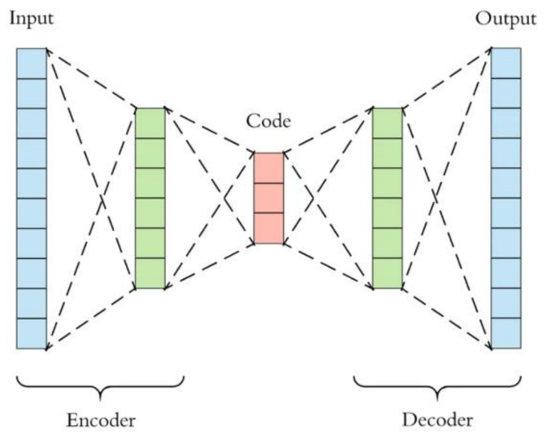
Valerie Melland
Final Exam
5-8-23

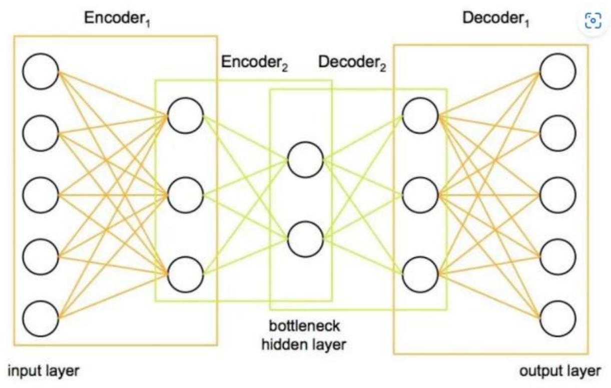
Question 1

-1- An autoencoder is a neural network that can be used to compress and reconstruct data. The encoder compresses input, and the decoder attempts to recreate the information from the compressed data. The network then is trained on the output data. The loss function of the autoencoder is typically either binary cross-entropy or mean squared error. Binary cross-entropy is appropriate when the input values of the data are in a 0-1 range. The loss function compares the output values with the original input and not the corrupted input.



The autoencoder is trained using a method called backpropagation. The outputs are slightly altered when using this algorithm. The algorithm computes the gradient of a loss function with respect to the weights of the network for a single input-output example one layer at a time while iterating backwards to avoid redundant calculations of intermediate terms in the chain rule.

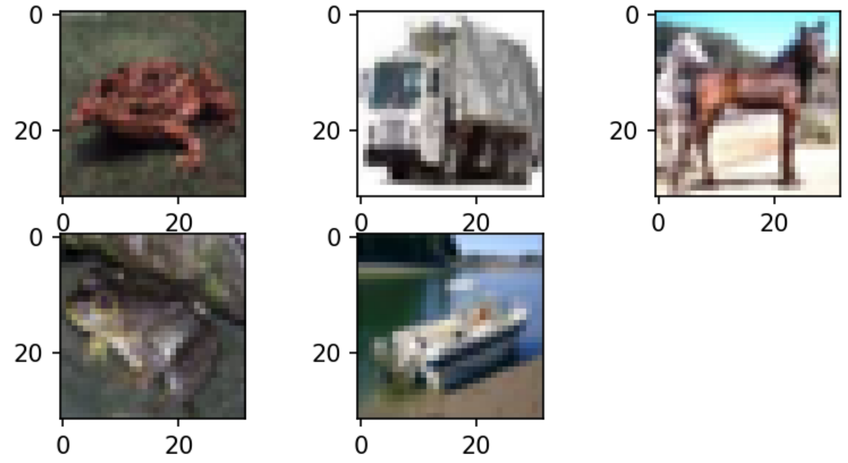
-2- A Stacked Autoencoder (SAE) are many autoencoders put together with multiple layers of encoding and decoding. This allows the algorithm to be more robust. The stacked autoencoder is trained layer by layer using the previously outputted data.



