Histopathologic Cancer Detection

August 22, 2023

1 Histopathologic Cancer Detection

1.1 Introduction

Kaggle competition: https://www.kaggle.com/competitions/histopathologic-cancer-detection/

The goal of this competition is to write an algorithm capable of identifying metastatic breast cancer from digital pathology scans. This clearly has strong implications in the medical imaging and diagnosis field. For the purpose of this class, we will be implementing a Convolutional Neural Network (CNN).

1.2 Setup

[1]:
```
# import statements
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, BatchNormalization,␣
 ↪Activation, Conv2D, MaxPooling2D, MaxPool2D
from sklearn.model_selection import train_test_split
from glob import glob
from PIL import Image
import os
```

[2]:
```
# path to data
path = "/Users/clairerobbins/Documents/MS-DS CU Boulder/Introduction to Deep␣
 ↪Learning/Histopathologic Cancer Detection/histopathologic-cancer-detection/"
train_path = path + 'train/'
test_path = path + 'test/'
```

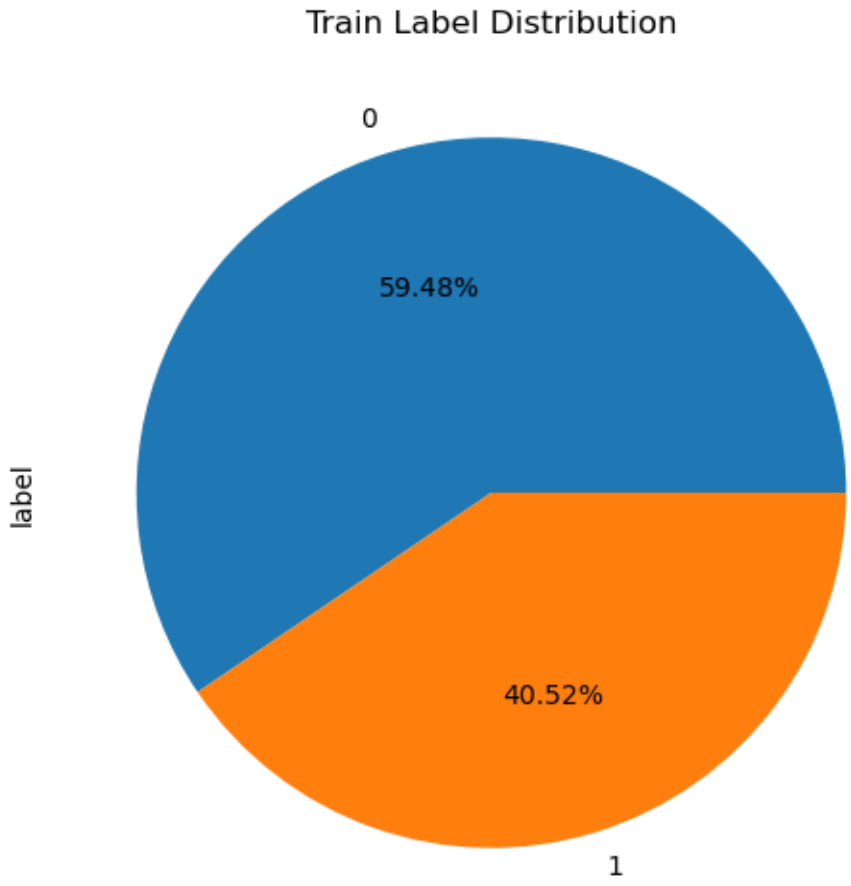
1.3 EDA

[3]:
```
# define & clean train dataframe
df_train = pd.DataFrame({'path': glob(os.path.join(train_path,'*.tif'))})
df_train['id'] = df_train.path.map(lambda x: x.split('/')[9].split(".")[0])
labels = pd.read_csv(train_path+"train_labels.csv")
df_train = df_train.merge(labels, on = "id")
df_train.head(3)
```

[3]:

	path	id	label
0	/Users/clairerobbins/Documents/MS-DS CU Boulde…	99ef485f205645918613cd04281098daa7c17819	1
1	/Users/clairerobbins/Documents/MS-DS CU Boulde…	6d1bb57c0606f05dbd75f90a8d9e21a57e1267e0	0
2	/Users/clairerobbins/Documents/MS-DS CU Boulde…	9c043ab2adadfeb758c71d21432fccd3e43565c0	1

