Deep Learning

Revolution and Modern Use Cases

Outline

Revolution of Computer Vision Domain over Time and Improvements

Revolution in Natural Language Processing Domain over Time and Improvements

Hands on Example with Latest CNN and VIT

Hands on Example with LangChains, LanGraphs and Vector Databases

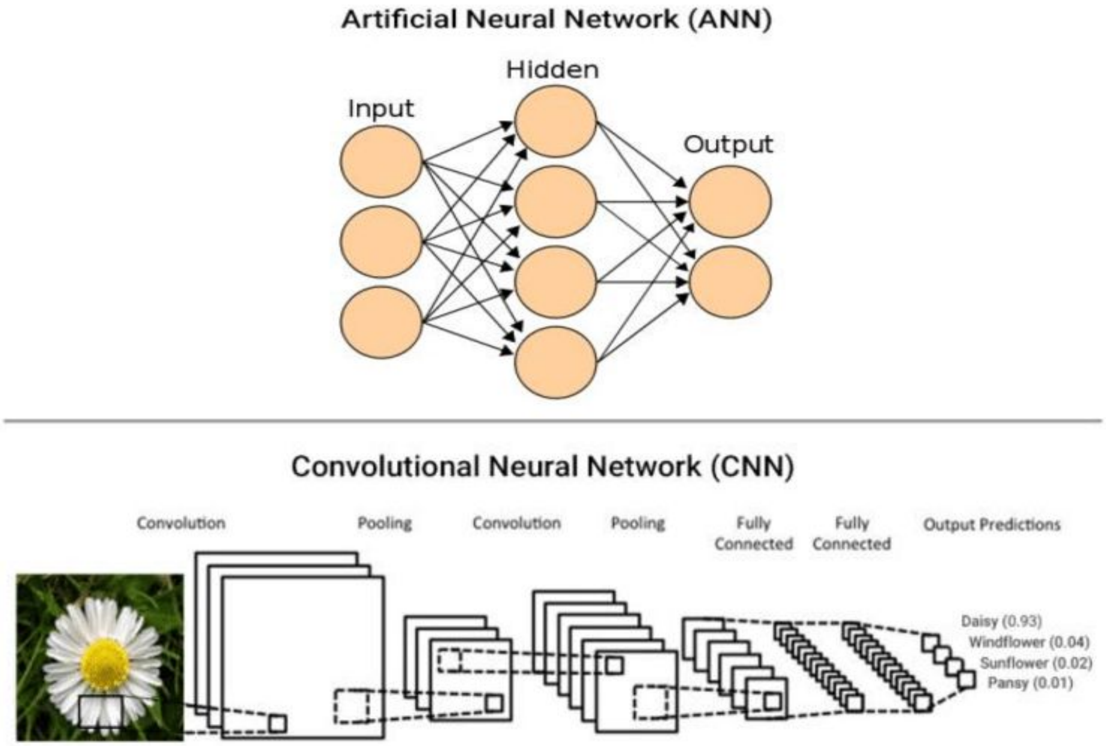
After this Lecture Students will be able to

Choose Right CNN Architecture for their Task

Run LLMs Locally with Ollama and on Cloud Ubuntu CLI

Use Prompt Engr., LangChains, Langraphs and Vector Databases

ANN and CNN


Artificial Neural Network (ANN)
Hidden
Input
Output
Convolutional Neural Network (CNN)
Convolution
Pooling
Convolution
Pooling
Fully Connected
Fully Connected
Output Predictions
Daisy (0.93)
Windflower (0.04)
Sunflower (0.02)
Pansy (0.01)

ANN and CNN

Why we Need CNN

Too Many Parameters:

For a 100×100 pixel image (10,000 inputs) by simple Flattening , even one hidden layer with 1,000 neurons means 10 million parameters — too large to train efficiently.

ANN and CNN

Why we Need CNN

Loss of Spatial Structure:

A dense network treats every pixel as independent, ignoring the fact that nearby pixels are related (e.g., eyes are near the nose in a face).

⚠️ All Methods we will discuss revolve around how can we

Extract, **Preserve** and **Propagate** Information from Images

ANN and CNN

Why we Need CNN

Overfitting and Poor Generalization:

Because of the high number of parameters, these networks easily memorized data instead of learning meaningful patterns.

ANN and CNN

When we had First CNN **Paper**

Yann LeCun, in the late 1980s and early 1990s, at AT&T Bell Labs.

Neural Network designed specifically for grid-like data such as images.

Digit Recognition specifically to read postal ZIP codes, bank checks, and other digit-based documents automatically.

Hand DD is still Unsolved

ANN and CNN

Key Innovations

Weight sharing: The same filter (kernel) is used across the whole image

Introduced **Convolution** + **Pooling** + **Fully Connected Layers** Structure

Used **Backpropagation** for Training CNNs

ANN and CNN

Problems Solved

Reduced Computational Cost by Convolution and Max Pooling Layers

Maintained Spatial Information Across NN

Model is more Generalized due to considering pixel neighbour information and translation invariance

Modern CNN

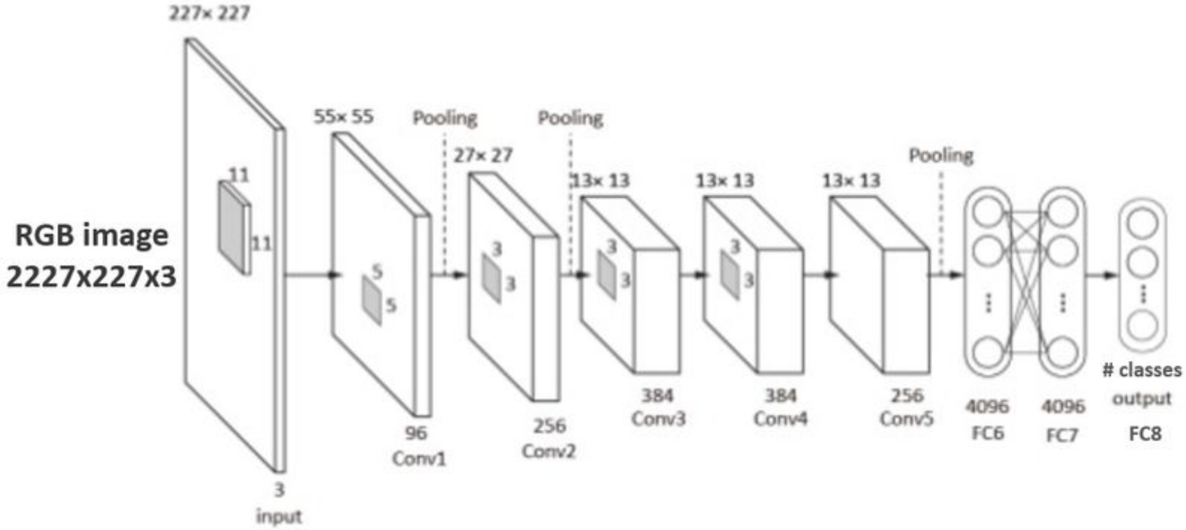
2012 **ALEXNET** **Paper**

Developed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton

Won the ImageNet competition (**ILSVRC** 2012)

Modern CNN

2012 **ALEXNET**



Modern CNN

2012 **ALEXNET**

ReLU (Rectified Linear Unit) **$f(x)=max(0,x)$**

Solved:

Vanishing Gradient Problem

Increase Training Speed (6x)

Modern CNN

2012 **ALEXNET**

Dropout randomly turn off (drop) some neurons with a probability **P**

Solved:

Overfitting

Better Generalization

Modern CNN

2012 **ALEXNET**

GPU Training (massive speed boost)

— making deep learning practical for the first time.

Modern CNN

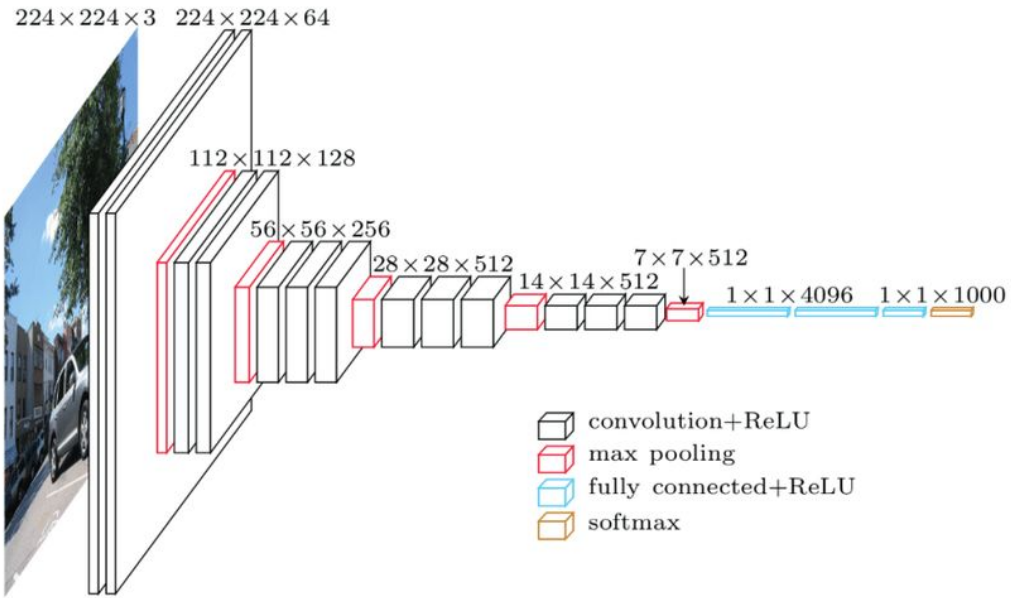
2014 **VGGNet** [Paper]

VGGNet (2014), developed by Karen Simonyan and Andrew Zisserman at the University of Oxford’s Visual Geometry Group (VGG)

VGG-16 and VGG-19 (16 or 19 Layers Deep)

Modern CNN

2014 **VGGNet** More Deeper for High Res Images



Modern CNN

2014 **VGGNet**

Using multiple **Small Filters** instead of one large one captures more complex patterns with fewer parameters.

