ICP7

Student Name: **Srinivas Musinuri**
Student id: 700758813

GitHub Link: https://github.com/srinivasmusinuri/700758813_NNDL_ICP7

Video Link:
https://drive.google.com/file/d/1jkPijxQsyEZDXnd93340q09kYo9tjFTV/view?usp=drive_link

1. 1. Follow the instruction below and then report how the performance changed.

700758813_NNDL_ICP7.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

```python
# Simple CNN model for CIFAR-10
import numpy
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.optimizers import SGD
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.utils import to_categorical
#from keras import backend as K
#K.set_image_dim_ordering('th')
```

```python
[2] # fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# load data
(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 [==============================] - 4s 0us/step
```

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

```
[4] # 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(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))
```

```
[5] # Compile model
    epochs = 5
    lrate = 0.01
    decay = lrate/epochs
    sgd = SGD(lr=lrate)
    model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
    print(model.summary())
    # Fit the model
    model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=32)
    # Final evaluation of the model
    scores = model.evaluate(X_test, y_test, verbose=0)
    print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.SGD.
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 (MaxPooling2  (None, 16, 16, 32)        0
 D)

 flatten (Flatten)           (None, 8192)              0

 dense (Dense)               (None, 512)               4194816

 dropout_1 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 10)                5130

=================================================================
Total params: 4210090 (16.06 MB)
Trainable params: 4210090 (16.06 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
Epoch 1/5
1563/1563 [==============================] - 19s 6ms/step - loss: 1.9187 - accuracy: 0.3112 - val_loss: 1.7344 - val_accuracy: 0.3906
Epoch 2/5
1563/1563 [==============================] - 9s 6ms/step - loss: 1.6657 - accuracy: 0.4103 - val_loss: 1.5415 - val_accuracy: 0.4607
Epoch 3/5
1563/1563 [==============================] - 10s 7ms/step - loss: 1.5524 - accuracy: 0.4520 - val_loss: 1.4646 - val_accuracy: 0.4976
Epoch 4/5
1563/1563 [==============================] - 9s 6ms/step - loss: 1.4555 - accuracy: 0.4851 - val_loss: 1.4146 - val_accuracy: 0.5093
Epoch 5/5
1563/1563 [==============================] - 10s 7ms/step - loss: 1.3675 - accuracy: 0.5152 - val_loss: 1.2835 - val_accuracy: 0.5437
Accuracy: 54.37%
```

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 Did the performance change?

```
[10]
import numpy
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
#from keras.constraints import maxnorm
from keras.optimizers import SGD
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.utils import to_categorical

# Fix random seed for reproducibility
numpy.random.seed(7)

(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')
X_test = X_test.astype('float32')
X_train = X_train / 255.0
X_test = X_test / 255.0
# one hot encode outputs
y_train =to_categorical(y_train)
y_test =to_categorical(y_test)
num_classes = 10
```

```
# 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
lrate = 0.01
decay = lrate/epochs
sgd = SGD(lr=lrate)
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)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
 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: 2915114 (11.12 MB)
Trainable params: 2915114 (11.12 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
Epoch 1/5
1563/1563 [==============================] - 14s 8ms/step - loss: 2.1582 - accuracy: 0.1918 - val_loss: 2.0066 - val_accuracy: 0.2972
Epoch 2/5
1563/1563 [==============================] - 12s 8ms/step - loss: 1.8660 - accuracy: 0.3234 - val_loss: 1.7094 - val_accuracy: 0.3782
Epoch 3/5
1563/1563 [==============================] - 12s 8ms/step - loss: 1.6595 - accuracy: 0.3948 - val_loss: 1.5780 - val_accuracy: 0.4422
Epoch 4/5
1563/1563 [==============================] - 12s 7ms/step - loss: 1.5394 - accuracy: 0.4419 - val_loss: 1.4614 - val_accuracy: 0.4809
Epoch 5/5
1563/1563 [==============================] - 11s 7ms/step - loss: 1.4504 - accuracy: 0.4737 - val_loss: 1.3986 - val_accuracy: 0.5071
Accuracy: 50.71%
```

Regarding performance, the second version with the more complex CNN architecture is likely to have better performance compared to the first version with the simpler architecture. This is because the second version has a larger and more sophisticated model with additional layers, which can capture more complex patterns and features in the CIFAR-10 dataset.

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.

```
[11] # Predict the first 4 images of the test data
     predictions = model.predict(X_test[:4])
     # Convert the predictions to class labels
     predicted_labels = numpy.argmax(predictions, axis=1)
     # Convert the actual labels to class labels
     actual_labels = numpy.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 112ms/step
Predicted labels: [3 8 8 8]
Actual labels:    [3 8 8 0]
```

In the above case, you can see that the model correctly predicted the first three labels (3, 8, and 8) but made an incorrect prediction for the fourth label (predicted 8, actual 0).

This information is useful for assessing how well the model is performing on this specific batch of data.

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()
```