Student ID: 700743212

Name: Gundlapally sreenidhi

NN & DL - Assignment 3

Video Link:
https://drive.google.com/file/d/1n_nYTv_sZ_YpEEXujkyAo9BNwy1hAtGv/view?usp=sharing

GitHub Link: https://github.com/sreenidhi-ux/assignment3

Question 1:

Importing set of libraries, training and testing the data

+ Code + Text

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

```
[ ] np.random.seed(7)
```

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

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 2s 0us/step
```

```
[ ] y_train = np_utils.to_categorical(y_train)
    y_test = np_utils.to_categorical(y_test)
    num_classes = y_test.shape[1]
```

```
[ ] 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"
```

```
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"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 dropout (Dropout)           (None, 32, 32, 32)        0

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

 flatten (Flatten)           (None, 8192)              0

 dense (Dense)               (None, 512)               4194816

 dropout_1 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 10)                5130

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

Fitting the data and evaluating the scores for accuracy

```
[ ] epochs = 5
    batch_size = 32
    model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=batch_size)
```

```
Epoch 1/5
1563/1563 [==============================] - 302s 192ms/step - loss: 1.6841 - accuracy: 0.3892 - val_loss: 1.3400 - val_accuracy: 0.5142
Epoch 2/5
1563/1563 [==============================] - 316s 202ms/step - loss: 1.3216 - accuracy: 0.5272 - val_loss: 1.1922 - val_accuracy: 0.5684
Epoch 3/5
1563/1563 [==============================] - 316s 202ms/step - loss: 1.1505 - accuracy: 0.5912 - val_loss: 1.1238 - val_accuracy: 0.6010
Epoch 4/5
1563/1563 [==============================] - 310s 199ms/step - loss: 1.0075 - accuracy: 0.6454 - val_loss: 1.0171 - val_accuracy: 0.6423
Epoch 5/5
1563/1563 [==============================] - 296s 189ms/step - loss: 0.8995 - accuracy: 0.6832 - val_loss: 0.9656 - val_accuracy: 0.6657
<keras.callbacks.History at 0x7f0b836071c0>
```

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

```
Accuracy: 66.57%
```

Accuracy we got here is: 66.57%

Finding the performance change in the below code

```
[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

     # Fixing the  random seed for reproducibility
     np.random.seed(7)

     # Loading the  data
     (X_train, y_train), (X_test, y_test) = cifar10.load_data()

     # Normalizing inputs from 0-255 to 0.0-1.0
     X_train = X_train.astype('float32') / 255.0
     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]
```

Creating

And compiling the model

```
# 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)))
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(num_classes, activation='softmax'))
```

Fitting and evaluating the model.

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

# Fitting 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))
```

Output:

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 dropout (Dropout)           (None, 32, 32, 32)        0

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 dropout_1 (Dropout)         (None, 16, 16, 64)        0

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 conv2d_4 (Conv2D)           (None, 8, 8, 128)         73856

 dropout_2 (Dropout)         (None, 8, 8, 128)         0

 conv2d_5 (Conv2D)           (None, 8, 8, 128)         147584

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 flatten (Flatten)           (None, 2048)              0

 dropout_3 (Dropout)         (None, 2048)              0

 dense (Dense)               (None, 1024)              2098176

 dropout_4 (Dropout)         (None, 1024)              0

 dense_1 (Dense)             (None, 512)               524800

 dropout_5 (Dropout)         (None, 512)               0

 dense_2 (Dense)             (None, 10)                5130

=================================================================
Total params: 2,915,114
Trainable params: 2,915,114
```

```
=================================================================
Total params: 2,915,114
Trainable params: 2,915,114
Non-trainable params: 0
_________________________________________________________________
/usr/local/lib/python3.10/dist-packages/keras/optimizers/legacy/gradient_descent.py:114: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
None
Epoch 1/5
1563/1563 [==============================] - 598s 381ms/step - loss: 1.9256 - accuracy: 0.2889 - val_loss: 1.6807 - val_accuracy: 0.3877
Epoch 2/5
1563/1563 [==============================] - 598s 382ms/step - loss: 1.5471 - accuracy: 0.4336 - val_loss: 1.4688 - val_accuracy: 0.4670
Epoch 3/5
1563/1563 [==============================] - 594s 380ms/step - loss: 1.4079 - accuracy: 0.4881 - val_loss: 1.3898 - val_accuracy: 0.4968
Epoch 4/5
1563/1563 [==============================] - 584s 374ms/step - loss: 1.3209 - accuracy: 0.5222 - val_loss: 1.2574 - val_accuracy: 0.5492
Epoch 5/5
1563/1563 [==============================] - 575s 368ms/step - loss: 1.2645 - accuracy: 0.5439 - val_loss: 1.2194 - val_accuracy: 0.5616
Accuracy: 56.16%
```

Accuracy we got here is: 56.16%.

