Report Submission

August 6, 2024

The Git Repository for this project can be found at `https://github.com/seel6470/CSPB-3202-HW5`

1 Description

brief description of the problem, data (e.g. size and dimension, structure etc)

The Histopathologic Cancer Detection Kaggle competition (located at https://www.kaggle.com/c/histopathologic-cancer-detection/overview) seeks to create a machine learning algorithm that can detect a cancer cell given a pathology image of tumor tissue. The problem itsel is a **binary classification problem** with either a negative of positive prognosis.

Furthermore, in the data description, it is specified that the center 32 x 32 pixel region of the image must contain at least one pixel of tumor tissue in order to be a positive prognosis.

The data is contained in a file structure of the data contains two directories, `train` and `test`

The `train` directory contains 220,025 tif images while the `test` directory contains 57,458 tif images.

Each file represents a color 96 x 96 image with each pixel represented as a 24 bit RGB value.

The classification of all images is contained in a csv file labeled train_labels.csv with two columns, "id" and "label"

The "id" column represents the filename (without file extension) which would be a categorical nominal data type since it represents a value with no order or ranking, while the "label" column represents the binary classification 0 or 1, which would also be considered categorical nominal data as well.

2 Exploratory Data Analysis

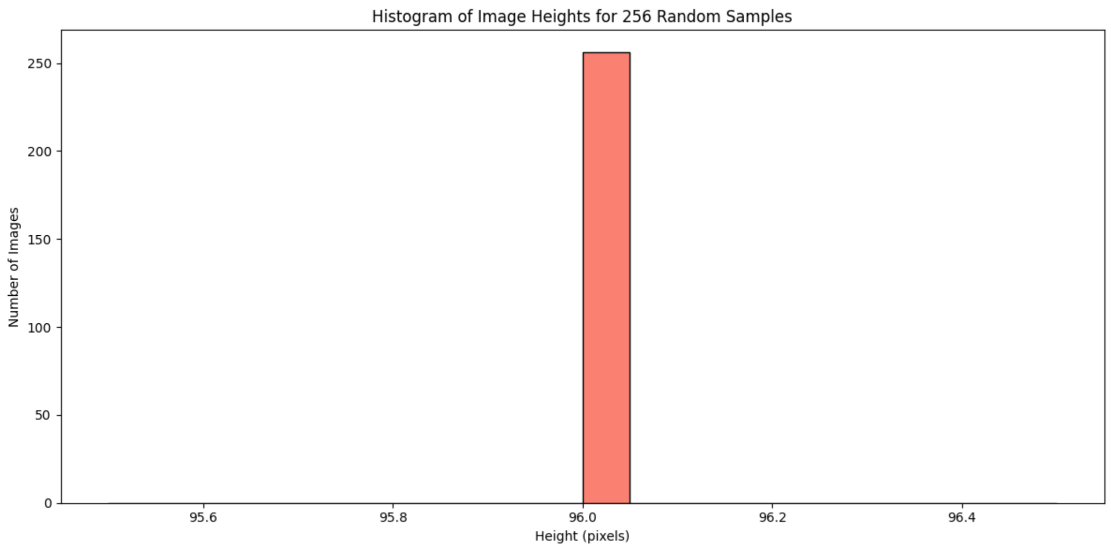
Exploratory data analysis showing a few visualization, histogram, etc, and a plan of analysis. Any data cleaning procedure.

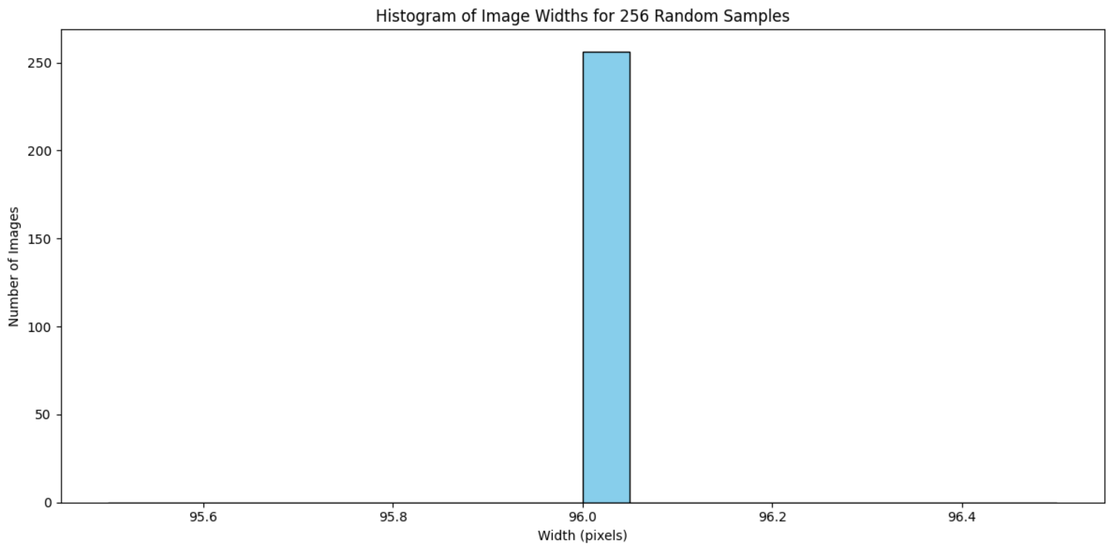
It would be helpful to determine if all images have a consistent resolution, or if there may be differing image sizes. It is challenging to do so efficiently, due to the size of the data set, but creating a subset of the images and determining the resolution sizes of the images may provide more clarity.

