LeNet-5 and SqueezeNet

Surafel Sentayehu

Table of Contents

LeNet-5: The Foundation of Modern CNNs

1. Created by Yann LeCun in 1998
2. Designed to recognize handwritten digits
3. One of the first Convolutional Neural Networks (CNNs)
4. Used by banks and postal services

Key Concepts

1. Convolutional Layers: Detect patterns (like edges)

2. Pooling Layers: Shrink image size, keep important info

3. Dense (Fully Connected) Layers: Make final decisions

4. Activation Functions: Help the network learn nonlinear rules

LeNet-5 Architecture

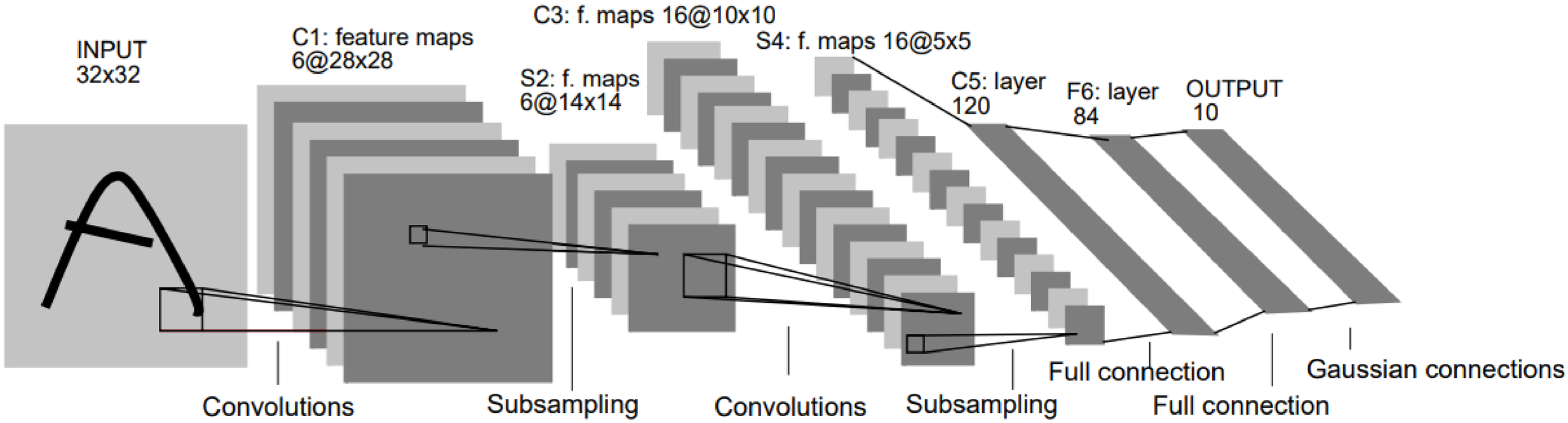


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

LeNet-5 Model Implementation

Layer	Type	Feature Maps	Output Size	Kernel Size	Stride	Activation	Purpose
Input	Image	1	32×32	–	–	–	Input grayscale image
1	Convolution	6	28×28	5×5	1	ReLU	Extract local features (edges/textures)
2	Max Pooling	6	14×14	2×2	2	–	Downsample and reduce spatial size
3	Convolution	16	10×10	5×5	1	ReLU	Learn more complex patterns
4	Max Pooling	16	5×5	2×2	2	–	Further reduce dimensionality
5	Convolution	120	1×1	5×5	1	ReLU	Compress features to abstract form
6	Fully Connected	–	84	–	–	ReLU	Learn classification decision logic
Output	Fully Connected	–	10	–	–	Softmax	Output class probabilities (digits 0–9)

Dataset & Preprocessing

- Dataset: MNIST (images of handwritten digits)
- Training samples: 60,000
- Validation samples: 10,000
- Preprocessing:
  - Images reshaped to 28×28×1
  - Pixel values normalized (0 to 1)
  - Labels one-hot encoded (e.g., 3 → [0 0 0 1 ...])

