ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1799

Secure Deep Feature Classification Framework for Pathology Images Leveraging Blockchain and Cloud Technologies

Prabhat Kumar Shah[1]; Ajeet Kumar Soni[2]; Aryan Parnami[3]; Kushagree Gupta[4]

[1,2,3,4]University Institute of Technology Barkatullah

Co-Guides: Madhav Chaturvedi[1]; Neha Lidoriya[2]

Publication Date: 2025/05/14

Abstract: The boom in smart e-healthcare services pop- ping up in far-flung and spread-out places has sparked some real worries about privacy and speed, especially when it comes to handling touchy patient info. Sharing clinical data across cloud platforms only makes it trickier to keep patient privacy locked down, pushing the need for fresh ideas to get people to trust medical research again. This study tackles those big concerns head-on with a new setup that mixes Deep Learning (DL), Blockchain, and Cloud Computing to whip up a fast and secure system for sorting out pathological images. DL models, with all their fancy complexity, can be a handful in cloud setups where crunching data efficiently often takes a hit [8]. Our framework leans on deep learning tricks to nail down ac- curate classifications of pathological images while keeping sensitive medical data under wraps. Blockchain steps in to build a decentralized, tamper-proof record-keeper, boosting the security and openness of the diagnostic gig. On top of that, cloud computing gets tapped to smooth out the heavy lifting of those tricky DL models, sidestepping the usual headaches tied to old-school cloud methods.

Keywords: *BreaKHis Database, Pixel Size, Ethereum, Blockchain, IPFS, Neural Compression, Computa- Tional Pathology.*

How to Cite: Prabhat Kumar Shah; Ajeet Kumar Soni; Aryan Parnami; Kushagree Gupta; (2025) Secure Deep Feature Classification Framework for Pathology Images Leveraging Blockchain and Cloud Technologies. *International Journal of Innovative Science and Research Technology,* 10(3), 3351-3376. https://doi.org/10.38124/ijisrt/25mar1799

I. INTRODUCTION

The Mixing blockchain tech with convolutional neural networks (CNNs) opens up a cool chance to shake up secure and speedy image classification in decentralized healthcare. For this project, we cooked up a slick framework that ties together blockchain, a web-based image grabber, and a trio of CNN models—VGG16, MobileNet, and ResNet—to nail down some solid image sorting. It all starts with a simple website where folks can upload images. Those pics get shipped off to the Pinata platform, which taps into the InterPlanetary File System (IPFS) [3] to whip up a unique hash ID for each one—keeping the data legit and unchangeable. Then, those hash IDs get locked onto the blockchain through MetaMask transactions, creating a clear, tamper-proof log of where every image came from.

For the classification gig, we rolled with three big CNN players: VGG16, MobileNet, and ResNet [16]. The images get pulled from the cloud using their hash IDs, and each model takes a crack at sorting them out, letting us stack up how they perform. We're digging into which one's the best fit for our dataset, weighing stuff like accuracy, how fast they crunch the numbers, and whether they can scale up.

To keep things smooth between users and the system, we went with a client-server setup. The results zip back to the website's interface, dishing out real-time feedback on what the image's diagnosis looks like. This fresh mashup of blockchain, CNNs, and a user-friendly web front not only locks in data security and transparency but also sets up a real-deal base for decentralized healthcare systems [17]. This work chips in at the crossroads of blockchain and deep learning, tackling the need for safe, efficient ways to classify images in spread-out healthcare setups. We're picturing a future where decentralized healthcare leans on cutting-edge tech to boost diagnostic precision and keep patient data locked down tight [13].