> **Note:** Due to the size of the data set, I chose to work in a local environment, downloading all images to a local directory and running my scripts from the command line.

The following code may not be executable in this notebook, but the python scripts are included in the GitHub repo.

```python
import pandas as pd
import matplotlib as plt
import os
from PIL import Image
import matplotlib.pyplot as plt

data = pd.read_csv('train_labels.csv', dtype=str)

data['id'] = data['id'] + '.tif'  # Add file extension

# Select a random subset of 256 images
subset = data.sample(n=256, random_state=1975)

train_directory = './train'

# Create lists to store image widths and heights
widths = []
heights = []

# Iterate over the subset to get image dimensions
for image_file in subset['id']:
    image_path = os.path.join(train_directory, image_file)
    if os.path.exists(image_path):
        with Image.open(image_path) as img:
            width, height = img.size
            widths.append(width)
            heights.append(height)
    else:
        print(f"Image file {image_file} does not exist.")

# Plot Histogram for Widths
plt.figure(figsize=(12, 6))
plt.hist(widths, bins=20, color='skyblue', edgecolor='black')
plt.xlabel('Width (pixels)')
plt.ylabel('Number of Images')
plt.title('Histogram of Image Widths for 256 Random Samples')
plt.tight_layout()
plt.savefig('image_widths_histogram.png')

# Plot Histogram for Heights
plt.figure(figsize=(12, 6))
plt.hist(heights, bins=20, color='salmon', edgecolor='black')
plt.xlabel('Height (pixels)')
plt.ylabel('Number of Images')
plt.title('Histogram of Image Heights for 256 Random Samples')
plt.tight_layout()
plt.savefig('image_heights_histogram.png')
```

