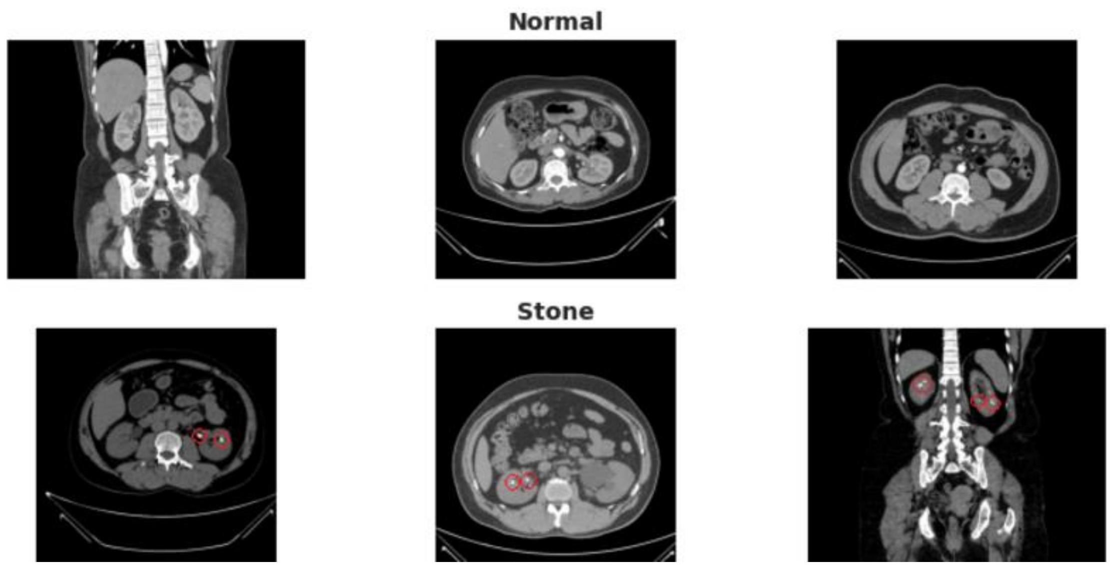
CT KIDNEY DATASET: Normal-Cyst-Tumor and Stone Prediction using Vision Transformer and Swin Transformer




```python
import numpy as np
import pandas as pd
import os

base_path = '/kaggle/input/ct-kidney-dataset-normal-cyst-tumor-and-stone/CT-KIDNEY-DATASET-Normal-Cyst-Tumor-Stone/CT-KIDNEY-DATASET-Normal-Cyst-Tumor-Stone'
categories = ["Cyst", "Normal", "Stone", "Tumor"]

image_paths = []
labels = []

for category in categories:
    category_path = os.path.join(base_path, category)
    for image_name in os.listdir(category_path):
        image_path = os.path.join(category_path, image_name)
        image_paths.append(image_path)
        labels.append(category)

df = pd.DataFrame({
    "image_path": image_paths,
    "label": labels
})

df.head()
```

```
                                          image_path label
0  /kaggle/input/ct-kidney-dataset-normal-cyst-tu...  Cyst
```

```
1  /kaggle/input/ct-kidney-dataset-normal-cyst-tu...  Cyst
2  /kaggle/input/ct-kidney-dataset-normal-cyst-tu...  Cyst
3  /kaggle/input/ct-kidney-dataset-normal-cyst-tu...  Cyst
4  /kaggle/input/ct-kidney-dataset-normal-cyst-tu...  Cyst
```

```
df.tail()
```

```
                                              image_path  label
12441  /kaggle/input/ct-kidney-dataset-normal-cyst-tu...  Tumor
12442  /kaggle/input/ct-kidney-dataset-normal-cyst-tu...  Tumor
12443  /kaggle/input/ct-kidney-dataset-normal-cyst-tu...  Tumor
12444  /kaggle/input/ct-kidney-dataset-normal-cyst-tu...  Tumor
12445  /kaggle/input/ct-kidney-dataset-normal-cyst-tu...  Tumor
```

```
df.columns
```

```
Index(['image_path', 'label'], dtype='object')
```

```
df.shape
```

```
(12446, 2)
```

```
df.duplicated().sum()
```

```
0
```

```
df.isnull().sum()
```

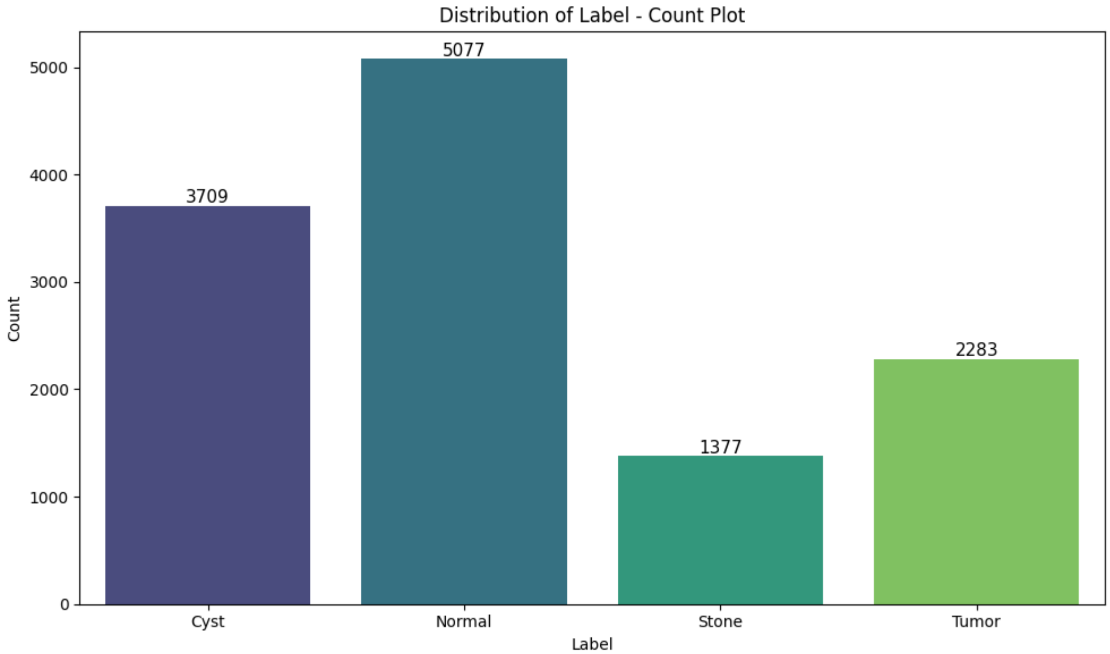
```
image_path    0
label         0
dtype: int64
```

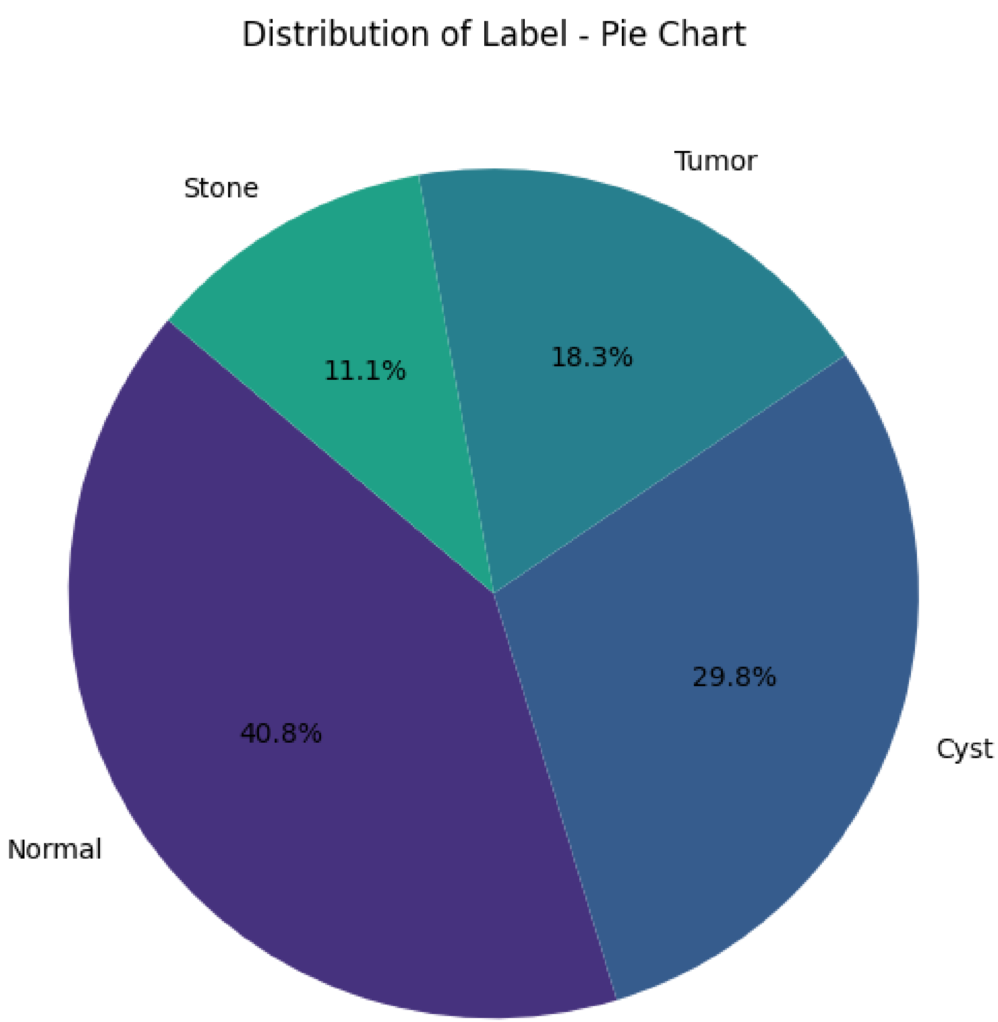
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12446 entries, 0 to 12445
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   image_path  12446 non-null  object
 1   label       12446 non-null  object
dtypes: object(2)
memory usage: 194.6+ KB
```

