

# HW5 Kaggle Competition

@Author: Jordon Young (joyo8856)

[Github Repo](#)

To start, let's get one thing straight--my goal was not to get the most accurate model here. My goal was to explore how these tools work and intentionally make a less complicated, less computationally intensive model and solution to this problem.

I like to do development locally and don't like to wait for models to train overnight. Focusing on efficiency for this assignment has allowed me to more rapidly iterate on my model and experiment with different strategies.

## What is the problem? Histopathologic Cancer Detection.

This Kaggle competition comes with about 6GB of training images from digital pathology scans. Each image is only 96x96 pixels, so that is a lot of entries. The data is divided into training and testing data already. My task is to make a model that classifies images as either cancer or not cancer.

## Load and Preprocess Data

The data provided is included as 96x96 pixel images, but reading the competition description, only the center 32x32 pixels are considered. In other words, is there cancer present in the center of the image? Cropping the images to focus on the center only was a convenient step to reduce the computation time--both on loading and training the model.

Another benefit of this cropping, I hypothesize, is that this allows my model to learn only on the part of the image that will be evaluated on as opposed to identifying patterns in the outside portions of the image which may not be relevant to "identifying cancer in the center of the image".

I normalized the RGB values, but that's not really to save space. I also included a `fraction` variable to randomly select a portion of the training data only. The selection happens with the IDs, so only the necessary images are loaded into memory. Using `0.1` as the `fraction`, I was able to load all the images in one minute.

```
In [ ]: # Check if TensorFlow is installed and if GPU is available

import tensorflow as tf
```

```
print("TensorFlow version:", tf.__version__)
print("GPU available:", tf.config.list_physical_devices('GPU'))
```

TensorFlow version: 2.16.1

GPU available: []

In [ ]: *# Load Data*

```
import os
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from sklearn.model_selection import train_test_split

# Load the labels
train_labels = pd.read_csv('data/train_labels.csv')
sample_submission = pd.read_csv('data/sample_submission.csv')

# The images sizes are 96x96 despite the competition focus on the center 32x32
IMAGE_SIZE = (96, 96)
CROP_SIZE = (32, 32)

# Crop an image and return only the center 32x32 pixels
def crop_center(image, crop_size=CROP_SIZE):
    center_x, center_y = image.shape[1] // 2, image.shape[0] // 2
    half_crop_size_x, half_crop_size_y = crop_size[1] // 2, crop_size[0] // 2
    return image[center_y - half_crop_size_y:center_y + half_crop_size_y,
                  center_x - half_crop_size_x:center_x + half_crop_size_x, :]

# Load, crop, and normalize the images
def load_image(image_path, crop=False):
    image = load_img(image_path, target_size=IMAGE_SIZE)
    image = img_to_array(image)
    if crop:
        image = crop_center(image)
    image = image / 255.0
    return image

def load_images(image_ids, image_dir, crop=False):
    image_paths = [os.path.join(image_dir, f'{image_id}.tif') for image_id in image_ids]
    images = [load_image(image_path, crop) for image_path in image_paths]
    return np.array(images)

# Load the images and labels
image_ids = train_labels['id'].values
labels = train_labels['label'].values

# Split the data into training and validation sets
train_ids, val_ids, train_labels, val_labels = train_test_split(image_ids, labels,
                                                                test_size=0.1,
                                                                random_state=42)

# Define a fraction of the dataset to use for testing
fraction = 1 # Use 0.1 for quick testing

# Sample the training and validation IDs
train_sample_ids = np.random.choice(train_ids, size=int(len(train_ids) * fraction),
                                     replace=False)
```

```

val_sample_ids = np.random.choice(val_ids, size=int(len(val_ids) * fraction))

# Load the training and validation images with cropping
train_images = load_images(train_sample_ids, 'data/train', crop=True)
val_images = load_images(val_sample_ids, 'data/train', crop=True)

# Convert labels to numpy arrays
train_labels = np.array([labels[image_ids.tolist().index(id)] for id in train_sample_ids])
val_labels = np.array([labels[image_ids.tolist().index(id)] for id in val_sample_ids])

# Check data shapes and types
print(f"train_images shape: {train_images.shape}, dtype: {train_images.dtype}")
print(f"train_labels shape: {train_labels.shape}, dtype: {train_labels.dtype}")
print(f"val_images shape: {val_images.shape}, dtype: {val_images.dtype}")
print(f"val_labels shape: {val_labels.shape}, dtype: {val_labels.dtype}")

# Check for NaNs in the data
print(f"NaNs in train_images: {np.isnan(train_images).sum()}")
print(f"NaNs in train_labels: {np.isnan(train_labels).sum()}")
print(f"NaNs in val_images: {np.isnan(val_images).sum()}")
print(f"NaNs in val_labels: {np.isnan(val_labels).sum()}")

