Fall 2023: CS5720

Neural Networks & Deep Learning - ICP-7

Name: Manisha Lakkarsu
Student Id: 700746573
Git link:
Video link: https://github.com/Mani543/Manisha_NNDL_ICP7.git
https://drive.google.com/file/d/12xBVJj3eHGERlQM_jAQhBjFEVfcIVUxL/view?usp=sharing

In class programming:

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.

In [12]:

```
import numpy as np
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.constraints import MaxNorm
from tensorflow.keras.utils import to_categorical
from keras.optimizers import SGD

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

# Load data
(a_train, b_train), (a_test, b_test) = cifar10.load_data()

# Normalize inputs from 0-255 to 0.0-1.0
a_train = a_train.astype('float32') / 255.0
a_test = a_test.astype('float32') / 255.0

# One hot encode outputs
b_train = to_categorical(b_train)
b_test = to_categorical(b_test)
num_classes = b_test.shape[1]

# 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))
```

```
# 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_rate = lRate / epochs
sgd = SGD(learning_rate=lRate, momentum=0.9, nesterov=False)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
print(model.summary())
```

```
# Fit the model
history = model.fit(a_train, b_train, validation_data=(a_test, b_test), epochs=epochs, batch_size=32)

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

Output:

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_18 (Conv2D)          (None, 32, 32, 32)        896

 dropout_18 (Dropout)        (None, 32, 32, 32)        0

 conv2d_19 (Conv2D)          (None, 32, 32, 32)        9248

 max_pooling2d_9 (MaxPoolin  (None, 16, 16, 32)        0
 g2D)

 conv2d_20 (Conv2D)          (None, 16, 16, 64)        18496

 dropout_19 (Dropout)        (None, 16, 16, 64)        0

 conv2d_21 (Conv2D)          (None, 16, 16, 64)        36928

 max_pooling2d_10 (MaxPooli  (None, 8, 8, 64)          0
 ng2D)

 conv2d_22 (Conv2D)          (None, 8, 8, 128)         73856

 dropout_20 (Dropout)        (None, 8, 8, 128)         0

 conv2d_23 (Conv2D)          (None, 8, 8, 128)         147584

 max_pooling2d_11 (MaxPooli  (None, 4, 4, 128)         0
 ng2D)

 flatten_3 (Flatten)         (None, 2048)              0

 dropout_21 (Dropout)        (None, 2048)              0

 dense_9 (Dense)             (None, 1024)              2098176

 dropout_22 (Dropout)        (None, 1024)              0

 dense_10 (Dense)            (None, 512)               524800

 dropout_23 (Dropout)        (None, 512)               0

 dense_11 (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 [==============================] - 216s 137ms/step - loss: 1.8654 - accuracy: 0.3102 - val_loss: 1.5189 - val_accu
racy: 0.4385
Epoch 2/5
1563/1563 [==============================] - 221s 141ms/step - loss: 1.4289 - accuracy: 0.4789 - val_loss: 1.2839 - val_accu
racy: 0.5338
```

```
Epoch 1/5
1563/1563 [==============================] - 216s 137ms/step - loss: 1.8654 - accuracy: 0.3102 - val_loss: 1.5189 - val_accu
racy: 0.4385
Epoch 2/5
1563/1563 [==============================] - 221s 141ms/step - loss: 1.4289 - accuracy: 0.4789 - val_loss: 1.2839 - val_accu
racy: 0.5338
Epoch 3/5
1563/1563 [==============================] - 245s 157ms/step - loss: 1.2073 - accuracy: 0.5655 - val_loss: 1.0837 - val_accu
racy: 0.6123
Epoch 4/5
1563/1563 [==============================] - 250s 160ms/step - loss: 1.0507 - accuracy: 0.6275 - val_loss: 1.0062 - val_accu
racy: 0.6406
Epoch 5/5
1563/1563 [==============================] - 234s 150ms/step - loss: 0.9395 - accuracy: 0.6662 - val_loss: 0.9356 - val_accu
racy: 0.6768
Accuracy: 67.68%
```

- Did the performance change?
    - With the addition of more layers and feature maps, the model's performance is likely to improve, but the complexity and training time of the model will also rise. The new model architecture described in the instruction has more feature maps and new layers, which could increase the model's accuracy.

In class programming:

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 Image Classification on the handwritten digits data set (mnist)

In [13]:
```
# Predict the first 4 images of the test data
predictions = model.predict(a_test[:4])

# Convert the predictions to class labels
predicted_labels = np.argmax(predictions, axis=1)

# Convert the actual labels to class labels
actual_labels = np.argmax(b_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 450ms/step
Predicted labels: [3 1 8 0]
Actual labels: [3 8 8 0]
```

In class programming:

3. Visualize Loss and Accuracy using the history object.

In [15]:
```
import matplotlib.pyplot as pplt

# Plot the training and validation loss
pplt.figure(figsize=(12, 6))
pplt.subplot(1, 2, 1)
pplt.plot(history.history['loss'], label='Training Loss')
pplt.plot(history.history['val_loss'], label='Validation Loss')
pplt.xlabel('Epoch')
pplt.ylabel('Loss')
pplt.title('Training and Validation Loss')
pplt.legend()

# Plot the training and validation accuracy
pplt.subplot(1, 2, 2)
pplt.plot(history.history['accuracy'], label='Training Accuracy')
pplt.plot(history.history['val_accuracy'], label='Validation Accuracy')
pplt.xlabel('Epoch')
pplt.ylabel('Accuracy')
pplt.title('Training and Validation Accuracy')
pplt.legend()

pplt.tight_layout()
pplt.show()
```

Output: