NEURAL NETWORKS & DEEP LEARNING

ICP - 7

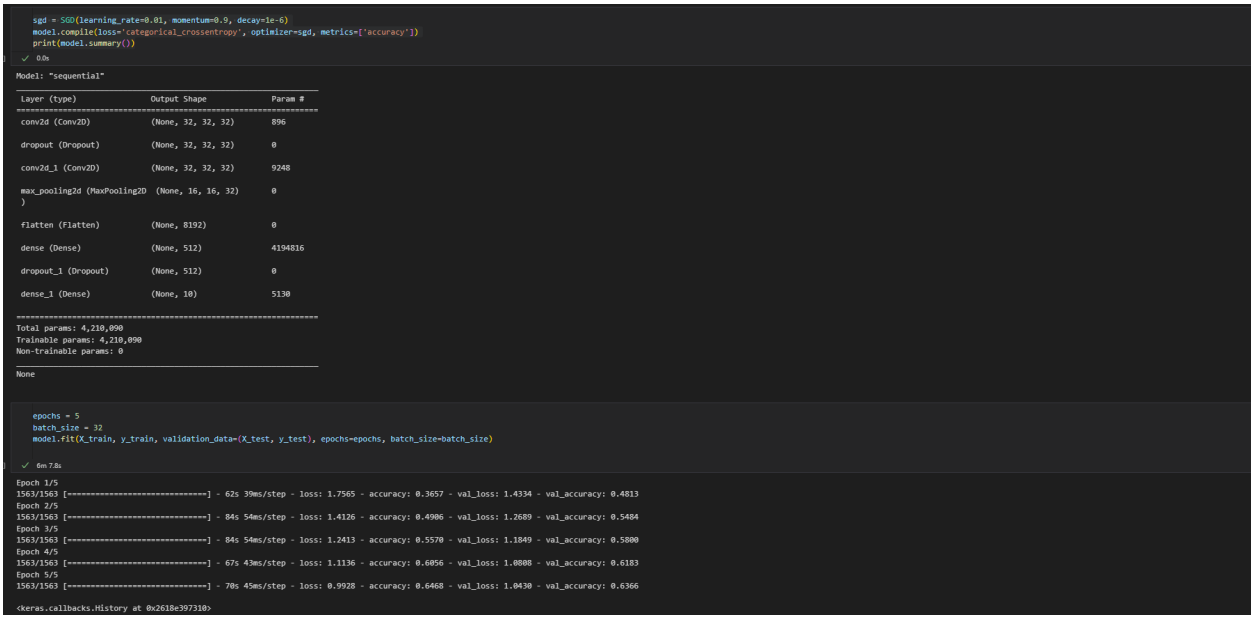
Student name: Sai pavan reddy pogula

Student id: 700741739

GitHub Link : https://github.com/pavan-reddy-28/icp7

1) Follow the instruction below and then report how the performance changed.(apply all at once)
    - Convolutional input layer, 32 feature maps with a size of 3×3 and a rectifier activation function.
    - Dropout layer at 20%.
    - Convolutional layer, 32 feature maps with a size of 3×3 and a rectifier activation function.
    - Max Pool layer with size 2×2.
    - Convolutional layer, 64 feature maps with a size of 3×3 and a rectifier activation function.
    - Dropout layer at 20%.
    - Convolutional layer, 64 feature maps with a size of 3×3 and a rectifier activation function.
    - Max Pool layer with size 2×2.
    - Convolutional layer, 128 feature maps with a size of 3×3 and a rectifier activation function.
    - Dropout layer at 20%.
    - Convolutional layer,128 feature maps with a size of 3×3 and a rectifier activation function.
    - Max Pool layer with size 2×2.

