Colon Cancer Image Classification - Final Project

By: Mandy Liu, Mt. San Antonio College, CISB62 Fall 2023

About this Project

Colon cancer is one of the most common forms of cancer worldwide and is a leading cause of cancer-related deaths. Colon cancer is often asymptomatic in its early stages, making regular screenings and research vital for early detection. Early diagnosis can significantly improve survival rates.

Classifying colon cancer images using deep learning models can help aid in early detection. Deep learning models can identify subtle patterns and abnormalities in medical images that might not be discernible to the human eye. This can lead to early detection of colon cancer, increasing the chances of successful treatment and survival. A good model can achieve high levels of accuracy in image classification, reducing the likelihood of misdiagnosis and unnecessary medical procedures. Deep learning models can process large volumes of medical images quickly, allowing for more efficiency and timely diagnoses. Analyzing a vast amount of classified images can provide insights into the progression and behavior of colon cancer, aiding in the development of better treatment strategies and contributing to the broader field of oncology research. Deep learning models for classifying colon cancer images not only enhance diagnostic accuracy, but also improve the efficiency and consistency of the diagnostic process, ultimately leading to better patient outcomes and advancing our understanding of this disease.

About the Dataset

The LC25000 Lung and Colon Histopathological Image Dataset, created by Andrew A. Borkowski et al., comprises 25,000 color images categorized into 5 classes: colon adenocarcinomas, benign colonic tissues, lung adenocarcinomas, lung squamous cell carcinomas, and benign lung tissues. For this project, only the colon images were utilized. Due to limitations in computer resources, a total of 700 images (350 cancerous and 350 benign) were selected for training and model compilation.

Summary and Conclusion

In this project, I constructed two Deep Learning models to classify colon tissue images as cancerous or noncancerous. Due to computational constraints, only 700 out of 10,000 available images were used. I developed models using an Artificial Neural Network (ANN) and a Convolutional Neural Network (CNN) for comparative analysis.

For the ANN model, the confusion matrix reveals a 61% accuracy in predicting cancerous tissue images. It incorrectly classifies noncancerous tissue 55% of the time but correctly identifies noncancerous images 15% of the time. Additionally, it misclassifies cancerous images as noncancerous 9% of the time.

The ANN model acheived a 0.7571 accuracy score during training. After retraining the ANN model with hyperparameter tuning, the results indicate suboptimal performance on the training dataset. The elevated loss (0.6932) and near-random guessing accuracy (0.5) suggest inadequate learning or potential overfitting. Further model tuning or adjustments may be necessary to improve performance.

The CNN model, while achieving a high training accuracy of 90.67% in the last epoch (Epoch 5), exhibits a lower validation accuracy of 48.21%. This discrepancy suggests potential overfitting, where the model excels on the training set but struggles with new data. Strategies such as adjusting the learning rate, modifying network layers, employing transfer learning, and increasing the dataset size could enhance model performance.

Reference Links

```
https://github.com/mandyliu-1/CISB62 Link to my Github

https://academictorrents.com/details/7a638ed187a6180fd6e464b3666a6ea0499af4af

https://www.kaggle.com/code/abdelruhmanessam/breast-cancer-classification-using-cnn

https://www.kimshospitals.com/blog/colorectal-cancer/  Colorectal cancer banner image
```

Import Libraries

In [1]:
```
import os

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from PIL import Image

# import warnings
import warnings
warnings.filterwarnings('ignore')

# import tensorflow and keras
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.models import Sequential
from tensorflow.keras.layers import Dense , Conv2D , MaxPooling2D , Flatten , Activation , Dropout
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.optimizers import Adam , Adamax

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

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import glob
```

```
2023-12-07 19:59:35.219182: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized t
o use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler
flags.
```

Load the Dataset

In [2]:
```
# Define the data folder path
# Set the directory paths for the image data
cancer_dir = "/Users/mmliu/Desktop/CISB62_Final_MandyLiu/data/cancer"
no_cancer_dir = "/Users/mmliu/Desktop/CISB62_Final_MandyLiu/data/cancer_none"
```

In [3]:
```
# Create Python List objects to store the files and labels
file_paths = []
labels = []

# Load images with cancer (label=1)
cancer_files = glob.glob(os.path.join(cancer_dir, "*.jpeg"))
file_paths.extend(cancer_files)
labels.extend([1] * len(cancer_files))

# Load images with no cancer (label=0)
no_cancer_files = glob.glob(os.path.join(no_cancer_dir, "*.jpeg"))
file_paths.extend(no_cancer_files)
labels.extend([0] * len(no_cancer_files))
```

In [4]:
```
# Combine the image file paths and labels into a list of (path, label) pairs
data = list(zip(file_paths, labels))
```

