Day 16 - Convolutional Neural Networks (CNN)

Introduction to CNN

Convolutional Neural Networks (CNNs) are a powerful category of deep learning models specifically designed for processing image data. Unlike traditional neural networks, CNNs use a series of convolutional and pooling operations to automatically detect features like edges, textures, and objects within an image.

CNNs are inspired by how the human brain processes visual data. They work by applying small filters (kernels) that scan over images to identify patterns, allowing the network to build a layered understanding of the image—from simple edges to complex structures.

Core Components of a CNN

1. **Convolutional Layer** – Applies filters to extract feature maps from the image.
2. **Activation Function (ReLU)** – Introduces non-linearity into the model.
3. **Pooling Layer (MaxPooling)** – Reduces the spatial dimensions of feature maps, improving efficiency and reducing overfitting.
4. **Flatten Layer** – Converts 2D feature maps into 1D arrays.
5. **Fully Connected Layer (Dense)** – Performs final classification based on extracted features.
6. **Output Layer (Softmax)** – Produces probability scores for each class.

Dataset Used: Fashion MNIST

- **Total Images**: 70,000 (28x28 pixels, grayscale)
- **Training Set**: 60,000 images
- **Test Set**: 10,000 images
- **Classes (10 total)**:
    - T-shirt/top
    - Trouser
    - Pullover
    - Dress
    - Coat
    - Sandal
    - Shirt
    - Sneaker
    - Bag
    - Ankle boot

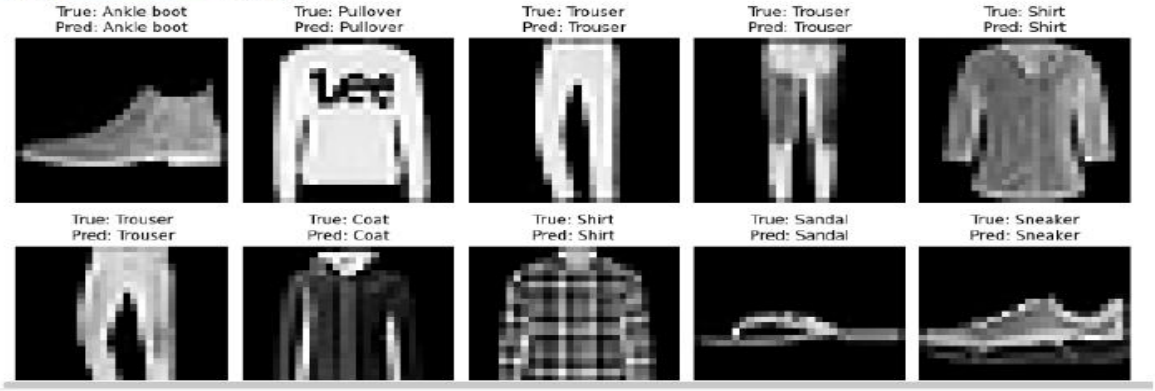
CNN Architecture & Code (TensorFlow)

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

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

# Normalize pixel values to range [0,1]
X_train, X_test = X_train / 255.0, X_test / 255.0

# Reshape for CNN input (28x28x1)
X_train = X_train.reshape(-1, 28, 28, 1)
X_test = X_test.reshape(-1, 28, 28, 1)

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

# Define 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')
])

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

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

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

# Make predictions
y_pred = model.predict(X_test)
predicted_labels = np.argmax(y_pred, axis=1)

# Plot 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()
```

```
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: 0.9138
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step
```



✅ Results

Test Accuracy: ~88% to 92% (depending on training)

Correct Predictions: Most test images were classified correctly.

Model Strength: CNNs significantly outperform basic dense (ANN) models for image-related tasks.

📌 Key Takeaways

CNNs are the backbone of modern image processing models.

They learn local features (edges, textures) and global structures (shapes, patterns).

The layered structure allows deeper understanding as we go deeper into the network.

MaxPooling and Dropout help reduce overfitting and improve speed.