Irfan Ahmed
ICP 5
Due Date: 9/28/2019

a. What you learned in the ICP.

I found this topic to be the most enjoyable so far. One of the main takes away from this ICP was how to logically setup my layers. After my data was train and tested on the dataset, it was normalized. Now the question arises, what should I perform next? I learned that applying one-hot encode would be the best option. One-hot encoding can be applied to the integer representation and doing so a new binary variable is added for each unique integer value. Also, few other concepts like convolution layer, pooling layer, and dropout method were experimented with to better understand the material. Different types of activation and optimizer were used to better understand what purpose they serve, and why one is used over the other. The importance of finding the right batch size and epochs can also affect the accuracy of the results.

b. ICP description what was the task you were performing

1. One of the first task was to upload the dataset and perform train/test on it.
2. Then the X and Y, train/test were normalized between 0 and 1 by dividing by 255.
3. One-hot encode was performed on the y_train, Y_test.
4. Next, the model was layed out using the Keras model, which acts as the blueprint.
5. The first layer of the model was the convolutional layer. This layer takes in the input and runs convolutional filters on them
6. Next, the drop layer was added to prevent overfitting, and batch normalization was also applied. Activation('relu') was used.
7. Polling layer was then added to make the classifier more robust.
8. Additional convolution layers were added to increase the number of filters.
9. After the convolution was finished the data was flattened and added dropout layer.
10. Denser layer was added specified the number of dense layers for each step.
11. In the final layer the number of classed was passed comparable to the number of neurons.
12. The model was finally compiled, and model fit was performed on the X_train, and y_train.
13. Then five new images were valeted to see the accuracy of the model.
14. In the final step 4 hyper-parameters were changed to see how it would affect the overall model accuracy(EX: Epochs, Batch Size, Dropout, and optimizers)

c. Challenges that you faced.

One of the difficult parts was understanding the different layers and how many to add. Adding can hurt the overall model and finding that sweet spot can be difficult. Mostly understanding the code and what is doing can be the most difficult part at times.

d. Screen shots that shows the successful execution of each required step of your code

1) Successfully executing the code and changing 4 hyperparameters in the model.

2) Validating the model on 5 new images (that are not present in the data set and are not used in tra

3) Providing the logical explanation of the changes that you made to hyper parameters and over all

Importing the required libaraies and packages.

```
[1]  !pip install -q tf-nightly
     import matplotlib.pyplot as plt
     import numpy as np
     import os
     import PIL
     import tensorflow as tf

     from tensorflow import keras
     from tensorflow.keras import layers
     #from tensorflow.keras.models import Sequential
     import numpy
     from keras.models import Sequential
     from keras.layers import Dense, Dropout, Flatten, BatchNormalization, Activation
     from keras.layers.convolutional import Conv2D, MaxPooling2D
     from keras.constraints import maxnorm
     from keras.utils import np_utils
     from keras.datasets import cifar10
```

Setting random see for reproducibity

```
[2]  seed = 25
```

```
[2]  seed = 25
```

Loading the dataset. Which can be done by using load_data()function

```
[3]  (X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

Looking at the shape of the data for all the previous varibales declared when the data was loaded

```
[4]  print("The X_shape of the data: ", "size:", " M X M: ", "RBG Value:", '\n')

     print("        x_train shape: ",X_train.shape, '\n')
     print("        x_test shape: ",X_test.shape, '\n')

     print("The X_shape of the data: ", "size:", "label:", '\n')

     print("        x_train shape: ",y_train.shape, '\n')
     print("        x_test shape: ",y_test.shape, '\n')
```

```
The X_shape of the data:  size:  M X M:  RBG Value:

        x_train shape:  (50000, 32, 32, 3)

        x_test shape:  (10000, 32, 32, 3)

The X_shape of the data:  size: label:

        x_train shape:  (50000, 1)

        x_test shape:  (10000, 1)
```

printing each label. It represetns each class.

+ Code

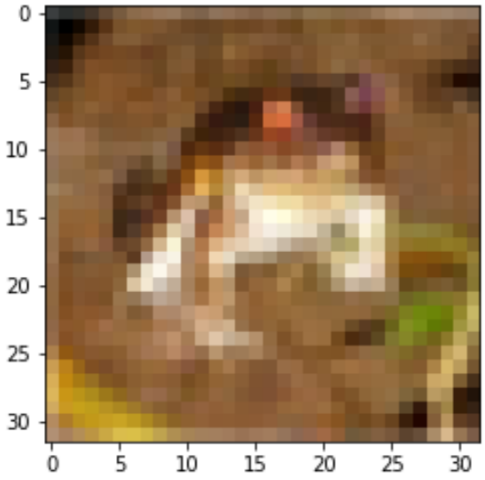
```
print(y_test)
```

```
[[3]
 [8]
 [8]
 ...
 [5]
 [1]
 [7]]
```

looking at the image at a random location

```
[6] plt.imshow(X_train[0])
```

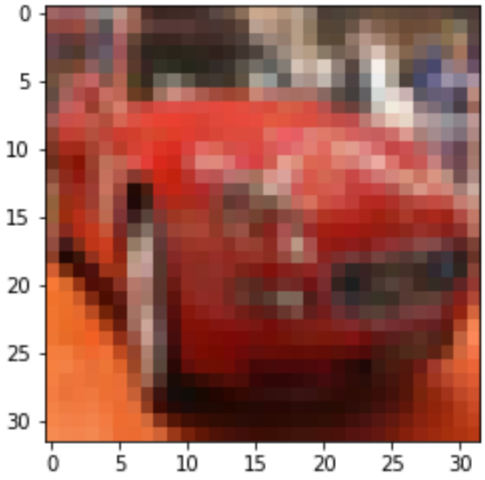
```
<matplotlib.image.AxesImage at 0x7f849b4d9e48>
```




```
[7] plt.imshow(X_train[5])
```

```
plt.imshow(X_train[5])
```

```
<matplotlib.image.AxesImage at 0x7f849bf892e8>
```




```
[8]  #num_classes = 10
     #cifar10_classes = ["airplane", "automobile", "bird", "cat", "deer",
                         #"dog", "frog", "horse", "ship", "truck"]
```

Normazling the data.

The input values are between 0 and 255.