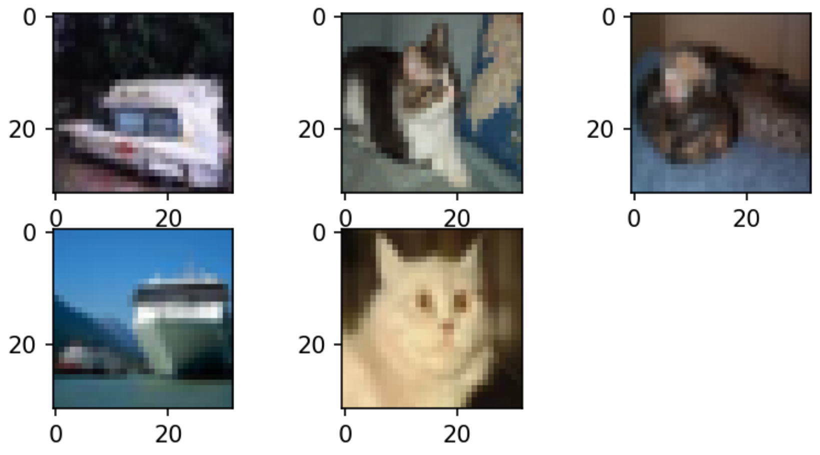
Question 2

CIFAR-10 dataset is used in this section.

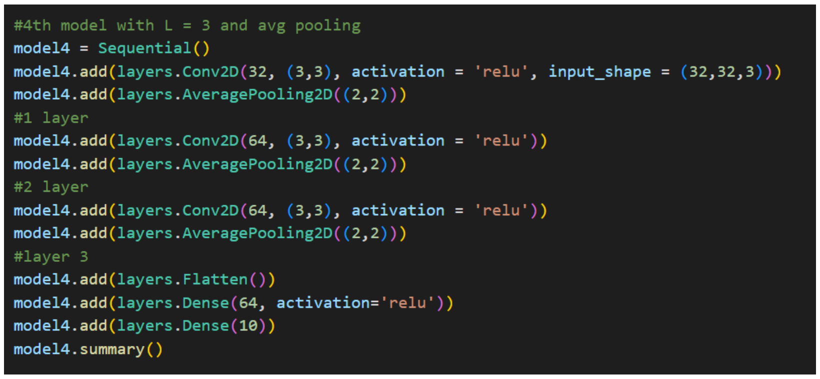
-a- Plot 5 random images inside the training set and 5 random images inside the testing set.



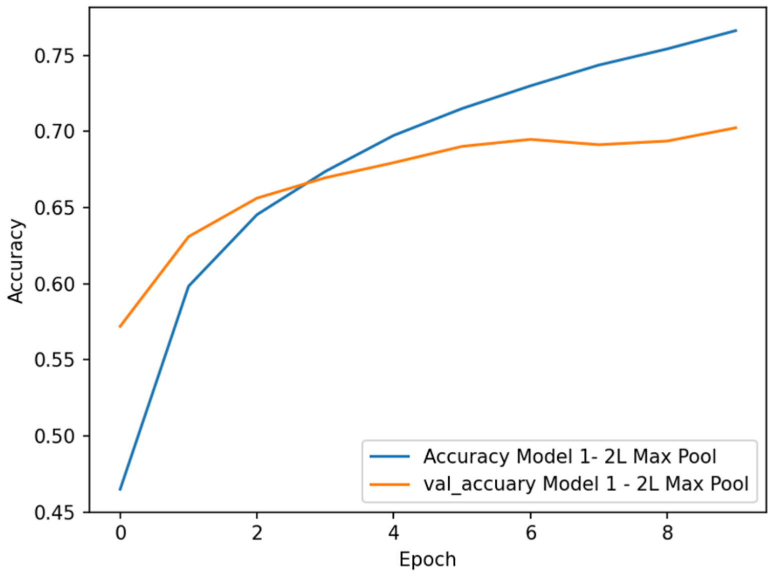
Here are five random images from the training set, and below are five random images from the testing set.