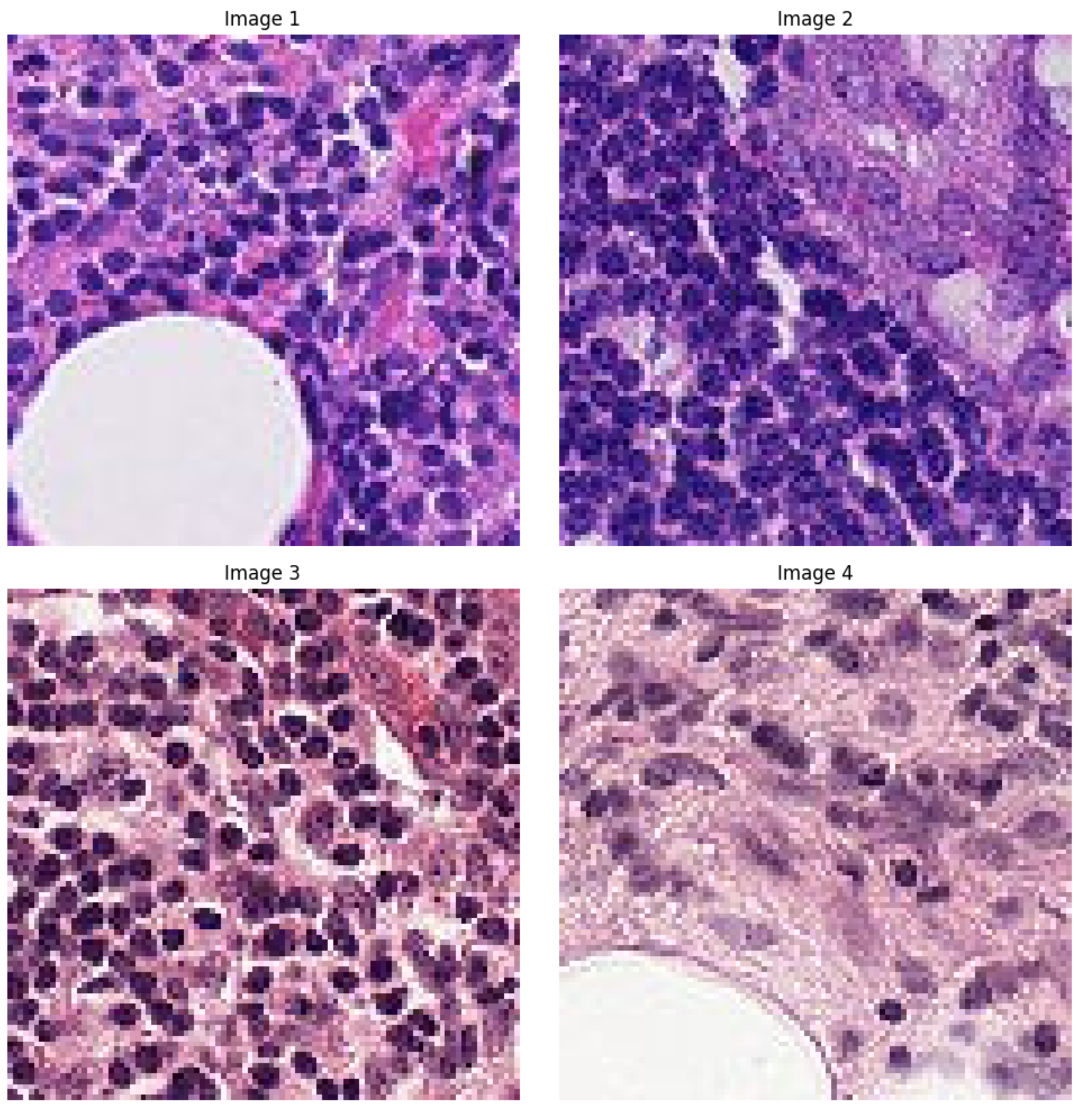




As we can see, the resolution sizes for the random sample is exclusively 96 x 96 x 96. This is helpful, since we will need to create a way to crop the center 32 x 32 pixels given the data description on Kaggle, and the uniformity of the image size will ensure that the center will exist in the same location for all images. We can further confirm this by outputting several random images:

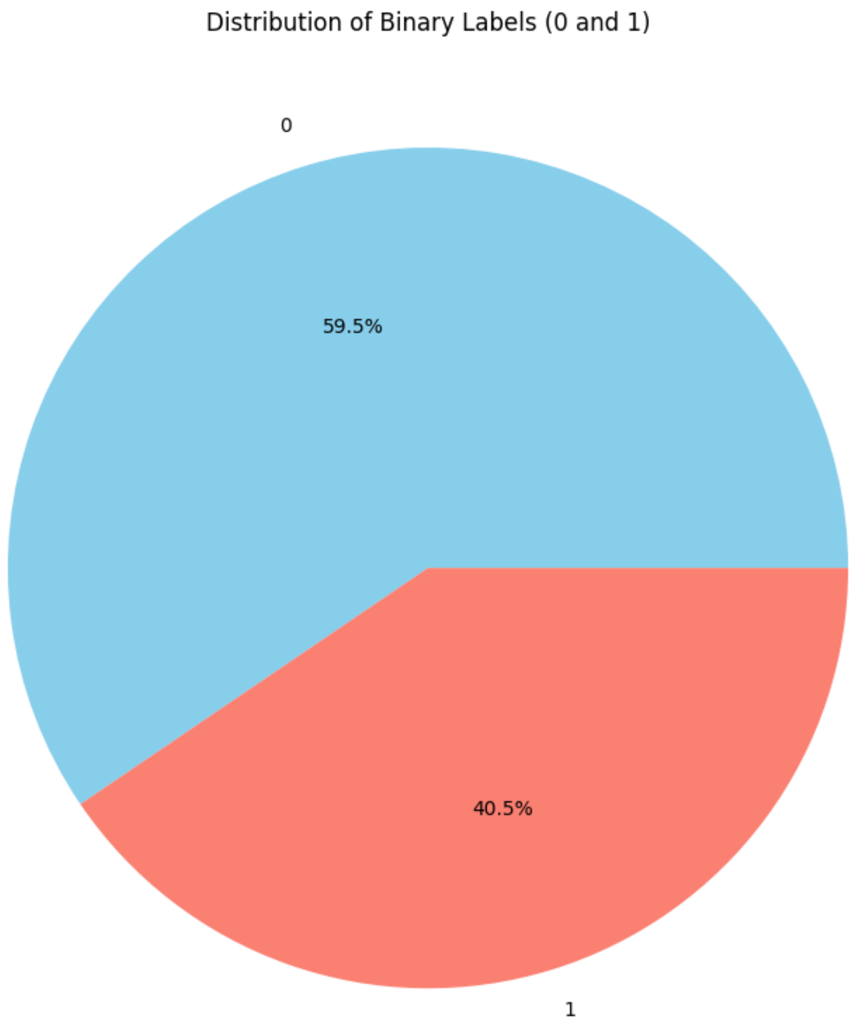
```
# get 4 random images
subset = data.sample(n=4, random_state=1975)

train_directory = './train'

# Create a figure to plot the images
```

```
plt.figure(figsize=(10, 10))

# Initialize index
index = 1

# Plot and save each image
for image_file in subset['id']:
    image_path = os.path.join(train_directory, image_file)
    if os.path.exists(image_path):
        with Image.open(image_path) as img:
            plt.subplot(2, 2, index)  # Use index to determine subplot position
            plt.imshow(img)
            plt.title(f'Image {index}')
            plt.axis('off')  # Turn off axis
            index += 1
    else:
        print(f"Image file {image_file} does not exist.")

# Save the plot
plt.tight_layout()
plt.savefig('sample_images.png')
```



