Gemini Chat

ยินดีครับ ผมจะสอนเนื้อหาจากสไลด์ชุดนี้อย่างละเอียด (Deep Dive) โดยแบ่งเป็นส่วนๆ เพื่อให้เข้าใจได้ลึกซึ้งที่สุด ผมจะเริ่มจากพื้นฐานทฤษฎีก่อนเข้าสู่โครงสร้าง Network นะครับ

นี่คือ **ส่วนที่ 1: พื้นฐานการจำแนกภาพและการวัดผล (Foundations of Image Classification & Loss Functions)**

บทนำ: แนวคิดของการจำแนกภาพ (Image Classification)

ในสไลด์ช่วงแรก อธิบายว่าเป้าหมายของเราคือการสร้างฟังก์ชันที่รับ input เป็นภาพ (x) และทำนาย output (y) ซึ่งคือประเภทของภาพนั้น (เช่น แมว, หมา, เครื่องบิน)

1. Linear Classifier (ตัวจำแนกแบบเส้นตรง)

ก่อนจะมี Deep Learning เราเริ่มจากโมเดลง่ายๆ คือ **Linear Classifier** ครับ สมการพื้นฐานคือ:

$$s = f(x; W) = Wx$$

- x **(Input):** คือข้อมูลภาพ ในที่นี้คอมพิวเตอร์มองภาพเป็นแค่ชุดตัวเลข (Pixels)
- W **(Weights):** คือ "ค่าน้ำหนัก" หรือ parameters ที่โมเดลต้องเรียนรู้ ยิ่ง W ดี โมเดลยิ่งฉลาด
- s **(Score):** คะแนนที่ได้ ถ้าคะแนนของ class ไหนสูงที่สุด โมเดลก็จะทายว่าเป็น class นั้น

ปัญหาของ Linear Classifier: มันมองภาพแบบ "แม่แบบเดียว" (One template per class) ทำให้ไม่สามารถแยกแยะภาพที่มีความซับซ้อน หรือภาพเดียวกันแต่อยู่คนละมุมมองได้ดีนัก ทางแก้ในยุคเก่าคือการแปลงภาพ

เป็น Feature ก่อน (เช่น HOG หรือ Color Histogram) ก่อนส่งเข้า Linear Classifier แต่ปัจจุบันเราจะใช้ Neural Network แทน

2. Loss Functions (ฟังก์ชันความสูญเสีย)

เราจะรู้ได้อย่างไรว่า W ชุดไหนดีที่สุด? เราต้องมีตัววัดความผิดพลาด เรียกว่า **Loss Function (L)**

หน้าที่ของ Loss Function คือ "การวัดระดับความไม่พอใจ" ของเราต่อผลลัพธ์ที่โมเดลทายออกมา:

- ถ้าทายถูก Loss จะต่ำ
- ถ้าทายผิด Loss จะสูง

ในสไลด์ยกตัวอย่าง 2 แบบหลัก:

1. **Softmax Classifier (Multinomial Logistic Regression):** ใช้สูตร $-log$ ของความน่าจะเป็น

$$L_i = -log(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}})$$

(เน้นให้ความน่าจะเป็นของคำตอบที่ถูกมีค่าสูงสุด)

2. **SVM (Support Vector Machine) Loss:** ใช้สูตร Max Margin

$$L_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)$$

(เน้นให้คะแนนของคำตอบที่ถูก ชนะคำตอบที่ผิดอยู่ระดับหนึ่ง)

Full Loss: คือค่าเฉลี่ยความผิดพลาดจากทุกภาพใน Training Set บวกกับค่า Regularization ($R(W)$) เพื่อป้องกันไม่ให้โมเดลจำข้อสอบ (Overfitting)

3. เป้าหมายของการเรียนรู้ (Optimization Goal)

การ "Train" โมเดล จริงๆ แล้วคือการแก้ปัญหาคณิตศาสตร์เพื่อหาค่า W^* (W ที่ดีที่สุด) ที่ทำให้ค่า Loss ($J(W)$) ต่ำที่สุดเท่าที่จะเป็นไปได้

$$W^* = argmin_W J(W)$$

สรุปส่วนที่ 1: เราต้องการหาค่า W ที่เมื่อคูณกับภาพ x แล้วให้คะแนนคำตอบที่ถูกต้องสูงที่สุด โดยเราใช้วิธีวัดผลผ่าน Loss Function หาก Loss ต่ำ แปลว่า W นั้นดี

