Machine Learning for Computer Vision

TD1: Image Preprocessing and Classification

Département d'informatique

Ruiwen HE

Course Review: Image Classification

1. **Definition:** Image classification is a task in computer vision where the goal is to categorize an input image into predefined classes. For instance, classifying an animal image as a cat, dog, or other.
2. **Application Areas:**
   - Medical diagnostics: Disease detection in medical images.
   - Autonomous vehicles: Object recognition on the road (cars, pedestrians).
   - Facial recognition: Recognizing individuals through facial features.
3. **Popular Datasets:**
   - MNIST: Handwritten digits.
   - CIFAR-10: 60,000 small color images in 10 classes.
   - ImageNet: Over 14 million labeled images for 1,000 classes.
4. **Preprocessing:**
   - Image resizing: Upsampling or downsampling to ensure consistency.
   - Standardization: Min-Max scaling, Z-score normalization.
5. **Image Augmentation:** Techniques like rotation, cropping, flipping to create more training data and prevent overfitting.
6. **Feature Extraction:**
   - Traditional methods: Sobel, SIFT, HOG.
   - Deep learning: Using CNNs to automatically learn hierarchical features.
7. **Convolutional Neural Networks (CNNs):**
   - Convolutional layers: Extract features with filters.
   - Pooling layers: Dimensionality reduction.
   - Fully connected layers: Mapping features to class labels.
8. **Model Training:** Using loss functions like cross-entropy, with optimizers such as SGD and Adam.
9. **Model Evaluation:** Key metrics include accuracy, precision, recall, and confusion matrix for analyzing model performance.
10. **Challenges:** Intra-class variability and inter-class similarity complicate classification.

Part I: Introduction to Image Classification with CNNs

Brief Recap of CNNs and Image Classification

- Review the key components of CNNs (convolutional layers, pooling layers, fully connected layers).
- Explain the workflow of image classification using CNNs.
- Overview of datasets like CIFAR-10, MNIST.

Exercise 1: Dataset Exploration and Preprocessing

- **Goal**: Load and explore a dataset, preprocess images for training.
- **Task**:
  - Load a dataset (e.g., CIFAR-10 or MNIST).
  - Normalize image pixel values between 0 and 1.
  - Split the dataset into training, validation, and test sets.
- **Code Example**:

```python
    # Import libraries
    from tensorflow.keras.datasets import cifar10
    (x_train, y_train), (x_test, y_test) = cifar10.load_data()

    # Normalize the dataset
    x_train = x_train.astype('float32') / 255.0
    x_test = x_test.astype('float32') / 255.0

    # Check dataset shapes
    print(f"Training data shape: {x_train.shape}")
    print(f"Test data shape: {x_test.shape}")
```

- **Discussion**: Why do we normalize images, and how does data preprocessing improve model performance?

Exercise 2: Image Preprocessing with OpenCV

- **Goal**: Use OpenCV to preprocess images (resize, apply transformations).
- **Task**:
  - Load an image using OpenCV.
  - Resize the image to the appropriate dimensions.
  - Apply basic transformations such as rotation and flipping.
- **Code Example**:

```python
import cv2
import numpy as np

# Load an image
img = cv2.imread('path_to_image.jpg')

# Resize the image
img_resized = cv2.resize(img, (32, 32))

# Rotate the image by 90 degrees
img_rotated = cv2.rotate(img_resized, cv2.ROTATE_90_CLOCKWISE)

# Flip the image horizontally
img_flipped = cv2.flip(img_rotated, 1)

# Display the processed images
cv2.imshow('Resized Image', img_resized)
cv2.imshow('Rotated Image', img_rotated)
cv2.imshow('Flipped Image', img_flipped)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- **Discussion**: What are the benefits of using OpenCV for image preprocessing? How do transformations affect model training?

Exercise 3: Image Classification using HOG and Machine Learning

- **Goal**: Use HOG (Histogram of Oriented Gradients) to extract features from images and apply a traditional machine learning algorithm (e.g., SVM) for classification.
- **Task**:
  - Use OpenCV or `skimage` to extract HOG features from images.
  - Train an SVM classifier using the extracted HOG features.
  - Evaluate the classifier on a test dataset.
- **Code Example**:

```
from skimage.feature import hog
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load dataset (e.g., digits dataset from sklearn)
digits = datasets.load_digits()
x = digits.images
y = digits.target

# Extract HOG features for each image
hog_features = [hog(image, pixels_per_cell=(8, 8), cells_per_block
    =(2, 2), visualize=False) for image in x]

# Split the data into training and test sets
x_train, x_test, y_train, y_test = train_test_split(hog_features, y
    , test_size=0.2, random_state=42)

# Train an SVM classifier
clf = SVC(kernel='linear')
clf.fit(x_train, y_train)

# Predict and evaluate the model
y_pred = clf.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Test accuracy: {accuracy}")
```

- **Discussion**: How does the HOG feature extraction process work, and why is it useful for image classification? What are the advantages and limitations of using traditional machine learning methods like SVM compared to deep learning models?

Exercise 4: Build a Simple CNN Model

- **Goal**: Create a CNN architecture using Keras.
- **Task**:
  - Define a simple CNN model with two convolutional layers, followed by pooling, and a fully connected layer.
  - Compile the model using an optimizer and loss function.
- **Code Example**:

```python
from tensorflow.keras import layers, models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
    input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

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

- **Discussion**: What is the role of convolutional layers? How does pooling help reduce computation?