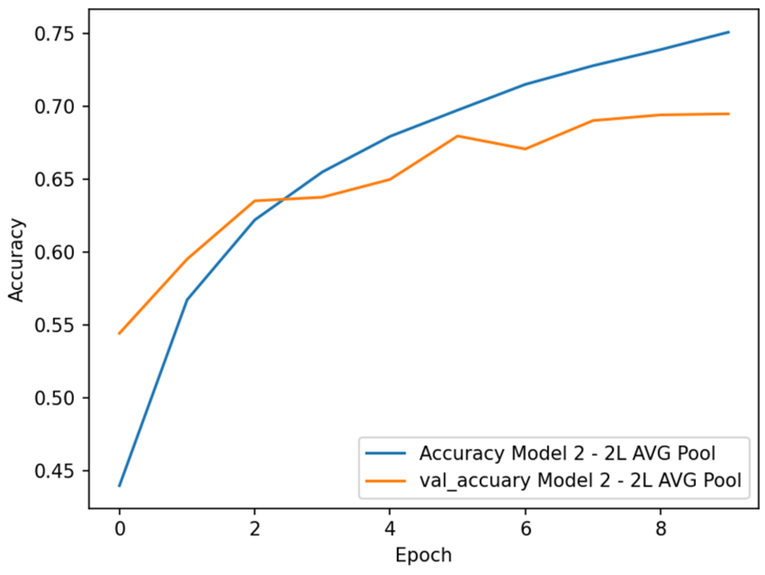


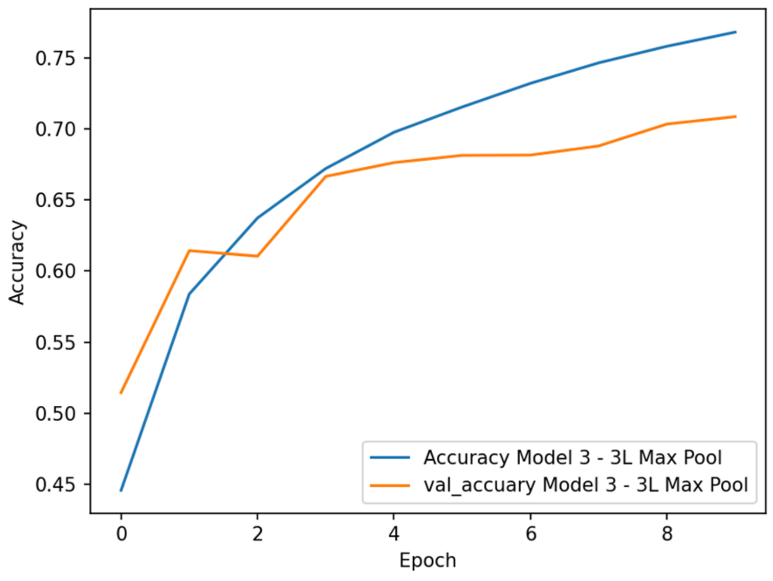
-b- Train CNN in Keras with L number of layers. I trained four different models. Model 1 has two layers with Max Pooling, Model 2 has two layers with Average Pooling, Model 3 has three layers with Max Pooling, and Model 4 has three layers with Average Pooling. For some reason, I couldn't get the 5th and 6th models to work correctly. When I tried to add the fourth layer, the model would not compile and had errors that I couldn't figure out how to resolve. Below I included a code snippet of where I created one of these models, and below that is the graphs for all the models of the Accuracy and val_accuracy for specific Epoch values. I used 10 Epochs for each of the models.