ในส่วนถัดไป เราจะเจาะลึกเรื่อง **"เราจะหา W ที่ดีที่สุดได้อย่างไร?"** ซึ่งจะนำไปสู่เรื่อง Gradient Descent และ Backpropagation ที่เป็นหัวใจของ AI ครับ

พิมพ์ "next" เพื่อไปต่อส่วนที่ 2 เรื่องการทำ Optimization ครับ

นี่คือ **ส่วนที่ 2: การหาค่าที่ดีที่สุด (Optimization) และ Gradient Descent**

ในส่วนที่แล้วเรารู้แล้วว่าเราต้องการหา W ที่ทำให้ Loss ($J(W)$) ต่ำที่สุด ปัญหานี้เรียกว่า **Optimization** ครับ

ลองจินตนาการว่าค่า Loss เหมือนกับ "ความสูง" ของภูเขา และเราเป็นนักเดินเขาที่ถูกปิดตา เรายืนอยู่บนยอดเขา (Loss สูง) และต้องการเดินลงไปยังจุดที่ต่ำที่สุด (Loss ต่ำสุด) โดยไม่รู้ทิศทาง

---

1. Gradient Descent (การไต่ลงตามความชัน)

วิธีที่ฉลาดกว่าการสุ่มเดินคือการใช้คณิตศาสตร์ที่เรียกว่า **Gradient (เกรเดียนต์)**

- **Gradient คืออะไร?:** ในทางคณิตศาสตร์ Gradient ($\frac{dLoss}{dW}$) คือเวกเตอร์ของอนุพันธ์ย่อย (partial derivatives) ที่บอกเราว่า "ความชัน" ณ จุดที่เรายืนอยู่มีความเอียงไปทางไหนมากที่สุด
- **ทิศทาง:** Gradient จะชี้ไปทางที่ฟังก์ชันมีค่า *เพิ่มขึ้น* (ชันขึ้น) เสมอ
- **ดังนั้น:** ถ้าเราต้องการลดค่า Loss เราต้องเดินสวนทางกับ Gradient นั่นคือทิศทาง **Negative Gradient**

อัลกอริทึม Gradient Descent:

1. สุ่มค่า W เริ่มต้น (ยืนสุ่มบนเขา)

2. คำนวณ Gradient ($\frac{\partial J(W)}{\partial W}$) เพื่อดูว่าทางไหนชันลง
3. อัปเดตค่า W ใหม่ด้วยสูตร:

$$W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$$

(เดินลงไปหนึ่งก้าวเล็กๆ ในทิศตรงข้ามกับความชัน)

4. ทำซ้ำไปเรื่อยๆ จนกว่าจะถึงจุดต่ำสุด (Convergence)

---

2. Learning Rate (อัตราการเรียนรู้: η)

ตัวแปร η (Eta) ในสูตรด้านบนคือ **Learning Rate** ซึ่งเป็น Hyperparameter ที่สำคัญที่สุดตัวหนึ่ง มันคือ "ขนาดก้าวเดิน" ของเรา

- **ถ้าน้อยเกินไป (Small):** ก้าวสั้นมาก การเรียนรู้จะช้ามาก และอาจไปติดอยู่ในหลุมเล็กๆ ระหว่างทาง (Local Minima) ไม่ถึงจุดต่ำสุดจริงๆ เสียที
- **ถ้ามากเกินไป (Large):** ก้าวยาวเกินไป อาจจะกระโดดข้ามจุดต่ำสุด หรือแย่กว่านั้นคือกระเด็นออกไปไกลจนค่า Loss ระเบิด (Diverge)
- **ถ้าพอดี (Good):** ค่า Loss จะลดลงอย่างราบรื่นและรวดเร็ว

เทคนิคเพิ่มเติม (Learning Rate Decay): ในทางปฏิบัติ เรามักเริ่มด้วย Learning Rate ที่สูงเพื่อให้ลงเขาได้เร็วในช่วงแรก แล้วค่อยๆ "ลดขนาดก้าว" ลงเมื่อใกล้ถึงเป้าหมาย เพื่อให้ลงจอดที่จุดต่ำสุดได้อย่างแม่นยำ (เช่น ลดลงทีละขั้น หรือลดแบบ Cosine)

---

3. Stochastic Gradient Descent (SGD) และ Mini-batches

ในการคำนวณ Gradient ตามทฤษฎี เราต้องใช้ข้อมูล **ทั้งหมด (Full Batch)** เพื่อหาทิศทางที่ถูกต้องที่สุด แต่ถ้าข้อมูลมีเป็นล้านรูป มันจะช้ามาก