Additionally, it would be helpful to know what distribution of labels we have in our training data. We would hope to see an equal amount of binary classifications to avoid bias in our model, however it would be beneficial to understand our data if the distribution is otherwise.

```
import pandas as pd
import matplotlib as plt
import os
from PIL import Image
import matplotlib.pyplot as plt

data = pd.read_csv('train_labels.csv', dtype=str)

label_counts = data['label'].value_counts()

plt.figure(figsize=(8, 8))
plt.pie(label_counts, labels=label_counts.index, autopct='%1.1f%%', colors=['skyblue', 'salmon
plt.title('Distribution of Binary Labels (0 and 1)')
plt.tight_layout()

# Save and show the pie chart
plt.savefig('binary_labels_pie_chart.png')
```



We can see that we have more negative than positive prognosis images in our training set. Given the large size of the data set, this should not cause too much of an issue. If there was a higher class imbalance (say 90/10) there would be cause for concern, however we still want to make sure we watch out to see if the model learns that it is more accurate to predict 0 than 1 given the current imbalance. Because of this, we may seek to use other metrics than accuracy, such as the area under the ROC curve when evaluating the model's performance.

3 Model Architecture

I have had some experience with creating classification models using Tensorflow and Keras, and additionally decided to use a pre-trained convolutional neural network model, or CNN with additional custom layers. To begin, we will need to import all of the libraries that will be useful:

```
import os
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout, BatchNormalization
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from sklearn.model_selection import train_test_split
```

Next, we will need to load our data in a format that is conducive to the Tensorflow Keras models using Pandas:

```
 Load the data
data = pd.read_csv('train_labels.csv', dtype=str)

data['id'] = data['id'] + '.tif'  # add tif to each filename
```

Next, we will split the train and test data and create Pandas dataframes from the :

```
filenames = data['id'].values
labels = data['label'].values

train_filenames, val_filenames, train_labels, val_labels = train_test_split(filenames, labels,

train_df = pd.DataFrame({'id': train_filenames, 'label': train_labels})
val_df = pd.DataFrame({'id': val_filenames, 'label': val_labels})
```

Now we will need to create a dataset that can be used in a Tensorflow Keras model. I struggled quite a bit with this step, but referencing other projects posted on the Kaggle site assisted greatly. Kern T.'s Kaggle page was very handy in particular (Kern, 2022)

Using the tensorflow.keras.preprocessing.image.ImageDataGenerator came in handy as it greatly simplified the process. Note that we are normalizing the RGB values as well as creating the target size of 32 x 32 pixels.

```
# create image data generators, normalizing RGB values
train_generator = ImageDataGenerator(rescale=1/255)
validation_generator = ImageDataGenerator(rescale=1/255)

train_dataset = train_generator.flow_from_dataframe(
    dataframe=train_df,
    directory=train_directory,
    x_col='id',
    y_col='label',
    batch_size=batch_size,
    seed=1,
    shuffle=True,
    class_mode='categorical',
    target_size=(32, 32)
)

val_dataset = validation_generator.flow_from_dataframe(
    dataframe=val_df,
    directory=train_directory,
    x_col='id',
    y_col='label',
    batch_size=batch_size,
    seed=1,
    shuffle=False,
    class_mode='categorical',
    target_size=(32, 32)
)
```

3.1 Initial Model

Initially, I began with a simple CNN model that used the EfficientNetB0 base model with some simple additional convolutional layers:

```
base_model = EfficientNetB0(weights='imagenet', include_top=False, input_shape=input_shape)

cnn = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(1024, activation='relu'),
    Dense(2, activation='softmax'),
    Dropout(0.5)
])
```

The Global Average Pooling layer reduces the spatial dimensions and flattens the data to a 1D array, the Dense layers are fully connected layers utilizing the ReLU activation function, and the dropout layers removes half of the neurons after the fully activated layers to prevent overfitting.

I then compiled the model fit the model using the Adam optimizer with an initial learning rate of 0.001 and the categorical crossentropy loss function using both accuracy and the area under the ROC curve (given the distribution of classifications found previously). I then fit this model using 40 epochs.

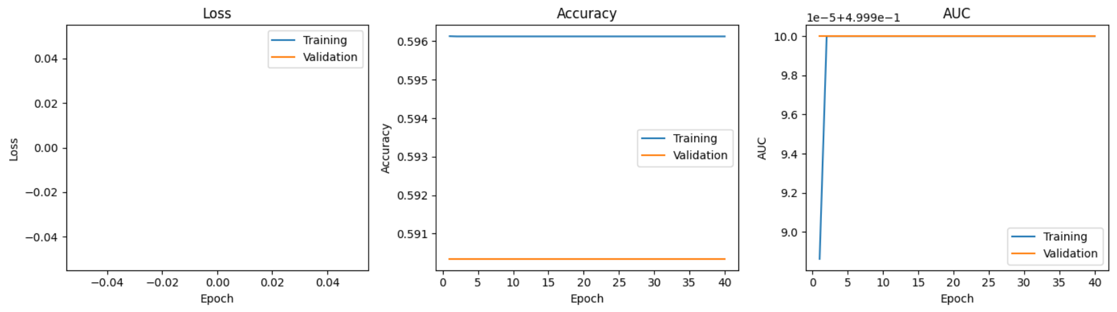
```
opt = tf.keras.optimizers.Adam(0.001)
cnn.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy', tf.keras.metri

model = cnn.fit(
    x = train_dataset,
    steps_per_epoch = len(train_dataset),
    epochs = 40,
    validation_data = val_dataset,
    validation_steps = len(val_dataset),
    verbose = 1
)
```

