Visual Detection of Melanoma Cancer with Artificial Intelligence

By: Andrew Morgan and Harry McKigney
Los Alamos Middle School

For our Supercomputing challenge project, we are going to use an artificial intelligence (AI) python library called keras that uses the google tensorFlow AI code to detect melanoma by analysing images. If you wish to use this AI then go to https://keras.io or tensorflow.org. Now, the way we made the AI code is by extending the keras code made to detect cats and dogs on the keras python page. We modified this code to work for our case.

The first term we need to go over is a mole. The definition of a mole is a dark spot on your skin [3]. Cancer is a group of cells that mutated making them uncontrollably multiply. Melanoma is a cancerous mole defined by the ABC's of melanoma.

The ABC's of melanoma are multiple ways to determine if a mole is or isn't threatening. A is stands for asymmetrical. A non-asymmetrical mole is a sign of melanoma. B stands for border irregularity. A mole has a defined border unlike cases of melanoma. Next is C and it stands for color where a brownish color is normal for a mole and melanoma having wild, random colors. Fourth is D for diameter. With a mole, it will likely be small. Melanoma/cancer grows uncontrollably making it large. Finally with E representing evolving. A mole will not change or evolve much unlike melanoma.

We used a set of images from a medical research group [5] consisting of 44 images of moles and 44 images of melanoma. We will use 36 images of each type for training the AI and 6 images of each type for testing the predictions from the AI.

Now we need to go into the convolutional part of the AI [1] starting with how it processes the image. The AI code will use convolution kernels that are a grid of weights, for instance 3x3, that combines them into a single value using the Relu function [4] and repeats this over the entire image. The AI trains these weights in the kernels to detect features in an image. A pooling step [2] takes the largest value over a small grid, for instance 2x2, to make the image have a smaller resolution. The output from the convolutional network is fed into a fully connected neural network described in the following paragraph.

The fully connected neural network has layers. A layer is a set of nodes. Each node has an activation function, Relu for this work, that takes inputs from other nodes multiplied by weights. The weights are numbers trained by the AI. The output from the neural network uses a sigmoid activation function that returns a value between 0 and 1. For example, a value of zero means the image is of a mole and a value of 1 means it is an image of melanoma.

Epoch is the number of training iterations (the number of times the AI is trained). The batch size is the number of images used to train the AI within an iteration. To find good weights, the AI has to go through multiple training interactions using known images that are the training set. One way you can think about it is that it takes "steps" in a certain direction to find better weights to match the images. It is basically trying to find the best weights by moving along a slope that reduces the error in the output from the entire neural network.

Our results were good and the AI got better and better with training iterations. Fig. 1 shows the loss, which is the mean-squared error, from the AI with each training iteration. Fig. 2 shows the accuracy of the AI with each training iteration. In the first image, the lower the line, the better. In the second image, the higher the line, the better. There were times the results from the AI jumped up or down but the errors continued on a downward line toward zero error. The AI did very well at predicting melanoma. The AI had a 92% accuracy after 100 iterations. One of the ways we could make the AI better is by adding more images to train the AI. The original, unmodified AI code for detecting dogs and cats contained a huge data set of about a terabyte in size. That AI performed really well due to a wider range of images for each category.

