# Building a Deep Learning Model to Classify Images of Humans and Robots Final Report

Ben Vuong

April 2022

**Abstract**

In this semester long project, I used deep learning in-order to create a model that can classify if an image is of a robot or human. Through out the many phases of this project I would collect the images need to build my data set and prepare it. Then I would then build a model that would over fit the data. After that I would use many different techniques in order to reduce said over fitting.

## 1 Introduction

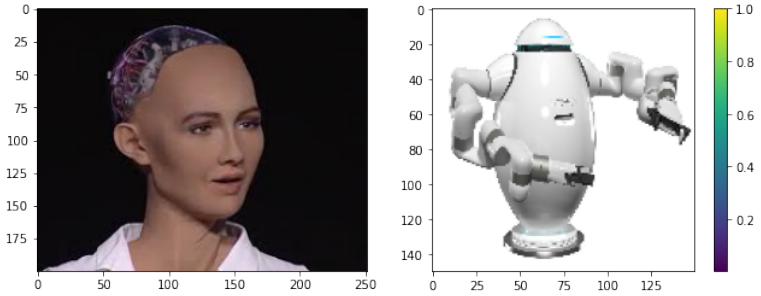
When thinking about what to make my project about I thought about how it would be interesting if a model can tell the difference between an image of a robot or a human. I thought that this could be a foundation that can be used for other applications. For example what if in the future we had life like androids walking around. A model that can tell the difference between a life like android and a human can be installed into security cameras or even in self-driving cars where the cars will be program to prioritize the life of humans crossing the road. For building a deep learning model, I will be using convolution neural networks so that the models can study the input images and learn patterns from them.

## 2 Phases

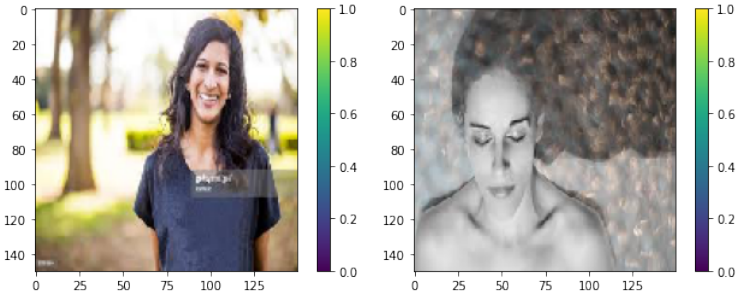
### 2.1 Phase 1

In the first phase, I started to collect images in order to build my data set. Using google images and a image scraper, I was about to obtain 1046 images, where 523 of those images were of humans and the other 523 images were of robots. The images of humans that were obtains were images of a single person often in a self portrait. There was a big diversity of types of peoples in the images in order to reduce any bias. For the images of robots many of the images was of robots that were humanoid where it had a human like shape with arms and a head. While other images of the robots, had robots with human like silicon

skin. Images likes these were included so that the model can work harder and find the nuances between a robot that could be mistaken as a human and a real human. After that when I would pass these images through the data generator in order to process the images into something that the model can read.



— Figure 1: Example images from the robot data set



— Figure 2: Example images from the human data set

## 2.2 Phase 2

During this phase I started to experimenting on making many different models to see if I can make a model that is the smallest that can still over fit the data. In these different models I used convolutional neural networks, and experimented on changing the number of layers, the sizes of the filters, and the arrangement of different layers like max-pooling and flatten layers. In the end I was able to find a very good model that gave me the highest accuracy and was relatively small. In this model I kept to only using 4 conv2d layers and 3 max pooling layers. I set the number of layers to 32, 16, 16, and 8 in decreasing order and the size of the kernels increased by 1 with every layer going down. The resulting accuracy was 98.65%. This model is the model that I would be using in order to build upon in later phases.

```
model5 = Sequential()
model5.add( Conv2D( 32, (2, 2 ), activation = 'relu', input_shape = (150, 150, 3) ) )
model5.add( MaxPooling2D(4 ,4) )
model5.add( Conv2D( 16, ( 3, 3 ), activation = 'relu' ) )
model5.add( MaxPooling2D(4, 4) )
model5.add( Conv2D( 16, ( 4, 4 ), activation = 'relu',padding="same"   ) )
model5.add( MaxPooling2D(4, 4) )
model5.add( Conv2D( 8, ( 5, 5 ), activation = 'relu',padding="same"  ) )
model5.add( Flatten() )
model5.add( Dense( 10, activation = 'relu' ) )
model5.add( Dense( 1, activation = 'sigmoid' ) )
```