```
import seaborn as sns
import matplotlib.pyplot as plt

def visualize_label_distribution(df, label_column="label", figsize=(10, 6), 
palette="viridis"):
    """
    Visualizes the distribution of labels in a DataFrame using count and pie 
charts.
```

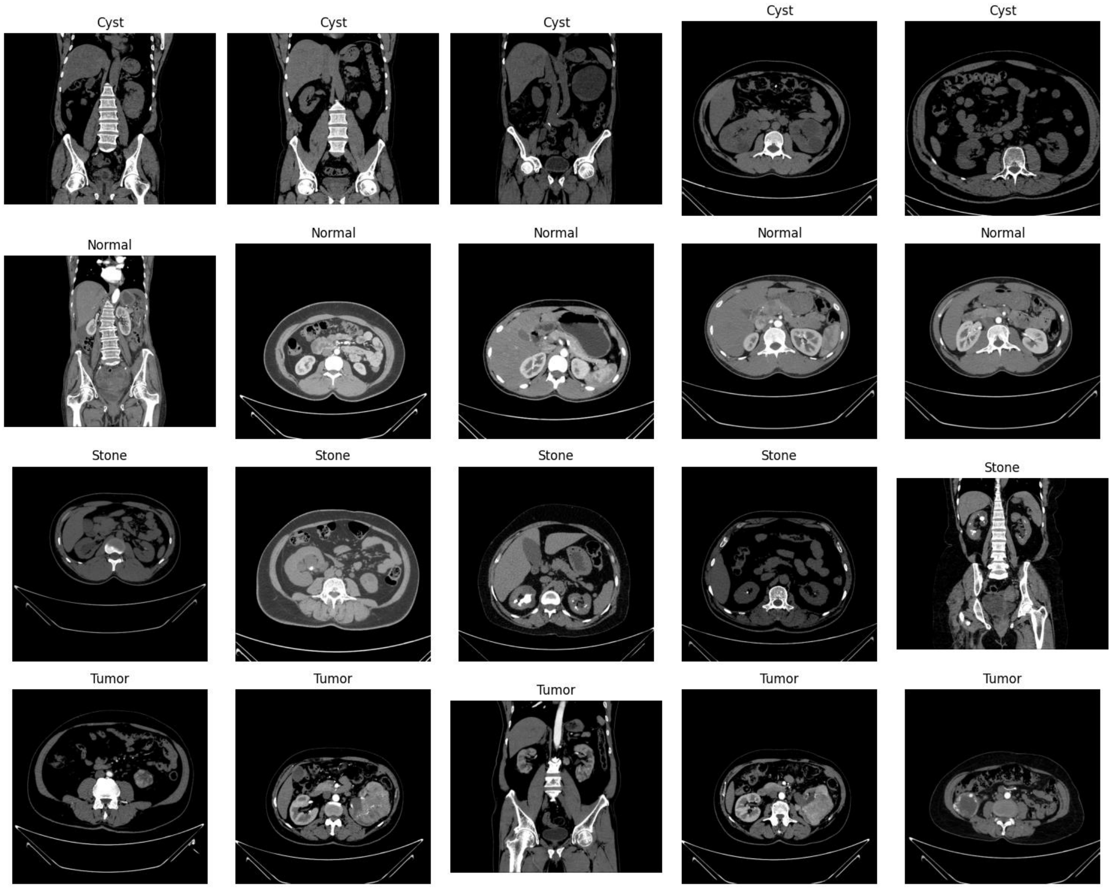
```
    Args:
        df (pd.DataFrame): The DataFrame containing the label data.
        label_column (str): The name of the column containing the labels.
Defaults to "label".
        figsize (tuple): The figure size for the plots. Defaults to (10, 6).
        palette (str): The color palette to use. Defaults to "viridis".
    """

    plt.figure(figsize=figsize)
    ax = sns.countplot(data=df, x=label_column, palette=palette)
    plt.title(f"Distribution of {label_column.capitalize()} - Count Plot")
    plt.xlabel(label_column.capitalize())
    plt.ylabel("Count")

    for p in ax.patches:
        ax.annotate(f'{int(p.get_height())}',
                    (p.get_x() + p.get_width() / 2., p.get_height()),
                    ha='center', va='center', fontsize=11, color='black',
xytext=(0, 5),
                    textcoords='offset points')
    plt.tight_layout() # Prevents labels from being cut off
    plt.show()


    label_counts = df[label_column].value_counts()
    plt.figure(figsize=figsize)
    plt.pie(label_counts, labels=label_counts.index, autopct='%1.1f%%',
startangle=140, colors=sns.color_palette(palette))
    plt.title(f"Distribution of {label_column.capitalize()} - Pie Chart")
    plt.tight_layout()
    plt.show()

visualize_label_distribution(df)
```


Distribution of Label - Count Plot
Count
Label
5000
4000
3000
2000
1000
0
5077
3709
2283
1377
Cyst
Normal
Stone
Tumor




```
import cv2

num_images = 5

plt.figure(figsize=(15, 12))

for i, category in enumerate(categories):
    category_images = df[df['label'] ==
category]['image_path'].iloc[:num_images]

    for j, img_path in enumerate(category_images):

        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

```
        plt.subplot(len(categories), num_images, i * num_images + j + 1)
        plt.imshow(img)
        plt.axis('off')
        plt.title(category)

plt.tight_layout()
plt.show()
```




```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()

df['category_encoded'] = label_encoder.fit_transform(df['label'])

df = df[['image_path', 'category_encoded']]

from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler(random_state=42)
X_resampled, y_resampled = ros.fit_resample(df[['image_path']],
df['category_encoded'])
```

```
df_resampled = pd.DataFrame(X_resampled, columns=['image_path'])
df_resampled['category_encoded'] = y_resampled

print("\nClass distribution after oversampling:")
print(df_resampled['category_encoded'].value_counts())
```

```
Class distribution after oversampling:
category_encoded
0    5077
1    5077
2    5077
3    5077
Name: count, dtype: int64
```

```
df_resampled
```

```
                                              image_path  category_encoded
0      /kaggle/input/ct-kidney-dataset-normal-cyst-tu...                 0
1      /kaggle/input/ct-kidney-dataset-normal-cyst-tu...                 0
2      /kaggle/input/ct-kidney-dataset-normal-cyst-tu...                 0
3      /kaggle/input/ct-kidney-dataset-normal-cyst-tu...                 0
4      /kaggle/input/ct-kidney-dataset-normal-cyst-tu...                 0
...                                                  ...               ...
20303  /kaggle/input/ct-kidney-dataset-normal-cyst-tu...                 3
20304  /kaggle/input/ct-kidney-dataset-normal-cyst-tu...                 3
20305  /kaggle/input/ct-kidney-dataset-normal-cyst-tu...                 3
20306  /kaggle/input/ct-kidney-dataset-normal-cyst-tu...                 3
20307  /kaggle/input/ct-kidney-dataset-normal-cyst-tu...                 3

[20308 rows x 2 columns]
```

```
df_resampled['category_encoded'] =
df_resampled['category_encoded'].astype(str)

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
Activation, Dropout, BatchNormalization
from tensorflow.keras import regularizers

import warnings
warnings.filterwarnings("ignore")

print ('check')
```