One 7×7 filter 49 parameters per Channel

Three stacked 3×3 filters 27 parameters, but deeper and more Expressive

Stacking small filters = more **Non-Linearity** and **Larger Receptive Field**

Modern CNN

2014 **VGGNet**

Using multiple **Small Filters** instead of one large one captures more complex patterns with fewer parameters.

One 7×7 filter 49 parameters per Channel

Three stacked 3×3 filters 27 parameters, but deeper and more Expressive

Stacking small filters = more **Non-Linearity** and **Larger Receptive Field**

Modern CNN

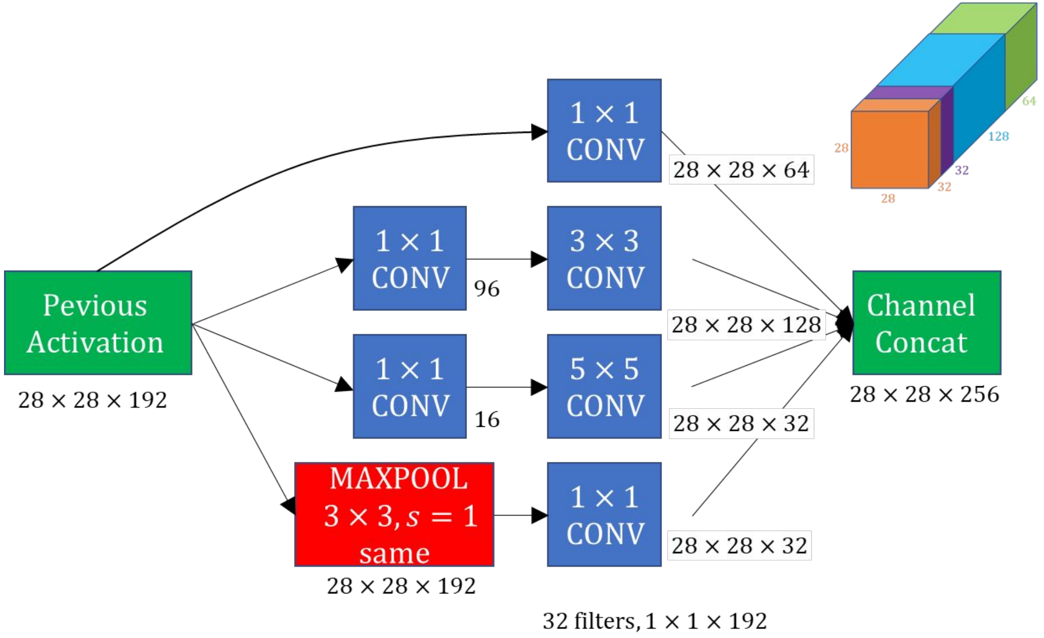
2014 **Google Net** **by Google Research Team** **[Paper]**

Core Idea — “**Inception Module**”

It’s a **mini-network inside a network** that processes input at multiple scales simultaneously.

Modern CNN

2014 **Google Net** **by Google Research Team**



Muhammed Hassan Mukhtar

Modern CNN

2014 **Google Net** **by Google Research Team**

1×1 Convolutions (Bottleneck) or PointWise / DepthWise Conv

size = 28 × 28 × 256 You apply 64 filters, each of size 1×1×256 Output: 28 × 28 × 64

Enables cross-channel interaction

Because it processes each spatial point (pixel) separately it doesn't look at Neighboring

Modern CNN

2015 **ResNet** **by Microsoft Research Team** **Paper**

Won the ImageNet **2015** challenge with a Top-5 error rate of 3.57%,

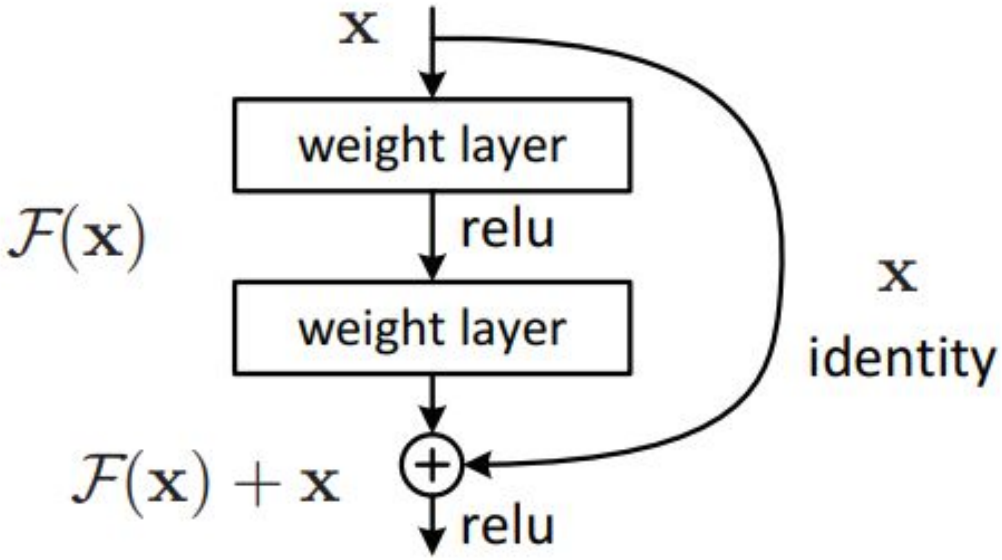
Network were getting Deeper causing

Vanishing / Exploding Gradients

Optimization Problem

Modern CNN

2015 **ResNet** by Microsoft Research Team



Modern CNN

2015 **Inception-v3** **by Google Research Team** **Paper**

1. Batch Normalization Normalizes the Activations within each **mini-Batch**
2. Factorized Convolutional Divide large Conv to small 3 x 3 to 1 x 3 and 3 x 1
3. RMSProp Optimizer **Avoid SDG**
4. Label Smoothing Don't use One Hot Encoding
5. Auxiliary Classifiers Classifier in Middle

Modern CNN

2015 **Inception-v3** **by Google Research Team**

Batch Normalization normalizes the activations within each **mini-batch**

$z=Wx+b \quad BN(z) \rightarrow \hat{z} \quad a = ReLU(\hat{z})$

In deep networks (especially CNNs and RNNs), some parameters change very fast, others very slowly.

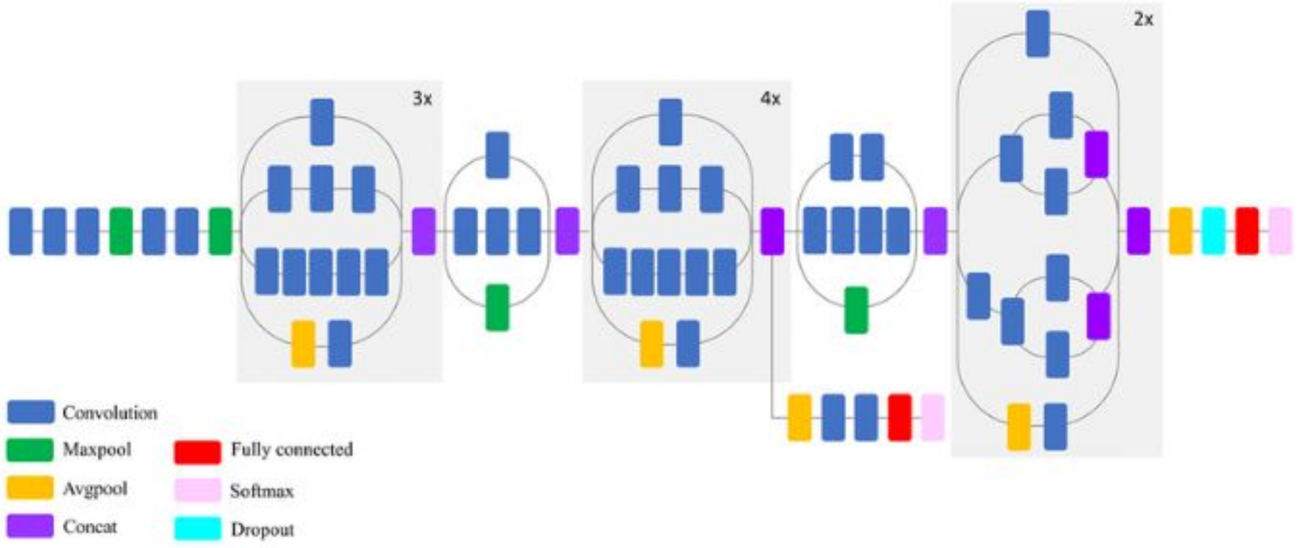
So we need adaptive learning rates per parameter.

