Assignment 3

Student Name: Krishna Vamsi Koppula
Student ID: 700745021

GitHub Link:
https://github.com/Krishnavamsikoppula/Assignment_3_NNDL_Summer

Video Link:
https://drive.google.com/file/d/1KWLwhk4mCklbBsRfarTvNvER5n2FIiTo/view?usp=sharing

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

NNDL_700745021_ICP3.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ 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.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]
```

```python
# Create the model
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3), padding='same', activation='relu'))
model.add(Dropout(0.2))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(Dropout(0.2))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dropout(0.2))
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
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(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)
```

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

Output:

```
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)

 conv2d_8 (Conv2D)           (None, 16, 16, 64)        18496

 dropout_7 (Dropout)         (None, 16, 16, 64)        0

 conv2d_9 (Conv2D)           (None, 16, 16, 64)        36928

 max_pooling2d_4 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 conv2d_10 (Conv2D)          (None, 8, 8, 128)         73856

 dropout_8 (Dropout)         (None, 8, 8, 128)         0

 conv2d_11 (Conv2D)          (None, 8, 8, 128)         147584

 max_pooling2d_5 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 flatten_1 (Flatten)         (None, 2048)              0

 dropout_9 (Dropout)         (None, 2048)              0

 dense_3 (Dense)             (None, 1024)              2098176
```

```
 dropout_10 (Dropout)        (None, 1024)              0

 dense_4 (Dense)             (None, 512)               524800

 dropout_11 (Dropout)        (None, 512)               0

 dense_5 (Dense)             (None, 10)                5130

=================================================================
Total params: 2,915,114
Trainable params: 2,915,114
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/5
1563/1563 [==============================] - 14s 8ms/step - loss: 1.8837 - accuracy: 0.3061 - val_loss: 1.5697 - val_accuracy: 0.4337
Epoch 2/5
1563/1563 [==============================] - 12s 8ms/step - loss: 1.5043 - accuracy: 0.4499 - val_loss: 1.4281 - val_accuracy: 0.4964
Epoch 3/5
1563/1563 [==============================] - 12s 8ms/step - loss: 1.3666 - accuracy: 0.5030 - val_loss: 1.3071 - val_accuracy: 0.5325
Epoch 4/5
1563/1563 [==============================] - 12s 8ms/step - loss: 1.2885 - accuracy: 0.5359 - val_loss: 1.2516 - val_accuracy: 0.5507
Epoch 5/5
1563/1563 [==============================] - 12s 8ms/step - loss: 1.2268 - accuracy: 0.5556 - val_loss: 1.1757 - val_accuracy: 0.5754
Accuracy: 57.54%
```

Did the performance change?

With the addition of more layers and feature maps, the model's performance is likely to improve, but the complexity and training time of the model will also rise. The new model architecture described in the instruction has more feature maps and new layers, which could increase the model's accuracy.

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.

```
# 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)
```

```
1/1 [==============================] - 0s 324ms/step
Predicted labels: [3 1 8 0]
Actual labels: [3 8 8 0]
```

3. Visualize Loss and Accuracy using the history object.

```python
import matplotlib.pyplot as plt

# Plot the training and validation loss
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()

# Plot the training and validation accuracy
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```