

Colorectal Cancer Prediction Project

By:
Kadir Altunel, David Mosciszki, Bryam Piedra

Source Code:

https://drive.google.com/file/d/1OMEF2CKNI5kNUNIPUzhZOsYBqYXXa4NN/view?usp=drive_link

Video:

https://drive.google.com/file/d/1z0Ae28eAWFQOSCvQN1TJj89Dgf5uTNkU/view?usp=drive_link

Background

Colorectal Cancer is a type of cancer that originates in the colon or rectum and as the name entails, it is often grouped together due to its similarities. Both the colon and rectum are parts of a key component of the digestive system. The colon structure is as follows:

- Ascending Colon: Begins with the cecum and travels upward on the right side.
- Transverse Colon: Crosses the abdomen from right to left.
- Descending Colon: Moves downward on the left side.
- Sigmoid Colon: S-shaped, connects to the rectum.

Most of the time, colorectal cancer begins as polyps in the colon or rectum. Polyps are dangerous since they can become cancerous over time. Some types of polyps include: Adenomatous Polyps, Hyperplastic and Inflammatory Polyps, Sessile Serrated Polyps (SSP) and Traditional Serrated Adenomas (TSA). Some factors that increase the risk of cancer include polyp size, number, and dysplasia (abnormal cells). Cancer can also spread through the colon and rectum and into the blood which can spread to distant organs. There are different Types of Colorectal Cancer, with the most common being: Adenocarcinomas, and the less common: Carcinoid Tumors, Gastrointestinal Stromal Tumors (GISTs), Lymphomas and Sarcomas.

Just like Colorectal Cancer, there are other multiple types of cancer that are a leading cause of death in the United States. The cancer mortality rate in the United States is 146 per 100000 men and women per year. And one of the factors for this is cancer treatment delay. There was a study done to be able to quantify the association of cancer treatment delay and the conclusion was that it does have a big impact and is definitely a problem in the health system worldwide.

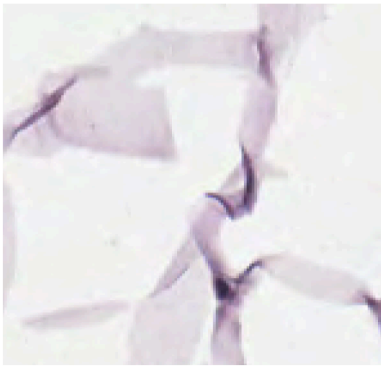
Luckily, with the advances in AI, Machine Learning and Deep Learning, there are models that can help with faster diagnosis and increased accuracy. So, overall, these models can be very helpful in enhancing decision-making in cancer prognosis and accelerate the treatment process as well.

The use of deep learning has been widely used in tasks such as: imaging diagnosis, digital pathology, and cancer prognosis. Cancer prognosis involves predicting cancer outcomes, recurrence probabilities, and patient survival estimates. In this project we will be focusing on Colorectal Cancer Prediction.

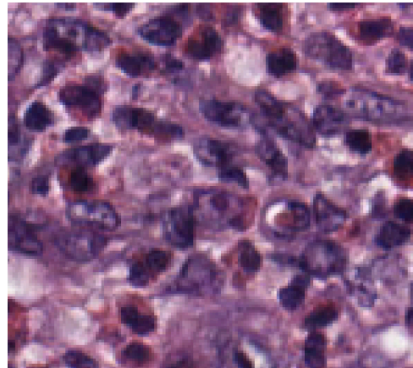
Dataset

The dataset used for this project is a colorectal cancer histology dataset, which was used along with a vision transformer model. The Colorectal cancer histology dataset contains 5000 histological images of 150x150 pixels. There are also 8 labels, which refer to the different textures in colorectal cancer histology. The labels are the following: Adipose, Complex, Debris, Empty, Lympho, Mucosa, Stroma, Tumor. Furthermore, the images in this dataset are RGB and were digitized with an Aperio ScanScope and magnification 20x. See below for examples of the images with the labels.

