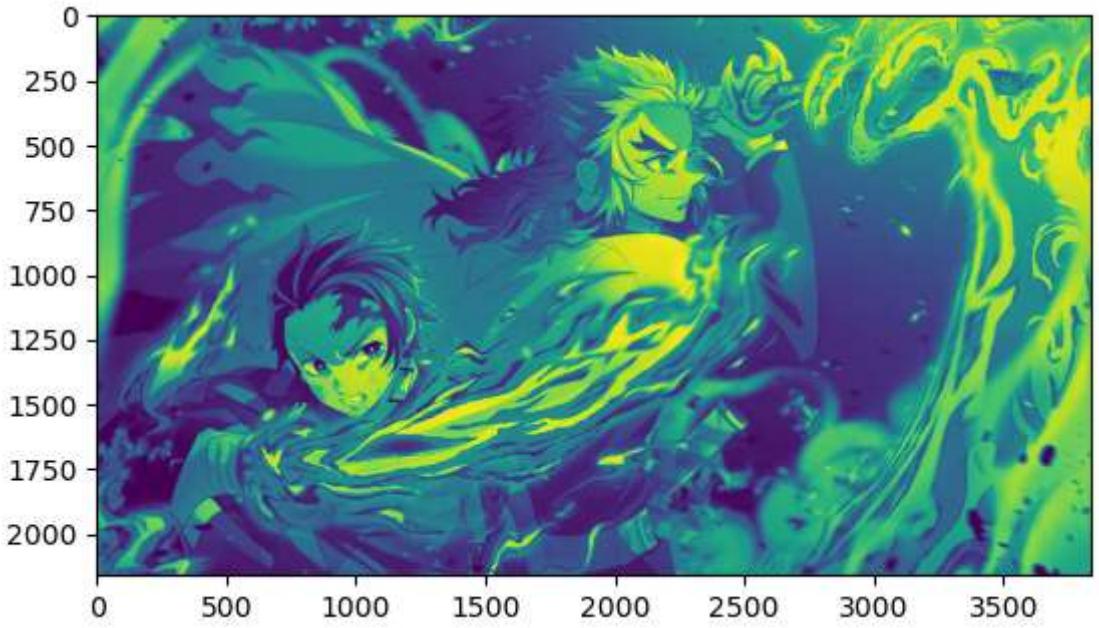
In [15]:
```
# create a new image to store the decompressed image
decompressed_image = np.zeros((height,width),dtype=np.uint8)

# initialize variables to store the current row and column
row=0
col=0

# iterate over the element of the compressed image
for pixel, run_length in compressed_image:

    # set the pixel value for the current run
    decompressed_image[row,col:col+run_length]=pixel

    # update the column index
    col += run_length

    # if the column index is greater than or equal to the width of the image,
    if col >= width:
        row +=1
        col = 0
```

```
C:\Users\Gaurav\AppData\Local\Temp\ipykernel_15320\2965223580.py:12: Deprecat
ionWarning: NumPy will stop allowing conversion of out-of-bound Python intege
rs to integer arrays.  The conversion of -1 to uint8 will fail in the future.
For the old behavior, usually:
    np.array(value).astype(dtype)
will give the desired result (the cast overflows).
  decompressed_image[row,col:col+run_length]=pixel
```

In [5]:
```
# display the original image
print('original image')
plt.imshow(gray_image)
```

```
original image
```