RMSProp (Root Mean Square Propagation) — proposed by Geoff Hinton (2012) — solves this by:

Scaling the learning rate individually for each parameter. Based on how large or small its recent gradients are

Modern CNN

2015 **Inception-v3** by Google Research Team



Modern CNN

2016 **Xception-v3** **by François Chollet** **Paper**

💡 "Xception" = Extreme Inception

Depthwise Separable Convolution = Depth + PointWise Conv

mixes both Spatial and Channel Information at once…

Modern CNN

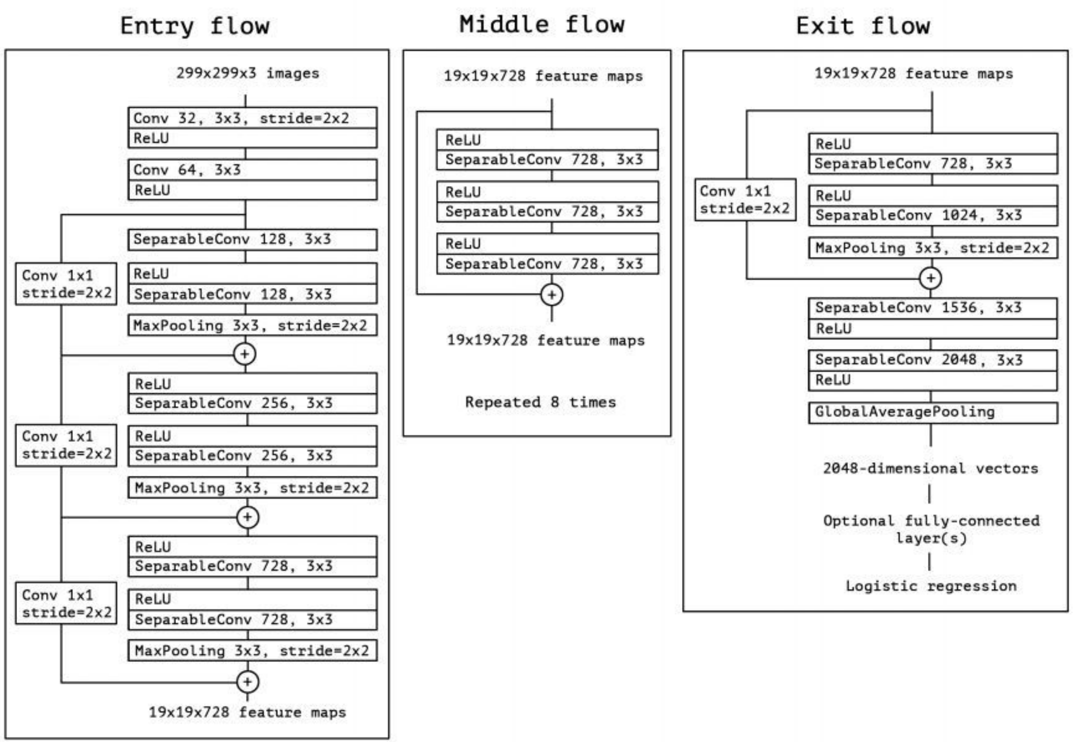
2016 **Xception-v3** by François Chollet

Depthwise Conv → BatchNorm → ReLU → Pointwise Conv → BatchNorm → ReLU

Input	Kernel	Channels	Parameters
Standard Conv	3×3	32→64	3×3×32×64 = **18,432**
Depthwise Separable	3×3 (depthwise) + 1×1 (pointwise)	32→64	(3×3×32) + (1×1×32×64) = **288 + 2048 = 2336**

Modern CNN

2016 **Xception-v3** by François Chollet



Modern CNN

2017 **MobileNet** at Google (Howard et al) [Paper]

Designed for Mobile, Embedded Devices

Depthwise Separable Convolutions (just like Xception)

h-swish(x) = $x \cdot \mathrm{ReLU6}(x+3)/6$ **ReLU** 6 = min(max(0,x),6) Avoid Sigmoid

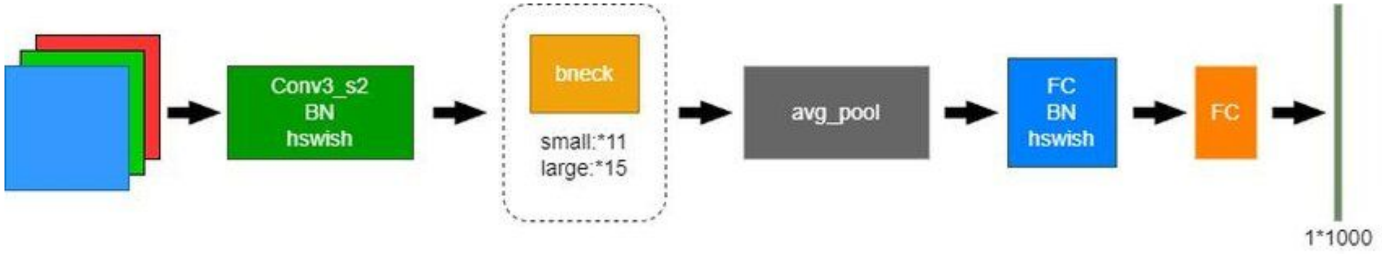
Model Size Control:

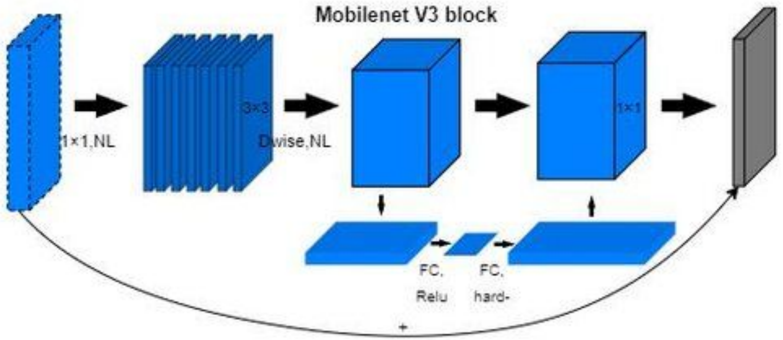
α (Width) Smaller α → fewer filters → fewer weights → faster & smaller model.

ρ (rho) scales the input image size or Resolution of Image

Modern CNN

2017 **MobileNet** at Google (Howard et al)





Muhammed Hassan Mukhtar

Modern CNN

2017 **SENet** by Hu et al [Paper]

Squeeze-and-Excitation Networks (won ImageNet 2017)

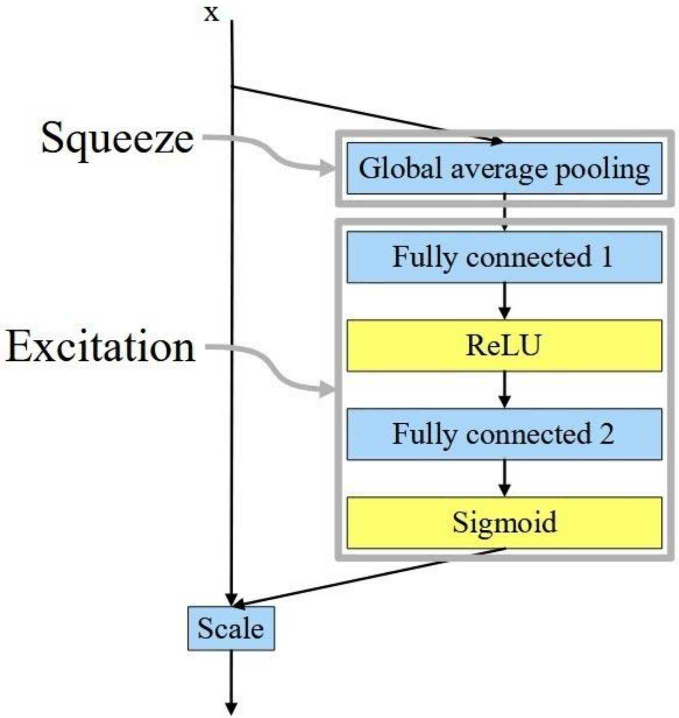
Global Avg Pool (Squeeze) → FC → ReLU → FC → Sigmoid (Excitation)