Figure 3: Final model chosen

## 2.3 Phase 3

For this phase, I looked into callback techniques in order to reduce the over fitting that was happening in the model that I made in phase 2. The callback techniques that I looked into were early stopping and model check pointing. These callbacks are used in order to prevent over training. If a model is trained for too long, it may start to stop focusing and learning the general pattern of the data set and focus more on the specific patterns of the input data set which will lead to the model over fitting. Before I used any callbacks, I split my data set into 3 sections, the training set, validation set, and the test set. Early stopping callback would stop the model training if the loss stops decreasing for a certain amount of epochs. Through out a normal training process of the model, the loss would decrease meaning that the amount of errors the model was making has decrease and has been learning. Once the loss stops going down that would mean that the model has learned enough and can't learn anymore. Model checkpoint callback on the other hand would save a copy of the weights of the models as the training process goes on. If the loss decreases from the previous epoch, the current weights would be saved. Later on when evaluating the model, the weights will be loaded back into the model, creating the best performing model during the training process. When adding these callbacks to my model I saw that accuracy did increase.

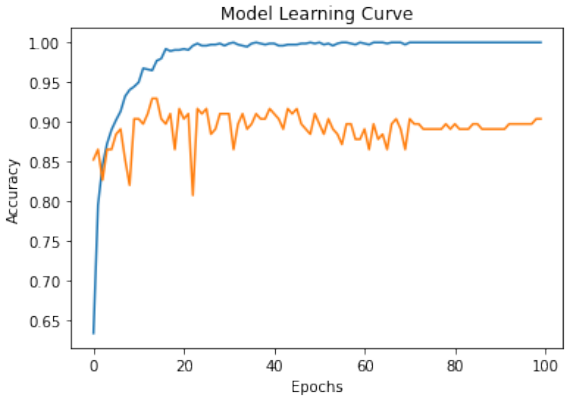
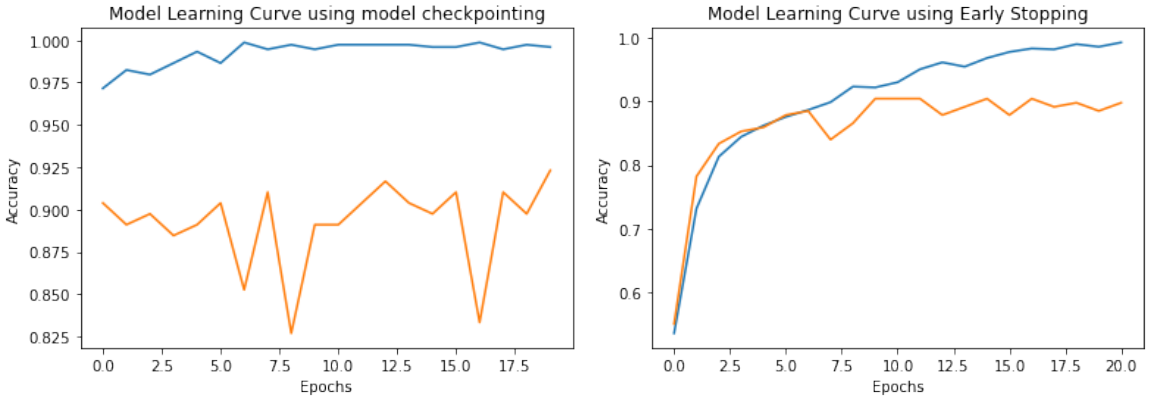


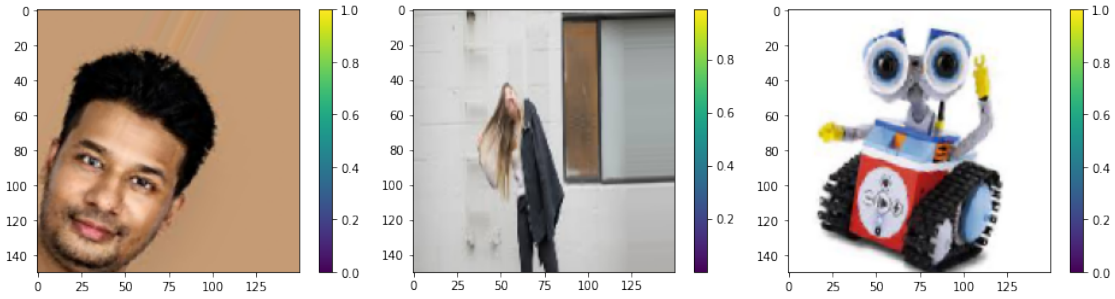

Figure 4: Model learning curve when validation was added without callbacks




— Figure 5: Model learning curve when added callbacks

## 2.4 Phase 4

In this phase I experimented with image augmentation in order to improve the accuracy of the model when evaluated on the validation set. Image augmentation in theory would create more data for the model to work with. This is done by augmenting the images of the training set in many ways like, random rotation, randomly shift the height or width, zooming in or out, and even horizontally flipping the image. Through out experimenting on different sets of image augmentation, I found that there is a sweet spot for augmentation, where you can't augment the image too much nor too little. Though in the end I found that even finding the sweet spot of image augmentation, it did not exactly improve the accuracy of my model all that much.



