**ASSIGNMENT 8**

**Name: varna nemulla**

**ID: 700744920**

**Use Case Description:**

**LeNet5, AlexNet, Vgg16, Vgg19**

**1. Training the model**

**2. Evaluating the model**

**Programming elements:**

**1. About CNN**

**2. Hyperparameters of CNN**

**3. Image classification with CNN**

**In class programming:**

**1. Tune hyperparameter and make necessary addition to the baseline model to improve validation accuracy and reduce validation loss.**

**2. Provide logical description of which steps lead to improved response and what was its impact on architecture behavior.**

**3. Create at least two more visualizations using matplotlib (Other than provided in the source file)**

**4. Use dataset of your own choice and implement baseline models provided.**

**5. Apply modified architecture to your own selected dataset and train it. 6. Evaluate your model on testing set.**

**7. Save the improved model and use it for prediction on testing data**

**8. Provide plot of confusion matric**

**9. Provide Training and testing Loss and accuracy plots in one plot using subplot command and history object.**

**10. Provide at least two more visualizations reflecting your solution.**

**11. Provide logical description of which steps lead to improved response for new dataset when compared with baseline model and enhance architecture and what was its impact on architecture behavior.**

**Source code:**

**# Import keras**

**from keras.models import Sequential**

**from keras.preprocessing import image**

**from keras.layers import Activation,Dense,Dropout,Conv2D,Flatten,MaxPooling2D,BatchNormalization**

**from keras.datasets import cifar10**

**from keras import optimizers**

**from matplotlib import pyplot as plt**

**#generate cifar10 data**

**(x\_train,y\_train),(x\_test,y\_test) = cifar10.load\_data()**

**#config parameters**

**num\_classes = 10**

**input\_shape = x\_train.shape[1:4]**

**optimizer = optimizers.Adam(lr=0.001)**

**#convert label to one-hot**

**one\_hot\_y\_train = keras.utils.to\_categorical(y\_train,num\_classes=num\_classes)**

**one\_hot\_y\_test = keras.utils.to\_categorical(y\_test,num\_classes=num\_classes)**

**# check data**

**plt.imshow(x\_train[1])**

**print(x\_train[1].shape)**

**# build model(similar to VGG16, only change the input and output shape)**

**model = Sequential()**

**model.add(Conv2D(64,(3,3),activation='relu',input\_shape=input\_shape,padding='same'))**

**model.add(BatchNormalization())**

**model.add(Conv2D(64,(3,3),activation='relu',padding='same'))**

**model.add(BatchNormalization())**

**model.add(MaxPooling2D(pool\_size=(2,2),strides=(2,2)))**

**model.add(Dropout(0.25))**

**model.add(Conv2D(128,(3,3),activation='relu',padding='same'))**

**model.add(BatchNormalization())**

**model.add(Conv2D(128,(3,3),activation='relu',padding='same'))**

**model.add(BatchNormalization())**

**model.add(MaxPooling2D(pool\_size=(2,2),strides=(2,2)))**

**model.add(Dropout(0.25))**

**model.add(Conv2D(256,(3,3),activation='relu',padding='same'))**

**model.add(BatchNormalization())**

**model.add(Conv2D(256,(3,3),activation='relu',padding='same'))**

**model.add(BatchNormalization())**

**model.add(Conv2D(256,(3,3),activation='relu',padding='same'))**

**model.add(BatchNormalization())**

**model.add(MaxPooling2D(pool\_size=(2,2),strides=(2,2)))**

**model.add(Dropout(0.25))**

**model.add(Conv2D(512,(3,3),activation='relu',padding='same'))**

**model.add(BatchNormalization())**

**model.add(Conv2D(512,(3,3),activation='relu',padding='same'))**

**model.add(BatchNormalization())**

**model.add(Conv2D(512,(3,3),activation='relu',padding='same'))**

**model.add(BatchNormalization())**

**model.add(MaxPooling2D(pool\_size=(2,2),strides=(2,2)))**

**model.add(Dropout(0.25))**

**model.add(Conv2D(512,(3,3),activation='relu',padding='same'))**

**model.add(BatchNormalization())**

**model.add(Conv2D(512,(3,3),activation='relu',padding='same'))**

**model.add(BatchNormalization())**

**model.add(Conv2D(512,(3,3),activation='relu',padding='same'))**

**model.add(BatchNormalization())**

**model.add(MaxPooling2D(pool\_size=(2,2),strides=(2,2)))**

**model.add(Dropout(0.25))**

**model.add(Flatten())**

**model.add(Dense(4096,activation='relu'))**

**model.add(Dense(2048, activation='relu'))**

**model.add(Dense(1024, activation='relu'))**

**model.add(Dropout(0.5))**

**model.add(Dense(num\_classes))**

**model.add(Activation('softmax'))**

**model.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])**

**model.summary()**

**history = model.fit(x=x\_train, y=one\_hot\_y\_train, batch\_size=128, epochs=30, validation\_split=0.1)**

**# evaluate**

**print(model.metrics\_names)**

**model.evaluate(x=x\_test,y=one\_hot\_y\_test,batch\_size=512)**

**model.save("keras-VGG16-cifar10.h5")**

**plt.imshow(x\_test[1000])**

**result = model.predict(x\_test[1000:1001]).tolist()**

**predict = 0**

**expect = y\_test[1000][0]**

**for i,\_ in enumerate(result[0]):**

**if result[0][i] > result[0][predict]:**

**predict = i**

**print("predict class:",predict)**

**print("expected class:",expect)**

**# save model**

**model.save("keras-VGG16-cifar10.h5")**

**#plot the training and validation accuracy**

**plt.plot(history.history['accuracy'], label='training accuracy')**

**plt.plot(history.history['val\_accuracy'], label='validation accuracy')**

**plt.title('Training and validation accuracy')**

**plt.xlabel('Epoch')**

**plt.ylabel('Accuracy')**

**plt.legend()**

**plt.show()**

**#plot the training and validation loss**

**plt.plot(history.history['loss'], label='training loss')**

**plt.plot(history.history['val\_loss'], label='validation loss')**

**plt.title('Training and validation loss')**

**plt.xlabel('Epoch')**

**plt.ylabel('Loss')**

**plt.legend()**

**plt.show()**

**import numpy as np**

**from sklearn.metrics import confusion\_matrix**

**# calculate the confusion matrix**

**y\_pred = model.predict(x\_test)**

**y\_pred\_classes = np.argmax(y\_pred, axis=1)**

**y\_true = y\_test.ravel()**

**cm = confusion\_matrix(y\_true, y\_pred\_classes)**

**# plot the confusion matrix**

**plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)**

**plt.title('Confusion matrix')**

**plt.colorbar()**

**tick\_marks = np.arange(num\_classes)**

**plt.xticks(tick\_marks, range(num\_classes))**

**plt.yticks(tick\_marks, range(num\_classes))**

**plt.xlabel('Predicted label')**

**plt.ylabel('True label')**

**plt.show()**

**# plot a histogram of the predicted probabilities for a sample image**

**plt.hist(y\_pred[1000])**

**plt.title('Predicted probabilities')**

**plt.xlabel('Probability')**

**plt.ylabel('Frequency')**

**plt.show()**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**import tensorflow as tf**

**from tensorflow.keras.datasets import mnist**

**from tensorflow.keras.optimizers import RMSprop**

**from keras.preprocessing import image**

**from tensorflow.keras.preprocessing.image import ImageDataGenerator**

**from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout, BatchNormalization**

**%matplotlib inline**

**#cifar100 = tf.keras.datasets.cifar100**

**(X\_train,Y\_train) , (X\_test,Y\_test) = cifar10.load\_data()**

**classes = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']**

**plt.figure(figsize = (16,16))**

**for i in range(100):**

**plt.subplot(10,10,1+i)**

**plt.axis('off')**

**plt.imshow(X\_train[i], cmap = 'gray')**

**from sklearn.model\_selection import train\_test\_split**

**x\_train, x\_val, y\_train, y\_val = train\_test\_split(X\_train,Y\_train,test\_size=0.2)**

**from keras.utils.np\_utils import to\_categorical**

**y\_train = to\_categorical(y\_train, num\_classes = 10)**

**y\_val = to\_categorical(y\_val, num\_classes = 10)**

**print(x\_train.shape)**

**print(y\_train.shape)**

**print(x\_val.shape)**

**print(y\_val.shape)**

**print(X\_test.shape)**

**print(Y\_test.shape)**

**train\_datagen = ImageDataGenerator(**

**preprocessing\_function = tf.keras.applications.vgg19.preprocess\_input,**

**rotation\_range=10,**

**zoom\_range = 0.1,**

**width\_shift\_range = 0.1,**

**height\_shift\_range = 0.1,**

**shear\_range = 0.1,**

**horizontal\_flip = True**

**)**

**train\_datagen.fit(x\_train)**

**val\_datagen = ImageDataGenerator(preprocessing\_function = tf.keras.applications.vgg19.preprocess\_input)**

**val\_datagen.fit(x\_val)**

**from keras.callbacks import ReduceLROnPlateau**

**learning\_rate\_reduction = ReduceLROnPlateau(monitor='val\_accuracy',**

**patience=3,**

**verbose=1,**

**factor=0.5,**

**min\_lr=0.00001)**

**vgg\_model = tf.keras.applications.VGG19(**

**include\_top=False,**

**weights=None,**

**input\_shape=(32,32,3),**

**)**

**vgg\_model.summary()**

**model = tf.keras.Sequential()**

**model.add(vgg\_model)**

**model.add(Flatten())**

**model.add(Dense(1024, activation = 'relu'))**

**model.add(BatchNormalization())**

**model.add(Dense(1024, activation = 'relu'))**

**model.add(BatchNormalization())**

**model.add(Dense(256, activation = 'relu'))**

**model.add(BatchNormalization())**

**model.add(Dropout(0.5))**

**model.add(Dense(10, activation = 'softmax'))**

**model.summary()**

**optimizer = tf.keras.optimizers.SGD(learning\_rate = 0.001, momentum = 0.9)**

**model.compile(optimizer= optimizer,**

**loss='categorical\_crossentropy',**

**metrics=['accuracy'])**

**history = model.fit(**

**train\_datagen.flow(x\_train, y\_train, batch\_size = 128),**

**validation\_data = val\_datagen.flow(x\_val,y\_val, batch\_size = 128),**

**epochs = 100,**

**verbose = 1,**

**callbacks = [learning\_rate\_reduction]**

**)**

**acc = history.history['accuracy']**

**val\_acc = history.history['val\_accuracy']**

**plt.figure()**

**plt.plot(acc,color = 'purple',label = 'Training Acuracy')**

**plt.plot(val\_acc,color = 'blue',label = 'Validation Accuracy')**

**plt.legend()**

**loss = history.history['loss']**

**val\_loss = history.history['val\_loss']**

**plt.figure()**

**plt.plot(loss,color = 'green',label = 'Training Loss')**

**plt.plot(val\_loss,color = 'red',label = 'Validation Loss')**

**plt.legend()**

**X\_test = tf.keras.applications.vgg19.preprocess\_input(X\_test)**

**y\_pred = np.argmax(model.predict(X\_test), axis=-1)**

**y\_pred[:10]**

**from sklearn.metrics import confusion\_matrix, accuracy\_score**

**print('Testing Accuarcy : ', accuracy\_score(Y\_test, y\_pred))**

**cm = confusion\_matrix(Y\_test, y\_pred)**

**cm**

**import itertools**

**def plot\_confusion\_matrix(cm, classes,**

**normalize=False,**

**title='Confusion matrix',**

**cmap=plt.cm.Greens):**

**"""**

**This function prints and plots the confusion matrix.**

**Normalization can be applied by setting `normalize=True`.**

**"""**

**plt.imshow(cm, interpolation='nearest', cmap=cmap)**

**plt.title(title)**

**plt.colorbar()**

**tick\_marks = np.arange(len(classes))**

**plt.xticks(tick\_marks, classes, rotation=30)**

**plt.yticks(tick\_marks, classes)**

**if normalize:**

**cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]**

**print("Normalized confusion matrix")**

**else:**

**print('Confusion matrix, without normalization')**

**#print(cm)**

**thresh = cm.max() / 2.**

**for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):**

**plt.text(j, i, cm[i, j],**

**horizontalalignment="center",**

**color="white" if cm[i, j] > thresh else "black")**

**plt.tight\_layout()**

**plt.ylabel('True label')**

**plt.xlabel('Predicted label')**

**plt.figure(figsize=(8,8))**

**plot\_confusion\_matrix(cm,classes)**

**Description:** In this use case, I have Tuned hyperparameter and made necessary addition to the baseline model in order to improve validation accuracy and reduce validation loss.

