Medicine Case Study Report

Alzheimer's disease has disrupted the lives of millions of individuals and their families. The disease is the most common type of dementia that occurs when parts of the brain used for learning, logical reasoning, memory and language are damaged or diseased. As of 2020, there were approximately 50 million people worldwide diagnosed with the disease and about 5.8 million reported cases in the U.S. The disease tends to affect those who are over the age of 65, but 10% of cases are of those between the ages of 30 and 65. Alzheimer's disease is progressive and irreversible and over time the production of an individual's brain cells starts to decline leading to cell death and brain shrinkage. There is no cure for the disease, but if detected early there are medications and therapies to help improve symptoms.

With advances in Deep Learning we are able to identify and extract features from images such as Magnetic Resonance Images (MRI) that can help with the detection of Alzheimer's disease. For this project I found an Alzheimer's disease dataset on Kaggle that contains over 6,000 MRI images separated into a train/test split and labeled based on different stages of dementia. To implement this solution I created an image classification model to analyze and determine the stage of dementia. The model that I created contains the standard Convolutional Neural Network architecture stack and to help improve the model performance I used techniques such as data augmentation, regularization and even created another model using transfer learning.

Overall the results were reasonable, but not as perfect as it should be for detecting Alzheimer's disease. After training the model and using various techniques to improve the performance the best model was the initial model that I created with an added dropout layer. The accuracy achieved for this model was 99% and validation accuracy 64%, which in this case is a good starting point. In conclusion, with my knowledge of how image classification works I believe that we could potentially achieve a validation accuracy of 90% or above if we are able to gain access to more data to feed into the model. However, this could potentially be a drawback since we would need to establish relationships with medical professionals and get consent from patients.

Detection for Alzheimer's Disease Image Classification

Installation

Create a new notebook file in Jupyter Notebook or Google Colab

Import required python libraries down below. If you do not have the libraries installed on your machine please install using pip.

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
```

Download Dataset from Kaggle and Upload to Google Drive

Download Alzheimer's Disease dataset from (Kaggle) and upload to Google Drive.

Mount drive to Colab to Access files and download zip

In order to load the data I needed to upload the zip file to Google Drive and unzip it in Colab.

```
from google.colab import drive
drive.mount("/content/drive")


#Download zip file and create a directory for the data

!unzip "/content/drive/MyDrive/alzheimers_data.zip" -d
"/content/drive/MyDrive/"
```

Load Data for Google Drive for train/test split

In this section there are two directories listed. The directories listed include one for the train data and another for the test data, which I used to gain access to the data directories using keras utils.

```
# Directory for training data
train_dir = '/content/drive/MyDrive/alzheimers_data/train'

# Use Keras Util to load training data from the directory

train_data = tf.keras.utils.image_dataset_from_directory(
 train_dir,
 validation_split=0.2,
 subset="training",
 seed=123,
 image_size=(224, 224),
 batch_size=32)

# Directory for validation data
test_dir = '/content/drive/MyDrive/alzheimers_data/test'

# Use Keras Util to load training data from the directory
test_data = tf.keras.utils.image_dataset_from_directory(
 test_dir,
 validation_split=0.2,
 subset="training",
 seed=123,
 image_size=(224, 224),
 batch_size=32)
```

Confirm the class names

There are 4 classes or labels that we will try to identify during the training process.

```
# Print class names
class_names = train_data.class_names
print(class_names)
```

Visualizing Random Images from the Alzheimer's Dataset

In this visualization there are 9 random images listed from the train dataset with labels.

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
for images, labels in train_data.take(1):
 for i in range(9):
   ax = plt.subplot(3, 3, i + 1)
   plt.imshow(images[i].numpy().astype("uint8"))
   plt.title(class_names[labels[i]])
   plt.axis("off")


# confirm image_batch and label shape

for image_batch, labels_batch in train_data:
 print(image_batch.shape)
 print(labels_batch.shape)
 break
```

Create the initial model

The model follows the standard architecture for image classification. For the first layer we standardize the images so that they fall in between a range of 0 and 1. Next we add three CNN layers with a relu activation and max pooling in between. Lastly we add fully connected layers to flatten the inputs.

```
#create model
model = Sequential()
model.add(layers.Rescaling(1./255, input_shape=(224, 224, 3)))
model.add(layers.Conv2D(16, (3, 3), padding='same', activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(32, (3,3), padding='same', activation='relu'))
```

```
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(4))


# get model summary
model.summary()
```

Compile and Train the Model

Here I defined a model checkpoint and earlystopping as learned in class, that would be applied while training the model.

```
#define model checkpoint and earlystopping
checkpoint_cb = keras.callbacks.ModelCheckpoint("alzheimers1.h5",
save_best_only=True)
early_stopping_cb = keras.callbacks.EarlyStopping(patience=10,
restore_best_weights=True)



#compile the model
model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(
 train_data,
 validation_data=val_data,
 epochs=30,
 callbacks=[checkpoint_cb, early_stopping_cb])
```

Visualize Model Results

To visualize the model results we create a plot for the training and validation accuracy and another plot for the training and validation loss.

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

After training the model the best achieved results for the training accuracy was 100% and validation accuracy 62%. The difference in the training and validation accuracy indicates that the model is overfitting and that we will need to make some improvements.