```

```

train_images shape: (176020, 32, 32, 3), dtype: float32
train_labels shape: (176020,), dtype: int64
val_images shape: (44005, 32, 32, 3), dtype: float32
val_labels shape: (44005,), dtype: int64
NaNs in train_images: 0
NaNs in train_labels: 0
NaNs in val_images: 0
NaNs in val_labels: 0

```

In [ ]: # Load Plot Function

```

import matplotlib.pyplot as plt

def plot_training_history(history):
    """
    Plots the training and validation accuracy and loss from the history object.

    Parameters:
    history (History): The history object returned by the fit method of a Keras model.
    """
    plt.figure(figsize=(12, 4))

    # Plot training & validation accuracy values
    plt.subplot(1, 2, 1)
    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.xticks(range(len(history.history['accuracy']))) # Set x-ticks to be epochs

    # Plot training & validation loss values
    plt.subplot(1, 2, 2)

```

```
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.xticks(range(len(history.history['loss']))) # Set x-ticks to be int

plt.show()
```

## Build and Train Model

In building the model, I tried not to use too many layers because each layer is additional computation. The main layers in my model attempt to make use of the RGB information first, then pool by half the amount of data, flatten, drop some of the data to prevent over-fitting, then look for more complex, dense patterns.

I included early stopping so if after a specified number of epochs there is no improvement in the monitored metric, it stops training. I try to also select the best model based on the validation accuracy and loss.

The longer I went on playing with this project, the more I wanted to improve the models accuracy. You'll see I added more to the model, trying to capture information by performing convolutions of different sizes since the sample images tend to have blobs of different sizes. I also increased the patience, lowered the min\_delta, and increased the maximum number of epoch to let the model over train itself. The early stopping class and best model selection should select the best one. Sometimes the accuracy goes down for a few epochs before increasing again, so it's worth seeing what happens.

After the training section, there are two graphs which show the model accuracy and loss for training and validation.

These are the configurations I used for submission. In the code blocks below, I used the same cropped image size and training data for all models just to show 1 for 1 comparison.

Model #	Image Size	Training Data Proportion	Kaggle Accuracy**
1	32x32	0.1	0.7652
2	32x32	0.1	0.7853
3*	96x96	0.1	0.8472
3	96x96	1	0.8266
3	32x32	1	0.

\*Model 3 was trained with uncropped images (96x96), where models 1 and 2 were trained on center cropped images (32x32).

**\*\***I also don't remember if I trained my model on the full data set for submissions 1 and 2. I don't think I did.

```
In [ ]: # Model 1




















import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

# Define a simple model
model1 = Sequential([
    Input(shape=(32, 32, 3)),
    Conv2D(16, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(32, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

# Compile the model
model1.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model with verbose output
history1 = model1.fit(train_images, train_labels, epochs=20, validation_data=(val_images, val_labels))

plot_training_history(history1)
```

