```
pip install -U tensorflow-addons
```

```
Requirement already satisfied: tensorflow-addons in /usr/local/lib/python3.10/dist-packages (0.22.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow-addons) (23.2)
Requirement already satisfied: typeguard<3.0.0,>=2.7 in /usr/local/lib/python3.10/dist-packages (from tensorflow-addons) (2.13.3)
```

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow_addons as tfa

import warnings
warnings.filterwarnings("ignore")

# Suppress specific warnings
warnings.filterwarnings("ignore", message="A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy")
warnings.filterwarnings("ignore", message="TensorFlow Addons (TFA) has ended development and introduction of new features.")
```

```
/usr/local/lib/python3.10/dist-packages/tensorflow_addons/utils/tfa_eol_msg.py:23: UserWarning:

TensorFlow Addons (TFA) has ended development and introduction of new features.
TFA has entered a minimal maintenance and release mode until a planned end of life in May 2024.
Please modify downstream libraries to take dependencies from other repositories in our TensorFlow community (e.g. Keras, Keras-CV, and Keras-NLP).

For more information see: https://github.com/tensorflow/addons/issues/2807

  warnings.warn(
```

```
# own dataset
import os
import numpy as np
from tensorflow import keras
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from sklearn.model_selection import train_test_split

num_classes = 100
input_shape = (32, 32, 3)

# Define path to your data folder
data_folder = "/home/DATA"  # Update with your actual path

# List all image filenames in the folder
image_filenames = os.listdir(data_folder)

# Initialize empty lists for images and labels
images = []
labels = []

# Load and preprocess the images
for img_filename in image_filenames:
    img_path = os.path.join(data_folder, img_filename)
    img = load_img(img_path, target_size=(32, 32))
    img_array = img_to_array(img)
    images.append(img_array)

    #Extract class label from filename (assuming filenames are like "class_1_image.jpg")
    class_label = img_filename.split('_')[0]
    labels.append(class_label)

# Convert lists to numpy arrays
images = np.array(images)
labels = np.array(labels)

# Split data into train and test sets
x_train, x_test, y_train, y_test = train_test_split(images, labels, test_size=0.2, random_state=42)

from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.fit_transform(y_test)

print(f"x_train shape: {x_train.shape} - y_train shape: {y_train_encoded.shape}")
print(f"x_test shape: {x_test.shape} - y_test shape: {y_test_encoded.shape}")
```

```
x_train shape: (12, 32, 32, 3) - y_train shape: (12,)
x_test shape: (3, 32, 32, 3) - y_test shape: (3,)
```

```
learning_rate = 0.001
weight_decay = 0.0001
batch_size = 256
num_epochs = 100
image_size = 72  # We'll resize input images to this size
patch_size = 6  # Size of the patches to be extract from the input images
num_patches = (image_size // patch_size) ** 2
projection_dim = 64
num_heads = 4
transformer_units = [
    projection_dim * 2,
    projection_dim,
]  # Size of the transformer layers
transformer_layers = 8
mlp_head_units = [2048, 1024]  # Size of the dense layers of the final classifier
```

```
data_augmentation = keras.Sequential(
    [
        layers.Normalization(),
        layers.Resizing(image_size, image_size),
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(factor=0.02),
        layers.RandomZoom(
            height_factor=0.2, width_factor=0.2
        ),
    ],
    name="data_augmentation",
```

```
)
# Compute the mean and the variance of the training data for normalization.
data_augmentation.layers[0].adapt(x_train)


def mlp(x, hidden_units, dropout_rate):
    for units in hidden_units:
        x = layers.Dense(units, activation=tf.nn.gelu)(x)
        x = layers.Dropout(dropout_rate)(x)
    return x


class Patches(layers.Layer):
    def __init__(self, patch_size):
        super().__init__()
        self.patch_size = patch_size

    def call(self, images):
        batch_size = tf.shape(images)[0]
        patches = tf.image.extract_patches(
            images=images,
            sizes=[1, self.patch_size, self.patch_size, 1],
            strides=[1, self.patch_size, self.patch_size, 1],
            rates=[1, 1, 1, 1],
            padding="VALID",
        )
        patch_dims = patches.shape[-1]
        patches = tf.reshape(patches, [batch_size, -1, patch_dims])
        return patches


import matplotlib.pyplot as plt

plt.figure(figsize=(4, 4))
image = x_train[np.random.choice(range(x_train.shape[0]))]
plt.imshow(image.astype("uint8"))
plt.axis("off")