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1799

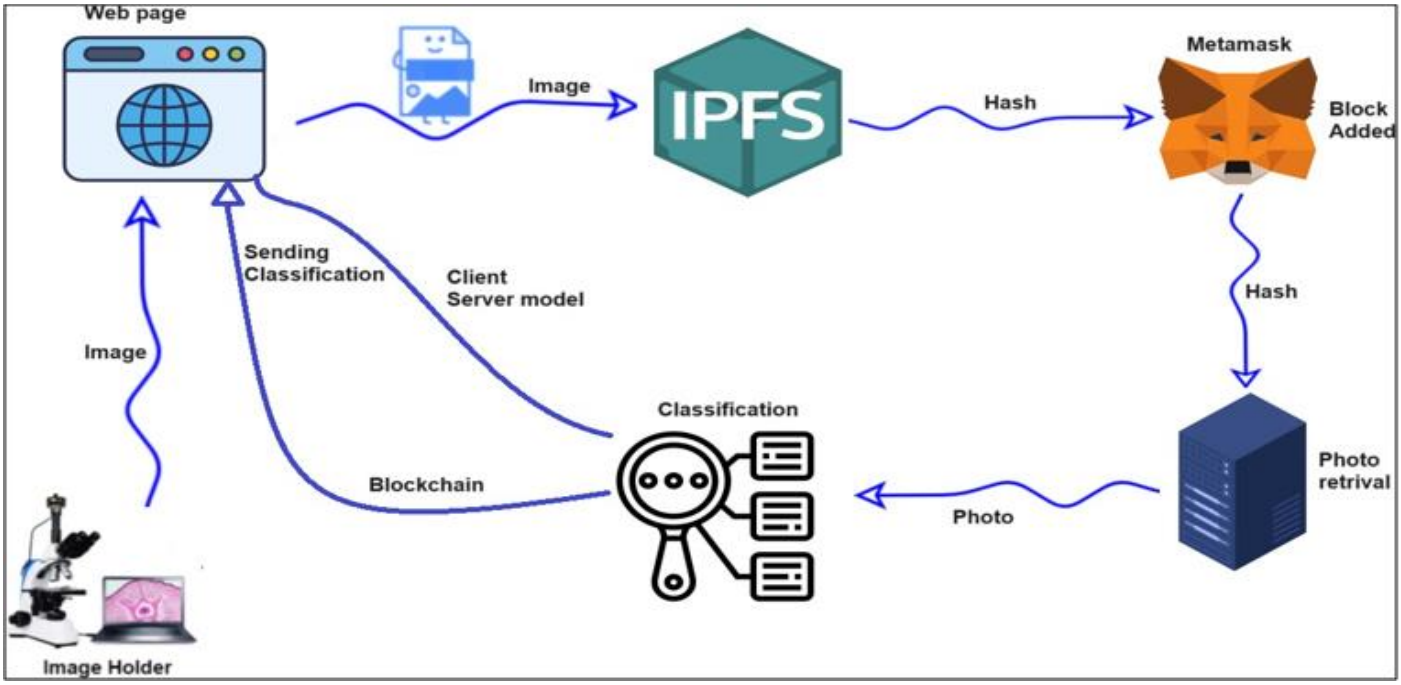

Fig 1 Proposed Framework

Bringing these tech pieces together is all about tackling the messy challenges in today's healthcare scene—juggling spot-on diagnoses, keeping data private, and making sure the computing side doesn't bog down. We're hoping this fresh framework pushes forward secure and solid e-healthcare fixes while building some trust in medical research efforts [6].

When you dig into the nuts and bolts of blockchain and deep learning, it's key to get how they each play a part in carving out a slick, secure image classification system. Blockchain's this decentralized, spread-out ledger tech that locks down data integrity and security with cryptographic hash tricks and group agreement rules. In our project, MetaMask transactions and the IPFS setup made it a breeze to whip up a clear, tamper-proof trail of where our images came from. Deep learning, a chunk of the machine learning world, steps in to sort images with heavy-hitting neural networks like VGG16, MobileNet, and ResNet [12]. These models, beefed up on big datasets, pull out all the tiny details from images, letting us classify them with precision. Mixing blockchain's openness and safety with deep learning's knack for spotting patterns gives us a rock- solid base for this research, setting the stage for next-level, decentralized healthcare solutions [5].

A. Blockchain Related Terms:

MetaMask: MetaMask is this handy Chrome add-on that hooks you up to the Ethereum blockchain. It's an Ethereum crypto-wallet where you can stash your account(s), and it's a go-to for building and testing smart contracts. When you fire off a transaction, MetaMask figures out the max gas you're cool with spending, then kicks back any leftovers to your account. It also sizes up how much gas a transaction needs and suggests a price to keep it rolling. That includes whipping up a cryptographic signature to double-check the transaction's legit, making sure the image data on the blockchain stays solid and unchangeable. MetaMask takes the headache out of dealing with the Ethereum blockchain, letting folks tap into our decentralized healthcare system while hanging onto their private keys and digital goodies. Adding it to our project bumps up the security and clarity of our image provenance records, sticking right to blockchain's core playbook.

IPFS: The InterPlanetary File System (IPFS) is a big deal for leveling up our project's security, openness, and decentralized vibe. IPFS is this distributed file system that's all about giving a peer-to-peer way to stash and share hypermedia across a network of files or objects. In our research gig, IPFS is like the glue between grabbing images on the website and handling the storage and pickup later on.

When someone drops an image onto the site, IPFS whips up a unique hash ID based on what's in the file. This hash—called the Content Identifier (CID)—is like a digital fingerprint for the image. Since it's tied to the file's guts, any tweak to the image would spit out a whole new hash, keeping the data legit. That's huge for making sure medical images stay real and unchangeable in a healthcare setup.

On top of that, IPFS doesn't lean on just one server—it's decentralized. The image gets spread out across a bunch of nodes in the network, and each one holds a copy that you can grab using the hash ID. That makes it tough for data to get lost or messed with. This whole spread-out, distributed thing vibes perfectly with blockchain's core rules, beefing up the security and trustworthiness of our image classification setup. All in all, IPFS plays a key role in locking down and smoothly running the lifecycle of medical images in our project, making the diagnostic process more solid in decentralized healthcare scenes.