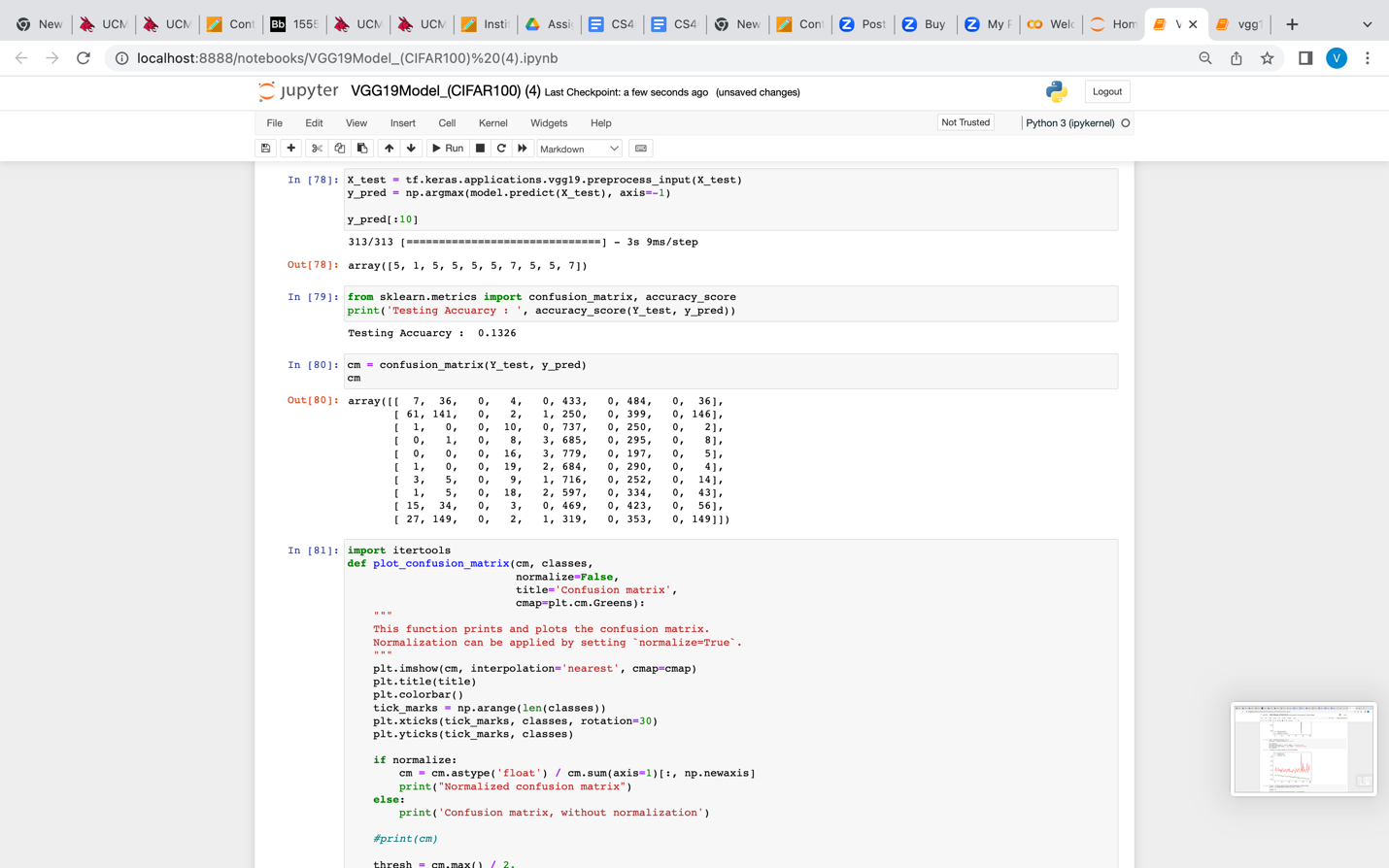
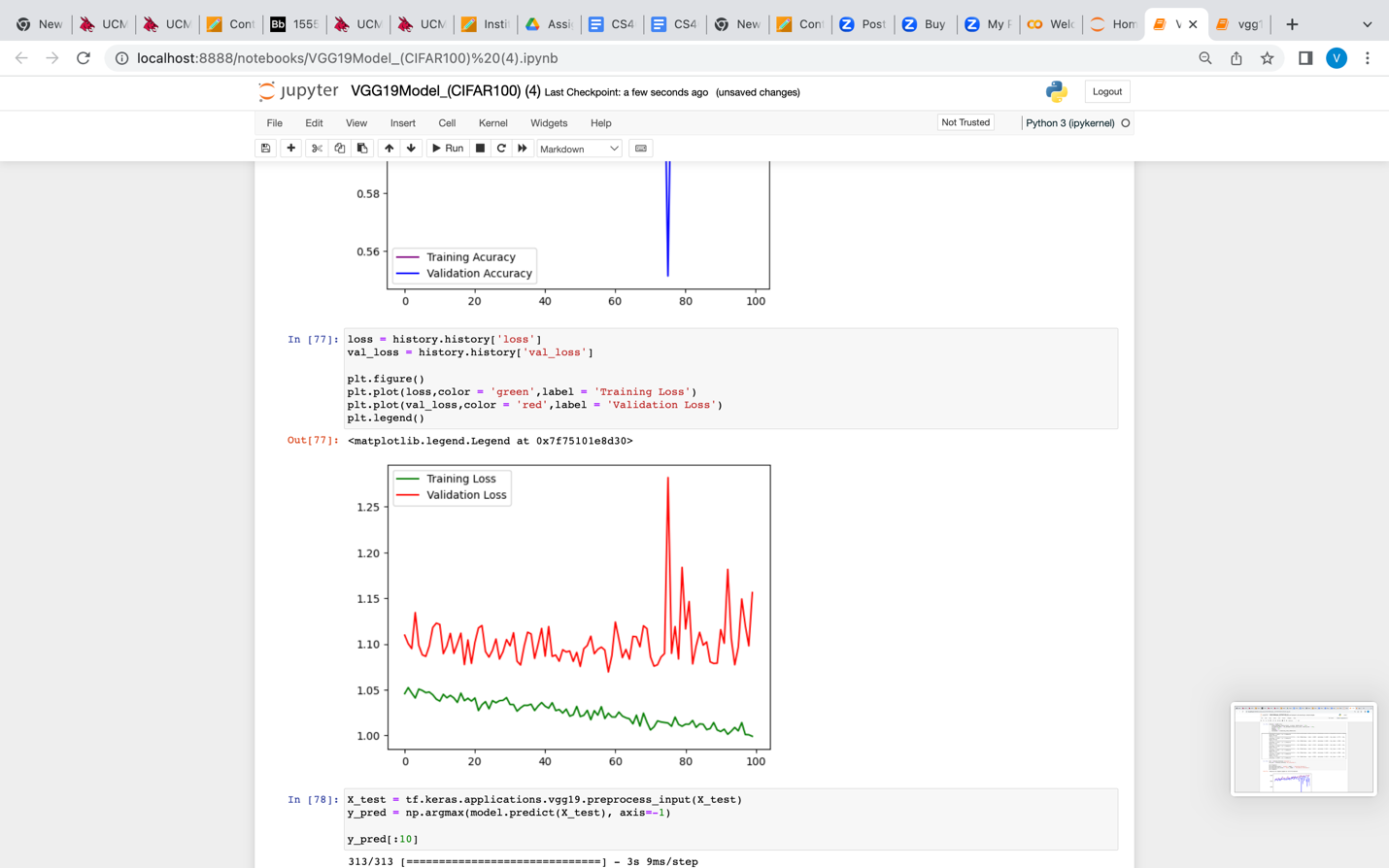
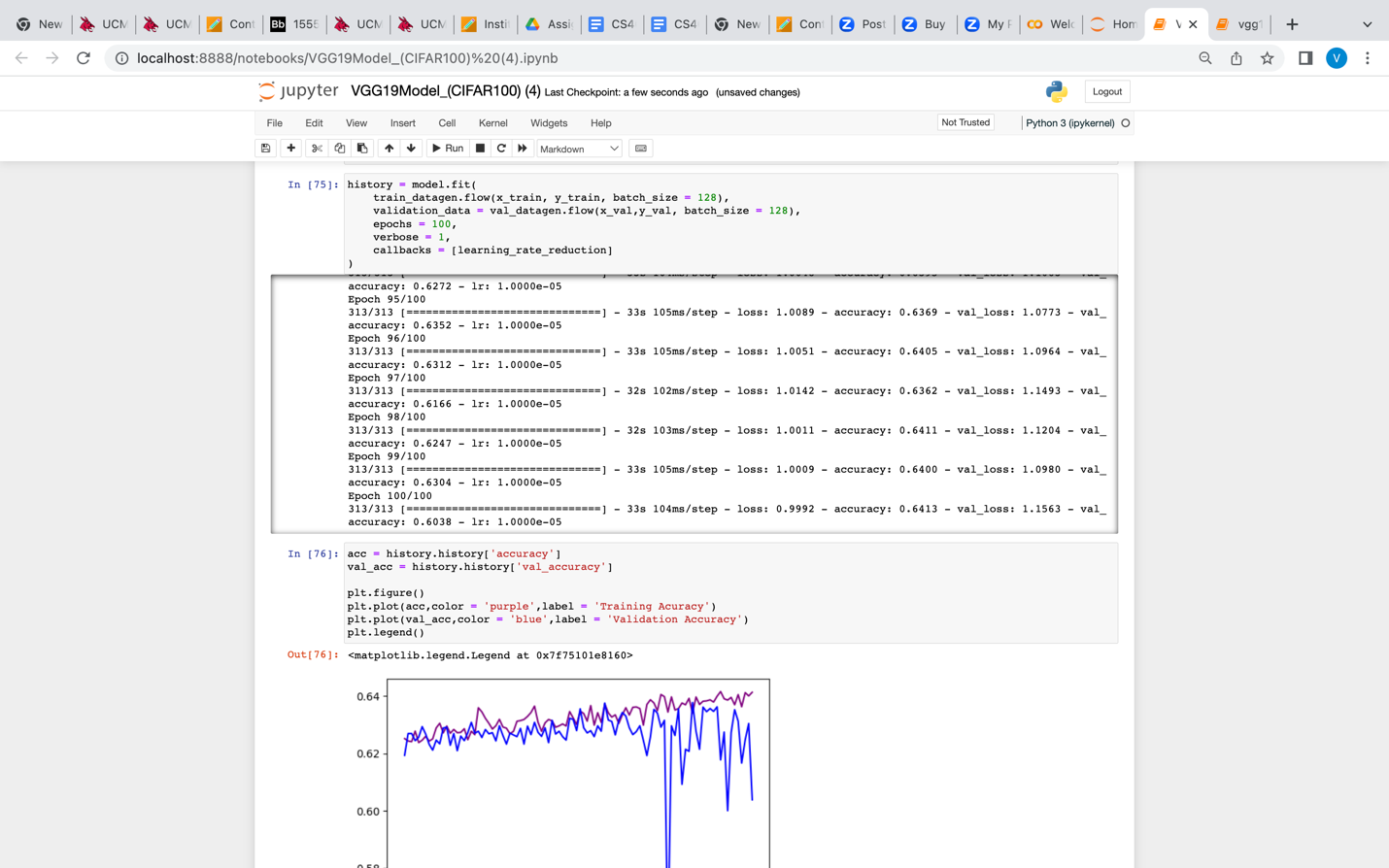
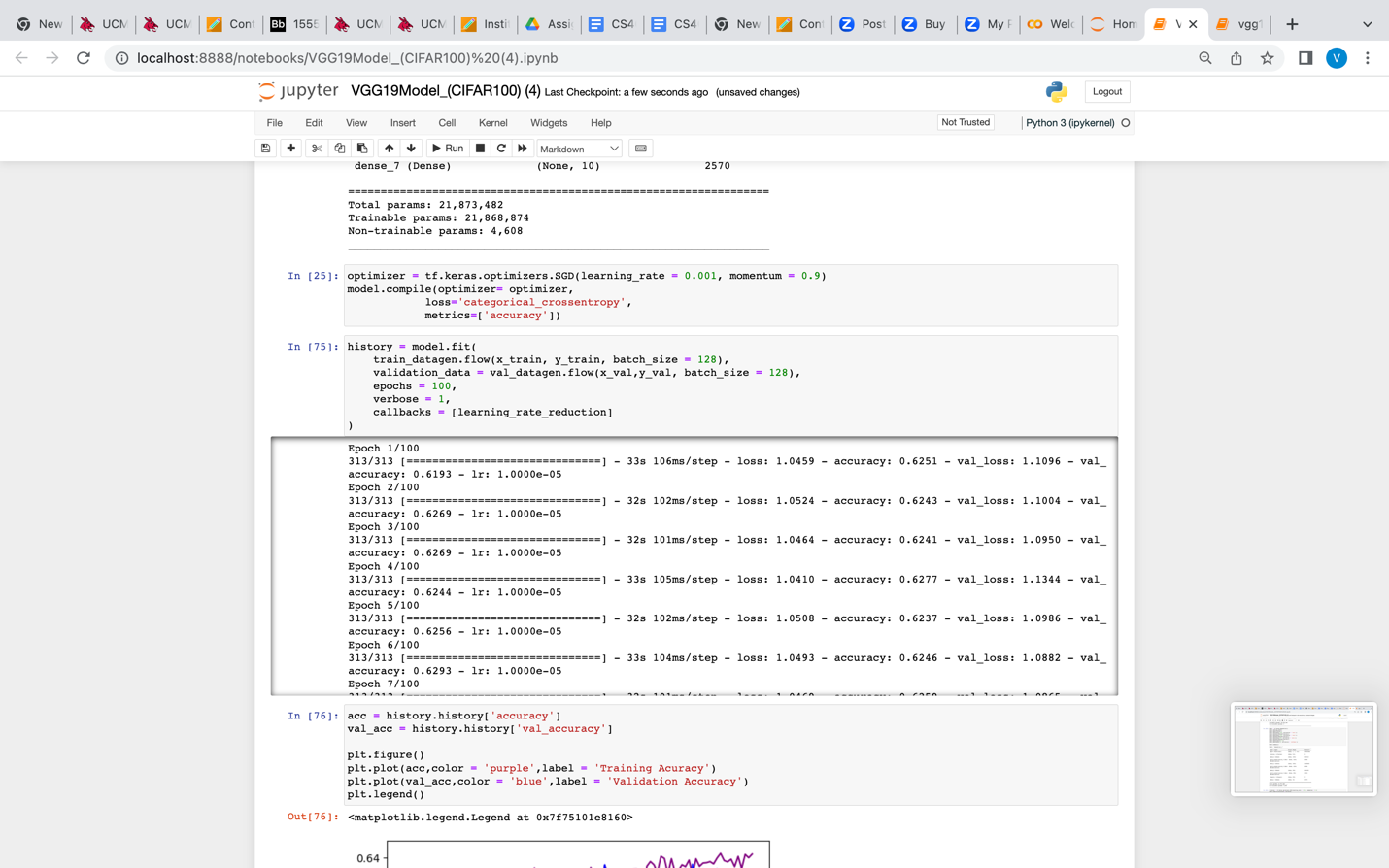
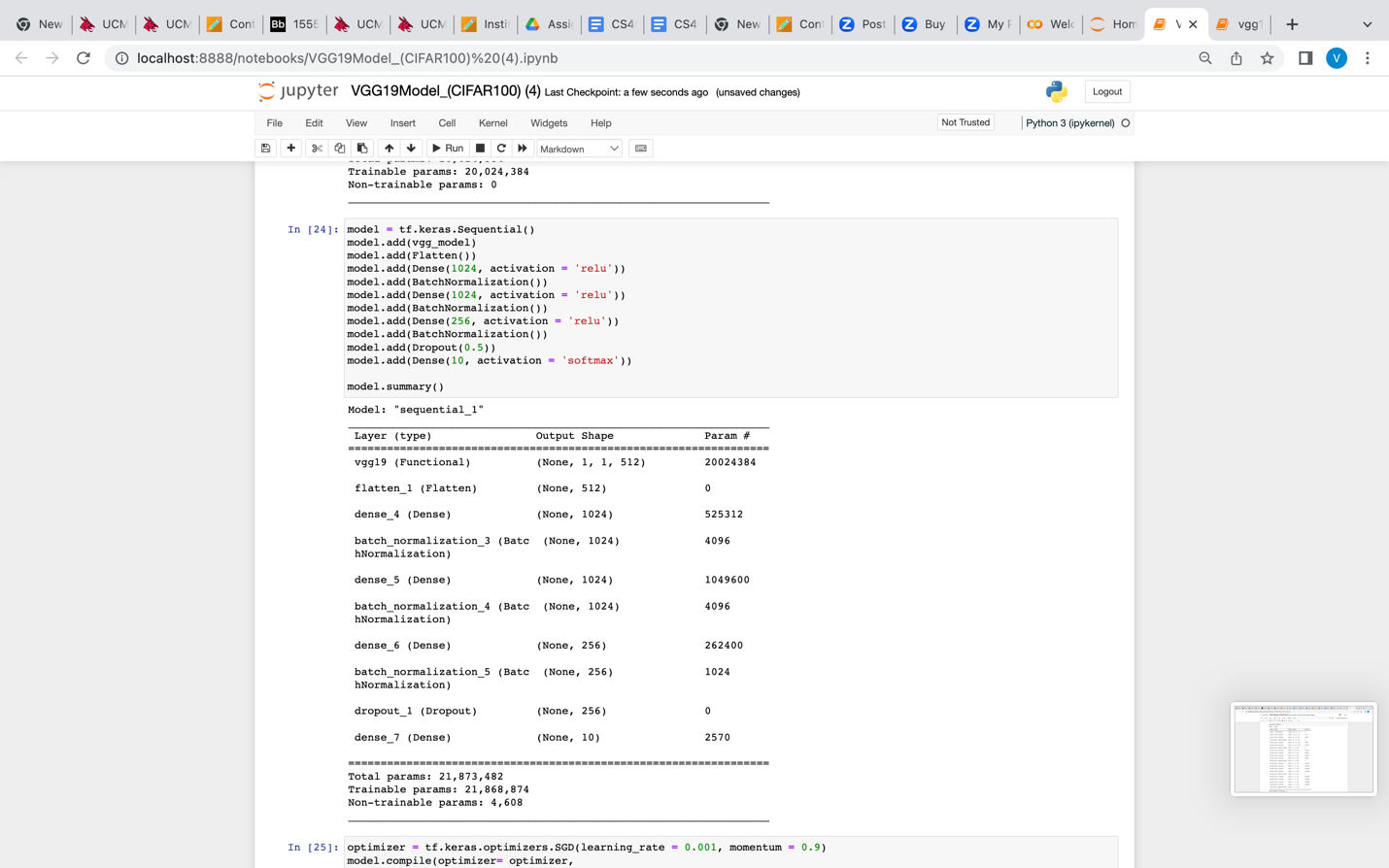