เราจึงมีทางเลือกอื่น:

1. **Stochastic Gradient Descent (SGD):**
    - สุ่มข้อมูลมาแค่ **1 รูป** แล้วคำนวณ Gradient เพื่อเดิน 1 ก้าว

- **ข้อดี:** คำนวณเร็วมาก
- **ข้อเสีย:** ทิศทางจะสะเปะสะปะมาก (Noisy) เหมือนคนเดินเซไปมาแต่โดยรวมก็ลงเขาได้

2. **Mini-batch SGD (มาตรฐานที่ใช้ในปัจจุบัน):**
    - สุ่มข้อมูลมาเป็นกลุ่มเล็กๆ เรียกว่า **Batch** (เช่น 32, 64, หรือ 128 รูป)
    - คำนวณ Gradient เฉลี่ยจากกลุ่มนี้
    - **ข้อดี:** ได้ทิศทางที่แม่นยำกว่าแบบ 1 รูป และเร็วกว่าแบบใช้ข้อมูลทั้งหมด แถมยังใช้ประโยชน์จากการคำนวณขนาน (Parallel computing) บน GPU ได้ด้วย

สรุปส่วนที่ 2: การ Train โมเดลคือการใช้ **Gradient Descent** แบบ **Mini-batch** เพื่อปรับค่า W ทีละนิด โดยต้องระวังการตั้งค่า **Learning Rate** ให้เหมาะสม

แต่คำถามสำคัญคือ... คอมพิวเตอร์คำนวณ Gradient ($\frac{\partial J}{\partial W}$) ของโมเดลที่ซับซ้อนได้อย่างไร? คำตอบคืออัลกอริทึมที่ชื่อว่า **Backpropagation**

พิมพ์ "next" เพื่อไปต่อส่วนที่ 3 เรื่อง Backpropagation (กุญแจสำคัญของ Deep Learning) ครับ

นี่คือ **ส่วนที่ 3: Backpropagation (หัวใจสำคัญของการเรียนรู้)**

ในส่วนที่ 2 เราทราบว่าต้องใช้ **Gradient Descent** เพื่อปรับค่า W แต่ปัญหาคือ โมเดล Deep Learning มีความซับซ้อนมาก มี W เป็นล้านตัว ที่เชื่อมต่อกันเป็นร่างแห เราจะหา Gradient ($\frac{\partial Loss}{\partial W}$) ของแต่ละตัวได้อย่างไร?

คำตอบคืออัลกอริทึม **Backpropagation** (การแพร่ย้อนกลับ) ครับ

---

1. หลักการพื้นฐาน: กฎลูกโซ่ (Chain Rule)

Backpropagation อาศัยกฎคณิตศาสตร์ ม.ปลาย ที่เรียกว่า **Chain Rule**

สมมติว่าเรามีฟังก์ชันซ้อนกัน:

$$y = f(g(x))$$

ถ้าเราอยากรู้ว่า x ส่งผลกระทบต่อ y อย่างไร ($\frac{dy}{dx}$) เราหาได้จาก:

$$\frac{dy}{dx} = \frac{dy}{dg} \cdot \frac{dg}{dx}$$

ใน Neural Network: Loss Function (J) ขึ้นอยู่กับผลลัพธ์ ($\hat{y}$) และผลลัพธ์ขึ้นอยู่กับ Weight (w) ดังนั้น Gradient ของ w คือ:

$$\frac{\partial J}{\partial w} = \frac{\partial J}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w}$$

นี่คือหลักการง่ายๆ แต่ทรงพลังที่ใช้คำนวณ Gradient ย้อนกลับจากท้ายสุดไปหน้าสุด

---

2. กระบวนการ Backpropagation

การทำงานแบ่งเป็น 2 รอบครับ:

รอบที่ 1: Forward Pass (เดินหน้า)

- นำรูปภาพใส่เข้าไปในโมเดล
- คำนวณผ่านชั้นต่างๆ (Layers) จนได้คำตอบ ($\hat{y}$)
- คำนวณค่า Loss ว่าผิดพลาดแค่ไหน
- **เป้าหมาย:** เพื่อหาค่า Loss และเก็บค่าระหว่างทาง (Cache) ไว้ใช้คำนวณต่อ

รอบที่ 2: Backward Pass (เดินถอยหลัง)