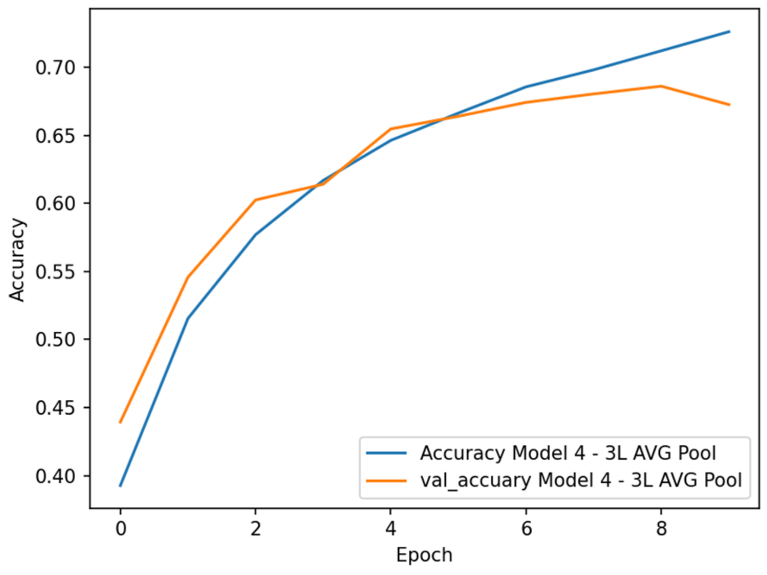


```
#4th model with L = 3 and avg pooling
model4 = Sequential()
model4.add(layers.Conv2D(32, (3,3), activation = 'relu', input_shape = (32,32,3)))
model4.add(layers.AveragePooling2D((2,2)))
#1 layer
model4.add(layers.Conv2D(64, (3,3), activation = 'relu'))
model4.add(layers.AveragePooling2D((2,2)))
#2 layer
model4.add(layers.Conv2D(64, (3,3), activation = 'relu'))
model4.add(layers.AveragePooling2D((2,2)))
#layer 3
model4.add(layers.Flatten())
model4.add(layers.Dense(64, activation='relu'))
model4.add(layers.Dense(10))
model4.summary()
```









Here is also the outputs of the loss and accuracy for each of the models:

Test 1 loss = 0.9280707240104675 Test 1 accuracy = 0.7024000287055969

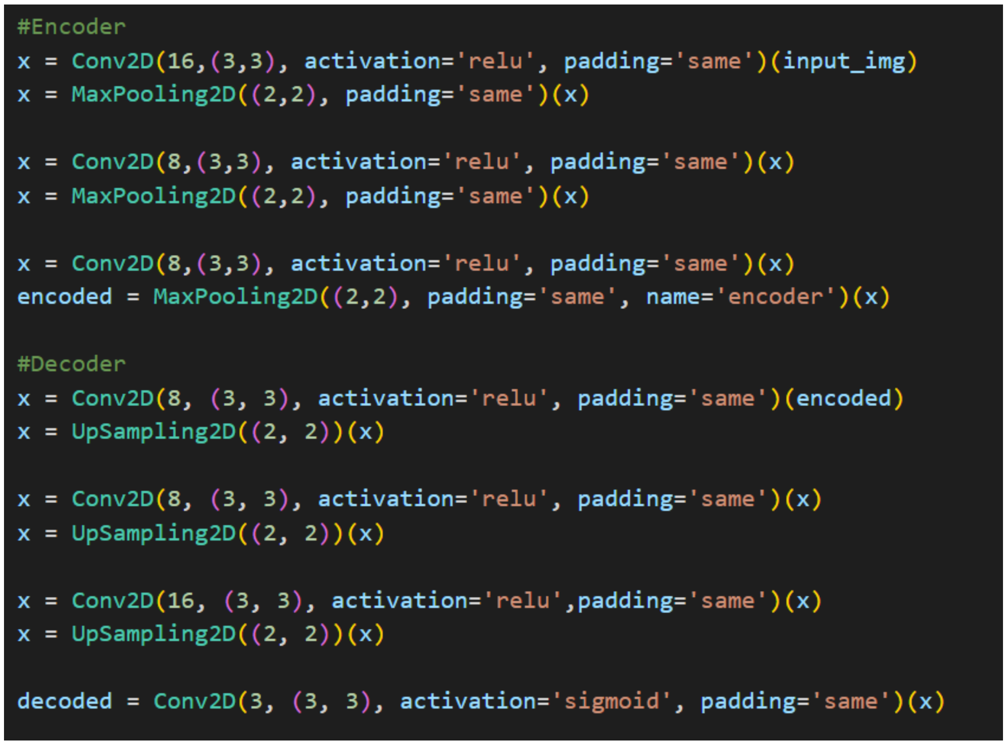
Test 2 loss = 0.9138304591178894 Test 2 accuracy = 0.6948999762535095