resized_image = tf.image.resize(
    tf.convert_to_tensor([image]), size=(image_size, image_size)
)
patches = Patches(patch_size)(resized_image)
print(f"Image size: {image_size} X {image_size}")
print(f"Patch size: {patch_size} X {patch_size}")
print(f"Patches per image: {patches.shape[1]}")
print(f"Elements per patch: {patches.shape[-1]}")

n = int(np.sqrt(patches.shape[1]))
plt.figure(figsize=(4, 4))
for i, patch in enumerate(patches[0]):
    ax = plt.subplot(n, n, i + 1)
    patch_img = tf.reshape(patch, (patch_size, patch_size, 3))
    plt.imshow(patch_img.numpy().astype("uint8"))
    plt.axis("off")
```

```
Image size: 72 X 72
Patch size: 6 X 6
Patches per image: 144
Elements per patch: 108
```

```
class PatchEncoder(layers.Layer):
    def __init__(self, num_patches, projection_dim):
        super().__init__()
        self.num_patches = num_patches
        self.projection = layers.Dense(units=projection_dim)
        self.position_embedding = layers.Embedding(
            input_dim=num_patches, output_dim=projection_dim
        )

    def call(self, patch):
        positions = tf.range(start=0, limit=self.num_patches, delta=1)
        encoded = self.projection(patch) + self.position_embedding(positions)
        return encoded


def create_vit_classifier():
    inputs = layers.Input(shape=input_shape)
    # Augment data.
```

```
    augmented = data_augmentation(inputs)
    # Create patches.
    patches = Patches(patch_size)(augmented)
    # Encode patches.
    encoded_patches = PatchEncoder(num_patches, projection_dim)(patches)

    # Create multiple layers of the Transformer block.
    for _ in range(transformer_layers):
        # Layer normalization 1.
        x1 = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
        # Create a multi-head attention layer.
        attention_output = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=projection_dim, dropout=0.1
        )(x1, x1)
        # Skip connection 1.
        x2 = layers.Add()([attention_output, encoded_patches])
        # Layer normalization 2.
        x3 = layers.LayerNormalization(epsilon=1e-6)(x2)
        # MLP.
        x3 = mlp(x3, hidden_units=transformer_units, dropout_rate=0.1)
        # Skip connection 2.
        encoded_patches = layers.Add()([x3, x2])

    # Create a [batch_size, projection_dim] tensor.
    representation = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
    representation = layers.Flatten()(representation)
    representation = layers.Dropout(0.5)(representation)
    # Add MLP.
    features = mlp(representation, hidden_units=mlp_head_units, dropout_rate=0.5)
    # Classify outputs.
    logits = layers.Dense(num_classes)(features)
    # Create the Keras model.
    model = keras.Model(inputs=inputs, outputs=logits)
    return model



def run_experiment(model):
    optimizer = tfa.optimizers.AdamW(
        learning_rate=learning_rate, weight_decay=weight_decay
    )

    model.compile(
        optimizer=optimizer,
        loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
        metrics=[
            keras.metrics.SparseCategoricalAccuracy(name="accuracy"),
            keras.metrics.SparseTopKCategoricalAccuracy(5, name="top-5-accuracy"),
        ],
    )

    checkpoint_filepath = "/tmp/checkpoint"
    checkpoint_callback = keras.callbacks.ModelCheckpoint(
        checkpoint_filepath,
        monitor="val_accuracy",
        save_best_only=True,
        save_weights_only=True,
    )

    history = model.fit(
        x=x_train,
        y=y_train_encoded,
        batch_size=batch_size,
        epochs=num_epochs,
        validation_split=0.1,
        callbacks=[checkpoint_callback],
    )

    model.load_weights(checkpoint_filepath)
    _, accuracy, top_5_accuracy = model.evaluate(x_test, y_test_encoded)
    print(f"Test accuracy: {round(accuracy * 100, 2)}%")
    print(f"Test top 5 accuracy: {round(top_5_accuracy * 100, 2)}%")

    return history


vit_classifier = create_vit_classifier()
history = run_experiment(vit_classifier)
```