check

```
train_df_new, temp_df_new = train_test_split(
    df_resampled,
    train_size=0.8,
    shuffle=True,
    random_state=42,
    stratify=df_resampled['category_encoded']
)

valid_df_new, test_df_new = train_test_split(
    temp_df_new,
    test_size=0.5,
    shuffle=True,
    random_state=42,
    stratify=temp_df_new['category_encoded']
)

batch_size = 16
img_size = (224, 224)
channels = 3
img_shape = (img_size[0], img_size[1], channels)

tr_gen = ImageDataGenerator(rescale=1./255)
ts_gen = ImageDataGenerator(rescale=1./255)

train_gen_new = tr_gen.flow_from_dataframe(
    train_df_new,
    x_col='image_path',
    y_col='category_encoded',
    target_size=img_size,
    class_mode='sparse',
    color_mode='rgb',
    shuffle=True,
    batch_size=batch_size
)

valid_gen_new = ts_gen.flow_from_dataframe(
    valid_df_new,
    x_col='image_path',
    y_col='category_encoded',
    target_size=img_size,
    class_mode='sparse',
    color_mode='rgb',
    shuffle=True,
    batch_size=batch_size
)

test_gen_new = ts_gen.flow_from_dataframe(
    test_df_new,
```

```
    x_col='image_path',
    y_col='category_encoded',
    target_size=img_size,
    class_mode='sparse',
    color_mode='rgb',
    shuffle=False,
    batch_size=batch_size
)
```

```
Found 16246 validated image filenames belonging to 4 classes.
Found 2031 validated image filenames belonging to 4 classes.
Found 2031 validated image filenames belonging to 4 classes.
```

```
import tensorflow as tf

print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
Num GPUs Available:  2
```

```
gpus = tf.config.list_physical_devices('GPU')
if gpus:
    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
        print("GPU is set for TensorFlow")
    except RuntimeError as e:
        print(e)
```

```
Physical devices cannot be modified after being initialized
```

```
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)

from tensorflow.keras import layers

class PatchEmbedding(layers.Layer):
    def __init__(self, patch_size, embed_dim):
        super(PatchEmbedding, self).__init__()
        self.patch_size = patch_size
        self.embed_dim = embed_dim
        self.proj = layers.Conv2D(embed_dim, patch_size, strides=patch_size,
padding='valid')

    def call(self, images):
        patches = self.proj(images)
        patches = tf.reshape(patches, (tf.shape(patches)[0], -1,
self.embed_dim))
        return patches
```

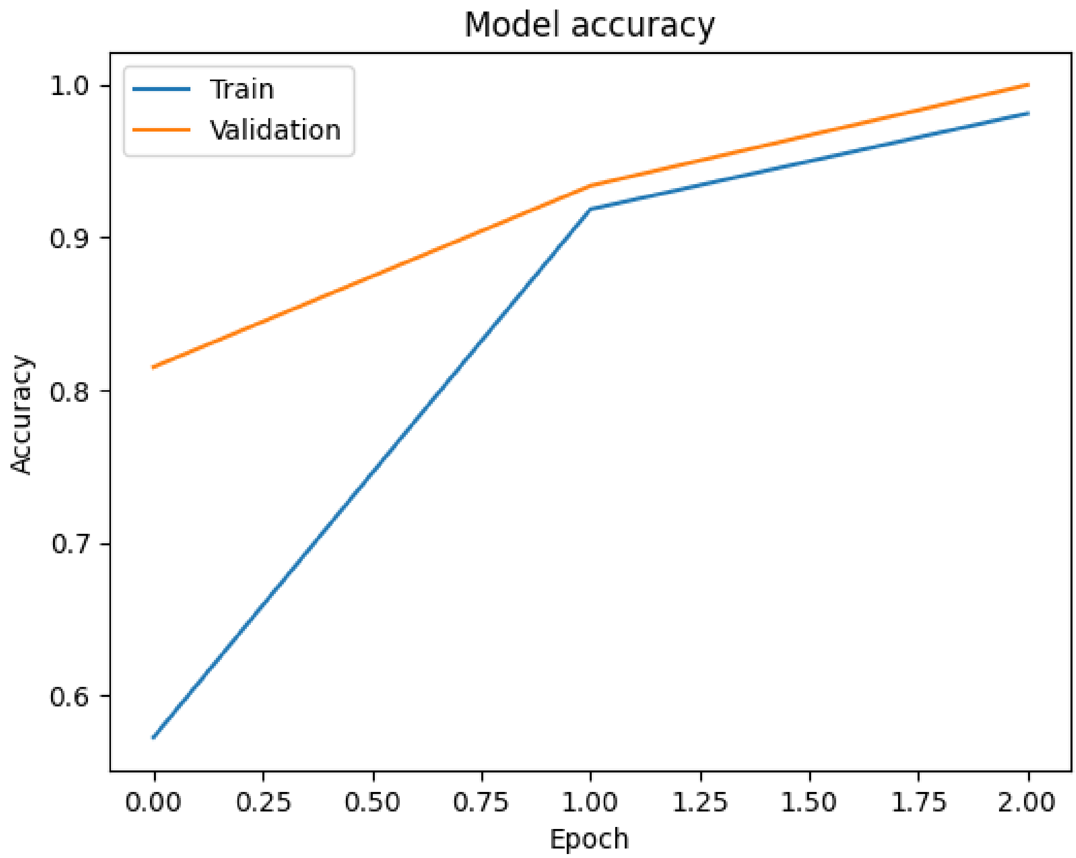
```python
class MultiHeadSelfAttention(layers.Layer):
    def __init__(self, num_heads, embed_dim):
        super(MultiHeadSelfAttention, self).__init__()
        self.attention = layers.MultiHeadAttention(num_heads=num_heads,
key_dim=embed_dim)

    def call(self, inputs):
        return self.attention(inputs, inputs)

class TransformerBlock(layers.Layer):
    def __init__(self, embed_dim, num_heads, mlp_dim, dropout_rate):
        super(TransformerBlock, self).__init__()
        self.attention = MultiHeadSelfAttention(num_heads, embed_dim)
        self.layernorm1 = layers.LayerNormalization(epsilon=1e-6)
        self.layernorm2 = layers.LayerNormalization(epsilon=1e-6)
        self.mlp = tf.keras.Sequential([
            layers.Dense(mlp_dim, activation='gelu'),
            layers.Dropout(dropout_rate),
            layers.Dense(embed_dim),
            layers.Dropout(dropout_rate)
        ])

    def call(self, inputs):
        x = self.layernorm1(inputs)
        x = self.attention(x)
        x = x + inputs
        x = self.layernorm2(x)
        x = self.mlp(x)
        return x + inputs

class VisionTransformer(tf.keras.Model):
    def __init__(self, image_size, patch_size, embed_dim, num_heads,
num_blocks, mlp_dim, num_classes, dropout_rate=0.1):
        super(VisionTransformer, self).__init__()
        self.patch_embed = PatchEmbedding(patch_size, embed_dim)

        height, width, _ = image_size
        num_patches = (height // patch_size) * (width // patch_size)

        self.pos_embed = self.add_weight(
            name="pos_embed",
            shape=(1, num_patches + 1, embed_dim),
            initializer=tf.initializers.RandomNormal(stddev=0.02),
            trainable=True
        )

        self.cls_token = self.add_weight(
            name="cls_token",
            shape=(1, 1, embed_dim),
```