Changing the inputs for the model

My first effort to improve the model was to increase the inputs for the CNN layers, which resulted in a decrease in the training accuracy from 100% to 98% and validation accuracy from 62% to 43%.

```
model = Sequential()
model.add(layers.Rescaling(1./255, input_shape=(224, 224, 3)))
```

```
model.add(layers.Conv2D(32, (3, 3), padding='same', activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3,3), padding='same', activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), padding='same', activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(4))


# Retrieve model summary
model.summary()
```

Compile and Train the Model

```
model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

#define model checkpoint and earlystopping
checkpoint_cb = keras.callbacks.ModelCheckpoint("alzheimers2.h5",
save_best_only=True)
early_stopping_cb = keras.callbacks.EarlyStopping(patience=10,
restore_best_weights=True)


history = model.fit(
 train_data,
 validation_data=val_data,
 epochs=30,
 callbacks=[checkpoint_cb, early_stopping_cb]
)
```

Visualize Model Results

To visualize the model results we create a plot for the training and validation accuracy and another plot for the training and validation loss.

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

Regularization

In my next attempt to improve the model I added an l1 regularizer to prevent overfitting. The results did not appear to change much and the training and validation accuracy decreased. The results for this model included a training accuracy of 99% and validation accuracy of 56%.

```
from keras import regularizers
```

```
model = Sequential()
model.add(layers.Rescaling(1./255, input_shape=(224, 224, 3)))
model.add(layers.Conv2D(16, (3, 3), padding='same', activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(32, (3,3), padding='same', activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(128, kernel_regularizer=regularizers.l1(0.0001),
activation='relu'))
model.add(layers.Dense(4))
```

Compile and Train the Model

```
model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

#define model checkpoint and earlystopping
checkpoint_cb = keras.callbacks.ModelCheckpoint("alzheimers3_reg.h5",
save_best_only=True)
early_stopping_cb = keras.callbacks.EarlyStopping(patience=10,
restore_best_weights=True)


history = model.fit(
 train_data,
 validation_data=val_data,
 epochs=30,
 callbacks=[checkpoint_cb, early_stopping_cb]
)

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
```

```
epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

Data Augmentation:

Due to the limited amount of training data I decided to use data augmentation to help generate additional data to prevent overfitting. After using the technique there were no improvements and the training accuracy decreased to 74% and the validation accuracy decreased to 50%.

```
data_augmentation = keras.Sequential(
 [
   layers.RandomFlip("horizontal",
                     input_shape=(224,
                                  224,
                                  3)),
   layers.RandomRotation(0.1),
   layers.RandomZoom(0.1),
 ])


plt.figure(figsize=(10, 10))
for images, _ in train_data.take(1):
 for i in range(9):
```

```
    augmented_images = data_augmentation(images)
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[0].numpy().astype("uint8"))
    plt.axis("off")
```

Create model with data augmentation:

```
model = Sequential()
model.add(data_augmentation)
model.add(layers.Rescaling(1./255, input_shape=(224, 224, 3)))
model.add(layers.Conv2D(16, (3, 3), padding='same', activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(32, (3,3), padding='same', activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(4))

model.summary()
```

Compile and Train the Model

```
model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

#define model checkpoint and earlystopping
checkpoint_cb = keras.callbacks.ModelCheckpoint("alzheimers4_aug.h5",
save_best_only=True)
early_stopping_cb = keras.callbacks.EarlyStopping(patience=10,
restore_best_weights=True)
```

```
history = model.fit(
 train_data,
 validation_data=val_data,
 epochs=30,
 callbacks=[checkpoint_cb, early_stopping_cb]
)

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

Adding a Dropout Layer:

After adding a dropout layer to prevent overfitting I noticed a slight improvement in the model. The training accuracy remained around 100% while the validation accuracy increased to about 64%. As a result, I used this model to make predictions.

```
model = Sequential()
model.add(layers.Rescaling(1./255, input_shape=(224, 224, 3)))
model.add(layers.Conv2D(16, (3, 3), padding='same', activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(32, (3,3), padding='same', activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.2))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(4))

model.summary()
```

Compile and Train the Model

```
model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

#define model checkpoint and earlystopping
checkpoint_cb = keras.callbacks.ModelCheckpoint("alzheimers5_dropout.h5",
save_best_only=True)
early_stopping_cb = keras.callbacks.EarlyStopping(patience=10,
restore_best_weights=True)


history = model.fit(
 train_data,
 validation_data=val_data,
 epochs=30,
 callbacks=[checkpoint_cb, early_stopping_cb]
)
```

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

Making Predictions

In order to make predictions I found a random NonDemented MRI brain scan online not related to the train or validation dataset and used the dropout model created above to make the prediction, which was accurate. The percent confidence after making the prediction was 84%.

```
mri_img = '/content/test_mri_image.jpeg'
img = tf.keras.utils.load_img(mri_img, target_size=(224, 224))
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])
```

```
print(
    "This image most likely belongs to {} with a {:.2f} percent
confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```

Transfer Learning

Out of curiosity I decided to experiment with using the pretrained VGG16 model. The results after running the model were 79% for the training accuracy and 56% for the validation accuracy, which was not better than my initial model but could be a good starting point.

Loading the data using ImageDataGenerator

```
#creating train, test and validator sets based on dog vs cat data

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        # This is the target directory
        train_dir,
        # All images will be resized to 150x150
        target_size=(224, 224),
        batch_size=32,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
        test_dir,
        target_size=(224, 224),
        batch_size=32,
        class_mode='binary')
```

Adding in pretrained model

```
from keras.applications import VGG16

conv_base = VGG16(weights='imagenet', include_top=False, input_shape=(224,
224, 3))

conv_base.summary()
```

Create Model

```
model = Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(4))


model.summary()
```

Compile and Train the Model

```
model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

#define model checkpoint and earlystopping
checkpoint_cb = keras.callbacks.ModelCheckpoint("alzheimers5_vgg16.h5",
save_best_only=True)
early_stopping_cb = keras.callbacks.EarlyStopping(patience=10,
restore_best_weights=True)
```

```
history = model.fit(
 train_data,
 validation_data=val_data,
 epochs=30,
 callbacks=[checkpoint_cb, early_stopping_cb]
)

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```