Project Name

Alphabet Classification

Teaching-Assistant: Eng/ Ahmed Ashraf

Inception V1 — Advanced Inception Architecture

Reference: Szegedy et al., 2015/2016 (Rethinking

Overview / Core Concept: Inception V1 builds on the original Inception (v1) by factorizing convolutions (e.g., replacing 5×5 convs with two 3×3, or n×n with 1×n followed by n×1), incorporating Batch Normalization, label smoothing, and RMSProp optimization. These improvements achieve higher accuracy without significantly increasing computation.

Forward Pass Flow (High-Level)

Stem: Optimized initial convolution layers reduce spatial dimensions efficiently.

Inception Modules: Multiple branches with factorized convolutions (e.g., 1×n followed by n×1) to approximate larger kernels efficiently.

Auxiliary Classifiers: Often removed or simplified; aggressive regularization and batch normalization are applied throughout.

Classification Head: Global Average Pooling → Dropout → Fully Connected Layer → Softmax.

Key Details / Hyperparameters

Factorized and asymmetric convolutions reduce computation cost (e.g., 1×7 → 7×1).

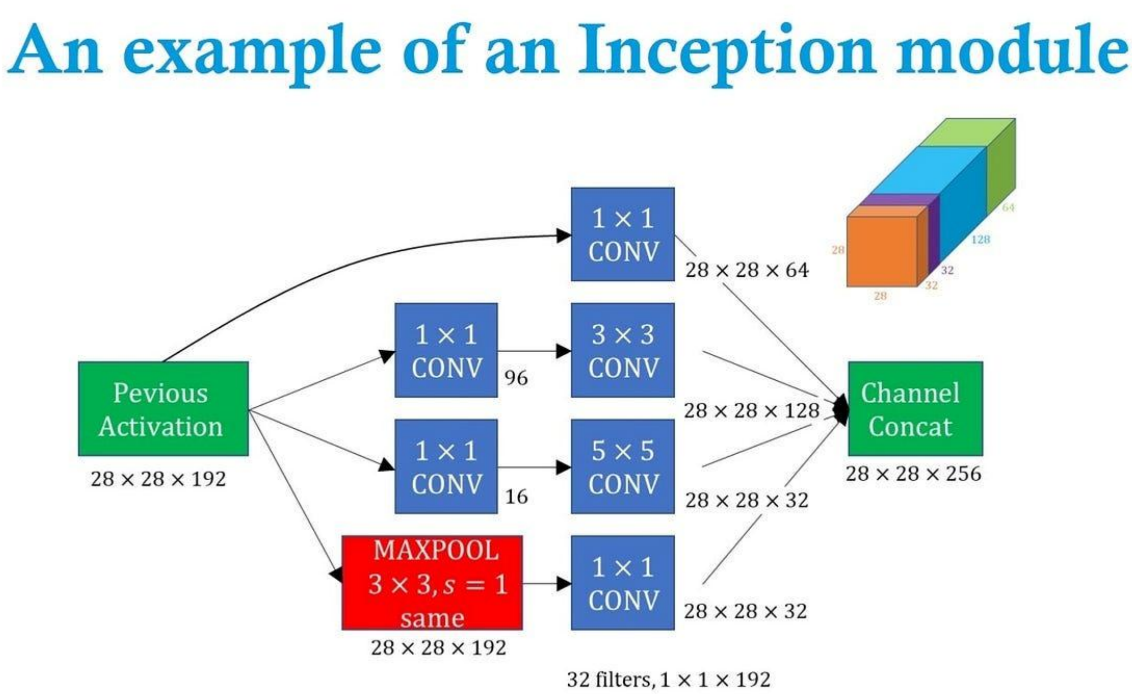
Training strategies such as batch normalization and label smoothing improve convergence.

Inception V1 is widely used as a high-performance backbone for image classification.

Advantages and Limitations

Pros: **Excellent accuracy-to-computation ratio; widely adopted in practice.**

Cons: **More complex architecture to implement and tune than standard residual networks; numerous hyperparameters and module variants make it harder to modify.**



2] VGG19 — Very Deep Convolutional Network

Reference: Simonyan & Zisserman, 2014

Overview / Core Concept :

VGG employs a straightforward and consistent architecture: sequences of 3×3 convolutional layers (activated by ReLU) interspersed with occasional 2×2 max-pooling layers. The network's depth is increased to enhance its feature representation capabilities . The VGG19, model specifically contains 19 layers with learnable weights composed of 16 convolutional layers followed by 3 fully connected layers.

Forward Pass Flow (VGG19)

Input: Image of size 224×224×3 (RGB).

Block 1: Two conv layers (3×3, 64 filters) → max-pool (2×2, stride 2) → output spatial dimensions halved.

Block 2: Two conv layers (3×3, 128 filters) → max-pool → output halved.

Block 3: Four conv layers (3×3, 256 filters) → max-pool.

Block 4: Four conv layers (3×3, 512 filters) → max-pool.

Block 5: Four conv layers (3×3, 512 filters) → max-pool.

Classification Head: Flatten → FC (4096) → ReLU → Dropout → FC (4096) → ReLU → Dropout → FC (1000) → Softmax.

Key Details / Hyperparameters:

All convolution layers use 3×3 kernels, stride 1, and padding 1.

Max-pooling uses 2×2 windows with stride 2.

Fully connected layers are large and account for the majority of the network's parameters.

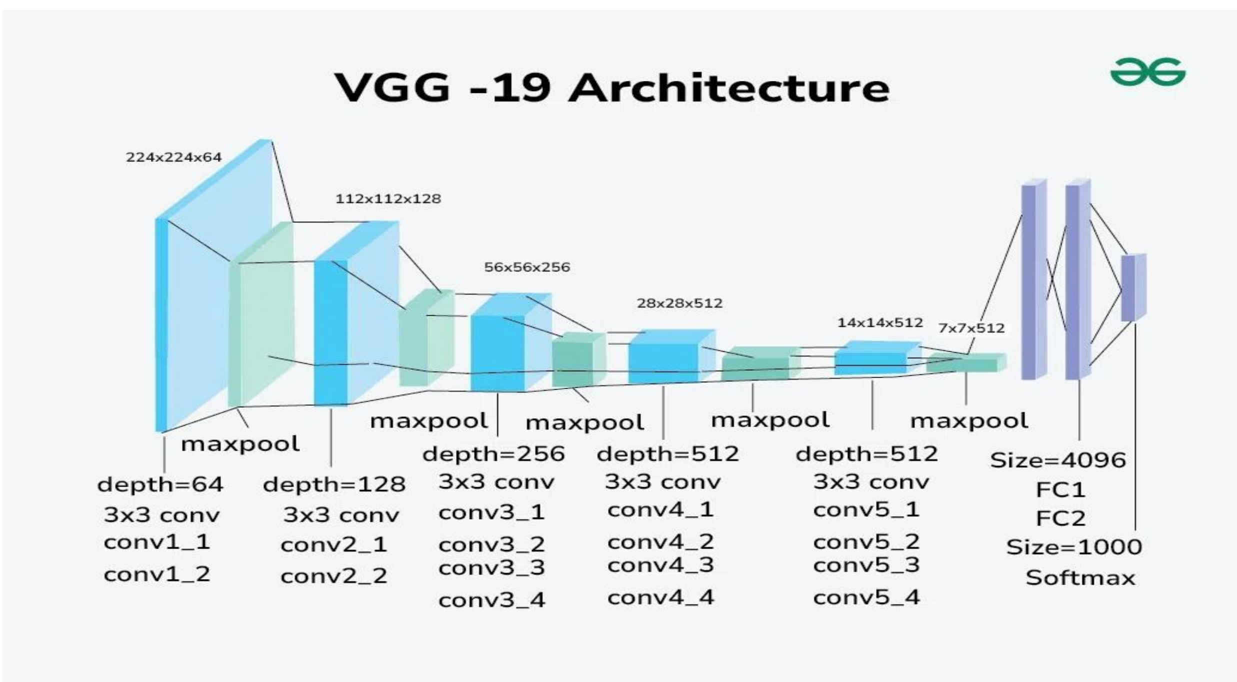
Typical input resolution: 224×224.

Total parameters: ~144 million (depends on exact FC layer configuration).

Advantages and Limitations:

Pros: **Simple and uniform design, strong performance baseline, easy to implement and fine-tune for transfer learning.**

Cons: **Very heavy in terms of parameters and memory; computationally slow; not ideal for mobile deployment or extremely deep scaling**



3] ResNet50 — Deep Residual Network

Reference: He et al., 2015 (Deep Residual Learning)

Overview / Core Concept:

ResNet introduces residual (skip) connections, which learn a residual

mapping $F(x) = H(x\ ) - x$

F(x)=H(x)−x instead of directly learning $H\ (x)$

H(x). This design allows very deep networks (e.g., 50, 101, 152 layers) to be

trained effectively by mitigating the vanishing gradient problem.

Forward Pass Flow (ResNet50)

Input: Image of size 224×224×3.

Initial Layer: Conv (7×7, stride 2) → BatchNorm → ReLU → MaxPool (3×3, stride 2).

Stage 1 (conv2_x): 3 residual bottleneck blocks, each composed of:

1×1 conv (channel reduction) → BatchNorm → ReLU

3×3 conv (spatial) → BatchNorm → ReLU

1×1 conv (channel expansion) → BatchNorm

Add input via identity or projection shortcut → ReLU

Stage 2 (conv3_x): 4 bottleneck blocks.

Stage 3 (conv4_x): 6 bottleneck blocks.

Stage 4 (conv5_x): 3 bottleneck blocks.

Classification Head: Global Average Pooling → Fully Connected Layer → Softmax for class probabilities.

Key Details / Hyperparameters:

Bottleneck structure reduces computation while maintaining network depth.

Typical ResNet50 has ~25 million parameters.

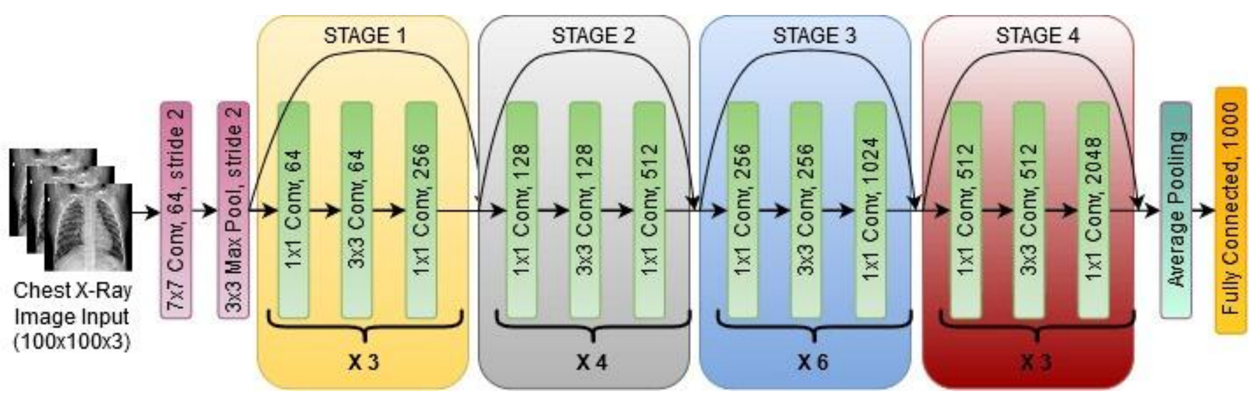
Uses Batch Normalization and He initialization.

Shortcuts: Identity connections are used when input/output dimensions match; projection shortcuts handle dimension changes.

Advantages and Limitations

Pros: Easily trainable for very deep networks, strong accuracy-to-computation ratio, widely adopted as a backbone for many tasks.

Cons: Still relatively heavy compared to lightweight mobile networks; residual connections may require task-specific adjustments (e.g., in detection or segmentation).



4] MobileNet — Efficient Networks with Depthwise Separable Convolutions

Reference: **Howard et al., 2017 (MobileNets)**

Overview / Core Concept:

MobileNet replaces standard convolution layers with depthwise separable convolutions, which split computation into two steps:

Depthwise convolution: performs spatial convolution independently on each input channel.

Pointwise convolution (1×1): combines channels across depth.

This decomposition significantly reduces both FLOPs and the number of parameters. Two main hyperparameters control the model size and accuracy: width multiplier (α) and resolution multiplier (ρ).

Forward Pass Flow (MobileNet v1)

Input: **e.g., 224×224×3 → Initial 3×3 convolution (stride 2).**

Repeated Blocks: **N × (Depthwise 3×3 conv → BatchNorm → ReLU → Pointwise 1×1 conv → BatchNorm → ReLU). Stride can be 1 or 2 depending on the block.**

Output: **Global Average Pooling → Fully Connected Layer → Softmax for classification.**

Hyperparameters:

Width multiplier $\alpha \in (0,1]$

α∈(0,1] scales channel numbers.

Resolution multiplier ρ

ρ adjusts input spatial dimensions.

Key Details / Hyperparameters

MobileNet v1 typically uses ReLU (or ReLU6) and Batch Normalization.

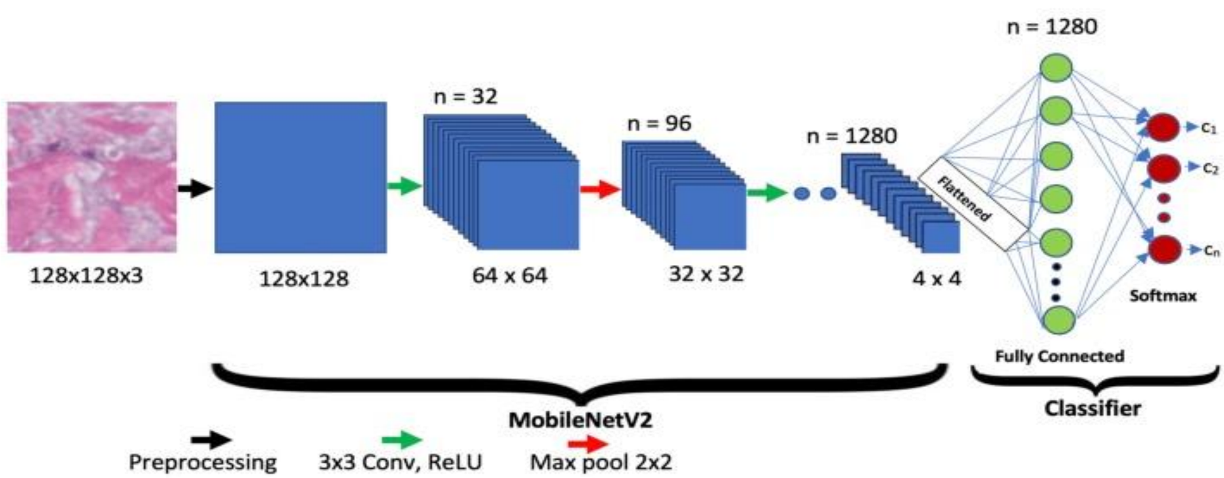
α = 1.0 corresponds to the full model; α < 1.0 produces a smaller, lighter version for mobile applications.

Designed to trade off accuracy for latency and memory efficiency.

Advantages and Limitations

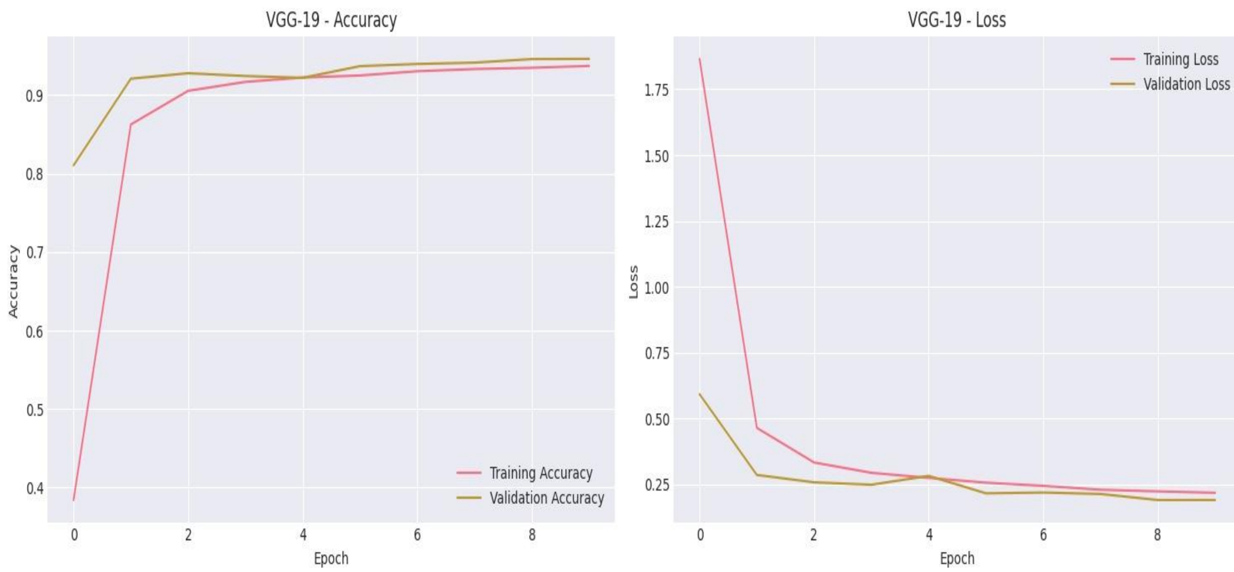
Pros: Extremely lightweight and fast, ideal for mobile and embedded devices.

Cons: Lower accuracy compared to large standard CNNs; later versions (MobileNetV2/V3) improved efficiency using inverted residual blocks and other enhancements

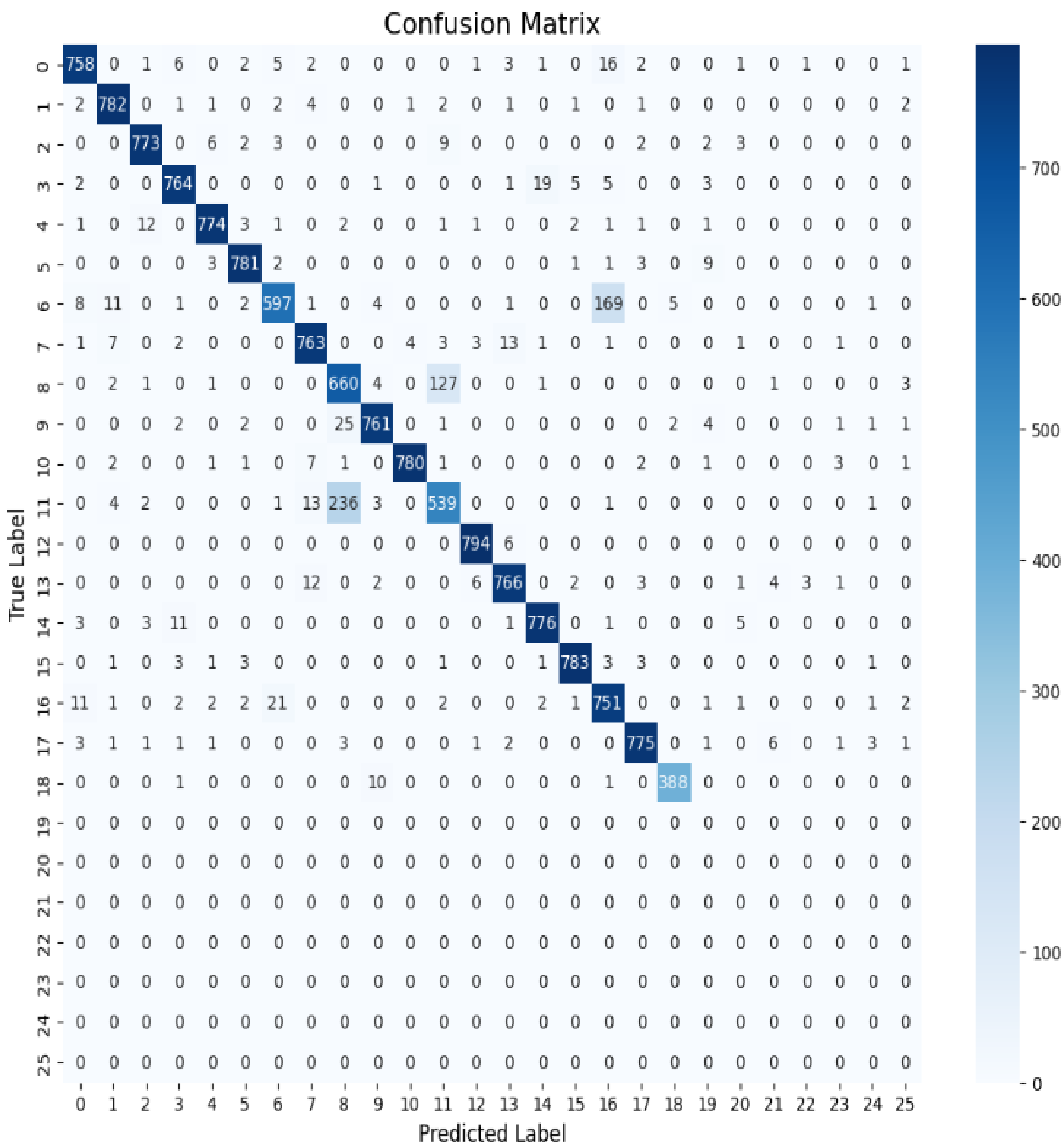


Results for each model:

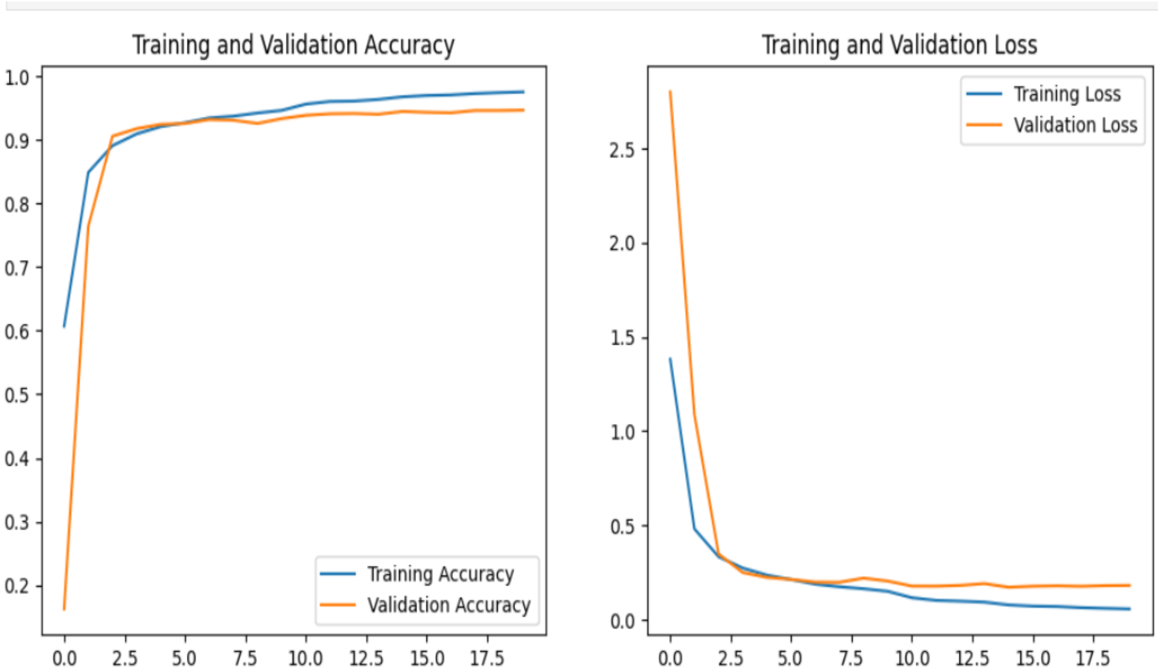
1) VGG19 :



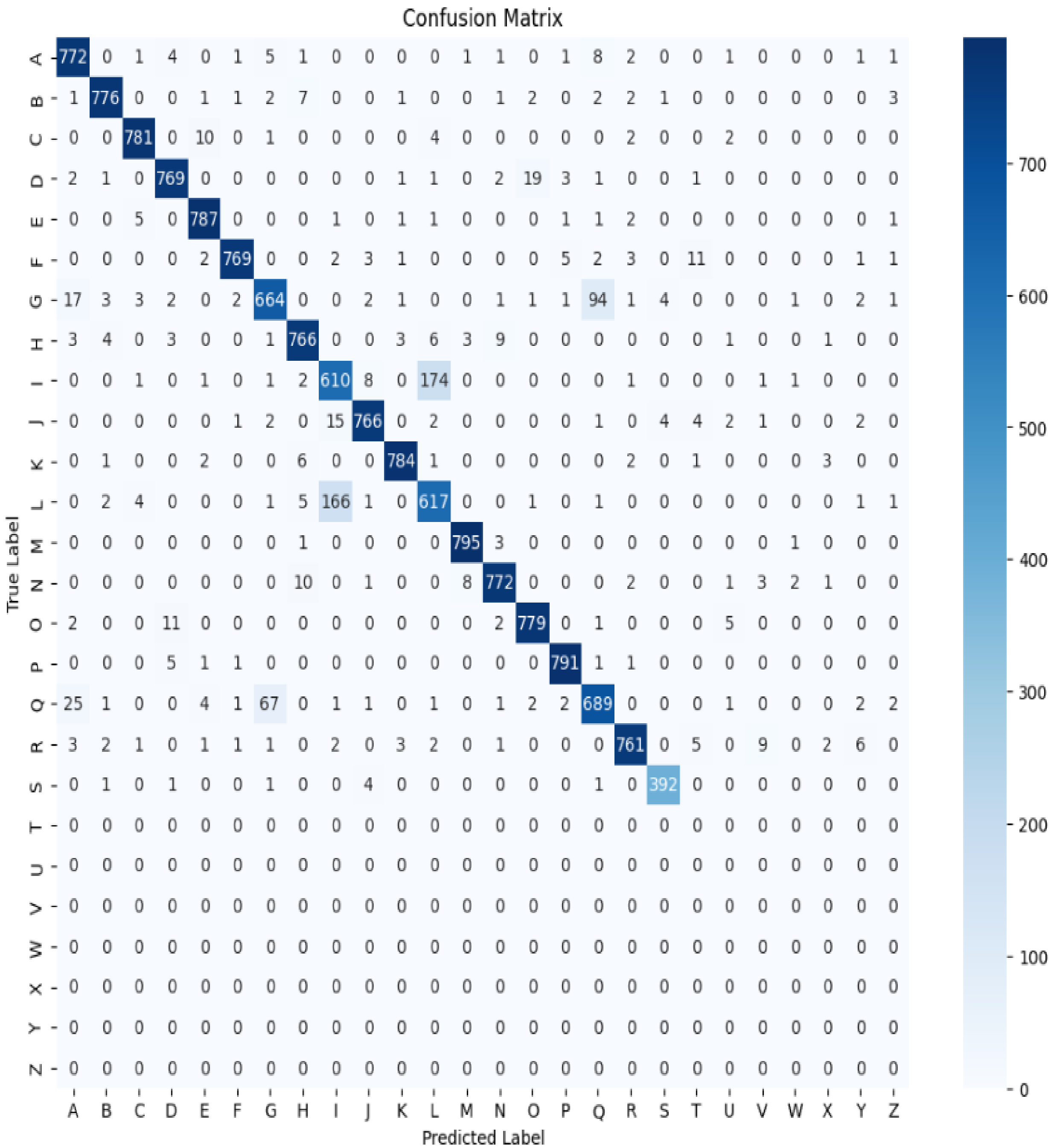
VGG -> accuracy: accuracy: 0.9371 - loss: 0.2178 - val_accuracy: 0.9461 - val_loss: 0.1904 - learning_rate: 1.0000e-04


Confusion Matrix
True Label
Predicted Label

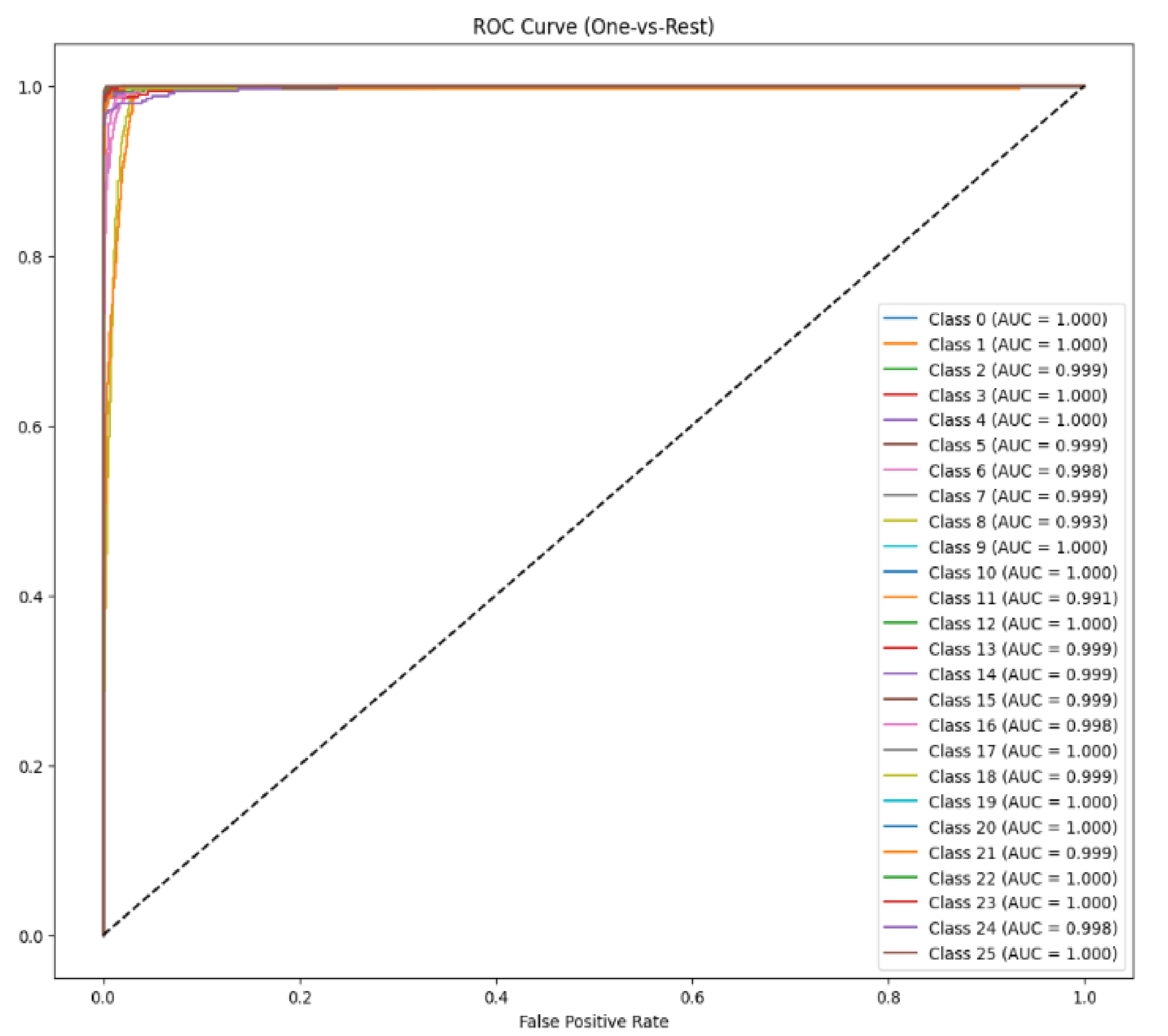
2) MobileNet:



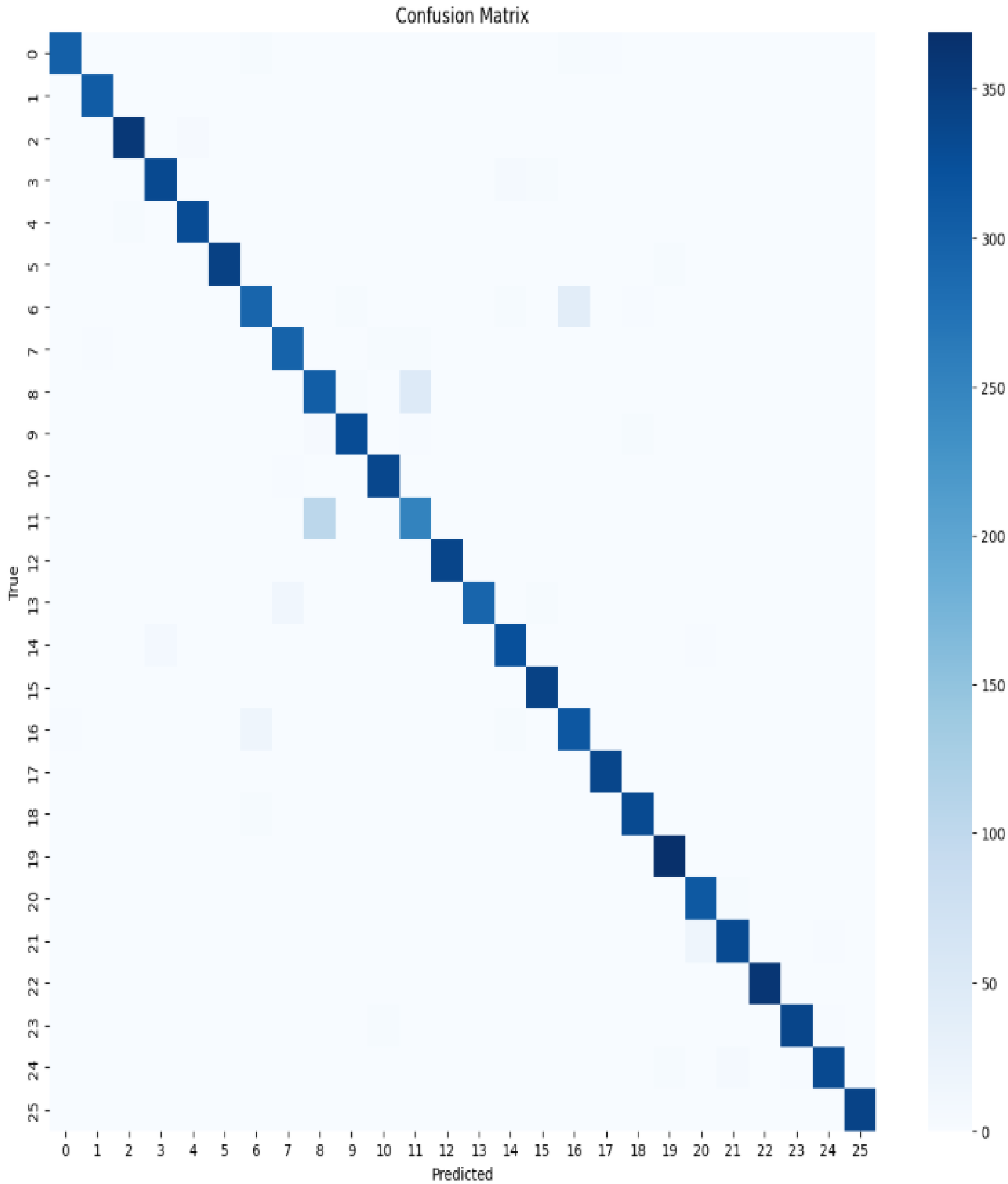
MobileNet：accuracy: 0.9748 - loss: 0.0573 - val_accuracy: 0.9467 – val_loss: 0.1814 - learning_rate: 6.2500e-06


Confusion Matrix
True Label
Predicted Label
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0
100
200
300
400
500
600
700

3) Inception V1:



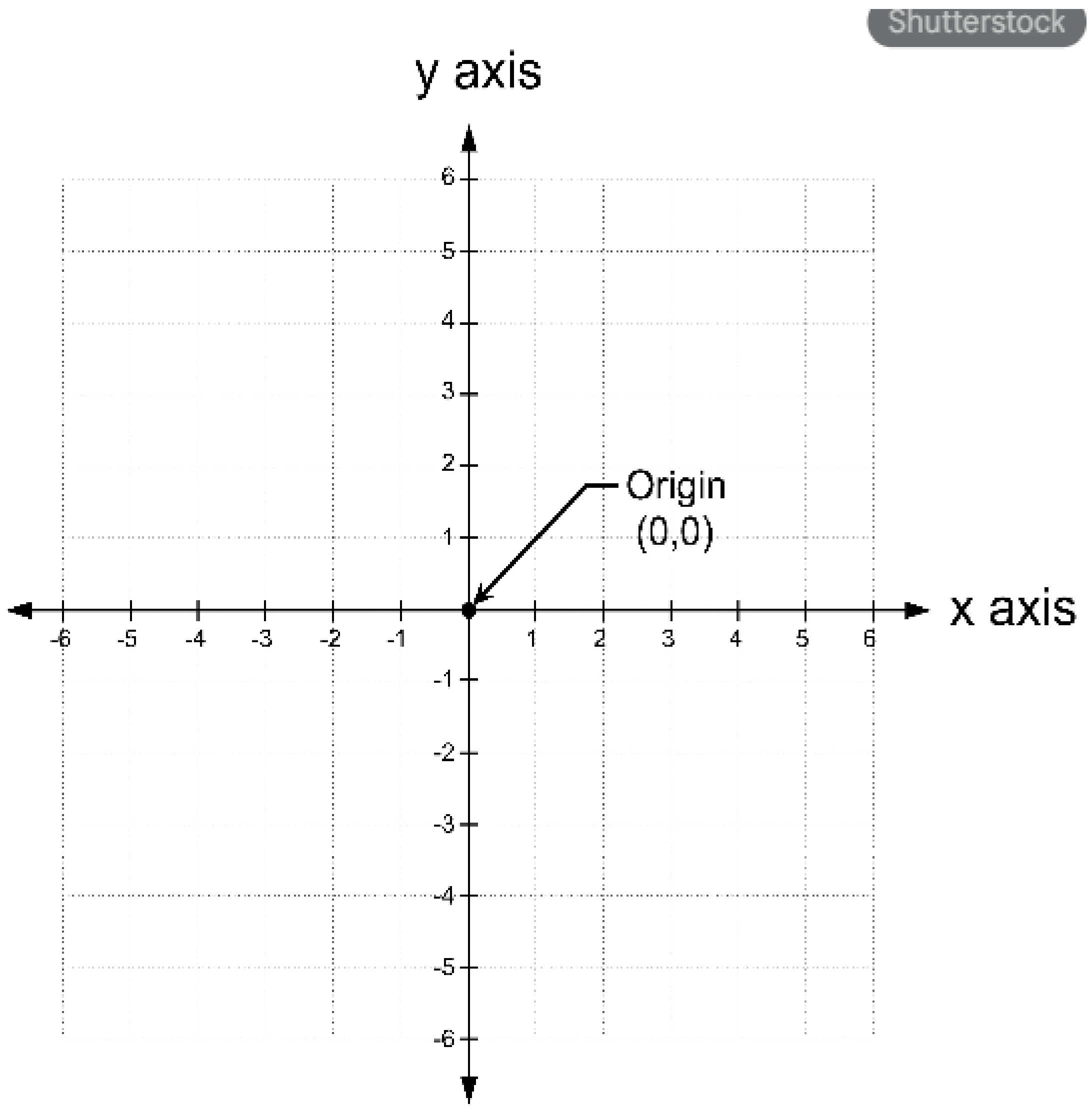
Inception V1: Epoch 10: Train Acc=0.977, Val Acc=0.947


Confusion Matrix
True
Predicted
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
350
300
250
200
150
100
50
0

comparison :

Model Architecture	Training Approach	Test Accuracy	Test Loss	Key Observations
VGG-style CNN (Custom)	Trained from scratch	0.9420 (Val Acc)	0.1652 (Val Loss)	High performance, fast training due to custom, smaller architecture. Excellent generalization ($\text{Overfitting gap} \approx 0.015$).
ResNet50 (Transfer Learning)	Feature Extraction (Frozen Base)	0.8889 (Val Acc)	0.3442 (Val Loss)	Significantly lower performance than VGG/MobileNet. Scaling up to 224x224 and conversion to 3 channels likely introduced artifacts/complexity that the frozen base could not handle optimally for this domain.
MobileNetV2 (Transfer Learning + Fine-Tuning)	Fine-Tuning after initial feature extraction	0.9351	0.2081	Strong performance, very fast inference due to efficient architecture. Fine-tuning provided a slight improvement over initial results.
Vision Transformer (ViT) (Trained from scratch)	Trained from scratch with Data Augmentation	0.8900 (Val Acc, Epoch 12)	0.3124 (Val Loss, Epoch 12)	Started slowly but showed strong learning ability. Performance is comparable to basic transfer learning models, but lower than the custom CNN/MobileNet, possibly due to limited training time and dataset size for a full ViT model.
Inception V1/GoogLeNet (Transfer Learning)	Fine-Tuning (Cut short due to OOM)	(N/A)	(N/A)	Performance could not be fully evaluated due to **CUDA Out of Memory** errors, highlighting its high memory requirement on this hardware configuration.

Pros and Cons of Each Architecture:



Pros and Cons of Model Architectures:

Architecture	Pros (+)	Cons (-)
VGG-style CNN (Custom)	+ Very high accuracy on this dataset.	- More parameters than MobileNet (less efficient).
	+ Robust against overfitting (small generalization gap).	- Design is relatively manual and less modular than ResNet.
ResNet50 (Transfer Learning)	+ Utilizes powerful pre-trained features (Transfer Learning).	- Low performance without full fine-tuning.
	+ Skip connections mitigate vanishing gradients.	- Requires input image resizing to 224x224, which alters the small 28x28 EMNIST images.
MobileNetV2 (Transfer Learning)	+ Extremely efficient (low number of parameters).	- Requires input image conversion to 3 channels (RGB) for pre-trained weights.
	+ Good balance between speed and accuracy.	- Slightly lower peak accuracy than the custom VGG-style CNN.
Vision Transformer (ViT) (Scratch)	+ Excellent for capturing global relationships through self-attention.	- Very high data requirement for training from scratch.
	+ State-of-the-art for many vision tasks.	- Slower convergence than CNNs on smaller datasets.
Inception V1 (GoogLeNet)	+ High parameter efficiency using Inception Modules.	- Very high memory footprint; failed due to **OOM error**.
	+ Uses auxiliary classifiers for better gradient flow.	- Complex architecture, sensitive to hyperparameter changes.