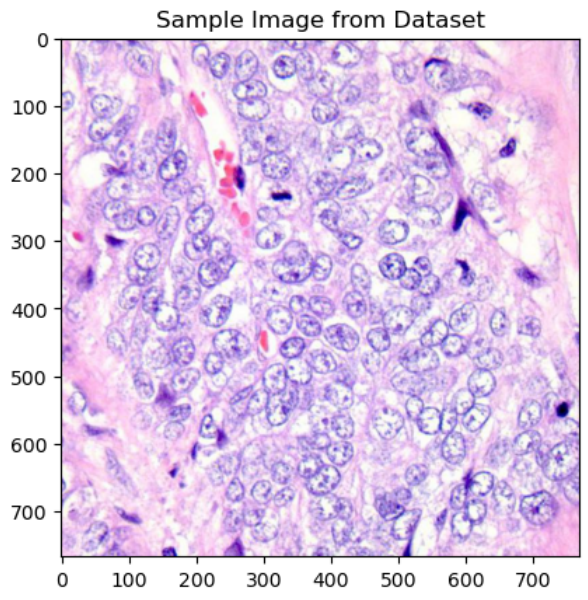
List the total number of images in the dataset

In [5]:
```
# Calculate the total number of images in the data folder
total_images = len(data)

# Print the total number of images
print(f"Total number of images in the data folder: {total_images}")
```

```
Total number of images in the data folder: 700
```

Plot an image from the datatset

In [9]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# Load and display a JPEG image
image_path = "/Users/mmliu/Desktop/CISB62_Final_MandyLiu/data/cancer/colonca1.jpeg"

# Read the image using matplotlib's image module
img = mpimg.imread(image_path)

# Create a figure and axis
fig, ax = plt.subplots()

# Display the image using imshow
ax.imshow(img)

# Set a title
ax.set_title("Sample Image from Dataset")

# Show the image
plt.show()
```



Data preparation

In [10]:
```
# Create a Pandas Series from the file_paths list and assign the name 'filepaths' to this Series
Files = pd.Series(file_paths , name= 'filepaths')

# Create another Pandas Series from the labels list and assign the name 'labels' to this Series
labels = pd.Series(labels ,name = 'labels' )

# Create a dataframe and concatenate the two Series horizontally along axis=1 (columns)
df = pd.concat([Files , labels] , axis = 1)
```

In [11]:
```
# Print the number of rows and columns of the dataframe
print('Number of rows: ', len(df))
print('Number of columns: ', len(df.columns))
```

```
Number of rows:  700
Number of columns:  2
```

In [12]:
```
# Display the information of the dataframe
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 700 entries, 0 to 699
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   filepaths  700 non-null    object
 1   labels     700 non-null    int64 
dtypes: int64(1), object(1)
memory usage: 11.1+ KB
```

In [13]:
```
# Display the head
df.head()
```

Out[13]:

	filepaths	labels
0	/Users/mmliu/Desktop/CISB62_Final_MandyLiu/dat...	1
1	/Users/mmliu/Desktop/CISB62_Final_MandyLiu/dat...	1
2	/Users/mmliu/Desktop/CISB62_Final_MandyLiu/dat...	1
3	/Users/mmliu/Desktop/CISB62_Final_MandyLiu/dat...	1
4	/Users/mmliu/Desktop/CISB62_Final_MandyLiu/dat...	1

Split the data

In [14]:
```
# X = image_files or List of file paths
# y = labels or List of corresponding labels (non-cancer: 0, cancer: 1)

# Split your data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df['filepaths'],df['labels'], test_size=0.2, random_state=35)
```

In [15]:
```
# Display the shape of the train_test_split data

print('X_train shape: ', X_train.shape)
print('X_test shape: ', X_test.shape)
print('y_train shape: ', X_train.shape)
print('y_test shape: ', X_test.shape)
```

```
X_train shape:  (560,)
X_test shape:  (140,)
y_train shape:  (560,)
y_test shape:  (140,)
```

Determine the image height, width and number of channels

In [16]:
```
# Select a sample image from the dataset
sample_image_path = file_paths[0]  # Change the index to select a different image

# Load the sample image
sample_image = plt.imread(sample_image_path)

# Get the dimensions of the sample image
image_height, image_width, num_channels = sample_image.shape

print("Image Information\n")
print(f"Number of images: {len(data):,}")
print(f"Height: {image_height} pixels, width: {image_width} pixels, number of channels: {num_channels}\n")
```

```
Image Information

Number of images: 700
Height: 768 pixels, width: 768 pixels, number of channels: 3
```

Artificial Neural Network (ANN)

Design the Neural Network Architecture and train the model

In [17]:
```
# Create a Sequential model
model = keras.Sequential([
    layers.Input(shape=(image_height, image_width, num_channels)),# Image size is 768 X 768 with 3 channels
    layers.Flatten(),                                          # Flatten the layer for dense layer compatability
    layers.Dense(32, activation='relu'),                       # 32 neurons hidden layer - learn patterns in data
    layers.Dense(16, activation='relu'),                       # 16 neurons hidden layer
    layers.Dense(1, activation='sigmoid')                      # Sigmoid activation for binary classification
])
```

In [18]:
```
# The model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 1769472)           0

 dense (Dense)               (None, 32)                56623136

 dense_1 (Dense)             (None, 16)                528

 dense_2 (Dense)             (None, 1)                 17

=================================================================
Total params: 56623681 (216.00 MB)
Trainable params: 56623681 (216.00 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [19]:
```
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Normalize pixel values and convert to NumPy arrays

In [20]:
```
# Prepare image data for the Deep Learning model by loading images, normalizing pixel values,
# and converting them into NumPy arrays
# plt.imread is a function from the Matplotlib library that reads an image
# and returns its pixel values as a NumPy array