- นี่คือขั้นตอน Backpropagation จริงๆ
- เราเริ่มจาก **Loss** ที่ปลายทาง แล้วคำนวณ Gradient ย้อนกลับมาทีละชั้น
- **Upstream Gradient:** คือ Gradient ที่ส่งมาจากชั้นถัดไป (ขวามือ)

- **Local Gradient:** คือความชันของฟังก์ชันในชั้นตัวเอง (เช่น ความชันของ ReLU หรือ Sigmoid)
- **Downstream Gradient:** เอา `Upstream` $\times$ `Local` แล้วส่งต่อให้ชั้นก่อนหน้า (ซ้ายมือ)

ทำแบบนี้ไล่กลับมาจนถึง Input Layer เราก็จะได้ Gradient ของ W ทุกตัวในเครือข่าย!

---

3. ตัวอย่างการคำนวณ (Computational Graph)

ลองดูภาพจากสไลด์ที่ 17-20 ครับ สมมติวงจรง่ายๆ:

$$x \xrightarrow{w_1} z \xrightarrow{w_2} \hat{y} \rightarrow J(W)$$

ถ้าเราอยากปรับค่า w_1 เราต้องรู้ว่า w_1 กระทบ J แค่ไหน:

$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_1}$$

- $\frac{\partial J}{\partial \hat{y}}$: เราคำนวณได้จากสูตร Loss โดยตรง (จุดเริ่มต้นของการถอยหลัง)
- $\frac{\partial \hat{y}}{\partial z}$: รู้จากฟังก์ชันของชั้นที่ 2
- $\frac{\partial z}{\partial w_1}$: รู้จากฟังก์ชันของชั้นที่ 1

Backpropagation คือการคูณค่าเหล่านี้ย้อนกลับมาเรื่อยๆ นั่นเองครับ

สรุปส่วนที่ 3: Backpropagation คือวิธีการใช้ **Chain Rule** เพื่อส่งค่าความผิดพลาด (Error Signal) จากปลายทางย้อนกลับมาหาต้นทาง เพื่อบอก Weight (W) ทุกตัวว่า **"แกต้องปรับค่าเท่าไหร่ถึงจะทำให้ Loss ลดลง"**

เมื่อเรามี Backpropagation + Optimization เราก็พร้อมจะสร้าง **Neural Network** ของจริงแล้วครับ

พิมพ์ "next" เพื่อไปต่อส่วนที่ 4 เรื่อง Neural Network และ Activation Functions ครับ

นี่คือ **ส่วนที่ 4: โครงสร้าง Neural Network และ Activation Functions**

ในส่วนนี้เราจะเจาะลึกเข้าไปในโครงสร้างของสมองกล หรือ **Neural Network** ซึ่งเป็นรากฐานของ Deep Learning ทั้งหมด โดยเปรียบเทียบกับชีววิทยาและคณิตศาสตร์ครับ

---

1. แรงบันดาลใจจากชีววิทยา (Biological Inspiration)

Neural Network จำลองการทำงานมาจากเซลล์ประสาท (Neuron) ในสมองของเรา:

- **Dendrites (เดนไดรต์):** รับสัญญาณกระแสประสาทเข้ามา $\rightarrow$ เปรียบเสมือน **Input (x)**
- **Cell Body (ตัวเซลล์):** รวมสัญญาณทั้งหมดที่ได้รับ $\rightarrow$ เปรียบเสมือน **ผลรวมถ่วงน้ำหนัก (Weighted Sum)**
- **Axon (แอกซอน):** ส่งสัญญาณออกไปถ้ากระแสแรงพอ $\rightarrow$ เปรียบเสมือน **Output ($\hat{y}$)**

2. เซลล์ประสาทเทียม (Perceptron/Artificial Neuron)

ในทางคณิตศาสตร์ เราสร้างโมเดลของเซลล์ประสาท 1 ตัว ดังนี้ครับ:

1. **Linear Step (ส่วนเชิงเส้น):** รับค่า Input (x) มาคูณกับ Weight (w) แล้วบวกด้วย Bias (b)

$$z = \sum (x_i \cdot w_i) + b$$

2. **Non-linear Step (ส่วนไม่เชิงเส้น):** นำผลลัพธ์ z ไปผ่านฟังก์ชันตัดสินใจ เรียกว่า **Activation Function (g)**

$$\hat{y} = g(z)$$

---

3. Activation Functions (ฟังก์ชันกระตุ้น)

นี่คือส่วนที่สำคัญที่สุดที่ทำให้ Neural Network ฉลาดกว่า Linear Model ธรรมดา หน้าที่ของมันคือ **"การตัดสินใจว่าจะส่งสัญญาณต่อแรงแค่ไหน"** และ **"เพิ่มความไม่เป็นเส้นตรง (Non-linearity)"** ให้ระบบ