We then capture the loss, accuracy, and Area under the AUC curve for the training and validation date:

```
history = model.history

epoch_range = range(1, len(history['loss'])+1)

plt.figure(figsize=[14,4])
plt.subplot(1,3,1)
plt.plot(epoch_range, history['loss'], label='Training')
plt.plot(epoch_range, history['val_loss'], label='Validation')
plt.xlabel('Epoch'); plt.ylabel('Loss'); plt.title('Loss')
plt.legend()
plt.subplot(1,3,2)
plt.plot(epoch_range, history['accuracy'], label='Training')
plt.plot(epoch_range, history['val_accuracy'], label='Validation')
plt.xlabel('Epoch'); plt.ylabel('Accuracy'); plt.title('Accuracy')
plt.legend()
plt.subplot(1,3,3)
plt.plot(epoch_range, history['auc'], label='Training')
plt.plot(epoch_range, history['val_auc'], label='Validation')
plt.xlabel('Epoch'); plt.ylabel('AUC'); plt.title('AUC')
plt.legend()
plt.tight_layout()
plt.savefig('graphs.png')
```

Finally, I used the model to predict the values in the test data and saved the values to a csv to submit:

```
test_filenames = [f for f in os.listdir(test_directory)]

test_labels = np.zeros(len(test_filenames), dtype=np.int64)
test_df = pd.DataFrame({'id': test_filenames, 'label': test_labels})
print(test_df.head())

test_dataset = validation_generator.flow_from_dataframe(
    dataframe=test_df,
    directory=test_directory,
    x_col='id',
    y_col='label',
    batch_size=batch_size,
    seed=1,
    shuffle=False,
    class_mode=None,
    target_size=(32, 32)
)

predictions = cnn.predict(test_dataset)
# predictions are presented as a list of tuples with probabilities for each category
# e.g. [0.5671,0.4329]
# the actual category will be equal to the index of the maximum element in the tuplepredicted_
predicted_labels = np.argmax(predictions, axis=1)
# Create a DataFrame with the IDs and predicted labels
submission = pd.DataFrame({'id': test_filenames, 'label': predicted_labels})

# Save the DataFrame to a CSV file
submission.to_csv('final_submission.csv', index=False)
```



Unfortunately, this initial model did not perform very well, with a Kaggle score of only 0.5 with a static prediction of 0 for all test images and NaN values for the loss for the train and validation data during training. Upon closer inspection, I realized the dropout layer after the final layer would cause significant issues, as droppout should only be used prior to the final fully connected layer. Otherwise, 50% of the data wijll be set to a class of 0. This does not indicate a successful model, and further work must be done to create a fully working model architecture suitable

Because of this, I went back to the drawing board to make some improvements.

3.2 Second Model

Firstly, let's remove the final dropout to correct the error from the first model. Additionally, I realized that incorporating 2D convolutional layers would be most appropriate as they are meant for feature extraction from 2D images by using filters over the 2 dimensional pixel area. Doing additional research, I realized MaxPooling would be most appropriate with Conv2D layers, as GlobalMaxPooling operates on the entire feature map and is better used at the end of the neural network. Additionally, I decided to repeat these convolutional layers twice using more features the second time, initially with 16, then 128, before performing the final fully connected layers at the end. This should help the model capture the low-level features initially and then capturing high-level abstract features when using a higher feature level. To gain additional insight, I also increased the number of epochs to 90.