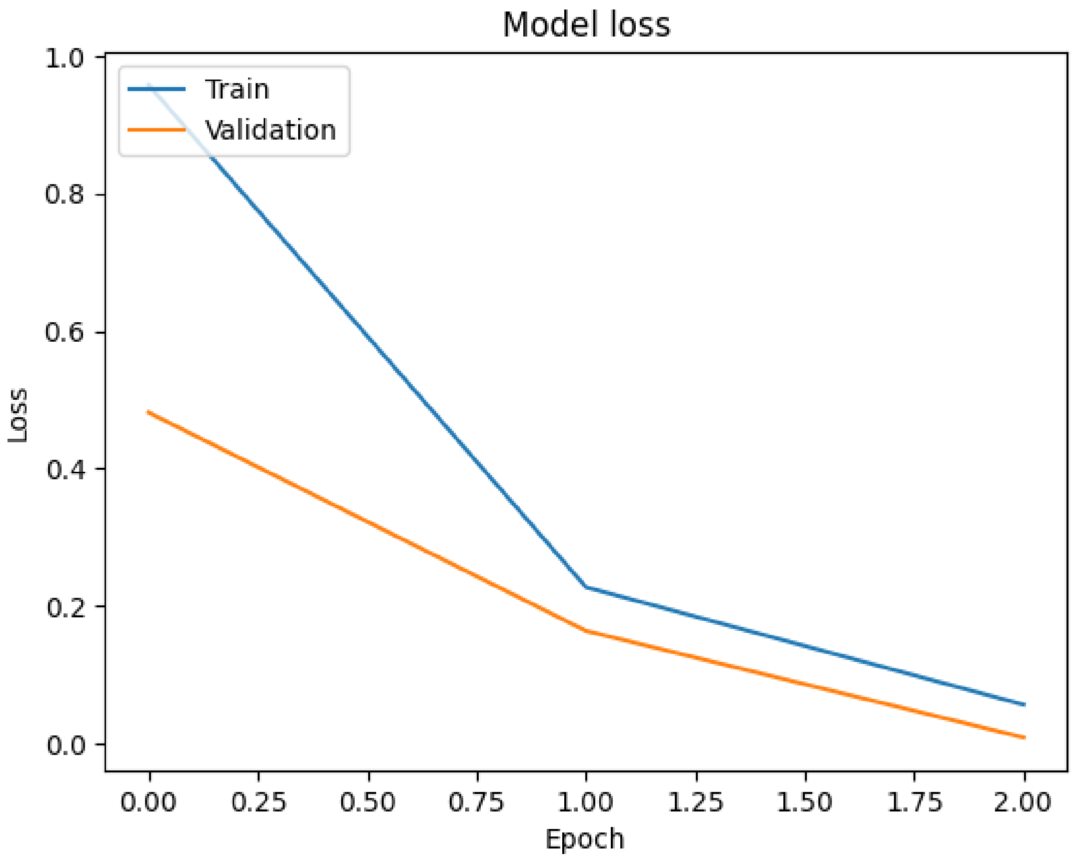
```
            initializer=tf.initializers.RandomNormal(stddev=0.02),
            trainable=True
        )

        self.dropout = layers.Dropout(dropout_rate)
        self.transformer_blocks = [TransformerBlock(embed_dim, num_heads,
mlp_dim, dropout_rate) for _ in range(num_blocks)]
        self.layernorm = layers.LayerNormalization(epsilon=1e-6)
        self.classifier = layers.Dense(num_classes, activation='softmax')

    def call(self, images):
        batch_size = tf.shape(images)[0]
        patches = self.patch_embed(images)

        cls_tokens = tf.repeat(self.cls_token, repeats=batch_size, axis=0)
        x = tf.concat([cls_tokens, patches], axis=1)

        pos_embed = tf.repeat(self.pos_embed, repeats=batch_size, axis=0)
        x = x + pos_embed
        x = self.dropout(x)

        for block in self.transformer_blocks:
            x = block(x)

        x = self.layernorm(x)
        cls_token_final = x[:, 0]
        return self.classifier(cls_token_final)


image_size = (224, 224, 3)
patch_size = 16
embed_dim = 256
num_heads = 8
num_blocks = 6
mlp_dim = 256
num_classes = 4
dropout_rate = 0.1
learning_rate = 1e-5

vit_model = VisionTransformer(image_size=image_size,
                              patch_size=patch_size,
                              embed_dim=embed_dim,
                              num_heads=num_heads,
                              num_blocks=num_blocks,
                              mlp_dim=mlp_dim,
                              num_classes=num_classes,
                              dropout_rate=dropout_rate)

vit_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_r
```

```
ate),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

epochs = 3
history = vit_model.fit(train_gen_new, epochs=epochs, batch_size = 32,
validation_data=valid_gen_new)
```

```
Epoch 1/3
1016/1016 ━━━━━━━━━━━━━━━━━━━━ 215s 185ms/step - accuracy: 0.4294 - loss:
1.2068 - val_accuracy: 0.8149 - val_loss: 0.4814
Epoch 2/3
1016/1016 ━━━━━━━━━━━━━━━━━━━━ 177s 173ms/step - accuracy: 0.8871 - loss:
0.3178 - val_accuracy: 0.9335 - val_loss: 0.1640
Epoch 3/3
1016/1016 ━━━━━━━━━━━━━━━━━━━━ 178s 174ms/step - accuracy: 0.9741 - loss:
0.0756 - val_accuracy: 0.9995 - val_loss: 0.0091
```