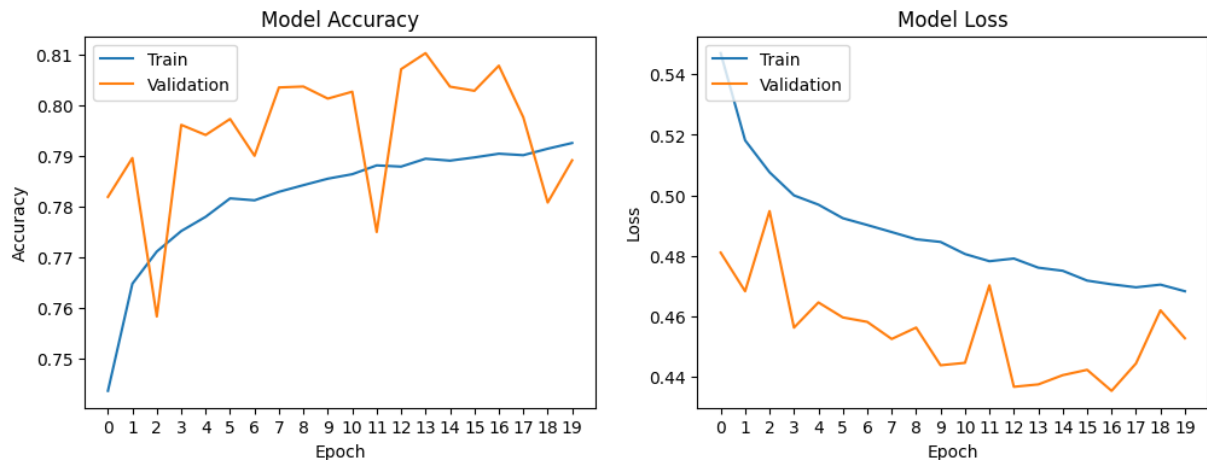
Epoch 1/20  
5501/5501  15s 3ms/step - accuracy: 0.7206 - loss: 0.569  
4 - val\_accuracy: 0.7818 - val\_loss: 0.4811  
Epoch 2/20  
5501/5501  15s 3ms/step - accuracy: 0.7605 - loss: 0.523  
1 - val\_accuracy: 0.7895 - val\_loss: 0.4683  
Epoch 3/20  
5501/5501  15s 3ms/step - accuracy: 0.7710 - loss: 0.507  
2 - val\_accuracy: 0.7582 - val\_loss: 0.4948  
Epoch 4/20  
5501/5501  15s 3ms/step - accuracy: 0.7765 - loss: 0.500  
2 - val\_accuracy: 0.7961 - val\_loss: 0.4563  
Epoch 5/20  
5501/5501  15s 3ms/step - accuracy: 0.7788 - loss: 0.495  
6 - val\_accuracy: 0.7941 - val\_loss: 0.4646  
Epoch 6/20  
5501/5501  15s 3ms/step - accuracy: 0.7831 - loss: 0.491  
6 - val\_accuracy: 0.7972 - val\_loss: 0.4597  
Epoch 7/20  
5501/5501  15s 3ms/step - accuracy: 0.7799 - loss: 0.490  
8 - val\_accuracy: 0.7900 - val\_loss: 0.4582  
Epoch 8/20  
5501/5501  15s 3ms/step - accuracy: 0.7826 - loss: 0.486  
9 - val\_accuracy: 0.8035 - val\_loss: 0.4525  
Epoch 9/20  
5501/5501  15s 3ms/step - accuracy: 0.7843 - loss: 0.485  
3 - val\_accuracy: 0.8036 - val\_loss: 0.4563  
Epoch 10/20  
5501/5501  15s 3ms/step - accuracy: 0.7866 - loss: 0.483  
4 - val\_accuracy: 0.8012 - val\_loss: 0.4439  
Epoch 11/20  
5501/5501  15s 3ms/step - accuracy: 0.7853 - loss: 0.481  
1 - val\_accuracy: 0.8026 - val\_loss: 0.4447  
Epoch 12/20  
5501/5501  16s 3ms/step - accuracy: 0.7882 - loss: 0.478  
1 - val\_accuracy: 0.7749 - val\_loss: 0.4703  
Epoch 13/20  
5501/5501  16s 3ms/step - accuracy: 0.7882 - loss: 0.477  
9 - val\_accuracy: 0.8071 - val\_loss: 0.4367  
Epoch 14/20  
5501/5501  15s 3ms/step - accuracy: 0.7903 - loss: 0.474  
1 - val\_accuracy: 0.8102 - val\_loss: 0.4375  
Epoch 15/20  
5501/5501  15s 3ms/step - accuracy: 0.7911 - loss: 0.472  
8 - val\_accuracy: 0.8036 - val\_loss: 0.4406  
Epoch 16/20  
5501/5501  15s 3ms/step - accuracy: 0.7907 - loss: 0.471  
8 - val\_accuracy: 0.8028 - val\_loss: 0.4423  
Epoch 17/20  
5501/5501  15s 3ms/step - accuracy: 0.7911 - loss: 0.469  
7 - val\_accuracy: 0.8077 - val\_loss: 0.4354  
Epoch 18/20  
5501/5501  15s 3ms/step - accuracy: 0.7910 - loss: 0.469  
2 - val\_accuracy: 0.7976 - val\_loss: 0.4445  
Epoch 19/20  
5501/5501  15s 3ms/step - accuracy: 0.7915 - loss: 0.469