```
[9]  X_train = X_train.astype('float32')
     X_test = X_test.astype('float32')
     X_train /= 255.0 # dividing by 255
     X_test /= 2550
```

```
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255.0 # dividing by 255
X_test /= 2550
```

Converts a class vector (integers) to binary class matrix. This will be done for y_train and y_test. Re

```
[10] y_train = np_utils.to_categorical(y_train,10)
     y_test = np_utils.to_categorical(y_test,10)
```

```
[11] print("y_train",'\n')
     print(y_train[0:5], '\n')
     print("------------------------------------------------------------")
     print("y_test", '\n')
     print(y_test[0:5], '\n')
```

```
y_train

[[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]]

------------------------------------------------------------
y_test

[[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]]
```

specifying the number of classes present in the dataset. The purpose of doing this allows us know in the final layer

```
class_num = y_test.shape[1]
```

Building the model(Blue print)

The convolutional layer simply put takes in a image which is the input and assigns the importance the image. This allows the it to differentiate them.

```
[13] model = Sequential()
     model.add(Conv2D(32, (3,3), input_shape=X_train.shape[1:], padding='same'))
```

The rectified linear activation function or ReLU for short is a piecewise linear function that will outp it will output zero. It has become the default activation function for many types of neural networks train and often achieves better performanc

```
[14] model.add(Activation('relu'))
```

Next, the dropout layer is added. The reason for adding this is to prevent neural networks from over dropout layer is to reduce the capacity or thinning the network during training. Another thing to not maybe needed when it is utilized.

Purpose of adding BatNormaliztaion is to standardize the inputs, meaning it will have mean of zero

```
[15] model.add(Dropout(0.2)) #adding 20% dropout
     model.add(BatchNormalization()) #used to strandarize the input
     model.add(Conv2D(64, (3, 3), padding = 'same'))#2D convolution layer
     model.add(Activation('relu'))
```

The pooling layer is added after the convolutional layer (ReLU). It is just a new layer that is added.

```
model.add(MaxPooling2D(pool_size=(2,2))) #takes in 2x2 pooling window
model.add(Dropout(0.2))
model.add(BatchNormalization())
```

Adding another pooling layer. total of 2 more layers added and reason for this is because the imag

```
[17] model.add(Conv2D(64, (3, 3), padding='same'))
     model.add(Activation('relu'))
     model.add(MaxPooling2D(pool_size=(2, 2)))
     model.add(Dropout(0.2))
     model.add(BatchNormalization())
```

```
[18] model.add(Conv2D(128, (3, 3), padding='same'))
     model.add(Activation('relu'))
     model.add(Dropout(0.2))
     model.add(BatchNormalization())
```

Flatteing the layer This will convert the pooled features map into a single column which is passed

```
[19] model.add(Flatten())
     model.add(Dropout(0.2))
```

The Dense layer is imported. Few things need to be addresed before doing so. First, the number of that the dense layer decreases and will approach the same number of neurons as there are classe

```
model.add(Dense(256, kernel_constraint=maxnorm(3)))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
```

Repeating the code now decresing the units from 256 to 128. Factor of 2

```
[21] model.add(Dense(256, kernel_constraint=maxnorm(3)))
     model.add(Activation('relu'))
     model.add(Dropout(0.2))
     model.add(BatchNormalization())
```

In the final stages of the layer the number of classes are passed for the number of neurons. class
line 12 of code. For the activation 'softmax is used becuase it will select the neuorn that have the

```
[22] model.add(Dense(class_num))
     model.add(Activation('softmax'))
```

The number of epocs are trained and the optiizer that will be used. Epoch is an dataset that is pas
through the nerual networks at once. Epoch is generally the what we want to train the CNN

```
[23]
     epochs = 2
```

Finally, compiling the data and see the results.

```
[24] model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
[25] print(model.summary())
```

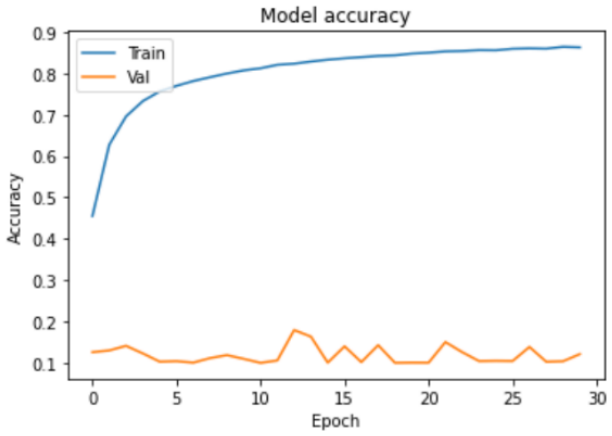
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
activation (Activation)      (None, 32, 32, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 64)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 64)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 64)        36928
```