X_train = np.array([plt.imread(image_path) for image_path in X_train]) / 255.0
X_test = np.array([plt.imread(image_path) for image_path in X_test]) / 255.0
y_train = np.array(y_train)
y_test = np.array(y_test)
```

Train the model

In [21]:
```
model.fit(X_train, y_train, epochs=10, batch_size=16, validation_data=(X_test, y_test))
```

```
Epoch 1/10
35/35 [==============================] - 28s 777ms/step - loss: 94.0304 - accuracy: 0.4875 - val_loss: 20.2177 - val_a
ccuracy: 0.5000
Epoch 2/10
35/35 [==============================] - 19s 533ms/step - loss: 18.7206 - accuracy: 0.4857 - val_loss: 6.8305 - val_ac
curacy: 0.5000
Epoch 3/10
35/35 [==============================] - 19s 544ms/step - loss: 15.0178 - accuracy: 0.4732 - val_loss: 19.2541 - val_a
ccuracy: 0.5000
Epoch 4/10
35/35 [==============================] - 19s 531ms/step - loss: 10.4031 - accuracy: 0.5214 - val_loss: 5.6995 - val_ac
curacy: 0.5000
Epoch 5/10
35/35 [==============================] - 19s 550ms/step - loss: 0.8586 - accuracy: 0.6893 - val_loss: 0.6989 - val_acc
uracy: 0.6214
Epoch 6/10
35/35 [==============================] - 19s 547ms/step - loss: 1.7241 - accuracy: 0.6357 - val_loss: 3.2701 - val_acc
uracy: 0.5000
Epoch 7/10
35/35 [==============================] - 19s 547ms/step - loss: 1.5018 - accuracy: 0.5929 - val_loss: 1.4831 - val_acc
uracy: 0.6071
Epoch 8/10
35/35 [==============================] - 21s 588ms/step - loss: 0.5934 - accuracy: 0.7571 - val_loss: 1.1726 - val_acc
uracy: 0.5500
Epoch 9/10
35/35 [==============================] - 20s 557ms/step - loss: 0.6419 - accuracy: 0.7375 - val_loss: 1.3631 - val_acc
uracy: 0.6071
Epoch 10/10
35/35 [==============================] - 19s 545ms/step - loss: 0.5564 - accuracy: 0.7482 - val_loss: 0.8486 - val_acc
uracy: 0.5429
```

Out[21]:
```
<keras.src.callbacks.History at 0x151a69590>
```

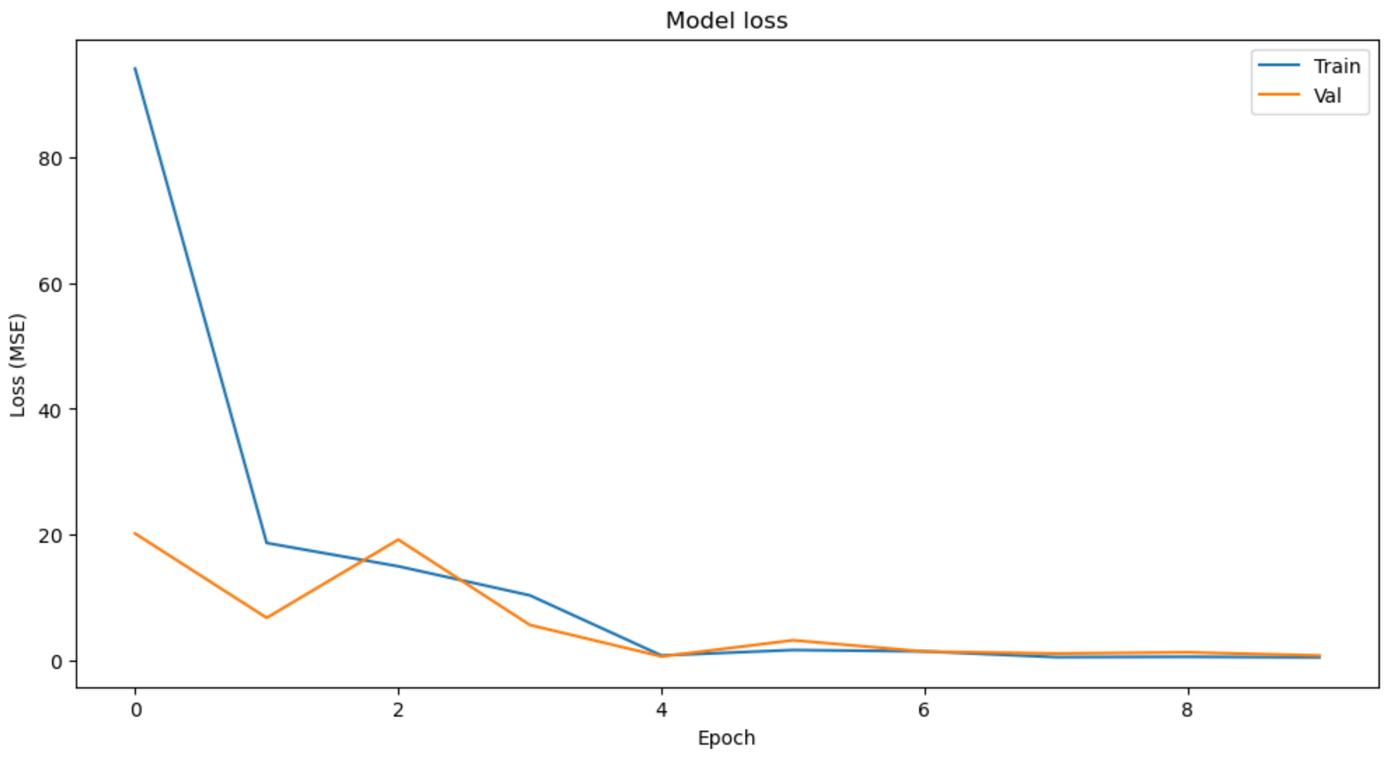
Save the Model