5 - val\_accuracy: 0.7807 - val\_loss: 0.4620

Epoch 20/20

**5501/5501** ————— **15s** 3ms/step - accuracy: 0.7919 - loss: 0.469

6 - val\_accuracy: 0.7891 - val\_loss: 0.4528



In [ ]: *# Model 2*

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Dense, Dropout, Input, BatchNormal

# Define a simple model
model2 = Sequential([
    Input(shape=(32, 32, 3)),

    Conv2D(4, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),

    Conv2D(8, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),

    Conv2D(16, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),

    GlobalAveragePooling2D(),


    Dense(32, activation='relu'),
    Dropout(0.33),


    Dense(1, activation='sigmoid')
])


# Compile the model
model2.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])


# Train the model with verbose output
history2 = model2.fit(train_images, train_labels, epochs=20, validation_data=(val_images, val_labels))


plot_training_history(history2)
```


Epoch 1/20  
**5501/5501**  **55s** 10ms/step - accuracy: 0.7657 - loss: 0.5009 - val\_accuracy: 0.7889 - val\_loss: 0.4600


Epoch 2/20  
**5501/5501**  **58s** 10ms/step - accuracy: 0.7992 - loss: 0.4497 - val\_accuracy: 0.8041 - val\_loss: 0.4334


Epoch 3/20  
**5501/5501**  **66s** 12ms/step - accuracy: 0.8045 - loss: 0.4414 - val\_accuracy: 0.8169 - val\_loss: 0.4143


Epoch 4/20  
**5501/5501**  **67s** 12ms/step - accuracy: 0.8056 - loss: 0.4368 - val\_accuracy: 0.8160 - val\_loss: 0.4148


Epoch 5/20  
**5501/5501**  **67s** 12ms/step - accuracy: 0.8123 - loss: 0.4253 - val\_accuracy: 0.8245 - val\_loss: 0.4010


Epoch 6/20  
**5501/5501**  **68s** 12ms/step - accuracy: 0.8136 - loss: 0.4242 - val\_accuracy: 0.8135 - val\_loss: 0.4281


Epoch 7/20  
**5501/5501**  **72s** 13ms/step - accuracy: 0.8176 - loss: 0.4177 - val\_accuracy: 0.7902 - val\_loss: 0.4732


Epoch 8/20  
**5501/5501**  **84s** 15ms/step - accuracy: 0.8213 - loss: 0.4091 - val\_accuracy: 0.8247 - val\_loss: 0.4205


Epoch 9/20  
**5501/5501**  **82s** 15ms/step - accuracy: 0.8239 - loss: 0.4058 - val\_accuracy: 0.8254 - val\_loss: 0.4067


Epoch 10/20  
**5501/5501**  **186s** 34ms/step - accuracy: 0.8250 - loss: 0.4024 - val\_accuracy: 0.8158 - val\_loss: 0.4136


Epoch 11/20  