Squeeze (Global Information)

Excitation (Learn Channel Weights)

Modern CNN

2017 **SENet** by Hu et al



Modern CNN

2017 **DenseNet** by Gao Huang et al [Paper]

Like Dense Layer here each Conv Layer is Connected to all others passing Info

Growth Rate (k) How many new Feature Maps each Layer adds to the Total

A transition layer is a bridge between two dense blocks.Its job is to:

- **Reduce** Number of Feature Maps → 1×1 Convolution (Compression)
- **Downsample** the spatial resolution → 2×2 Average Pooling (Downsampling)

Modern CNN

2017 **DenseNet** by Gao Huang et al

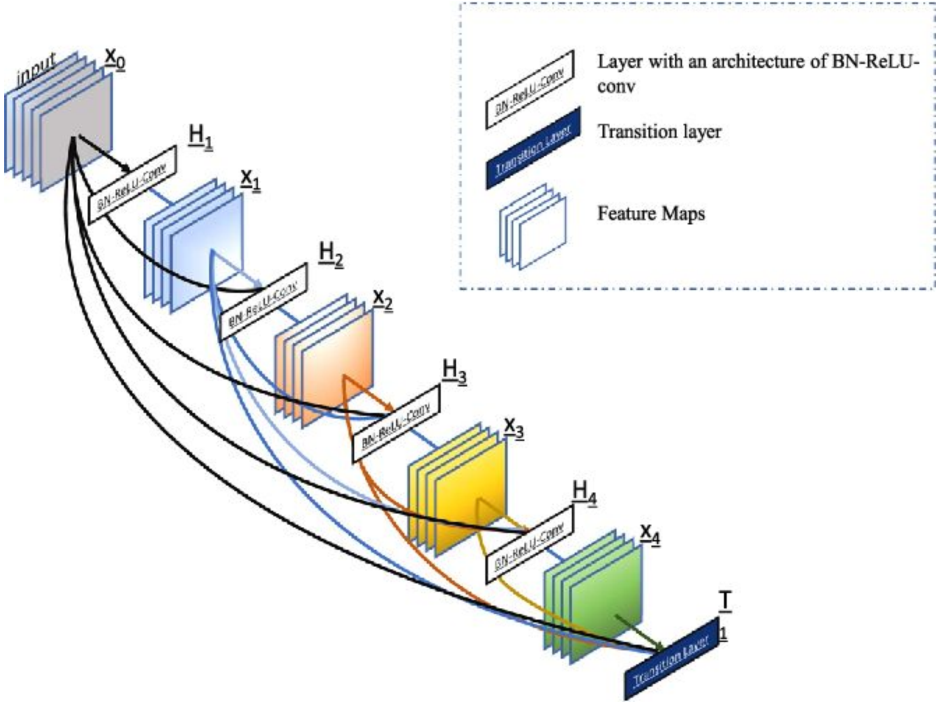
If each layer adds k = 32 channels,

and you have L = 3 layers:

Layer	Input Channels	Output Channels	Total Channels
0	64	+32	96
1	96	+32	128
2	128	+32	160

Modern CNN

2017 **DenseNet** by Gao Huang et al



Modern CNN

2019 **EfficientNet** by Google AI [Paper]

Compound Scaling Depth scaling, Width scaling, Resolution scaling

depth: $d=\alpha\phi$, width: $w=\beta\phi$, resolution: $r=\gamma\phi$

Depth scaling → Adding more Layers

Width Scaling → Adding more Filters

Resolution Scaling → Use Large Shape Input Images

Modern CNN

2019 **EfficientNet** by Google AI

Compound Scaling Depth scaling, Width scaling, Resolution scaling

depth: d=αϕ, width: w=βϕ, resolution: r=γϕ

a. b^2.y^2 Approx Eql 2

Where Each time you increase ϕ by 1 → model ~2× more expensive

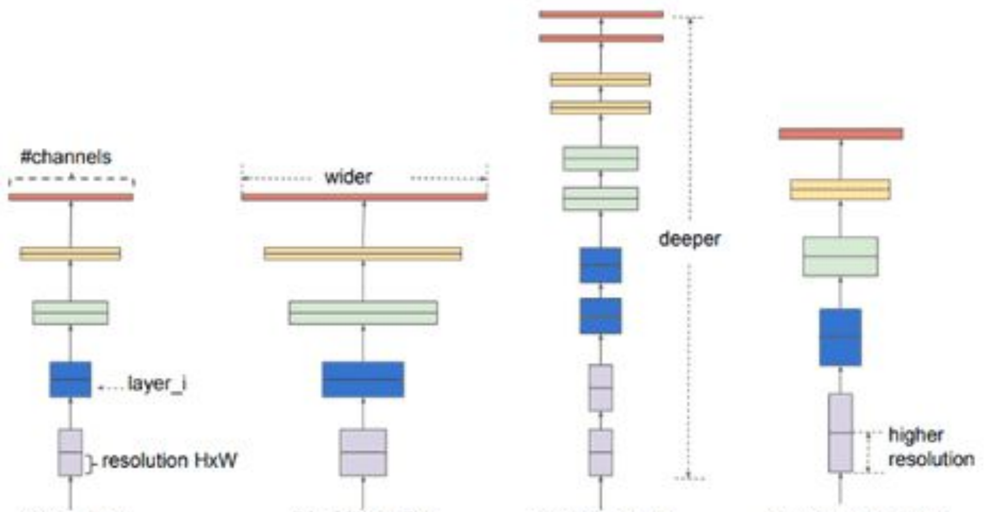
Modern CNN

2019 **EfficientNet** by Google AI

Model	φ	Resolution	Depth ↑	Width ↑
B0	0	224×224	1.0×	1.0×
B1	1	240×240	1.1×	1.0×
B2	2	260×260	1.2×	1.1×
B3	3	300×300	1.4×	1.2×
B4	4	380×380	1.8×	1.4×
B5	5	456×456	2.2×	1.6×
B6	6	528×528	2.6×	1.8×
B7	7	600×600	3.1×	2.0×

Modern CNN

2019 **EfficientNet** by Google AI



(a) baseline (b) width scaling (c) depth scaling (d) resolution scaling

Concept Sharing NLP and CV

Bag of Words and **Bag of Visual Words**

Step 1: Feature Extraction   SIFT,SURF, ORB, or HOG

Step 2: Build a Visual Vocabulary (Codebook)

Collect all local descriptors from the dataset

Cluster them using k-means clustering into K clusters (e.g., K=500)

Each cluster center represents a visual word

Concept Sharing NLP and CV

Bag of Words and **Bag of Visual Words**

Step 4: Encodes images into BoVW histograms

Step 5: Trains a Simple Classifier

Modern Successors

- Spatial Pyramid Matching (SPM) → adds spatial information
- Fisher Vectors → uses probabilistic feature encoding
- VLAD (Vector of Locally Aggregated Descriptors) → compact representation

Concept Sharing NLP and CV

2020-2022 **Visionary Transformer** by Google Research [Paper]

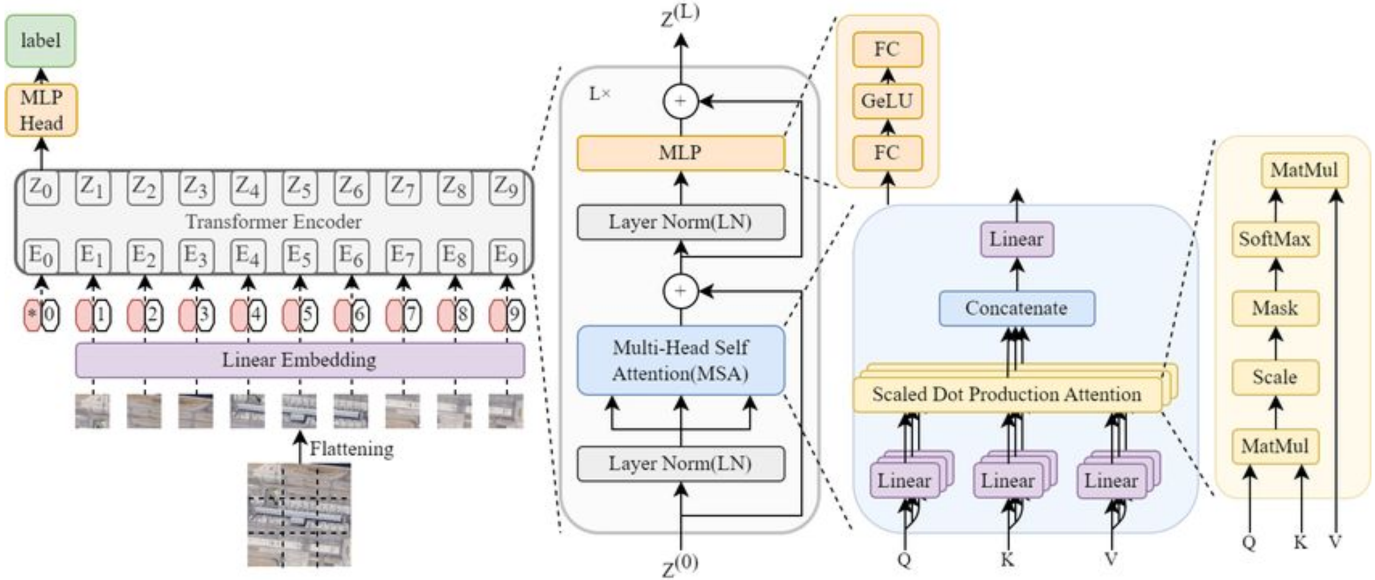
Treat an image as a sequence of patches — just like words in a sentence.

