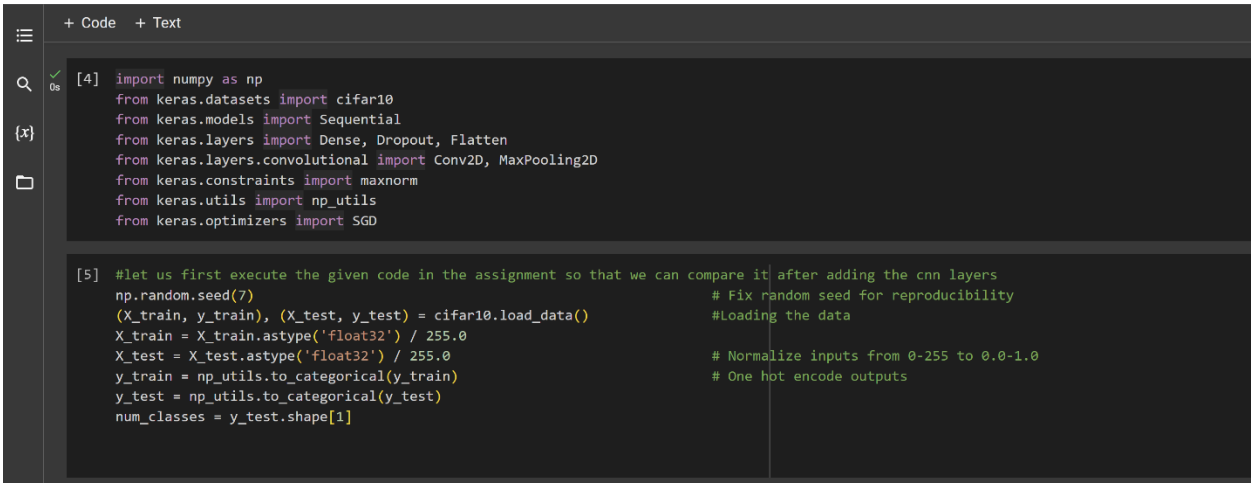
Neural Network and Deep Learning-Assignment 3

Name:Sai Swetha Nambari

Github link: https://github.com/SwethaNam/nnassignment-3.git

Video link:
https://drive.google.com/drive/folders/1BHNdwUH1QGhE_p31O0s30KrjbOQNw79e?usp=sharing

Let us first execute the code in image_classification.py which is given in the assignment, so that we can check both the performances after adding the layers given in the first question.

