Building and Training a Basic CNN Model for Image Classification

A Convolutional Neural Network (CNN) can be built and trained to **classify images into categories** by learning hierarchical features automatically. The process involves **data preparation, model architecture design, training, and evaluation**.

Steps to Build and Train a Basic CNN

1. Data Preparation

- Load the dataset (e.g., MNIST, CIFAR-10) and preprocess it.
- Normalize pixel values to range [0, 1] for faster convergence.
- Convert labels to one-hot encoded format for classification.

2. Model Architecture

- **Convolutional Layers:** Extract features from images using filters.
- **Activation (ReLU):** Introduces non-linearity.
- **Pooling Layers (MaxPooling):** Reduce spatial dimensions and computation.
- **Flatten Layer:** Convert 2D feature maps into 1D vector.
- **Fully Connected (Dense) Layers:** Perform classification.
- **Output Layer (Softmax):** Provides probabilities for each class.

3. Model Compilation

- Specify **loss function**, **optimizer**, and **evaluation metrics**.
- For multi-class classification: loss='categorical_crossentropy' and metrics=['accuracy'].

4. Model Training

- Feed training data into the model.
- Use **validation data** to monitor performance.
- Adjust **epochs** and **batch size** based on dataset size.

5. Model Evaluation

- Evaluate accuracy on test data.
- Optionally, plot learning curves to analyze training and validation loss/accuracy.

Python Example (Basic CNN for MNIST Digit Classification):

```
from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

from tensorflow.keras.datasets import mnist

from tensorflow.keras.utils import to_categorical


# Load dataset

(X_train, y_train), (X_test, y_test) = mnist.load_data()

X_train = X_train.reshape(-1,28,28,1).astype('float32') / 255

X_test = X_test.reshape(-1,28,28,1).astype('float32') / 255

y_train = to_categorical(y_train, 10)

y_test = to_categorical(y_test, 10)


# Build CNN model

model = Sequential()

model.add(Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)))

model.add(MaxPooling2D((2,2)))

model.add(Conv2D(64, (3,3), activation='relu'))

model.add(MaxPooling2D((2,2)))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dense(10, activation='softmax'))


# Compile model

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


# Train model
```

```
model.fit(X_train, y_train, epochs=5, batch_size=64, validation_data=(X_test, y_test))


# Evaluate model
loss, accuracy = model.evaluate(X_test, y_test)
print("Test Accuracy:", accuracy)
```

Tips for Improving CNN Performance

- Add more convolutional layers to capture complex patterns.
- Use **dropout layers** to reduce overfitting.
- Experiment with **data augmentation** to increase training data diversity.
- Tune **hyperparameters** like learning rate, batch size, and number of filters.

Building a basic CNN model allows you to **understand the workflow of image classification**, from data preprocessing to model evaluation, forming the foundation for more advanced computer vision projects.