Department of Information Engineering Technology

The Superior University, Lahore

Artificial Intelligence Lab

Experiment No.12

Practice the Image Classification using CNN and Data Augmentation to Address Overfitting

Prepared for

_____

By:

Name: _____

ID: _____

Section: _____

Semester: _____

Total Marks: _____

Obtained Marks: _____

Signature: _____

Date: _____

Experiment No. 12

Practice the Image Classification using CNN and Data Augmentation to Address Overfitting.
(Rubrics)

Name: _____ **Roll No. : _____**

A. PSYCHOMOTOR

Sr. No.	Criteria	Allocated Marks	Unacceptable 0%	Poor 25%	Fair 50%	Good 75%	Excellent 100%	Total Obtained
1	Follow Procedures	1	0	0.25	0.5	0.75	1	
2	Software and Simulations	2	0	0.5	1	1.5	2	
3	Accuracy in Output Results	3	0	0.75	1.5	2.25	3	
Sub Total		**6**	**Sub Total marks Obtained in Psychomotor(P)**					

B. AFFECTIVE

Sr. No.	Criteria	Allocated Marks	Unacceptable 0%	Poor 25%	Fair 50%	Good 75%	Excellent 100%	Total Obtained
1	Respond to Questions	1	0	0.25	0.5	0.75	1	
2	Lab Report	1	0	0.25	0.5	0.75	1	
3	Assigned Task	2	0	0.5	1	1.5	2	
Sub Total		**4**	**Sub Total marks Obtained in Affective (A)**					

Instructor Name:_____ **Total Marks (P+A): _____**

Instructor Signature:_____ **Obtained Marks (P+A): _____**

Introduction to Convolutional Neural Network

A Convolutional Neural Network (CNN) is a specialized type of deep neural network designed for processing and analyzing visual data, such as images and videos. CNNs have proven to be highly effective in tasks related to computer vision, including image classification, object detection, and image recognition.

The architecture of a CNN is inspired by the visual processing that occurs in the human brain. It consists of three main types of layers: convolutional layers, pooling layers, and fully connected layers.

Convolutional Layers:

- The core building blocks of a CNN are convolutional layers. These layers apply convolutional operations to the input data, allowing the network to learn spatial hierarchies of features.
- Convolution involves sliding a small filter (also called a kernel) over the input image and computing the dot product at each position. This helps detect patterns like edges, textures, and more complex structures.

Pooling Layers:

- After convolution, pooling layers are typically added to reduce the spatial dimensions of the input volume. Max pooling is a common pooling operation, where the maximum value in a local region of the input is retained, effectively down-sampling the data.
- Pooling helps reduce the computational complexity and the number of parameters in the network while retaining the most important information.

Fully Connected Layers:

- The final layers of a CNN are fully connected layers, which connect every neuron in one layer to every neuron in the next layer. These layers combine the local features learned by the convolutional and pooling layers to make high-level predictions.
- In the case of image classification, the fully connected layers often output probabilities for different classes.

Activation Functions:

- Non-linear activation functions, such as ReLU (Rectified Linear Unit), are applied after convolutional and fully connected layers to introduce non-linearity into the model. This enables the network to learn complex relationships in the data.

Flattening:

- Before entering the fully connected layers, the output from the convolutional and pooling layers is typically flattened into a vector. This allows the network to process the spatial information learned in previous layers.

CNNs are trained using labeled datasets through a process called backpropagation, where the network adjusts its weights to minimize the difference between its predictions and the true labels. Transfer learning is also commonly used with CNNs, where pre-trained models on large datasets (like ImageNet) are fine-tuned for specific tasks.