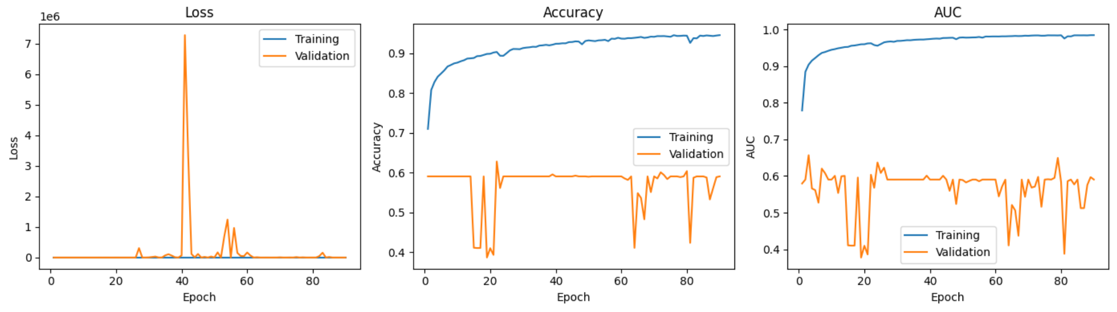
```
# Load pre-trained EfficientNetB0 model + higher level layers
base_model = EfficientNetB0(weights='imagenet', include_top=False, input_shape=input_shape)
```

```
cnn = Sequential([
    base_model,

    # 16 filters capture low level features (e.g. edges)
    Conv2D(64, (3,3), activation='relu', padding='same'),
    Conv2D(64, (3,3), activation='relu', padding='same'),
    MaxPooling2D(2,2, padding='same'),
    Dropout(0.5),
    BatchNormalization(),

    # 128 filters capture high-level abstract features
    Conv2D(128, (3,3), activation='relu', padding='same'),
    Conv2D(128, (3,3), activation='relu', padding='same'),
    MaxPooling2D(2,2, padding='same'),
    Dropout(0.5),
    BatchNormalization(),

    GlobalAveragePooling2D(),
    Flatten(),

    # final fully connected layers
    Dense(512, activation='relu'),
    Dropout(0.5),
    BatchNormalization(),

    Dense(16, activation='relu'),
    Dropout(0.5),
    Dense(8, activation='relu'),
    Dropout(0.5),
    BatchNormalization(),

    Dense(2, activation='softmax')
])

opt = tf.keras.optimizers.Adam(0.001)
cnn.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy', tf.keras.metri
```

I also decided to train the model for 90 epochs as opposed to the initial 40 in the hopes of gaining further fine tuned adjustments to the model.

```
model = cnn.fit(
    x = train_dataset,
    steps_per_epoch = len(train_dataset),
    epochs = 90,
    validation_data = val_dataset,
    validation_steps = len(val_dataset),
    verbose = 0
)
```

And captured the graphs for this submission as well:



As we can see from the graphs above, the accuracy and AUC values for the training data increase and flatten close to 1 as would be expected from a working model. The validation data, however, shows significant instability with a massive peak in the loss and valleys along the accuracy and AUC values, the latter of which never increase to any significant value that would indicate that it is going to work with the test dataset. This indicates that the model is overfitting to the training data and is not making valid insights into the images themselves. The output for the submission file also shows that all predictions for the test data are all zeros. Looking at the class distribution of the training data, it could possibly be biased towards the class with the highest percentage in the training data, choosing 0 every time.

Because of this, I chose to take one more shot to make some improvements to create a model that gathers some applicable insights into classification.

3.3 Third Model

Upon researching what could be causing the model to perform poorly over the validation data, I realized that a high learning rate can cause the model to make large updates to the weights, causing instability. To counteract this, I will create a learning rate scheduler to reduce the learning rate for different thresholds of epochs. This should allow the model to create fine-tuned adjustments with higher iterations.

I also looked over the original problem description and noticed the following passage:

> A positive label indicates that the center 32x32px region of a patch contains at least one pixel of tumor tissue. Tumor tissue in the outer region of the patch does not influence the label.

I realize that using `target_size = (32,32)` in the image data generator simply reduces the entire image size to a 32 x 32 pixel resolution. It would be beneficial to create a preprocessing function that could be used with the `ImageDataGenerator` in the `preprocessor` parameter that crops the center 32 x 32 pixels. With these two improvements, I am hoping the model will be more functional and provide a significant Kaggle score.

The following is the preprocessing function I will use to crop the center 32 x 32 pixels:

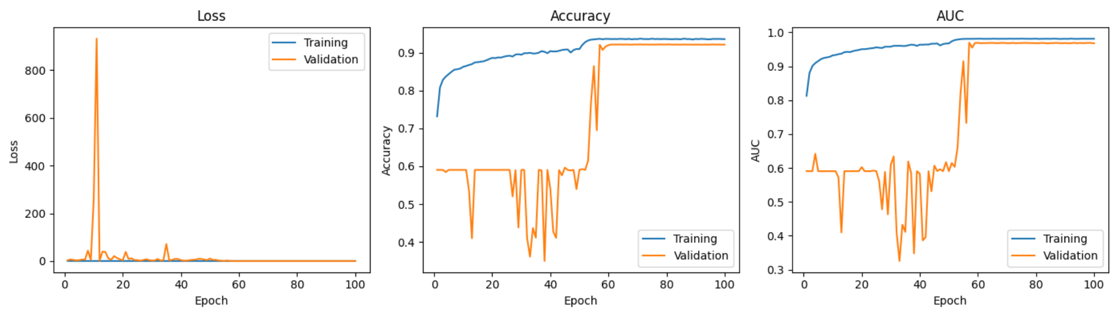
```
# Define a preprocessing function to crop the center 32x32 pixels
def center_crop(image):
    center = (image.shape[0] // 2, image.shape[1] // 2)
    half_crop_size = img_width // 2
    cropped_image = image[center[0] - half_crop_size:center[0] + half_crop_size, center[1] - ha
    return cropped_image

# create image data generators, normalizing RGB values
train_generator = ImageDataGenerator(rescale=1/255, preprocessing_function = center_crop)
validation_generator = ImageDataGenerator(rescale=1/255, preprocessing_function = center_crop)
```