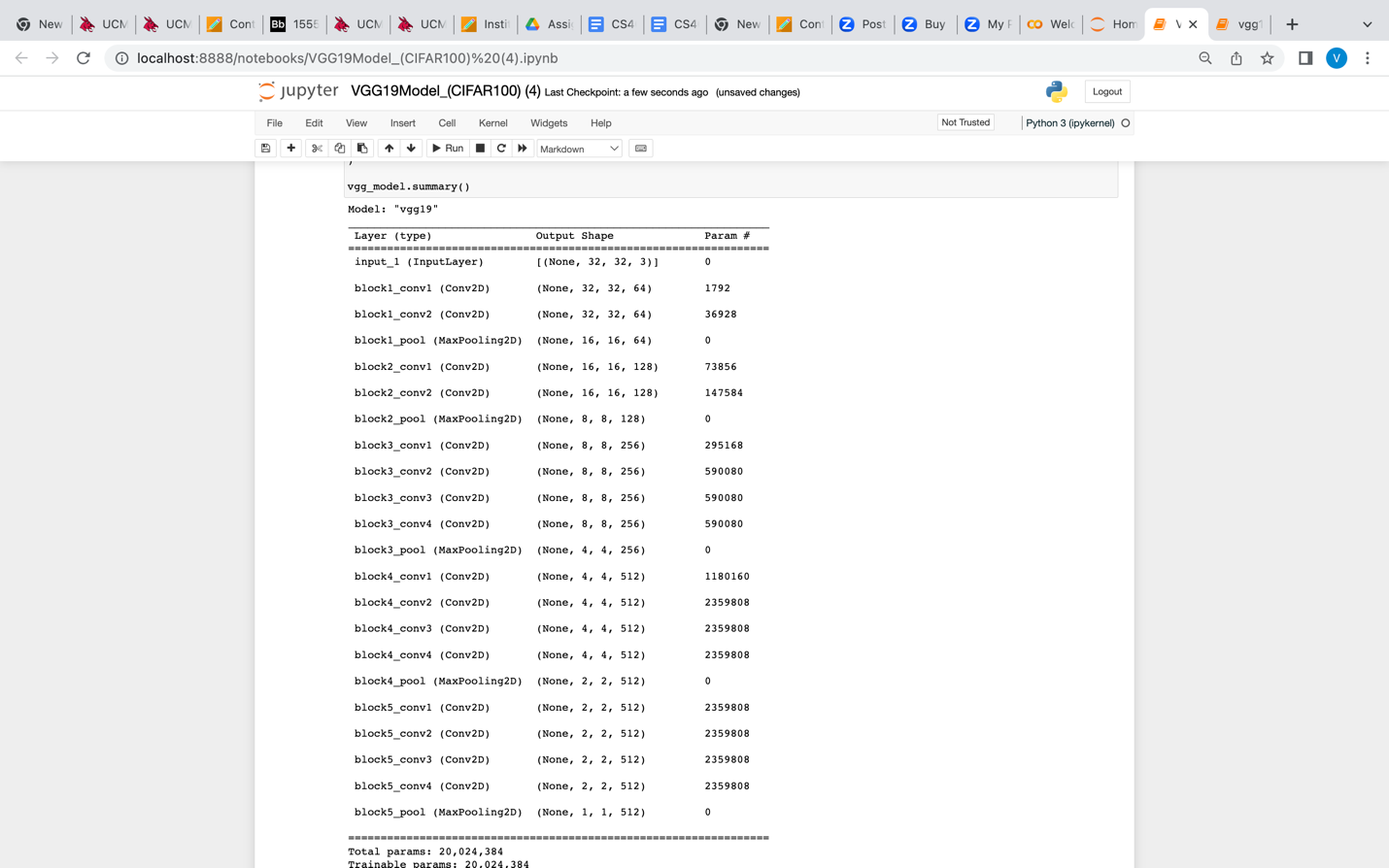
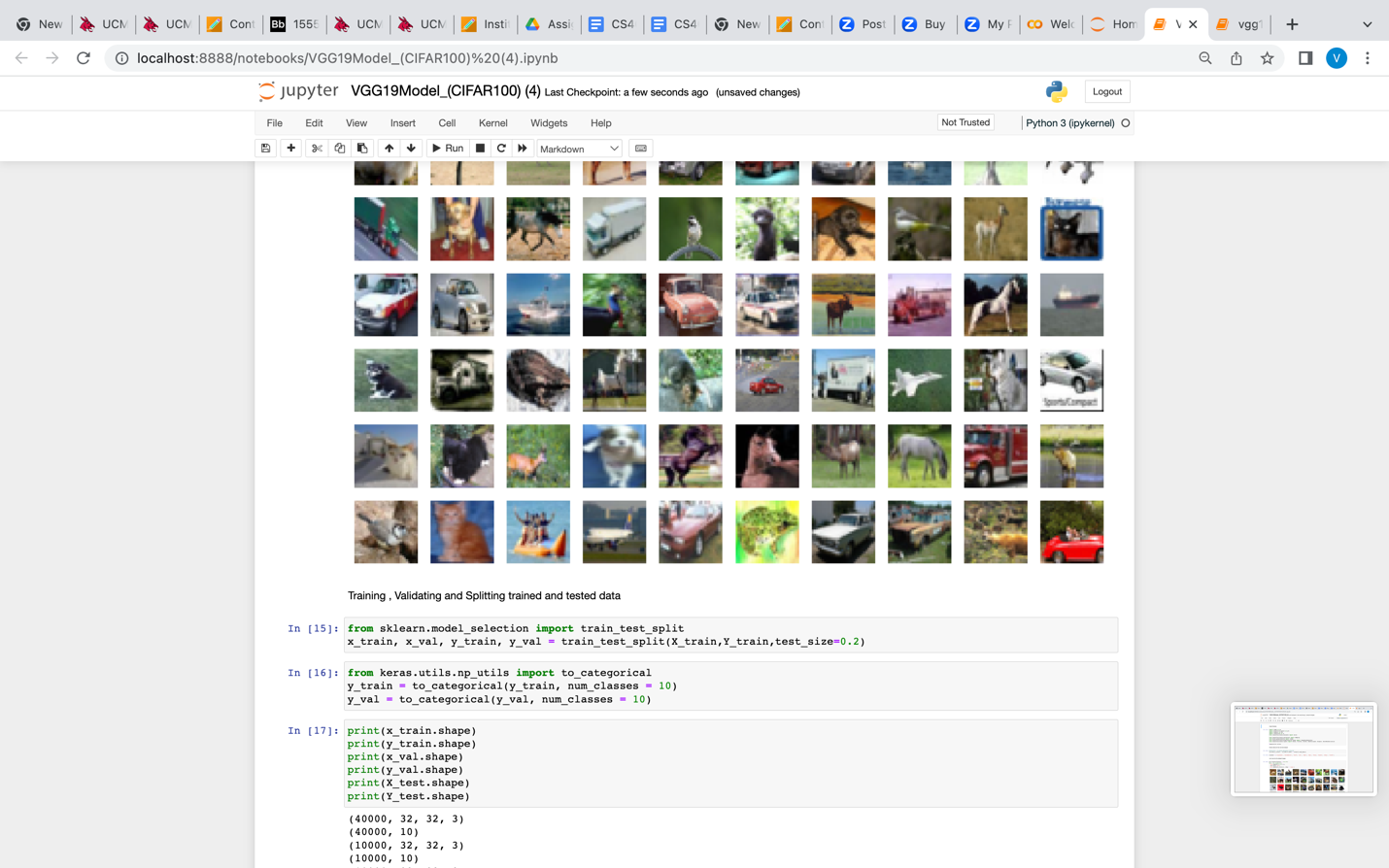
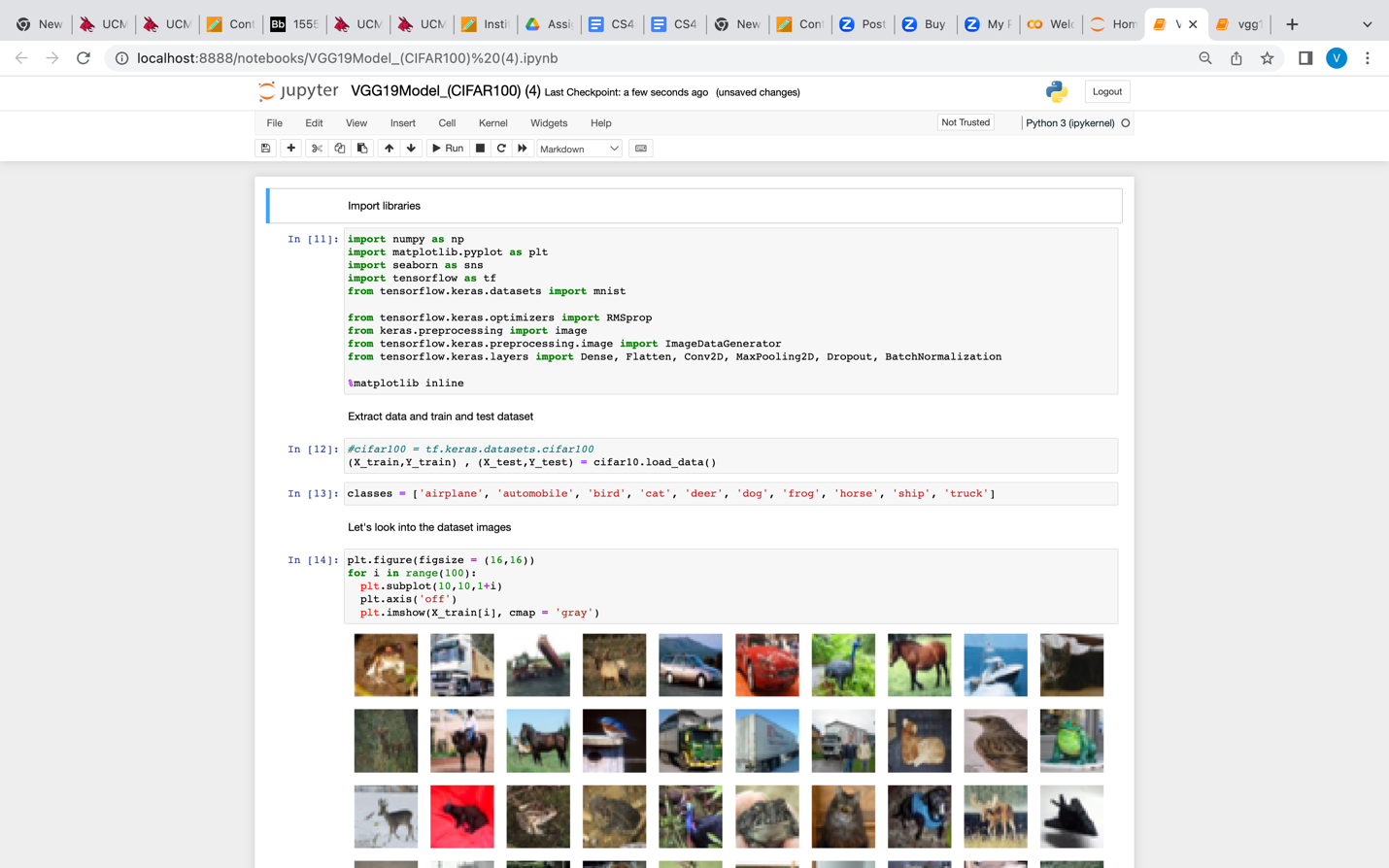
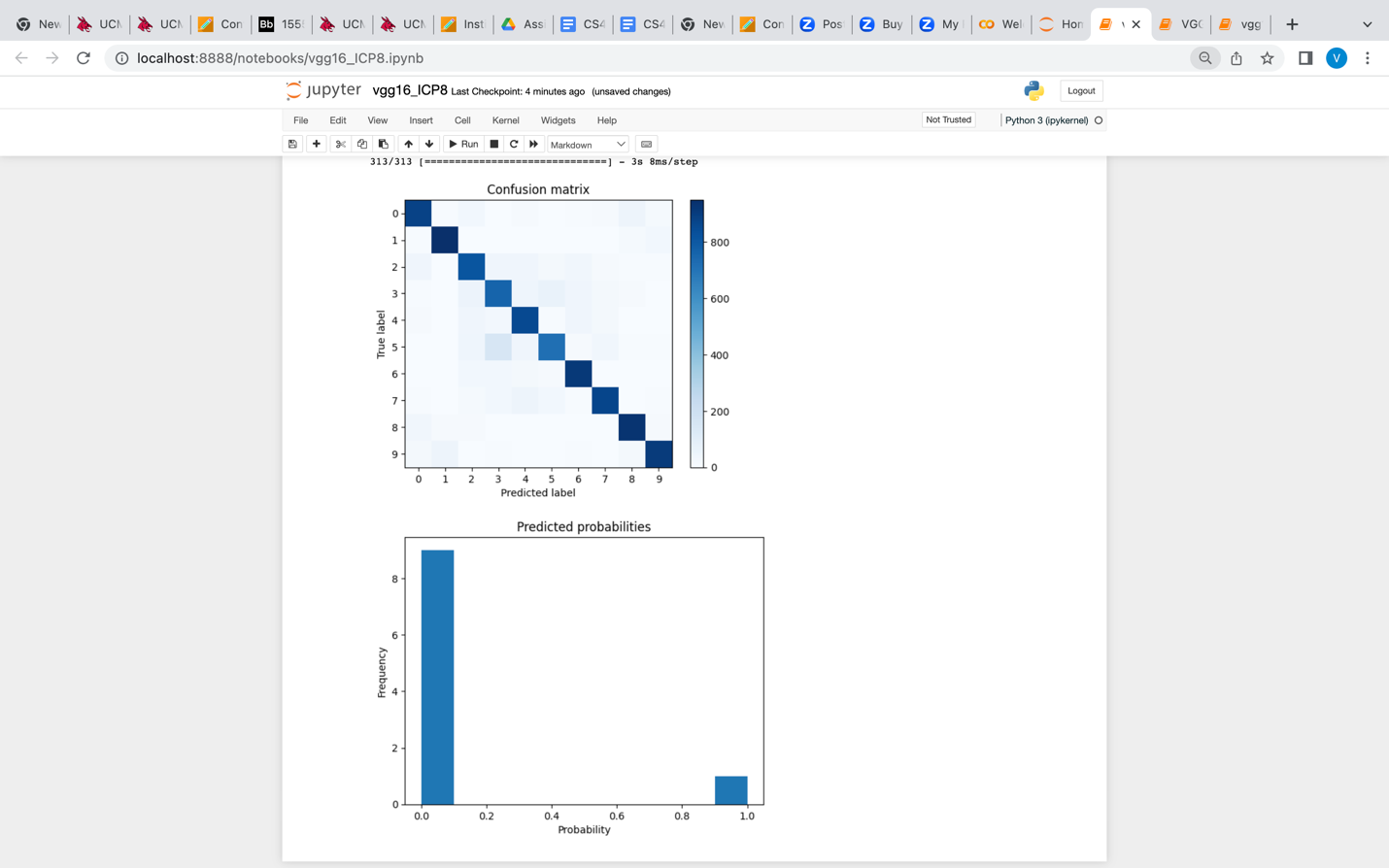
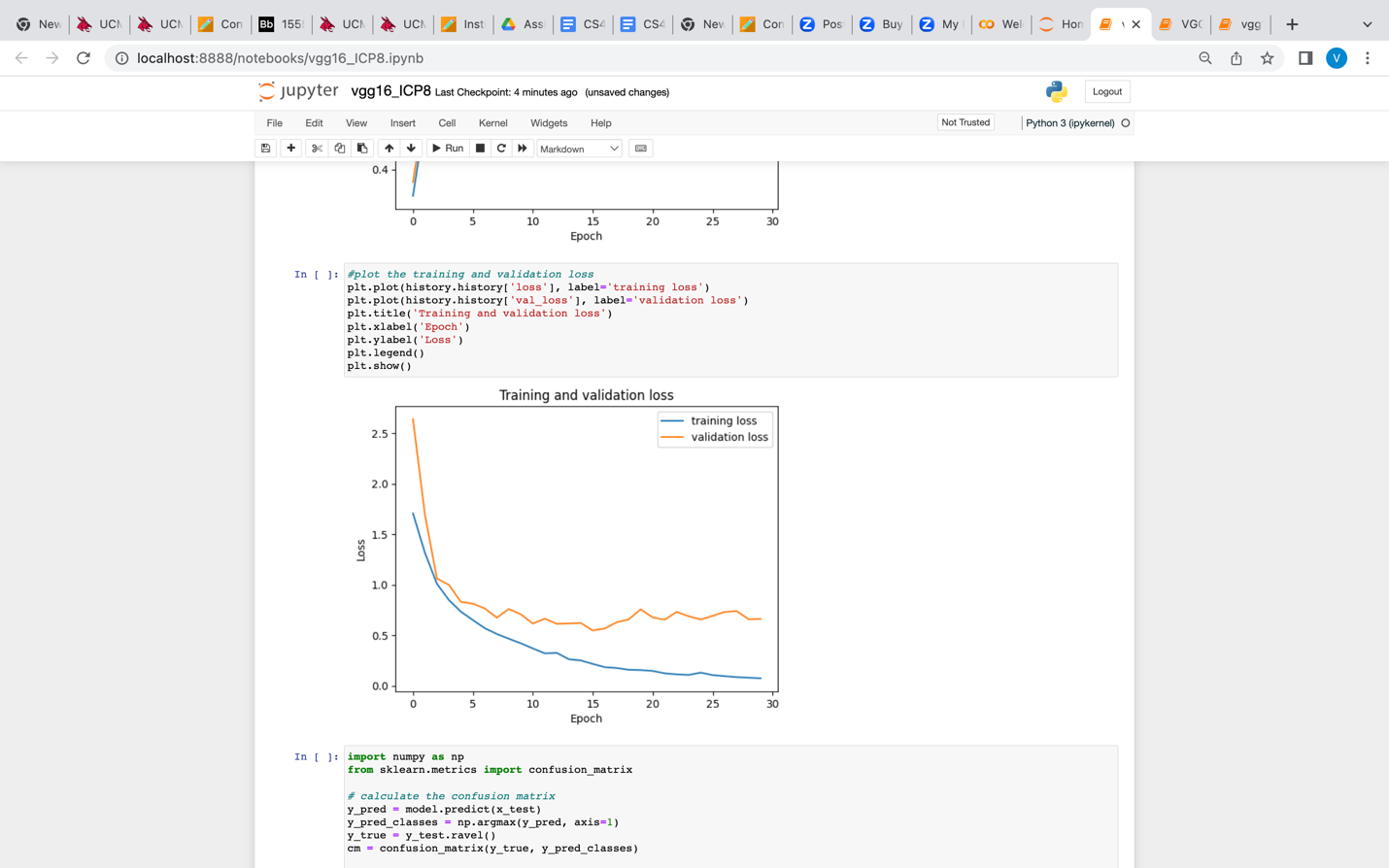