Exercise 5: Train the Model

Exercise 5: Train the Model

- **Goal**: Train the CNN model using the prepared dataset.
- **Task**:
  - Train the model for a few epochs (e.g., 5-10) and plot the loss/accuracy curves.
- **Code Example**:

```python
history = model.fit(x_train, y_train, epochs=5,
    validation_data=(x_test, y_test))

# Plot the training and validation accuracy and loss
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='
    val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
```

- **Discussion**: How do the training and validation curves indicate underfitting or overfitting?

Part II: Advanced CNN Technique

In this section, we will delve into more advanced convolutional neural network (CNN) techniques to enhance model performance and generalization capabilities. Specifically, we will focus on key aspects such as data augmentation, hyperparameter tuning, regularization, and model evaluation. These techniques are crucial for building effective deep learning models.

Exercise 6: Data Augmentation

- **Goal**: Apply data augmentation to the training dataset to enhance model generalization.
- **Task**:
  - Use Keras's `ImageDataGenerator` to apply real-time data augmentation like rotations, flips, zooms, etc.
- **Code Example**:

```python
        from tensorflow.keras.preprocessing.image import
            ImageDataGenerator

        datagen = ImageDataGenerator(
            rotation_range=40,
            width_shift_range=0.2,
            height_shift_range=0.2,
            shear_range=0.2,
            zoom_range=0.2,
            horizontal_flip=True)

        # Fit the data generator to the training data
        datagen.fit(x_train)

        # Train using the augmented data generator
        model.fit(datagen.flow(x_train, y_train, batch_size=32),
            epochs=5)
```

- **Discussion**: How does data augmentation prevent overfitting, and why is it important in deep learning?

Exercise 7: Fine-tuning Hyperparameters

- **Goal**: Experiment with hyperparameters like learning rate, batch size, and number of epochs.
- **Task**:
  - Adjust learning rate and batch size and observe how it affects the model's performance.
  - Use learning rate schedules (e.g., exponential decay) or optimizers like Adam or RMSProp.
- **Code Example**:

```python
        from tensorflow.keras.optimizers import Adam

        # Compile the model with a lower learning rate
        model.compile(optimizer=Adam(learning_rate=0.0001), loss='
            categorical_crossentropy', metrics=['accuracy'])
```

- **Discussion**: How do changes in the learning rate or batch size impact training speed and accuracy?

Exercise 8: Dropout and Regularization

- **Goal**: Introduce regularization techniques like dropout to improve model generalization.
- **Task**:
  - Add dropout layers to the model to randomly disable some neurons during training.
- **Code Example**:

```
        model.add(layers.Dropout(0.5))  # Add dropout layer
```

- **Discussion**: How does dropout help in reducing overfitting? What are other regularization techniques?

Exercise 9: Model Evaluation

- **Goal**: Evaluate the model's performance on the test set.
- **Task**:
  - Compute and display the test accuracy and loss.
  - Use confusion matrix for detailed error analysis.
- **Code Example**:

```
        from sklearn.metrics import confusion_matrix
        import seaborn as sns

        test_loss, test_acc = model.evaluate(x_test, y_test)
        print(f'Test accuracy: {test_acc}')

        # Confusion matrix
        y_pred = model.predict(x_test)
        y_pred_classes = np.argmax(y_pred, axis=1)
        cm = confusion_matrix(y_test, y_pred_classes)
        sns.heatmap(cm, annot=True)
```

- **Discussion**: What insights can be drawn from the confusion matrix? How can it inform further model improvements?

Exercise 10: Model Saving and Deployment

- **Goal**: Save the trained model for future use.
- **Task**:
  - Save the model using Keras's built-in methods.
- **Code Example**:

```
        model.save('my_model.h5')  # Save the model
```

- **Discussion**: What are the different formats for saving models? When and why would you deploy a model?

Part III:Real-World Application

In this section, we will explore the practical applications of convolutional neural networks (CNNs) in real-world scenarios. This part focuses on using advanced techniques such as transfer learning and implementing a complete image classification pipeline.

Exercise 11: Transfer Learning

- **Goal**: Use a pre-trained model (e.g., VGG16, ResNet) for image classification on a custom dataset.
  Task:
  - Load a pre-trained model (without the top layer) and fine-tune it on the CIFAR-10 dataset.
- Code Example:

```python
from tensorflow.keras.applications import VGG16
from tensorflow.keras import layers, models

# Load the pre-trained VGG16 model
base_model = VGG16(weights='imagenet', include_top=False,
    input_shape=(32, 32, 3))
base_model.trainable = False  # Freeze the base model

# Add custom layers for CIFAR-10 classification
model = models.Sequential([
     base_model,
     layers.Flatten(),
     layers.Dense(64, activation='relu'),
     layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy',
    metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5, validation_data=(x_test,
    y_test))
```

- **Discussion**: Why is transfer learning useful, and how can it speed up training for smaller datasets?

Exercise 12: Final Mini Project

- **Goal**: Implement a complete end-to-end image classification pipeline using a dataset of your choice.
- **Task**:
  - Choose a dataset (e.g., a subset of ImageNet, or a custom dataset).
  - Apply all learned techniques (data augmentation, regularization, transfer learning, etc.).
  - Train the model, evaluate performance, and draw conclusions.
- **Optional Extensions**:
  - Experiment with other architectures like ResNet or MobileNet.
  - Compare results with and without transfer learning.

Q&A and Summary

Summary: Review the key concepts learned in this session.

- Transfer learning and its benefits for smaller datasets.
- Building and fine-tuning models using pre-trained networks like VGG16, ResNet, or MobileNet.
- Data augmentation, dropout, and regularization techniques.
- Model evaluation, confusion matrices, and model saving.

Resources for Further Learning:

- Academic papers on transfer learning and CNN architectures.
- Blogs on image classification best practices.
- Additional datasets like ImageNet, Pascal VOC, or custom datasets for experimentation.