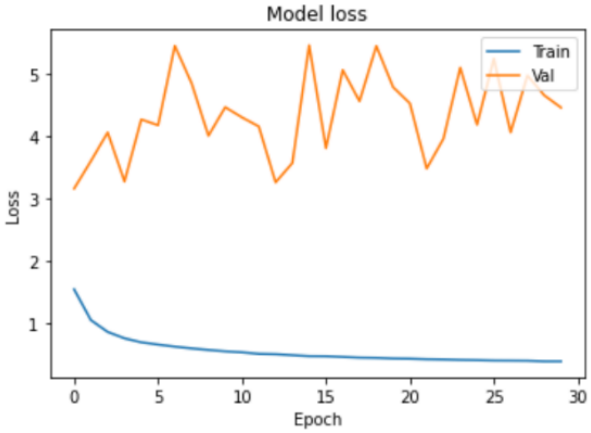
```
numpy.random.seed(seed)
data_history11 = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=
```

```
782/782 [==============================] - 412s 526ms/step - loss: 1.1162 - accuracy:
Epoch 3/30
782/782 [==============================] - 408s 522ms/step - loss: 0.8809 - accuracy:
Epoch 4/30
782/782 [==============================] - 410s 524ms/step - loss: 0.7557 - accuracy:
Epoch 5/30
782/782 [==============================] - 407s 521ms/step - loss: 0.6870 - accuracy:
Epoch 6/30
782/782 [==============================] - 407s 521ms/step - loss: 0.6424 - accuracy:
Epoch 7/30
782/782 [==============================] - 407s 520ms/step - loss: 0.6154 - accuracy:
Epoch 8/30
782/782 [==============================] - 408s 521ms/step - loss: 0.5753 - accuracy:
Epoch 9/30
782/782 [==============================] - 409s 523ms/step - loss: 0.5551 - accuracy:
Epoch 10/30
782/782 [==============================] - 413s 528ms/step - loss: 0.5210 - accuracy:
Epoch 11/30
782/782 [==============================] - 410s 525ms/step - loss: 0.5104 - accuracy:
Epoch 12/30
782/782 [==============================] - 405s 518ms/step - loss: 0.4790 - accuracy:
Epoch 13/30
782/782 [==============================] - 404s 517ms/step - loss: 0.4736 - accuracy:
Epoch 14/30
782/782 [==============================] - 405s 518ms/step - loss: 0.4537 - accuracy:
Epoch 15/30
782/782 [==============================] - 402s 514ms/step - loss: 0.4500 - accuracy:
Epoch 16/30
782/782 [==============================] - 404s 516ms/step - loss: 0.4397 - accuracy:
Epoch 17/30
782/782 [==============================] - 408s 522ms/step - loss: 0.4267 - accuracy:
Epoch 18/30
782/782 [==============================] - 406s 520ms/step - loss: 0.4209 - accuracy:
Epoch 19/30
782/782 [==============================] - 406s 519ms/step - loss: 0.4182 - accuracy:
Epoch 20/30
782/782 [==============================] - 406s 519ms/step - loss: 0.4030 - accuracy:
Epoch 21/30
782/782 [==============================] - 411s 526ms/step - loss: 0.4096 - accuracy:
```

```
plt.plot(data_history11.history['accuracy'])
plt.plot(data_history11.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()

plt.plot(data_history11.history['loss'])
plt.plot(data_history11.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```





```
scores11 = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores11[1]*100))
```

```
Accuracy: 12.09%
```

Now validating the results with random images

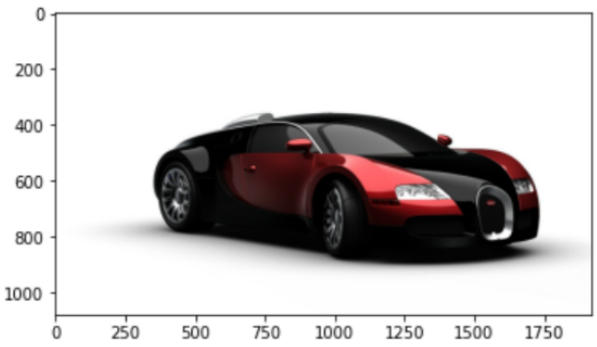
```
#Load the data
from google.colab import files # Use to load data on Google Colab
uploaded = files.upload() # Use to load data on Google Colab
new_image = plt.imread("/content/car-49278_1920.jpg") #Read in the image
```

Choose Files car-49278_1920.jpg

- **car-49278_1920.jpg**(image/jpeg) - 106000 bytes, last modified: 9/26/2020 - 100% done

```
Saving car-49278_1920.jpg to car-49278_1920.jpg
```

```
img = plt.imshow(new_image)
```




```
from skimage.transform import resize
resized_image = resize(new_image, (32,32,3))
img = plt.imshow(resized_image)
```

```
[34] predictions = model.predict(np.array( [resized_image] ))
```

```
predictions
```

```
array([[1.0230566e-02, 7.7509576e-01, 4.0212185e-03, 1.7838222e-01,
        1.8251414e-04, 7.1143741e-03, 7.5949458e-03, 2.6186512e-04,
        1.2788793e-02, 4.3276763e-03]], dtype=float32)
```

```
[36] classification = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

```
[37] list_index = [0,1,2,3,4,5,6,7,8,9]
     x = predictions
     for i in range(10):
       for j in range(10):
         if x[0][list_index[i]] > x[0][list_index[j]]:
           temp = list_index[i]
           list_index[i] = list_index[j]
           list_index[j] = temp
     #Show the sorted labels in order from highest probability to lowest
     print(list_index)
```

```
[1, 3, 8, 0, 6, 5, 9, 2, 7, 4]
```

```
[38] i=0
     for i in range(5):
       print(classification[list_index[i]], ':', round(predictions[0][list_index[i]] * 100, 2), '%')
```

```
automobile : 77.51 %
cat : 17.84 %
ship : 1.28 %
airplane : 1.02 %
frog : 0.76 %
```