Pinata: Pinata's a big player in our project, acting as the go-between for storing and grabbing images while tying into the blockchain and deep learning action [4]. It taps into

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1799

the InterPlanetary File System (IPFS), this decentralized, spread- out storage setup, to crank up the security and keep the data rock-solid. In our setup, Pinata's the bridge between the website's front end and the blockchain moves that follow. When someone uploads an image to the site, Pinata cooks up a unique hash ID for it through IPFS. That hash is like a crypto fingerprint, pegging the image's content. Then, that hash gets locked onto the blockchain with MetaMask transactions, creating a permanent record of where the image came from and who's got it. The cool part? Pinata spreads the image data across the IPFS network, so there's no single weak spot to worry about—boosting the whole system's toughness and safety. Hooking Pinata into the mix gives us a secure, clear-cut base for sorting and pulling images in the project [15].

Smart Contract: In our project, bringing blockchain tech into the mix means leaning on smart contracts to beef up the security and openness of our image classification system. A smart contract's like a self-running deal where the rules are baked right into the code. Here's how it plays out: when someone uploads an image to the website, a MetaMask trans- action fires up a smart contract. That contract takes charge, automatically logging the image's unique hash ID onto the blockchain, locking in a permanent, decentralized record of where it came from.

The smart contract's the middleman between the user, the Pinata platform, and the blockchain network. It double- checks the transaction details—like the hash ID and any extra metadata—before stashing it all safely on the blockchain. This setup doesn't just give us a clear, open log of the image's history; it also builds a no-trust-needed, tamper-proof space since the contract runs by the blockchain's built-in playbook.

Solidity: Solidity's a key piece in our project since it's a programming language built from the ground up for crafting smart contracts on blockchain setups, with Ethereum being one of the big dogs that runs with it. Smart contracts are these self-running deals where the rules are coded right in. In our research, Solidity's probably pulling some serious weight, tying together the image pickup process with the blockchain. With Solidity, we can spell out the smart contract's game plan, making sure transactions fire off safely and on their own. In our case, when you drop an image onto the website, Solidity likely teams up with the MetaMask wallet to handle transactions on the Ethereum blockchain. The unique hash ID that Pinata whips up for each image gets locked onto the blockchain through a Solidity smart contract. That contract makes sure the hash ID—acting like a digital fingerprint—gets etched into the blockchain for good, giving us a clear, tamper- proof record of where every image came from.

B. Deep Learning Related Terms:

Convolutional Neural Networks (CNNs): CNNs are a flavor of deep neural networks that really shine when it come to crunching images. They're built with convolutional layers that figure out layered patterns in the data all on their own.

In our research, we're rolling with CNNs like VGG16, Mo- bileNet, and ResNet as our go-to models for sorting images. These networks are champs at picking up the tiny details in medical pics, which boosts how spot-on our diagnoses turn out [10].

Training Datasets: Training datasets are big piles of labeled examples we use to whip deep learning models into shape. They're super important for teaching the models to spot patterns and features across all kinds of inputs.

For our project, we trained the deep learning models on a mix of datasets packed with pathological images. The variety and depth in these sets help the models get sharp at handling new images they've never seen before and nailing the classifications.

Data Preprocessing: Data preprocessing is all about taking raw data and tweaking it into something deep learning algo- rithms can work with—like normalizing, resizing, or beefing it up with augmentation.

Before we let the models loose on training, we spruce up the input images to keep things consistent and make it easier for the models to pull out the good stuff.

Transfer Learning: Transfer learning's all about taking what you've learned from one gig and using it to crack another related one. In deep learning, it usually means tapping into models that've already been trained on huge datasets.

In our project, we're jumping on the transfer learning train by using pre-trained CNN models like VGG16, MobileNet, and ResNet. It's a slick way to save time and still get dead-on image classifications, even if our dataset's on the smaller side.

Model Evaluation Metrics: Model evaluation metrics are how we size up how well machine learning models are doing. For classification jobs, the usual suspects are accuracy, precision, recall, and F1 score.

We use these stats to check how sharp our deep learning models are in this study, giving us a full rundown on how they're holding up with pathological image sorting.

Epoch: An epoch's one full lap through the whole training dataset while a machine learning model's getting schooled. How many epochs you run sets how many times the algorithm chews through all that data. Our deep learning models get put through their paces over multiple epochs, letting them soak up the dataset bit by bit.

MaxPooling: MaxPooling's a trick in convolutional neural networks to shrink things down by picking the biggest value from a chunk of numbers in a set window.

You'll spot MaxPooling layers a lot in CNN setups—they're there to trim down the size of the representation step-by-step, helping zero in on the standout features.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1799

Dropout: Dropout's this handy trick we use while training to keep overfitting in check. It's all about randomly switching off a chunk of neurons during each training round, pushing the network to figure out tougher, more solid features.

In our project, we've tossed dropout layers into the neural network setup to help the model roll with new stuff better and cut down the chances of it getting too hung up on the training data.

