ML_Assignment2_Presentation.pptx

by Ma Chue

Submission date: 16-Jul-2023 10:51AM (UTC-0400)
Submission ID: 2131850971
File name: 17820_Ma_Chue_ML_Assignment2_Presentation_809655_324215170.pptx (939.54K)
Word count: 641
Character count: 3538

Neural Network Models for Object Recognition using CIFAR-10 dataset

A brief overview of the task: Building a Convolutional Neural Network (CNN) for object recognition using the CIFAR-10 dataset.

Dataset and Validation Set Creation

- Load data from CIFAR-10 dataset which include 60,000 32x32 colour images in 10 classes, 6000 per class
- First split data into training and test datasets.
- Normalize images data to be between 0 or 1 by dividing with 255.
- Split full training dataset into 80% training and 20% validation datasets with 0 random state.

```
[1] # Import necessary libraries
    import tensorflow as tf
    import matplotlib.pyplot as plt

    # Load the CIFAR-10 dataset from tensorflow datasets
    data = tf.keras.datasets.cifar10

    # Split the dataset into training and testing sets
    (train_images_full, train_labels_full), (test_images, test_labels) = data.load_data()

[2] # Normalize the images to be between 0 and 1
    train_images_full, test_images = train_images_full / 255.0, test_images / 255.0

    # Split the full training set into a validation set and a (smaller) training set
    from sklearn.model_selection import train_test_split
    train_images, val_images, train_labels, val_labels = train_test_split(train_images_full, train_labels_full, test_size=0.2, random_state=0)
```

Figure 1: Loading and Splitting Datasets

Model Architecture

- We applied Convolutional Neural Network (CNN) which is ideal for image-based tasks to the model.
- The model includes six convolutional layers with 32 ,64,128 filters of size(3,3), and followed by batch normalization, max pooling, and dropout layers.
- The flattened layer is added to convert 3D tensor output to 1D tensor output.
- Finally added a fully connected layer with 128 units and an output layer with 10 units.

```python
# Define the model architecture using a Sequential model
model = tf.keras.models.Sequential([
    # First convolutional layer with 32 filters, followed by Batch Normalization
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(32, 32, 3)),
    tf.keras.layers.BatchNormalization(),

    # Second convolutional layer with 32 filters, followed by Batch Normalization and Max Pooling
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Dropout(0.25),

    # Repeat the same pattern but with 64 filters
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Dropout(0.3),

    # Repeat the same pattern but with 128 filters
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Flatten(),

    # Fully connected layer with 128 units, followed by Batch Normalization and Dropout
    tf.keras.layers.Dense(128, activation='relu', kernel_initializer='he_uniform'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),

    # Output layer with 10 units (for the 10 classes)
    tf.keras.layers.Dense(10, activation='softmax')])
```

Figure 2: CNN Model Architecture

Activation and Loss Functions

- We use Rectified Linear Unit (ReLU) activation function in the convolutional and dense layers.
- For the initializing weight of the neurons, we use "he_uniform" kernel initializer to minimize the issues of dead neurons
- In the output layer, we use softmax activation function for the purpose of multi-class classification.
- As for the loss function, we use "sparse_categorical_crossentropy" which is suitable for multi-class classification tasks.

```
# Repeat the same pattern but with 128 filters
tf.keras.layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'),
tf.keras.layers.BatchNormalization(),
tf.keras.layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'),
tf.keras.layers.BatchNormalization(),
tf.keras.layers.MaxPooling2D(2, 2),
tf.keras.layers.Dropout(0.4),
tf.keras.layers.Flatten(),

# Fully connected layer with 128 units, followed by Batch Normalization and Dropout
tf.keras.layers.Dense(128, activation='relu', kernel_initializer='he_uniform'),
tf.keras.layers.BatchNormalization(),
tf.keras.layers.Dropout(0.5),

# Output layer with 10 units (for the 10 classes)
tf.keras.layers.Dense(10, activation='softmax')])
```

Figure 3: Activation Functions

```
[ ] # Compile the model with Adam optimizer and sparse categorical crossentropy loss function
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Figure 4: Loss Function

Training the Model

- We trained model for 200 epochs
- Finally we validate the accuracy of our model by using test data.

