CNN Model with Image Augmentation: Detecting Cataracts

This notebook goes into detail about how we imported images, implemented image augmentation, structured our Convolutional Neural Network model, and analyzed our results. It also incudes sections on Feature Maps and Filters at the end of the notebook.

The only change made to the model was to adjust the EarlyStopping callback so that the model would run for additional epochs.

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

```
Mounted at /content/drive
```

Imports

```
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd

from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooli
ng2D
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.preprocessing.image import img_to_array, load_img, Imag
eDataGenerator
from tensorflow.keras.callbacks import EarlyStopping

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classif
ication_report
```

```
# Setting a random seed for reproducibility
np.random.seed(42)
```

Reading in image data as X and y

```
# specify the path with the subfolders of cleaned eye images
clean_path = '/content/drive/MyDrive/cleaned_eye_images/'

# make a dictionary of eye conditions and integers because y needs to be a nu
mber, not a string
# setting the condition to be 1 so that 'positive' results mean the eye has t
he condition
condition_dict = {'cataracts':1, 'normal': 0}

# make empty lists for X and y
X=[]
y=[]

# iterate through each subfolder (= condition)
```

```python
for condition in os.listdir(clean_path):

    # make sure the subfolder is actually the name of a condition (e.g., not 
'DS_Store')
    if condition in condition_dict.keys():

        # allows you to specify how many images to collect from each folder
        # this allows the final number to be divisible by the desired batch s
ize
        number=0
        total_num = 468

        # iterate through each image file in the subfolder
        for file in os.listdir(clean_path+condition):

            if number < total_num:

            # added a try/except so that DS_Store files don't trip an error
                try:
                    # load the image file
                    image = load_img(clean_path+condition+'/'+file)

                    # turn the image into an array
                    image_arr = img_to_array(image)

                    # add the image array to X
                    X.append(image_arr)

                    # use the condition_dict to add the right number to y tha
t corresponds to the eye condition
                    y.append(condition_dict[condition])

                    number+=1

                except:
                    continue

# change X and y into numpy arrays
X = np.array(X)
y = np.array(y)

# checking the shape of X
# there are 768 512x512 images with 3 channels (RGB)
X.shape
```

```
(768, 512, 512, 3)
```

```python
# y matches the number of images in X
y.shape
```

```
(768,)
```

Baseline Model

Our dataset is unevenly distributed between normal eyes and eyes with cataracts, and the model needs to make correct predictions more than 60.9% of the time to beat the baseline.

```
print(f'There are {len(y)-y.sum()} non-diseased eye images and {y.sum()} imag
es of eyes with cataracts in the dataset.')

# Since y is binary with values of 0 and 1, the baseline accuracy can be foun
d by summing y and dividing by the length of y
# The baseline accuracy is the higher of this value and 1 - this value
# rounding the numbers to 3 places to make the formatting nicer
print(f'This gives a baseline accuracy of {round(np.array([(y.sum() / len(y))
,(1-(y.sum()/len(y)))]).max(), 3)}')
```

```
There are 300 non-diseased eye images and 468 images of eyes with cataracts i
n the dataset.
This gives a baseline accuracy of 0.609
```

Train-Test Split

```
# regular train-test-split
# no need for StandardScaler because the images have already been normalized

# changed test_size to 128 so test and train sizes are both divisible by 64,
the batch_size
# to use with image augmentation
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=128, rand
om_state=42, stratify=y)
```

```
X_train.shape
```

```
(640, 512, 512, 3)
```

Image Augmentation

```
# creating the same ImageDataGenerators as the ones made in the Glaucoma mode
l

image_generator_train = ImageDataGenerator(rescale=1.0/255.0, horizontal_flip
=True, rotation_range=30, brightness_range=(.8,1.2))
image_generator_test = ImageDataGenerator(rescale=1.0/255.0)
image_generator_train.fit(X_train)
```