[4]:
```
# visualize classification breakdown
df_train['label'].value_counts().plot(figsize=(6,6),kind='pie',autopct='%.2f%%')
plt.title('Train Label Distribution')
plt.show()
```



[5]:
```
# define & clean test dataframe
df_test = pd.DataFrame({'path': glob(os.path.join(test_path,'*.tif'))})
df_test['id'] = df_test.path.map(lambda x: x.split('/')[9].split(".")[0])
df_test['label'] = [None for i in range(len(df_test))]
df_test.head(3)
```

[5]:
```
                                                path  \
0  /Users/clairerobbins/Documents/MS-DS CU Boulde…
1  /Users/clairerobbins/Documents/MS-DS CU Boulde…
2  /Users/clairerobbins/Documents/MS-DS CU Boulde…

                                         id label
0  fd0a060ef9c30c9a83f6b4bfb568db74b099154d  None
1  1f9ee06f06d329eb7902a2e03ab3835dd0484581  None
2  19709bec800f372d0b1d085da6933dd3ef108846  None
```

1.4 Model Development & Architecture

In my initial set up, I chose to split my train data into train & validation data, to get a sense of how the model is performing in real time. Given the smaller image size, I started off with a simple model to minimize overfitting, and tuned my hyperparameters from that point. Additionally, I found I had to clean the data because some of the files downloaded from the were erroneous.

[6]:
```
# split into train & validate datasets
train, validate = train_test_split(df_train, test_size=0.2)
```

[7]:
```
# extract image metadata from each train image & drop invalid files
def clean_files(df):
    raw_data = []
    labels = []
    for index,row in df.iterrows():
        try:
            raw_data.append(np.array(Image.open(row['path'])))
            labels.append(row['label'])
        except:
            pass
    return np.array(raw_data), np.array(labels)
```

[8]:
```
# run clean function on data
train_raw, train_labels = clean_files(train)
validate_raw, validate_labels = clean_files(validate)
```

[9]:
```
kernel_size = (3, 3)
pool_size = (2, 2)
first_filters = 16
second_filters = 32
third_filters = 64

dropout_conv = 0.2
dropout_dense = 0.4

model = Sequential()

# Conv layer 1
model.add(Conv2D(first_filters, kernel_size, input_shape=(96, 96, 3)))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(MaxPool2D(pool_size=pool_size))
model.add(Dropout(dropout_conv))

# Conv layer 2
model.add(Conv2D(second_filters, kernel_size))
model.add(BatchNormalization())
```

```
model.add(Activation("relu"))
model.add(MaxPool2D(pool_size=pool_size))
model.add(Dropout(dropout_conv))

# Conv layer 3
model.add(Conv2D(third_filters, kernel_size))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(MaxPool2D(pool_size=pool_size))
model.add(Dropout(dropout_conv))

# Fully connected (dense) layer
model.add(Flatten())
model.add(Dense(128))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(Dropout(dropout_dense))

# Activation function
model.add(Dense(1, activation="sigmoid"))

batch_size = 64
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 94, 94, 16)        448

 batch_normalization (Batch  (None, 94, 94, 16)        64
 Normalization)

 activation (Activation)     (None, 94, 94, 16)        0

 max_pooling2d (MaxPooling2  (None, 47, 47, 16)        0
 D)

 dropout (Dropout)           (None, 47, 47, 16)        0

 conv2d_1 (Conv2D)           (None, 45, 45, 32)        4640

 batch_normalization_1 (Bat  (None, 45, 45, 32)        128
 chNormalization)
```

```
activation_1 (Activation)   (None, 45, 45, 32)        0

max_pooling2d_1 (MaxPoolin  (None, 22, 22, 32)        0
g2D)

dropout_1 (Dropout)         (None, 22, 22, 32)        0

conv2d_2 (Conv2D)           (None, 20, 20, 64)        18496

batch_normalization_2 (Bat  (None, 20, 20, 64)        256
chNormalization)

activation_2 (Activation)   (None, 20, 20, 64)        0

max_pooling2d_2 (MaxPoolin  (None, 10, 10, 64)        0
g2D)

dropout_2 (Dropout)         (None, 10, 10, 64)        0

flatten (Flatten)           (None, 6400)              0

dense (Dense)               (None, 128)               819328

batch_normalization_3 (Bat  (None, 128)               512
chNormalization)

activation_3 (Activation)   (None, 128)               0

dropout_3 (Dropout)         (None, 128)               0

dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 844001 (3.22 MB)
Trainable params: 843521 (3.22 MB)
Non-trainable params: 480 (1.88 KB)
_________________________________________________________________
```