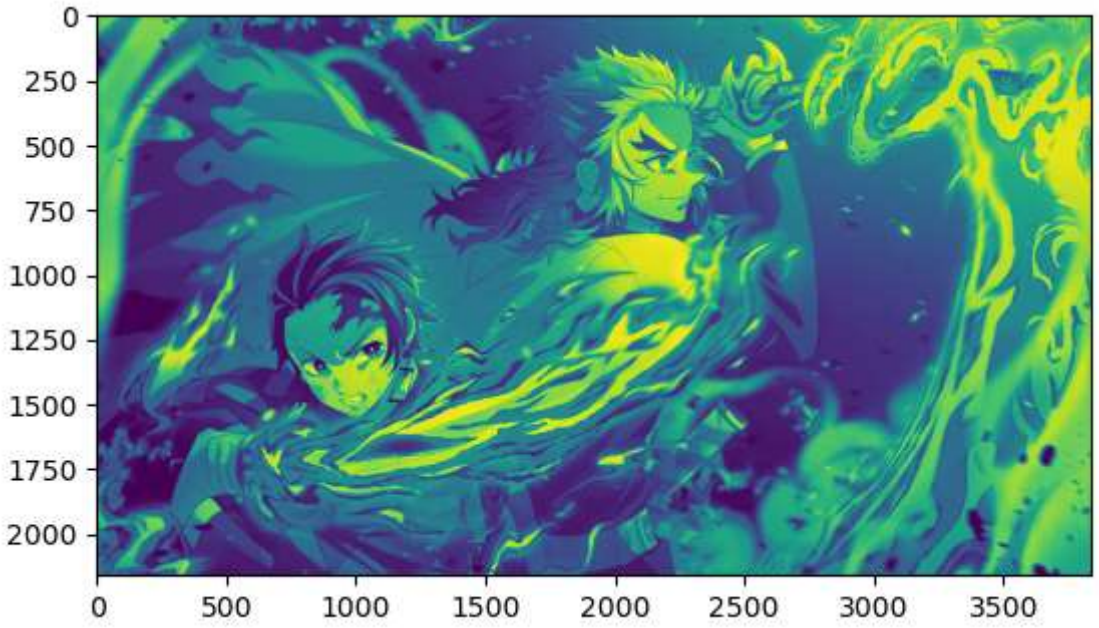
Out[5]:
```
<matplotlib.image.AxesImage at 0x1785ba79d90>
```



In [6]:
```
# display the decompressed image
print('Decompressed Image')
plt.imshow(decompressed_image)
```

```
Decompressed Image
```

Out[6]:
```
<matplotlib.image.AxesImage at 0x1785cc3d3a0>
```

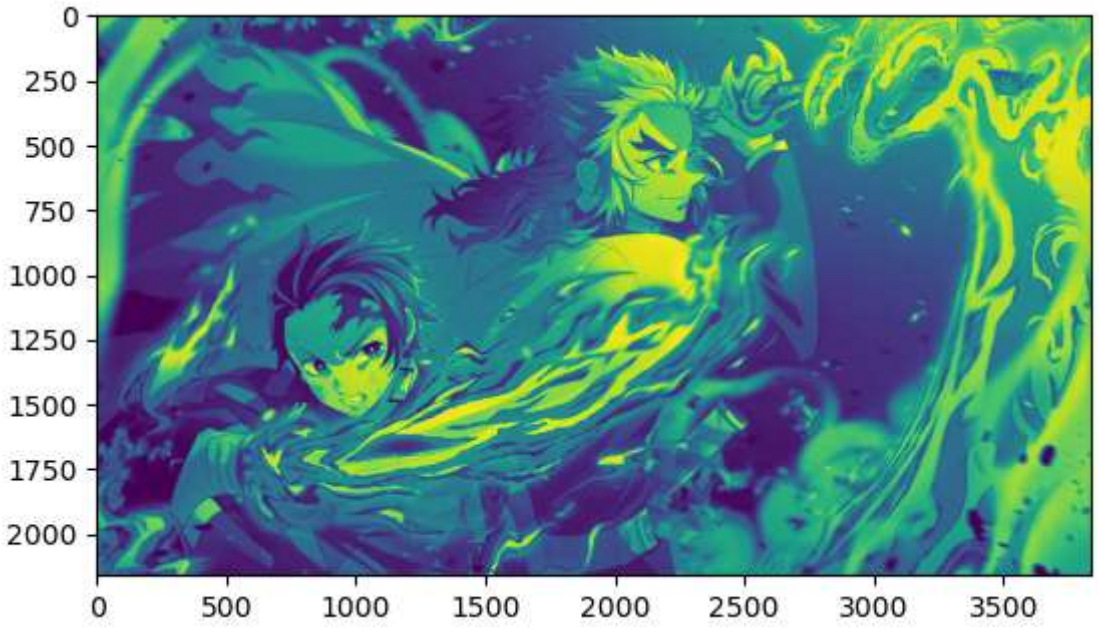


In [7]:
```
# convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# get the shape of the the image
height, width = gray_image.shape

# create a function to compress a region of the image using the DCT image comp
def compress_region(region):

   # get the shape of the region region
   region_height, region_width = region.shape

   # apply the DCT to the region
   dct_region = cv2.dct(region.astype(np.float32))

   # set the low frequency coefficients to zero
   dct_region[:region_height//8, :region_width//8] = 0

   # apply the inverse DCT to the region
   idct_region= cv2.idct(dct_region)

   # return the region
   return idct_region

#create a copy of the image to store the compressed image
compressed_image = gray_image.copy()

#iterate over the blocks in the image
for i in range(0, height, 8):
    for j in range(0, width, 8):

      # get the current block
      block = gray_image[i:i+8, j:j+8]

      # compress the block using the DCT image compression technique
      compressed_block = compress_region(block)

      # insert the compressed block into the compressed image
      compressed_image[i:i+8, j:j+8] = compressed_block
```