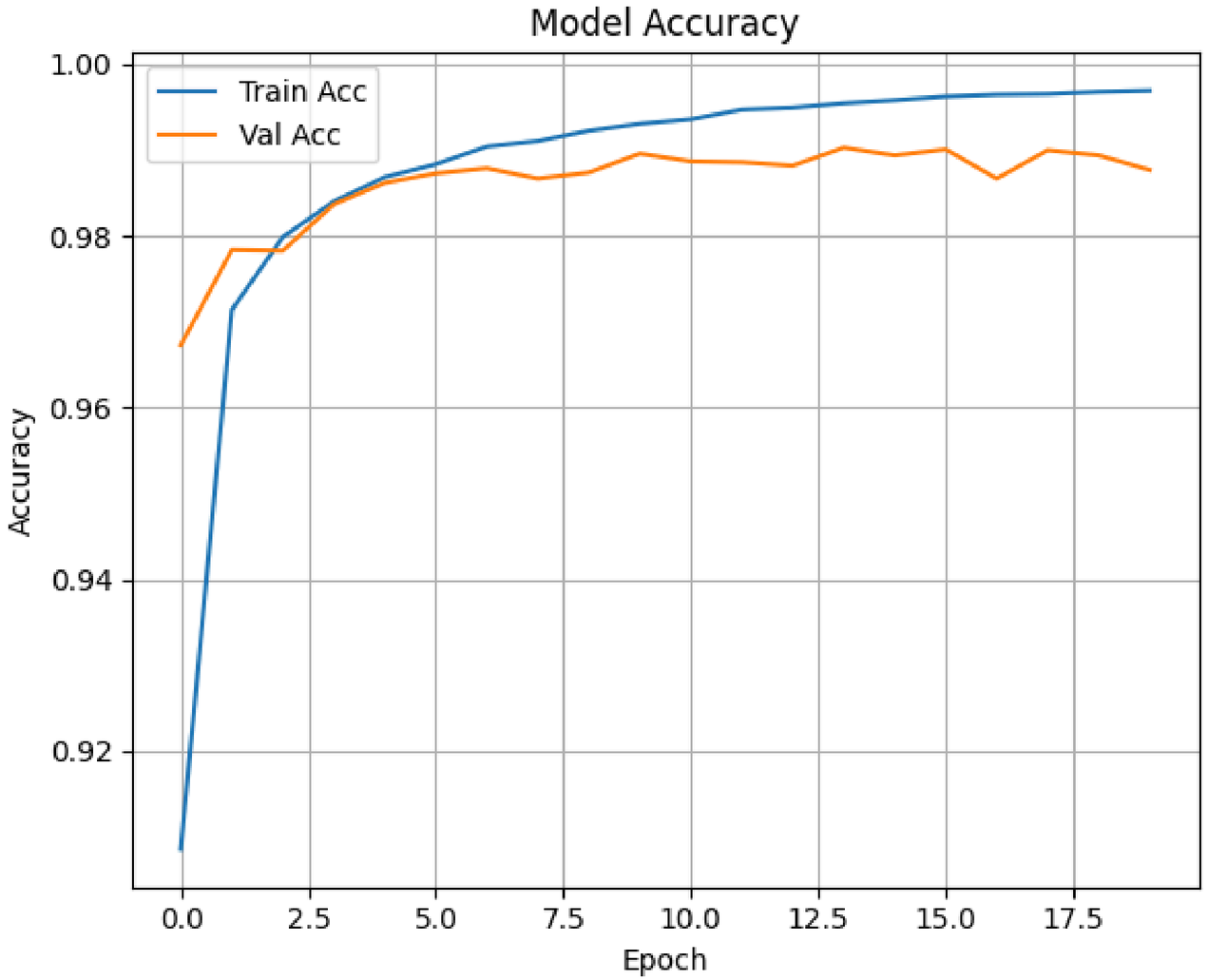
Model Building

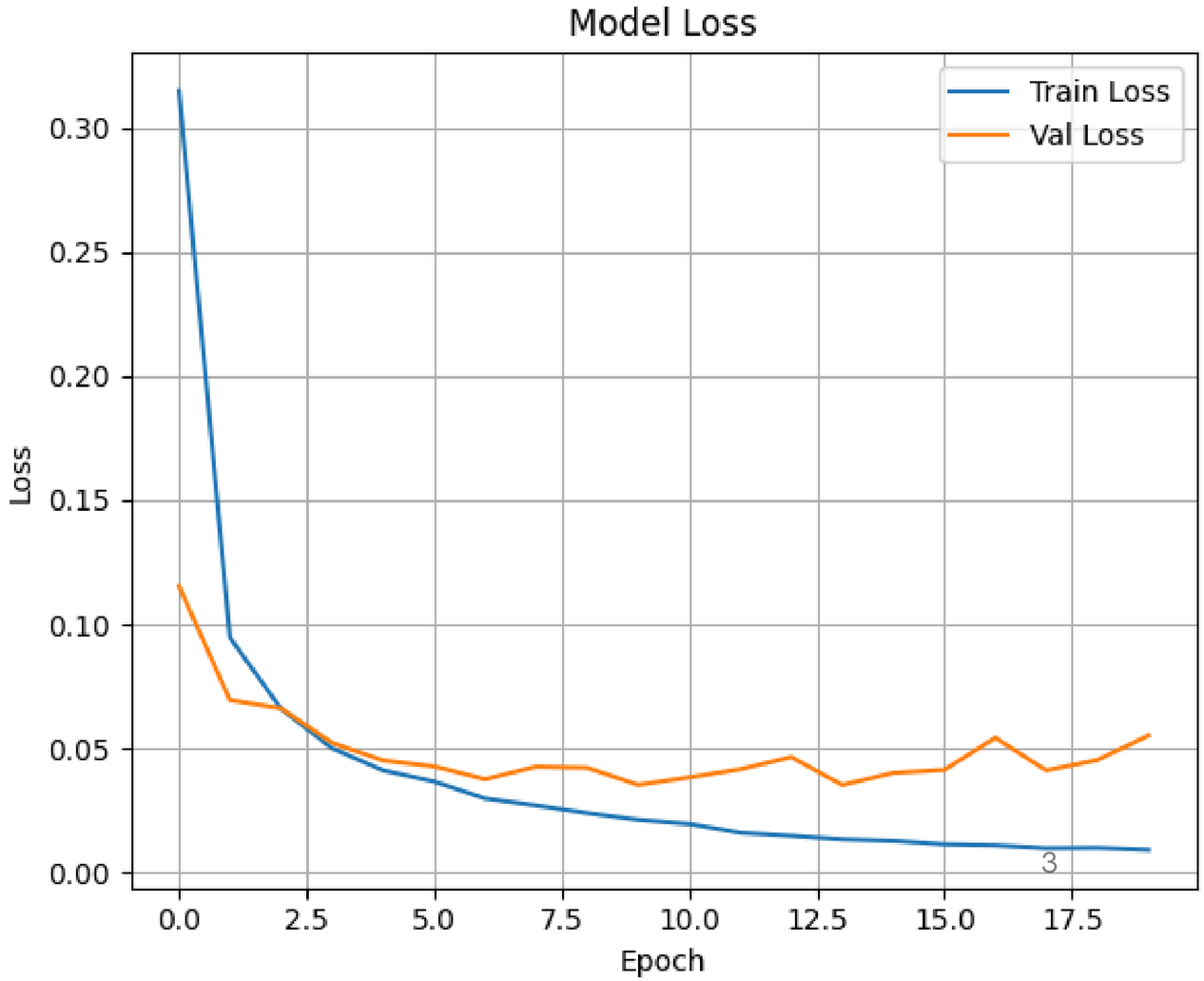
```
model = Sequential()
model.add(Conv2D(6, (5, 5), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(16, (5, 5), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(120, activation='relu'))
model.add(Dense(84, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

Results

Final Validation Accuracy: ≈ 98.97%

Final Validation Loss: 0.049





LeNet-5: Strengths vs. Limitations

Why It Works Well	Limitations
Learns **basic to complex patterns** step-by-step	Only works on **fixed image size** (28×28)
