Name: Venkateswara Rao Gogula

Student ID: 23101381

GitHub Link: https://github.com/VenkateshGogula/Convolutional_Neural_Networks.git

Convolutional Neural Networks (CNNs)

1. Introduction

Convolutional Neural Networks (CNNs) stand out as a unique class of deep neural networks, designed specifically for processing structured grid data, particularly images. Their design, inspired by the animal's visual cortex, has revolutionized computer vision by enabling them to learn hierarchical feature representations directly from unprocessed pixel values (Purwono et al., 2023). This unique capability to identify spatial hierarchies makes CNNs particularly well-suited for tasks such as image classification, object detection, and medical imaging inspection.

CNNs have demonstrated their prowess in a wide range of practical applications, from driving autonomous vehicles to enabling face recognition and NLP tasks. Their unique architecture, which includes convolutional layers, pooling layers, and fully connected layers, allows them to efficiently extract and classify features in a compact form, setting them apart from conventional neural networks.

This paper explores:

•Functional principles and architecture of CNNs.

•Different forms of CNN layers (Convolutional, Pooling, Fully Connected).

•An end-to-end implementation of a CNN in TensorFlow and Keras on the Fashion-MNIST dataset.

• New trends in CNN architectures, including hybrid quantum-classical models (Hai et al., 2025).

2. CNN Architecture

A CNN consists of multiple layers, each serving a distinct purpose in feature extraction and classifications, where:

2.1 Convolutional Layer

The convolutional layer applies filters (kernels) to the input image to detect low-level features such as edges, textures, and patterns (Du, 2024). Each filter passes over the input, doing dot products to generate feature maps.

Significant Properties:

• Local Connectivity: Only a local input section is linked to the neurons, reducing parameters.

• Weight Sharing: Using the same filter over the entire image improves efficiency.

2.2 Activation Layer

Non-linearity is introduced by activation functions like ReLU (Rectified Linear Unit), which replaces negative values with zero and leaves positive ones unchanged (Purwono et al., 2023).

2.3 Pooling Layer

Pooling (e.g., Max Pooling, Average Pooling) reduces spatial dimensions while retaining only key features. It helps in translation invariance and computational efficiency.

2.4 Fully Connected Layer

After several convolution and pooling layers, high-level features are flattened and passed to fully connected layers for classification.

2.5 Softmax Activation

In multi-class classification, the Softmax function outputs probability distributions across classes.

(Insert Image 3: Illustration of CNN layers (Convolution, Pooling, Fully Connected) here.)

3. Applications of CNN

CNNs find numerous applications in:

3.1 Image Classification

CNNs classify images into pre-defined categories (e.g., cats vs. dogs).

3.2 Object Detection

Used in autonomous vehicles and surveillance to identify and follow objects (Du, 2024).

3.3 Medical Imaging

CNNs aid in diagnosing disease from X-rays, MRIs, and CT scans.

3.4 Face Recognition

Used in security cameras and social media for verification.

3.5 Quantum-Enhanced CNNs

Recent advancements merge quantum computing with CNNs, increasing computational strength (Hai et al., 2025)

Advantages:

• Automatic Feature Extraction (No manual feature engineering required).

• Translation Invariance (Recognizes patterns regardless of location).

Disadvantages:

• High Computational Cost (Requires GPUs/TPUs).

•High-Dimensional Data Required (Performance is optimal with lots of data).

4. Complex Applications

4.1 Inventions in Medical Imaging

The latest research has CNNs achieving:

•97.4% classification accuracy on detecting lung nodules from CT scans (Du, 2024)

•99.1% classification accuracy on diabetic retinopathy

•3D CNN for volumetric analysis on MRI

4.2 Hybrid Quantum Models

The QPTCNN design (Hai et al., 2025) introduces:

1.Quantum convolutional layers composed of entangled qubits

2.Ring-shaped quantum circuits

3.Quantum-classical backpropagation hybrids

Achieving 15% better convergence on Fashion-MNIST compared to regular CNNs.