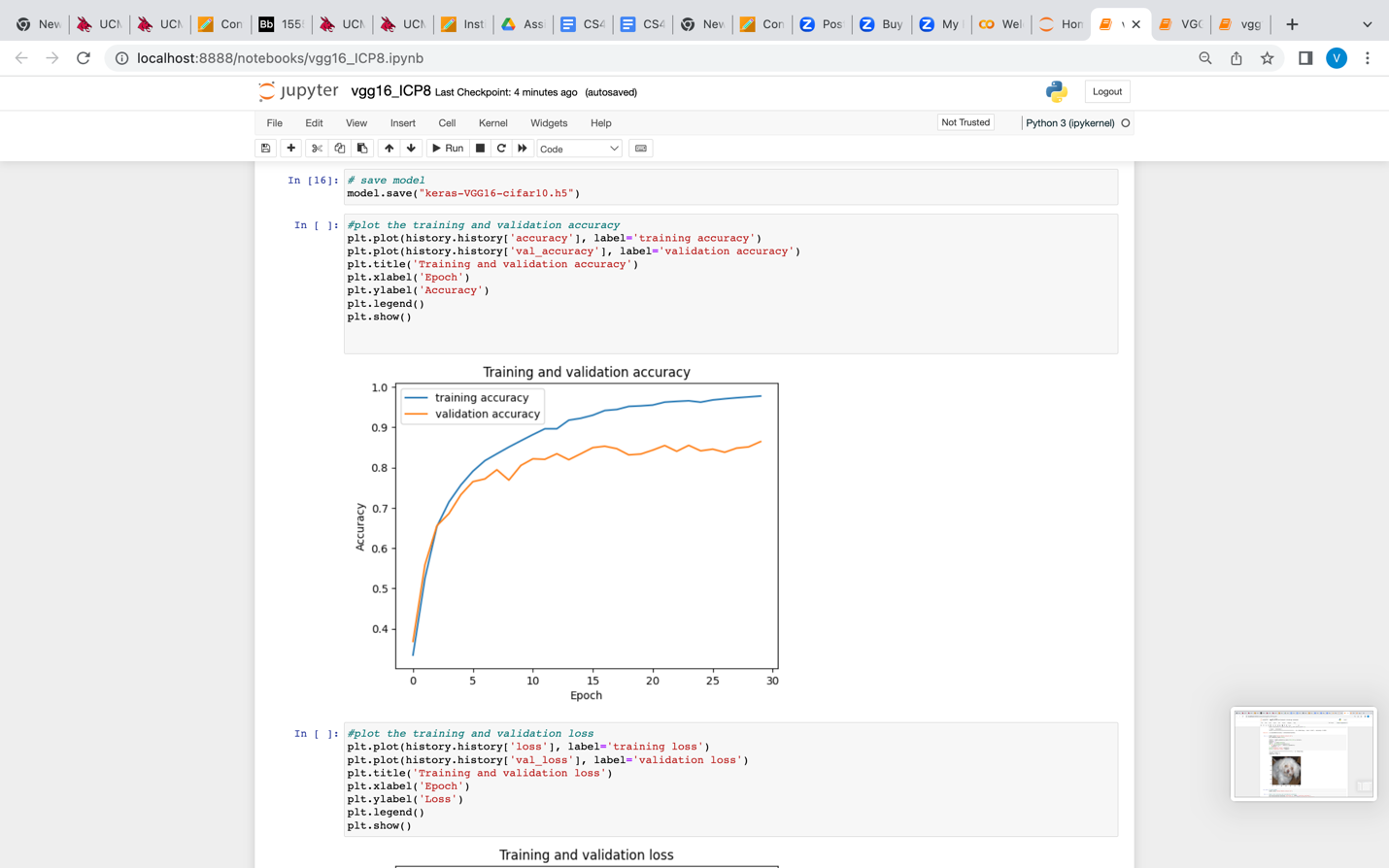
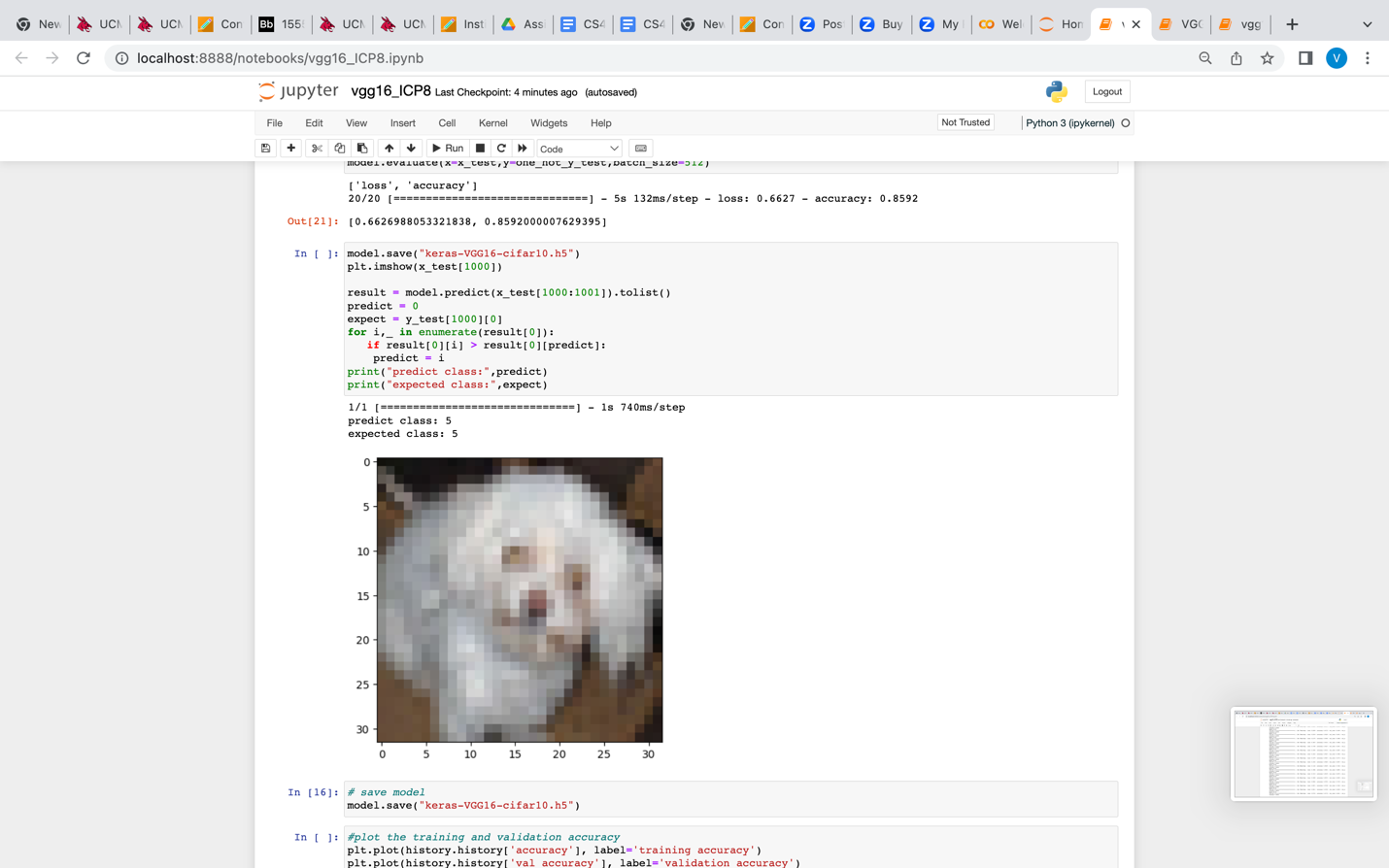
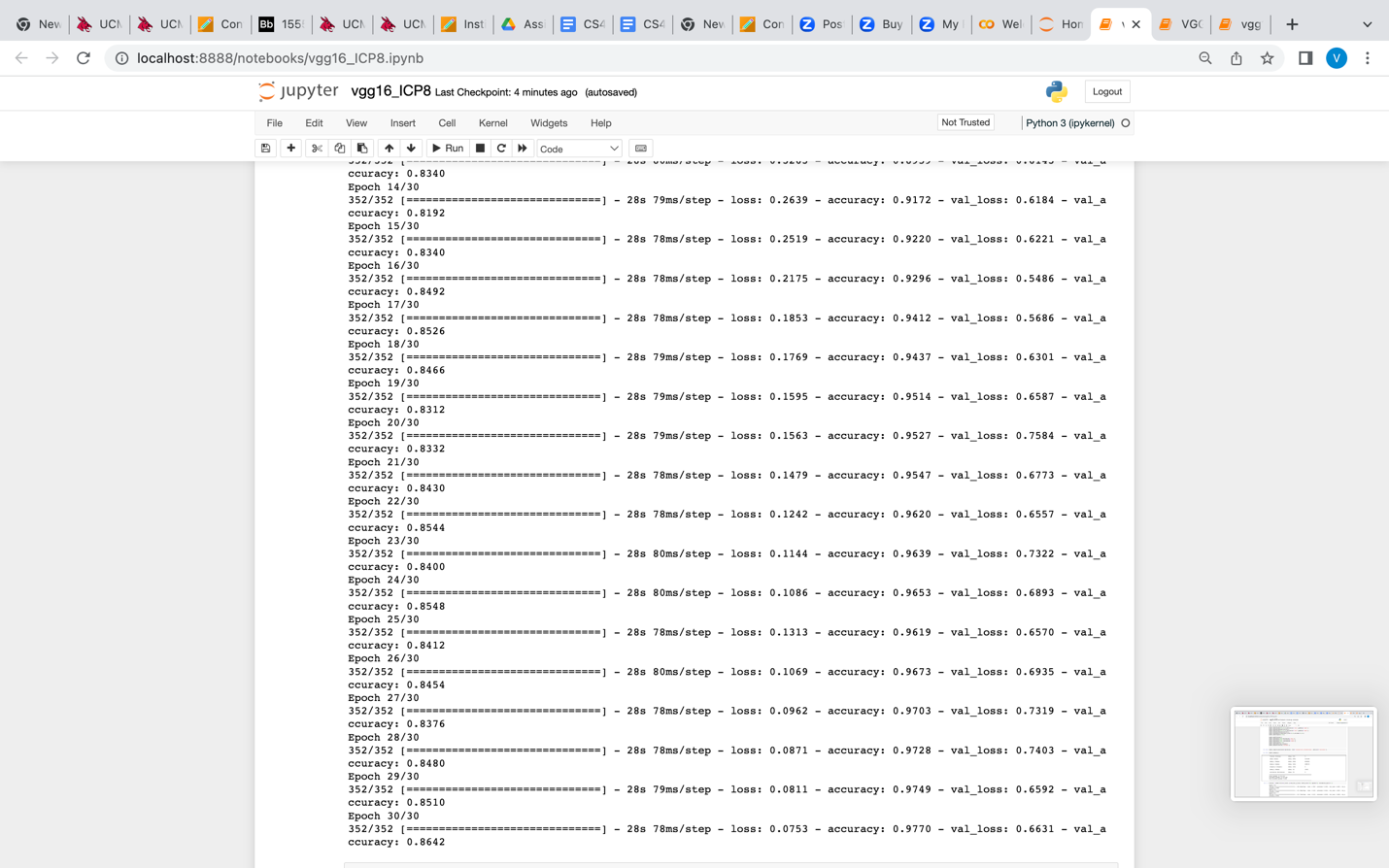
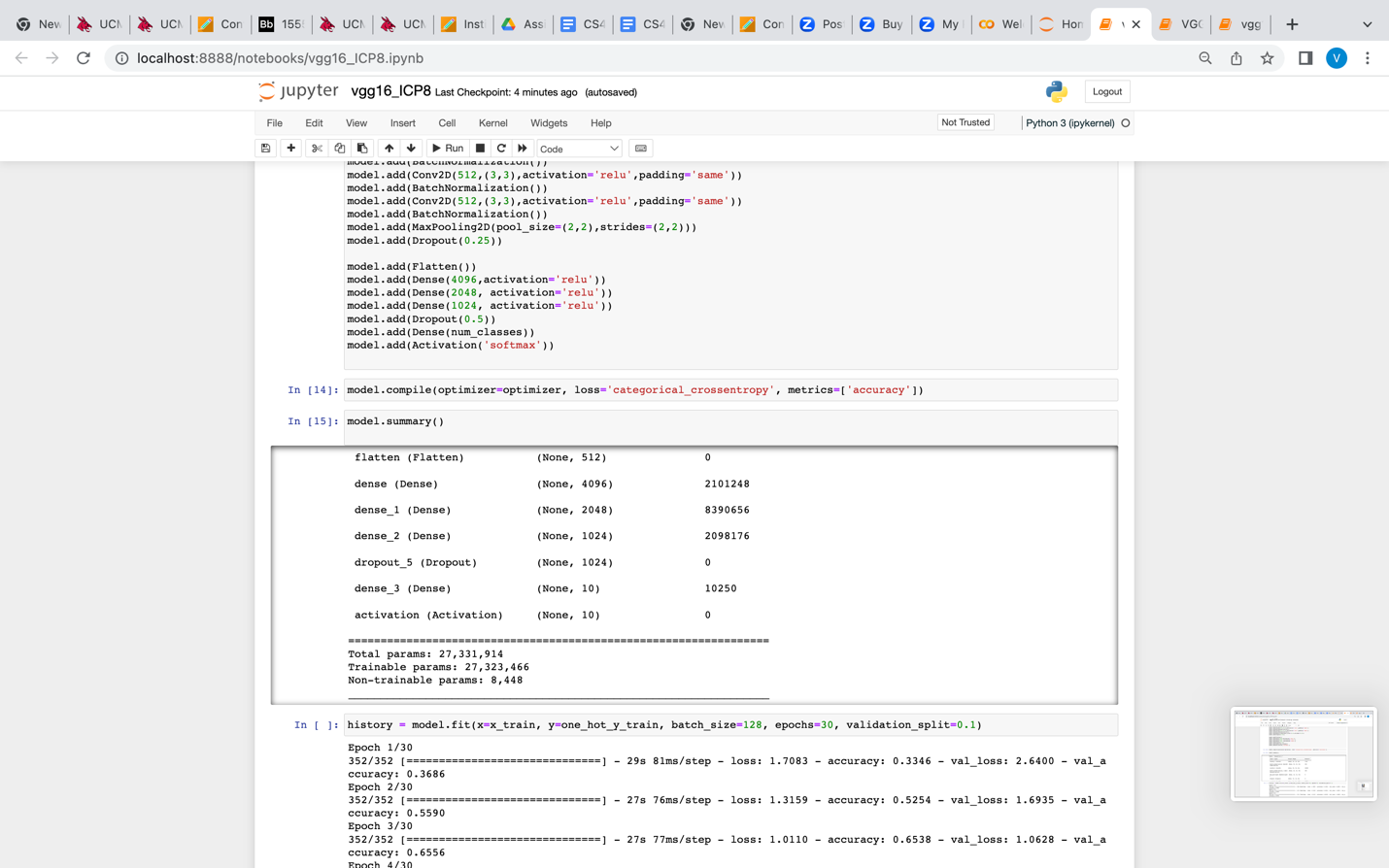
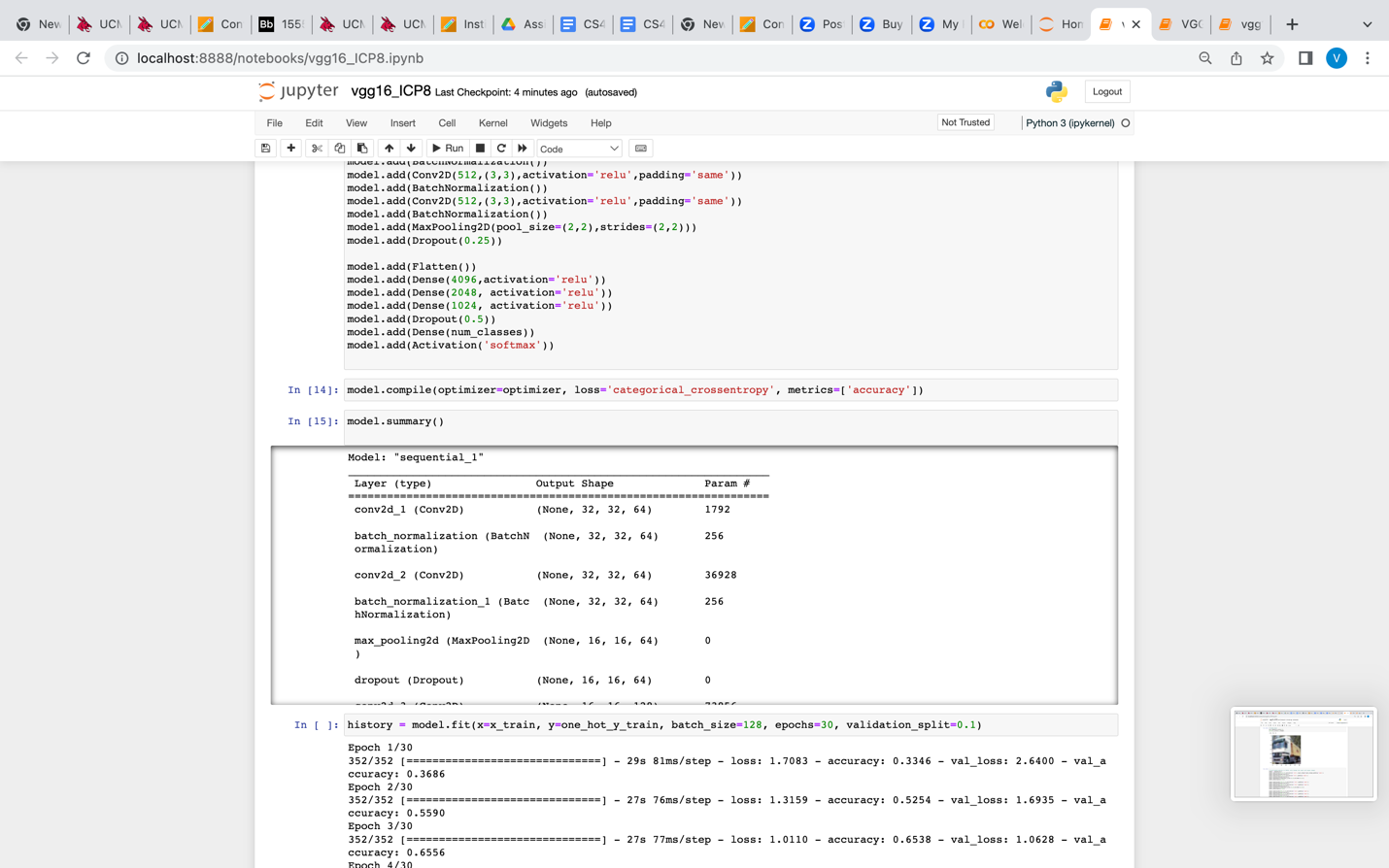