**5501/5501**  **88s** 16ms/step - accuracy: 0.8279 - loss: 0.3999 - val\_accuracy: 0.7909 - val\_loss: 0.4942


Epoch 12/20  
**5501/5501**  **90s** 16ms/step - accuracy: 0.8260 - loss: 0.4002 - val\_accuracy: 0.8156 - val\_loss: 0.4063


Epoch 13/20  
**5501/5501**  **87s** 16ms/step - accuracy: 0.8286 - loss: 0.3960 - val\_accuracy: 0.7979 - val\_loss: 0.4556


Epoch 14/20  
**5501/5501**  **84s** 15ms/step - accuracy: 0.8293 - loss: 0.3942 - val\_accuracy: 0.8129 - val\_loss: 0.4181

Epoch 15/20  
**5501/5501**  **87s** 16ms/step - accuracy: 0.8318 - loss: 0.3924 - val\_accuracy: 0.8302 - val\_loss: 0.3852

Epoch 16/20  
**5501/5501**  **79s** 14ms/step - accuracy: 0.8299 - loss: 0.3934 - val\_accuracy: 0.7689 - val\_loss: 0.5387

Epoch 17/20  
**5501/5501**  **84s** 15ms/step - accuracy: 0.8311 - loss: 0.3905 - val\_accuracy: 0.8275 - val\_loss: 0.3854

Epoch 18/20  
**5501/5501**  **85s** 15ms/step - accuracy: 0.8334 - loss: 0.3887 - val\_accuracy: 0.8248 - val\_loss: 0.4011

Epoch 19/20  
**5501/5501**  **88s** 16ms/step - accuracy: 0.8337 - loss: 0.38

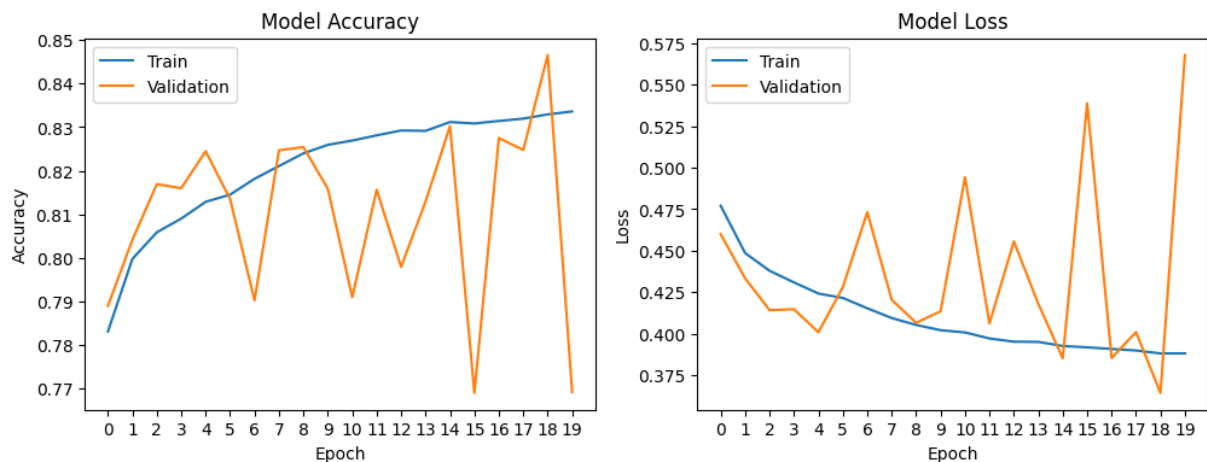


58 - val\_accuracy: 0.8466 - val\_loss: 0.3645

Epoch 20/20

**5501/5501** ————— **91s** 17ms/step - accuracy: 0.8343 - loss: 0.38

62 - val\_accuracy: 0.7691 - val\_loss: 0.5677



In [ ]: *# Model 3*

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Dense, Dropout, Input, BatchNorm

# Define a simple model
model3 = Sequential([
    Input(shape=(32, 32, 3)),

    Conv2D(4, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),

    Conv2D(8, (3, 3), activation='relu', padding='same', strides=(2, 2)),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),

    Conv2D(16, (3, 3), activation='relu', padding='same', strides=(2, 2)),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),

    Conv2D(32, (3, 3), activation='relu', padding='same', strides=(2, 2)),
    BatchNormalization(),
    GlobalAveragePooling2D(),




















    Dense(32, activation='relu'),
    Dropout(0.33),

    Dense(1, activation='sigmoid')
])

# Compile the model
model3.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model with verbose output
history3 = model3.fit(train_images, train_labels, epochs=20, validation_data=(val_images, val_labels))

plot_training_history(history3)
```