- Flatten layer.
- Dropout layer at 20%.
- Fully connected layer with 1024 units and a rectifier activation function.
- Dropout layer at 20%.
- Fully connected layer with 512 units and a rectifier activation function.
- Dropout layer at 20%.
- Fully connected output layer with 10 units and a Softmax activation function



Accuracy : 63.66 %

Did the Performance changed? **Yes**

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

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

[9] ✓ 2.7s Python

```
Accuracy: 63.66%
```

```
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()

# Normalize 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]

# Create 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'))
```

```
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'))

# Compile model
epochs = 5
learning_rate = 0.01
decay_rate = learning_rate / epochs
sgd = SGD(learning_rate=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)

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

[14] ✓ 13m 8.9s Pyt

```
Output exceeds the size limit. Open the full output data in a text editor
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 32, 32, 32)        896

 dropout_8 (Dropout)         (None, 32, 32, 32)        0

 conv2d_9 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d_4 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 conv2d_10 (Conv2D)          (None, 16, 16, 64)        18496

 dropout_9 (Dropout)         (None, 16, 16, 64)        0

 conv2d_11 (Conv2D)          (None, 16, 16, 64)        36928

 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
```

```
dropout_9 (Dropout)          (None, 16, 16, 64)        0

conv2d_11 (Conv2D)           (None, 16, 16, 64)        36928

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

...
1563/1563 [==============================] - 166s 106ms/step - loss: 1.3246 - accuracy: 0.5214 - val_loss: 1.3273 - val_accuracy: 0.5176
Epoch 5/5
1563/1563 [==============================] - 215s 137ms/step - loss: 1.2663 - accuracy: 0.5423 - val_loss: 1.2298 - val_accuracy: 0.5526
Accuracy: 55.26%
```

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

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

```
[15] ✓ 0.1s                                                    Python

1/1 [==============================] - 0s 70ms/step
Predicted labels: [3 1 8 8]
Actual labels:    [3 8 8 0]
```

Accuracy 55.26%; The model has predicted correctly.

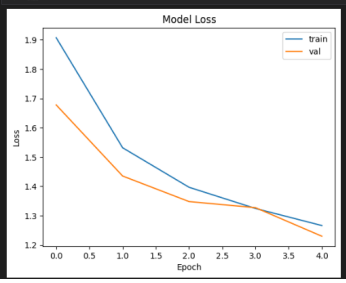
3) Visualize Loss and Accuracy using the history object

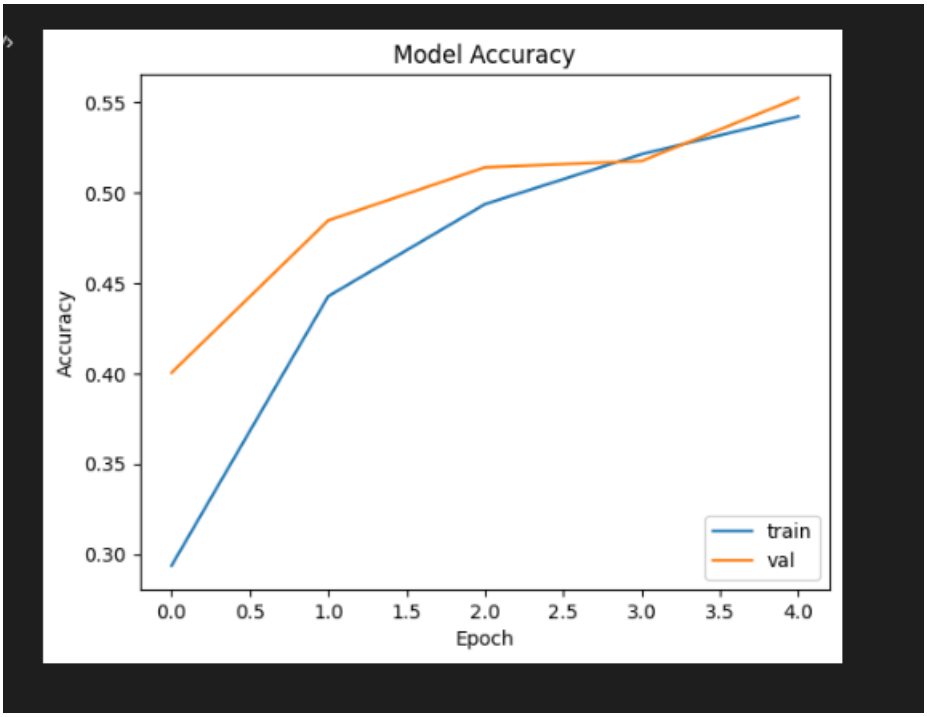
```
import matplotlib.pyplot as plt

# 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()
```

```
[16] ✓ 0.2s                                                    Pyth
```





Loss and Accuracy using the history object.