Project 3A Summary

Benson Ou-yang (301277342) Kaggle Username: BensonOuyang

December 5, 2021

Contents

1 Methods **2**

1.1 Reading the Data 2

1.2 Feature Engineering 2

1.3 Convolutional Neural Network 3

2 Results **3**

2.1 Colour 3

2.2 Image size 3

2.3 Data Augmentation 3

3 Submission Code **4**

3.1 Starting Code (Importing libraries and reading in data) 4

3.2 pred1.csv (CNN for grayscale inputs) 4

3.3 pred2.csv (pred1.csv rounded) 5

3.4 pred3.csv (all 0s) 5

3.5 pred4.csv (all 1s) 5

3.6 pred5.csv (Included RBG channels) 5

3.7 pred6.csv (More layers CNN) 7

3.8 pred7.csv (CNN with Data Augmentation) 7

1 Methods

1.1 Reading the Data

We used a for loop to loop through all the images in the training dataset, and from the package *imageio*, the function "imread" takes each image and produces an array of the shape (256,256,3). The NumPy array corresponds to the height, width and RGB colour channel. The package *PIL* uses the function "fromarray" and "resize" to change the height and width of each image. Next, inside the loop, the mean at the second axis is taken for the grayscale version of the picture. The final image array gets stored inside a NumPy array of size (Number of images, Height, Width).

1.2 Feature Engineering

Colour can be a feature that can help to identify kilns. The RGB array is kept by not taking the mean of the picture array at the second axis.

For the size of the images, we started with the image dimension of (28,28), but this is a small image thus can be difficult for the model to determine the correct identification of kilns. Afterwards, we tested the image sizes (200,200), (256,256), and (300,300) by splitting the training data into a training and testing set to evaluate the model's AUC.

Another way for a model to increase predictive power and decrease the risk of overfitting is to augment the images. It is also a way to increase the number of samples for the neural network. From the *albumentations* package, *RandomCrop, GaussianBlur, and Flip* changes the images in different ways. *RandomCrop* takes each image and randomly subsets a portion of an image. *GaussianBlur* blurs an image by using a Gaussian function. *Flip* flips an image either horizontally, vertically or both.

Figure 1: Original & RandomCrop Augmented Image

Figure 2: GaussianBlur & Flip Augmented Image

Figure 1 shows an image in its original state(left) and after augmenting with *RandomCrop*(right).

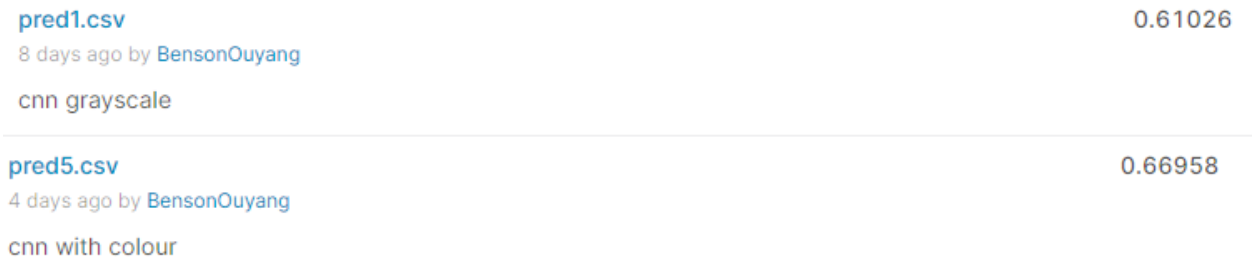
Figure 2 shows an image after *GuassianBlur* augmentation(left) and after augmenting with *Flip*(right).

1.3 Convolutional Neural Network

The first layer of our *Convolutional Neural Network* uses 16 filters with a ReLU activation function. The input shape of our data is (height, width, 3). Next, we have a max-pooling layer that divides each spatial dimension by 2 to reduce image size while retaining as much information as possible. Then the layers structure is repeated two times where each convolutional layer doubles the number of filters. Afterwards, we flatten the input to produce a 1D array of features for the two dense layers to produce the outputs. The kilns prediction is for only two classes, either zero for not a chimney kiln or one for either no kiln or modernized kiln. There are only two classes to predict, thus a sigmoid activation function in the last dense layer outputs either class.

2 Results

2.1 Colour



After submitting on Kaggle, the *Convolutional Neural Network* with colour inputs performed better than the grayscale inputs.

2.2 Image size

We chose the image dimensions (256,256) as the *Convolutional Neural Network* produced a higher mean validation AUC out of the epochs when compared to the dimensions (200,200) and (300,300). If the mean validation AUC was higher for the other dimensions, we would have explored in that direction.

2.3 Data Augmentation

Due to memory allocation issues, we removed the random cropped images so our final model inputs were the original, the gaussian blurred, and flipped images. With the right image dimensions, incorporating colour and augmentating data, we produced our best results.