Image 2

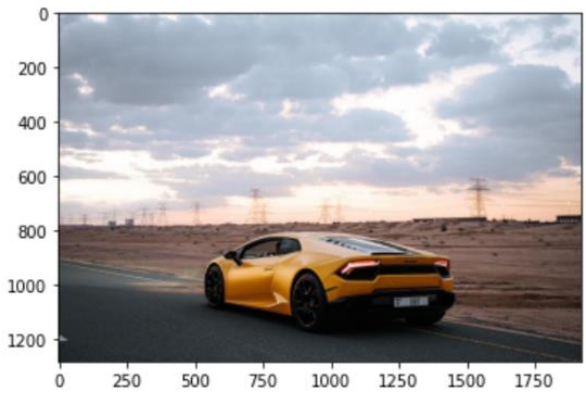
```
[39] #Load the data
     from google.colab import files # Use to load data on Google Colab
     uploaded = files.upload() # Use to load data on Google Colab
     new_image = plt.imread("/content/car-5548242_1920.jpg") #Read in the image
```

Choose Files car-5548242_1920.jpg

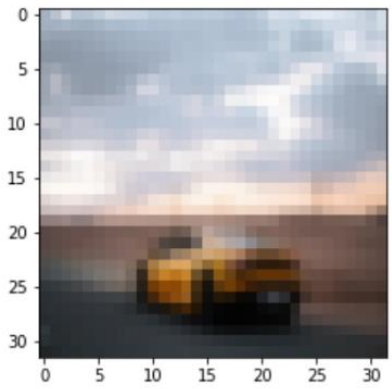
- **car-5548242_1920.jpg**(image/jpeg) - 290156 bytes, last modified: 9/26/2020 - 100% done

```
Saving car-5548242_1920.jpg to car-5548242_1920.jpg
```

```
[40] img = plt.imshow(new_image)
```




```
[41] from skimage.transform import resize
     resized_image = resize(new_image, (32,32,3))
     img = plt.imshow(resized_image)
```



```
predictions = model.predict(np.array( [resized_image] ))
predictions
```

```
array([[2.1332918e-01, 2.7268991e-02, 6.1268936e-04, 9.6222095e-05,
        1.3771025e-04, 6.0589850e-06, 4.2500244e-05, 1.5197129e-05,
        6.9925004e-01, 5.9241354e-02]], dtype=float32)
```

```
[43] list_index = [0,1,2,3,4,5,6,7,8,9]
     x = predictions
     for i in range(10):
       for j in range(10):
         if x[0][list_index[i]] > x[0][list_index[j]]:
           temp = list_index[i]
           list_index[i] = list_index[j]
           list_index[j] = temp
     #Show the sorted labels in order from highest probability to lowest
     print(list_index)
```

```
[8, 0, 9, 1, 2, 4, 3, 6, 7, 5]
```

```
[44] i=0
     for i in range(5):
       print(classification[list_index[i]], ':', round(predictions[0][list_index[i]] * 100, 2), '%')
```

```
ship : 69.93 %
airplane : 21.33 %
truck : 5.92 %
automobile : 2.73 %
bird : 0.06 %
```

image 3

```
[45] #Load the data
     from google.colab import files # Use to load data on Google Colab
     uploaded = files.upload() # Use to load data on Google Colab
     new_image = plt.imread("/content/cuba-1197800_1920.jpg") #Read in the image

     img = plt.imshow(new_image)
```

image 3

```
#Load the data
from google.colab import files # Use to load data on Google Colab
uploaded = files.upload() # Use to load data on Google Colab
new_image = plt.imread("/content/cuba-1197800_1920.jpg") #Read in the image

img = plt.imshow(new_image)
```

```
Choose Files cuba-1197800_1920.jpg
```

- **cuba-1197800_1920.jpg**(image/jpeg) - 837906 bytes, last modified: 9/26/2020 - 100% done

```
Saving cuba-1197800_1920.jpg to cuba-1197800_1920.jpg
```


```
[46] from skimage.transform import resize
resized_image = resize(new_image, (32,32,3))
img = plt.imshow(resized_image)

predictions = model.predict(np.array( [resized_image] ))
predictions
```

```
array([[1.3416114e-04, 1.3246661e-02, 1.0425618e-02, 1.5270984e-03,
        9.4200277e-01, 1.5187241e-03, 8.1295198e-05, 3.5432293e-03,
        2.6010284e-03, 2.4919322e-02]], dtype=float32)
```

```
list_index = [0,1,2,3,4,5,6,7,8,9]
x = predictions
for i in range(10):
  for j in range(10):
    if x[0][list_index[i]] > x[0][list_index[j]]:
      temp = list_index[i]
      list_index[i] = list_index[j]
      list_index[j] = temp
#Show the sorted labels in order from highest probability to lowest
print(list_index)

i=0
for i in range(5):
  print(classification[list_index[i]], ':', round(predictions[0][list_index[i]] * 100, 2), '%')
```

```
[4, 9, 1, 2, 7, 8, 3, 5, 0, 6]
deer : 94.2 %
truck : 2.49 %
automobile : 1.32 %
bird : 1.04 %
horse : 0.35 %
```

image 4

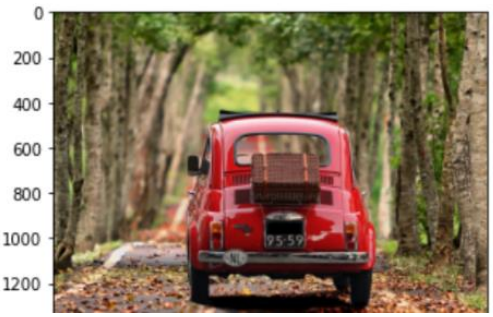
```
[48] #Load the data
from google.colab import files # Use to load data on Google Colab
uploaded = files.upload() # Use to load data on Google Colab
new_image = plt.imread("/content/fiat-4322521_1920.jpg") #Read in the image

img = plt.imshow(new_image)
```