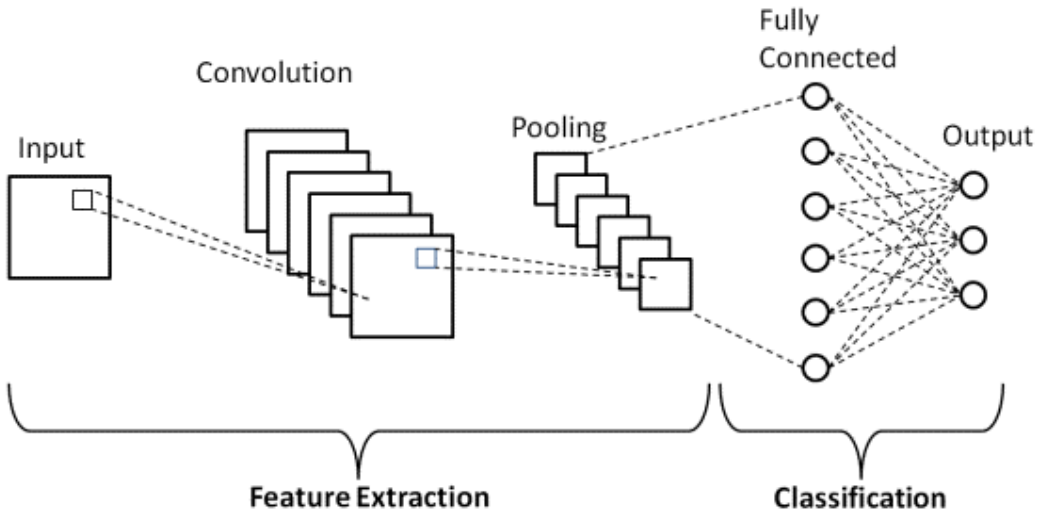

Figure 1: CNN Architecture

Introduction to Data Augmentation

Data augmentation is a technique commonly used in machine learning and computer vision to artificially increase the size of a training dataset by applying various transformations to the original data. The goal of data augmentation is to enhance the generalization and robustness of a machine learning model by exposing it to a wider variety of training examples.

Here are key points in the introduction to data augmentation:

Purpose:

- Increased Dataset Size: Data augmentation helps in creating a larger and more diverse dataset without collecting additional data. This is especially useful when the original dataset is limited in size.
- Improved Generalization: Exposure to augmented data helps the model generalize better to variations and improves its ability to handle unseen examples.

Common Data Augmentation Techniques:

Image Data:

- Rotation: Rotate images by a certain angle.
- Flip: Flip images horizontally or vertically.
- Zoom: Zoom in or out of images.
- Crop: Crop a portion of the image.
- Brightness and Contrast Adjustments: Modify the brightness and contrast of images.

Text Data:

- Random Insertion/Deletion: Introduce noise by randomly adding or removing words.

- Synonym Replacement: Replace words with their synonyms.
- Audio Data:
- Pitch Shift: Change the pitch of audio signals.
- Time Stretch: Modify the speed of audio signals.

Implementation:

- Data augmentation is typically applied during the training phase. Original training samples are randomly transformed before being fed into the model.
- In deep learning frameworks, libraries often provide built-in functions for applying data augmentation, making it convenient for practitioners to integrate into their workflows.

Benefits:

- Regularization: Data augmentation serves as a form of regularization, preventing overfitting by exposing the model to a more extensive range of input variations.
- Improved Robustness: Models trained with augmented data are more robust to variations, noise, and distortions present in real-world scenarios.

Considerations:

- The choice of augmentation techniques depends on the nature of the data and the problem domain.
- Extreme augmentations may introduce unrealistic examples, so the practitioner needs to balance diversity with relevance.

Data augmentation is widely used in various domains, including image classification, object detection, natural language processing, and audio analysis. It has proven to be an effective tool for enhancing the performance of machine learning models, particularly when working with limited training data.

BUILDING A MODEL

Through Artificial Neural Network (ANN)

Import the libraries

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

This part of the code loads the CIFAR-10 dataset using TensorFlow and Keras. The dataset consists of 50,000 training images and 10,000 test images, each belonging to one of the 10 classes. X_train and X_test are arrays of images, and y_train and y_test are arrays of labels.

```
(X_train, y_train), (X_test,y_test) = datasets.cifar10.load_data()
X_train.shape
X_test.shape
y_train.shape
y_train[:5]
```

Here, y_train and y_test are initially 2D arrays, and reshape(-1,) is used to convert them into 1D arrays. The classes array is created to map the numerical labels to their corresponding class names.

```
# Reshape the label arrays to 1D
y_train = y_train.reshape(-1,)
y_train[:5]
y_test = y_test.reshape(-1,)
# Define the class labels
classes =
["airplane","automobile","bird","cat","deer","dog","frog","horse","ship"
,"truck"]
```