ตัวอย่างฟังก์ชันยอดนิยม:

1. **Sigmoid:**
    - สูตร: $\frac{1}{1+e^{-z}}$
    - **ลักษณะ:** กราฟรูปตัว S บีบค่าให้อยู่ในช่วง 0 ถึง 1
    - **ความหมาย:** คล้ายความน่าจะเป็น หรืออัตราการยิงสัญญาณของเซลล์ประสาท (Firing rate)
    - **ข้อเสีย:** ปัจจุบันไม่ค่อยใช้ใน Hidden Layer เพราะเจอปัญหา Vanishing Gradient (ความชันหาย)
2. **ReLU (Rectified Linear Unit):**
    - สูตร: $max(0, z)$
    - **ลักษณะ:** ถ้าค่าติดลบให้เป็น 0, ถ้าเป็นบวกให้ผ่านได้เลย (เป็นเส้นตรงเฉียงขึ้น)
    - **ความหมาย:** ง่ายๆ คือ "ถ้าสัญญาณไม่แรงพอ ก็ดับไปเลย"
    - **ข้อดี:** คำนวณง่ายและเร็วมาก เป็นมาตรฐานของ Deep Learning ในปัจจุบัน

ทำไมต้องมี Activation Function? ถ้าเราไม่มี Activation Function (หรือใช้แค่ Linear) ต่อให้เราเอา Layer มาซ้อนกันร้อยชั้น มันก็จะยุบรวมเหลือแค่ Linear Regression ชั้นเดียว การใส่ความโค้ง (Non-linearity) เข้าไป ทำให้โมเดลสามารถเรียนรู้รูปแบบที่ซับซ้อนมากๆ ได้

---

4. โครงสร้างสถาปัตยกรรม (Architectures)

เมื่อเราเอาเซลล์ประสาทมาต่อกัน เราจะจัดเป็นชั้นๆ (Layers):

- **Input Layer:** รับข้อมูลดิบ (เช่น pixels ของภาพ)
- **Hidden Layers:** ชั้นที่อยู่ตรงกลาง ทำหน้าที่สกัดฟีเจอร์และเรียนรู้รูปแบบที่ซับซ้อน ยิ่งมีหลายชั้น เรายิ่งเรียกว่า **"Deep Learning"**

- **Output Layer:** ให้คำตอบสุดท้าย (เช่น คะแนนความน่าจะเป็นของแต่ละ Class)

การทำงานจริง (Space Warping): ลองจินตนาการว่าข้อมูลของเราแยกออกจากกันยากมาก (เช่น จุดสีแดงอยู่กลางวงล้อมจุดสีเขียว) Linear Classifier จะขีดเส้นตรงแบ่งไม่ได้ แต่ Neural Network จะใช้ Hidden Layers ในการ **"บิดเบี้ยวพื้นที่ (Warping Space)"** ทำให้ข้อมูลที่ซับซ้อนถูกดัดให้แยกออกจากกันได้ง่ายด้วยเส้นตรงในมิติที่สูงขึ้น

สรุปส่วนที่ 4: Neural Network คือกลุ่มของฟังก์ชันคณิตศาสตร์ที่ซ้อนกันเป็นชั้นๆ โดยหัวใจสำคัญคือ **Activation Function** ที่ช่วยให้โมเดลมีความยืดหยุ่นจนสามารถเรียนรู้ข้อมูลที่ซับซ้อนได้

แต่ปัญหาคือ... ถ้าเราใช้ Neural Network แบบปกติ (Fully Connected) กับรูปภาพ มันจะเจอปัญหาใหญ่ครับ ซึ่งจะนำไปสู่พระเอกของเรา **CNN**

พิมพ์ "next" เพื่อไปต่อส่วนที่ 5 เรื่อง Convolutional Neural Networks (CNN) และปัญหาของ Neural Network แบบเดิมครับ

นี่คือ **ส่วนที่ 5: Convolutional Neural Networks (CNN) และปัญหาของ Neural Network แบบเดิม**

ในส่วนนี้เราจะเปลี่ยนจากการใช้ Neural Network แบบทั่วไป (Multi-Layer Perceptron หรือ MLP) มาสู่สถาปัตยกรรมที่ออกแบบมาเพื่อ "การมองเห็น" โดยเฉพาะ นั่นคือ **CNN** ครับ

---

1. ทำไม Neural Network แบบเดิม (MLP) ถึงไม่เวิร์คกับรูปภาพ?