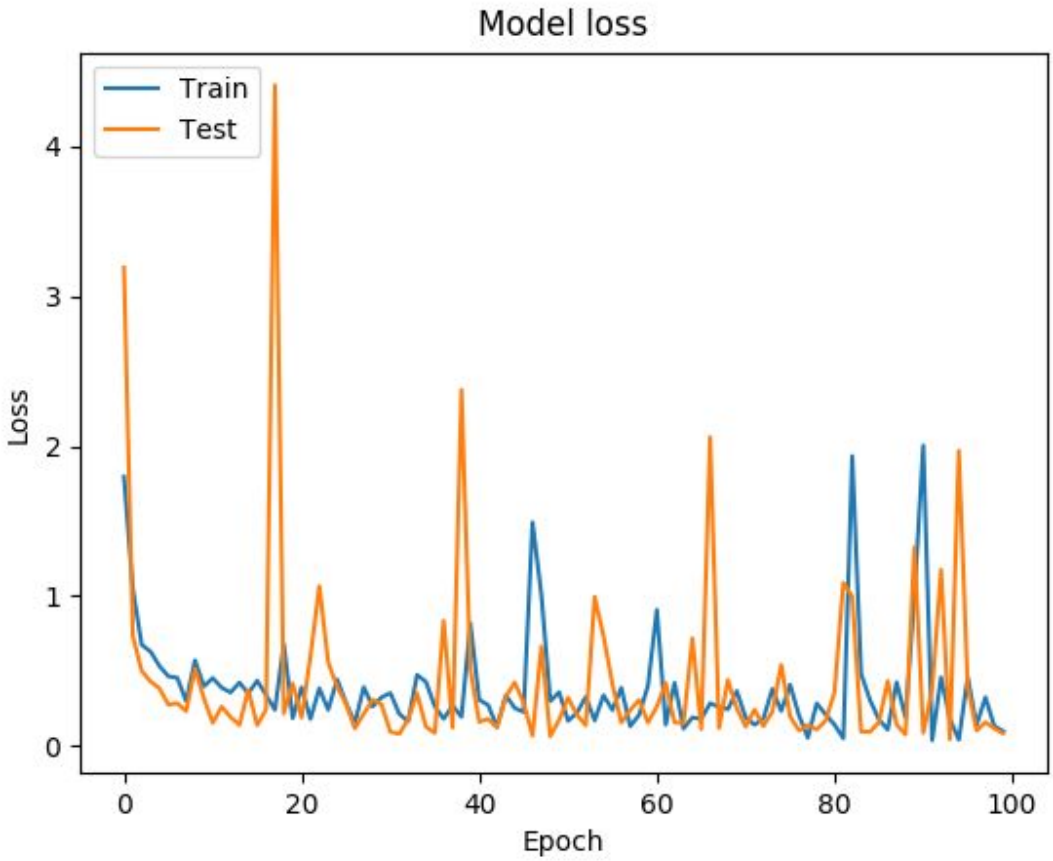

Fig. 1. The loss value is the mean squared error from the AI. This error decreases with iterations. A value of 0 is the goal. The training set is shown in blue. The AI was also used on a different set of images called the test suite that were not part of the training set. The test set is shown in orange.