Very few parameters → **fast to train**	Can’t handle **color images or complex scenes**
Performs well on **small image data** like MNIST	Uses **simple activation and pooling** (outdated)

Impact of LeNet-5

- Inspired powerful models like AlexNet, VGG, ResNet
- Introduced convolution + pooling + dense pipeline
- Still used in education and small embedded systems

Introduction to SqueezeNet

1. Developed by Iandola et al. (2016)

2. Match AlexNet accuracy with 50x fewer parameters

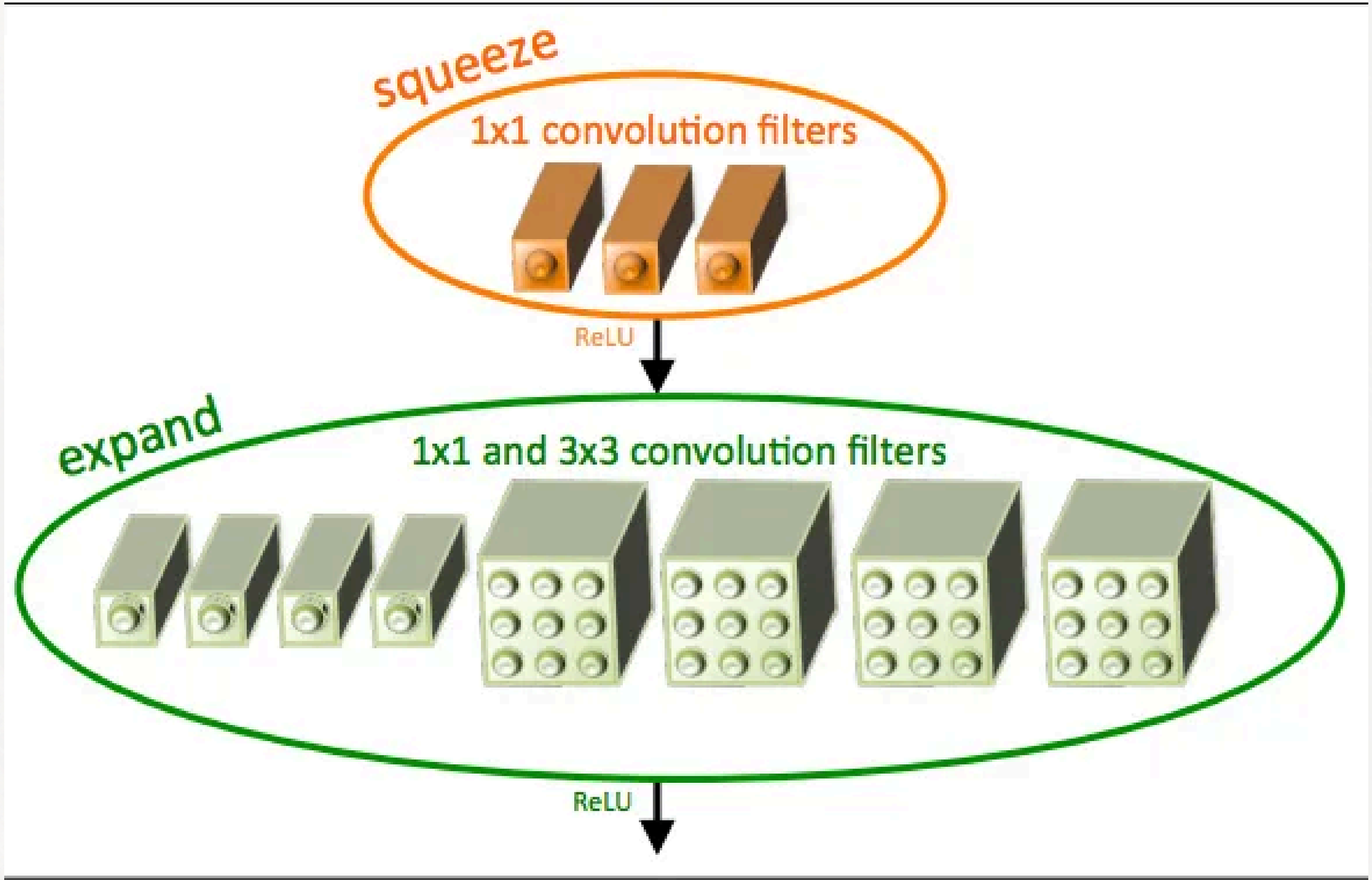
3. Model size: ~4.8 MB, compressible to <0.5 MB

4. Ideal for mobile and edge devices (phones, drones, FPGAs)

What Makes SqueezeNet Special?

- Uses mostly 1×1 filters (instead of 3×3)
- Fire modules combine “squeeze” and “expand” layers
- No fully connected layers, uses global average pooling
- Final model has only 1.25M parameters

The Fire Module



Overall Architecture of SqueezeNet

TensorFlow Implementation Overview

- Framework: TensorFlow / Keras
- Dataset: CIFAR-10 (32×32 color images)
- Output Classes: 10 (airplane, dog, cat, etc.)
- Training Epochs: 80

Model Building