Additionally, we will train the model using a learning rate scheduler to decrease the learning rate as the epochs go on. We can do this by simply defining a scheduler function and create a callback through tf.keras.callbacks.LearningRateScheduler and pass it to the fit paramaters:

```
# Learning rate scheduler function
def scheduler(epoch, lr):
    if epoch < 50:
        return lr
    elif epoch < 80:
        return lr * 0.5
    else:
        return lr * 0.1

lr_callback = tf.keras.callbacks.LearningRateScheduler(scheduler)

# Train the model
model = cnn.fit(
    x = train_dataset,
    steps_per_epoch = len(train_dataset),
    epochs = 100,
    validation_data = val_dataset,
    validation_steps = len(val_dataset),
    verbose = 1,
    callbacks = [lr_callback]
)
```

We will start with a learning rate of 0.001, dividing it by 2 at 50 epochs to a learning rate of 0.0005, and finally multiplying it by 0.1 to get a learning rate of 0.00005 for the final 20 epochs. This should help the model to converge better and hopefully create more stability.

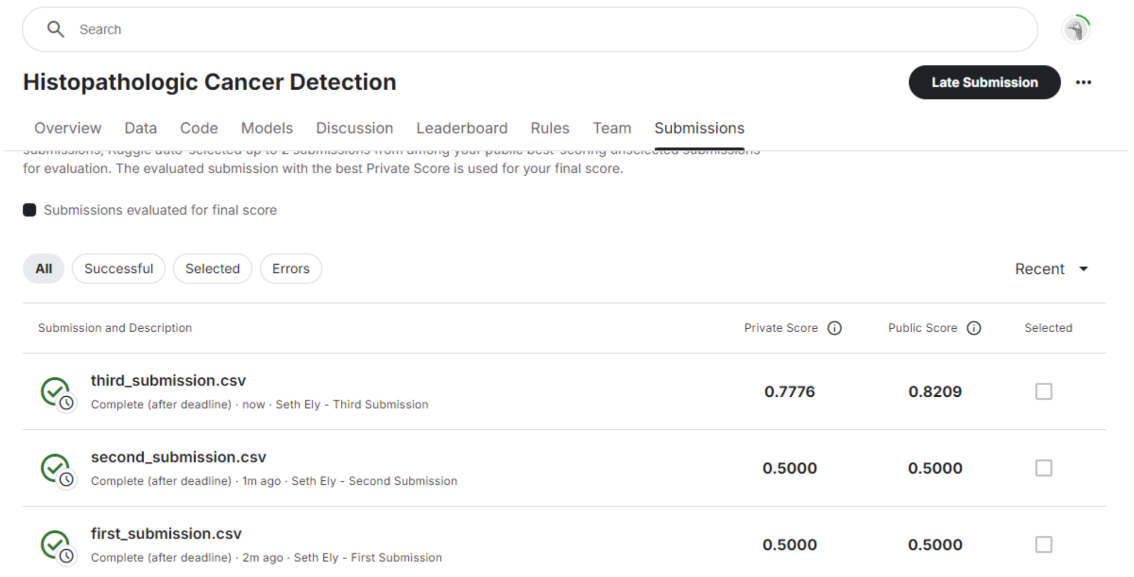
As usual, we plot the loss, accuracy and AUC values of the training and validation:



We see a similar spike in the loss values initially, which skews our data, however the values do reduce over time.

The Accuracy and AUC values for the training data rises logarithmically and converges close to 1 as previously, however there is a noticeable change in the validation trends that occurs somewhat near the first reduction in the learning rate at 50 epochs. The validation data also flattens pit close to 0.9, which is what would be much more expected with a working model. Looking at the submission data, there are a combination of both 1's and 0's, which means the improvements were likely successful.

Submitting these classifications to the Kaggle competition resulted in a private score of 0.7776 and a public score of 0.8209. This means our architecture for the model was successful in allowing the model to make significant correlations into the classification of the image data. The full breakdown of scores for all three models can be seen below:



Histopathologic Cancer Detection

for evaluation. The evaluated submission with the best Private Score is used for your final score.

■ Submissions evaluated for final score

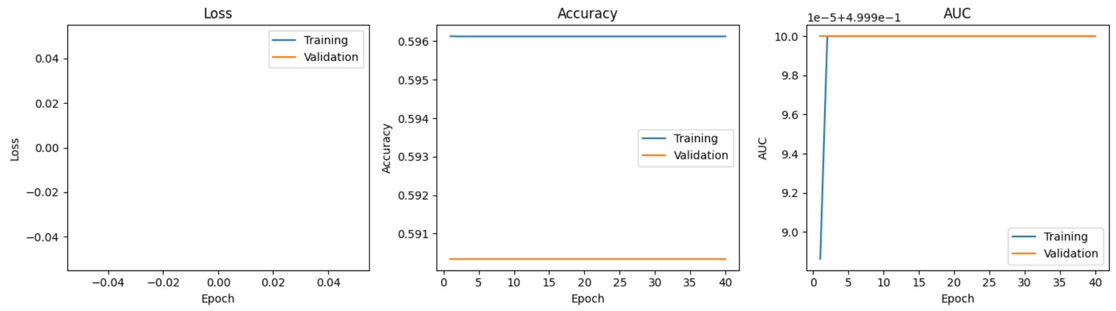
Submission and Description	Private Score	Public Score	Selected
third_submission.csv Complete (after deadline) · now · Seth Ely - Third Submission	0.7776	0.8209	☐
second_submission.csv Complete (after deadline) · 1m ago · Seth Ely - Second Submission	0.5000	0.5000	☐
first_submission.csv Complete (after deadline) · 2m ago · Seth Ely - First Submission	0.5000	0.5000	☐