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```


Model accuracy
Train
Validation
1.0
0.9
0.8
0.7
0.6
Accuracy
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Epoch




```
test_labels = test_gen_new.classes
predictions = vit_model.predict(test_gen_new)
predicted_classes = np.argmax(predictions, axis=1)
```

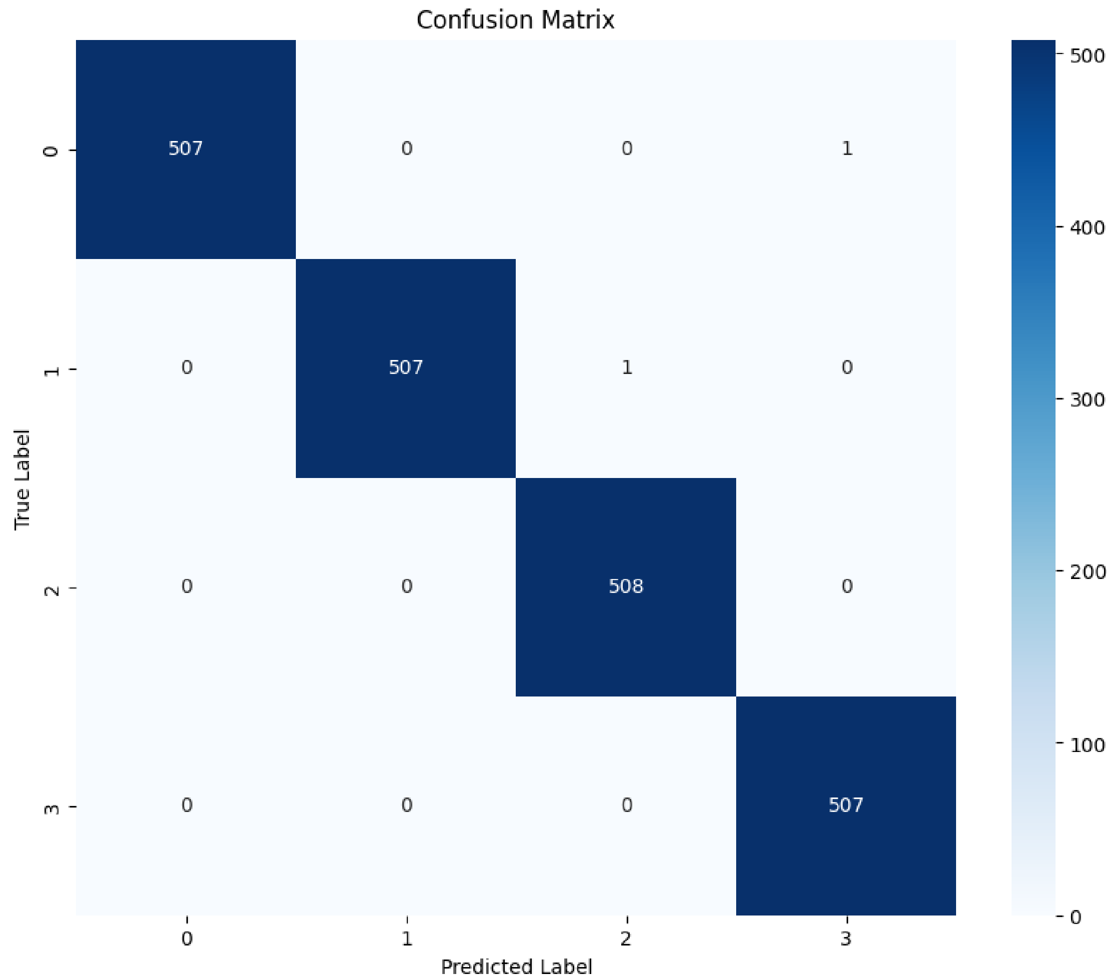
```
127/127 ━━━━━━━━━━━━━━━━━━━━ 16s 106ms/step
```

```
report = classification_report(test_labels, predicted_classes,
target_names=list(test_gen_new.class_indices.keys()))
print(report)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       508
           1       1.00      1.00      1.00       508
           2       1.00      1.00      1.00       508
           3       1.00      1.00      1.00       507

    accuracy                           1.00      2031
   macro avg       1.00      1.00      1.00      2031
weighted avg       1.00      1.00      1.00      2031
```

```
conf_matrix = confusion_matrix(test_labels, predicted_classes)
```

```
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
xticklabels=list(test_gen_new.class_indices.keys()),
yticklabels=list(test_gen_new.class_indices.keys()))
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```




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

class WindowAttention(layers.Layer):
    def __init__(self, dim, num_heads, window_size):
        super().__init__()
        self.num_heads = num_heads
        self.scale = (dim // num_heads) ** -0.5
        self.qkv = layers.Dense(dim * 3, use_bias=False)
        self.proj = layers.Dense(dim)
```

```
        self.window_size = window_size

    def call(self, x):
        B, N, C = tf.shape(x)[0], tf.shape(x)[1], tf.shape(x)[2]
        qkv = self.qkv(x)
        qkv = tf.reshape(qkv, (B, N, 3, self.num_heads, C // self.num_heads))
        qkv = tf.transpose(qkv, [2, 0, 3, 1, 4])
        q, k, v = qkv[0], qkv[1], qkv[2]
        attn = tf.matmul(q, k, transpose_b=True) * self.scale
        attn = tf.nn.softmax(attn)
        x = tf.matmul(attn, v)
        x = tf.transpose(x, [0, 2, 1, 3])
        x = tf.reshape(x, (B, N, C))
        return self.proj(x)

class SwinTransformerBlock(layers.Layer):
    def __init__(self, dim, num_heads, window_size):
        super().__init__()
        self.norm1 = layers.LayerNormalization()
        self.attn = WindowAttention(dim, num_heads, window_size)
        self.norm2 = layers.LayerNormalization()
        self.mlp = keras.Sequential([
            layers.Dense(dim * 4, activation='gelu'),
            layers.Dense(dim)
        ])

    def call(self, x):
        x = x + self.attn(self.norm1(x))
        x = x + self.mlp(self.norm2(x))
        return x

class SwinTransformer(layers.Layer):
    def __init__(self, input_shape, patch_size=4, embed_dim=96, num_heads=3,
window_size=7):
        super().__init__()
        self.patch_embed = layers.Conv2D(embed_dim, kernel_size=patch_size,
strides=patch_size, padding='same')
        self.swin_block = SwinTransformerBlock(embed_dim, num_heads,
window_size)
        self.pool = layers.GlobalAveragePooling1D()
        self.fc = layers.Dense(4, activation='softmax')

    def call(self, x):
        x = self.patch_embed(x)
        x = tf.reshape(x, (tf.shape(x)[0], -1, x.shape[-1]))
        x = self.swin_block(x)
        x = self.pool(x)
        return self.fc(x)
```

```python
input_shape = (224, 224, 3)
model = keras.Sequential([
    layers.Input(shape=input_shape),
    SwinTransformer(input_shape)
])

model.summary()
```