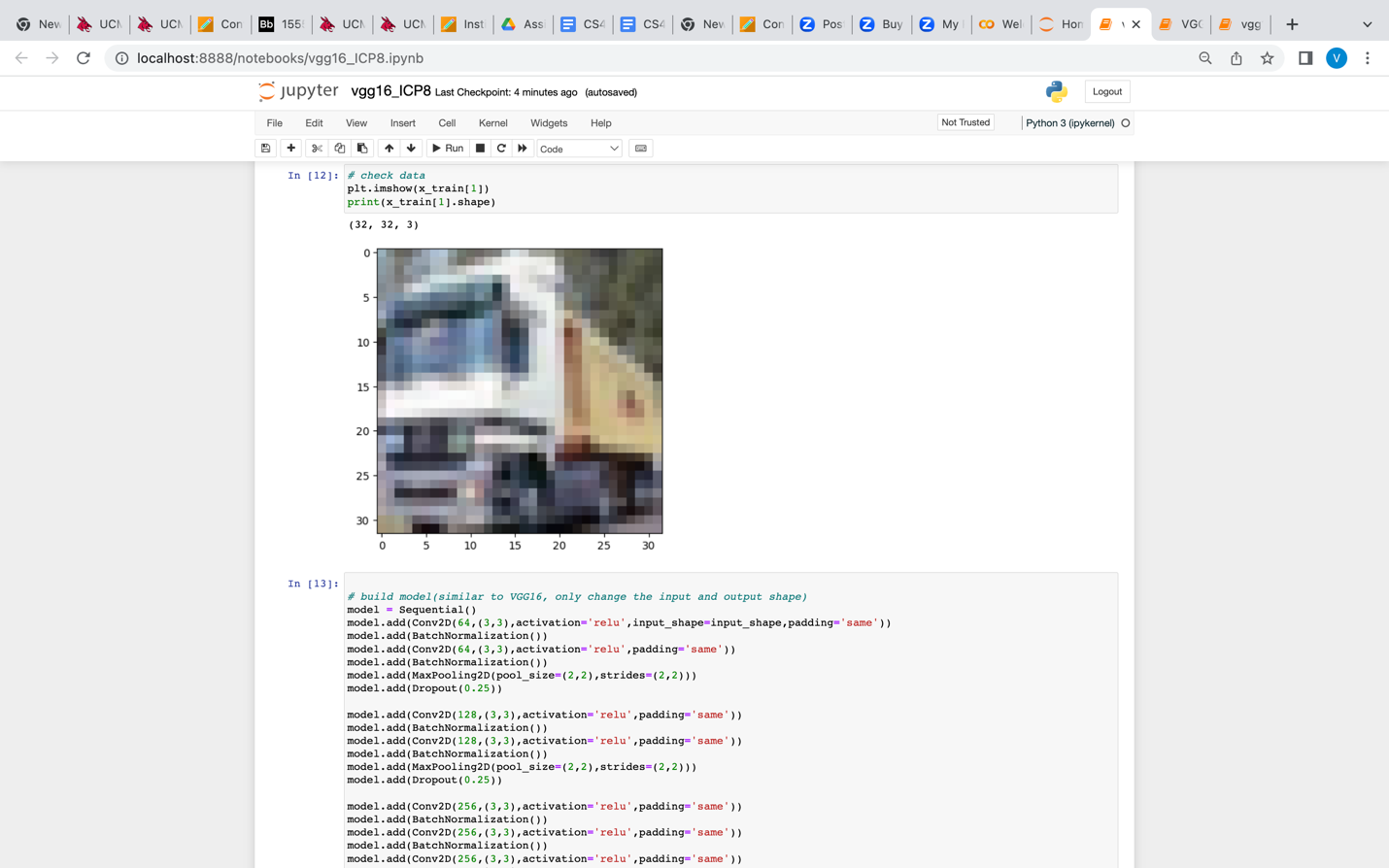
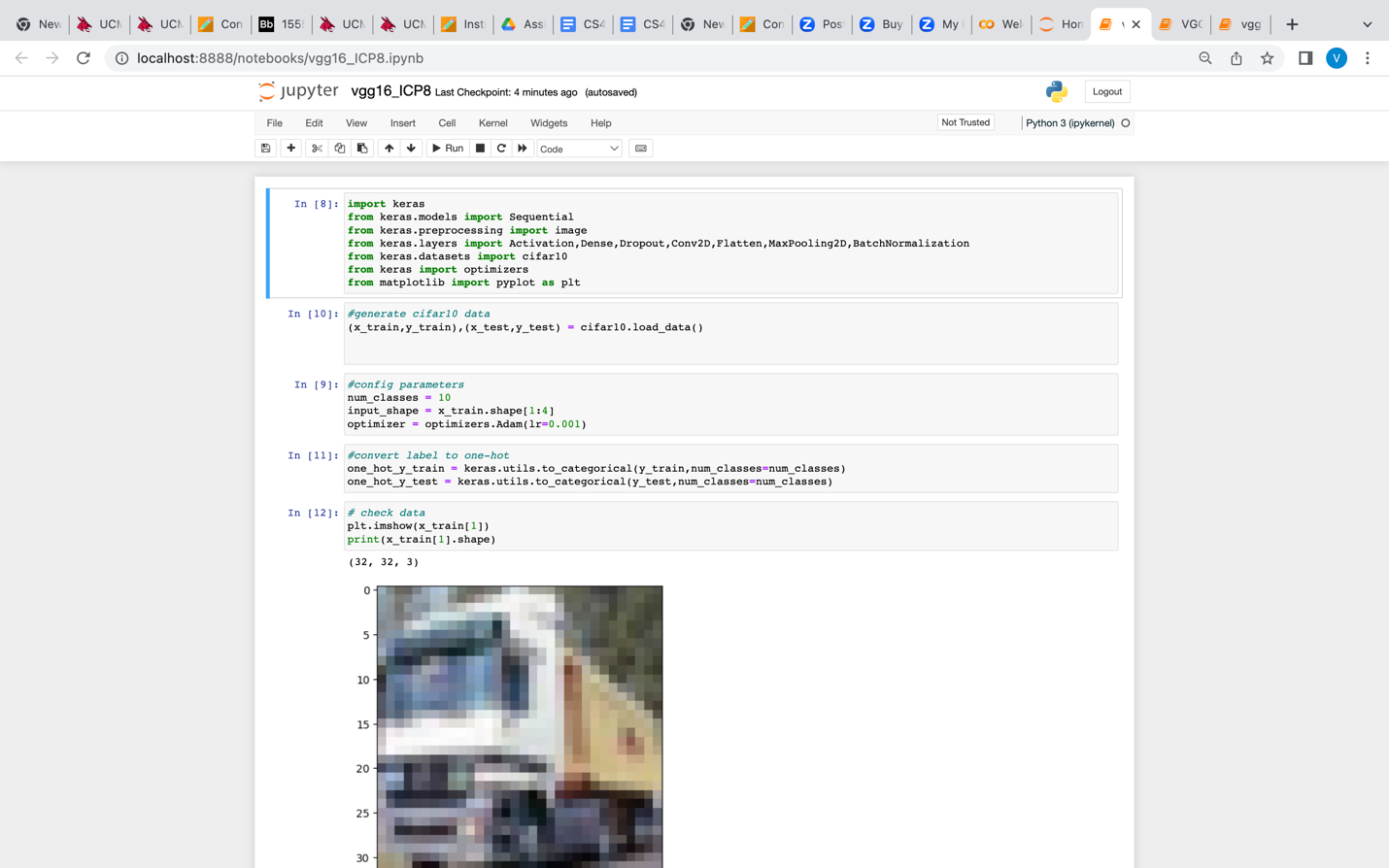
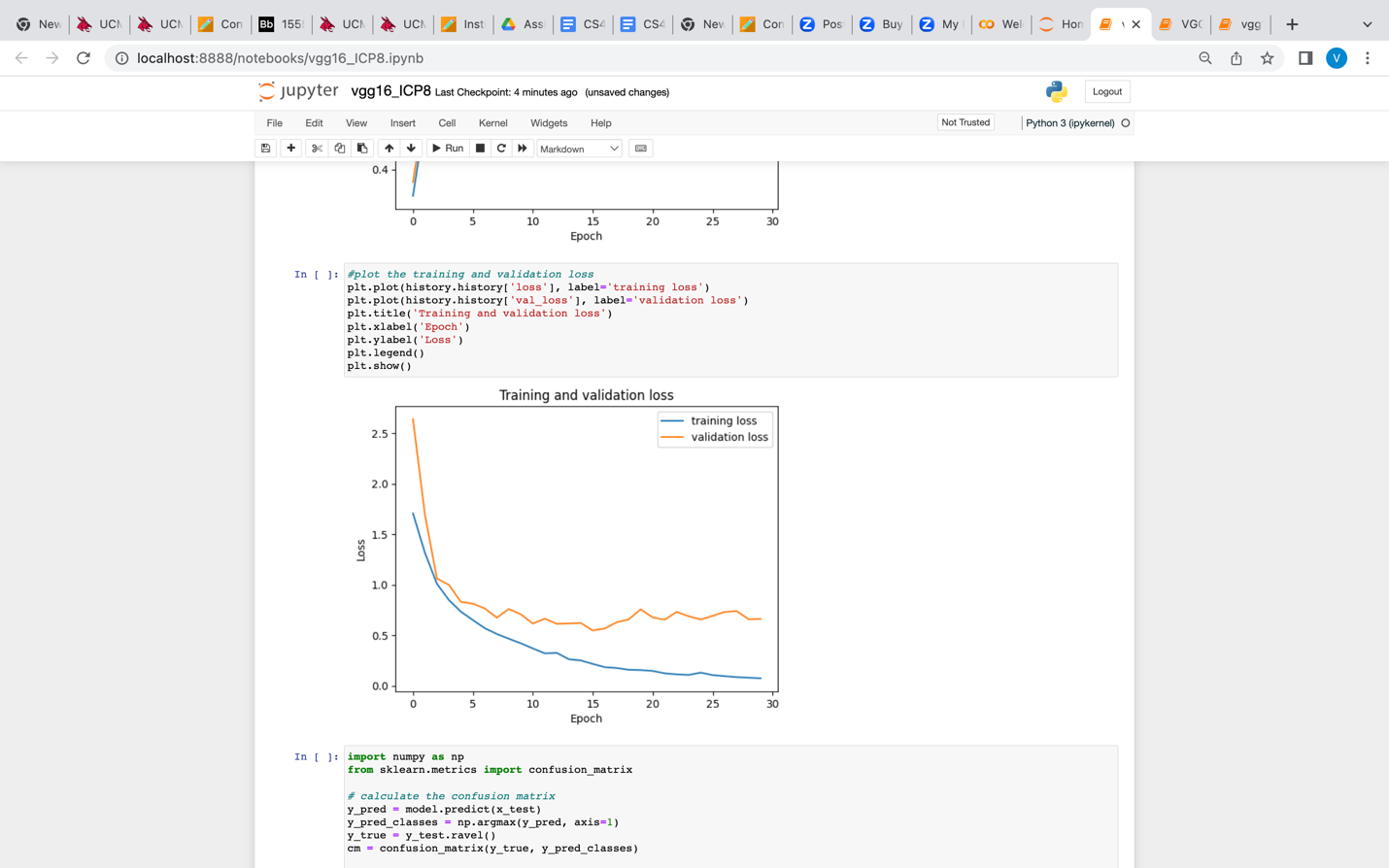
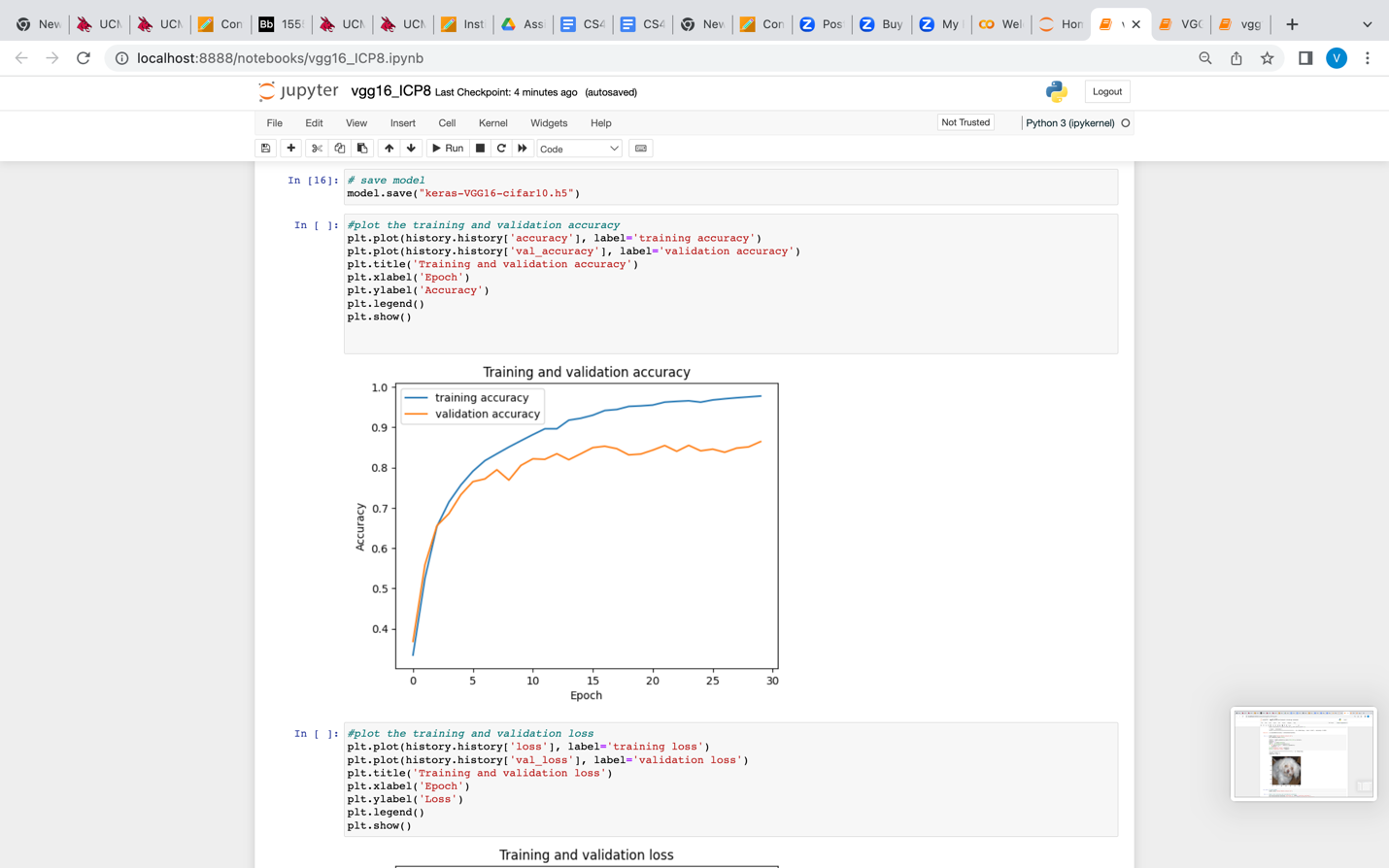