Choose Files fiat-4322521_1920.jpg

- **fiat-4322521_1920.jpg**(image/jpeg) - 696404 bytes, last modified: 9/26/2020 - 100% done

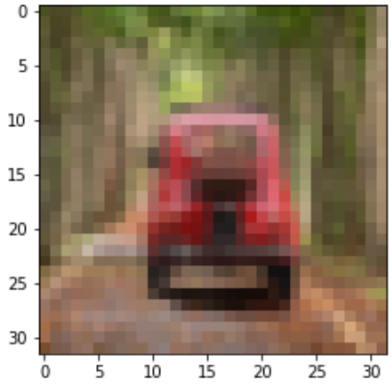
```
Saving fiat-4322521_1920.jpg to fiat-4322521_1920.jpg
```



```
from skimage.transform import resize
resized_image = resize(new_image, (32,32,3))
img = plt.imshow(resized_image)

predictions = model.predict(np.array( [resized_image] ))
predictions
```

```
array([[1.0125967e-04, 6.4570587e-03, 5.8513653e-04, 1.8047021e-03,
        1.0990113e-03, 2.4159544e-04, 7.3753879e-05, 3.6307792e-05,
        4.9243611e-03, 9.8467690e-01]], dtype=float32)
```




```
[50] list_index = [0,1,2,3,4,5,6,7,8,9]
     x = predictions
     for i in range(10):
       for j in range(10):
         if x[0][list_index[i]] > x[0][list_index[j]]:
           temp = list_index[i]
           list_index[i] = list_index[j]
           list_index[j] = temp
     #Show the sorted labels in order from highest probability to lowest
     print(list_index)

     i=0
     for i in range(5):
       print(classification[list_index[i]], ':', round(predictions[0][list_index[i]] * 100, 2), '%')
```

```
[9, 1, 8, 3, 4, 2, 5, 0, 6, 7]
truck : 98.47 %
automobile : 0.65 %
ship : 0.49 %
cat : 0.18 %
deer : 0.11 %
```

image 5

```
#Load the data
from google.colab import files # Use to load data on Google Colab
uploaded = files.upload() # Use to load data on Google Colab
new_image = plt.imread("/content/ford-498244_1920.jpg") #Read in the image

img = plt.imshow(new_image)
```

Choose Files ford-498244_1920.jpg

- **ford-498244_1920.jpg**(image/jpeg) - 582919 bytes, last modified: 9/26/2020 - 100% done

```
Saving ford-498244_1920.jpg to ford-498244_1920.jpg
```

[52]

```
from skimage.transform import resize
resized_image = resize(new_image, (32,32,3))
img = plt.imshow(resized_image)

predictions = model.predict(np.array( [resized_image] ))
predictions
```

```
array([[1.2390652e-03, 1.7010628e-01, 6.8462832e-04, 8.4482197e-04,
        4.1197411e-05, 7.4195523e-05, 7.6745840e-04, 8.3668010e-06,
        6.3244629e-01, 1.9378777e-01]], dtype=float32)
```

```
list_index = [0,1,2,3,4,5,6,7,8,9]
x = predictions
for i in range(10):
  for j in range(10):
    if x[0][list_index[i]] > x[0][list_index[j]]:
      temp = list_index[i]
      list_index[i] = list_index[j]
      list_index[j] = temp
#Show the sorted labels in order from highest probability to lowest
print(list_index)

i=0
for i in range(5):
  print(classification[list_index[i]], ':', round(predictions[0][list_index[i]] * 100, 2), '%')
```

```
[8, 9, 1, 0, 3, 6, 2, 5, 4, 7]
ship : 63.24 %
truck : 19.38 %
automobile : 17.01 %
airplane : 0.12 %
cat : 0.08 %
```

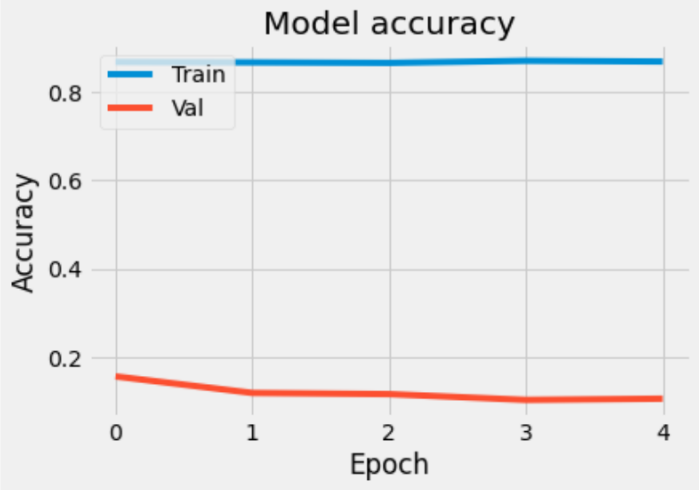
4 hyper-parameters that were changed

```
numpy.random.seed(seed)
data_history00 = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=5, batch_size=64)
```

```
Epoch 1/5
782/782 [==============================] - 407s 520ms/step - loss: 0.3779 - accuracy: 0.8672 - val_loss: 3.50
Epoch 2/5
782/782 [==============================] - 408s 522ms/step - loss: 0.3819 - accuracy: 0.8664 - val_loss: 3.88
Epoch 3/5
782/782 [==============================] - 410s 525ms/step - loss: 0.3806 - accuracy: 0.8653 - val_loss: 4.42
Epoch 4/5
782/782 [==============================] - 409s 523ms/step - loss: 0.3705 - accuracy: 0.8700 - val_loss: 5.83
Epoch 5/5
782/782 [==============================] - 407s 520ms/step - loss: 0.3691 - accuracy: 0.8683 - val_loss: 5.59
```

```
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
```

```
[ ] plt.plot(data_history00.history['accuracy'])
    plt.plot(data_history00.history['val_accuracy'])
    plt.title('Model accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Val'], loc='upper left')
    plt.show()
```



```
plt.plot(data_history00.history['loss'])
plt.plot(data_history00.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```