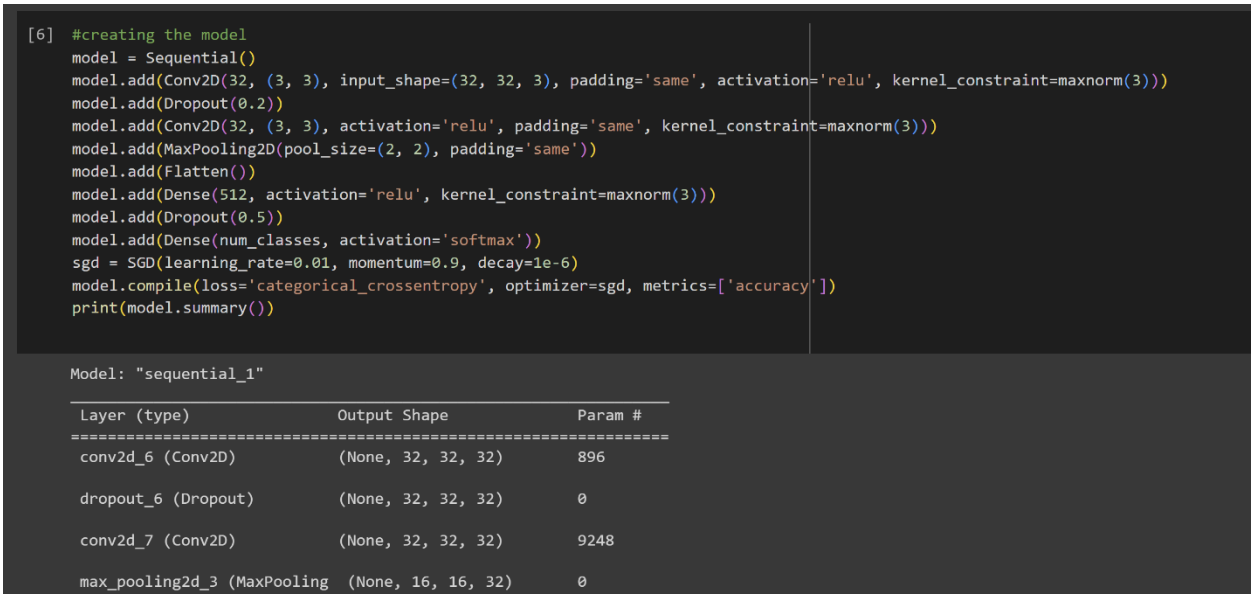


```
+ Code  + Text

[4]  import numpy as np
     from keras.datasets import cifar10
     from keras.models import Sequential
     from keras.layers import Dense, Dropout, Flatten
     from keras.layers.convolutional import Conv2D, MaxPooling2D
     from keras.constraints import maxnorm
     from keras.utils import np_utils
     from keras.optimizers import SGD

[5]  #let us first execute the given code in the assignment so that we can compare it after adding the cnn layers
     np.random.seed(7)                                          # Fix random seed for reproducibility
     (X_train, y_train), (X_test, y_test) = cifar10.load_data()  #Loading the data
     X_train = X_train.astype('float32') / 255.0
     X_test = X_test.astype('float32') / 255.0                  # Normalize inputs from 0-255 to 0.0-1.0
     y_train = np_utils.to_categorical(y_train)                 # One hot encode outputs
     y_test = np_utils.to_categorical(y_test)
     num_classes = y_test.shape[1]
```

Firstly, we import the libraries and then fix the random seed.Later we load the data from cifar10 daatset as shown above and then normalize the inputs.



```
[6]  #creating the model
     model = Sequential()
     model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3), padding='same', activation='relu', kernel_constraint=maxnorm(3)))
     model.add(Dropout(0.2))
     model.add(Conv2D(32, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
     model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
     model.add(Flatten())
     model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
     model.add(Dropout(0.5))
     model.add(Dense(num_classes, activation='softmax'))
     sgd = SGD(learning_rate=0.01, momentum=0.9, decay=1e-6)
     model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
     print(model.summary())

     Model: "sequential_1"
     _________________________________________________________________
      Layer (type)                 Output Shape              Param #
     =================================================================
      conv2d_6 (Conv2D)            (None, 32, 32, 32)        896

      dropout_6 (Dropout)          (None, 32, 32, 32)        0

      conv2d_7 (Conv2D)            (None, 32, 32, 32)        9248

      max_pooling2d_3 (MaxPooling  (None, 16, 16, 32)        0
```

Then we create the model as shown above.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 32, 32, 32)        896

 dropout_6 (Dropout)         (None, 32, 32, 32)        0

 conv2d_7 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d_3 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 flatten_1 (Flatten)         (None, 8192)              0

 dense_3 (Dense)             (None, 512)               4194816

 dropout_7 (Dropout)         (None, 512)               0

 dense_4 (Dense)             (None, 10)                5130

=================================================================
Total params: 4,210,090
Trainable params: 4,210,090
Non-trainable params: 0
_________________________________________________________________
None
```

Output of the above sceenshot.

