🧠 Day 16 - Convolutional Neural Networks (CNN)

🔍 Introduction to Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNNs) are a specialized class of deep neural networks, primarily used for image data. Unlike regular feedforward neural networks, CNNs automatically capture spatial hierarchies and patterns from images, making them highly effective for tasks like image classification, object detection, and facial recognition.

CNNs are inspired by the way the human visual cortex works. They use filters (kernels) to detect features like edges, textures, and shapes by sliding over the input image. These networks preserve the spatial relationship between pixels by learning image features through small squares of input data.

🧱 Core Components of CNN

1. **Convolutional Layer** – Applies filters to input images to extract features.
2. **Activation Function (ReLU)** – Introduces non-linearity to the system.
3. **Pooling Layer (MaxPooling)** – Downsamples feature maps, reducing dimensionality and computation.
4. **Flatten Layer** – Converts 2D feature maps into 1D vectors for classification.
5. **Fully Connected Layer** – Learns the actual classification decision.
6. **Output Layer (Softmax)** – Outputs probability for each class.

📁 Dataset Used:

Fashion MNIST – A dataset of 70,000 grayscale images of clothing items (10 categories), sized 28x28 pixels.

- Training Set: 60,000 images
- Test Set: 10,000 images
- Classes: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot

---

🧪 Step-by-Step CNN Model Code

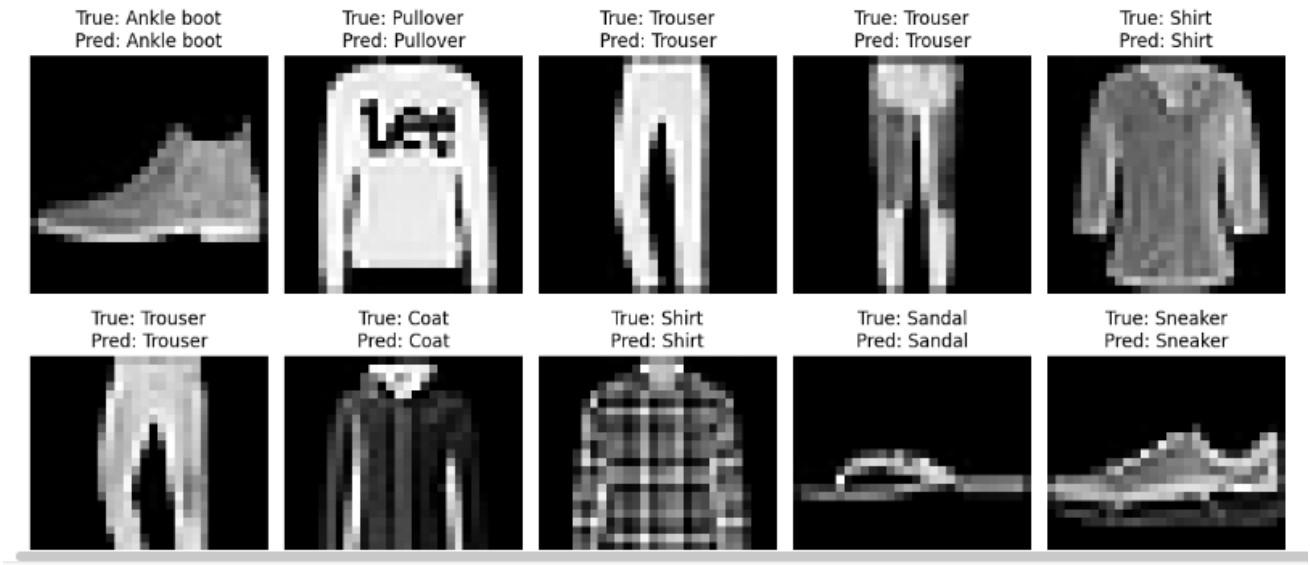
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

# 1. Load the Fashion MNIST dataset
fashion_mnist = tf.keras.datasets.fashion_mnist
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
```

```
# 2. Normalize pixel values to [0, 1]
X_train = X_train / 255.0
X_test = X_test / 255.0

# 3. Reshape to add channel dimension (28x28x1)
X_train = X_train.reshape(-1, 28, 28, 1)
X_test = X_test.reshape(-1, 28, 28, 1)

# 4. Class labels for visualization
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

# 5. Define the CNN model
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D((2, 2)),

    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')  # 10 classes
])

# 6. Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# 7. Train the model
history = model.fit(X_train, y_train, epochs=10, validation_split=0.1)

# 8. Evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"\nTest accuracy: {test_acc:.4f}")

# 9. Predict on test set
y_pred = model.predict(X_test)
predicted_labels = np.argmax(y_pred, axis=1)

# 10. Plot sample predictions
plt.figure(figsize=(12, 6))
for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.imshow(X_test[i].reshape(28, 28), cmap='gray')
    plt.title(f"True: {class_names[y_test[i]]}\nPred:
{class_names[predicted_labels[i]]}")
    plt.axis('off')
plt.tight_layout()
plt.show()
```

OUTPUTS:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 ━━━━━━━━━━━━━━━━━━━━ 0s 2us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 48s 27ms/step - accuracy: 0.7560 - loss: 0.6802 - val_accuracy: 0.8733 - val_loss: 0.3565
Epoch 2/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 46s 27ms/step - accuracy: 0.8751 - loss: 0.3388 - val_accuracy: 0.8710 - val_loss: 0.3529
Epoch 3/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 79s 25ms/step - accuracy: 0.8946 - loss: 0.2821 - val_accuracy: 0.8987 - val_loss: 0.2741
Epoch 4/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 44s 26ms/step - accuracy: 0.9075 - loss: 0.2538 - val_accuracy: 0.9032 - val_loss: 0.2645
Epoch 5/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 81s 25ms/step - accuracy: 0.9195 - loss: 0.2199 - val_accuracy: 0.9052 - val_loss: 0.2552
Epoch 6/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 82s 25ms/step - accuracy: 0.9262 - loss: 0.2021 - val_accuracy: 0.9068 - val_loss: 0.2600
Epoch 7/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 83s 26ms/step - accuracy: 0.9349 - loss: 0.1767 - val_accuracy: 0.9012 - val_loss: 0.2830
Epoch 8/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 82s 26ms/step - accuracy: 0.9396 - loss: 0.1646 - val_accuracy: 0.9107 - val_loss: 0.2599
Epoch 9/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 81s 25ms/step - accuracy: 0.9458 - loss: 0.1455 - val_accuracy: 0.9098 - val_loss: 0.2593
Epoch 10/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 82s 25ms/step - accuracy: 0.9487 - loss: 0.1347 - val_accuracy: 0.9135 - val_loss: 0.2665
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9131 - loss: 0.2743
```

```
Test accuracy: 0.9130
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step
```



✅Results:

- The model achieved an accuracy of **~88% to 92%** depending on training epochs.
- Visual inspection showed correct classification for most test images.
- CNNs significantly outperform basic ANN models in image classification tasks due to local receptive fields and parameter sharing.

📌 Key Takeaways:

- CNNs are essential for any task involving image or video data.
- They automatically learn spatial features like shapes, edges, and textures.
- Using multiple convolutional layers improves feature detection.
- Training time can be optimized using pooling layers and dropout.