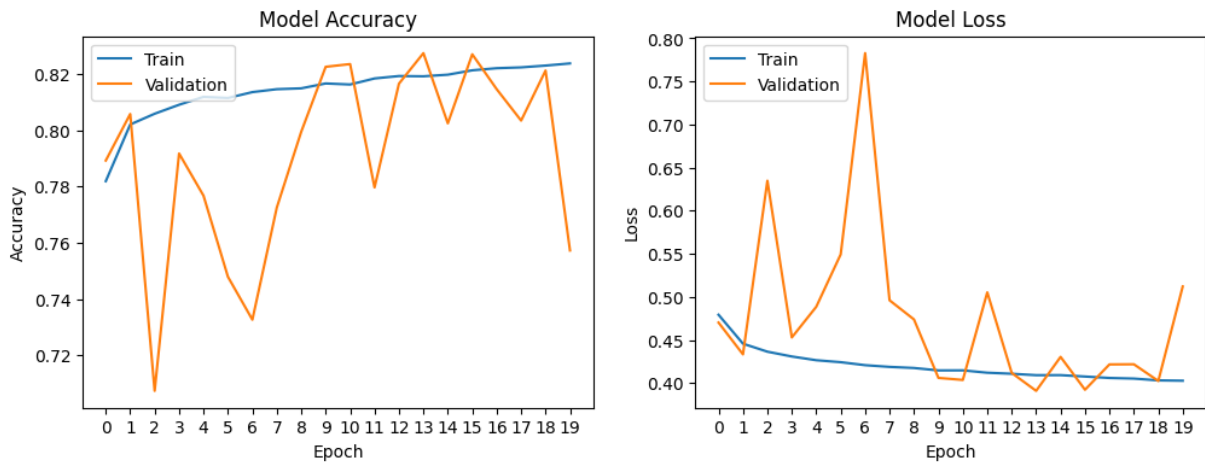
Epoch 1/20  
**5501/5501**  **49s** 9ms/step - accuracy: 0.7580 - loss: 0.517  
4 - val\_accuracy: 0.7892 - val\_loss: 0.4703  
Epoch 2/20  
**5501/5501**  **45s** 8ms/step - accuracy: 0.7999 - loss: 0.449  
3 - val\_accuracy: 0.8057 - val\_loss: 0.4334  
Epoch 3/20  
**5501/5501**  **45s** 8ms/step - accuracy: 0.8056 - loss: 0.436  
7 - val\_accuracy: 0.7073 - val\_loss: 0.6348  
Epoch 4/20  
**5501/5501**  **42s** 8ms/step - accuracy: 0.8085 - loss: 0.432  
3 - val\_accuracy: 0.7917 - val\_loss: 0.4530  
Epoch 5/20  
**5501/5501**  **46s** 8ms/step - accuracy: 0.8123 - loss: 0.426  
6 - val\_accuracy: 0.7767 - val\_loss: 0.4886  
Epoch 6/20  
**5501/5501**  **42s** 8ms/step - accuracy: 0.8110 - loss: 0.425  
4 - val\_accuracy: 0.7479 - val\_loss: 0.5494  
Epoch 7/20  
**5501/5501**  **40s** 7ms/step - accuracy: 0.8144 - loss: 0.418  
7 - val\_accuracy: 0.7326 - val\_loss: 0.7828  
Epoch 8/20  
**5501/5501**  **47s** 9ms/step - accuracy: 0.8146 - loss: 0.419  
7 - val\_accuracy: 0.7725 - val\_loss: 0.4961  
Epoch 9/20  
**5501/5501**  **43s** 8ms/step - accuracy: 0.8158 - loss: 0.415  
7 - val\_accuracy: 0.7995 - val\_loss: 0.4739  
Epoch 10/20  
**5501/5501**  **45s** 8ms/step - accuracy: 0.8169 - loss: 0.413  
9 - val\_accuracy: 0.8225 - val\_loss: 0.4061  
Epoch 11/20  
**5501/5501**  **46s** 8ms/step - accuracy: 0.8180 - loss: 0.414  
1 - val\_accuracy: 0.8235 - val\_loss: 0.4037  
Epoch 12/20  
**5501/5501**  **47s** 9ms/step - accuracy: 0.8178 - loss: 0.412  
9 - val\_accuracy: 0.7796 - val\_loss: 0.5053  
Epoch 13/20  
**5501/5501**  **46s** 8ms/step - accuracy: 0.8183 - loss: 0.411  
6 - val\_accuracy: 0.8165 - val\_loss: 0.4119  
Epoch 14/20  
**5501/5501**  **45s** 8ms/step - accuracy: 0.8196 - loss: 0.409  
5 - val\_accuracy: 0.8274 - val\_loss: 0.3911  
Epoch 15/20  
**5501/5501**  **45s** 8ms/step - accuracy: 0.8204 - loss: 0.408  
4 - val\_accuracy: 0.8024 - val\_loss: 0.4304  
Epoch 16/20  
**5501/5501**  **43s** 8ms/step - accuracy: 0.8236 - loss: 0.402  
4 - val\_accuracy: 0.8270 - val\_loss: 0.3924  
Epoch 17/20  
**5501/5501**  **45s** 8ms/step - accuracy: 0.8217 - loss: 0.406  
2 - val\_accuracy: 0.8146 - val\_loss: 0.4217  
Epoch 18/20  
**5501/5501**  **41s** 7ms/step - accuracy: 0.8219 - loss: 0.406  
5 - val\_accuracy: 0.8034 - val\_loss: 0.4219  
Epoch 19/20  
**5501/5501**  **47s** 8ms/step - accuracy: 0.8243 - loss: 0.402