Test 3 loss = 0.899131715297699 Test 3 accuracy = 0.7085000276565552

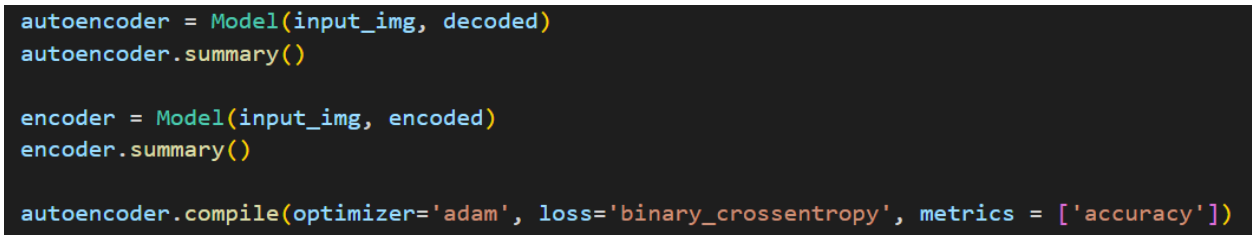
Test 4 loss = 0.9286627173423767 Test 4 accuracy = 0.6725000143051147

It looks like as the model increases in layers, the accuracy of the model stays about the same. The 3rd model is a little more accurate than the 1st model; but the 2nd model is a little better than the 4th model. From the graphs I am concluding that the Max Pooling works better than the Average Pooling because both the Average Pooling models had a lower accuracy rating than the Max Pooling models for both the level 2 and level 3 versions. I think that maybe increasing the layers and using Max Pooling will help the accuracy of the model.

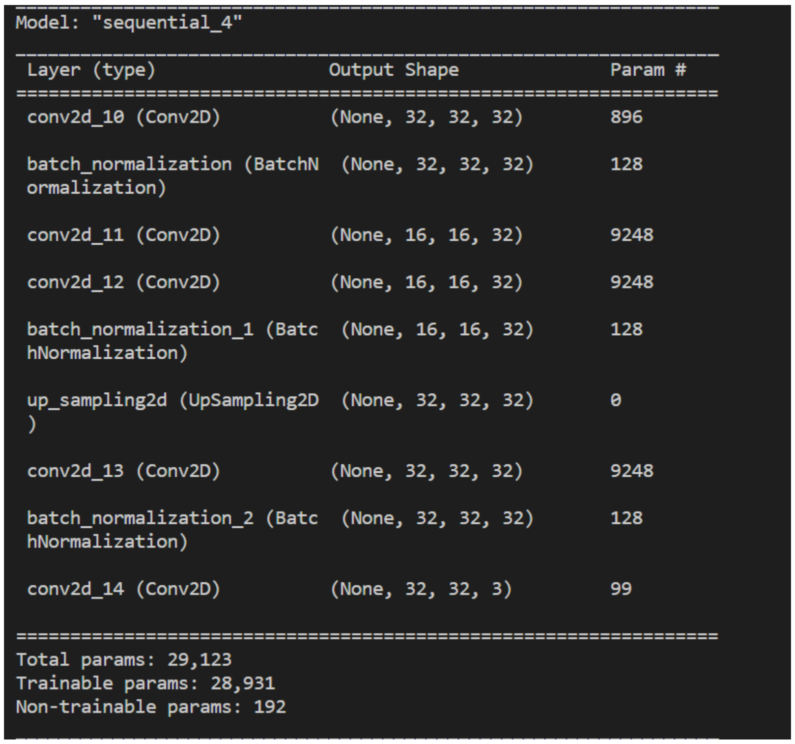
-d- Creating a Convolutional Autoencoder that uses convolution and deconvolution layers to reconstruct the images. Below is a code snippet where I am creating the autoencoder.



```
#Encoder
x = Conv2D(16,(3,3), activation='relu', padding='same')(input_img)
x = MaxPooling2D((2,2), padding='same')(x)

x = Conv2D(8,(3,3), activation='relu', padding='same')(x)
x = MaxPooling2D((2,2), padding='same')(x)

x = Conv2D(8,(3,3), activation='relu', padding='same')(x)
encoded = MaxPooling2D((2,2), padding='same', name='encoder')(x)

#Decoder
x = Conv2D(8, (3, 3), activation='relu', padding='same')(encoded)
x = UpSampling2D((2, 2))(x)

x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)

x = Conv2D(16, (3, 3), activation='relu',padding='same')(x)
x = UpSampling2D((2, 2))(x)

decoded = Conv2D(3, (3, 3), activation='sigmoid', padding='same')(x)
```