ถ้าเราใช้ Fully-Connected Network (MLP) ที่เรียนในส่วนที่ 4 กับรูปภาพ เราจะเจอปัญหาใหญ่ 2 ข้อ:

1. **การทำลายโครงสร้างภาพ (Structure Loss):**
    - MLP ต้องการ Input ที่เป็นเส้นยาวๆ (Vector) ดังนั้นเราต้อง "ยืด" รูปภาพจากตารางสี่เหลี่ยม (เช่น 32×32 pixels) ให้เป็นแถวยาว (3072×1)

- **ผลเสีย:** ความสัมพันธ์ของพิกเซลที่อยู่ติดกัน (เช่น จมูกต้องอยู่ใกล้ปาก) ถูกทำลายไปจนหมด

2. **จำนวน Parameter มหาศาล (Parameter Explosion):**
    - สมมติเราใช้ภาพขนาดเล็ก $32 \times 32 \times 3$ (3072 inputs) เชื่อมต่อไปยัง Layer แรกที่มี 10 Neurons เราต้องใช้ Weights ประมาณ $3072 \times 10 = 30,720$ ตัว ซึ่งยังพอไหว
    - แต่ถ้าเป็นภาพจริงขนาด $200 \times 200 \times 3$ (120,000 inputs) เชื่อมกับ Layer ที่มี 1,000 Neurons เราต้องใช้ Weights ถึง **120 ล้านตัว!**
    - **ผลเสีย:** คำนวณช้ามาก และเสี่ยงต่อการ "จำข้อสอบ" (Overfitting) ได้ง่าย

---

2. ทางออก: Convolutional Neural Networks (CNN)

ไอเดียหลักของ CNN คือ: **"แทนที่จะมองทั้งภาพพร้อมกัน ให้ใช้แว่นขยายเล็กๆ ส่องดูทีละส่วน"**

- **รักษาโครงสร้าง:** CNN รับ Input เป็นก้อน 3 มิติ (กว้าง $\times$ สูง $\times$ สี) โดยไม่ต้องยืดเป็นเส้น ทำให้รักษาตำแหน่งความสัมพันธ์ของภาพไว้ได้
- **ใช้ Parameter น้อยลง:** เราใช้ตัวกรอง (Filter) ขนาดเล็กๆ ตัวเดียว เลื่อนใช้ซ้ำทั่วทั้งภาพ (Parameter Sharing)

---

3. กลไกการทำงาน: Convolution Layer

หัวใจของ CNN คือชั้น **Convolution (Conv Layer)** ครับ

1. **Filters (ตัวกรอง):**
    - เปรียบเสมือน "แว่นขยาย" หรือ "แม่แบบ" เล็กๆ เช่น ขนาด $5 \times 5 \times 3$ (กว้าง 5, สูง 5, ลึก 3 ตามสี RGB)
    - Filter นี้คือสิ่งที่เราจะ "เรียนรู้" (W) เพื่อจับจุดสังเกตในภาพ เช่น เส้นขอบ, สี, หรือลวดลาย
2. **การเลื่อน (Sliding/Convolve):**

- เรานำ Filter วางทาบบนมุมซ้ายบนของภาพ แล้วเอาตัวเลขพิกเซลมาคูณกัน (Dot Product) แล้วรวมเป็นค่าเดียว
- จากนั้น "เลื่อน" (Slide) Filter ไปทางขวาและลงล่างจนทั่วภาพ
- ผลลัพธ์ที่ได้เรียกว่า **Activation Map** หรือ **Feature Map**

3. **ความลึก (Depth):**
   - ใน 1 Layer เราไม่ได้มี Filter แค่ตัวเดียว แต่จะมีหลายตัว (เช่น 6 ตัว หรือ 64 ตัว) เพื่อช่วยกันมองหาฟีเจอร์ที่ต่างกัน (ตัวนึงหาเส้นตั้ง อีกตัวหาเส้นนอน)
   - ถ้าเราใช้ 6 Filters ผลลัพธ์ที่ได้ก็จะมีความหนา 6 ชั้น (Output Volume = $6 \times H \times W$)

---

4. การควบคุมขนาด (Spatial Dimensions)