```
[8]  epochs = 5
     batch_size = 32
     model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=batch_size)
     scores = model.evaluate(X_test, y_test, verbose=0)
     print("Accuracy: %.2f%%" % (scores[1]*100))

     Epoch 1/5
     1563/1563 [==============================] - 11s 6ms/step - loss: 1.6937 - accuracy: 0.3860 - val_loss: 1.4235 - val_accuracy: 0.4820
     Epoch 2/5
     1563/1563 [==============================] - 10s 6ms/step - loss: 1.3466 - accuracy: 0.5169 - val_loss: 1.2552 - val_accuracy: 0.5492
     Epoch 3/5
     1563/1563 [==============================] - 10s 7ms/step - loss: 1.1950 - accuracy: 0.5729 - val_loss: 1.1437 - val_accuracy: 0.5894
     Epoch 4/5
     1563/1563 [==============================] - 10s 6ms/step - loss: 1.0785 - accuracy: 0.6180 - val_loss: 1.0886 - val_accuracy: 0.6141
     Epoch 5/5
     1563/1563 [==============================] - 9s 6ms/step - loss: 0.9817 - accuracy: 0.6546 - val_loss: 1.0235 - val_accuracy: 0.6401
     Accuracy: 64.01%
```

the accuracy is 64.01%.

Question 1:

```
#question 1
import numpy as np
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.constraints import maxnorm
from keras.utils import np_utils
from keras.optimizers import SGD


                                                    # Fix random seed for reproducibility
np.random.seed(7)
                                                    # Load data
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

X_train = X_train.astype('float32') / 255.0         # Normalize inputs from 0-255 to 0.0-1.0
X_test = X_test.astype('float32') / 255.0
                                                    # One hot encode outputs
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
```

In the above screenshot, we have imported the libraries then fixed the random seed ,loaded data from the cifar10 dataset and split it into test and train data.

```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3), padding='same', activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dropout(0.2))
model.add(Dense(1024, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))
```

Creating the model by adding the all the given layers at once.

```
epochs = 5
learning_rate = 0.01
decay_rate = learning_rate / epochs
sgd = SGD(lr=learning_rate, momentum=0.9, decay=decay_rate, nesterov=False)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
print(model.summary())


                                                                    # Fit the model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=32)

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

Here we fit the model and then evaluate the accuracy.

Output:

```
 max_pooling2d_5 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 conv2d_12 (Conv2D)          (None, 8, 8, 128)         73856

 dropout_10 (Dropout)        (None, 8, 8, 128)         0

 conv2d_13 (Conv2D)          (None, 8, 8, 128)         147584

 max_pooling2d_6 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 flatten_2 (Flatten)         (None, 2048)              0

 dropout_11 (Dropout)        (None, 2048)              0

 dense_5 (Dense)             (None, 1024)              2098176

 dropout_12 (Dropout)        (None, 1024)              0

 dense_6 (Dense)             (None, 512)               524800

 dropout_13 (Dropout)        (None, 512)               0

 dense_7 (Dense)             (None, 10)                5130

=================================================================
Total params: 2,915,114
Trainable params: 2,915,114
Non-trainable params: 0
```

```
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/5
1563/1563 [==============================] - 14s 8ms/step - loss: 2.0150 - accuracy: 0.2561 - val_loss: 1.7354 - val_accuracy: 0.3790
Epoch 2/5
1563/1563 [==============================] - 12s 8ms/step - loss: 1.5822 - accuracy: 0.4243 - val_loss: 1.4515 - val_accuracy: 0.4811
Epoch 3/5
1563/1563 [==============================] - 13s 8ms/step - loss: 1.4386 - accuracy: 0.4792 - val_loss: 1.3633 - val_accuracy: 0.5103
Epoch 4/5
1563/1563 [==============================] - 13s 8ms/step - loss: 1.3661 - accuracy: 0.5041 - val_loss: 1.3006 - val_accuracy: 0.5412
Epoch 5/5
1563/1563 [==============================] - 13s 8ms/step - loss: 1.3148 - accuracy: 0.5249 - val_loss: 1.3236 - val_accuracy: 0.5304
Accuracy: 53.04%
```

The accuracy we got here is 53.04%.The accuracy has decreased compared to the above example given in the assignment.