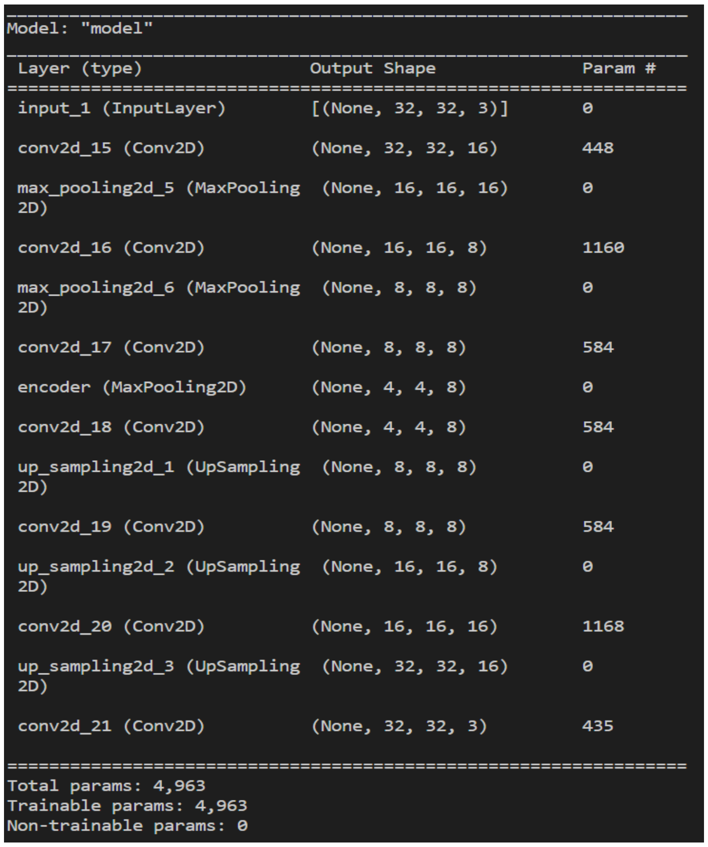


```
autoencoder = Model(input_img, decoded)
autoencoder.summary()

encoder = Model(input_img, encoded)
encoder.summary()

autoencoder.compile(optimizer='adam', loss='binary_crossentropy', metrics = ['accuracy'])
```

