NNDL ASSIGNMENT- 07

Praveena Goli (700743010)

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?

```
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
np.random.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), 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'))


sgd = SGD(learning_rate=0.01, momentum=0.9, decay=1e-6)

model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])

print(model.summary())

epochs = 5

batch_size = 32

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


scores = model.evaluate(X_test, y_test, verbose=0)

print("Accuracy: %.2f%%" % (scores[1]*100))

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]

# 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(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(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'))


# 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)

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

# 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)
```

Did the performance change?

The performance of the model is likely to improve with the addition of more layers and higher number of feature maps, but it will also increase the complexity and the training time of the model. The new model architecture provided in the instruction includes several new layers and higher number of feature maps, which may improve the accuracy of the model.

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.

```
# Print the predicted and actual labels for the first 4 images
print("Predicted labels:", predicted_labels)
print("Actual labels:   ", actual_labels)
```

```
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()
```

3. Visualize Loss and Accuracy using the history object.

```
# 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()
```

GIT HUB LINK : https://github.com/Goli18/NNDL_ASS7.git

VIDEO LINK: https://github.com/Goli18/NNDL_ASS7/blob/main/icp7_nndl.mp4