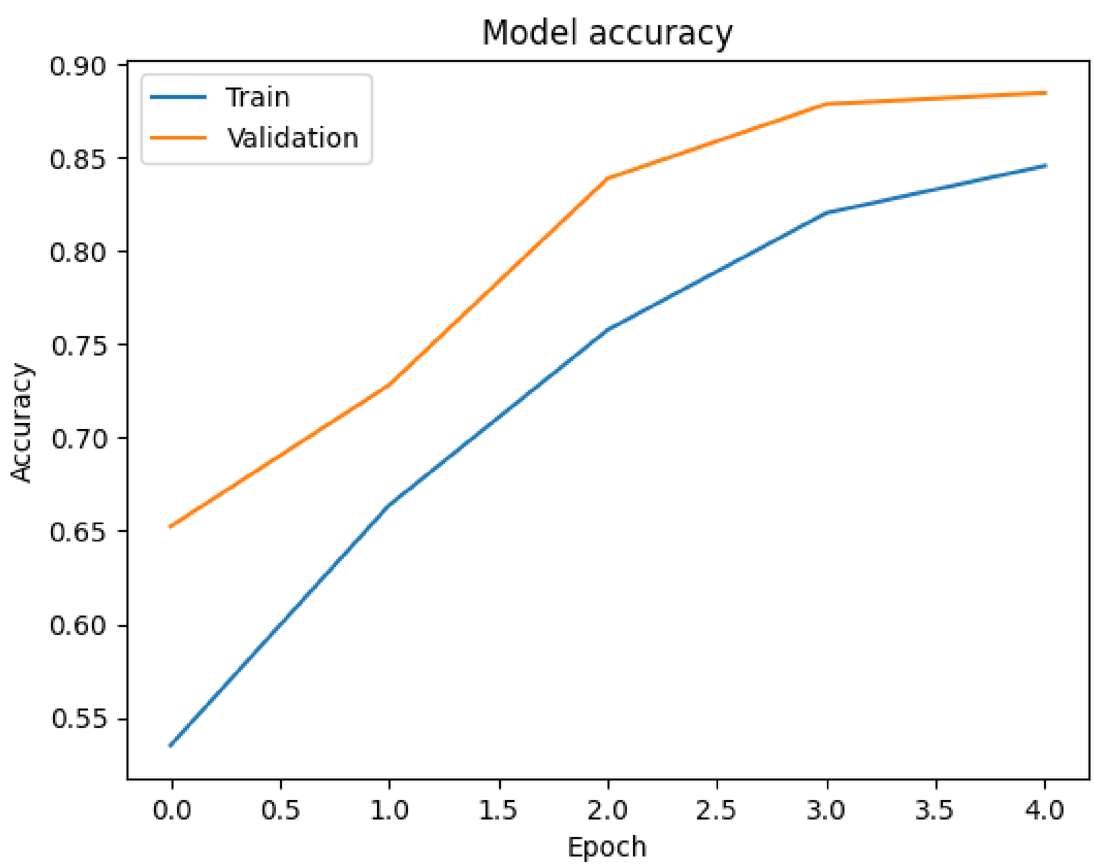
```
Model: "sequential_57"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━┓
┃ Layer (type)                          ┃ Output Shape                  ┃     Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━┩
│ swin_transformer_12                   │ (None, 4)                     │     116,644 │
│ (SwinTransformer)                     │                               │             │
└───────────────────────────────────────┴───────────────────────────────┴─────────────┘

 Total params: 116,644 (455.64 KB)

 Trainable params: 116,644 (455.64 KB)

 Non-trainable params: 0 (0.00 B)
```