The plot_sample function is defined to visualize sample images with their corresponding labels. Two sample images are then plotted using this function.

```
def plot_sample(X, y, index):
    plt.figure(figsize = (15,2))
    plt.imshow(X[index])
    plt.xlabel(classes[y[index]])
plot_sample(X_train, y_train, 0)
plot_sample(X_train, y_train, 1)
```

This part of the code normalizes the pixel values of the images to be in the range [0, 1]. This is a common preprocessing step in neural network models.

```
X_train = X_train / 255.0
X_test = X_test / 255.0
```

This defines a simple Artificial Neural Network (ANN) using the Sequential API from Keras. It consists of a flattening layer to convert the 3D image data into a 1D array, followed by two dense (fully connected) hidden layers with ReLU activation functions, and an output layer with softmax activation for multiclass classification. The model is then compiled with the stochastic gradient descent ('SGD') optimizer, sparse categorical crossentropy loss, and accuracy as the metric. Finally, the model is trained on the training data for 5 epochs.

```
ann = models.Sequential([
        layers.Flatten(input_shape=(32,32,3)),
        layers.Dense(3000, activation='relu'),
```

```
        layers.Dense(1000, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])

ann.compile(optimizer='SGD',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

ann.fit(X_train, y_train, epochs=5)
```

This classification report provides a more detailed evaluation of the model's performance on each class in addition to the overall metrics. It's a useful tool to understand how well the model is doing for different categories in our classification problem.

```
from sklearn.metrics import confusion_matrix , classification_report
import numpy as np
y_pred = ann.predict(X_test)
y_pred_classes = [np.argmax(element) for element in y_pred]

print("Classification Report: \n", classification_report(y_test,
y_pred_classes))
```

The reported accuracy at the end of the training is around 49%, indicating how well the model is performing on the training set. Depending on our task, we might need to adjust hyperparameters, increase training epochs, or consider more complex models for better performance on the test set.

Through Convolutional Neural Network (CNN)

Now let us build a convolutional neural network to train our images

Convolutional Layers:

- Two convolutional layers with 32 and 64 filters, respectively. Each uses a 3x3 kernel and ReLU activation.
- After each convolutional layer, there's a MaxPooling layer with a 2x2 pool size to downsample the spatial dimensions.

Flatten and Dense Layers:

- Flatten layer to convert the 3D output to 1D.
- Dense layers with ReLU activation for higher-level features.

- Output layer with 10 units and softmax activation for classification.

Compilation with the Adam optimizer, sparse categorical crossentropy loss, and accuracy as the metric.

Training the CNN on the training data for 10 epochs.

```
cnn = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu',
input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
cnn.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
cnn.fit(X_train, y_train, epochs=10)
```

With CNN, at the end 5 epochs, accuracy was at around 70% which is a significant improvement over ANN. CNNs are best for image classification and gives superb accuracy. Also, computation is much less compared to simple ANN as max-pooling reduces the image dimensions while still preserving the features

Evaluating the model on the test set to get the loss and accuracy.

```
cnn.evaluate(X_test,y_test)
```

Using the trained CNN to predict classes for the test set and obtaining the predicted classes.

```
y_pred = cnn.predict(X_test)
y_pred[:5]
y_classes = [np.argmax(element) for element in y_pred]
y_classes[:5]
y_test[:5]
```

Visualizing a sample image from the test set and printing its predicted class.

```
plot_sample(X_test, y_test,3)
classes[y_classes[3]]
```

The CNN architecture we've defined is more suitable for image data, and we can observe a significant improvement in accuracy compared to the simple Artificial Neural Network (ANN) we initially created. The convolutional layers capture spatial hierarchies and reduce the spatial dimensions, making them powerful for image-related tasks.

Task:

Use CNN to do handwritten digits classification using MNIST dataset. Above we used ANN for digits classification. You need to modify this code to use CNN instead. Check how accuracy improves fast with CNN and figure out how CNN can be a better choice for doing image classification compared to ANN.