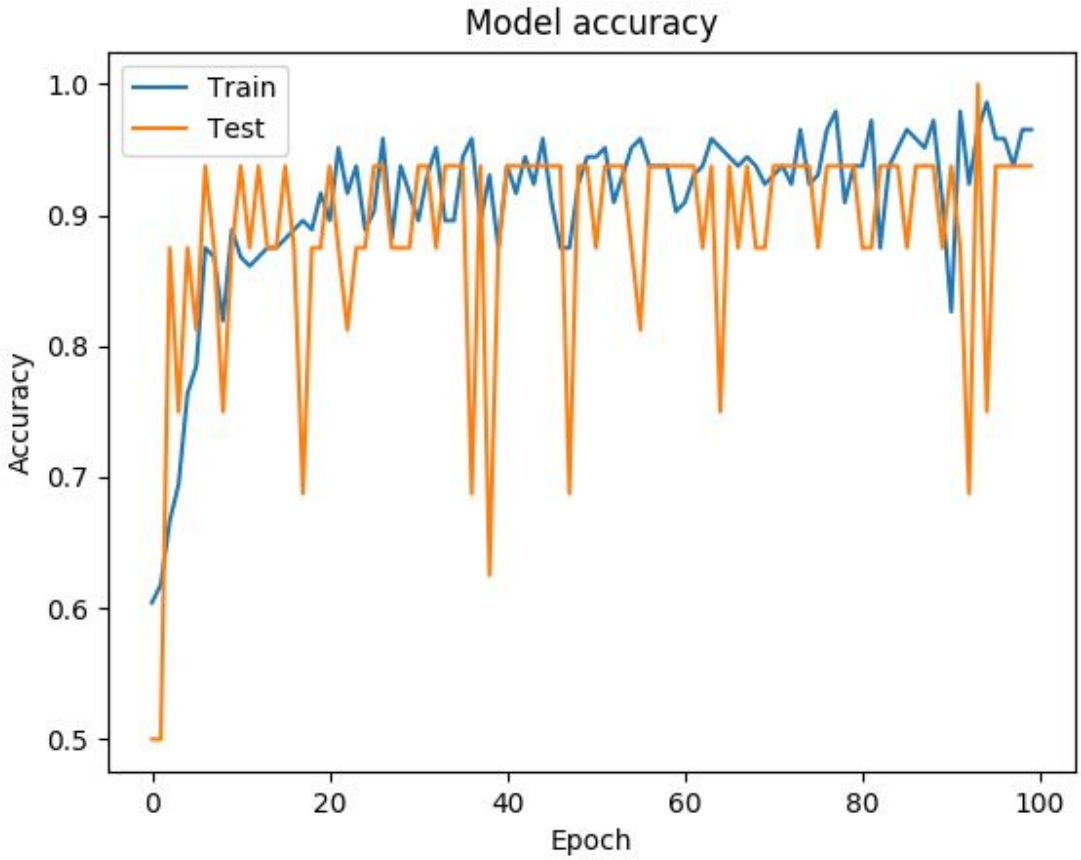

Fig. 2. The accuracy of the AI is shown. A value of 1 is the goal. The training set is shown in blue. The AI was also used on a different set of images called the test suite that were not part of the training set. The test set is shown in orange.

References:

1. Ferdinand, Nushaine. “A Simple Guide to Convolutional Neural Networks.” *Medium*, Towards Data Science, 19 Jan. 2020, towardsdatascience.com/a-simple-guide-to-convolutional-neural-networks-751789e7bd88.
2. Brownlee, Jason. “A Gentle Introduction to Pooling Layers for Convolutional Neural Networks.” *Machine Learning Mastery*, 5 July 2019, machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/.
3. Young, Alison Z. “The Importance of Early Detection of Melanoma.” *Spectrum Dermatology*, 15 May 2017, www.spectrumdermatologyseattle.com/importance-early-detection-melanoma/.
4. “ReLU - Machine Learning for Humans.” *TinyMind*, www.tinymind.com/learn/terms/relu.

5. Mendonça, Teresa & Ferreira, Pedro & Marques, Jorge & Marçal, André & Rozeira, Jorge. (2013). PH2 - A dermoscopic image database for research and benchmarking. Conference proceedings : ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference. 2013. 5437-5440. 10.1109/EMBC.2013.6610779.

Code:

```
# https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html


"""
  To run the code type:
   python3 *py


data/
   train/
      moles/
         mole001.jpg
         mole002.jpg
         ...
      cancer/
         cancer001.jpg
         cancer002.jpg
         ...


   validation/
      moles/
         mole001.jpg
         mole002.jpg
         ...
      cancers/
         cancer001.jpg
         cancer002.jpg
         ...

   """

import os
import sys
```

```
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras import backend as K
import matplotlib.pyplot as plt

# dimensions of our images.
img_width, img_height = 255, 255

# image directory
train_data_dir = 'data/train'
validation_data_dir = 'data/validation'

''' Conv NN Parameters '''
nb_train_samples = 36     # number of training images to use in each folder
nb_validation_samples = 8 # number of validation images to use in each folder
batch_size = 4            # batch size of training images (2 = 2 images at a time)
nclasses = 1              # number of image folders in director (=1 for binary fitting of 2 folders)

''' Input Parameters '''
epochs = 5              # number of training iterations


if K.image_data_format() == 'channels_first':
    input_shape = (3, img_width, img_height)
else:
    input_shape = (img_width, img_height, 3)

# The Convolution Neural Net
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

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

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5)) # randomly ignore 50% of the neurons when training, it prevents over fitting
model.add(Dense(nclasses))
model.add(Activation('sigmoid'))

# use categorical in place of binary for more than 2 folders (> 2 cases)
model.compile(
    loss='binary_crossentropy',
    optimizer='rmsprop',
    metrics=['accuracy'])

# this is the augmentation configuration we will use for training
train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

# only rescaling
test_datagen = ImageDataGenerator(rescale=1. / 255)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size = (img_width, img_height),
    batch_size = batch_size,
    class_mode = 'binary')

validation_generator = test_datagen.flow_from_directory(
    validation_data_dir,
    target_size = (img_width, img_height),
    batch_size = batch_size,
    class_mode ='binary')

history = model.fit_generator(
    train_generator,
    steps_per_epoch = nb_train_samples,
    epochs = epochs,
    validation_data = validation_generator,
    validation_steps = nb_validation_samples)
```

```
model.save_weights('first_try.h5')


# Plot training & validation accuracy values
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```