1 - val\_accuracy: 0.8212 - val\_loss: 0.4025

Epoch 20/20

**5501/5501** ————— 42s 8ms/step - accuracy: 0.8236 - loss: 0.402

4 - val\_accuracy: 0.7572 - val\_loss: 0.5121



## Evaluate Model

```
In [ ]: # Choose Model for Evaluation/Submission
model = model3
```

```
In [ ]: # Evaluate the model on the validation set
val_loss, val_accuracy = model.evaluate(val_images, val_labels)
print(f'Validation Accuracy: {val_accuracy}')
print(f'Validation Loss: {val_loss}')
```

**1376/1376** ————— 3s 3ms/step - accuracy: 0.7562 - loss: 0.5109

Validation Accuracy: 0.757232129573822

Validation Loss: 0.5121429562568665

## Make Predictions

```
In [ ]: # Create Submission File

# Load the test images
test_ids = sample_submission['id'].values
test_images = load_images(test_ids, 'data/test', crop=True)

# Make predictions
predictions = model.predict(test_images)
predictions = (predictions > 0.5).astype(int).flatten()

# Prepare the submission file
submission = pd.DataFrame({'id': test_ids, 'label': predictions})
submission.to_csv('submission.csv', index=False)
```

**1796/1796** ————— 5s 2ms/step

Search

joyo8856

X

KAGGLE · PLAYGROUND PREDICTION COMPETITION · 5 YEARS AGO

Late Submission

Histopathologic Cancer Detection

Identify metastatic tissue in histopathologic scans of lymph node sections

Overview

Data

Code

Models

Discussion

Leaderboard

Rules

Team

Submissions

0/2

AllSuccessfulSelectedErrorsPublic Score ▾

Submission and Description	Private Score ⓘ	Public Score ⓘ	Selected
<div><div><div>✓</div><div>🕒</div></div><div>submission.csv</div><div>Complete (after deadline) · 20h ago · Third Model (Full Image / 0.1 Trai...</div></div>	0.7681	0.8472	<input type="checkbox"/>
<div><div><div>✓</div><div>🕒</div></div><div>submission.csv</div><div>Complete (after deadline) · 20h ago · Model 3, All Data, Full Image</div></div>	0.7582	0.8266	<input type="checkbox"/>
<div><div><div>✓</div><div>🕒</div></div><div>submission.csv</div><div>Complete (after deadline) · 21h ago · Second model</div></div>	0.7400	0.7853	<input type="checkbox"/>
<div><div><div>✓</div><div>🕒</div></div><div>submission.csv</div><div>Complete (after deadline) · 20h ago · Model Three (All Data, Cropped I...</div></div>	0.7459	0.7704	<input type="checkbox"/>
<div><div><div>✓</div><div>🕒</div></div><div>submission.csv</div><div>Complete (after deadline) · 2d ago · First try -- using training on only t...</div></div>	0.7261	0.7652	<input type="checkbox"/>

## Conclusion

I reformatted this notebook to have all three models run in their own cells and generate plots for review. They are all run with the cropped 32x32 images and the full training dataset.

For every Kaggle submissions I left some notes on what I did, and I think what is most interesting is comparing how the different configurations of model three performed. The first model performed the worst, but not unremarkably worse than the second.

The three model three submissions in increasing score: all data, cropped image; all data, full image; 10% data, full image. It appears that in the case of model three, more training data is not better, which means I must be over-fitting. It also appears that using the full 96x96 image is better than using the cropped images as the two models submitted with cropped images were at the end of the list. Initially I thought training on more than the

center 32x32 pixels would teach the model to use information it wouldn't have access to, but despite the competition claiming the challenge is to classifying the center of the image only, the test images are 96x96, which explains why using the full image in training leads to higher accuracy. If they really only wanted to classify the center of the image, that is all the data that should be used in evaluation.