```
[ ] # Model evaluation
    scores2 = model.evaluate(X_test, y_test, verbose=0)
    print("Accuracy: %.2f%%" % (scores2[1]*100))
```

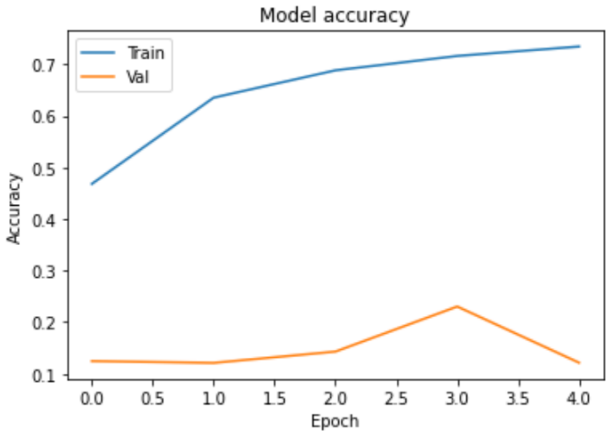
```
Accuracy: 10.56%
```

changing batch size to 32

```
numpy.random.seed(seed)
data_history1 = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=5, batch_size=32)
```

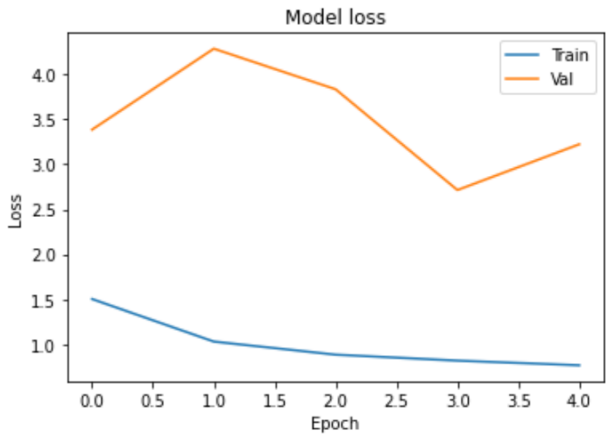
```
Epoch 1/5
1563/1563 [==============================] - 436s 278ms/step - loss: 1.8076 - accuracy: 0.3782 - val_loss: 3.
Epoch 2/5
1563/1563 [==============================] - 433s 277ms/step - loss: 1.0773 - accuracy: 0.6168 - val_loss: 4.
Epoch 3/5
1563/1563 [==============================] - 433s 277ms/step - loss: 0.8875 - accuracy: 0.6876 - val_loss: 3.
Epoch 4/5
1563/1563 [==============================] - 433s 277ms/step - loss: 0.8112 - accuracy: 0.7154 - val_loss: 2.
Epoch 5/5
1563/1563 [==============================] - 434s 277ms/step - loss: 0.7467 - accuracy: 0.7392 - val_loss: 3.
```

[41]
```
plt.plot(data_history1.history['accuracy'])
plt.plot(data_history1.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()
```



Epoch

```
[42] plt.plot(data_history1.history['loss'])
     plt.plot(data_history1.history['val_loss'])
     plt.title('Model loss')
     plt.ylabel('Loss')
     plt.xlabel('Epoch')
     plt.legend(['Train', 'Val'], loc='upper right')
     plt.show()
```




```
[43] # Model evaluation
     scores3 = model.evaluate(X_test, y_test, verbose=0)
     print("Accuracy: %.2f%%" % (scores3[1]*100))
```

```
Accuracy: 12.15%
```

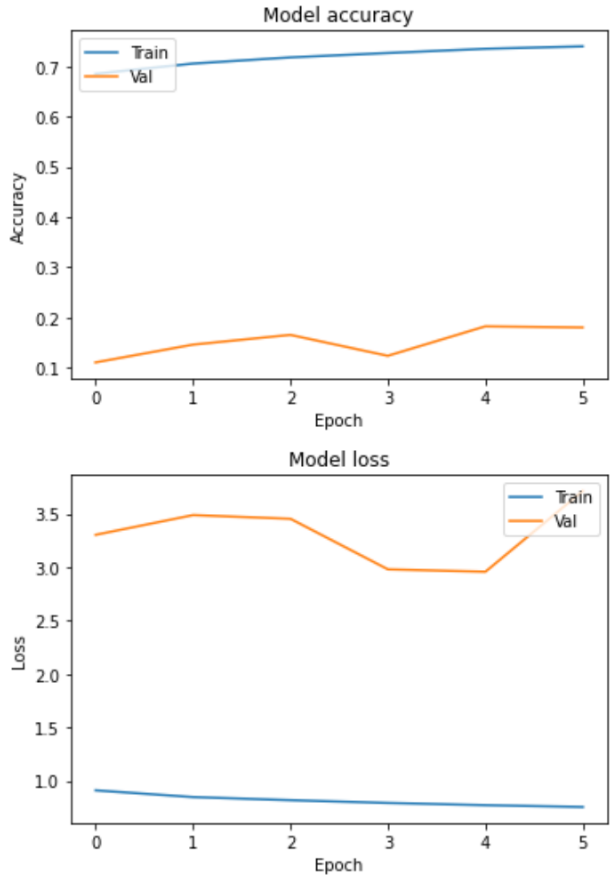
changing epochs to 6

