Convolutional Neural Network(CNN) Tutorial for Image Classification

1. Introduction

The main success of Convolutional Neural Networks (CNNs) led to image classification becoming a fundamental challenge in computer vision. Our objective in this tutorial is to simplify CNN understanding through an analysis of essential parts and clarifications about their distinctions from traditional learning models followed by practical implementation of two image classifiers. There is a home-built classifier alongside another model that benefits from transfer learning capabilities of VGG16. Our selected "Butterfly Species Classification" dataset serves for originality as it distinguishes our tutorial from standard tutorials based on MNIST or CIFAR-10 datasets.

The learning activity addresses people who understand basic Python and machine learning principles using Jupyter Notebook structure to provide runnable and reproducible code blocks.

2. CNN Architecture Overview

The specialized neural network architecture of CNNs functions with grid-patterned data which includes images. The networks automatically build feature hierarchies through their ability to use local connectivity together with shared parameters.

2.1 Convolution Layer

CNNs comprise the convolution layer as their fundamental functional component. The system uses filters also known as kernels to convolve across the input image for generating feature maps. Filters in CNNs detect particular features among textures, edges and colors.

Key points:

- Each neuron inside the **convolution layer maintains connections** with only a limited area within the input data.
- Different regions use one shared filter to decrease the number of **parameters** required in the network.
- After convolution the network applies a **non-linear function called ReLU** which serves to introduce non-linearity.

2.2 Pooling Layer

By using pooling layers, the spatial dimensions from feature maps decrease which enables both amount of data to be processed more efficiently while aiding the reduction of overfitting.

Common types of pooling:

- **Max Pooling:** Returns the maximum value in each patch of the feature map.
- **Average Pooling:** Returns the average value.

Through the process of pooling the most vital elements remain intact but the map resolution becomes decreased.

2.3 Fully Connected Layer

Many convolutional as well as pooling layers process data before the high-level attributes get reshaped to enter fully connected (dense) layers. The final predictions emerge from combining all features which the convolutional part of the network extracted. In image classification models the last dense layer features neurons which match the number of defined classes.

3. CNN vs. Traditional ML Feature Extraction

Previous to CNNs traditional ML approaches for image classification required engineers to create features using Canny edge detection and color histograms and texture descriptors. Artificially designed features originated from human hands were thereafter presented to SVMs or decision tree classifiers.

Advantages of CNNs:

- The ability to **learn features automatically** by CNNs eliminates any requirement for manual feature extraction from data.
- CNN architecture progresses **higher-level object part detection from lower-level edge** and corner features.
- The **adaptability** of CNNs allows them to perform well on different tasks using few modifications while transfer learning capability provides additional flexibility.

Raw images produce robust features specific for tasks through CNNs which eliminate the necessity for manual feature extraction.

4. Implementing an Image Classification Model

The following section includes the development of two models for butterfly species classification through TensorFlow/Keras implementation. We initiate developing a CNN from the beginning before shifting to making use of a pre-trained VGG16 model that receives additional training on our dataset.

4.1 Data Preparation and Exploration

We retrieve the data from Kaggle before starting model development. This unique dataset contains images of different butterfly species under the name Butterfly Species Classification. The dataset spreads its information across individual directories that correspond to different classes.

Dataset source: https://www.kaggle.com/datasets/gpiosenka/butterfly-images40-species

This code illustration shows how to obtain and process data before model construction.:

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define dataset paths
train_dir = 'data/butterflies/train'
val_dir = 'data/butterflies/validation'

# Create ImageDataGenerators with data augmentation for training
train_datagen = ImageDataGenerator(
    rescale=1./255,
```

```
    rotation_range=30,
    zoom_range=0.2,
    horizontal_flip=True
)

val_datagen = ImageDataGenerator(rescale=1./255)

# Generate batches of image data and labels
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical'
)

val_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical'
)
```

Explanation:
The code adopts ImageDataGenerator from Keras to implement automatic training image augmentations including rotation and zooming and flipping which results in model robustness improvement. The rescaling operation creates pixel value ranges from 0 to 1 that are required for inputting images into CNN networks.

4.2 Building a CNN from Scratch

The following code demonstrates construction of a CNN model in Keras from start to finish:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

# Build a CNN from scratch
cnn_model = Sequential([
    # Convolutional Block 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D(pool_size=(2, 2)),

    # Convolutional Block 2
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),

    # Convolutional Block 3
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),

    # Flatten and Fully Connected Layers
```

```
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(train_generator.num_classes, activation='softmax')
])

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

Explanation:

- The model begins with three convolutional blocks containing each successive block with a convolutional layer followed by a pooling layer.
- After flattening the output, the data enters a regularized fully connected layer.
- A softmax activation layer functions to generate probabilities that match the different butterfly species.
- Multi-class classification tasks commonly utilize adam optimizer together with categorical_crossentropy loss function as their primary choices.

4.3 Using a Pre-Trained Model and Fine-Tuning

Using transfer learning enables the access of pre-trained networks that possess extensive feature representation capabilities. The process involves adopting a pre-trained VGG16 model and securing its base layers while designing new custom layers for a particular classification operation.