Dense Layer: Dense layers—aka fully connected lay- ers—are where every neuron in one layer hooks up with every neuron in the next. They're key for piecing together the tricky patterns in the data.

We often stick dense layers at the end of neural networks for image classification gigs, letting the network call the shots based on what it's learned.

ROC-AUC (Receiver Operating Characteristic - Area Under the Curve): ROC-AUC's a go-to stat for sizing up binary classification jobs. It measures the area under the Receiver Operating Characteristic curve, showing how well the model splits the two classes apart.

It's a solid way to gauge how sharp our deep learning models are at picking out differences in our project, especially when it's a straight-up two-class showdown.

ReLU (Rectified Linear Unit): ReLU's a popular activa- tion function in neural networks. It zaps any negative inputs to zero while letting the positive ones slide through, tossing some nonlinearity into the mix.

You'll see ReLU popping up in the hidden layers of neural networks a lot—it's there to shake things up and help the network tackle complex patterns.

Binary Accuracy: Binary accuracy's a straightforward met- ric that tells you the percentage of predictions the model got right in a two-class problem. It's an easy way to check how on-point our models are overall, especially when we're just dealing with a pair of classes.

C. Algorithms

CNN: The convolutional neural network (CNN) setup laid out in the code is all about tackling image classification jobs. It's pieced together with a bunch of core layers—like con- volutional layers, max-pooling layers, global average pooling, dropout layers, and fully connected (dense) layers—working together to get the job done.

➢ *Input Layer:*

The input layer represents the raw pixel values of the input image. It defines the shape of the input data, which in this case is a 3D tensor with dimensions (IMG SIZE, IMG SIZE, 3), where 3 corresponds to the RGB color channels.

$$\text{Input Layer: } X \in \mathrm{R}^{(\text{IMG SIZE, IMG SIZE, 3})}$$

➢ *Data Augmentation Layers (RandomFlip, RandomRotation):*

Data augmentation layers, such as RandomFlip and RandomRotation, introduce variations to the training data to improve model generalization by exposing it to diverse perspectives of the same image.

➢ *Normalization and Batch Normalization Layers:*

The Rescaling layer normalizes pixel values to the range [0, 1], and Batch Normalization stabilizes and accelerates the training process by normalizing inputs during each batch.

$$\text{Rescaling Layer: } X \leftarrow \frac{X}{255}$$

$$\text{Batch Normalization: } \hat{X} = \frac{X-\mu}{\sqrt{\sigma^2+\epsilon}}, \quad Y = \gamma \,\&\, \hat{X} + \beta$$

➢ *Convolutional Layers with ReLU Activation:*

Convolutional layers perform feature extraction by applying convolutional filters to the input image. The ReLU activation function introduces non-linearity. Convolutional Layer: $Z = W * X + b, A = \text{ReLU}(Z)$

➢ *Max-Pooling Layers:*

Max-pooling layers downsample the spatial dimensions of the feature maps, retaining the most important information. Max-Pooling Layer: $Y = \text{MaxPooling2D}(X)$ modules inside its MBConv blocks to keep things humming. The network's layout comes from a mashup of hardware- smart network architecture search (NAS) and the NetAdapt algorithm to nail down what works best.

➢ *The hard Swish Activation Function is Defined as:*

$$h_{swish}(x) = x \cdot \sigma(x) = x \cdot \frac{1}{1 + e^{-\beta x}}$$

Where β is this tweak that dials in how steep the sigmoid function's slope gets. The hard swish activation function's a smoother take on the ReLU trick, and it's been shown to bump up both the accuracy and speed of the network.

The squeeze-and-excitation module's this cool setup that fine-tunes how each channel's features respond by figuring out how they lean on each other. It's got two parts: squeeze and excitation. The squeeze bit crunches down channel-specific stats using global average pooling. Then, in the excitation step, it cooks up channel-wise weights with a non-linear function the network learns along the way. Here's how the module comes together:

➢ *Global Average Pooling Layer:*

Global Average Pooling H W reduces the spatial dimensions to a single value per feature map, capturing essential information for classification.

- *Global Average Pooling Layer:*

$$\mathbf{z} = F_{sq}(\mathbf{u}) = \frac{1}{H \times W} \sum_{i=1}^{HW} \sum_{j=1} \mathbf{u}_{i,j}$$

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1799

$$Y = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} X_{ij}$$

$$\mathbf{s} = F_{ex}(\mathbf{z}, \mathbf{W}) = \sigma(g(\mathbf{z}, \mathbf{W})) = \sigma(\mathbf{W}_2\delta(\mathbf{W}_1\mathbf{z}))$$

➢ *Dropout Layers:*

Dropout layers randomly deactivate neurons during training, preventing overfitting by enhancing model generalization.

➢ *Dense Layers with ReLU and Sigmoid Activation:*

Dense layers serve as fully connected layers, learning high-level features from the global average pooling output. The final layer employs a sigmoid activation for binary classification.