```
In [22]: model.save('Colon_Cancer_Image_Classification_ANN.keras')
```

Plot the model loss

```
In [23]: plt.figure(figsize=(12,6))
         plt.plot(model.history.history['loss'][:])
         plt.plot(model.history.history['val_loss'][:])
         plt.title('Model loss')
         plt.xlabel('Epoch')
         plt.ylabel('Loss (MSE)')
         plt.legend(['Train', 'Val'], loc='upper right')
```

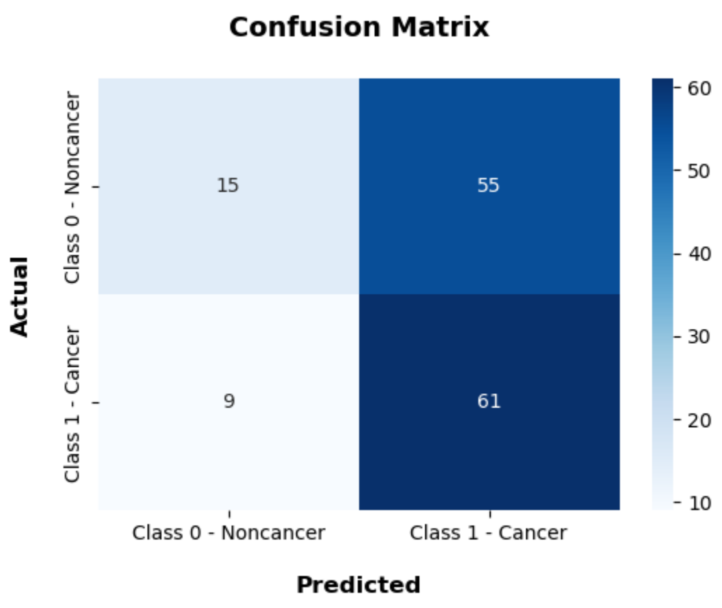
```
Out[23]: <matplotlib.legend.Legend at 0x1523f8f90>
```



Confusion Matrix

In [24]:
```
# Define Y_true and Y_pred_classes
Y_true = np.array(y_test)
Y_pred_classes = np.round(model.predict(X_test))

# Calculate the confusion matrix
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)

# Create a heatmap for the confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(confusion_mtx, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Class 0 - Noncancer', 'Class 1 - Cancer'],
            yticklabels=['Class 0 - Noncancer', 'Class 1 - Cancer'])

plt.xlabel('\nPredicted', fontsize=12, fontweight='bold')
plt.ylabel('Actual\n', fontsize=12, fontweight='bold')
plt.title('\nConfusion Matrix\n', fontsize=14, fontweight='bold')
plt.show()
```

```
5/5 [==============================] - 0s 36ms/step
```



Find the number of incorrect predictions

In [30]:
```
predictions = model.predict(X_test)
```

```
5/5 [==============================] - 1s 39ms/step
```

In [31]:
```
num_samples = X_test.shape[0]
images = X_test.reshape((num_samples, 768, 768, 3))
incorrect_predictions = []

for i, (p, e) in enumerate(zip(predictions, y_test)):
    predicted, expected = np.argmax(p), np.argmax(e)

    if predicted != expected:
        incorrect_predictions.append((i, images[i], predicted, expected))
```

In [32]:
```
# Print the length of incorrection predictions
len(incorrect_predictions)
```

Out[32]:
```
0
```

Hyperparameters Tuning

In [33]:
```
import keras_tuner as kt
```

```
Using TensorFlow backend
```

In [34]:
```
import os
import shutil

# shutil module is part of the Python standard library and provides a
# collection of utility functions for working with files and directories.

folder_path = "my_dir/intro_to_kt/"

# Check if the folder exists before attempting to delete it
if os.path.exists(folder_path):
    # Remove the folder and its contents recursively
    shutil.rmtree(folder_path)
    print(f"The folder '{folder_path}' has been deleted.")
else:
    print(f"The folder '{folder_path}' does not exist.")
```

```
The folder 'my_dir/intro_to_kt/' has been deleted.
```

In [35]:
```
def model_builder(hp):