```
Epoch 1/200
1250/1250 [==============================] - 24s 8ms/step - loss: 1.6420 - accuracy: 0.4336 - val_loss: 1.2198 - val_accuracy: 0.5627
Epoch 2/200
1250/1250 [==============================] - 9s 7ms/step - loss: 1.1373 - accuracy: 0.5968 - val_loss: 1.0235 - val_accuracy: 0.6364
Epoch 3/200
1250/1250 [==============================] - 9s 7ms/step - loss: 0.9717 - accuracy: 0.6587 - val_loss: 0.8819 - val_accuracy: 0.6852
Epoch 4/200
1250/1250 [==============================] - 9s 7ms/step - loss: 0.8547 - accuracy: 0.7043 - val_loss: 0.7376 - val_accuracy: 0.7404
Epoch 5/200
1250/1250 [==============================] - 9s 7ms/step - loss: 0.7970 - accuracy: 0.7255 - val_loss: 0.7474 - val_accuracy: 0.7376
Epoch 6/200
1250/1250 [==============================] - 9s 7ms/step - loss: 0.7229 - accuracy: 0.7525 - val_loss: 0.6300 - val_accuracy: 0.7825
Epoch 7/200
1250/1250 [==============================] - 9s 7ms/step - loss: 0.6713 - accuracy: 0.7711 - val_loss: 0.6556 - val_accuracy: 0.7747
Epoch 8/200
1250/1250 [==============================] - 9s 7ms/step - loss: 0.6252 - accuracy: 0.7864 - val_loss: 0.5769 - val_accuracy: 0.8020
Epoch 9/200
```

```
1250/1250 [==============================] - 9s 7ms/step - loss: 0.1066 - accuracy: 0.9632 - val_loss: 0.5233 - val_accuracy: 0.8721
Epoch 195/200
1250/1250 [==============================] - 9s 7ms/step - loss: 0.1062 - accuracy: 0.9628 - val_loss: 0.5164 - val_accuracy: 0.8738
Epoch 196/200
1250/1250 [==============================] - 9s 7ms/step - loss: 0.1012 - accuracy: 0.9651 - val_loss: 0.5095 - val_accuracy: 0.8762
Epoch 197/200
1250/1250 [==============================] - 9s 7ms/step - loss: 0.1051 - accuracy: 0.9641 - val_loss: 0.5224 - val_accuracy: 0.8736
Epoch 198/200
1250/1250 [==============================] - 9s 7ms/step - loss: 0.0991 - accuracy: 0.9653 - val_loss: 0.5453 - val_accuracy: 0.8717
Epoch 199/200
1250/1250 [==============================] - 9s 7ms/step - loss: 0.1055 - accuracy: 0.9648 - val_loss: 0.5385 - val_accuracy: 0.8705
Epoch 200/200
1250/1250 [==============================] - 9s 7ms/step - loss: 0.1043 - accuracy: 0.9641 - val_loss: 0.4961 - val_accuracy: 0.8789
```

Figure 5: Training Model with 200 Epochs

Neural Network Design

- Usage of Keras library with TensorFlow backend for ease of use and flexibility.
- Choice of CNN [12] for image classification tasks due to its ability to capture spatial relationships.
- Implementation of batch normalization and dropout to improve model performance and control overfitting.
- Usage of Adam optimizer for efficient gradient descent.

Accuracy Analysis

- The model achieved a high test accuracy of approximately 87.02%.
- The validation accuracy 87.96% was lower than training accuracy 96.53%, indicating that the model was able to generalize well to unseen data.
- Overfitting was controlled by using dropout layers and early stopping, which helped maintain a balance between the training and validation accuracies.
- The slight difference between the training and validation accuracies indicates that the model has learned the underlying patterns in the data well, without memorizing the training data. This is a good sign of a well-performing model.

