**You are designing a CNN to enhance edge detection in images. What changes would you make to a standard CNN architecture to optimize for edge features?**

**Code:**

import tensorflow as tf

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

from tensorflow.keras.models import Model

from tensorflow.keras.preprocessing import image

import numpy as np

# Custom Sobel edge detection layer

class SobelLayer(Layer):

    def \_\_init\_\_(self):

        super(SobelLayer, self).\_\_init\_\_()

    def call(self, inputs):

        # Applying Sobel edge detection using TensorFlow's image processing

        sobel\_edges = tf.image.sobel\_edges(inputs)  # Sobel edge detection

        sobel\_magnitude = tf.sqrt(tf.reduce\_sum(tf.square(sobel\_edges), axis=-1))  # Magnitude of edges

        return sobel\_magnitude

# Custom Concatenate Layer to merge input and Sobel output

class ConcatenateLayer(Layer):

    def \_\_init\_\_(self):

        super(ConcatenateLayer, self).\_\_init\_\_()

    def call(self, inputs):

        image\_input, sobel\_output = inputs

        return tf.concat([image\_input, sobel\_output], axis=-1)

# Define the CNN model for edge detection

def edge\_detection\_cnn(input\_shape):

    inputs = Input(shape=input\_shape)

    # Early Sobel edge detection layer to capture edge features directly

    sobel\_layer = SobelLayer()(inputs)

    # Concatenate original input with the Sobel edge detection output

    x = ConcatenateLayer()([inputs, sobel\_layer])

    # Apply convolutional layers with small kernels to capture detailed features

    x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)

    x = MaxPooling2D((2, 2))(x)

    x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)

    x = MaxPooling2D((2, 2))(x)

    x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)

    x = MaxPooling2D((2, 2))(x)

    # Flatten the output for fully connected layers

    x = Flatten()(x)

    # Fully connected layers for classification (assuming classification task)

    x = Dense(256, activation='relu')(x)

    x = Dense(64, activation='relu')(x)

    # Output layer for 10-class classification (adjust as needed)

    outputs = Dense(10, activation='softmax')(x)

    # Construct the model

    model = Model(inputs=inputs, outputs=outputs)

    return model

# Define the class labels (customize as needed)

class\_labels = {

    0: 'Class A',

    1: 'Class B',

    2: 'Class C',

    3: 'Class D',

    4: 'Class E',

    5: 'Class F',

    6: 'Class G',

    7: 'Class H',

    8: 'Class I',

    9: 'Class J'

}

# Function to load and preprocess the image

def predict\_on\_image(model, image\_path):

    # Load the image with grayscale mode and target size

    img = image.load\_img(image\_path, target\_size=(28, 28), color\_mode='grayscale')

    img\_array = image.img\_to\_array(img) / 255.0  # Normalize to [0, 1]

    img\_array = np.expand\_dims(img\_array, axis=0)  # Add batch dimension

    # Predict using the model

    predictions = model.predict(img\_array)

    # Get the index of the highest probability

    predicted\_index = np.argmax(predictions[0])

    # Map the index to the class label

    predicted\_label = class\_labels[predicted\_index]

    print("Predicted class:", predicted\_label)

    print("Class probabilities:", predictions[0])

# Example input shape for 28x28 grayscale images

input\_shape = (28, 28, 1)

# Instantiate and compile the model

model = edge\_detection\_cnn(input\_shape)

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Show model architecture summary

model.summary()

# Predict on an image file (replace 'photo.png' with your image path)

predict\_on\_image(model, 'photo.png')

**Output:**

Model: "functional"

┏━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━┳━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓

┃ Layer (type) ┃ Output Shape ┃ Param # ┃ Connected to ┃

┡━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━╇━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩

│ input\_layer │ (None, 28, 28, │ 0 │ - │

│ (InputLayer) │ 1) │ │ │

├──────────────────┼────────────────┼──────────┼─────────────────┤

│ sobel\_layer │ (None, 28, 28, │ 0 │ input\_layer[0]… │

│ (SobelLayer) │ 1) │ │ │

├──────────────────┼────────────────┼──────────┼─────────────────┤

│ concatenate\_lay… │ (None, 28, 28, │ 0 │ input\_layer[0]… │

│ (ConcatenateLay… │ 2) │ │ sobel\_layer[0]… │

├──────────────────┼────────────────┼──────────┼─────────────────┤

│ conv2d (Conv2D) │ (None, 28, 28, │ 608 │ concatenate\_la… │

│ │ 32) │ │ │

├──────────────────┼────────────────┼──────────┼─────────────────┤

│ max\_pooling2d │ (None, 14, 14, │ 0 │ conv2d[0][0] │

│ (MaxPooling2D) │ 32) │ │ │

├──────────────────┼────────────────┼──────────┼─────────────────┤

│ conv2d\_1 │ (None, 14, 14, │ 18,496 │ max\_pooling2d[… │

│ (Conv2D) │ 64) │ │ │

├──────────────────┼────────────────┼──────────┼─────────────────┤

│ max\_pooling2d\_1 │ (None, 7, 7, │ 0 │ conv2d\_1[0][0] │

│ (MaxPooling2D) │ 64) │ │ │

├──────────────────┼────────────────┼──────────┼─────────────────┤

│ conv2d\_2 │ (None, 7, 7, │ 73,856 │ max\_pooling2d\_… │

│ (Conv2D) │ 128) │ │ │

├──────────────────┼────────────────┼──────────┼─────────────────┤

│ max\_pooling2d\_2 │ (None, 3, 3, │ 0 │ conv2d\_2[0][0] │

│ (MaxPooling2D) │ 128) │ │ │

├──────────────────┼────────────────┼──────────┼─────────────────┤

│ flatten │ (None, 1152) │ 0 │ max\_pooling2d\_… │

│ (Flatten) │ │ │ │

├──────────────────┼────────────────┼──────────┼─────────────────┤

│ dense (Dense) │ (None, 256) │ 295,168 │ flatten[0][0] │

├──────────────────┼────────────────┼──────────┼─────────────────┤

│ dense\_1 (Dense) │ (None, 64) │ 16,448 │ dense[0][0] │

├──────────────────┼────────────────┼──────────┼─────────────────┤

│ dense\_2 (Dense) │ (None, 10) │ 650 │ dense\_1[0][0] │

└──────────────────┴────────────────┴──────────┴─────────────────┘

Total params: 405,226 (1.55 MB)

Trainable params: 405,226 (1.55 MB)

Non-trainable params: 0 (0.00 B)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 214ms/step

Predicted class: Class I

Class probabilities: [0.10936522 0.08179193 0.09012273 0.12951909 0.08509644 0.08812457

0.10036314 0.07812262 0.13758092 0.09991337]