Flattening, Patching in Sequence, Feeding to Transformer

A classification token [CLS] is prepended to the patch sequence.

Concept Sharing NLP and CV

2020-2022 **Visionary Transformer** by Google Research



(a) (b) (c) (d)

Modern CNN

2022 **ConvNeXt** **by Facebook AI Research** **[Paper]**

ConvNeXt is a pure CNN, but redesigned to mimic Transformer Architecture Choices

Large Patch Sizes
Fewer Stages like RESNET
Layer Normalization
GELU Activations
Inverted Bottlenecks expands → processes → compresses
Better Training Techniques

Modern CNN

2022 **ConvNeXt** **by Facebook AI Research**

Fewer Stages

Fewer downsampled (Better Spatial Fidelity)

Wider feature maps (More Expressive)

Inverted Bottlenecks (ConvNeXt Block)

expands → processes → compresses

Depthwise 7×7 conv → LayerNorm → 1×1 conv (expand 4×) → GELU activation → 1×1 conv

Modern CNN

2022 **ConvNeXt** **by Facebook AI Research**

Better Training Techniques

SGD + Momentum - **AdamW** Handles adaptive learning rates, better convergence, and adds decoupled weight decay (for regularization).

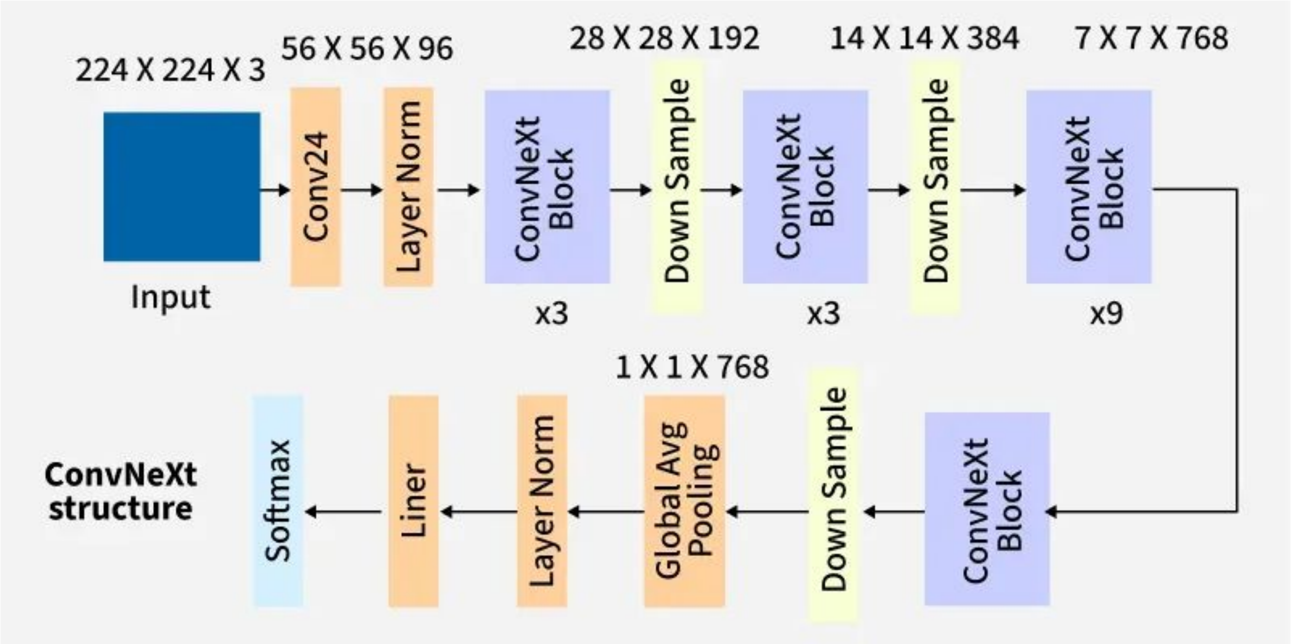
cosine LR Smoothly decays learning rate instead of abrupt drops

Layer-wise LR decay Lower layers train slower, higher layers faster mimics ViT

Dropout

Modern CNN

2022 **ConvNeXt** by Facebook AI Research



Modern NLP

Problems: Before LLMs

Poor Semantic Understanding

No Words Relation RNN, LSTM , GRU, N-Grams

Manual Feature Engineering Word Embedding

Poor Handling of Long Dependencies

Difficulty Understanding Ambiguity & Nuance **Sarcastic**

Modern NLP

Generative AI refers to AI systems that can create new content — text, images, audio, code, or even video — *based on the data they were trained on*.

Problems: After LLMs

Limited Training Data No Recent Event Info

No memory or Context across Long Tasks

Can't call external tools or APIs

Hallucinations **(Solved by Real Time Data Access Through DataBases)**

Modern NLP

LangChain an open-source framework for building applications powered by LLMs

It helps developers connect LLMs (like GPT or Claude) to:

External Data (databases, PDFs, APIs, etc.)

Tools (search engines, calculators, code interpreters)

Memory (so the model can "remember" previous context)

A “middleware” that connects an LLM to the real world.

Modern NLP

RAG (Retrieval-Augmented Generation) combines information retrieval with text generation.

Why it's useful:

- Easier debugging and visualization
- Each node represents a function (e.g., document loader, retriever, LLM call)
- Ideal for complex RAG systems, multi-agent setups, or pipelines

Modern NLP

RAG (Retrieval-Augmented Generation) combines information retrieval with text generation.

How it works:

User asks a question

The system retrieves relevant documents (from a database, PDFs, etc.) using embeddings

The LLM reads those documents and generates an answer using the context

Pipeline summary:

Query → **Retrieve Relevant Data** → Feed to LLM → Generate Answer

Modern NLP

LangGraph extends LangChain to create complex, stateful workflows modeled as graphs. It’s ideal for applications requiring dynamic decision-making, multi-step processes, or agentic behavior.

Nodes: Individual tasks (e.g., call LLM, fetch data).

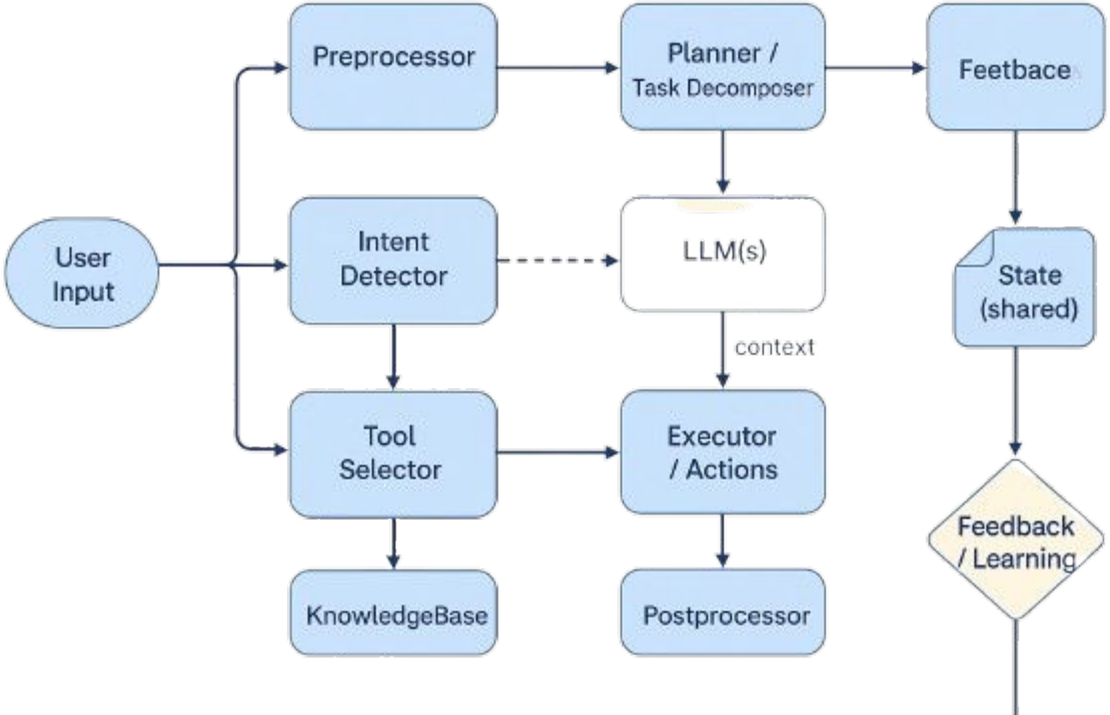
Edges: Transitions between nodes (can be conditional Prob).