Building the CNN Model

```
# Instantiate a Sequential model (that will process each layer sequentially)
model = Sequential()
```

```
# add a Convolutional 2D layer that will create 16 3x3 filters to detect imag
e features
```

```
model.add(Conv2D(16, (3,3), activation='relu', input_shape=(512,512,3)))

# add a MaxPooling 2D layer that will take the maximum value in every 2x2 gri
d (with a stride defaulting to the pool_size)
# this effectively cuts the dimensions of the data in half, and helps get rid
of noise caused by small variations in the image

model.add(MaxPooling2D(pool_size=(2,2)))

# add more convolutional layers (with max pooling between each one)

# increasing filters to 32
# input shape is only needed for the first layer above
model.add(Conv2D(32, (3,3), activation='relu'))
model.add(MaxPooling2D((2,2)))

model.add(Conv2D(32, (3,3), activation='relu'))
model.add(MaxPooling2D((2,2)))

# increasing filters to 64
model.add(Conv2D(64, (3,3), activation='relu'))
model.add(MaxPooling2D((2,2)))

# add a flatten layer to bridge between the convolutional layers and the dens
e layers
model.add(Flatten())

# the dense layers analyze the features that were identified in the convoluti
onal layers
model.add(Dense(256, activation='relu'))
model.add(Dense(256, activation='relu'))

# add the output layer with sigmoid activatin since it's a binary classificat
ion
model.add(Dense(1, activation='sigmoid'))

# compile the model using binary_crossentropy, accuracy metrics, and the adam
optimizer
model.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer='ad
am')
```

Fitting the Model

The model is the same as the model used for glaucoma and diabetic retinopathy, with the minor adjustment of altering the min_delta of the EarlyStopping callback so that the model would run for more epochs. This was done because the other two models appeared to stop too early -- the models were not improving much, but they also weren't overfit and it was unclear if the model could still improve with more training.

```
# add early stopping

# changing the min_delta to .001 instead of .01 to see what happens when the
model doesn't stop so early
# this is the only change between this model and the Glaucoma and Diabetic Re
tinopathy models (aside from the inputs)

early_stop = EarlyStopping(monitor='val_loss', min_delta=.001, patience=5, ve
rbose=1, mode='auto')

# fit the model and save it as h so the accuracy and loss scores for each epo
ch can be visualized

# use a batch size be divisible by the number of images in X_train
h = model.fit(image_generator_train.flow(X_train, y_train, batch_size=32, see
d=42), validation_data=(image_generator_test.flow(X_test, y_test, batch_size=
32, seed=42)), steps_per_epoch=len(X_train)/32, epochs=30, callbacks=[early_s
top])

Epoch 1/30
20/20 [==============================] - 55s 2s/step - loss: 0.5178 - accurac
y: 0.7125 - val_loss: 0.1981 - val_accuracy: 0.9375
Epoch 2/30
20/20 [==============================] - 43s 2s/step - loss: 0.3128 - accurac
y: 0.8813 - val_loss: 0.1548 - val_accuracy: 0.9531
Epoch 3/30
20/20 [==============================] - 42s 2s/step - loss: 0.2728 - accurac
y: 0.8875 - val_loss: 0.1726 - val_accuracy: 0.9531
Epoch 4/30
20/20 [==============================] - 42s 2s/step - loss: 0.2466 - accurac
y: 0.9031 - val_loss: 0.1439 - val_accuracy: 0.9531
Epoch 5/30
20/20 [==============================] - 45s 2s/step - loss: 0.2205 - accurac
y: 0.9187 - val_loss: 0.1481 - val_accuracy: 0.9531
Epoch 6/30
20/20 [==============================] - 42s 2s/step - loss: 0.2309 - accurac
y: 0.9031 - val_loss: 0.1484 - val_accuracy: 0.9453
Epoch 7/30
20/20 [==============================] - 43s 2s/step - loss: 0.2112 - accurac
y: 0.9234 - val_loss: 0.1403 - val_accuracy: 0.9375
Epoch 8/30
20/20 [==============================] - 42s 2s/step - loss: 0.1701 - accurac
y: 0.9359 - val_loss: 0.1259 - val_accuracy: 0.9531
Epoch 9/30
20/20 [==============================] - 42s 2s/step - loss: 0.1716 - accurac
y: 0.9312 - val_loss: 0.1601 - val_accuracy: 0.9531
Epoch 10/30
20/20 [==============================] - 43s 2s/step - loss: 0.1729 - accurac
y: 0.9266 - val_loss: 0.1243 - val_accuracy: 0.9531
Epoch 11/30
```

```
20/20 [==============================] - 43s 2s/step - loss: 0.1881 - accurac
y: 0.9328 - val_loss: 0.1549 - val_accuracy: 0.9453
Epoch 12/30
20/20 [==============================] - 42s 2s/step - loss: 0.1641 - accurac
y: 0.9344 - val_loss: 0.1467 - val_accuracy: 0.9688
Epoch 13/30
20/20 [==============================] - 42s 2s/step - loss: 0.1491 - accurac
y: 0.9484 - val_loss: 0.1144 - val_accuracy: 0.9609
Epoch 14/30
20/20 [==============================] - 42s 2s/step - loss: 0.1265 - accurac
y: 0.9609 - val_loss: 0.0825 - val_accuracy: 0.9609
Epoch 15/30
20/20 [==============================] - 43s 2s/step - loss: 0.1383 - accurac
y: 0.9484 - val_loss: 0.0818 - val_accuracy: 0.9766
Epoch 16/30
20/20 [==============================] - 42s 2s/step - loss: 0.1201 - accurac
y: 0.9531 - val_loss: 0.0999 - val_accuracy: 0.9688
Epoch 17/30
20/20 [==============================] - 43s 2s/step - loss: 0.1368 - accurac
y: 0.9484 - val_loss: 0.1030 - val_accuracy: 0.9531
Epoch 18/30
20/20 [==============================] - 42s 2s/step - loss: 0.1576 - accurac
y: 0.9312 - val_loss: 0.0983 - val_accuracy: 0.9844
Epoch 19/30
20/20 [==============================] - 42s 2s/step - loss: 0.1324 - accurac
y: 0.9453 - val_loss: 0.1325 - val_accuracy: 0.9375
Epoch 19: early stopping
```

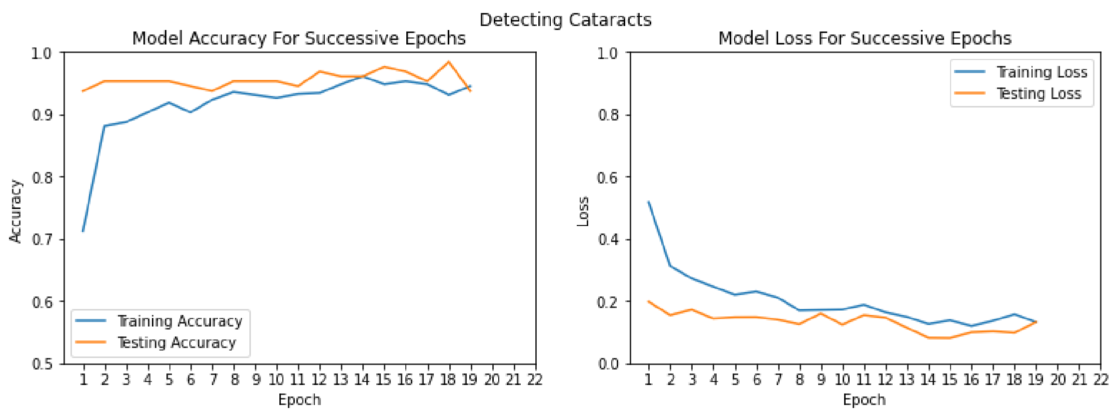
Saving the Model

```
# code to save the model as an h5 file so that it can be used in Flask

# commenting out the code so it doesn't run again by accident
model.save("/content/drive/MyDrive/FINAL MODELS/CATARACT/cataract_model.h5")
```