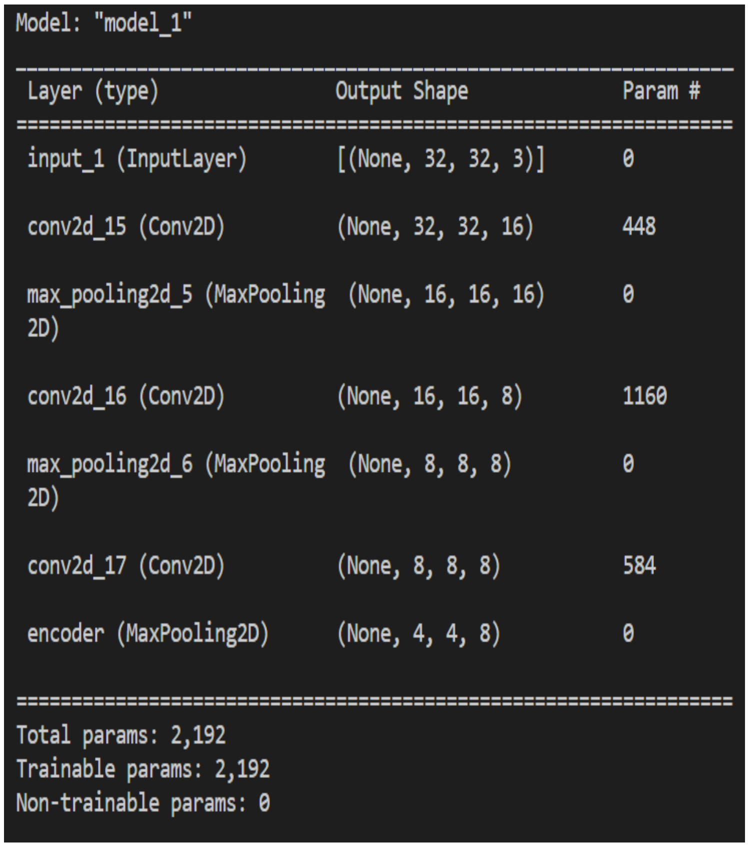


```
_________________________________________________________________
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_10 (Conv2D)          (None, 32, 32, 32)        896

 batch_normalization (BatchN  (None, 32, 32, 32)       128
 ormalization)

 conv2d_11 (Conv2D)          (None, 16, 16, 32)        9248

 conv2d_12 (Conv2D)          (None, 16, 16, 32)        9248

 batch_normalization_1 (Batc  (None, 16, 16, 32)       128
 hNormalization)

 up_sampling2d (UpSampling2D  (None, 32, 32, 32)       0
 )

 conv2d_13 (Conv2D)          (None, 32, 32, 32)        9248

 batch_normalization_2 (Batc  (None, 32, 32, 32)       128
 hNormalization)

 conv2d_14 (Conv2D)          (None, 32, 32, 3)         99

=================================================================
Total params: 29,123
Trainable params: 28,931
Non-trainable params: 192
_________________________________________________________________
```

Above is the output when compiling the model that I will be using in the autoencoder, and below are the results from compiling the encoder and autoencoder. You can see the convolutional and pooling layers.



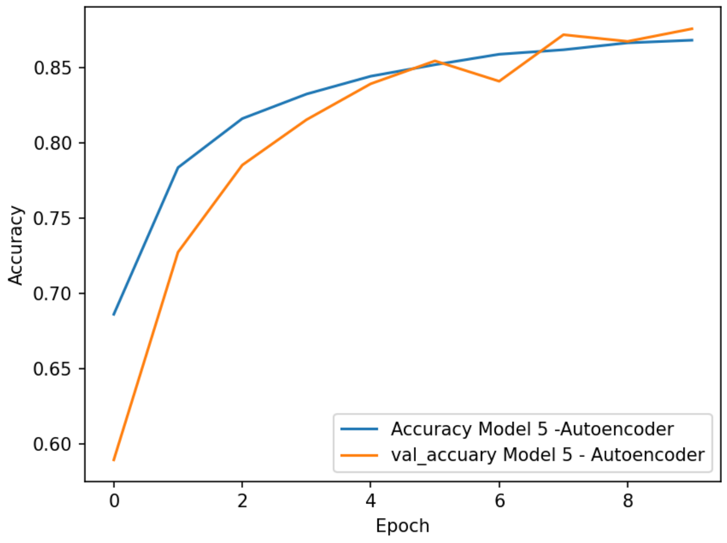
```
_________________________________________________________________
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 32, 32, 3)]       0

 conv2d_15 (Conv2D)          (None, 32, 32, 16)        448

 max_pooling2d_5 (MaxPooling  (None, 16, 16, 16)       0
 2D)

 conv2d_16 (Conv2D)          (None, 16, 16, 8)         1160

 max_pooling2d_6 (MaxPooling  (None, 8, 8, 8)          0
 2D)

 conv2d_17 (Conv2D)          (None, 8, 8, 8)           584

 encoder (MaxPooling2D)      (None, 4, 4, 8)           0

 conv2d_18 (Conv2D)          (None, 4, 4, 8)           584

 up_sampling2d_1 (UpSampling  (None, 8, 8, 8)          0
 2D)

 conv2d_19 (Conv2D)          (None, 8, 8, 8)           584

 up_sampling2d_2 (UpSampling  (None, 16, 16, 8)        0
 2D)

 conv2d_20 (Conv2D)          (None, 16, 16, 16)        1168

 up_sampling2d_3 (UpSampling  (None, 32, 32, 16)       0
 2D)

 conv2d_21 (Conv2D)          (None, 32, 32, 3)         435

=================================================================
Total params: 4,963
Trainable params: 4,963
Non-trainable params: 0
```

