Summer 2024: CS5720 NNDL - ICP-4

Lasya Vanga **(700762893)**

GitHub Link: https://github.com/Lasya-vanga/NNDL-ICP4

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?

Untitled5.ipynb

File Edit View Insert Runtime Tools Help Last saved at 12:16 PM

+ Code + Text

```
from re import X
import numpy
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.optimizers.legacy import SGD #using legacy optimizer so
#that we can use lr schedular to show more flexibility in how lr changes overtime
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.constraints import MaxNorm
#from keras import backend as K
#K.set_image_dim_ordering('th')

# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# 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')
X_test = X_test.astype('float32')
X_train = X_train / 255.0
X_test = X_test / 255.0
# one hot encode outputs
```

✓ Connected to Python 3 Google Compute Engin

File Edit View Insert Runtime Tools Help Last saved at 12:16 PM

+ Code + Text

```
X_test = X_test / 255.0
# one hot encode outputs
y_train = to_categorical(y_train,10)
y_test = to_categorical(y_test,10)
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(Flatten())
model.add(Dense(512, activation='relu', kernel_constraint=MaxNorm(3)))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

# Compile the model
epochs = 5
lrate = 0.01
decay = lrate/epochs
sgd = SGD(learning_rate=lrate, momentum=0.9, decay=decay, nesterov=False)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# fit the model
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))
```

```
# Compile the model
epochs = 5
lrate = 0.01
decay = lrate/epochs
sgd = SGD(learning_rate=lrate, momentum=0.9, decay=decay, nesterov=False)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# fit the model
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))
```

```
Epoch 1/5
1563/1563 [==============================] - 294s 164ms/step - loss: 1.5381 - accuracy: 0.4480 - val_loss: 1.2124 - val_accuracy: 0.5791
Epoch 2/5
1563/1563 [==============================] - 255s 163ms/step - loss: 1.1949 - accuracy: 0.5719 - val_loss: 1.0813 - val_accuracy: 0.6290
Epoch 3/5
1563/1563 [==============================] - 246s 158ms/step - loss: 1.0858 - accuracy: 0.6149 - val_loss: 0.9977 - val_accuracy: 0.6581
Epoch 4/5
1563/1563 [==============================] - 252s 161ms/step - loss: 1.0212 - accuracy: 0.6396 - val_loss: 0.9576 - val_accuracy: 0.6620
Epoch 5/5
1563/1563 [==============================] - 250s 160ms/step - loss: 0.9724 - accuracy: 0.6558 - val_loss: 0.9477 - val_accuracy: 0.6708
Accuracy: 67.08%
```

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.

```
[ ] #2. Predict the first 4 images of the test data using the above model. Then, compare wi
    import numpy as np

    # Predict the first 4 images of the test data
    predictions = model.predict(x_test[:4])
    predicted_classes = np.argmax(predictions, axis=1)

    # Get the actual labels for the first 4 images
    actual_classes = np.argmax(y_test[:4], axis=1)

    # Compare the predicted classes with the actual classes
    for i in range(4):
        print(f"Image {i+1}:")
        print(f"Predicted: {predicted_classes[i]}, Actual: {actual_classes[i]}")
        print(f"Correct: {predicted_classes[i] == actual_classes[i]}")
```

```
1/1 [==============================] - 0s 34ms/step
Image 1:
Predicted: 3, Actual: 3
Correct: True
Image 2:
Predicted: 8, Actual: 8
Correct: True
Image 3:
Predicted: 8, Actual: 8
Correct: True
Image 4:
Predicted: 8, Actual: 0
Correct: False
```

✓ Connected to Python 3 Google (

3. Visualize Loss and Accuracy using the history object

```
2]  import matplotlib.pyplot as plt
    history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=32)


    # Extract the loss and accuracy from the history object
    train_loss = history.history['loss']
    val_loss = history.history['val_loss']
    train_accuracy = history.history['accuracy']
    val_accuracy = history.history['val_accuracy']

    # Define the number of epochs
    epochs = range(1, len(train_loss) + 1)

    # Plot training and validation loss
    plt.figure(figsize=(14, 5))
    plt.subplot(1, 2, 1)
    plt.plot(epochs, train_loss, label='Training Loss')
    plt.plot(epochs, val_loss, label='Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title('Training and Validation Loss')
    plt.legend()

    # Plot training and validation accuracy
    plt.subplot(1, 2, 2)
    plt.plot(epochs, train_accuracy, label='Training Accuracy')
    plt.plot(epochs, val_accuracy, label='Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
```

Code + Text

```
[2] plt.plot(epochs, val_accuracy, label='Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.legend()

    # Display the plots
    plt.show()
```

```
Epoch 1/5
1563/1563 [==============================] - 250s 160ms/step - loss: 0.9397 - accuracy: 0.6685 - val_loss: 0.8971 - val_accuracy: 0.6877
Epoch 2/5
1563/1563 [==============================] - 243s 155ms/step - loss: 0.9046 - accuracy: 0.6798 - val_loss: 0.8886 - val_accuracy: 0.6899
Epoch 3/5
1563/1563 [==============================] - 240s 154ms/step - loss: 0.8789 - accuracy: 0.6894 - val_loss: 0.9100 - val_accuracy: 0.6842
Epoch 4/5
1563/1563 [==============================] - 242s 155ms/step - loss: 0.8545 - accuracy: 0.7008 - val_loss: 0.8776 - val_accuracy: 0.6938
Epoch 5/5
1563/1563 [==============================] - 227s 145ms/step - loss: 0.8372 - accuracy: 0.7067 - val_loss: 0.8827 - val_accuracy: 0.6945
```