The initialization of our model was done using the Sequential function. We then added the first layer of our model by using model.add and adding a convolutional layer with 8 filters in the convolution and a (3,3) filter matrix size. This creates a convolutional network that processes 2D images. In the specific line of code is adding the first layer,model.add(layers.Conv2D(X, (n, n), activation="relu", padding="same", input_shape=(img_height, img_width, 1))), Conv2D creates the network, X specifies the number of filters in the convolution, and activation="relu" is the activation function used. Rectified Linear Activation (ReLU) is a common activation function in neural networks that helps the network learn non-linear decision boundaries. The input_shape=(img_height, img_width, 1) portion of the code specifies the shape qualities of the input images, such as height, width, and the number of color channels.

We then added a MaxPooling2D layer, which reduces the number of parameters and computation in the network and helps with not overfitting while also retaining important features of each image. Then a second convolutional layer is added that does the same as the first layer but fewer filters are used to reduce complexity and computation. A second pooling layer is then added to reduce the dimensionality of the data even more. A flattening layer is then added to change the 2D matrix data to a 1D vector to maintain fully connected layers after the convolutional and pooling layers are added. Next, we added a fully connected layer with X neurons, where each neuron in a dense layer gets inputs from other neurons in the previous layer to make sure all the layers and their neurons are fully connected. To improve training speed and stability, we want to add a transformation that maintains the mean output to around 0 and the standard deviation of the output to around 1. Thus we added batch normalization that normalizes the activations from each previous layer.

The last layer we add is a dropout layer, where in value of 0.X, we set X to a random value and set a random fraction of inputs to 0 at each update during training, and this helps with avoiding overfitting. Lastly, we compile the total model and get it ready for training. In our case, we used the Adam optimizer with a learning rate of 0.001. We test the losses in our model using binary_crossentropy which tests logarithmic loss by finding wrong data class labels and disputing them if there are any deviations in probability found. In this particular model, we did not use any defined metrics.