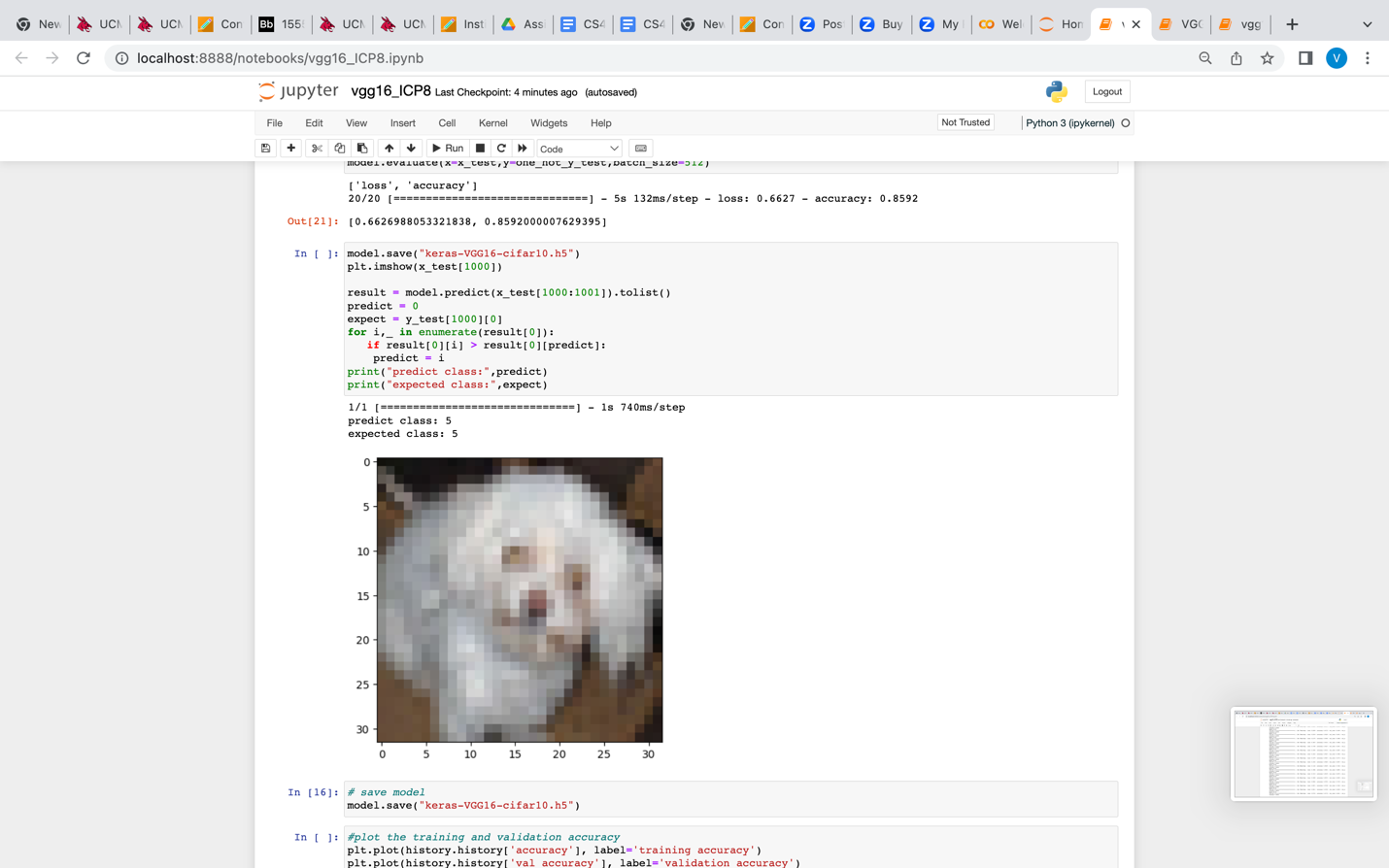
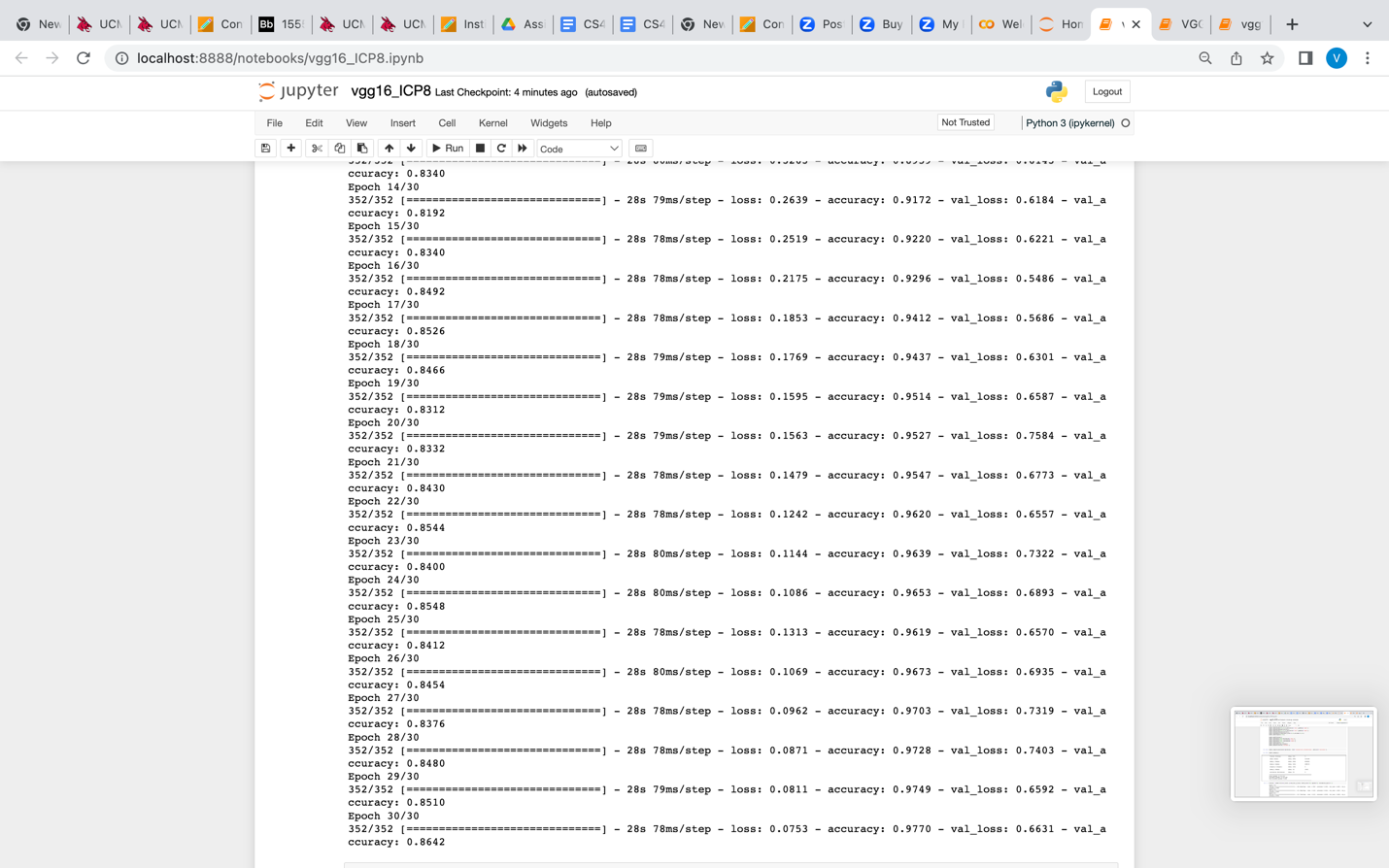
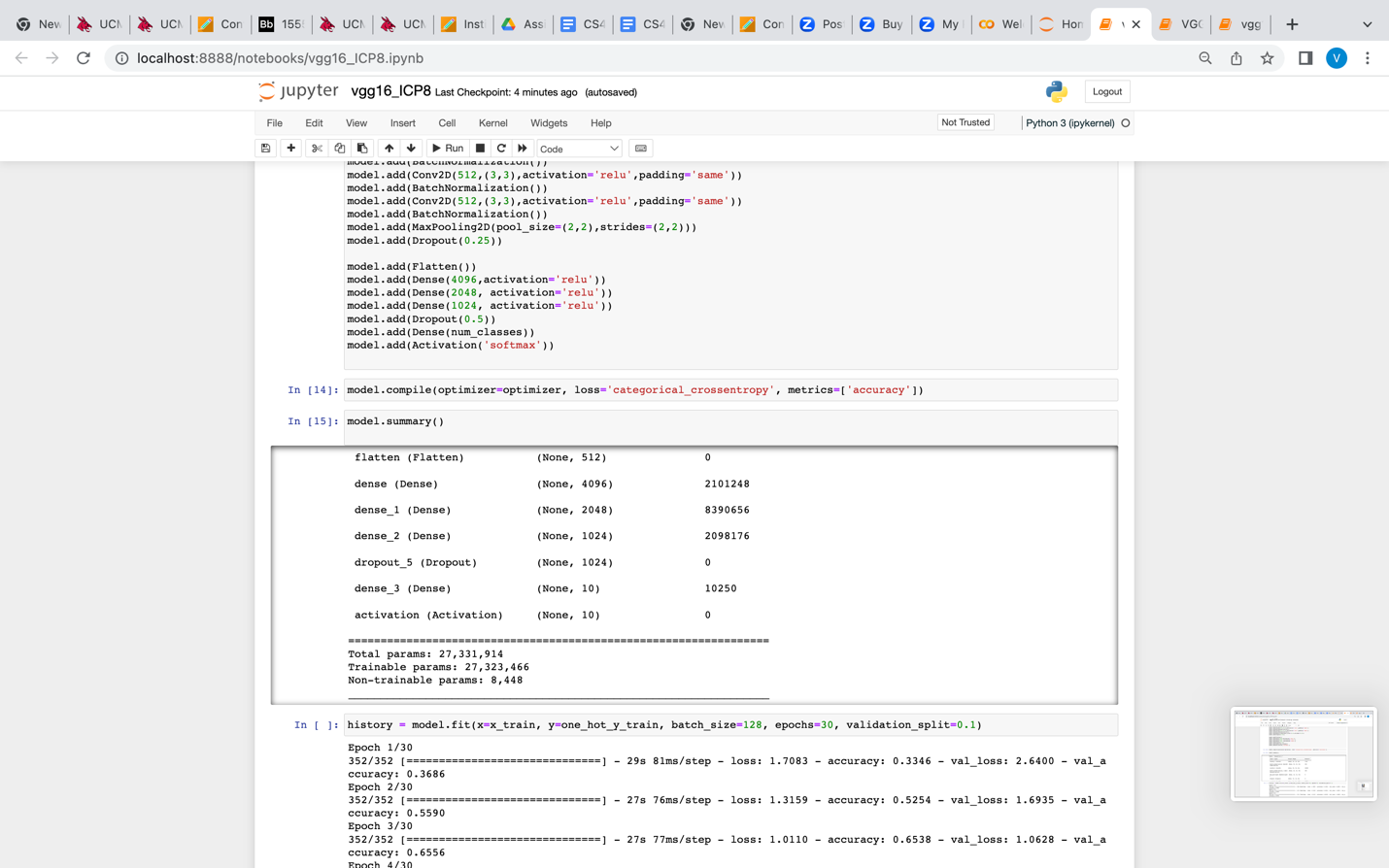
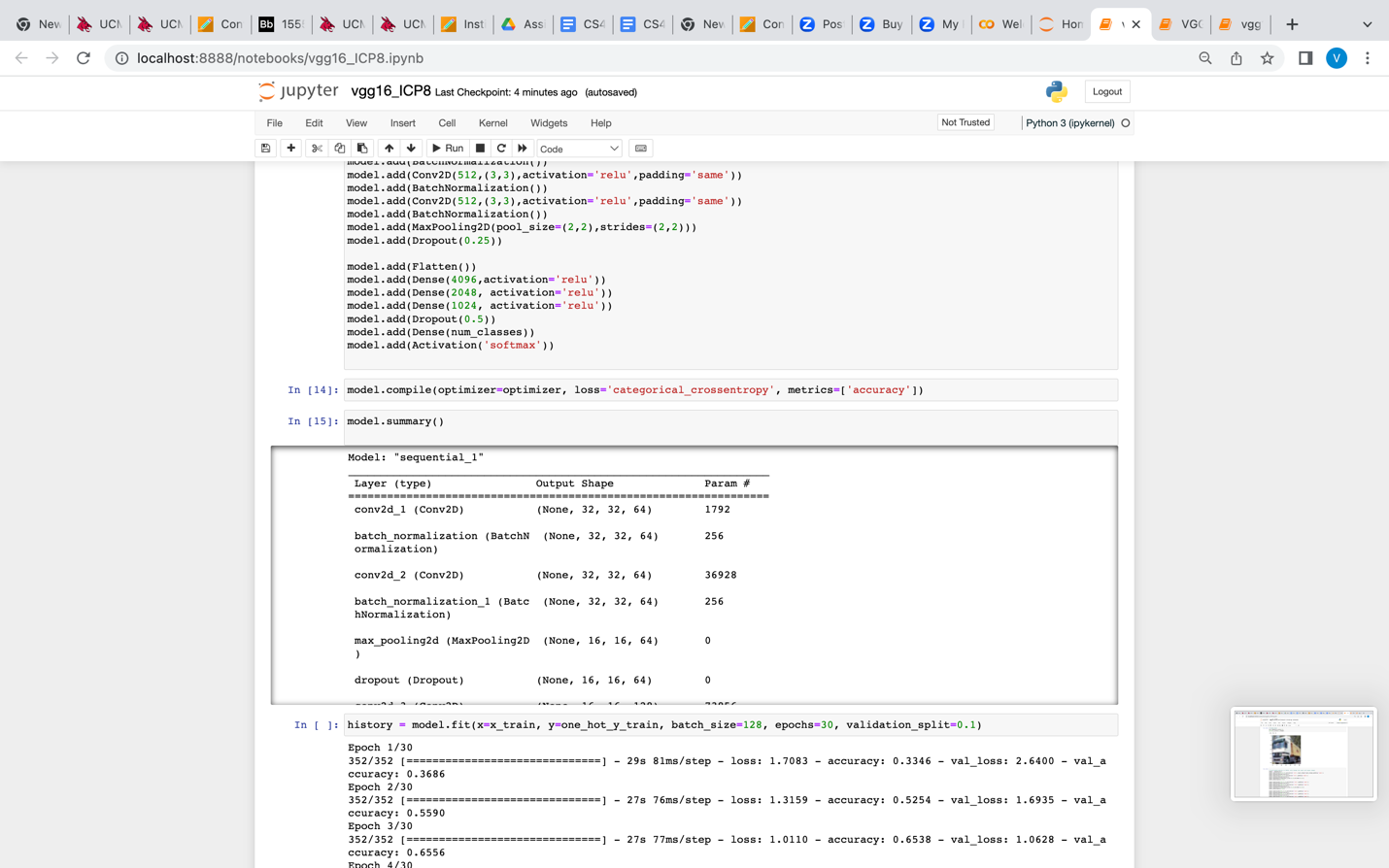
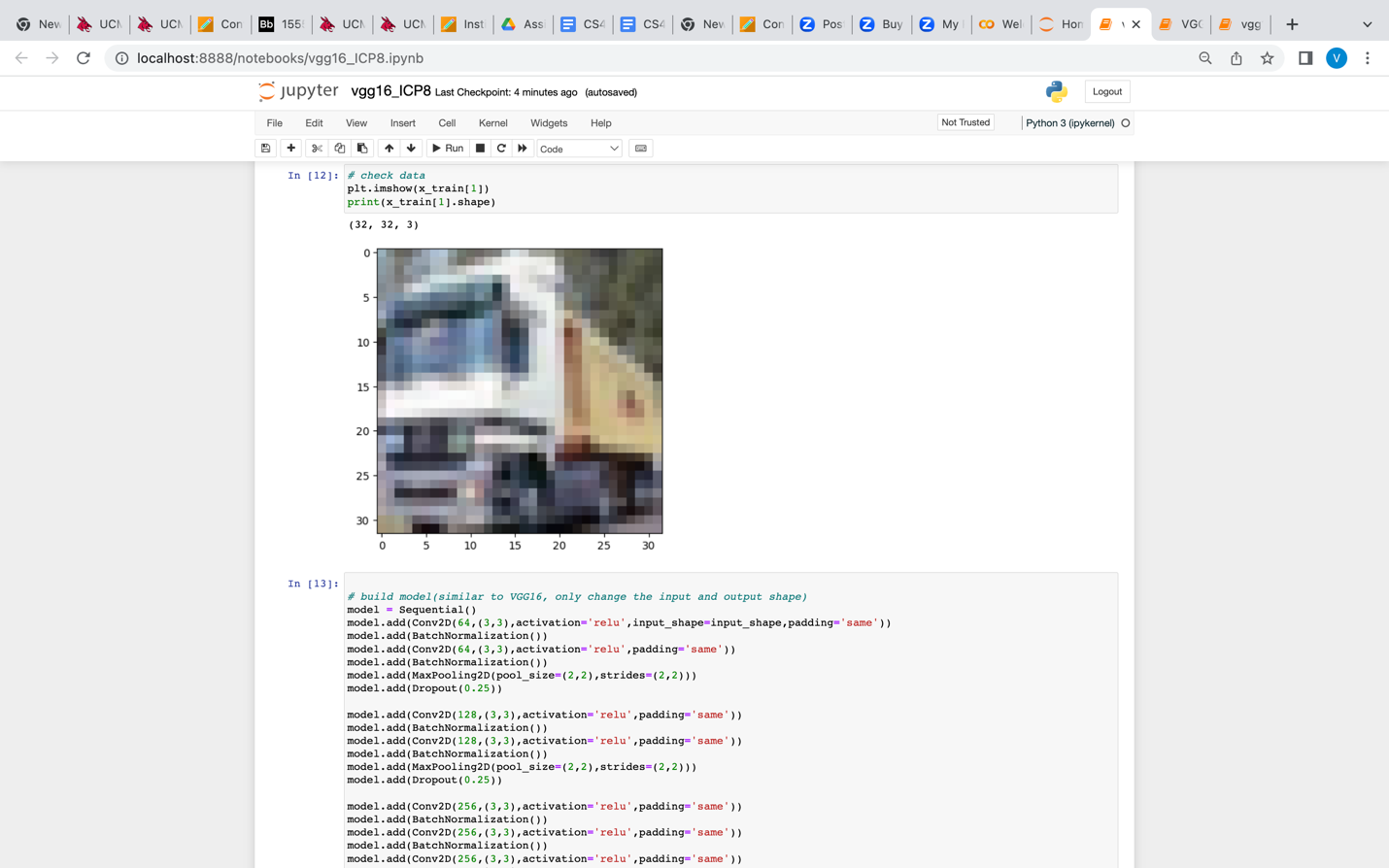