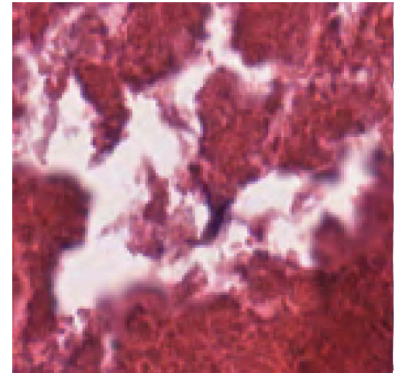
Adipose



Complex



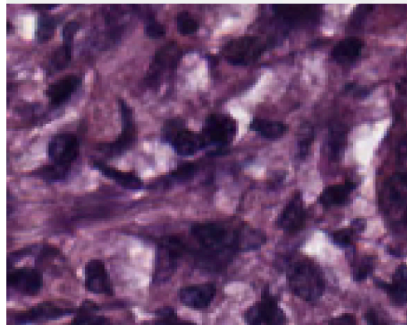
Debris



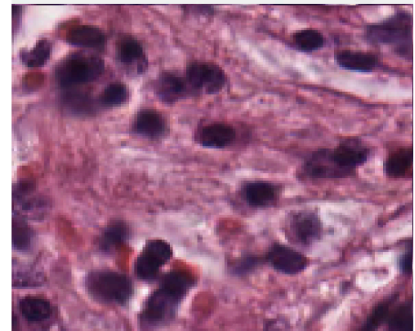
Empty



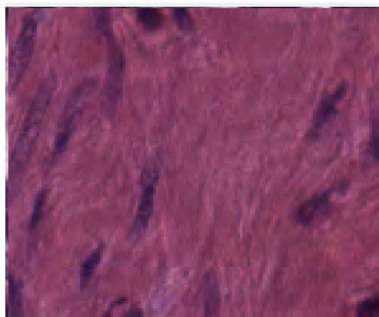
Lympho



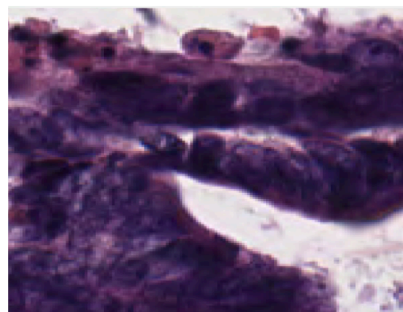
Mucosa



Stroma

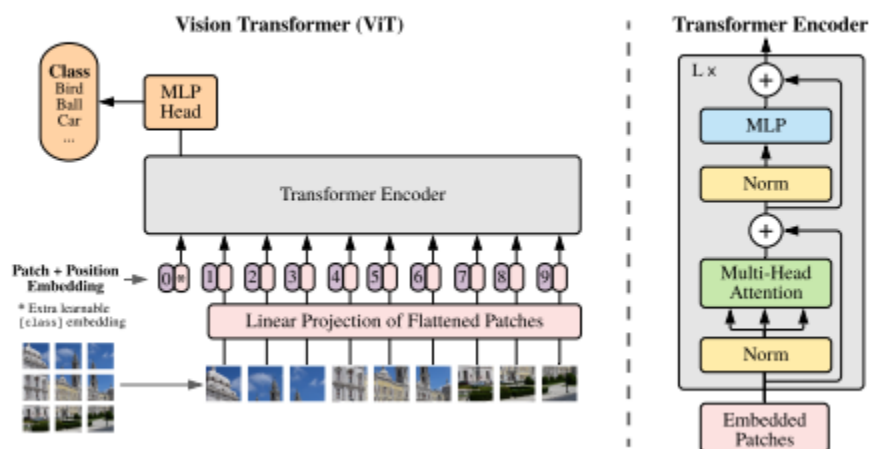


Tumor



VISION TRANSFORMER

The core idea of ViT is to treat an image as a sequence of patches. An image is divided into fixed-size patches (16x16 pixels), and each patch is then flattened into a vector. These vectors are linearly embedded and combined with positional embeddings to retain spatial information. This sequence of embedded patches is then fed into the standard Transformer encoder, which consists of layers of multi-head self-attention and feed-forward neural networks (FFNs). It is pre-trained on ImageNet-21k, which has 14 million images and 21843 classes at a resolution of 224x224.



From “An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale.”

The self-attention mechanism allows the model to weigh the importance of different patches relative to each other, enabling it to capture complex patterns and relationships across the entire image. Attention weights are interpreted as a probability distribution, which is how importance is chosen for the patches. The output from the Transformer encoder is then passed through a classification layer (linear) to produce the final class predictions^[1].