  # Create a Keras Sequential model (linear stack of layers)
  model = keras.Sequential()

  # Add a Flatten layer to reshape the input data
  model.add(keras.layers.Flatten(input_shape=(768, 768,3)))

  # hp.Int is a function that represents the number of units (neurons) in a dense (fully connected) layer
  hp_units = hp.Int('units', min_value=4, max_value=20, step=4)

  # Add a fully connected dense layer
  model.add(keras.layers.Dense(units=hp_units, activation='relu'))

  # Output layer
  model.add(keras.layers.Dense(1, activation='sigmoid'))

  # Compile the model
  model.compile(loss='binary_crossentropy', optimizer='Adam', metrics=['accuracy'])

  return model
```

In [36]:
```
# Set up and configure a Keras Tuner Hyperband object for hyperparameter optimization
tuner = kt.Hyperband(model_builder,
                     objective='val_accuracy',
                     max_epochs=10,
                     factor=3,
                     directory='my_dir',
                     project_name='intro_to_kt')
```

In [37]:
```
# create an Early Stopping callback to use during the training to prevent overfitting and save training time
stop_early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3)
```

In [38]:
```
tuner.search(X_train, y_train, epochs=5, validation_split=0.2, callbacks=[stop_early])

# Get the optimal hyperparameters
best_hps=tuner.get_best_hyperparameters(num_trials=1)[0]

print(f"""
The hyperparameter search is complete. The optimal number of units in the first densely-connected
layer is {best_hps.get('units')}.
""")
```

```
Trial 5 Complete [00h 01m 13s]
val_accuracy: 0.5446428656578064

Best val_accuracy So Far: 0.5446428656578064
Total elapsed time: 00h 06m 15s

The hyperparameter search is complete. The optimal number of units in the first densely-connected
layer is 4.
```

Train the model

Find the optimal number of epochs to train the model with the hyperparameters obtained from the search.

In [39]:
```
# Build the model with the optimal hyperparameters and train it on the data for 10 epochs
model = tuner.hypermodel.build(best_hps)
history = model.fit(X_train, y_train, epochs=10, validation_split=0.2)

val_acc_per_epoch = history.history['val_accuracy']
best_epoch = val_acc_per_epoch.index(max(val_acc_per_epoch)) + 1
print('Best epoch: %d' % (best_epoch,))
```

```
Epoch 1/10
14/14 [==============================] - 12s 791ms/step - loss: 35.6678 - accuracy: 0.4933 - val_loss: 0.6933 - val_ac
curacy: 0.4554
Epoch 2/10
14/14 [==============================] - 3s 226ms/step - loss: 0.6931 - accuracy: 0.5112 - val_loss: 0.6934 - val_accu
racy: 0.4554
Epoch 3/10
14/14 [==============================] - 2s 130ms/step - loss: 0.6931 - accuracy: 0.5112 - val_loss: 0.6935 - val_accu
racy: 0.4554
Epoch 4/10
14/14 [==============================] - 2s 128ms/step - loss: 0.6931 - accuracy: 0.5112 - val_loss: 0.6935 - val_accu
racy: 0.4554
Epoch 5/10
14/14 [==============================] - 2s 130ms/step - loss: 0.6931 - accuracy: 0.5112 - val_loss: 0.6936 - val_accu
racy: 0.4554
Epoch 6/10
14/14 [==============================] - 2s 130ms/step - loss: 0.6931 - accuracy: 0.5112 - val_loss: 0.6936 - val_accu
racy: 0.4554
Epoch 7/10
14/14 [==============================] - 2s 133ms/step - loss: 0.6931 - accuracy: 0.5112 - val_loss: 0.6936 - val_accu
racy: 0.4554
Epoch 8/10
14/14 [==============================] - 2s 137ms/step - loss: 0.6930 - accuracy: 0.5112 - val_loss: 0.6937 - val_accu
racy: 0.4554
Epoch 9/10
14/14 [==============================] - 2s 133ms/step - loss: 0.6930 - accuracy: 0.5112 - val_loss: 0.6937 - val_accu
racy: 0.4554
Epoch 10/10
14/14 [==============================] - 2s 114ms/step - loss: 0.6930 - accuracy: 0.5112 - val_loss: 0.6937 - val_accu
racy: 0.4554
Best epoch: 1
```

In [40]:
```
# Re-instantiate the hypermodel and train it with the optimal number of epochs from above.
hypermodel = tuner.hypermodel.build(best_hps)