Visualizing the accuracy and loss scores for each epoch

```
fig, axs = plt.subplots(1, 2, figsize=(13,4))

fig.suptitle('Detecting Cataracts')

# plot training and testing accuracy
axs[0].plot(h.history['accuracy'], label='Training Accuracy')
axs[0].plot(h.history['val_accuracy'], label='Testing Accuracy')

# add titles, labels, and tick formatting
axs[0].set_title('Model Accuracy For Successive Epochs')
axs[0].set_xlabel('Epoch')
axs[0].set_xticks(ticks=range(0,22))
axs[0].set_xticklabels(labels=range(1,23))
axs[0].set_ylabel('Accuracy')
axs[0].set_ylim(.5,1)
```

```
# add a legend
axs[0].legend()

# plot training and testing loss
axs[1].plot(h.history['loss'], label='Training Loss')
axs[1].plot(h.history['val_loss'], label='Testing Loss')

# add titles, labels, and tick formatting
axs[1].set_title('Model Loss For Successive Epochs')
axs[1].set_xlabel('Epoch')
axs[1].set_xticks(ticks=range(0,22))
axs[1].set_xticklabels(labels=range(1,23))
axs[1].set_ylabel('Loss')
axs[1].set_ylim(0,1)

# add a legend
axs[1].legend();
```



Making a Confusion Matrix

```
# generate predictions from X_test
# in order to generate predictions correctly, the X_test images need to be fe
d to model.predict through
# image_generator_test with shuffle set to False
preds = model.predict(image_generator_test.flow(X_test, shuffle=False))
```

```
4/4 [==============================] - 1s 223ms/step
```

```
# change predictions from probabilities to 0s and 1s
preds = [int(np.round(x)) for x in preds]

print(preds)
```

```
[0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0
, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0,
1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1,
0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0,
0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0]
```

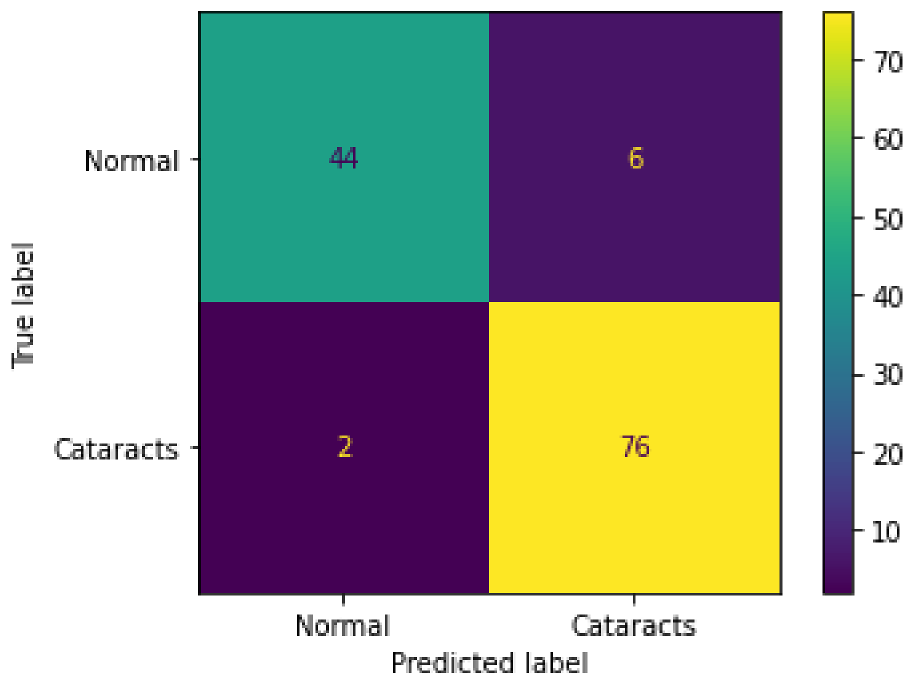
```
y_test
```

```
array([0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0,
       1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0,
       1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0,
       0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0,
       0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0])
```

```
y_test.sum()
```

```
78
```

```
# make and display a confusion matrix
conf_matrix = confusion_matrix(y_test, preds)
ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=['Normal'
, 'Cataracts']).plot();
```




```
# display the classification report
print(classification_report(y_test, preds))
```

```
              precision    recall  f1-score   support

           0       0.96      0.88      0.92        50
           1       0.93      0.97      0.95        78

    accuracy                           0.94       128
   macro avg       0.94      0.93      0.93       128
weighted avg       0.94      0.94      0.94       128
```