METHOD IMPLEMENTATION AND RESULTS

For the colon histology report, we incorporate a comprehensive collection of histology images labeled "ColonHistology" into our Python environment. By using the Google Drive API, we mount Google Drive to access this extensive image collection. Leveraging the os library, known for its powerful file manipulation features, we enumerate the files in the specified directory, allowing us to handle the images systematically.

```
image_directory = '/content/drive/MyDrive/ColonHistology'
```

```
files = os.listdir(image_directory)
files
```

Then, we generate a DataFrame from the histology image file names and their respective labels. We start by defining an empty list to store the data and a regular expression pattern to identify labels in the file names. Using the **os** library, we list all files in the specified directory and filter for those with a **.tif** extension. We then match the file names to our label pattern to extract the labels.

With the **pandas** library, we create a DataFrame containing columns for the file paths and their associated labels. To make the labels suitable for machine learning models, we utilize the **LabelEncoder** from the **sklearn** library, converting the string labels into numerical format. Finally, we construct a dictionary to map the encoded labels back to their original string representation and print the classes present in the dataset.

```
[ ] # Create DataFrame from file names and labels
data = []
label_pattern = re.compile(r'^[a-zA-Z]+')

for filename in os.listdir(image_directory):
    if filename.endswith('.tif'):
        match = label_pattern.match(filename)
        if match:
            label = match.group(0)
            full_path = os.path.join(image_directory, filename)
            data.append([full_path, label])

df = pd.DataFrame(data, columns=['file_path', 'class'])

label_encoder = LabelEncoder()
df['class'] = label_encoder.fit_transform(df['class'])
class_dict = dict(zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_)))
print("Classes in the dataset are:", class_dict)
```

We split the dataset into training, validation, and test sets to prepare it for machine learning model training and evaluation. We begin by using the **train_test_split** function from the **sklearn** library to divide the data into training/validation and test sets, with the test set comprising 15% of the total data. The split is stratified to ensure that the class distribution remains consistent across the sets.

Next, we further split the training/validation set into separate training and validation sets, with the validation set comprising approximately 5.88% of the total data. Again, we use stratified splitting to maintain the class distribution.

```
train_val_df, test_df = train_test_split(df, test_size=0.15, stratify=df['class'], random_state=42)

train_df, val_df = train_test_split(train_val_df, test_size=0.0588, stratify=train_val_df['class'], random_state=42)

print(f'Training set size: {len(train_df)}')
print(f'Validation set size: {len(val_df)}')
print(f'Test set size: {len(test_df)}')
```

We create a custom dataset class to handle the histology images and their corresponding labels. This class, named `ColonHistologyDataset`, inherits from the `Dataset` class in PyTorch. We start by initializing the class with the `DataFrame` containing the file paths and labels, and an optional transformation parameter.

The `__len__` method returns the number of samples in the dataset. The `__getitem__` method retrieves an image and its corresponding label by index, converts the image to RGB format, and applies any specified transformations.

We define a set of transformations using **`transforms.Compose`** to resize the images to 224x224 pixels and convert them to tensors. These transformations are applied to the training, validation, and test datasets.

```
class ColonHistologyDataset(Dataset):
    def __init__(self, dataframe, transform=None):
        self.dataframe = dataframe
        self.transform = transform

    def __len__(self):
        return len(self.dataframe)

    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()

        img_name = self.dataframe.iloc[idx, 0]
        image = Image.open(img_name).convert('RGB')
        class_label = self.dataframe.iloc[idx, 1]

        if self.transform:
            image = self.transform(image)

        return image, class_label

transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor()
])

[ ] train_dataset = ColonHistologyDataset(dataframe=train_df, transform=transform)
    val_dataset = ColonHistologyDataset(dataframe=val_df, transform=transform)
    test_dataset = ColonHistologyDataset(dataframe=test_df, transform=transform)
```

We go ahead and design a custom **Vision Transformer (ViT)** model for image classification. This class, named **`ViTForImageClassification`**, extends the **`nn.Module`** from PyTorch. We start by initializing the model with the specified number of labels, defaulting to 8.

In the initialization method, we load a pre-trained **ViT** model from the Hugging Face library and set up a dropout layer with a 50% dropout rate. We also define a linear classifier layer that maps the hidden states to the number of labels.

The forward method processes the input pixel values through the **ViT** model, applies dropout to the output, and then passes it through the classifier to get the logits. If labels are provided, the method computes the cross-entropy loss between the logits and the labels. The method returns the logits and the loss (if computed).

```

class ViTForImageClassification(nn.Module):
    def __init__(self, num_labels=8):
        super(ViTForImageClassification, self).__init__()
        self.vit = ViTModel.from_pretrained('google/vit-base-patch16-224-in21k')
        self.dropout = nn.Dropout(0.5)
        self.classifier = nn.Linear(self.vit.config.hidden_size, num_labels)
        self.num_labels = num_labels

    def forward(self, pixel_values, labels):
        outputs = self.vit(pixel_values=pixel_values)
        output = self.dropout(outputs.last_hidden_state[:,0])
        logits = self.classifier(output)

        loss = None
        if labels is not None:
            loss_fct = nn.CrossEntropyLoss()
            loss = loss_fct(logits.view(-1, self.num_labels), labels.view(-1))
        if loss is not None:
            return logits, loss.item()
        else:
            return logits, None

```

We set up the training parameters and initialize the **ViT** model for image classification. First, we define key hyperparameters, including the number of epochs, batch size, validation batch size, and learning rate.

Next, we identify the unique classes in the dataset to determine the number of labels for the model. The **ViTForImageClassification** model is then instantiated with the appropriate number of labels. We also utilize a pre-trained **ViT** feature extractor from the Hugging Face library.

We configure the optimizer using the Adam algorithm with the specified learning rate and define the loss function as cross-entropy loss. To leverage the computational power of GPUs, we check for CUDA availability and set the device accordingly, moving the model to the GPU if available.

```

EPOCHS = 50
BATCH_SIZE = 10
VAL_BATCH = 1
LEARNING_RATE = 0.000001

[ ] unique_classes = df['class'].unique()

model = ViTForImageClassification(num_labels=len(unique_classes))

feature_extractor = ViTFeatureExtractor.from_pretrained('google/vit-base-patch16-224-in21k', do_rescale=False)

optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)

loss_func = nn.CrossEntropyLoss()

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
if torch.cuda.is_available():
    model.cuda()

```

We set up the data loaders and lists to store metrics for training, validation, and testing the **ViT** model for image classification. Data loaders are configured for the training, validation, and test sets, each with specified batch sizes.

We initialize lists to store the losses and accuracies for the training, validation, and test sets. The training loop iterates over the specified number of epochs, processing each batch of training data by converting it to a **numpy** array, applying the feature extractor, and feeding it through the model. The optimizer updates the model parameters based on the calculated loss.

At the end of each epoch, we calculate the test loss and accuracy by evaluating the model on the test set without updating the model parameters. Similarly, we calculate the validation loss and accuracy by evaluating the model on the validation set.