[10]:
```
# model fit
model.fit(train_raw, train_labels, epochs=10, validation_data=(validate_raw,␣
 ↪validate_labels))
```

```
Epoch 1/10
3530/3530 [==============================] - 162s 46ms/step - loss: 0.3989 -
accuracy: 0.8264 - val_loss: 0.7339 - val_accuracy: 0.6742
Epoch 2/10
3530/3530 [==============================] - 155s 44ms/step - loss: 0.3429 -
```

```
accuracy: 0.8545 - val_loss: 0.4416 - val_accuracy: 0.7947
Epoch 3/10
3530/3530 [==============================] - 159s 45ms/step - loss: 0.2981 -
accuracy: 0.8771 - val_loss: 0.3924 - val_accuracy: 0.8252
Epoch 5/10
3530/3530 [==============================] - 159s 45ms/step - loss: 0.2635 -
accuracy: 0.8931 - val_loss: 0.5034 - val_accuracy: 0.7757
Epoch 7/10
3530/3530 [==============================] - 159s 45ms/step - loss: 0.2502 -
accuracy: 0.9009 - val_loss: 0.6774 - val_accuracy: 0.8097
Epoch 8/10
3530/3530 [==============================] - 158s 45ms/step - loss: 0.2401 -
accuracy: 0.9056 - val_loss: 0.6318 - val_accuracy: 0.7655
Epoch 9/10
3530/3530 [==============================] - 158s 45ms/step - loss: 0.2308 -
accuracy: 0.9092 - val_loss: 0.2291 - val_accuracy: 0.9058
Epoch 10/10
3530/3530 [==============================] - 160s 45ms/step - loss: 0.2254 -
accuracy: 0.9126 - val_loss: 0.2266 - val_accuracy: 0.9093
```

[10]: `<keras.src.callbacks.History at 0x29fe0f0d0>`

[ ]:

1.5 Hyperparameter Tuning & Analysis

Here is my rationale for the chosen hyperparameters:

1. **Reduced Number of Filters**:
   - Improve training times while still capturing relevant features in your images and prevent overfitting
2. **Adjusted Dropout Rates**:
   - Retaining more information from the network's activations
3. **Smaller Dense Layer (128 neurons)**:
   - Reduced dense layer from 256 neurons to 128 neurons, to execute faster training and less risk of overfitting
4. **Batch Normalization**:
   - Faster convergence and better generalization
5. **Activation Functions**:
   - ReLU (Rectified Linear Activation) is used as the activation function after each convolutional and dense layer.
   - Computationally efficient and helps mitigate the vanishing gradient problem
6. **Loss Function and Optimizer**:
   - Binary cross-entropy loss is a natural choice for binary classification
   - The Adam optimizer is chosen for its adaptive learning rate and efficient convergence

1.6 Predictions

[11]:
```
# extract image metadata from each test image
def clean_files_test(df):
    ids = []
    raw_data = []
    for index,row in df.iterrows():
        raw_data.append(np.array(Image.open(row['path'])))
    return np.array(raw_data)
```

[12]:
```
# run clean function on test data
test_raw = clean_files_test(df_test)
```

[13]:
```
# predict using trained model
predictions = model.predict(test_raw)
```

```
1796/1796 [==============================] - 18s 10ms/step
```

[14]:
```
# convert prediction percentages to binary classification
final_predictions = []
for i in predictions:
    if i > 0.5:
        final_predictions.append(1)
    else:
        final_predictions.append(0)
```

[15]:
```
# generate submission file
submission = pd.DataFrame({'id': df_test['id'], 'label': final_predictions})
submission.to_csv('submission.csv')
```

1.7 Conclusion

My model achieved a 91% accuracy on the training data, 90% on the validation data, and 85% on my kaggle submission. This indicates there was some aspect of overfitting, although not an agregious amount.

In an effort to prevent overfitting, I implemented concepts such as fewer filters per convolutional layer and coupled with calibrated dropout rates. This project aimed to excel in pinpointing cancerous patterns within small digital pathology images. Progress in this space fuels the ongoing quest for sharper, swifter cancer diagnoses, propelling CNNs' potential in medical diagnostics.