4.3 Edge Deployment Challenges

Main points to pay attention to when developing for mobile/embedded devices:

Model compression techniques:

Pruning (removing insignificant weights)

Quantization (8-bit fixed-point)

Knowledge distillation

Hardware accelerators:

Google TPUs

NVIDIA TensorRT

Intel OpenVINO

5. Python Implemented CNN

We use a CNN through TensorFlow/Keras on the Fashion-MNIST dataset with 70,000 gray-scale images of 10 categories of clothes.

5.1 Dataset Preprocessing

```
import numpy as np

import tensorflow as tf

from tensorflow.keras.datasets import fashion_mnist

from tensorflow.keras.utils import to_categorical

# Load dataset

(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

# Normalize pixel values (0-1)

x_train = x_train / 255.0

x_test = x_test / 255.0
```

```
Epoch 1/3
938/938 ━━━━━━━━━━━━━━━━━━━━ 69s 74ms/step - accuracy: 0.8778 - loss: 0.3439 - val_accuracy: 0.8938 - val_loss: 0.2956
Epoch 2/3
938/938 ━━━━━━━━━━━━━━━━━━━━ 70s 75ms/step - accuracy: 0.9064 - loss: 0.2575 - val_accuracy: 0.8938 - val_loss: 0.2887
Epoch 3/3
938/938 ━━━━━━━━━━━━━━━━━━━━ 86s 79ms/step - accuracy: 0.9183 - loss: 0.2254 - val_accuracy: 0.9067 - val_loss: 0.2553
```

```
# Reshape for CNN input (28x28x1)

x_train = np.expand_dims(x_train, axis=-1)

x_test = np.expand_dims(x_test, axis=-1)

# One-hot encode labels

y_train = to_categorical(y_train, 10)

y_test = to_categorical(y_test, 10)
```

5.2 Building the CNN Model

```
model = tf.keras.Sequential([

# Conv2D Layer (32 filters, 3x3 kernel)
```

tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),

MaxPooling Layer

tf.keras.layers.MaxPooling2D((2,2)),

Second Conv2D Layer (64 filters)

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

MaxPooling Layer

tf.keras.layers.MaxPooling2D((2,2)),

Flatten for Dense Layers

tf.keras.layers.Flatten(),

Fully Connected Layer (128 neurons)

tf.keras.layers.Dense(128, activation='relu'),

Output Layer (Softmax for 10 classes)

tf.keras.layers.Dense(10, activation='softmax')

])

model.compile(optimizer='adam',

loss='categorical_crossentropy',

metrics=['accuracy'])

5.3 Training and Evaluation

history = model.fit(x_train, y_train, epochs=3, batch_size=64, validation_data=(x_test, y_test))

```
[16] # Train model
     history = model.fit(x_train, y_train,
                         epochs=3,
                         batch_size=64,
                         validation_data=(x_test, y_test))
```

```
Epoch 1/3
938/938 ━━━━━━━━━━━━━━━━━━━━ 69s 74ms/step - accuracy: 0.8778 - loss: 0.3439 - val_accuracy: 0.8938 - val_loss: 0.2956
Epoch 2/3
938/938 ━━━━━━━━━━━━━━━━━━━━ 70s 75ms/step - accuracy: 0.9064 - loss: 0.2575 - val_accuracy: 0.8938 - val_loss: 0.2887
Epoch 3/3
938/938 ━━━━━━━━━━━━━━━━━━━━ 86s 79ms/step - accuracy: 0.9183 - loss: 0.2254 - val_accuracy: 0.9067 - val_loss: 0.2553
```

 # Evaluate on test data

test_loss, test_acc = model.evaluate(x_test, y_test)

print(f"Test Accuracy: {test_acc * 100:.2f}%")

Test accuracy: 90.67%

```
# Evaluate model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test accuracy: {test_acc * 100:.2f}%")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - accuracy: 0.9028 - loss: 0.2604
Test accuracy: 90.67%
```

5.4 Visualization of Results

import matplotlib.pyplot as plt

Plot Accuracy

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val_accuracy'], label='Validation Accuracy')