— Figure 6: Example images of the different image augmentations

## 2.5 Phase 5

During this fifth phase, I experimented with regularization, in order to improve the accuracy of the model on the validation set. The various regularization techniques that I experimented with was dropout, batch-normalization, and L2 regularization. These types of techniques should increase the accuracy of the model and reduce the over fitting that would happen. After many experimentation on of these regularization techniques, I found that batch-normalization improve the performance of my model the most.

| Model | Evaluation Accuracy |
|---|---|
| Control Model | 83.33% |
| Batch Normalization | 87.50% |
| Dropout | 86.81% |
| L2 Regularization | 84.72% |

— Figure 7: Accuracy of different regularization

## 2.6 Phase 6

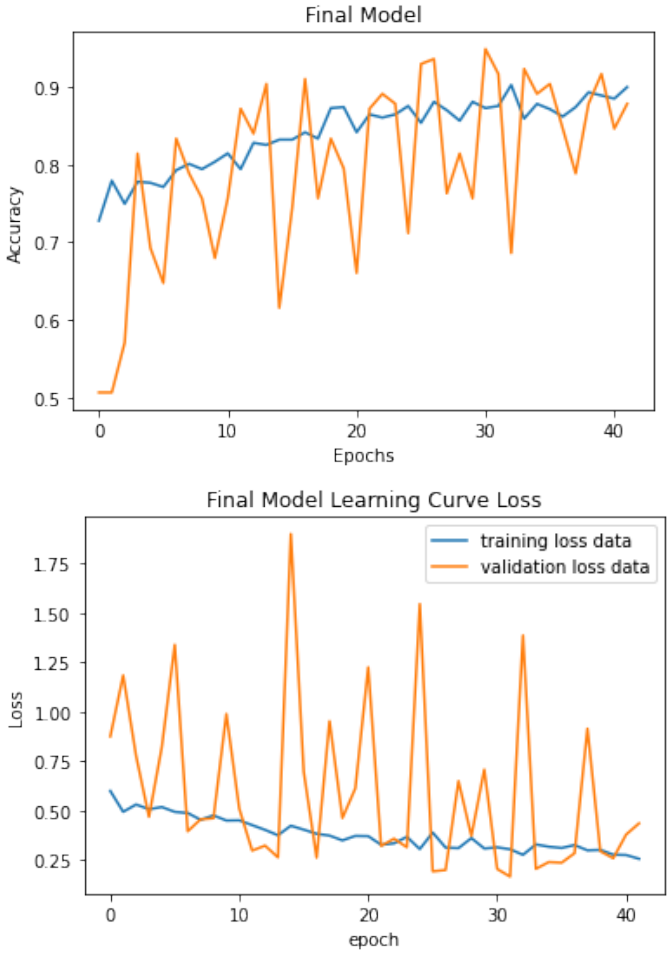
In this phase, I tried to use different models and architectures in order to improve the accuracy of my model on the validation set. There were two ways I went with this. One way would be to use a more power architecture like DenseNet, and the other way would be to use a per trained model like VGG16 or Resnet50. In the end I found that the the VGG16 pre-trained model increased my accuracy while Resnet50 gave didn't really improve the overall accuracy compared to my previous models that I made. DenseNet on the other hand actually got a worse accuracy than my other models.

| Model | Evaluation Accuracy | Training Time (in seconds) |
|---|---|---|
| ResNet50 | 83.33% | 2491.376 |
| VGG16 | 93.06% | 2499.898 |
| DenseNet | 49.31% | 8893.569 |

— Figure 6: Accuracy of different architecture and pre-trained models

# 3 Putting it all together

After experimenting with all of these parts that makes a deep learning model, it is time to take all of the parts that improved the accuracy and put it together and see what final accuracy we will get. This final model gave me an accuracy of 91.66%. This final resulting accuracy is actually higher than any other models that I made in previous phases, and is almost as high than the resulting accuracy that was produced by the VGG16 pre trained model.




— Figure 7: Learning Curve of the final model