Spring 2025: CS5720

Neural Networks and Deep Learning - ICP-5

Gonaboyina Vijay Vardhan (700755141)

Github link:https://github.com/Vijayvardhan02/NEURAL-NETWORK-DEEP-LEARNING-ICP5

Video link:

https://drive.google.com/file/d/1accMtlVqds5gahCCb5RBXJyU6qKDFhE1/view?usp=drive_link

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?

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.**
3. **Visualize Loss and Accuracy using the history object**

Code:

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

# Load the dataset (e.g., CIFAR-10)
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()

# Normalize pixel values to between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0

# Create the model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.Dropout(0.2),
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Dropout(0.2),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.Dropout(0.2),
    layers.Conv2D(128, (3, 3), activation='relu'),
    # Remove one max pooling layer here to prevent dimension collapse
    layers.Flatten(),
    layers.Dropout(0.2),
    layers.Dense(1024, activation='relu'),
    layers.Dropout(0.2),
```

```
    layers.Dense(512, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(10, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test))

# Evaluate the model on the test data
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)

print(f"Test accuracy: {test_acc}")
print(f"Test loss: {test_loss}")

# Predict the first 4 images
predictions = model.predict(x_test[:4])
for i, prediction in enumerate(predictions):
    print(f"Predicted label for image {i+1}: {prediction.argmax()} (Actual label: {y_test[i][0]})")

# Visualizing the loss and accuracy using the history object
plt.figure(figsize=(12, 6))
 # Plotting loss
 plt.subplot(1, 2, 1)
 plt.plot(history.history['loss'], label='Training Loss')
 plt.plot(history.history['val_loss'], label='Validation Loss')
 plt.title('Loss during training')
 plt.xlabel('Epochs')
 plt.ylabel('Loss')
 plt.legend()

 # Plotting accuracy
 plt.subplot(1, 2, 2)
 plt.plot(history.history['accuracy'], label='Training Accuracy')
 plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
 plt.title('Accuracy during training')
 plt.xlabel('Epochs')
 plt.ylabel('Accuracy')
 plt.legend()

 plt.show()
```

Output:

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 208s 130ms/step - accuracy: 0.2766 - loss: 1.9018 - val_accuracy: 0.4703 - val_loss: 1.4382
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 261s 130ms/step - accuracy: 0.4907 - loss: 1.4007 - val_accuracy: 0.5250 - val_loss: 1.3330
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 208s 133ms/step - accuracy: 0.5668 - loss: 1.2128 - val_accuracy: 0.6161 - val_loss: 1.0945
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 260s 132ms/step - accuracy: 0.6134 - loss: 1.1116 - val_accuracy: 0.6393 - val_loss: 1.0429
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 259s 130ms/step - accuracy: 0.6402 - loss: 1.0292 - val_accuracy: 0.6459 - val_loss: 0.9990
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 265s 132ms/step - accuracy: 0.6648 - loss: 0.9637 - val_accuracy: 0.6754 - val_loss: 0.9467
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 201s 128ms/step - accuracy: 0.6850 - loss: 0.9187 - val_accuracy: 0.6858 - val_loss: 0.9070
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 206s 131ms/step - accuracy: 0.6948 - loss: 0.8886 - val_accuracy: 0.7090 - val_loss: 0.8535
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 199s 127ms/step - accuracy: 0.7054 - loss: 0.8579 - val_accuracy: 0.7075 - val_loss: 0.8560
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 205s 129ms/step - accuracy: 0.7176 - loss: 0.8224 - val_accuracy: 0.7132 - val_loss: 0.8441
313/313 - 9s - 29ms/step - accuracy: 0.7132 - loss: 0.8441
Test accuracy: 0.7131999731063843
Test loss: 0.8440958857536316
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 163ms/step
Predicted label for image 1: 3 (Actual label: 3)
Predicted label for image 2: 8 (Actual label: 8)
Predicted label for image 3: 8 (Actual label: 8)
Predicted label for image 4: 0 (Actual label: 0)
```