# Retrain the model
hypermodel.fit(X_train, y_train, epochs=best_epoch, validation_split=0.2)
```

```
14/14 [==============================] - 10s 647ms/step - loss: 17.7367 - accuracy: 0.5268 - val_loss: 0.6930 - val_ac
curacy: 0.5446
```

Out[40]:
```
<keras.src.callbacks.History at 0x153335810>
```

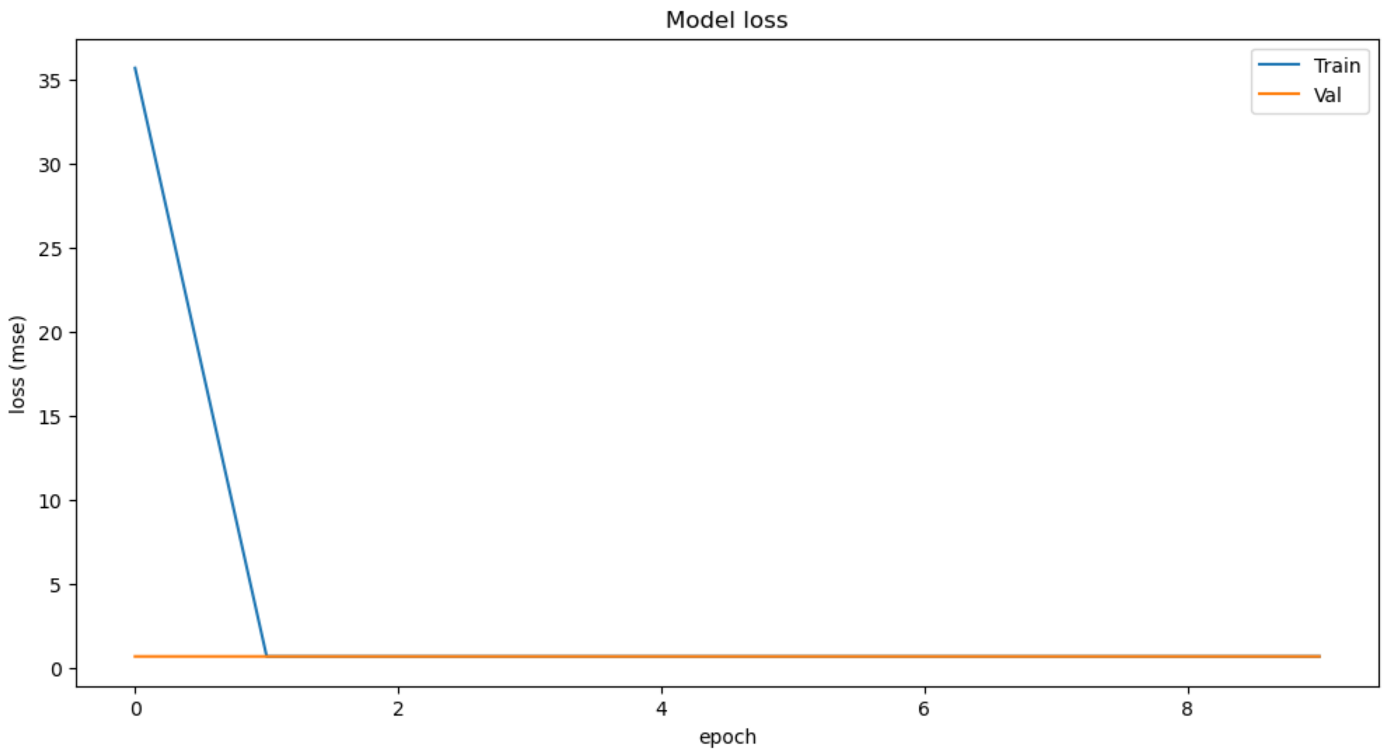
In [53]:
```
# Evaluate the model's performance on training dataset and print the results
eval_result = hypermodel.evaluate(X_train, y_train)
print('[test loss, test accuracy]:', eval_result)
```

```
18/18 [==============================] - 3s 77ms/step - loss: 0.6931 - accuracy: 0.5000
[test loss, test accuracy]: [0.6931494474411011, 0.5]
```

Plot the model loss

In [42]:
```
plt.figure(figsize=(12,6))
plt.plot(model.history.history['loss'][:])
plt.plot(model.history.history['val_loss'][:])
plt.title('Model loss')
plt.xlabel('epoch')
plt.ylabel('loss (mse)')
plt.legend(['Train', 'Val'], loc='upper right')
```

Out[42]:
```
<matplotlib.legend.Legend at 0x1535feb10>
```



Convolutional Neural Network (CNN)

In [43]:
```
model = Sequential()

# first conv-pool block:
model.add(Conv2D(96, kernel_size=(11, 11), strides=(4, 4), activation='relu',
                 input_shape=(image_height, image_width, num_channels)))
model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))
model.add(BatchNormalization())

# second conv-pool block
model.add(Conv2D(256, kernel_size=(5, 5), activation='relu'))
model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))
model.add(BatchNormalization())

# third conv-pool block
model.add(Conv2D(256, kernel_size=(3, 3), activation='relu'))
model.add(Conv2D(384, kernel_size=(3, 3), activation='relu'))
model.add(Conv2D(384, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))
model.add(BatchNormalization())

# dense layers
model.add(Flatten())
model.add(Dense(4096, activation='tanh'))
model.add(Dropout(0.5))
model.add(Dense(4096, activation='tanh'))
model.add(Dropout(0.5))