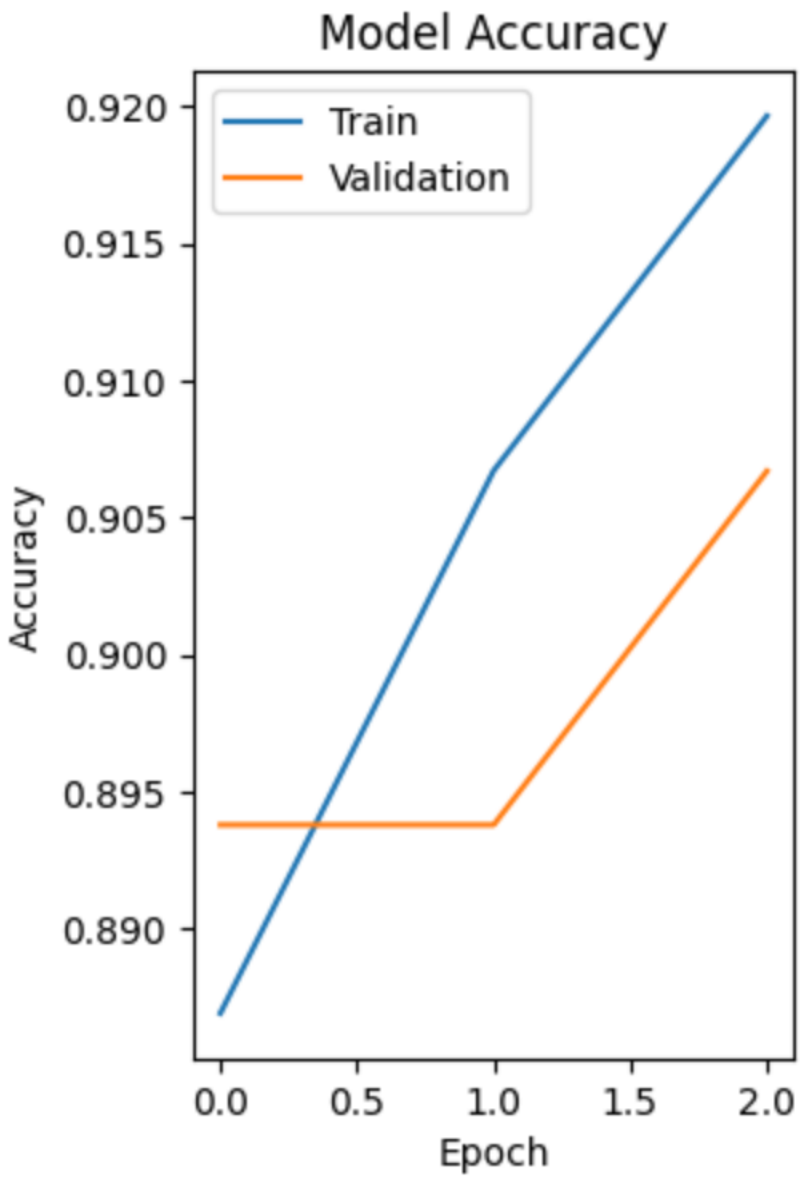
plt.title('Training vs Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