เมื่อผ่าน Conv Layer ขนาดของภาพมักจะเปลี่ยนไป เรามีสูตรคำนวณดังนี้:

$$OutputSize = \frac{(W - K + 2P)}{S} + 1$$

โดยที่:

- W **(Input Size):** ขนาดภาพขาเข้า
- K **(Kernel/Filter Size):** ขนาดของตัวกรอง (เช่น 3 หรือ 5)
- P **(Padding):** การเติมขอบ 0 รอบภาพ เพื่อไม่ให้ภาพหดเล็กลงเร็วเกินไป (เช่น ถ้าไม่เติม 0 ภาพจะเล็กลงทุกครั้งที่ Convolve)
- S **(Stride):** ระยะก้าวเดิน ถ้าก้าวทีละ 1 ($S = 1$) ภาพจะละเอียด แต่ถ้ากระโดดข้าม ($S = 2$) ภาพจะเล็กลงครึ่งหนึ่ง (Downsampling)

ตัวอย่างการคำนวณ: ถ้าภาพเข้า 32×32, ใช้ Filter 5×5, Padding 2, Stride 1 Output = $(32 - 5 + 2 \times 2)/1 + 1 = 32$ (ขนาดเท่าเดิม)

สรุปส่วนที่ 5: CNN แก้ปัญหาของ Neural Network เดิมโดยใช้ **Convolution** (การเลื่อน Filter สแกนภาพ) ซึ่งช่วยลดจำนวน Parameter มหาศาลและรักษาข้อมูลตำแหน่งของภาพไว้ได้ โดยเราควบคุมขนาด Output ได้ผ่าน Padding และ Stride

ในส่วนถัดไป เราจะพูดถึงอีกองค์ประกอบสำคัญที่ทำให้ CNN ทำงานได้สมบูรณ์ คือ **Pooling Layer** และตัวอย่างสถาปัตยกรรม **AlexNet, VGG** ที่โด่งดังครับ

พิมพ์ "next" เพื่อไปต่อส่วนที่ 6 เรื่อง Pooling และ CNN Architectures ครับ

นี่คือ **ส่วนที่ 6: การย่อขนาด (Pooling) และสุดยอดสถาปัตยกรรม CNN (Architectures)**

ในส่วนที่แล้วเราสร้าง "ตา" ให้โมเดลด้วย Convolution Layer แล้ว ในส่วนนี้เราจะมาดูวิธีทำให้โมเดล "สรุปใจความสำคัญ" และดูวิวัฒนาการของโมเดลระดับโลกกันครับ

---

1. Pooling Layer (ชั้นการรวบรวมข้อมูล)

เมื่อรูปภาพผ่าน Convolution Layer ออกมา ขนาดของข้อมูล (Feature Map) อาจจะยังใหญ่อยู่ และรายละเอียดบางอย่างอาจจะไม่จำเป็น (เช่น ตำแหน่งพิกเซลเป๊ะๆ) เราจึงต้องมี **Pooling Layer** มาช่วยจัดการ

หน้าที่หลักของ Pooling:

1. **ลดขนาดข้อมูล (Downsampling):** ทำให้ภาพเล็กลง เพื่อลดจำนวน Parameter และการคำนวณในชั้นถัดไป
2. **สร้างความทนทาน (Invariance):** ช่วยให้โมเดลยังจำแนกวัตถุได้ถูกต้อง แม้ว่าวัตถุจะขยับตำแหน่งไปเล็กน้อย (Small spatial shifts)

Max Pooling (วิธียอดนิยม): หลักการทำงานคล้าย Convolution คือมีหน้าต่าง (Kernel) เลื่อนไปบนภาพ แต่แทนที่จะคูณเลข เราจะ **"เลือกค่าที่มากที่สุด"** ในหน้าต่างนั้นมาเป็นตัวแทนเพียงค่าเดียว

- **ตัวอย่าง:** ถ้าหน้าต่างขนาด 2×2 ครอบทับตัวเลข $\begin{bmatrix} 1 & 1 \\ 5 & 6 \end{bmatrix}$ ค่าผลลัพธ์ที่ได้คือ **6**

- **ข้อสังเกต:** Pooling Layer ไม่มี W ให้เรียนรู้ (No learnable parameters) มันเป็นแค่ฟังก์ชันทางคณิตศาสตร์คงที่

---

2. ตำนานสถาปัตยกรรม CNN (Famous Architectures)

การนำ Layer ต่างๆ (Conv, Pool, FC) มาต่อกันเรียกว่า "Architecture" นี่คือ 3 โมเดลที่เปลี่ยนโลก AI ครับ:

