NEURAL NETWORKS & DEEP LEARNING: ICP3

Name: Reshma Maddala

ID: 700740808

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?

]:
```
#importing the Libraries
import numpy as np
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.constraints import maxnorm
from keras.optimizers import SGD
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.utils import np_utils

#Set random seed value
np.random.seed(7)
#Load and preprocess data
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]

#Create 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), padding='same'))
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'))
```

```python
#Compile model
sgd = SGD(learning_rate=0.01, momentum=0.9, decay=1e-6)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
print(model.summary())

#Train model
epochs = 5
batch_size = 32
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=batch_size)
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 32, 32, 32)        896

 dropout_2 (Dropout)         (None, 32, 32, 32)        0

 conv2d_3 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d_1 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 flatten_1 (Flatten)         (None, 8192)              0

 dense_2 (Dense)             (None, 512)               4194816

 dropout_3 (Dropout)         (None, 512)               0

 dense_3 (Dense)             (None, 10)                5130

=================================================================
Total params: 4,210,090
Trainable params: 4,210,090
Non-trainable params: 0
```

```
_________________________________________________________________
None
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 32, 32, 32)        896

 dropout_2 (Dropout)         (None, 32, 32, 32)        0

 conv2d_3 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d_1 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 flatten_1 (Flatten)         (None, 8192)              0

 dense_2 (Dense)             (None, 512)               4194816

 dropout_3 (Dropout)         (None, 512)               0

 dense_3 (Dense)             (None, 10)                5130

=================================================================
Total params: 4,210,090
Trainable params: 4,210,090
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/5
1563/1563 [==============================] - 164s 104ms/step - loss: 1.7320 - accuracy: 0.3701 - val_loss: 1.4194 - val_accurac
y: 0.4944
Epoch 2/5
1563/1563 [==============================] - 165s 105ms/step - loss: 1.3552 - accuracy: 0.5128 - val_loss: 1.2045 - val_accurac
y: 0.5676
```

```
Epoch 3/5
1563/1563 [==============================] - 165s 106ms/step - loss: 1.1688 - accuracy: 0.5842 - val_loss: 1.0648 - val_accurac
y: 0.6252
Epoch 4/5
1563/1563 [==============================] - 160s 102ms/step - loss: 1.0351 - accuracy: 0.6336 - val_loss: 0.9853 - val_accurac
y: 0.6544
Epoch 5/5
1563/1563 [==============================] - 165s 106ms/step - loss: 0.9280 - accuracy: 0.6732 - val_loss: 0.9793 - val_accurac
y: 0.6610
```

Out[5]: <keras.callbacks.History at 0x7f67c770eb80>

In [ ]:

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

```
Accuracy: 66.10%
```

:
```
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

seed = 7

(X_train, y_train), (X_test, y_test) = cifar10.load_data()

X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
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(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'))
```

[ ]:
```
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())
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 32, 32, 32)        896

 dropout_4 (Dropout)         (None, 32, 32, 32)        0

 conv2d_5 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d_2 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 conv2d_6 (Conv2D)           (None, 16, 16, 64)        18496

 dropout_5 (Dropout)         (None, 16, 16, 64)        0

 conv2d_7 (Conv2D)           (None, 16, 16, 64)        36928

 max_pooling2d_3 (MaxPooling  (None, 8, 8, 64)         0
 2D)
```

```
conv2d_8 (Conv2D)           (None, 8, 8, 128)         73856

dropout_6 (Dropout)         (None, 8, 8, 128)         0

conv2d_9 (Conv2D)           (None, 8, 8, 128)         147584

max_pooling2d_4 (MaxPooling  (None, 4, 4, 128)        0
2D)

flatten_2 (Flatten)         (None, 2048)              0

dropout_7 (Dropout)         (None, 2048)              0

dense_4 (Dense)             (None, 1024)              2098176

dropout_8 (Dropout)         (None, 1024)              0

dense_5 (Dense)             (None, 512)               524800

dropout_9 (Dropout)         (None, 512)               0

dense_6 (Dense)             (None, 10)                5130

=================================================================
Total params: 2,915,114
Trainable params: 2,915,114
Non-trainable params: 0
_________________________________________________________________
None
```

```
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=32)

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

```
Epoch 1/5
1563/1563 [==============================] - 320s 204ms/step - loss: 1.8690 - accuracy: 0.3094 - val_loss: 1.5686 - val_accurac
y: 0.4288
Epoch 2/5
1563/1563 [==============================] - 319s 204ms/step - loss: 1.4913 - accuracy: 0.4537 - val_loss: 1.3887 - val_accurac
y: 0.4972
Epoch 3/5
1563/1563 [==============================] - 319s 204ms/step - loss: 1.3537 - accuracy: 0.5097 - val_loss: 1.3130 - val_accurac
y: 0.5285
Epoch 4/5
1563/1563 [==============================] - 318s 203ms/step - loss: 1.2687 - accuracy: 0.5440 - val_loss: 1.1876 - val_accurac
y: 0.5719
Epoch 5/5
1563/1563 [==============================] - 318s 204ms/step - loss: 1.2044 - accuracy: 0.5676 - val_loss: 1.1627 - val_accurac
y: 0.5805
Accuracy: 58.05%
```

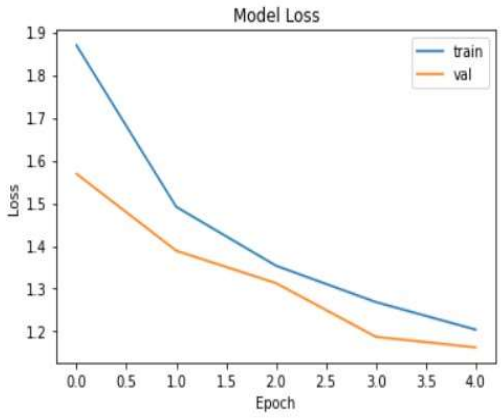
2. Predict the first 4 images of the test data using the above model. Then, compare with the actual label for those 4 images to check whether or not the model has predicted correctly.

```
import numpy as np
predictions = model.predict(X_test[:4])
predicted_labels = np.argmax(predictions, axis=1)
actual_labels = np.argmax(y_test[:4], axis=1)

print("Predicted labels:", predicted_labels)
print("Actual labels:   ", actual_labels)
```

```
1/1 [==============================] - 0s 25ms/step
Predicted labels: [3 1 8 0]
Actual labels:    [3 8 8 0]
```

3. Visualize Loss and Accuracy using the history object

```
]: import matplotlib.pyplot as plt

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()
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()
```



GITHUB REPO LINK: https://github.com/maddalareshma/NNDL-ICP3