Dense Layer with ReLU Activation:

$Z = W \cdot A + b, \quad A = \text{ReLU}(Z)$

Dense Layer with Sigmoid Activation:

$Z = W \cdot A + b, \quad A = \text{Sigmoid}(Z)$

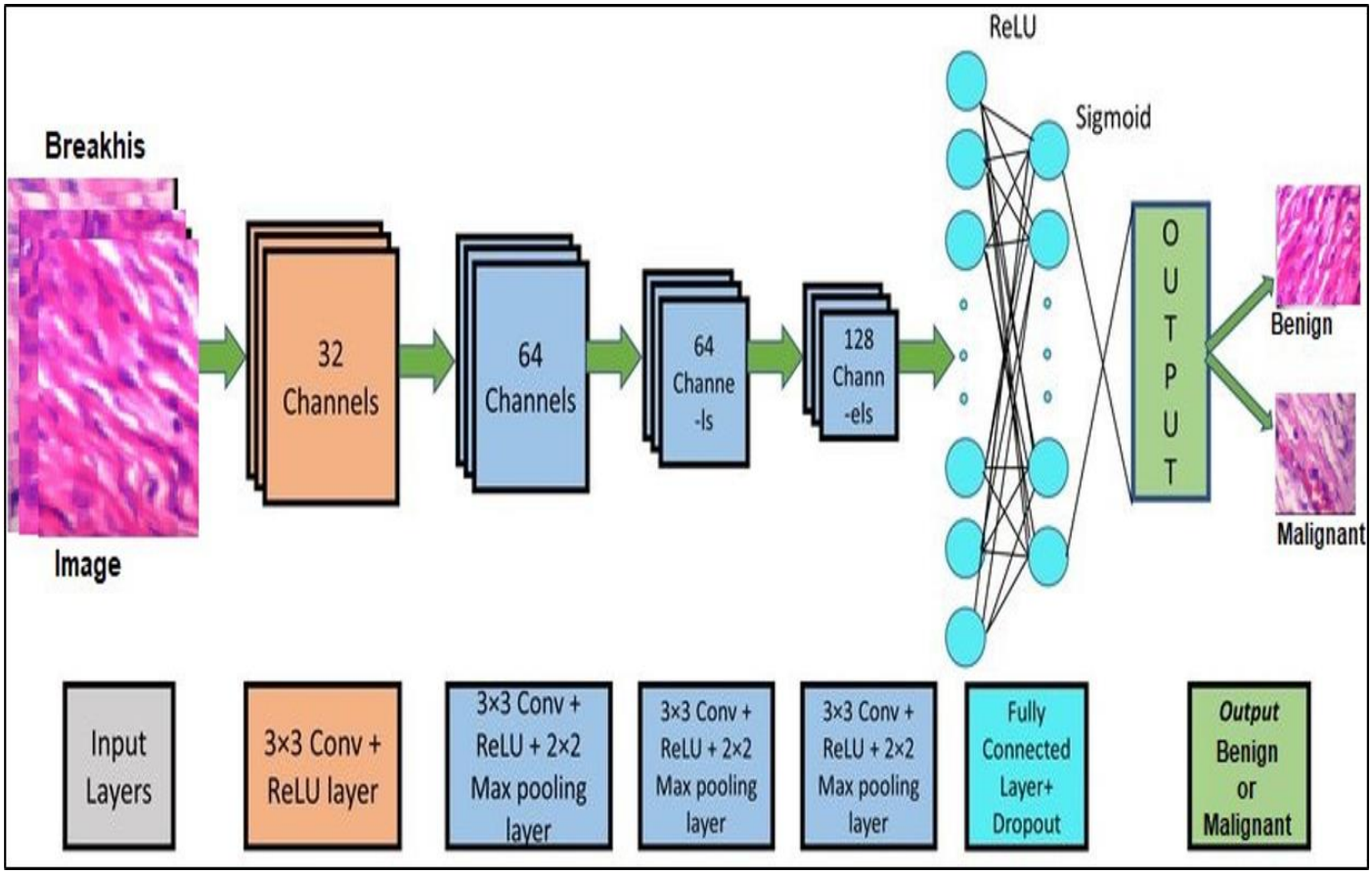

Fig 2 how to cnn classifier work

➢ *MobileNetV3:*

MobileNetV3 is a convolutional neural net- work built from the ground up for mobile phone CPUs. It leans on hard swish activation and squeeze-and-excitation

$$\mathbf{x} = F_{scale}(\mathbf{u}, \mathbf{s}) = \mathbf{s} \cdot \mathbf{u}$$

where u is the input feature map, z is the channel descriptor, s is the channel weight vector, x is the output feature map, W1 and W2 are the parameters of the excitation network, δ is the ReLU function, and σ is the sigmoid function. The squeeze-and-excitation module's a neat trick that boosts the network's ability to shine by picking up on how the channels play off each other.