The losses and accuracies are stored in their respective lists for plotting. Finally, we plot the training, test, and validation losses, as well as the test and validation accuracies, over the epochs.

```
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True, num_workers=0)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=True, num_workers=0)
val_loader = DataLoader(val_dataset, batch_size=VAL_BATCH, shuffle=True, num_workers=0)

# Lists to store metrics
train_losses = []
test_losses = []
val_losses = []
test_accuracies = []
val_accuracies = []

# Train the model
for epoch in range(EPOCHS):
    train_loss = 0.0
    for step, (x, y) in enumerate(train_loader):
        # Convert batch to numpy array
        x = np.array(x)
        # Apply feature extractor
        features = feature_extractor(x, return_tensors="pt")['pixel_values']
        # Send to GPU if available
        x, y = features.to(device), y.to(device)
        b_x = Variable(x) # batch x (image)
        b_y = Variable(y) # batch y (target)
        # Feed through model
        output, loss = model(b_x, None)
        # Calculate loss
        if loss is None:
            loss = loss_func(output, b_y)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
        train_loss += loss.item() if isinstance(loss, torch.Tensor) else loss

    # Calculate test loss and accuracy
    test_loss = 0.0
    test_correct = 0
    test_total = 0
    with torch.no_grad():
        for test_x, test_y in test_loader:
            # Reshape and get feature matrices as needed
            test_x = feature_extractor(np.array(test_x), return_tensors="pt")['pixel_values']
            test_x = test_x.to(device)
            test_y = test_y.to(device)
            # Generate prediction
            test_output, test_loss_batch = model(test_x, test_y)
            test_loss += test_loss_batch.item() if isinstance(test_loss_batch, torch.Tensor) else test_loss_batch
            # Predicted class value using argmax
            test_predicted_class = test_output.argmax(dim=1)
            test_correct += (test_predicted_class == test_y).sum().item()
            test_total += test_y.size(0)
    test_accuracy = test_correct / test_total
    avg_test_loss = test_loss / len(test_loader)
```

```

# Calculate validation loss and accuracy at the end of each epoch
val_loss = 0.0
val_correct = 0
val_total = 0
with torch.no_grad():
    for val_x, val_y in val_loader:
        # Reshape and get feature matrices as needed
        val_x = feature_extractor(np.array(val_x), return_tensors="pt")['pixel_values']
        val_x = val_x.to(device)
        val_y = val_y.to(device)
        # Generate prediction
        val_output, val_loss_batch = model(val_x, val_y)
        val_loss += val_loss_batch.item() if isinstance(val_loss_batch, torch.Tensor) else val_loss_batch
        # Predicted class value using argmax
        val_predicted_class = val_output.argmax(dim=-1)
        val_correct += (val_predicted_class == val_y).sum().item()
        val_total += val_y.size(0)
val_accuracy = val_correct / val_total
avg_train_loss = train_loss / len(train_loader)
avg_val_loss = val_loss / len(val_loader)

# Store metrics for plotting
train_losses.append(avg_train_loss)
test_losses.append(avg_test_loss)
val_losses.append(avg_val_loss)
test_accuracies.append(test_accuracy)
val_accuracies.append(val_accuracy)

print(f'Epoch: {epoch} | train loss: {avg_train_loss:.4f} | test loss: {avg_test_loss:.4f} | test accuracy: {test_accuracy:.2f} | val loss: {avg_val_loss:.4f} | val accuracy: {val_accuracy:.2f}')

```

```

# Plotting the results
epochs_range = range(EPOCHS)

plt.figure(figsize=(12, 8))
plt.subplot(2, 1, 1)
plt.plot(epochs_range, train_losses, label='Training Loss')
plt.plot(epochs_range, test_losses, label='Test Loss')
plt.plot(epochs_range, val_losses, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Loss Over Epochs')

plt.subplot(2, 1, 2)
plt.plot(epochs_range, test_accuracies, label='Test Accuracy')
plt.plot(epochs_range, val_accuracies, label='Validation Accuracy')
plt.legend(loc='upper left')
plt.title('Accuracy Over Epochs')

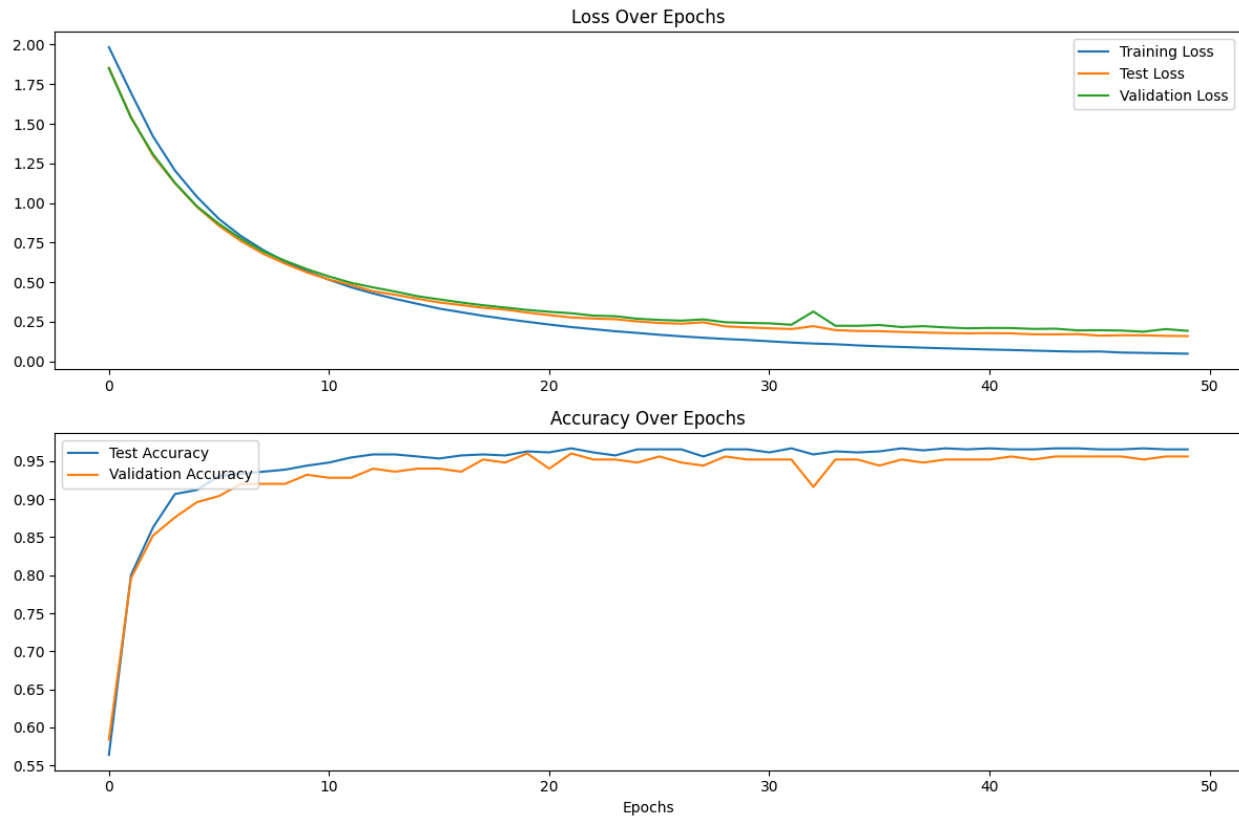
plt.xlabel('Epochs')
plt.tight_layout()
plt.show()

