Documentation

Group #3 Derek Chen, Thomas Griffin

AI

Github link:
https://github.com/987derek/987derek/blob/1cd8922761a22acb99f2445b67c857952e53a1a0/CNN+Increasing_Dropout_+_Data_Augmentation_+_Batch_Normalization.ipynb

Changes Made to Code:

Imported Libraries:

Several extra libraries were imported to be able to improve the code. The code segment below shows all the libraries that were imported.

```
# test harness for evaluating models on the cifar10 dataset
import sys
from matplotlib import pyplot
from keras.datasets import cifar10
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import Flatten
from keras.optimizers import SGD
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dropout
from keras.layers import BatchNormalization
# extra libraries imported

import tensorflow as tf

from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

Convolutional Base:

The convolutional base of the code was improved using dropout regularization. Dropout randomly removes nodes from the network, which forces the remaining nodes to adapt and make up for the nodes that were dropped. Batch normalization was also utilized in the model. Batch normalization standardizes the inputs to a layer, and helps to accelerate the training process. The code segment below shows the dropout regularization and batch normalization being implemented in the model.

```
model = models.Sequential()
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(32, 32, 3)))
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.3))
model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.4))
model.add(Flatten())
model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
# compile model
opt = SGD(lr=0.001, momentum=0.9)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
```

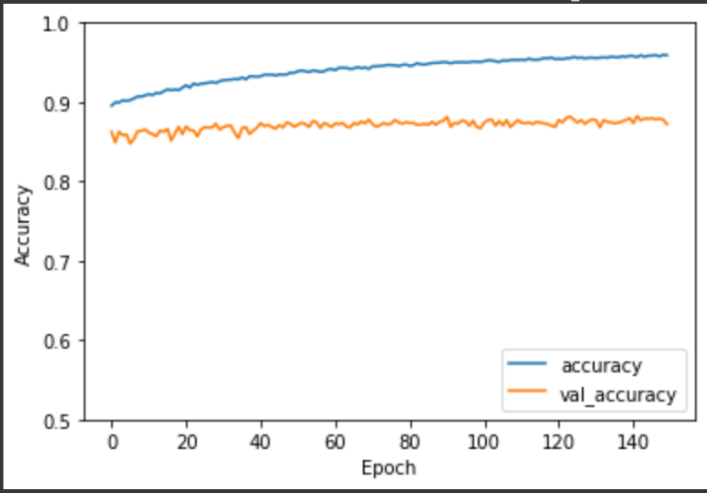
Train results:

```
Epoch 144/150
1563/1563 [==============================] - 14s 9ms/step - loss: 0.1272 - accuracy: 0.9570 - val_loss: 0.4696 - val_accuracy: 0.8792
Epoch 145/150
1563/1563 [==============================] - 14s 9ms/step - loss: 0.1212 - accuracy: 0.9584 - val_loss: 0.4751 - val_accuracy: 0.8789
Epoch 146/150
1563/1563 [==============================] - 16s 10ms/step - loss: 0.1198 - accuracy: 0.9584 - val_loss: 0.4508 - val_accuracy: 0.8796
Epoch 147/150
1563/1563 [==============================] - 15s 9ms/step - loss: 0.1194 - accuracy: 0.9590 - val_loss: 0.4873 - val_accuracy: 0.8782
Epoch 148/150
1563/1563 [==============================] - 13s 9ms/step - loss: 0.1232 - accuracy: 0.9575 - val_loss: 0.4538 - val_accuracy: 0.8790
Epoch 149/150
1563/1563 [==============================] - 14s 9ms/step - loss: 0.1216 - accuracy: 0.9594 - val_loss: 0.4579 - val_accuracy: 0.8779
Epoch 150/150
1563/1563 [==============================] - 14s 9ms/step - loss: 0.1213 - accuracy: 0.9589 - val_loss: 0.4871 - val_accuracy: 0.8721
```

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

```
313/313 - 1s - loss: 0.4871 - accuracy: 0.8721 - 1s/epoch - 3ms/step
```




```
print(test_acc)
```

```
0.8720999956130981
```

Game

You need to modify the code at line 40 between the “”, to the path that you save the file named: high-scores.txt to run the game.

Four Hacks and Tweaks are added:

More high scores:

Simply add more 0 spaced with “ “ in the document: high-scores.txt

Lives:

Add a global variable called lives and set it to 3.The game over flag: game_over is always false, unless lives is equal to 0. Conditions are add to update the number of lives left:

```
# game over
if balloon.top < 0 or balloon.bottom > 560:
    lives -=1
    if lives == 0:
        game_over = True
        update_high_scores()

# collisions with obstacles
if balloon.collidepoint(bird.x, bird.y) or \
    balloon.collidepoint(house.x, house.y) or \
    balloon.collidepoint(tree.x, tree.y):
        lives -=1
        if lives == 0:
            game_over = True
            update_high_scores()
```

Speed it up:

To change the speed of the bird, change the number of pixels to update its x coordinates.

```
# move the bird
if bird.x > 0:
    bird.x -= 6
    if number_of_updates == 9:
        flap()
        number_of_updates = 0
    else:
        number_of_updates += 1
```

Space out the obstacles:

We added a condition that if the x coordinates of the tree and the house are the same, the x coordinates of the tree will be set randomly again so that the tree and the house won't overlap.

```
# move the tree
if tree.right > 0:
        tree.x -= 2
else:
    tree.x = randint(800, 1600)
    score += 1
    if bird.x == house.x:
        tree.x = randint(800, 1600)
```