# output layer
model.add(Dense(1, activation='sigmoid'))    #sigmoid for binary classification
```

In [44]:
```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 190, 190, 96)      34944     
                                                                 
 max_pooling2d (MaxPooling2  (None, 94, 94, 96)        0         
 D)                                                              
                                                                 
 batch_normalization (Batch  (None, 94, 94, 96)        384       
 Normalization)                                                  
                                                                 
 conv2d_1 (Conv2D)           (None, 90, 90, 256)       614656    
                                                                 
 max_pooling2d_1 (MaxPoolin  (None, 44, 44, 256)       0         
 g2D)                                                            
                                                                 
 batch_normalization_1 (Bat  (None, 44, 44, 256)       1024      
 chNormalization)                                                
                                                                 
 conv2d_2 (Conv2D)           (None, 42, 42, 256)       590080    
                                                                 
 conv2d_3 (Conv2D)           (None, 40, 40, 384)       885120    
                                                                 
 conv2d_4 (Conv2D)           (None, 38, 38, 384)       1327488   
                                                                 
 max_pooling2d_2 (MaxPoolin  (None, 18, 18, 384)       0         
 g2D)                                                            
                                                                 
 batch_normalization_2 (Bat  (None, 18, 18, 384)       1536      
 chNormalization)                                                
                                                                 
 flatten_3 (Flatten)         (None, 124416)            0         
                                                                 
 dense_6 (Dense)             (None, 4096)              509612032 
                                                                 
 dropout (Dropout)           (None, 4096)              0         
                                                                 
 dense_7 (Dense)             (None, 4096)              16781312  
                                                                 
 dropout_1 (Dropout)         (None, 4096)              0         
                                                                 
 dense_8 (Dense)             (None, 1)                 4097      
                                                                 
=================================================================
Total params: 529852673 (1.97 GB)
Trainable params: 529851201 (1.97 GB)
Non-trainable params: 1472 (5.75 KB)
_________________________________________________________________
```

In [45]:
```
# binary_crossentropy for binary classification
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [46]:
```
model.fit(X_train, y_train, batch_size=64, epochs=5, verbose=1, validation_split=0.1)
```

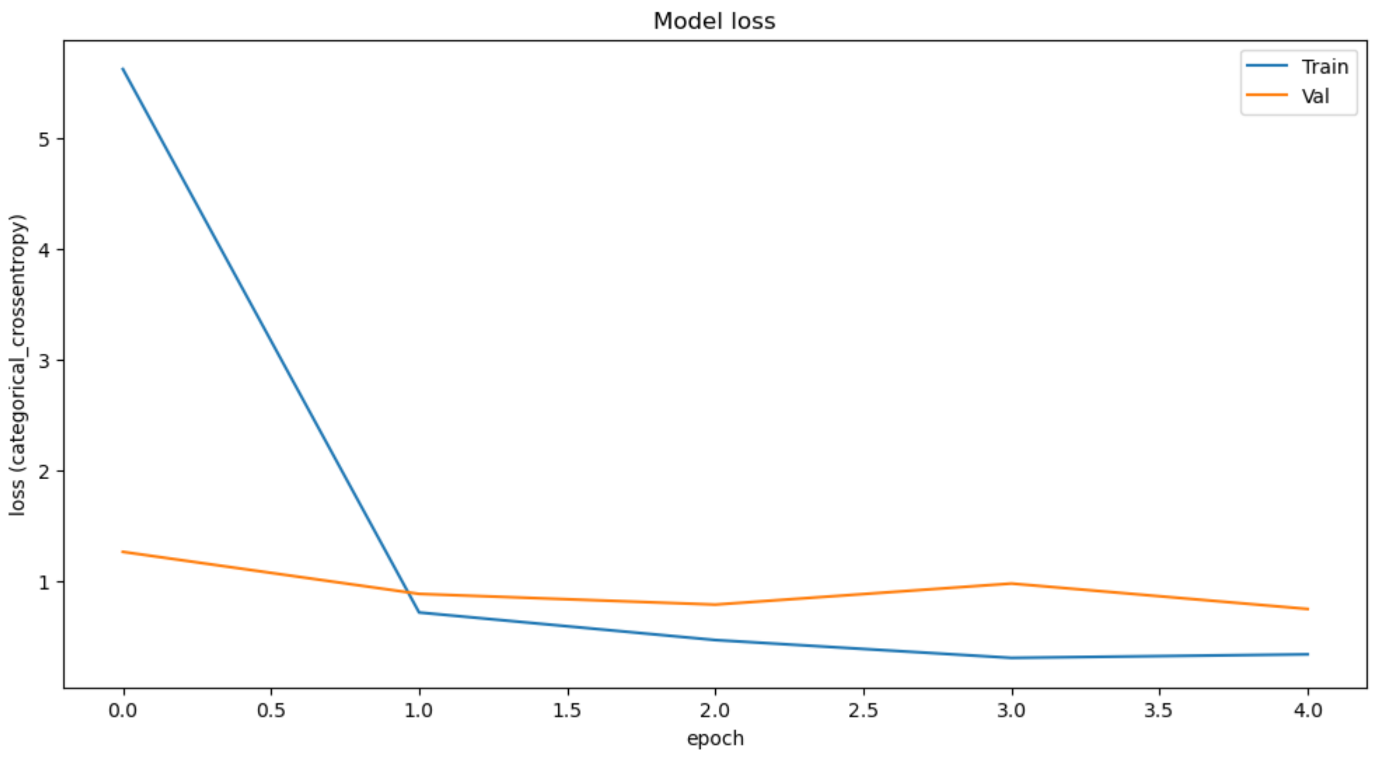
```
Epoch 1/5
8/8 [==============================] - 382s 47s/step - loss: 5.6196 - accuracy: 0.6131 - val_loss: 1.2667 - val_accura
cy: 0.5179
Epoch 2/5
8/8 [==============================] - 329s 41s/step - loss: 0.7200 - accuracy: 0.8056 - val_loss: 0.8875 - val_accura
cy: 0.4821
Epoch 3/5
8/8 [==============================] - 329s 40s/step - loss: 0.4715 - accuracy: 0.8512 - val_loss: 0.7910 - val_accura
cy: 0.4821
Epoch 4/5
8/8 [==============================] - 334s 42s/step - loss: 0.3106 - accuracy: 0.8829 - val_loss: 0.9810 - val_accura
cy: 0.4821
Epoch 5/5
8/8 [==============================] - 314s 39s/step - loss: 0.3425 - accuracy: 0.9067 - val_loss: 0.7516 - val_accura
cy: 0.4821
```