```python
# Define the Fire module
def fire_module(x, fire_id, squeeze=16, expand=64):
    squeeze_name = f'fire{fire_id}_squeeze1x1'
    expand1x1_name = f'fire{fire_id}_expand1x1'
    expand3x3_name = f'fire{fire_id}_expand3x3'
    relu_name = f'fire{fire_id}_relu'

    x = layers.Conv2D(squeeze, (1,1), padding='valid', name=squeeze_name)(x)
    x = layers.Activation('relu', name=f'{relu_name}_squeeze1x1')(x)

    expand1x1 = layers.Conv2D(expand, (1,1), padding='valid', name=expand1x1_name)(x)
    expand1x1 = layers.Activation('relu', name=f'{relu_name}_expand1x1')(expand1x1)

    expand3x3 = layers.Conv2D(expand, (3,3), padding='same', name=expand3x3_name)(x)
    expand3x3 = layers.Activation('relu', name=f'{relu_name}_expand3x3')(expand3x3)

    x = layers.concatenate([expand1x1, expand3x3], axis=3, name=f'fire{fire_id}_concat')
    return x
```

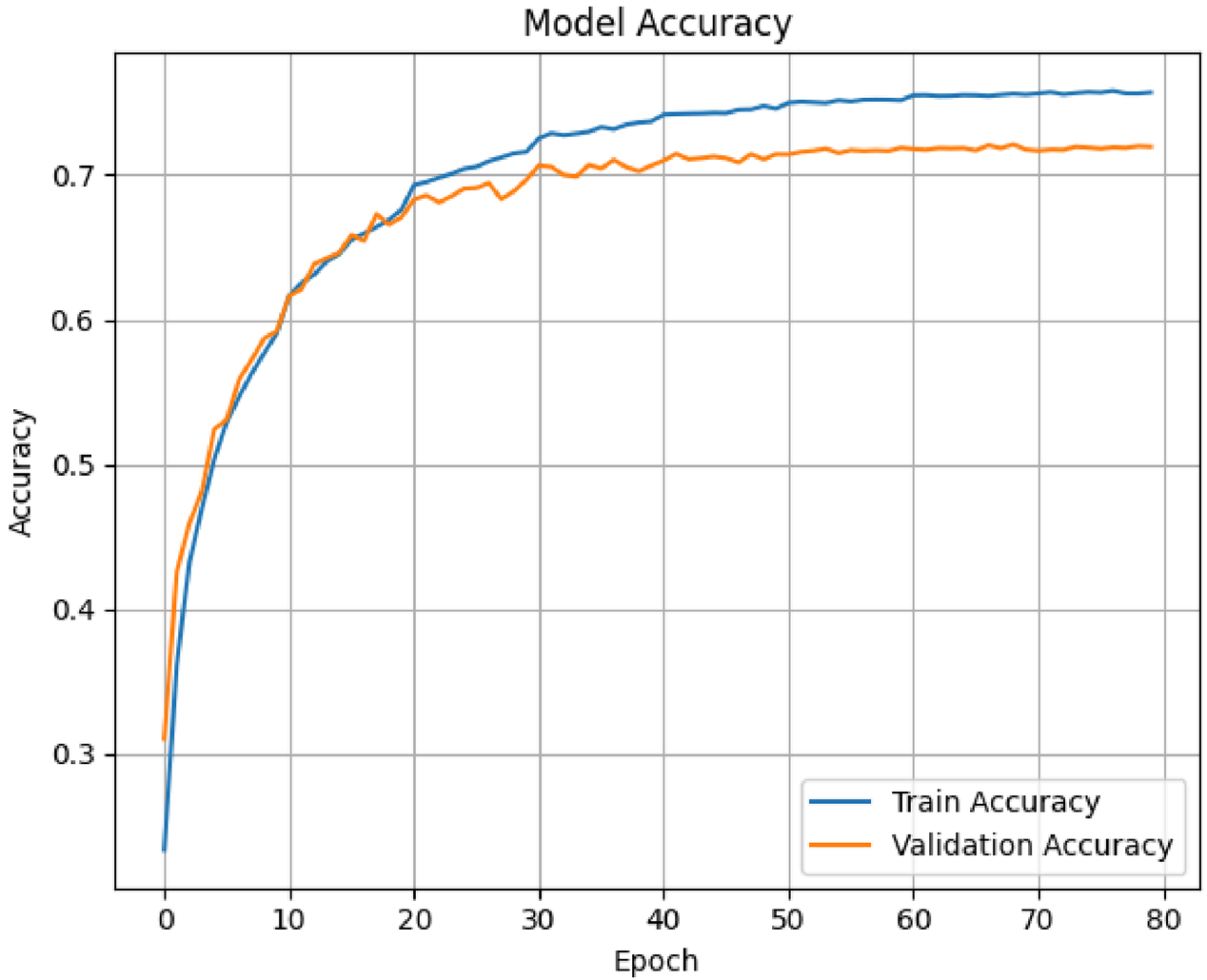
Continued ...

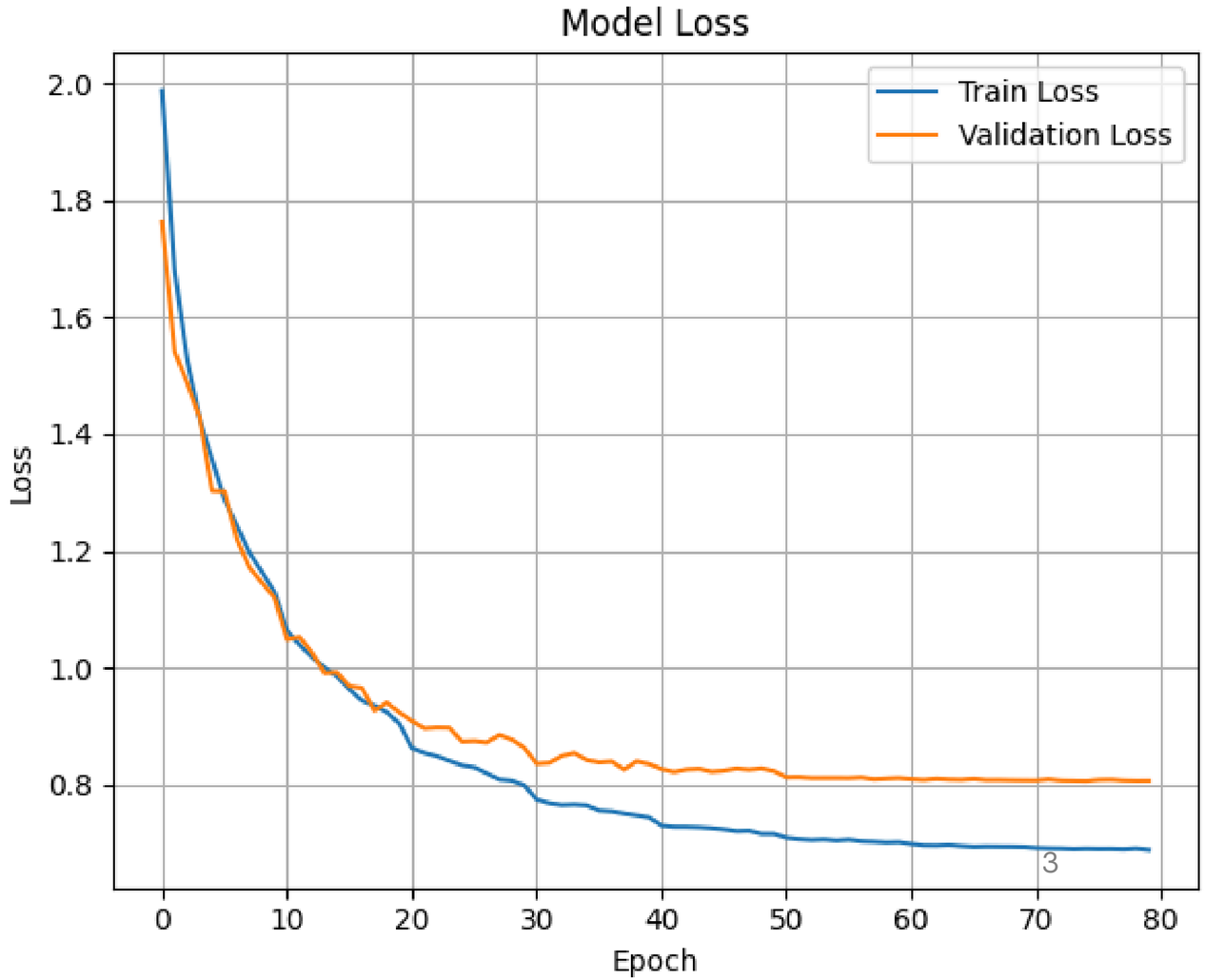
```python
# Build the SqueezeNet model
def SqueezeNet(input_shape=(32,32,3), classes=10):
    input_img = layers.Input(shape=input_shape)