The MBConv block's a spin on the inverted residual bot- tleneck block you see a lot in MobileNetV2. It's built from three layers: a pointwise convolution that beefs up the input channels, a depthwise convolution that handles the spatial filtering, and another pointwise convolution that squeezes the channels down to the output size we want. Here's how the MBConv block breaks down:

$$\mathbf{x} = F_{MBConv}(\mathbf{u})$$

$$= \mathbf{u} + h_{swish}(\mathbf{W}_3 \cdot h_{swish}(F_{SE}(\mathbf{W}_2 \cdot h_{swish}(\mathbf{W}_1 \cdot \mathbf{u}))))$$

where **u** and **x** are the input and output feature maps, $\mathbf{W}_1$, $\mathbf{W}_2$, and $\mathbf{W}_3$ are the convolutional filters, F_{SE} is the squeeze-and-excitation module, and h_{swish} is the hard swish activation function. The MBConv block can improve the efficiency and performance of the network by reducing the computational cost and increasing the non-linearity.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1799

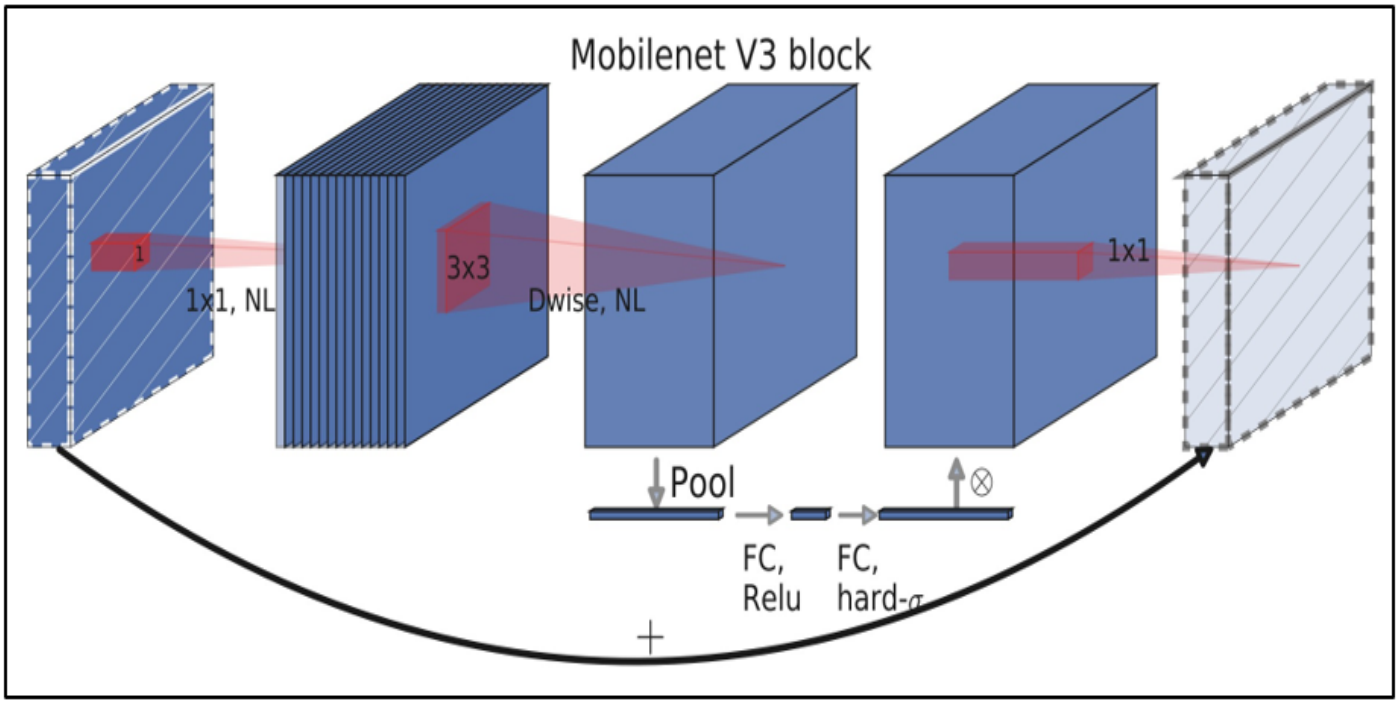

Fig 3 how to mobilenet-v3 classifier work

EfficientNetB0: EfficientNetB0 is a convolutional neural network (CNN) that rolls with the EfficientNet setup and its scaling trick. The EfficientNet design uses this cool compound scaling approach that bumps up the network's depth, width, and resolution all at once, guided by a set batch of scaling numbers. Those numbers come from running a grid search on the starter model (EfficientNet-B0), tweaking it to max out accuracy without hogging too many resources.

The compound scaling method can be expressed by the following equation:

$$\text{depth: } d = \alpha^{\phi}$$

$$\text{width: } w = \beta^{\phi}$$

$$\text{resolution: } r = \gamma^{\phi}$$

$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$

$$\text{and } \alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

where ϕ is the compound coefficient that controls the amount of scaling, and α, β, and γ are the constants that are determined by the grid search. The base model (EfficientNet-B0) has ϕ = 1, α = 1.2, β = 1.1, and γ = 1.15.