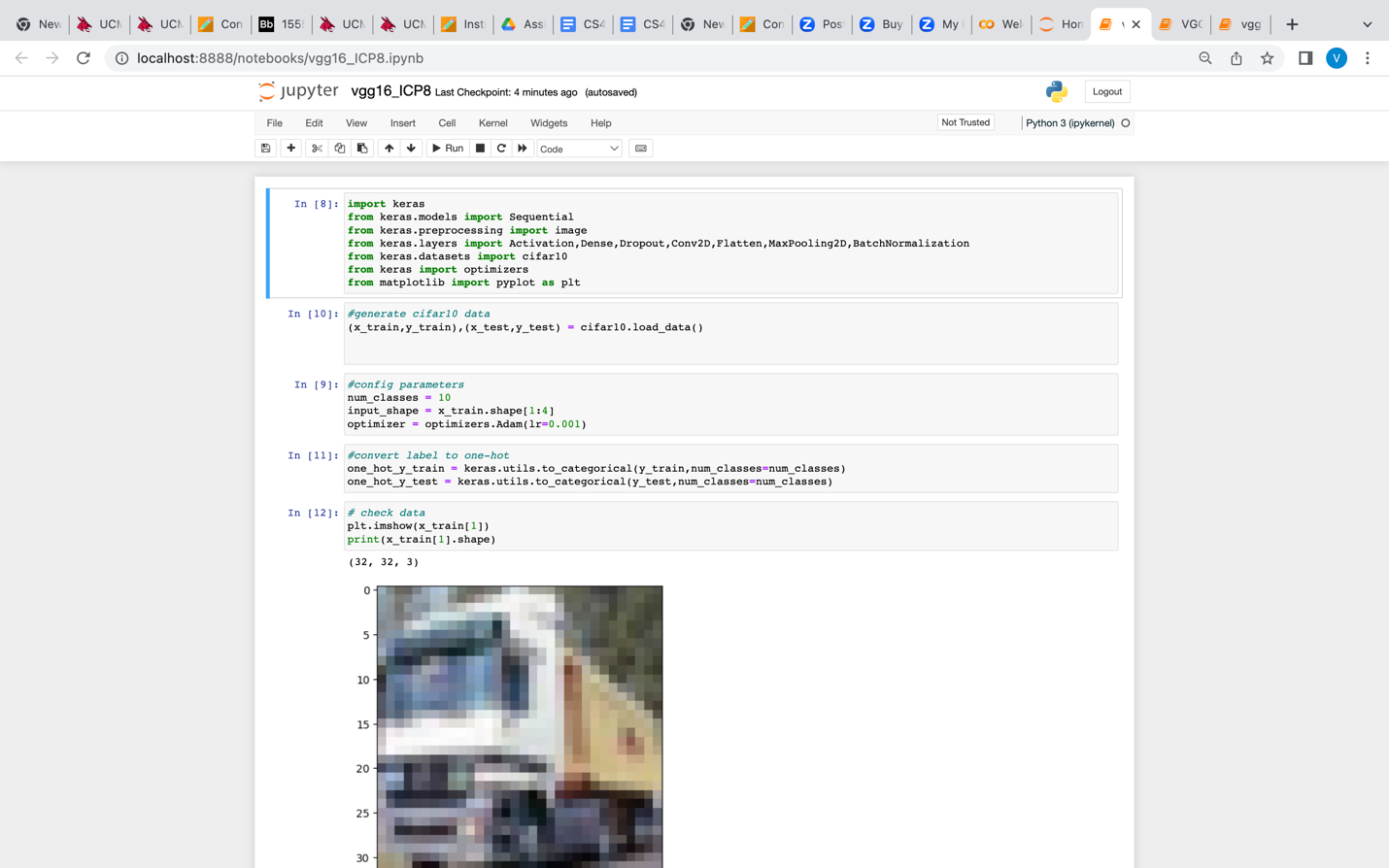
and provided logical description of steps that lead to improved response and its impact on architecture behavior and, created two more visualizations using matplotlib.