```
1/1 [==============================] - 1s 849ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - top-5-accuracy: 1.0000 - val_loss: 53.5429 - val_accuracy: 0.0000e+00 - val_top-5-ac
Epoch 92/100
1/1 [==============================] - 1s 854ms/step - loss: 2.3842e-08 - accuracy: 1.0000 - top-5-accuracy: 1.0000 - val_loss: 52.8731 - val_accuracy: 0.0000e+00 - val_top-5-ac
Epoch 93/100
1/1 [==============================] - 1s 846ms/step - loss: 1.9417e-04 - accuracy: 1.0000 - top-5-accuracy: 1.0000 - val_loss: 52.3590 - val_accuracy: 0.0000e+00 - val_top-5-ac
Epoch 94/100
1/1 [==============================] - 1s 850ms/step - loss: 6.6517e-06 - accuracy: 1.0000 - top-5-accuracy: 1.0000 - val_loss: 51.8561 - val_accuracy: 0.0000e+00 - val_top-5-ac
Epoch 95/100
1/1 [==============================] - 1s 835ms/step - loss: 2.8020 - accuracy: 0.8000 - top-5-accuracy: 1.0000 - val_loss: 48.1744 - val_accuracy: 0.0000e+00 - val_top-5-accura
Epoch 96/100
1/1 [==============================] - 1s 842ms/step - loss: 0.0606 - accuracy: 1.0000 - top-5-accuracy: 1.0000 - val_loss: 45.2737 - val_accuracy: 0.0000e+00 - val_top-5-accura
Epoch 97/100
1/1 [==============================] - 1s 849ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - top-5-accuracy: 1.0000 - val_loss: 43.3523 - val_accuracy: 0.0000e+00 - val_top-5-ac
Epoch 98/100
1/1 [==============================] - 1s 1s/step - loss: 0.0011 - accuracy: 1.0000 - top-5-accuracy: 1.0000 - val_loss: 42.2914 - val_accuracy: 0.0000e+00 - val_top-5-accuracy:
Epoch 99/100
1/1 [==============================] - 1s 1s/step - loss: 1.2517e-06 - accuracy: 1.0000 - top-5-accuracy: 1.0000 - val_loss: 41.5260 - val_accuracy: 0.0000e+00 - val_top-5-accur
Epoch 100/100
1/1 [==============================] - 1s 1s/step - loss: 0.0000e+00 - accuracy: 1.0000 - top-5-accuracy: 1.0000 - val_loss: 40.8931 - val_accuracy: 0.0000e+00 - val_top-5-accur
1/1 [==============================] - 0s 105ms/step - loss: 3.2327 - accuracy: 0.6667 - top-5-accuracy: 0.6667
Test accuracy: 66.67%
Test top 5 accuracy: 66.67%
```

```
print(type(x_train))
```

```
<class 'numpy.ndarray'>
```

```
print(type(y_train_encoded))
```

```
<class 'numpy.ndarray'>
```

```
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np

# Load your image
image_path = "/home/DATA/Safeimagekit-resized-img (1).jpg"  # Update with the path to your image
image = load_img(image_path, target_size=(32, 32))  # Resize to match your model's input size
image_array = img_to_array(image)

# Preprocess the image (same as you did for training data)
image_array = image_array / 255.0  # Normalize pixel values (if that's what you did during training)
image_array = np.expand_dims(image_array, axis=0)  # Add a batch dimension

# Use your locally defined model to make predictions
predictions = vit_classifier.predict(image_array)
predicted_class = np.argmax(predictions)

# Print the predicted class (and the associated class label if you have one)
print(f"Predicted class index: {predicted_class}")
# If you have a list of class labels, you can use it to get the class name
# class_labels = ["Healthy", "Unhealthy"]
# predicted_class_label = class_labels[predicted_class]
# print(f"Predicted class: {predicted_class_label}")
```

```
1/1 [==============================] - 1s 1s/step
Predicted class index: 80
```

```
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np

# Load your image
image_path = "/home/TEST/class_2_image (15).jpg"  # Update with the path to your image
image = load_img(image_path, target_size=(32, 32))  # Resize to match your model's input size
image_array = img_to_array(image)

# Preprocess the image (same as you did for training data)
image_array = image_array / 255.0  # Normalize pixel values (if that's what you did during training)
image_array = np.expand_dims(image_array, axis=0)  # Add a batch dimension

# Use your locally defined model to make predictions
predictions = vit_classifier.predict(image_array)
predicted_class = np.argmax(predictions)

# Print the predicted class (and the associated class label if you have one)
print(f"Predicted class index: {predicted_class}")
# If you have a list of class labels, you can use it to get the class name
# class_labels = ["Healthy", "Unhealthy"]
# predicted_class_label = class_labels[predicted_class]
# print(f"Predicted class: {predicted_class_label}")
```

```
1/1 [==============================] - 0s 55ms/step
Predicted class index: 80
```