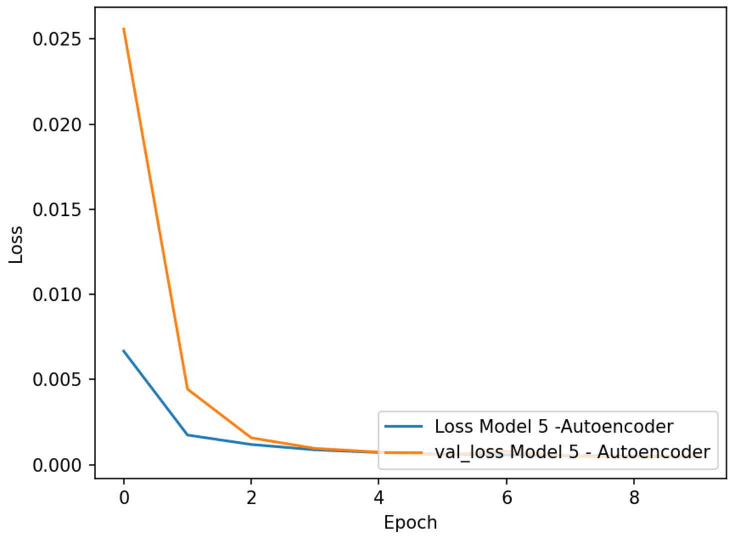


```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 32, 32, 3)]       0

 conv2d_15 (Conv2D)          (None, 32, 32, 16)        448

 max_pooling2d_5 (MaxPooling  (None, 16, 16, 16)       0
 2D)

 conv2d_16 (Conv2D)          (None, 16, 16, 8)         1160

 max_pooling2d_6 (MaxPooling  (None, 8, 8, 8)          0
 2D)

 conv2d_17 (Conv2D)          (None, 8, 8, 8)           584

 encoder (MaxPooling2D)      (None, 4, 4, 8)           0

=================================================================
Total params: 2,192
Trainable params: 2,192
Non-trainable params: 0
```

Ran the autoencoder model on the data for 50 epochs. Took 2 hours to run. The model had around a 90% accuracy rating for both the train accuracy and the validation accuracy. I think adding more epochs and using the autoencoder helped with increasing the accuracy of the model. The above images are to show that the images were successfully reconstructed using the autoencoder. I forgot to make a plot for the 50 epoch test run like I had for the previous part, so I am going to run a smaller version with only 10 epochs on the autoencoder output and see how that looks.

Below are the final two plots, one for the accuracy of the autoencoder model and the other is the loss function for the model as a function of the epochs. The accuracy of the model is growing as the number of epochs grows and the loss function is minimized as well. It looks like the final accuracy of the model is around 85% which is pretty good and comparable to the 50 epoch version I did before.


Accuracy
Epoch
Accuracy Model 5 -Autoencoder
val_accuary Model 5 - Autoencoder


Loss
Epoch
Loss Model 5 -Autoencoder
val_loss Model 5 - Autoencoder