Out[46]:
```
<keras.src.callbacks.History at 0x15332edd0>
```

Plot the model loss

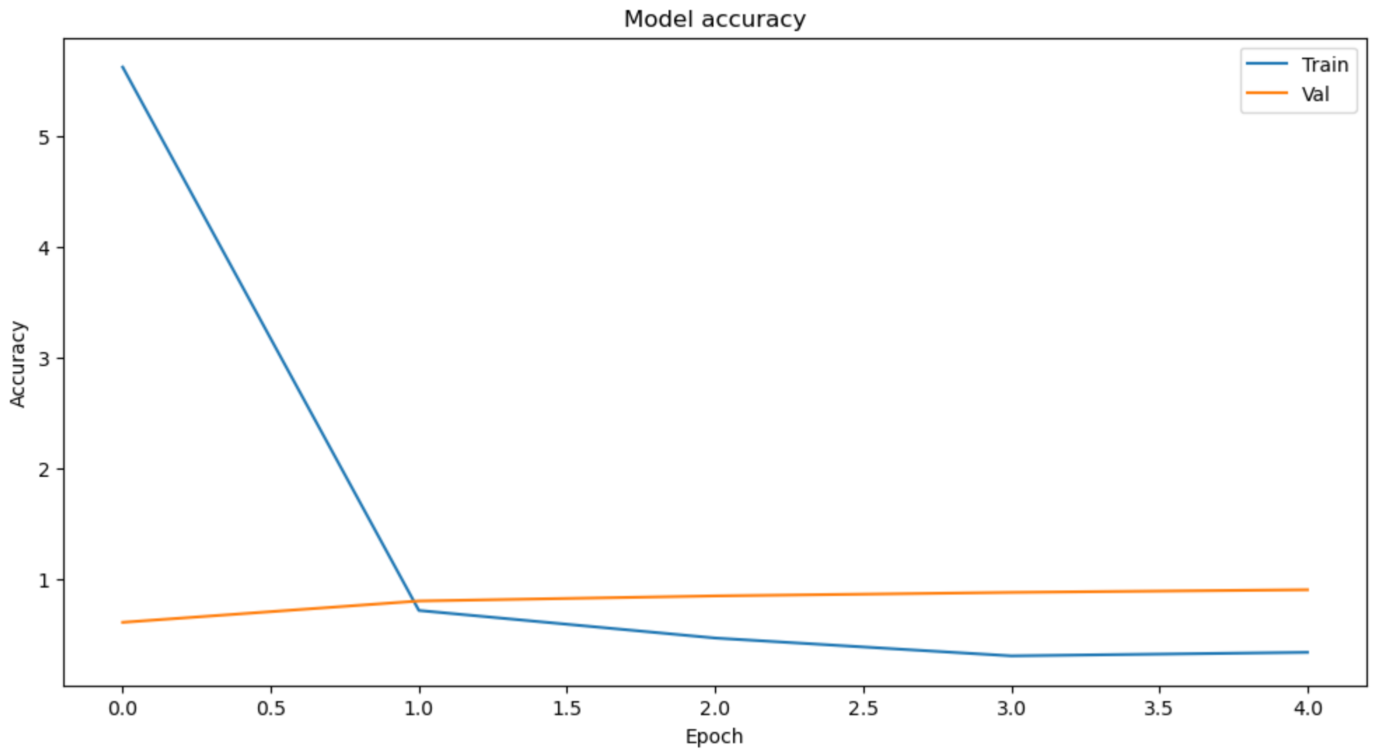
```
In [47]: # Model loss (Train vs Validation)
         plt.figure(figsize=(12,6))
         plt.plot(model.history.history['loss'][:])
         plt.plot(model.history.history['val_loss'][:])
         plt.title('Model loss')
         plt.xlabel('epoch')
         plt.ylabel('loss (categorical_crossentropy)')
         plt.legend(['Train', 'Val'], loc='upper right')
```

```
Out[47]: <matplotlib.legend.Legend at 0x153f26c90>
```



Plot the model accuracy

In [48]:
```
# Model accuracy (Train vs Validation)
plt.figure(figsize=(12, 6))
plt.plot(model.history.history['loss'][:])
plt.plot(model.history.history['accuracy'][:])  # Use 'accuracy' instead of 'acc'
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```



Save the model

In [49]:
```
model.save('Colon_Cancer_Image_Classification_CNN.keras')
```