State: A shared data structure to track progress and context.

Agents: Systems that dynamically choose actions based on input.

Modern NLP

LangGraph



Modern NLP

Vector Databases A vector database stores and retrieves embeddings — numerical representations of text, images, or other data.

Vector databases make it easy to:

Search for semantic similarity (find content “like” this text).

Store, index, and retrieve large sets of embeddings efficiently.

RAG systems: When the user asks a

question → embeddings are generated → compared with stored vectors

→ top similar docs are retrieved → sent to the LLM

Modern NLP

Popular Vector DBs:

Pinecone

FAISS (by Facebook)

Chroma

If whole WWW is source of Information *everything*

Retriever	Google Search Engine
LLM	GPT / Deep Seek Reasoning Model

Code Walk Through CV

MAE (Masked Autoencoders Work) pretraining, the model learns to predict missing patches using a Vision Transformer encoder–decoder

…. Load Model ….

processor = **ViTImageProcessor**.from_pretrained("facebook/vit-mae-base")

model = **ViTMAEForPreTraining**.from_pretrained("facebook/vit-mae-base")

model.eval();

….

Code Walk Through CV

MAE (Masked Autoencoders Work) pretraining, the model learns to predict missing patches using a Vision Transformer encoder–decoder

…. Forward Pass Through Model ….

```
inputs = processor(images=image, return_tensors="pt")

with torch.no_grad():

outputs = model(**inputs)

….
```

Code Walk Through CV

MAE (Masked Autoencoders Work) pretraining, the model learns to predict missing patches using a Vision Transformer encoder–decoder

…. Structuring Model Output ….

reconstruction = model.**unpatchify**(outputs.logits).detach().squeeze()

reconstruction = reconstruction.**permute**(1, 2, 0)

reconstruction = torch.**clamp**(reconstruction, 0, 1)

….

Code Walk Through CV

Playing with ConvNeXt v2 Nano   **Modern CNN with VIT Concepts**

…. SetUp and Splitting Images to Folder Train/Val/Test ….

device = torch.**device**('cuda' if torch.cuda.is_available() else 'cpu')

BASE = "/content/content/cropped_aug_final"

OUT = "/content/candy_split"

SPLITS = {"**train**": 0.70, "**val**": 0.20, "**test**": 0.10}

….

Code Walk Through CV

Playing with ConvNeXt v2 Nano

…. Data Augmentation ….

```
  transforms.Resize((IMG_SIZE, IMG_SIZE)),
   transforms.RandomHorizontalFlip(),
   transforms.RandomRotation(5),
   transforms.RandomAffine(degrees=0, translate=(0.05, 0.05)),
   transforms.ToTensor(),
   transforms.Normalize(mean=[0.485, 0.456, 0.406],
                        std=[0.229, 0.224, 0.225])
```

….

Code Walk Through CV

Playing with ConvNeXt v2 Nano

…. Modeling ….

```
model = timm.create_model(
    'convnextv2_nano.fcmae_ft_in22k_in1k',  # ConvNeXt V2 Nano pretrained on ImageNet-22k
    pretrained=True,
    num_classes=num_classes
)
model = model.to(device)
….
```

Code Walk Through CV

Playing with ConvNeXt v2 Nano

…. Modeling ….

NUM_EPOCHS = 10

LEARNING_RATE = 1e-3

criterion = nn.**CrossEntropyLoss**()

optimizer = optim.**AdamW**(model.parameters(), lr=LEARNING_RATE)

scheduler = optim.**lr_scheduler**.**ReduceLROnPlateau**(optimizer, mode='max', factor=0.5, patience=2)

….

Code Walk Through CV

Playing with ConvNeXt v2 Nano

…. Training ….

FP optimizer.**zero_grad**() — outputs = **model**(images) — loss = **criterion**(outputs, labels)

BP loss.**backward**() — optimizer.**step**()

Statistics

train_loss += loss.item() * images.size(0)

_, predicted = torch.max(outputs.data, 1)

train_total += labels.size(0)

train_correct += (predicted == labels).sum().item()

….

Code Walk Through CV

Playing with ConvNeXt v2 Nano

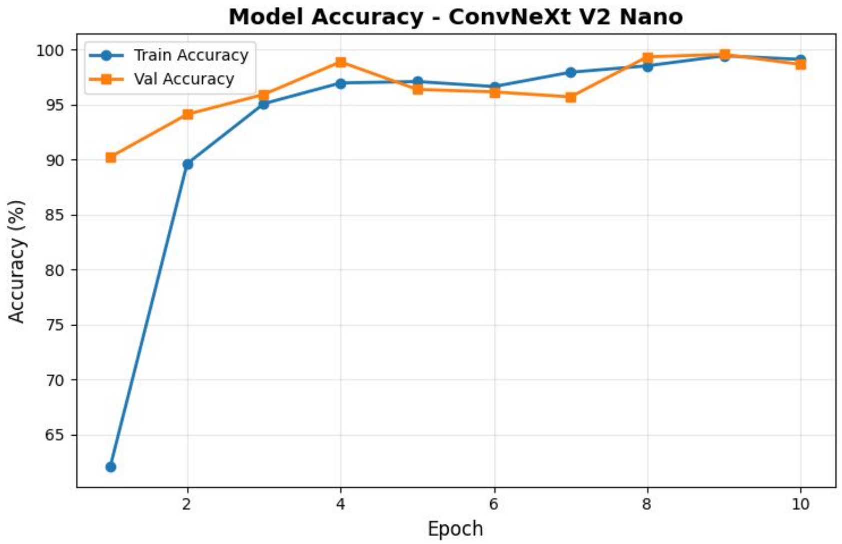
…. Save Model ….

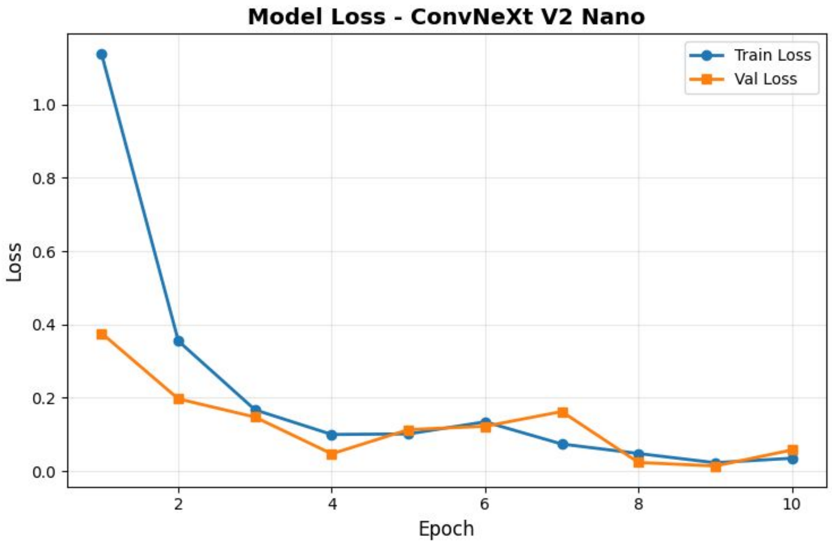
```
final_model_path = os.path.join(base_dir, 'convnext_v2_nano_final.pth')
torch.save({
    'model_state_dict': model.state_dict(),
    'class_names': class_names,
    'num_classes': num_classes
}, final_model_path)
….
```

Code Walk Through CV

Playing with ConvNeXt v2 Nano

…. Evaluation ….