```
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import GlobalAveragePooling2D

# Load the pre-trained VGG16 model without the top layers
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(150, 150, 3))
base_model.trainable = False  # Freeze the convolutional base

# Build the transfer learning model
tl_model = Sequential([
    base_model,
    GlobalAveragePooling2D(),  # Reduce each feature map to a single value
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(train_generator.num_classes, activation='softmax')
])

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

Explanation:

- The initial weights of VGG16 derived from ImageNet form the basis for this model after removing its classification sections.
- The GlobalAveragePooling2D layer acts to reduce spatial information from the previous convolutional operations.
- Additional dense layers become part of the model architecture since they help adjust the model for butterfly classification.
- The base layer freezing mechanism stops weight updates during initial training but it helps fasten convergence while preventing overfitting on the small dataset.
- You can unfreeze specific VGG16 base layers after the initial training for potentially better performance during fine-tuning.

5. Comparison: Fully Connected Network vs. CNN

CNNs can be better understood through their comparison to image classification systems based exclusively on fully connected layers.

Simple Fully Connected Model

In order for a fully connected network to process data it needs the input image converted into a single one-dimensional vector arrangement. Here's an example:

```
# Fully Connected Model (for comparison)
fc_model = Sequential([
    Flatten(input_shape=(150, 150, 3)),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(256, activation='relu'),
    Dense(train_generator.num_classes, activation='softmax')
])

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

Drawbacks of the Fully Connected Model:

- Reorganizing an image through flattening leads to the **elimination of spatial relationships** between elements. The model fails to recognize vital local patterns which include edges and textures because of dimensional compression.
- Consequences of **Increased Parameters** Lead to Model Vulnerability Along with Altered Computational Efficiency.
- Research studies together with practical reality show that fully connected models achieve **worse results** in image classification than CNNs.

Why CNNs Excel

Rephrase the following sentence. The two-dimensional structure of images along with local features recognition capabilities of convolutional layers makes CNNs successful. The network performs higher

efficiency at learning complex patterns such as shapes together with textures through its two-dimensional structure.

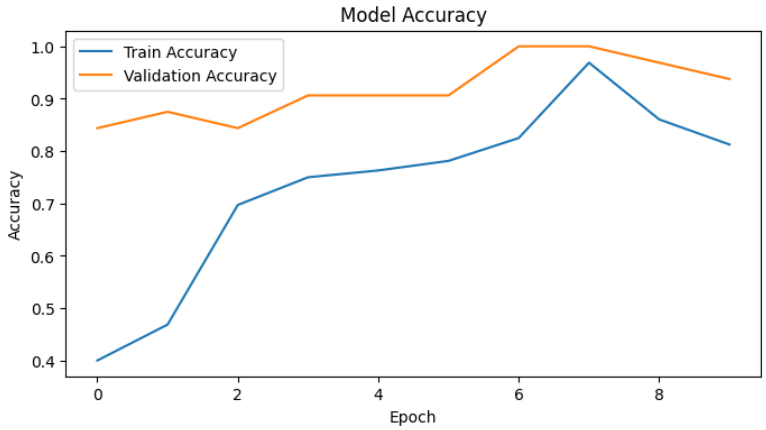
6. Conclusion and Further Resources

In this tutorial, we explored the fundamental concepts behind Convolutional Neural Networks (CNNs) and their advantages over traditional machine learning techniques. The tutorial explained how convolution combines with pooling and fully connected layers operates within CNNs and why these components enable successful image feature extraction. The models were developed using TensorFlow/Keras to classify butterfly species images through a training process.:

- A CNN exists as a self-made model which identifies spatial features throughout different levels of abstraction.
- Our research incorporated VGG16 model pre-training followed by fine-tuning it on our available dataset through a transfer learning method.

We also compared the CNN with a simple fully connected model to highlight the efficiency and accuracy improvements gained through convolutional layers. Every section in this guide comes with detailed comments to maintain understanding and we highlighted a simple design incorporating colorful plots together with reproducible code.

CNN Model Accuracy



Further Reading and Citations

- **Deep Learning** by Ian Goodfellow, Yoshua Bengio, and Aaron Courville provides an in-depth look into neural networks and CNNs.
- **Krizhevsky et al. (2012)** introduced the groundbreaking AlexNet model that popularized CNNs for image classification.
- For additional information about transfer learning and fine-tuning methods refer to Keras documentation pages alongside relevant blog content.
- The tutorial and dataset creation received inspiration from the instructions in the assignment document.

All files including complete code and detailed Jupyter Notebook and supplementary material can be found at https://github.com/jahanzaib3611/CNNs-for-Image-Classification-Tutorial.git on our GitHub repository. Additional resources such as a README file and license file can be found in the repository to foster easier adoption of this work by others.

References

1. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
2. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *ImageNet Classification with Deep Convolutional Neural Networks*. In Advances in Neural Information Processing Systems.
3. Keras Documentation. Retrieved from https://keras.io