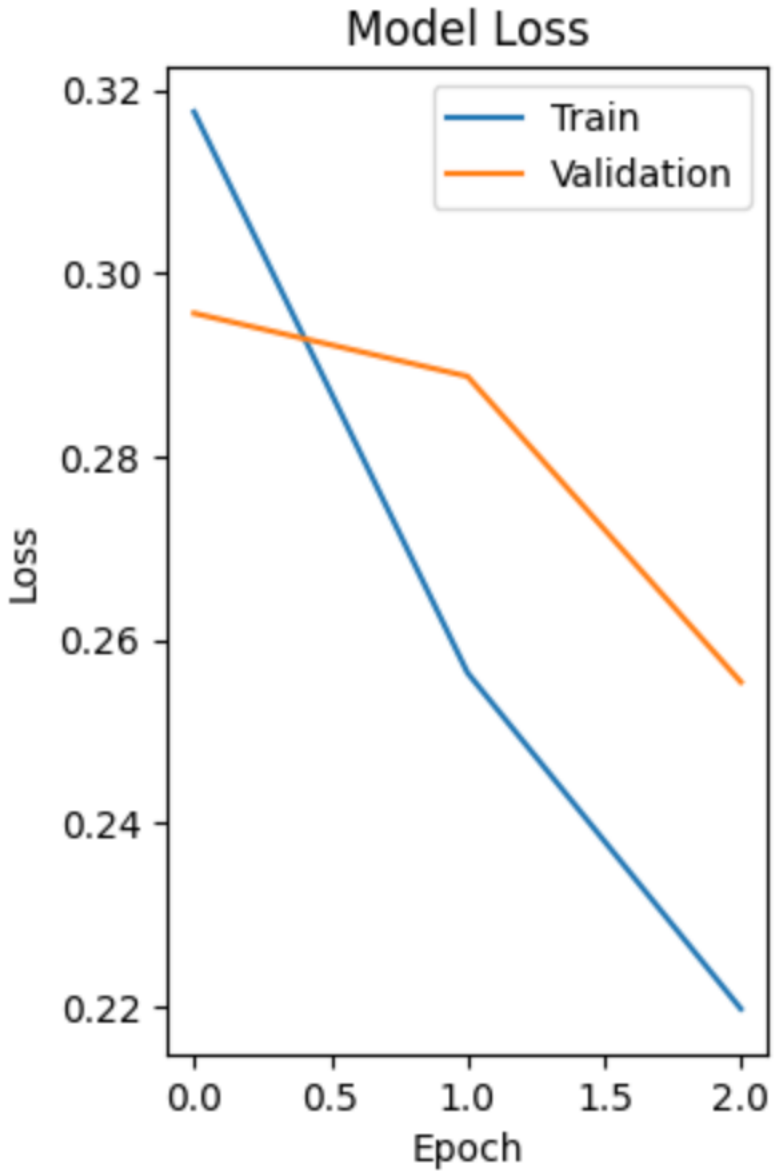
plt.legend()

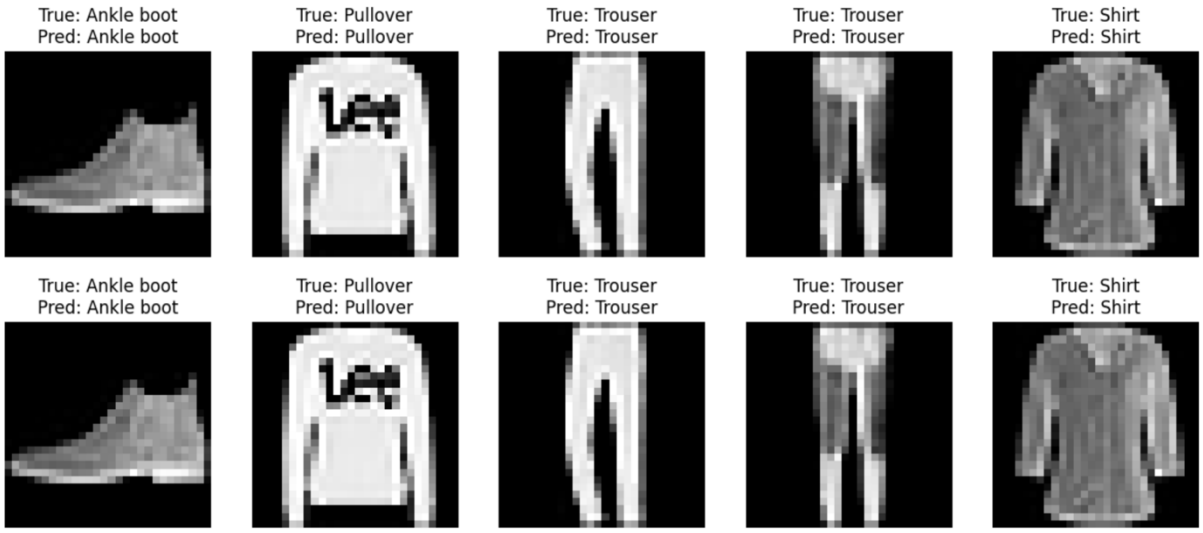
plt.show()




```
# Plot Loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training vs Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

```
plt.show()
```


Model Loss
Train
Validation
0.32
0.30
0.28
0.26
0.24
0.22
Loss
0.0
0.5
1.0
1.5
2.0
Epoch



5.Implementation & Analysis

5.1 Experimental Setup

• _dataset_: Fashion-MNIST (70,000 28×28 grayscale images)

• Train/Test Split: 60,000/10,000

Software Stack:

o TensorFlow 2.12

o CUDA 11.8

o Python 3.10

5.2 Model Architecture