```

The output shows that at the end of the final epoch, the model achieved high test and validation accuracies of 97% and 96%, respectively.

Epoch: 49	train loss: 0.0486	test loss: 0.1598	test accuracy: 0.97	val loss: 0.1928	val accuracy: 0.96
-----------	--------------------	-------------------	---------------------	------------------	--------------------

The loss over epochs plot illustrates the training, test, and validation losses throughout the epochs. We can observe a steady decrease in all three losses, indicating that the model is effectively learning and enhancing its performance. The accuracy over epochs plot demonstrates the test and validation accuracies over the epochs. Both accuracies increase rapidly during the initial epochs and stabilize around 0.97 for the test set and 0.96 for the validation set, reflecting consistent and robust performance.



CONCLUSION

Recent machine learning advancements have allowed us to attempt training a newer model, the Vision Transformer (ViT), on a dataset of Colorectal cancer tissue images. Using the Colorectal cancer histology dataset, we can classify each of the textures of the cancerous tissue (as well as none, or the case of no cancer present).

In this experiment, we have demonstrated classification for colon histology images using a pre-trained ViT model. The custom ColonHistologyDataset class, built using PyTorch, enabled efficient handling and transformation of the images. This transformation is a simple resizing of the dataset to images of size 224x224 to match the size of the pre-trained model's dataset.

Our custom ViTForImageClassification model, using pre-trained ViT weights from Hugging Face, showed exceptional performance in image classification. The training process, utilizing hyperparameters and the Adam optimizer, effectively minimized the cross-entropy loss. The model achieved impressive test and validation accuracies of 97% and 96%, respectively. These results reflect the model's robustness and reliability in classifying colon histology images.

Vision Transformers have proved to be a valid classifier for this dataset. Future work could include different kinds of medical images to be trained, as well as other vision transformers and classifiers for in depth comparison. A more complex vision transformer model can also be developed, with more layers than used in this implementation.

BIBLIOGRAPHY

1. Dosovitskiy, Alexey, et al. "An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale." *arXiv.Org*, 3 June 2021, arxiv.org/abs/2010.11929.
2. Arun, Ishita, et al. "Automation in Histopathology: Present and Future." *Journal of Pathology Informatics*, vol. 11, 2020, PMC7139576.
3. "Colorectal Cancer: Epidemiology, Risk Factors, and Prognostic Factors." *BMJ*, 371, 2020, bmj.com/content/371/bmj.m4087.
4. "Colorectal Histology Dataset." TensorFlow Datasets, tensorflow.org/datasets/catalog/colorectal_histology.
5. "ViT Base Patch16 224 In21k." Hugging Face, huggingface.co/google/vit-base-patch16-224-in21k.