Question 2:

Predict the first 4 images of the test data using the above model.Then ,compare with the with the actual label for those 4 images to check whether or not the model has predicted correctly.

```
[13] #question2
     predictions = model.predict(X_test[:4])                    # Predicting the first 4 images of the test data
     predicted_labels = np.argmax(predictions, axis=1)          # Converting the predictions into class labels
     actual_labels = np.argmax(y_test[:4], axis=1)              # Converting the actual labels to class labels

     print("Predicted labels:", predicted_labels)               # the predicted and actual labels for the first 4 images
     print("Actual labels:   ", actual_labels)

     1/1 [==============================] - 0s 21ms/step
     Predicted labels: [3 8 8 8]
     Actual labels:    [3 8 8 0]
```

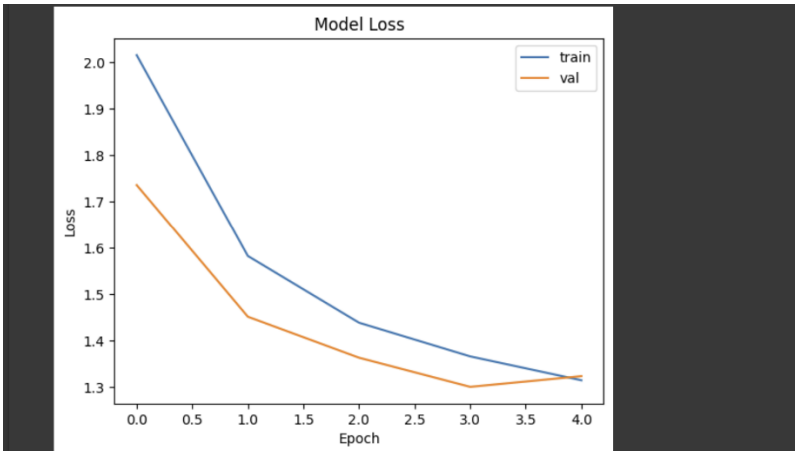
Question 3:

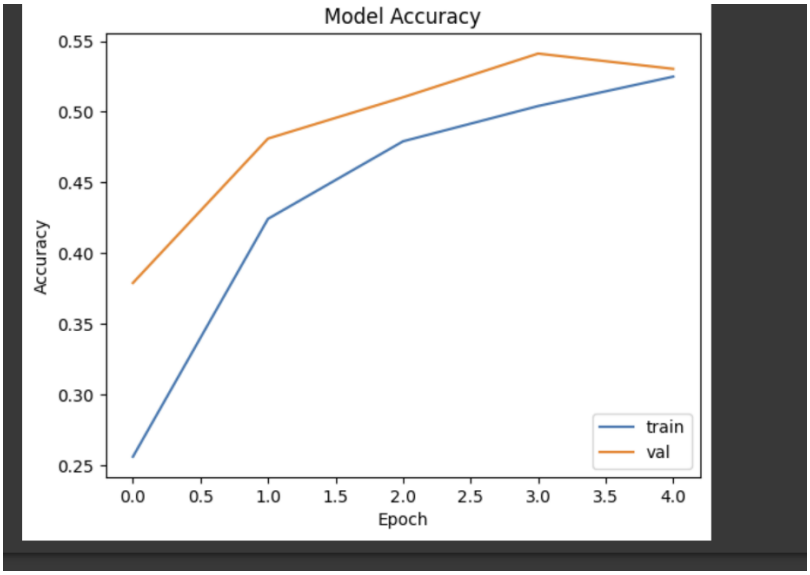
Visualize Loss and Accuracy using the history object.

```
[14] #question 3:
     import matplotlib.pyplot as plt

     # Plot the training and validation loss
     plt.plot(history.history['loss'])
     plt.plot(history.history['val_loss'])
     plt.title('Model Loss')
     plt.ylabel('Loss')
     plt.xlabel('Epoch')
     plt.legend(['train', 'val'], loc='upper right')
     plt.show()

     # Plot the training and validation accuracy
     plt.plot(history.history['accuracy'])
     plt.plot(history.history['val_accuracy'])
     plt.title('Model Accuracy')
     plt.ylabel('Accuracy')
     plt.xlabel('Epoch')
     plt.legend(['train', 'val'], loc='lower right')
     plt.show()
```


Model Loss
train
val
Loss
Epoch
2.0
1.9
1.8
1.7
1.6
1.5
1.4
1.3
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Model Accuracy
Accuracy
Epoch
train
val
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0