4 Results

results (tables, figures etc) and analysis (reasoning of why or why not something worked well, also troubleshooting and hyperparameter optimization procedure summary)

Once again, here are the graphs of loss, accuracy, and AUC for the three models:

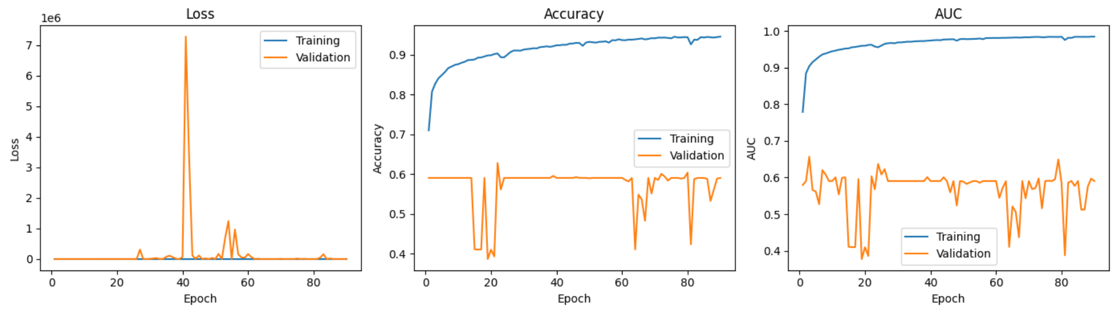
4.1 First Model



No meaningful or significant data, indicating serious problems with the model architecture. **Public Kaggle Score:** 0.5000

4.2 Second Model

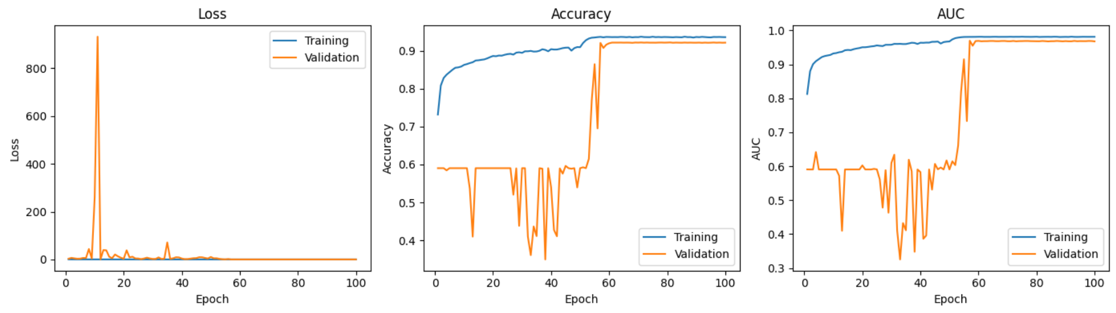
- *Removing final dropout layer after dense layer (made in error)*
- *adding Conv2d layers*
- *duplicating architecture with increasing features second time around*



Training accuracy and AUC indicates model is learning, but validation values indicate model instability. **Public Kaggle Score:** 0.5000

4.3 Third Model

- *Cropping center 32 x 32 pixels instead of size reduction to 32 x 32 pixels*
- *Create learning rate scheduler*



Training accuracy and AUC flatten out near 1.0, indicating a working model and validation values increase significantly near 50 epochs with learning rate reduction, indicating decent performance over validation data. **Public Kaggle Score:** 0.8209

5 Conclusion

Working with this large amount of data proved challenging, as many times I would begin to fit my model over the training data only for it to crash after several hours. I was able to use the Tensorflow with GPU to speed this up, but working with the number of epochs I chose for the architecture cost considerable processing time that was exceeding 7 hours and would still be processing in the morning after I initiated the script the night before. Simply getting a working model took a majority of my time with this project, and even the first model was a significant and noteworthy milestone.

A 0.8209 public score means that the architecture and model fitting was able to interpret a variety of features in the test data, overcoming issues with overfitting and instability found in the previous two models.

All in all, I feel like I have a better understanding of Machine Learning modeling after this project than I did before, and for that reason, as well as the final Kaggle score, I believe this to be a successful implementation.

6 References

Cukierski, W. (2018). Histopathologic Cancer Detection. Kaggle. https://kaggle.com/competitions/histopathologic-cancer-detection

Kern, T. (2022). PyTorch CNN: Histopathologic Cancer Detection. Kaggle. https://kaggle.com/code/taylorkern/histopathologic-cancer-detection