In [8]:
```
# display the original image
print('original image')
plt.imshow(gray_image)
```

```
original image
```

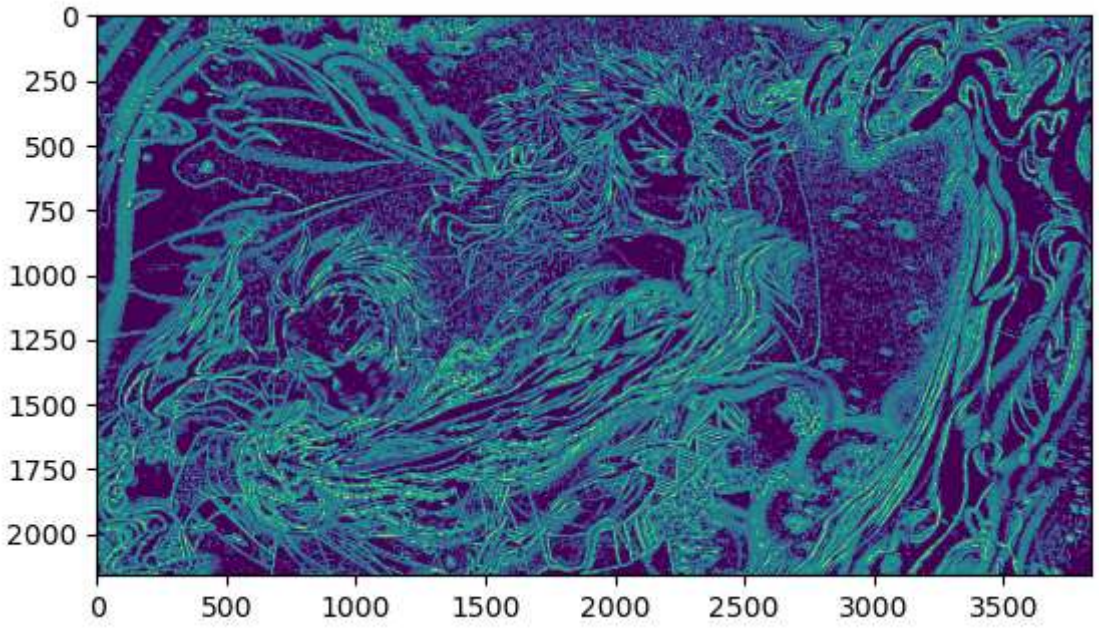
Out[8]:
```
<matplotlib.image.AxesImage at 0x1785ba293d0>
```



In [9]:
```
# display the compressed image
print('Compressed Image')
plt.imshow(compressed_image)
```

```
Compressed Image
```