    x = layers.Conv2D(64, (3,3), strides=(2,2), padding='valid', name='conv1')(input_img)
    x = layers.Activation('relu', name='relu_conv1')(x)
    x = layers.MaxPooling2D(pool_size=(3,3), strides=(2,2), name='pool1')(x)

    x = fire_module(x, fire_id=2, squeeze=16, expand=64)
    x = fire_module(x, fire_id=3, squeeze=16, expand=64)

    x = fire_module(x, fire_id=4, squeeze=32, expand=128)
    x = fire_module(x, fire_id=5, squeeze=32, expand=128)
    x = layers.Dropout(0.5, name='drop9')(x)

    x = layers.Conv2D(classes, (1,1), padding='valid', name='conv10')(x)
    x = layers.Activation('relu', name='relu_conv10')(x)
    x = layers.GlobalAveragePooling2D()(x)
    output = layers.Activation('softmax', name='softmax')(x)

    model = models.Model(inputs=input_img, outputs=output, name='squeezenet')
    return model
```

Results

Final Validation Accuracy: ≈ 71%

Final Validation Loss: 0.83





Why Use SqueezeNet?

- Tiny model size, even smaller with compression
- Fast training and inference
- Ideal for mobile/edge AI use cases
- Strong accuracy despite being lightweight

SqueezeNet: Strengths vs. Limitations

Why It Works Well	Limitations
Very small model size (as low as 0.5 MB with compression)	Lower accuracy than larger models on complex datasets
Achieves AlexNet-level accuracy with 50× fewer parameters	Harder to fine-tune due to tight parameter budget
Great for edge devices, FPGAs, and mobile deployment	Requires careful initialization and tuning to converge well
Fully convolutional → works on varying input sizes	No fully connected layers → may limit flexibility in some use cases

The End