3 Submission Code

3.1 Starting Code (Importing libraries and reading in data)

```
import tensorflow as tf
import imageio
import os
from PIL import Image
import pandas as pd
import numpy as np
```

```
N = tr.shape[0]
```

```
tr = pd.read_csv('Ytr.txt')
te = pd.read_csv('pred.txt')
```

```
def read_image(N,D,X):
    Xt = np.zeros([N,D,D])
    #found = list()
    for ii in range(N):
        if ii % 100 == 0:
            print('%d / %d' % (ii, N))
        teId = X['id'][ii]
        path = 'images/%05d.png' %teId

        pic = imageio.imread(path)
        pic = Image.fromarray(pic).resize((D,D)) # NOTE : this can be improved.
        pic = np.mean(pic,axis = 2) # NOTE: this can be improved
        pic = np.array(pic)
        # print(pic.shape)
        Xt[ii,:,:] = pic
    Xt = Xt/255
    return Xt
```

```
Xtr_1 = read_image(tr.shape[0],200,tr)
```

```
Xtr_2 = Xtr_1.reshape(-1,200,200,1)
```

```
Xte_1 = read_image(te.shape[0],200,te)
Xte_2 = Xte_1.reshape(-1,200,200,1)
```

3.2 pred1.csv (CNN for grayscale inputs)

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(16,(3,3), activation = 'relu', input_shape = (200,200,1)),
    tf.keras.layers.MaxPool2D(2,2),

    tf.keras.layers.Conv2D(32,(3,3), activation = 'relu'),
```

```
    tf.keras.layers.MaxPool2D(2,2),

    tf.keras.layers.Conv2D(64,(3,3), activation = 'relu'),
    tf.keras.layers.MaxPool2D(2,2),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation = 'relu'),
    tf.keras.layers.Dense(1, activation = 'sigmoid') # output neuron for number of classes
])
```

```
model.compile(loss = 'binary_crossentropy',
              optimizer = tf.keras.optimizers.RMSprop(learning_rate = 0.001),
               metrics = ['accuracy'])
```

```
model.fit(Xtr_2,np.array(tr['label']), epochs = 30)
```

```
pred = model.predict(Xte_2)
```

```
te1 = te.copy()
```

```
te1['score'] = pred
```

```
te1.to_csv('pred1.csv',index = False)
```

3.3 pred2.csv (pred1.csv rounded)

```
te2 = te.copy()
```

```
te2['score'] = np.round(pred)
```

```
te2.to_csv('pred2.csv',index = False)
```

3.4 pred3.csv (all 0s)

```
te2['score'] = 0
te2.to_csv('pred3.csv',index = False)
```

3.5 pred4.csv (all 1s)

```
te2['score'] = 1
te2.to_csv('pred4.csv',index = False)
```

3.6 pred5.csv (Included RBG channels)

```
def read_image(N,D,X):
    Xt = np.zeros([N,D,D,3])
    #found = list()
    for ii in range(N):
        #if ii % 100 == 0:
            #print('%d / %d' % (ii, N))
        teId = X['id'][ii]
        path = 'images/%05d.png' %teId

        pic = imageio.imread(path)
        pic = Image.fromarray(pic).resize((D,D)) # NOTE : this can be improved.
        # pic = np.mean(pic,axis = 2) # NOTE: this can be improved
        pic = np.array(pic)
        # print(pic.shape)
        Xt[ii,:,:,:] = pic
    Xt = Xt/255
    return Xt

Xtr_1 = read_image(tr.shape[0],200,tr)
Xtr_2 = Xtr_1.reshape(-3,200,200,3)

model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(16,(3,3), activation = 'relu', input_shape = (200,200,3)),
    tf.keras.layers.MaxPool2D(2,2),

    tf.keras.layers.Conv2D(32,(3,3), activation = 'relu'),
    tf.keras.layers.MaxPool2D(2,2),

    tf.keras.layers.Conv2D(64,(3,3), activation = 'relu'),
    tf.keras.layers.MaxPool2D(2,2),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation = 'relu'),
    tf.keras.layers.Dense(1, activation = 'sigmoid') # output neuron for number of classes
])

model.compile(loss = 'binary_crossentropy',
              optimizer = tf.keras.optimizers.RMSprop(learning_rate = 0.001),
               metrics = ['accuracy'])

model.fit(Xtr_2,np.array(tr['label']), epochs = 30)

Xte_1 = read_image(te.shape[0],200,te)
Xte_2 = Xte_1.reshape(-3,200,200,3)

pred = model.predict(Xte_2)
te1 = te.copy()
te1['score'] = pred
te1.to_csv('pred5.csv',index = False)
```

3.7 pred6.csv (More layers CNN)

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(16,(3,3), activation = 'relu', input_shape = (200,200,3)),
    tf.keras.layers.MaxPool2D(2,2),

    tf.keras.layers.Conv2D(32,(3,3), activation = 'relu'),
    tf.keras.layers.Conv2D(32,(3,3), activation = 'relu'),

    tf.keras.layers.MaxPool2D(2,2),

    tf.keras.layers.Conv2D(64,(3,3), activation = 'relu'),
    tf.keras.layers.Conv2D(64,(3,3), activation = 'relu'),

    tf.keras.layers.MaxPool2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation = 'relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(256, activation = 'relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation = 'sigmoid') # output neuron for number of classes
])

model.compile(loss = 'binary_crossentropy',
              optimizer = tf.keras.optimizers.RMSprop(learning_rate = 0.001),
               metrics = ['accuracy'])

model.fit(Xtr_2,np.array(tr['label']), epochs = 75)
pred = model.predict(Xte_2)
te1 = te.copy()
te1['score'] = pred
te1.to_csv('pred6.csv',index = False)
```