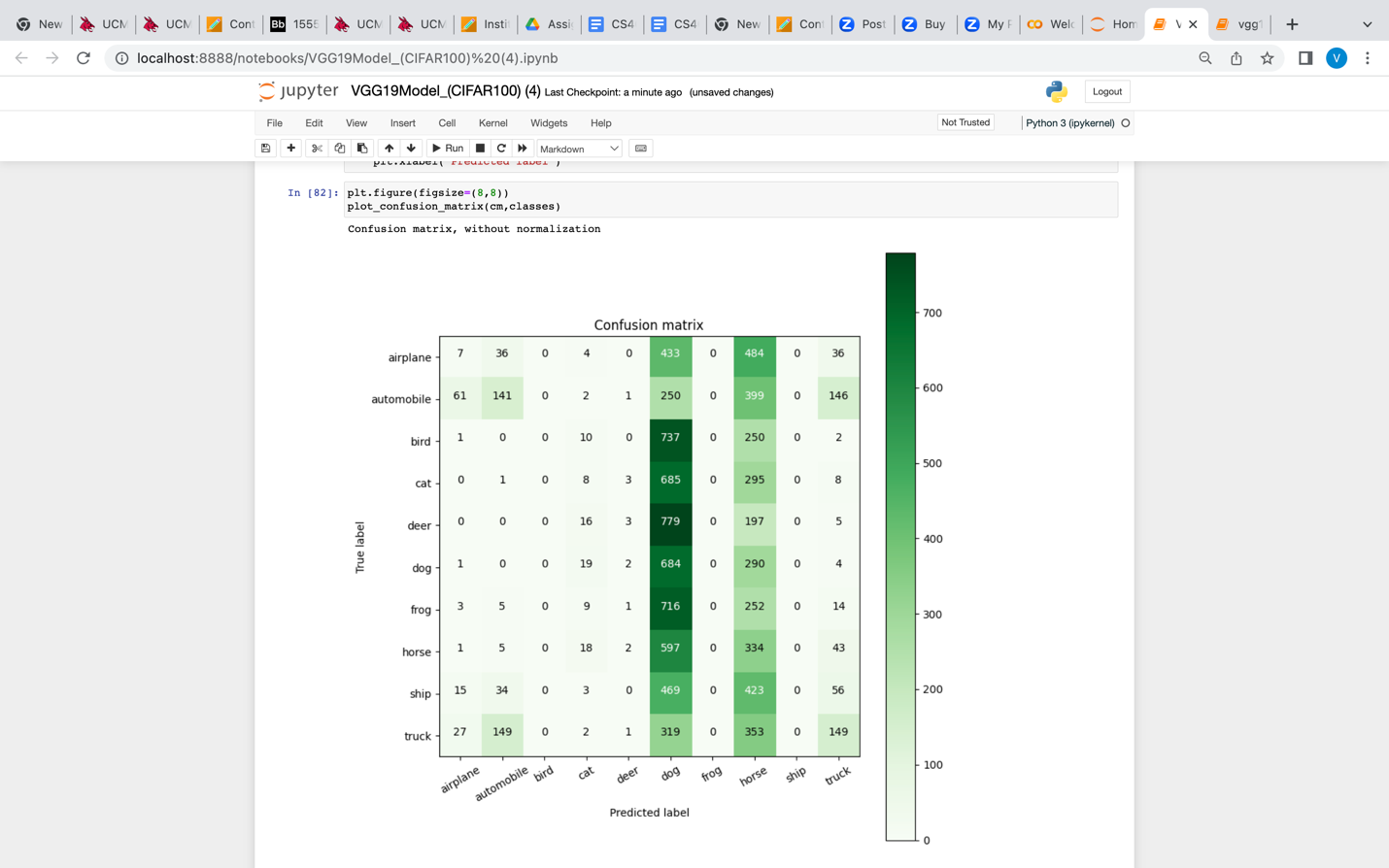
Used dataset and implemented baseline models. Applied modified architecture dataset and trained it. Evaluated model on testing set. Saved the improved model and used it for prediction on testing data.

then Provided plot of confusion matric, after that, Provided Training and testing Loss and accuracy plots in one plot using subplot command and history object.

Provided two more visualizations reflecting solution. These steps lead to improved response for new dataset when compared with baseline model and enhanced architecture.

**Screenshots of the output:**





**Video Link:** **https://drive.google.com/file/d/1fr6ir0Ie\_hqJj4JSa3As066ZNrVj5rzS/view?usp=sharing**

**GitHub Link:** https://github.com/VarnaNemulla/Assignment8