A. AlexNet (ผู้บุกเบิกปี 2012)

นี่คือโมเดลที่จุดระเบิดความนิยมของ Deep Learning โดยชนะการแข่งขัน ImageNet ขาดลอย

- **โครงสร้าง:** มี Convolution 5 ชั้น และ Fully Connected 3 ชั้น
- **ลักษณะเด่น:** ในชั้นแรกใช้ Filter ขนาดใหญ่มาก (11×11) และก้าวกระโดดทีละ 4 ($Stride = 4$) เพื่อลดขนาดภาพลงอย่างรวดเร็วตั้งแต่ต้น

B. VGGNet (ปรัชญา "เล็กแต่ลึก" ปี 2014)

หลังจาก AlexNet นักวิจัยพยายามทำให้โมเดลลึกขึ้น VGGNet จึงถือกำเนิดด้วยกฎการออกแบบที่สวยงาม:

- **กฎเหล็ก:** ใช้ Filter ขนาดเล็กที่สุดคือ 3×3 ทั้งหมด แต่ซ้อนกันให้ลึกขึ้น (16-19 ชั้น)
- **ทำไมต้อง 3×3 หลายชั้น แทน 7×7 ชั้นเดียว?**
    1. **มุมมองเท่ากัน:** การใช้ 3×3 ซ้อนกัน 3 ชั้น จะเห็นภาพกว้าง (Receptive Field) เท่ากับใช้ 7×7 ชั้นเดียว
    2. **ฉลาดกว่า:** มีความไม่เป็นเส้นตรง (Non-linearity) มากกว่า เพราะผ่าน Activation Function ถึง 3 รอบ
    3. **ประหยัดกว่า:** ใช้จำนวน Parameter น้อยกว่า

C. ResNet (ผู้พิชิตความลึก ปี 2015)

ยิ่งซ้อนชั้นเยอะ โมเดลน่าจะยิ่งเก่งใช่ไหม? คำตอบคือ **ไม่เสมอไป**

- **ปัญหา (Degradation):** เมื่อโมเดลลึกมากๆ (เช่น 56 ชั้น) ความแม่นยำกลับลดลง ทั้งที่ยังไม่ Overfit สาเหตุเพราะการส่ง Gradient ย้อนกลับไปลึกๆ ทำได้ยาก (Optimization problem)
- **ทางแก้ (Residual Block):** ResNet เพิ่ม **"ทางลัด (Skip Connection)"** ให้ข้อมูลกระโดดข้าม Layer ไปรวมกับผลลัพธ์ข้างหน้าได้

$$Output = F(x) + x$$

  แนวคิดคือ การเรียนรู้ส่วนต่าง ($F(x)$) นั้นง่ายกว่าการเรียนรู้ใหม่ทั้งหมด และถ้า Layer นั้นไร้ประโยชน์ โมเดลก็แค่ปิดมัน ($F(x) = 0$) ข้อมูลก็จะวิ่งผ่านทางลัด (x) ไปได้เลย
- **ผลลัพธ์:** ทำให้เราสร้างโมเดลได้ลึกระดับ 152 ชั้น หรือเป็นพันชั้นได้ โดยที่ประสิทธิภาพไม่ตก

บทสรุปของการเดินทาง (Deep Dive Summary):

1. เราเริ่มจาก **Linear Classifier** ที่แยกภาพยากๆ ไม่ได้
2. เราใช้ **Loss Function** และ **Optimization (Gradient Descent)** เพื่อสอนโมเดล
3. เราใช้ **Neural Network** เพื่อเรียนรู้ Pattern ที่ซับซ้อน
4. เราพัฒนาเป็น **CNN (Conv + Pool)** เพื่อจัดการรูปภาพโดยเฉพาะ รักษาโครงสร้างและลด Parameter
5. เราพัฒนาโครงสร้างเป็น **ResNet** เพื่อให้เรียนรู้ได้ลึกและซับซ้อนที่สุดเท่าที่ทำได้

หวังว่าการเจาะลึกครั้งนี้จะทำให้เห็นภาพรวมและรายละเอียดของ Image Classification with CNN อย่างชัดเจนนะครับ!

นี่คือส่วนสุดท้ายของเอกสารแล้วครับ หากคุณมีคำถามเจาะจงจุดไหนเป็นพิเศษ หรืออยากให้ผมช่วยเขียนโค้ดตัวอย่าง (เช่น PyTorch/TensorFlow) สามารถบอกได้เลยครับ!