Code Walk Through CV

Playing with ConvNeXt v2 Nano

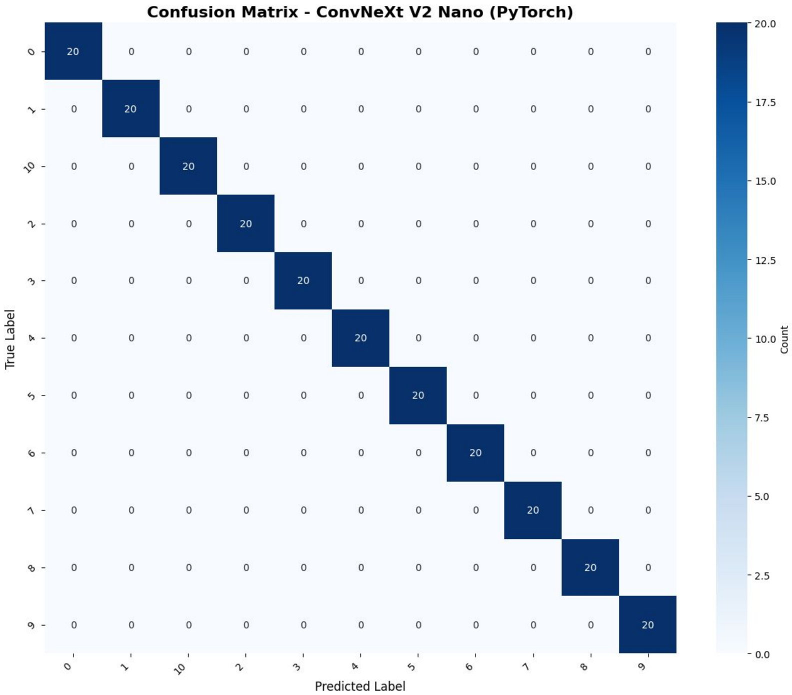
…. Testing ….

```
with torch.no_grad():
    for images, labels in tqdm(test_loader, desc="Predicting"):
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        probs = torch.softmax(outputs, dim=1)
        _, predicted = torch.max(outputs, 1)
….
```

Code Walk Through CV

Playing with ConvNeXt v2 Nano

…. Confusion Matrix ….



Muhammed Hassan Mukhtar

Code Walk Through CV

Playing with ConvNeXt v2 Nano

…. Conclusion ….

With limited Data

Less Training

More Augmentation

Model performed Very Well

….

Code Walk Through CV

Playing with MSCViT Tiny Hybrid CNN + Vision Transformer [Paper]

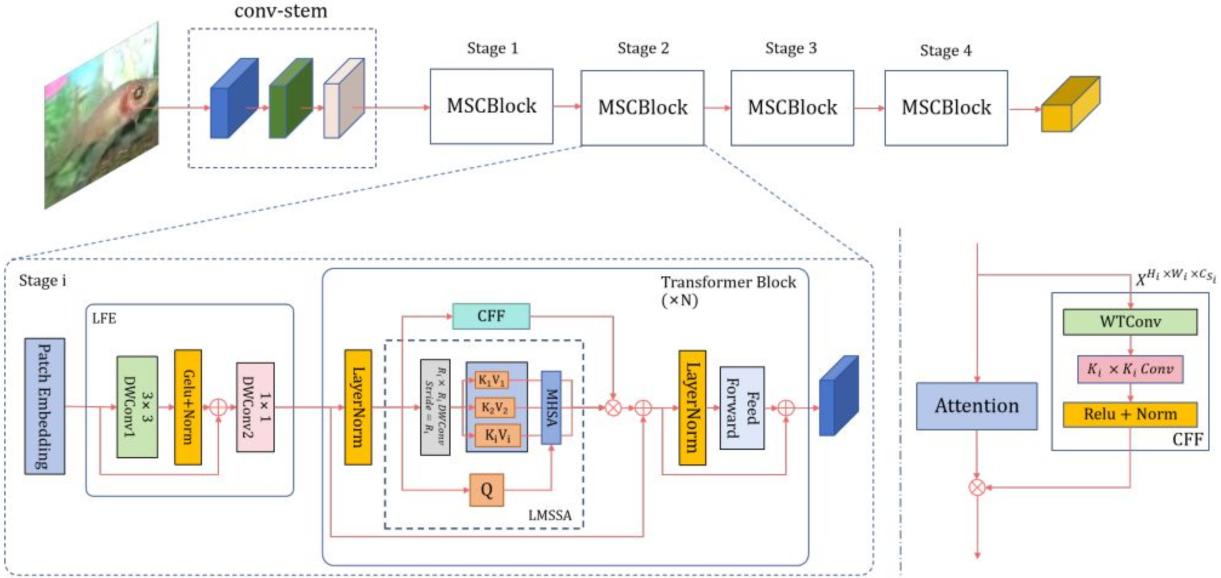


Figure 2: The overall architecture of the proposed MSCViT.

Code Walk Through CV

Playing with MSCViT Tiny **Hybrid CNN + Vision Transformer**

…. SetUp , Data Augmentation, **Modeling** ….

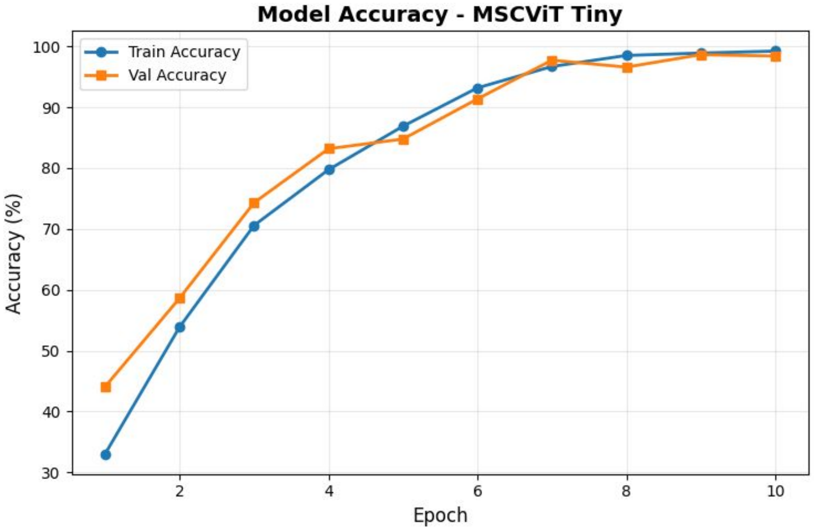
```
model_name = 'convit_tiny'
model = timm.create_model(
 model_name,
 pretrained=True,
 num_classes=num_classes
) ….
```

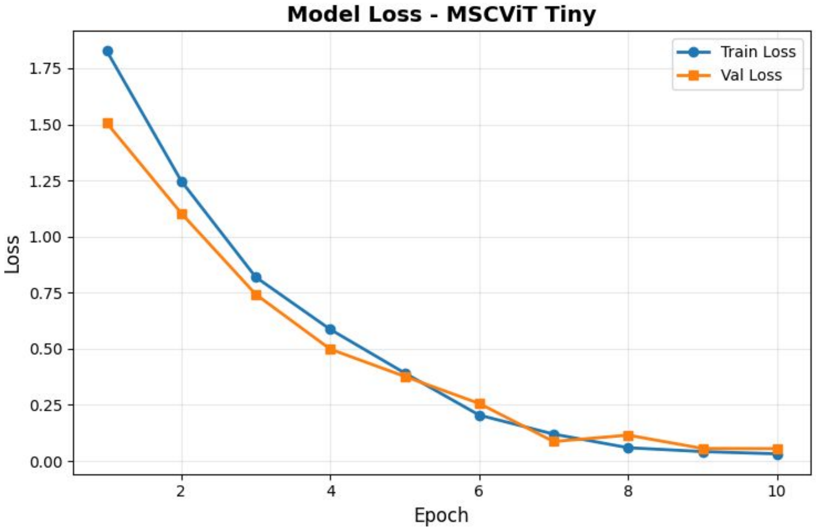
Code Walk Through CV

Playing with MSCViT Tiny

…. Same Training Para, ….

.



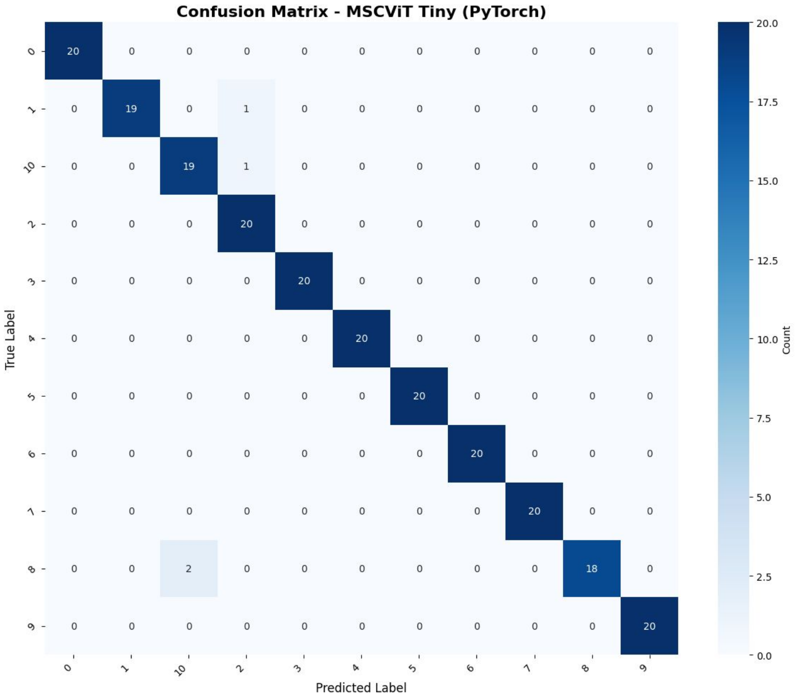


University of Salford MANCHESTER

Code Walk Through CV

Playing with MSCViT Tiny **Hybrid CNN + Vision Transformer**

…. Same Test Data ….