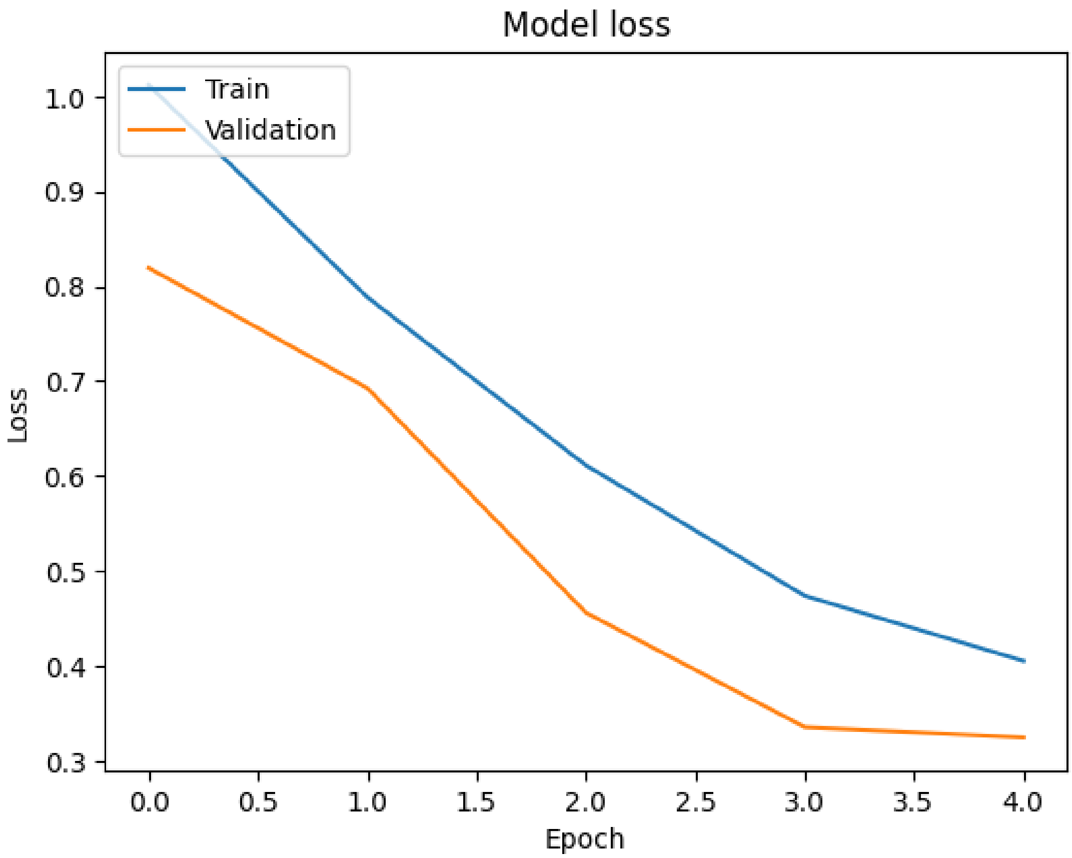
```python
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(
    train_gen_new,
    validation_data=valid_gen_new,
    epochs=5
)
```

```
Epoch 1/5
1016/1016 ━━━━━━━━━━━━━━━━━━━━ 215s 200ms/step - accuracy: 0.4505 - loss: 1.1858 - val_accuracy: 0.6524 - val_loss: 0.8194
Epoch 2/5
1016/1016 ━━━━━━━━━━━━━━━━━━━━ 194s 191ms/step - accuracy: 0.6477 - loss: 0.8107 - val_accuracy: 0.7282 - val_loss: 0.6928
Epoch 3/5
1016/1016 ━━━━━━━━━━━━━━━━━━━━ 198s 194ms/step - accuracy: 0.7376 - loss: 0.6505 - val_accuracy: 0.8390 - val_loss: 0.4559
Epoch 4/5
1016/1016 ━━━━━━━━━━━━━━━━━━━━ 195s 191ms/step - accuracy: 0.8061 - loss: 0.4997 - val_accuracy: 0.8789 - val_loss: 0.3353
```

```
Epoch 5/5
1016/1016 ━━━━━━━━━━━━━━━━━━━━ 194s 191ms/step - accuracy: 0.8396 - loss:
0.4155 - val_accuracy: 0.8848 - val_loss: 0.3244
```

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```






```
test_labels = test_gen_new.classes
predictions = model.predict(test_gen_new)
predicted_classes = np.argmax(predictions, axis=1)
```

```
127/127 ━━━━━━━━━━━━━━━━━━━━ 11s 77ms/step
```

```
report = classification_report(test_labels, predicted_classes,
target_names=list(test_gen_new.class_indices.keys()))
print(report)
```

```
              precision    recall  f1-score   support

           0       0.91      0.83      0.87       508
           1       0.87      0.91      0.89       508
           2       0.89      0.87      0.88       508
           3       0.83      0.89      0.86       507

    accuracy                           0.87      2031
   macro avg       0.87      0.87      0.87      2031
weighted avg       0.87      0.87      0.87      2031
```

```
conf_matrix = confusion_matrix(test_labels, predicted_classes)
```

```
conf_matrix
```

```
array([[422,   5,  41,  40],
       [  0, 460,   8,  40],
       [ 13,  42, 440,  13],
       [ 29,  22,   5, 451]])
```

```
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
xticklabels=list(test_gen_new.class_indices.keys()),
yticklabels=list(test_gen_new.class_indices.keys()))
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```