➢ *The Efficientnet-B0 Model has the following Architecture:*

- The input layer grabs an image sized 224 × 224 × 3 and runs a 3 × 3 convolution with 32 filters and a stride of 2.
- The first stage rolls with one inverted residual block, using an expansion factor of 1, a kernel size of 3, and spitting out 16 channels.
- The second stage stacks up two inverted residual blocks, cranking the expansion factor to 6, keeping the kernel size at 3, and pushing out 24 channels. The first block's got a stride of 2.
- The third stage also has two inverted residual blocks, with an expansion factor of 6, a bigger kernel size of 5, and an output of 40 channels. The first block strides at 2.
- The fourth stage brings three inverted residual blocks, sticking with an expansion factor of 6, a kernel size of 3, and bumping up to 80 channels. The first block's stride is 2.
- The fifth stage keeps it at three inverted residual blocks, with an expansion factor of 6, a kernel size of 5, and 112 output channels. The first block's stride drops to 1.
- The sixth stage goes big with four inverted residual blocks, an expansion factor of 6, a kernel size of 5, and 192 channels on the output. The first block strides at 2.
- The seventh stage wraps with one inverted residual block, an expansion factor of 6, a kernel size of 3, and 320 output channels. Its stride's a chill 1.
- The output stage ties it up with a 1 × 1 convolution rocking 1280 filters, a global average pooling, and a fully connected layer with 1000 units and softmax activation to seal the deal.

Now, the inverted residual block's a twist on the usual residual block, swapping in depthwise separable convolutions instead of the regular kind. It's got three main pieces: an expansion layer, a depthwise convolution layer, and a projection layer. The expansion layer pumps up the channel count by a factor of t (that's the expansion factor), the depthwise convolution layer slaps a spatial convolution on each channel, and the projection layer trims the channels back down to match the output. Plus, it's got a skip connection that tosses the input onto the output if they're the same size.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1799