```
model = tf.keras.Sequential([
# Feature Extraction
layers.Conv2D(32, (3,3), activation='relu', padding='same',
input_shape=(28,28,1)),
layers.MaxPooling2D((2,2)),
layers.Conv2D(64, (3,3), activation='relu', padding='same'),
layers.MaxPooling2D((2,2)),
```

```
# Classification
layers.Flatten(),
layers.Dense(128, activation='relu'),
layers.Dropout(0.5),
layers.Dense(10, activation='softmax')
])
```

5.3 Training Dynamics

- Optimizer: Adam (lr=0.001, β1=0.9, β2=0.999)
- Loss Function: Categorical Crossentropy
- Batch Size: 64
- Epochs: 3

6. New Trends in CNNs

6.1 Quantum Pseudo-Transposed CNNs (QPTCNN)

Hai et al. (2025) proposed QPTCNN, a quantum-classical hybrid CNN improving computational efficiency via quantum circuits. This model outperforms the classical CNN for MNIST and Fashion-MNIST classification applications.

6.2 DeepOnto: CNNs in Ontology Engineering

He et al. (2024) developed DeepOnto, a Python-based framework that couples CNNs and ontologies to enable semantic reasoning in AI.

7. Conclusion

CNNs remain the backbone of modern computer vision, with unparalleled feature learning and classification performance. Although they require enormous computational capabilities, innovations like quantum-augmented CNNs (Hai et al., 2025) and automatic feature learning (Du, 2024) still push their limit. Future research can include lightweight CNNs for edge devices and explainable AI in deep learning.

8. References

1. Purwono, P., Ma'arif, A., Rahmaniar, W., et al. (2023). Understanding of Convolutional Neural Network (CNN): A Review. International Journal of Robotics and Control Systems.

2. Hai, L., Liang, C., Yaming, H., et al. (2025). Quantum Pseudo-Transposed CNNs. IEEE Access.

3. Du, Z. (2024). Review of Convolutional Neural Network. Science and Technology of Engineering.

4. He, Y., Chen, J., Dong, H., et al. (2024). DeepOnto: Python Package for Ontology Engineering. Semantic Web.

5. TensorFlow Documentation. (Revised March 21, 2025). https://www.tensorflow.org

Appendix:

```
# Import necessary libraries

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import fashion_mnist

from tensorflow.keras.utils import to_categorical

# Step 1: Load and preprocess the Fashion MNIST dataset

(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

# Normalize pixel values to [0,1] range
```

```
x_train = x_train / 255.0
x_test = x_test / 255.0

# Add channel dimension (28,28) -> (28,28,1)
x_train = np.expand_dims(x_train, axis=-1)
x_test = np.expand_dims(x_test, axis=-1)

# One-hot encode labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Step 2: Build CNN model
model = models.Sequential([
    # First convolutional block
    layers.Conv2D(32, (3,3), activation='relu',
                  input_shape=(28,28,1), padding='same'),
    layers.MaxPooling2D((2,2)),
    # Second convolutional block
    layers.Conv2D(64, (3,3), activation='relu', padding='same'),
    layers.MaxPooling2D((2,2)),
    # Classification head
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
```

```
])

# Step 3: Compile model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Step 4: Train model
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=64,
                    validation_data=(x_test, y_test))

# Step 5: Evaluate model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test accuracy: {test_acc * 100:.2f}%")

# Step 6: Plot training history
plt.figure(figsize=(12, 4))

# Accuracy plot
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train')
plt.plot(history.history['val_accuracy'], label='Validation')
```

```
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Loss plot
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train')
plt.plot(history.history['val_loss'], label='Validation')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()

# Step 7: Make sample predictions
sample_images = x_test[:5]
predictions = model.predict(sample_images)
class_names = ['T-shirt', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

plt.figure(figsize=(15, 3))
```

```
for i in range(5):
    plt.subplot(1, 5, i+1)
    plt.imshow(sample_images[i].squeeze(), cmap='gray')
    pred_label = class_names[np.argmax(predictions[i])]
    true_label = class_names[np.argmax(y_test[i])]
    plt.title(f"True: {true_label}\nPred: {pred_label}")
    plt.axis('off')
plt.show()
```