The performance of the model is likely to improve with the addition of more layers and higher number of feature maps, but it will also increase the complexity and the training time of the model. The new model architecture provided in the instruction includes several new layers and higher number of feature maps, which may improve the accuracy of the model.

Question 2:

Predicting the first 4 images of the test data

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

     # Printing the predicted and actual labels for the first 4 images
     print("Predicted labels:", predicted_lbls)
     print("Actual labels:   ", actual_lbls)
```

```
1/1 [==============================] - 0s 113ms/step
Predicted labels: [3 8 8 8]
Actual labels:    [3 8 8 0]
```

Question 3:

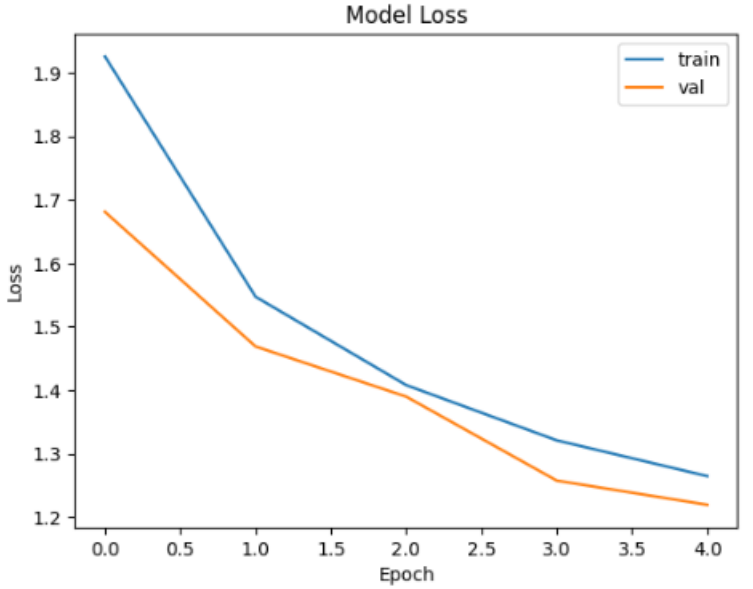
Importing the libraries and plottinh the training and validation accuracy

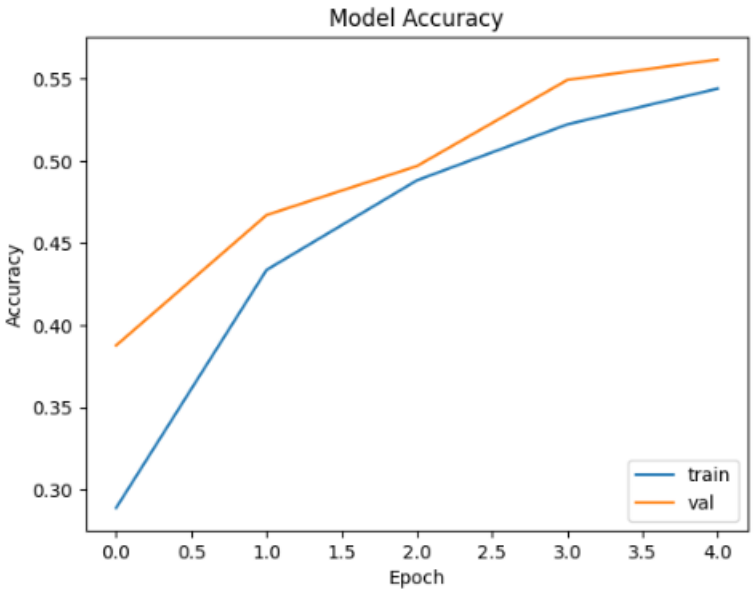
```python
import matplotlib.pyplot as plt

# Plotting 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()

# Plotting 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()
```

Output:


Model Loss
train
val
Loss
Epoch
1.9
1.8
1.7
1.6
1.5
1.4
1.3
1.2
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


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