```
[9] test_loss, test_acc = model.evaluate(test_images, test_labels, verbose = 2)

313/313 - 1s - loss: 0.5214 - accuracy: 0.8702 - 722ms/epoch - 2ms/step

print("Max Training Accuracy:", round(max(history.history['accuracy']), 4))
print("Max Validation Accuracy:", round(max(history.history['val_accuracy']), 4))
print("Min Loss:", round(max(history.history['loss']), 4))
print("Min Validation Loss:", round(max(history.history['val_loss']), 4))

Max Training Accuracy: 0.9653
Max Validation Accuracy: 0.8796
Min Loss: 1.642
Min Validation Loss: 1.2198
```

Figure 6: Accuracy and Loss Values

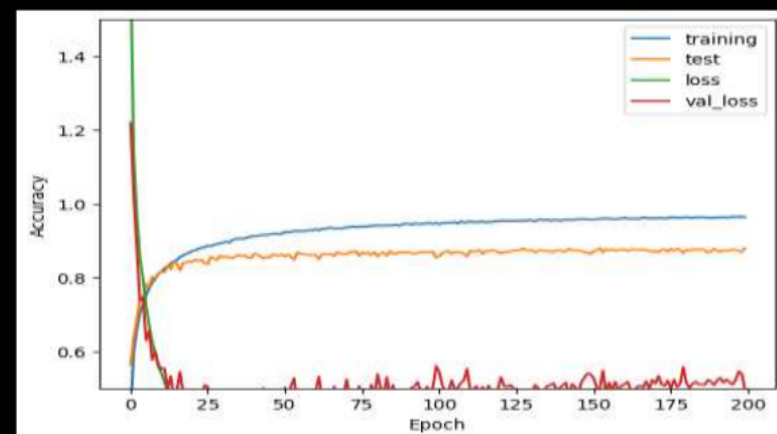

Figure 7: Plot Diagram of Accuracy and Loss Value Comparison

Conclusion

- A deeper understanding of CNNs and their application in object recognition was gained through this exercise.
- Importance of a validation set and techniques to improve model performance. (Change the values, add functions as appropriate.)
- Successful implementation of a CNN for object recognition with good accuracy.

References

- Krizhevsky, A., Nair, V. and Hinton, G., n.d. CIFAR-10 (Canadian Institute for Advanced Research). [online] Available at: https://www.cs.toronto.edu/~kriz/cifar.html [Accessed: 14 July 2023].
- Goodfellow, I., Bengio, Y. and Courville, A., 2016. Deep Learning. MIT Press. Available at: http://www.deeplearningbook.org/ [Accessed: 14 July 2023].
- LeCun, Y., Bengio, Y. and Hinton, G., 2015. Deep learning. Nature, 521(7553), pp.436–444. Available at: https://doi.org/10.1038/nature14539 [Accessed: 14 July 2023].
- Prechelt, L., 1998. Early Stopping - But When? In: G. Orr and K.-R. Müller, eds., Neural Networks: Tricks of the Trade, pp.55–69. Springer. Available at: https://doi.org/10.1007/3-540-49430-8_3 [Accessed: 14 July 2023].
- LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P., 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), pp.2278–2324. Available at: https://doi.org/10.1109/5.726791 [Accessed: 14 July 2023].

ML_Assignment2_Presentation.pptx

ORIGINALITY REPORT

37%	27%	26%	26%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	etheses.whiterose.ac.uk Internet Source	9%
2	research.aota.org Internet Source	5%
3	Submitted to City University Student Paper	4%
4	Submitted to Queen's University of Belfast Student Paper	3%
5	Submitted to Oxford Brookes University Student Paper	3%
6	Submitted to Berlin School of Business and Innovation Student Paper	2%

7	Submitted to Intercollege Student Paper	2%
8	pubs.aip.org Internet Source	2%
9	repositorio.itm.edu.co Internet Source	2%
10	web.archive.org Internet Source	1%
11	www.frontiersin.org Internet Source	1%
12	www.mitpressjournals.org Internet Source	1%
13	Milad Rayka, Ali Mohammad Latifi, Morteza Mirzaei. "Assisting In-Silico Drug Discovery Through Protein-Ligand Binding Affinity Prediction By Convolutional Neural Networks", Research Square Platform LLC, 2023 Publication	1%
14	Jinhyeok Jang, Hyunjoong Cho, Jaehong Kim, Jaeyeon Lee, Seungjoon Yang. "Deep neural networks with a set of node-wise varying activation functions", Neural Networks, 2020	1%

Publication

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		