```
numpy.random.seed(seed)
data_history3 = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=6, batch_size=16)
```

```
Epoch 1/6
3125/3125 [==============================] - 486s 155ms/step - loss: 0.9100 - accuracy: 0.6856 - val_loss
Epoch 2/6
3125/3125 [==============================] - 483s 154ms/step - loss: 0.8469 - accuracy: 0.7059 - val_loss
Epoch 3/6
3125/3125 [==============================] - 490s 157ms/step - loss: 0.8178 - accuracy: 0.7183 - val_loss
Epoch 4/6
3125/3125 [==============================] - 488s 156ms/step - loss: 0.7920 - accuracy: 0.7270 - val_loss
Epoch 5/6
3125/3125 [==============================] - 489s 157ms/step - loss: 0.7709 - accuracy: 0.7354 - val_loss
Epoch 6/6
3125/3125 [==============================] - 488s 156ms/step - loss: 0.7541 - accuracy: 0.7403 - val_loss
```

```
plt.plot(data_history3.history['accuracy'])
plt.plot(data_history3.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()

plt.plot(data_history3.history['loss'])
plt.plot(data_history3.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```



```
[46] # Model evaluation
     scores4 = model.evaluate(X_test, y_test, verbose=0)
     print("Accuracy: %.2f%%" % (scores4[1]*100))
```

```
Accuracy: 17.97%
```

chaning optimizer using Adamax instead of Adam. Also, the learing rate is changed It is a variant of Adam based on the infin
parameters follow those provided in the paper. Adamax is sometimes superior to adam, specially in models with embedding

```
[61] model.compile(loss='categorical_crossentropy', optimizer='adamax', metrics=['accuracy'])
     numpy.random.seed(seed)
     data_history6 = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=5, batch_size=64)
```

```
Epoch 1/5
782/782 [==============================] - 399s 508ms/step - loss: 0.4548 - accuracy: 0.8434 - val_loss: 3.19
Epoch 2/5
782/782 [==============================] - 398s 509ms/step - loss: 0.3904 - accuracy: 0.8643 - val_loss: 3.37
Epoch 3/5
782/782 [==============================] - 397s 508ms/step - loss: 0.3584 - accuracy: 0.8737 - val_loss: 3.21
Epoch 4/5
782/782 [==============================] - 397s 508ms/step - loss: 0.3360 - accuracy: 0.8821 - val_loss: 3.01
Epoch 5/5
782/782 [==============================] - 397s 507ms/step - loss: 0.3171 - accuracy: 0.8889 - val_loss: 3.13
```

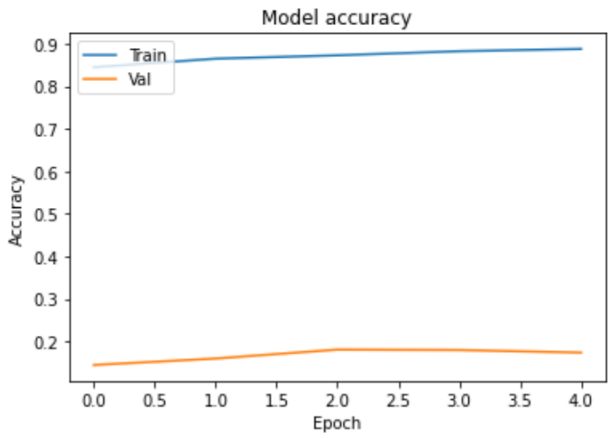
```
[48] numpy.random.seed(seed)
     data_history6 = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=5, batch_size=64)
     #numpy.random.seed(seed)
     #data_history5 = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=5, batch_size=64)
```

```
Epoch 1/5
782/782 [==============================] - 401s 511ms/step - loss: 0.7548 - val_loss: 2.7824
Epoch 2/5
782/782 [==============================] - 400s 512ms/step - loss: 0.6258 - val_loss: 3.6033
Epoch 3/5
782/782 [==============================] - 401s 512ms/step - loss: 0.6160 - val_loss: 3.8410
Epoch 4/5
782/782 [==============================] - 401s 512ms/step - loss: 0.5665 - val_loss: 3.5670
Epoch 5/5
782/782 [==============================] - 401s 513ms/step - loss: 0.5655 - val_loss: 2.9442
```

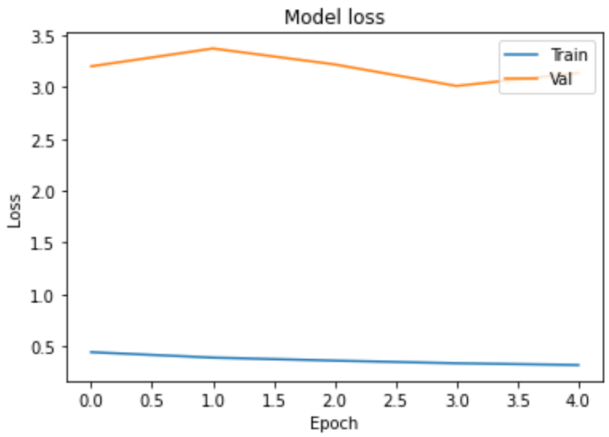
```
[62] history_dict = data_history6.history
     print(history_dict.keys())
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
plt.plot(data_history6.history['accuracy'])
plt.plot(data_history6.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()
```



```
plt.plot(data_history6.history['loss'])
plt.plot(data_history6.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```




```
[65] # Model evaluation
     scores5 = model.evaluate(X_test, y_test, verbose=0)
     print("Accuracy: %.2f%%" % (scores5[1]*100))
```

```
Accuracy: 17.36%
```

chaning dropout to 0.3

```
model.add(Dense(256, kernel_constraint=maxnorm(3)))
model.add(Activation('relu'))
model.add(Dropout(0.3))
model.add(BatchNormalization())
```

```
[67] model.add(Dense(class_num))
     model.add(Activation('softmax'))
```

```
[68] model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

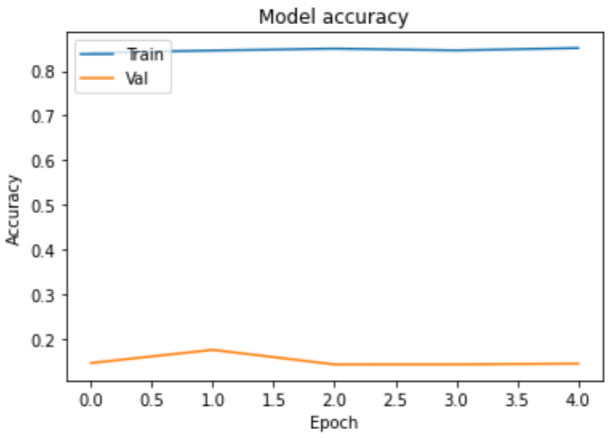
```
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
activation_7 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
dropout_7 (Dropout)          (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization_6 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
activation_8 (Activation)    (None, 32, 32, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
dropout_8 (Dropout)          (None, 16, 16, 64)        0
_________________________________________________________________
batch_normalization_7 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
activation_9 (Activation)    (None, 16, 16, 64)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
dropout_9 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
batch_normalization_8 (Batch (None, 8, 8, 64)          256
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
activation_10 (Activation)   (None, 8, 8, 128)         0
_________________________________________________________________
dropout_10 (Dropout)         (None, 8, 8, 128)         0
_________________________________________________________________
batch_normalization_9 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dropout_11 (Dropout)         (None, 8192)              0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               2097408
_________________________________________________________________
activation_11 (Activation)   (None, 256)               0
_________________________________________________________________
dropout_12 (Dropout)         (None, 256)               0
_________________________________________________________________
batch_normalization_10 (Batc (None, 256)               1024
_________________________________________________________________
dense_4 (Dense)              (None, 256)               65792
```

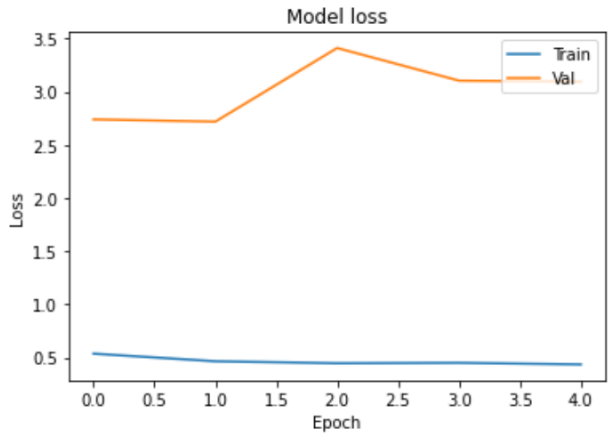
```
numpy.random.seed(seed)
data_history7 = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=5, batch_size=64)
```

```
Epoch 1/5
782/782 [==============================] - 402s 512ms/step - loss: 0.6218 - accuracy: 0.8215 - val_loss: 2.7
Epoch 2/5
782/782 [==============================] - 400s 512ms/step - loss: 0.4504 - accuracy: 0.8529 - val_loss: 2.7
Epoch 3/5
782/782 [==============================] - 398s 508ms/step - loss: 0.4251 - accuracy: 0.8589 - val_loss: 3.4
Epoch 4/5
782/782 [==============================] - 399s 510ms/step - loss: 0.4265 - accuracy: 0.8559 - val_loss: 3.1
Epoch 5/5
782/782 [==============================] - 399s 511ms/step - loss: 0.4128 - accuracy: 0.8570 - val_loss: 3.0
```

[71]
```
plt.plot(data_history7.history['accuracy'])
plt.plot(data_history7.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()
```



```
plt.plot(data_history7.history['loss'])
plt.plot(data_history7.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```




```
# Model evaluation
scores6 = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores6[1]*100))
```

```
Accuracy: 14.39%
```

e. Output file link if applicable.

https://github.com/UMKC-APL-BigDataAnalytics/icp5-irfancheemaa

f. Video link (YouTube or any other publicly available video platform).

You should have access to the link. If there are any issues, please let me know.

https://umkc.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=a7676aed-4d2a-4e69-95a5-ac440187923e

g. Any inside about the data or the ICP in general

I enjoyed this topic and always wanted to learn more about it and how to implement it. It would be nice to learn few more variations of this method with smaller dataset.