3.8 pred7.csv (CNN with Data Augmentation)

```
def read_image(N,D,X):
    Xt = np.zeros([N,D,D,3])
    #found = list()
    for ii in range(N):
        #if ii % 100 == 0:
            #print('%d / %d' % (ii, N))
        teId = X['id'][ii]
        path = 'images/%05d.png' %teId

        pic = imageio.imread(path)
        pic = Image.fromarray(pic).resize((D,D)) # NOTE : this can be improved.
        # pic = np.mean(pic,axis = 2) # NOTE: this can be improved
        pic = np.array(pic)
        # print(pic.shape)
        Xt[ii,:,:,:] = pic
    Xt = Xt/255
```

```
    return Xt

# inspired by
# https://towardsdatascience.com/fast-feature-engineering-in-python-image-data-5d3a8a7bf616

def read_augimage2(N,D,X):
    Xt = np.zeros([N,D,D,3])
    #found = list()
    for ii in range(N):
        #if ii % 100 == 0:
            #print('%d / %d' % (ii, N))
        teId = X['id'][ii]
        path = 'images/%05d.png' %teId

        pic = imageio.imread(path)
        pic = Image.fromarray(pic).resize((D,D)) # NOTE : this can be improved.
        # pic = np.mean(pic,axis = 2) # NOTE: this can be improved
        pic1 = np.array(pic)
        #pic2 = A.RandomCrop(width = 256, height = 256)(image = pic1)
        pic3 = A.GaussianBlur(p=0.8)(image=pic1)['image']
        #pic4 = A.Flip(0.8)(image=pic1)
        # print(pic.shape)
        #Xt[ii,:,:,:] = pic1
        #Xt[ii,:,:,:] = pic2
        Xt[ii,:,:,:] = pic3
        #Xt[ii+3,:,:,:] = pic4
    Xt = Xt/255
    return Xt


def read_augimage3(N,D,X):
    Xt = np.zeros([N,D,D,3])
    #found = list()
    for ii in range(N):
        #if ii % 100 == 0:
            #print('%d / %d' % (ii, N))
        teId = X['id'][ii]
        path = 'images/%05d.png' %teId

        pic = imageio.imread(path)
        pic = Image.fromarray(pic).resize((D,D)) # NOTE : this can be improved.
        # pic = np.mean(pic,axis = 2) # NOTE: this can be improved
        pic1 = np.array(pic)
        #pic2 = A.RandomCrop(width = 256, height = 256)(image = pic1)
        #pic3 = A.GaussianBlur(p=0.8)(image=pic1)
        pic4 = A.Flip(0.8)(image=pic1)['image']
        # print(pic.shape)
        #Xt[ii,:,:,:] = pic1
        #Xt[ii,:,:,:] = pic2
        #Xt[ii+2,:,:,:] = pic3
        Xt[ii,:,:,:] = pic4
    Xt = Xt/255
    return Xt
```

```
Xtr_1 = read_image(tr.shape[0],256,tr)
Xtr_1 = Xtr_1.reshape(-3,256,256,3)
Xte_1 = read_image(te.shape[0],256,te)
Xte_2 = Xte_1.reshape(-3,256,256,3)

aug_X2 = read_augimage2(tr.shape[0],256,tr)
aug_X3 = read_augimage3(tr.shape[0],256,tr)
aug_X2 = aug_X2.reshape(-3,256,256,3)
aug_X3 = aug_X3.reshape(-3,256,256,3)

allX = np.concatenate((Xtr_1,aug_X2,aug_X3),axis = 0)
y = np.concatenate((np.array(tr['label']),np.array(tr['label']),np.array(tr['label'])),axis = 0)
```

```
model1 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(16,(3,3), activation = 'relu', input_shape = (256,256,3)),
    tf.keras.layers.MaxPool2D(2,2),

    tf.keras.layers.Conv2D(32,(3,3), activation = 'relu'),
    tf.keras.layers.MaxPool2D(2,2),

    tf.keras.layers.Conv2D(64,(3,3), activation = 'relu'),
    tf.keras.layers.MaxPool2D(2,2),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation = 'relu'),
    tf.keras.layers.Dense(1, activation = 'sigmoid') # output neuron for number of classes
])


model1.compile(loss = 'binary_crossentropy',
              optimizer = tf.keras.optimizers.RMSprop(learning_rate = 0.001),
               metrics = ['accuracy','AUC'])

model1.fit(allX,y, epochs = 30)

pred = model1.predict(Xte_2)

te1 = te.copy()
te1['score'] = pred
te1.to_csv('pred7.csv',index = False)
```