Out[9]:
```
<matplotlib.image.AxesImage at 0x1785cb37070>
```

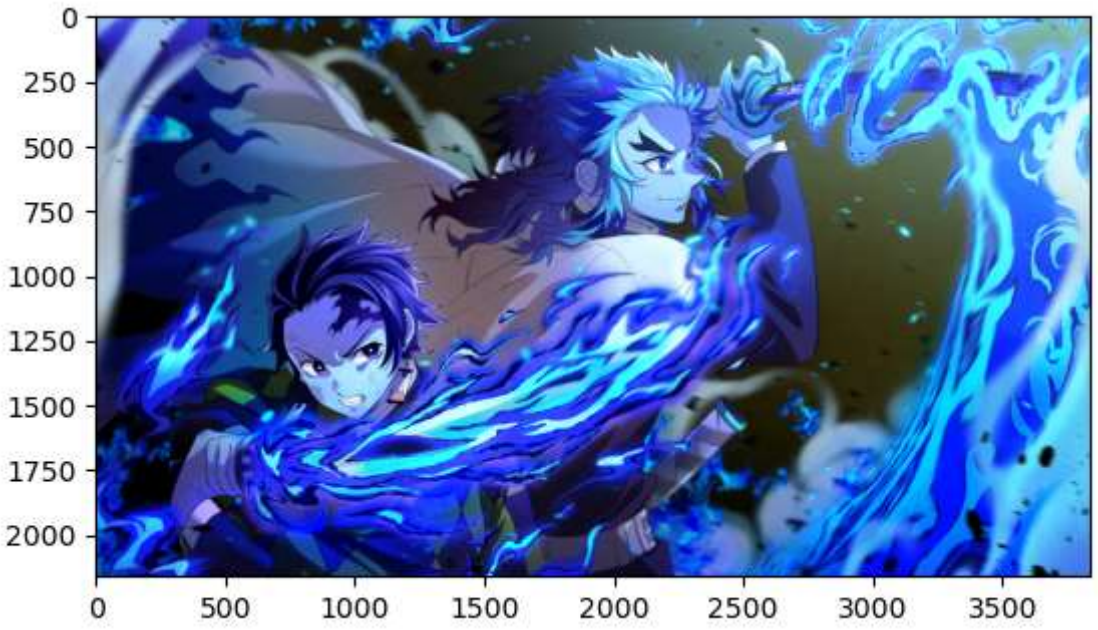


In [10]:
```
def quantize(image, levels):
    # Flatten the image
    flattened_image = image.flatten()

    # Find the maximum and minimum pixel values
    max_val = np.max(flattened_image)
    min_val = np.min(flattened_image)

    # Divide the range into evenly spaced intervals
    intervals = np.linspace(min_val, max_val, levels+1)

    # Find the mean value of each interval
    means = [(intervals[i] + intervals[i+1]) / 2 for i in range(levels)]

    # Quantize the pixel values
    quantized_image = np.array([means[np.searchsorted(intervals, val) - 1] for

    # Reshape the quantized image back into its original shape
    quantized_image = quantized_image.reshape(image.shape)

    return quantized_image

# Example usage
# Assuming you have 'image' defined somewhere
# quantized_image = quantize(image, 16)
```

In [11]:
```
# dispaly the original image
print ('Original Image')
plt.imshow(image)
```

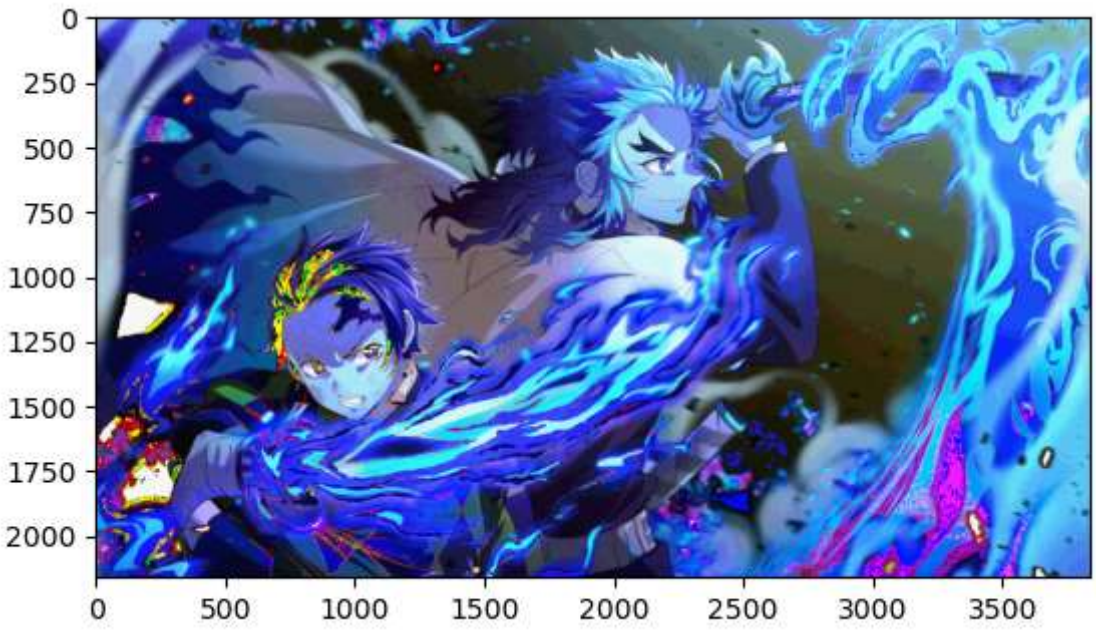
```
Original Image
```

Out[11]:
```
<matplotlib.image.AxesImage at 0x1785b9dfd60>
```



In [12]:
```
quantized_image = quantize(image, 16)
quantized_image = quantized_image.astype(np.float32) / 255.0
print("Compressed Image : ")
# Display the image
plt.imshow(quantized_image)
plt.show()
```

```
Compressed Image :
```



In [ ]:
```

```