Muhammed Hassan Mukhtar

Code Walk Through NLP

…. Ollama CLI ….

!curl -s https://ollama.com/install.sh | **bash**

ollama **serve** **Run in Terminal**

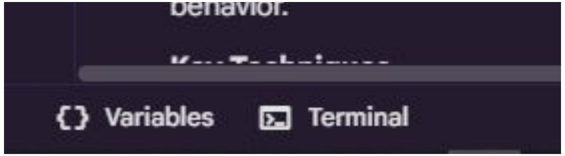
!ollama pull **llama3.2:1b**

….

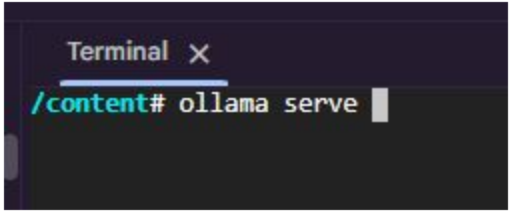
Code Walk Through NLP

…. Colab Terminal ….

In Bottom Left



In Top Right



The need to be running in background on top you will run LLM

Code Walk Through NLP

Prompt Engineering

```
….
def query_ollama(prompt):
    response = ollama.chat(model='llama3.2:1b',
        messages=[{'role': 'user', 'content': prompt}])
    return response['message']['content']
….
```

Code Walk Through NLP

Prompt Engineering

Zero-Shot Prompting: Ask the model to perform a task without examples.

…. prompt_zero_shot = """

Translate the following English sentence into French:

"The quick brown fox jumps over the lazy dog."

"""

output_zero_shot = **query_ollama**(prompt_zero_shot) ….

Code Walk Through NLP

Prompt Engineering

Few-Shot Prompting: Provide examples to guide the model.

…. **prompt_few_shot** = """

You are translating English to French. Here are some examples:

English: Hello, how are you? French: Bonjour, comment ça va?

English: What is your name? French: Comment tu t'appelles?

Now translate the following: English: I like learning languages.

""" ….

Code Walk Through NLP

Prompt Engineering

Chain-of-Thought (CoT): Encourage step-by-step reasoning.

…. **prompt_chain_of_thought** = """

Let's solve this step by step.

Question: If a car travels 60 km in 2 hours, what is its average speed in km/h?

Answer (show your reasoning):

"""

output_chain_of_thought = query_ollama(prompt_chain_of_thought) ….

Code Walk Through NLP

Prompt Engineering

Role-Based Prompting: Assign a role (e.g., "Act as a teacher") to shape the response.

…. **prompt_role_based** = """

You are an experienced English teacher.

Explain the difference between 'affect' and 'effect' to a student in simple terms.

"""

output_role_based = query_ollama(prompt_role_based) ….

Provide Context, Role in Your Prompt as String pass to LLM.

Code Walk Through NLP

LangChains **API**

….

llm = **Ollama**(model="llama3.2:1b")

chain = **LLMChain**(llm=llm, prompt=prompt)

response = chain**.run**("I love learning languages.")

….

Code Walk Through NLP

LangChains **Memory (Conversation Context)**

….

memory = **ConversationBufferMemory**(memory_key="chat_history", input_key="question")

chat_chain = **LLMChain**(llm=*llm*, prompt=*prompt_with_memory*, memory=**memory**)

memory.**clear**()

….

Code Walk Through NLP

LangChains **RAG Simulation** RAW Text

```
….
def simple_retrieval(query):
    for d in docs:
        if any(word.lower() in d.page_content.lower() for word in query.split()):
            return d.page_content
    return "No relevant data found."
….
```

Code Walk Through NLP

LangChains **RAG Simulation** **RAW Text**

…. **rag_template** = """Use the context below to answer the question accurately.

Context: {context} Question: {question} Answer:"""

rag_prompt = **PromptTemplate**(

input_variables=["context", "question"],

template=rag_template,

) ….

Code Walk Through NLP

LangChains **RAG Simulation** RAW Text

….

rag_chain = **LLMChain**(llm=llm, prompt=rag_prompt)

rag_response = rag_chain.**run**({"context": context, "question": query})

….

Code Walk Through NLP

LangGraoh **Conversational** **same example as LongChains**

…. Same WrokFlow as in LangChain Except

graph = **StateGraph**(ConversationState)

graph.**add_node**("capture_input", capture_input) graph.**add_node**("generate_response", generate_response)

graph.**add_node**("summarize_conversation", summarize_conversation) graph.**add_edge**("capture_input", "generate_response")

graph.**add_edge**("generate_response", "summarize_conversation") graph.**add_edge**("summarize_conversation", END)

graph.**set_entry_point**("capture_input")

workflow = graph.**compile**()

….

Code Walk Through NLP

Logging Functions

def log_console(message: str):

def log_file(title: str, content: str):

Helper Functions

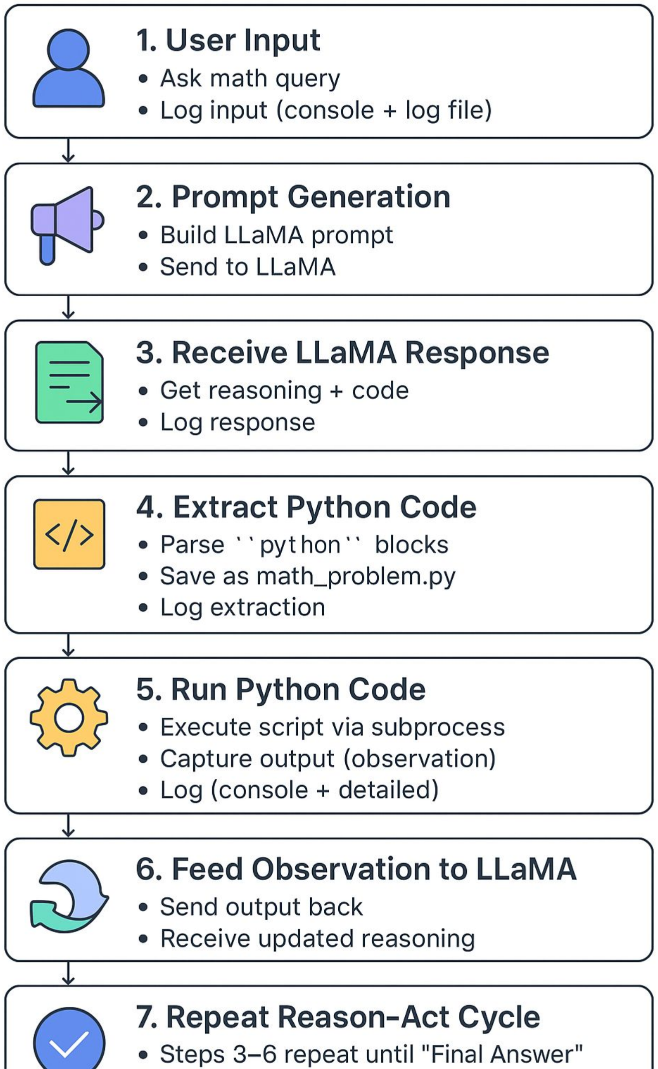
def extract_code(response: str):

def save_code(code: str) -> str:

def run_code(filename: str) -> str:

Main ReAct Agent

def main():



Muhammed Hassan Mukhtar

Code Walk Through NLP

🔢 ReAct Math Agent running. Type 'exit' to quit.

Query: Matrix Multiplication of identity Matrix 3x3 and [1,5,9][3,5,7][4,5,6]

[**Input**] Matrix Multiplication of identity Matrix 3x3 and [1,5,9][3,5,7][4,5,6]

[**Step**] Sent query to LLaMA

[**Step**] LLaMA response received

[**Step**] Created Python file: math_problem.py

[**Step**] **Python file error**: **math_problem.py** -> Traceback (most recent call last):

File "/content/math_problem.py", line 9, in <module>

Code Walk Through NLP

```
C = np.dot(I_3x3, B)
    ^^^^^^^^^^^^^^^^
```

ValueError: shapes (3,3) and (1,3) not aligned: 3 (dim 1) != 1 (dim 0)

[Step] Sent error back to LLaMA to fix code

[Step] LLaMA response received

[Step] Created Python file: math_problem.py

[Step] Ran Python file: math_problem.py -> Output: [[1 5 9] [3 5 7] [4 5 6]]

[Final Answer] [[1 5 9] [3 5 7] [4 5 6]]

Thanks

Today we have learned

How concepts evolve in both domains CV and NLP

How can we Evaluate and Choose which model is best for our Use Case

Some Use Cases of Modern CV and NLP Concepts

LOOK HOW FAR WE COME NOW