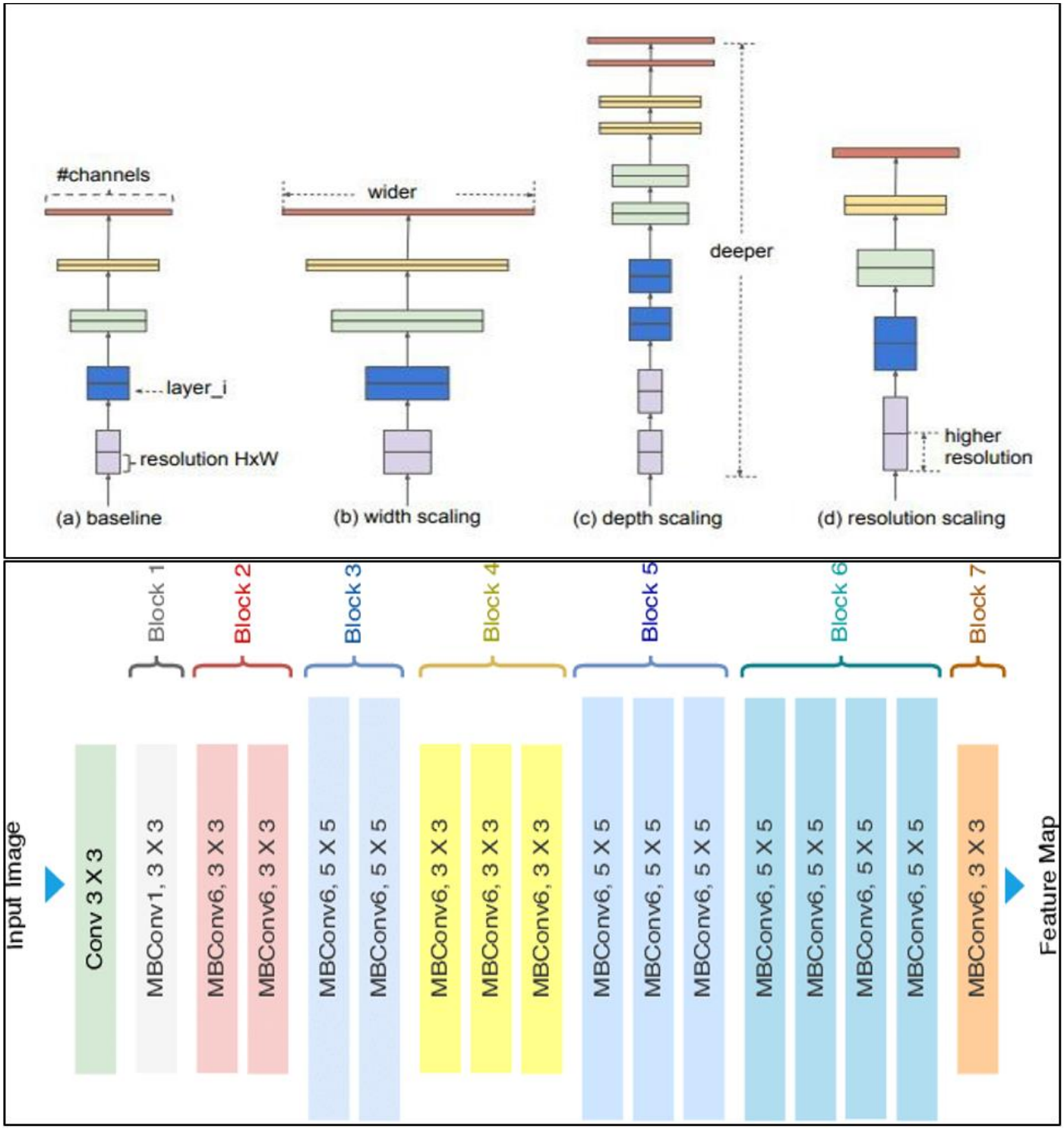

Fig 4 how to EfficientNet-B0 classifier work

VGG16: Simonyan and Zisserman rolled out VGG16, a deep convolutional neural network (CNN), back in 2014. It crushed it on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), which is like the gold standard for image classification and object detection. VGG16's got 16 layers in total—13 convolutional layers, a couple of fully connected ones, and a final output layer. What makes it stand out is how it sticks to 3x3 filters with a stride of 1 and padding of 1 across all its convolutional layers. That setup keeps the parameter count low while letting the network dig really deep. It also throws in max pooling layers with a 2x2 window and stride of 2 after some of the convolutional layers to shrink down

the feature maps' size. At the end, the last feature map gets flattened and sent through two beefy 4096-unit fully connected layers, with a 0.5 dropout layer in between to keep overfitting in check. The output's a 1000-unit softmax layer, matching up with ImageNet's 1000 classes [1].

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1799

➢ *The Architecture of VGG16 can be Summarized as follows:*

Table 1 Network Architecture

Layer	Type	Filter size	Number of filters	Output size
Input	Image	-	-	224 × 224 × 3
Conv1 1	Convolution	3 × 3	64	224 × 224 × 64
Conv1 2	Convolution	3 × 3	64	224 × 224 × 64
Pool1	Max pooling	2 × 2	-	112 × 112 × 64
Conv2 1	Convolution	3 × 3	128	112 × 112 × 128
Conv2 2	Convolution	3 × 3	128	112 × 112 × 128
Pool2	Max pooling	2 × 2	-	56 × 56 × 128
Conv3 1	Convolution	3 × 3	256	56 × 56 × 256
Conv3 2	Convolution	3 × 3	256	56 × 56 × 256
Conv3 3	Convolution	3 × 3	256	56 × 56 × 256
Pool3	Max pooling	2 × 2	-	28 × 28 × 256
Conv4 1	Convolution	3 × 3	512	28 × 28 × 512
Conv4 2	Convolution	3 × 3	512	28 × 28 × 512
Conv4 3	Convolution	3 × 3	512	28 × 28 × 512
Pool4	Max pooling	2 × 2	-	14 × 14 × 512
Conv5 1	Convolution	3 × 3	512	14 × 14 × 512
Conv5 2	Convolution	3 × 3	512	14 × 14 × 512
Conv5 3	Convolution	3 × 3	512	14 × 14 × 512
Pool5	Max pooling	2 × 2	-	7 × 7 × 512
FC1	Fully connected	-	4096	4096
Dropout1	Dropout	-	-	4096
FC2	Fully connected	-	4096	4096
Dropout2	Dropout	-	-	4096
FC3	Fully connected	-	1000	1000
Softmax	Softmax	-	-	1000

➢ *Convolutional Layer:*

$$y_{i,j,k} = \sum_{l=1}^{C} \sum_{m=0}^{F-1} \sum_{n=0}^{F-1} w_{m,n,l,k} x_{i+m,j+n,l} + b_k$$

where x is the input feature map of size H × W × C, y is the output feature map of size H′ ×W′ ×K, w is the filter of size F × F × C × K, b is the bias of size K, i and j are the spatial indices of the output feature map, and k is the channel index of the output feature map. The output height and width H′ and W′ are computed as:

$$H' = \frac{H + 2P - F}{S} + 1$$

$$W' = \frac{W + 2P - F}{S} + 1$$

where P is the padding size and S is the stride size. The mathematical formula of a max pooling layer can be written as:

$$y_{i,j,k} = \max_{m=0}^{R-1} \max_{n=0}^{R-1} x_{iS+m,jS+n,k}$$

where x is the input feature map of size H × W × C, y is the output feature map of size H′ ×W′ ×C, R is the pooling window size, S is the stride size, and i, j, and k are the indices of the output feature map. The output height and width H′ and W′ are computed as:

$$H' = \frac{H - R}{S} + 1$$

$$W' = \frac{W - R}{S} + 1$$

➢ *Fully Connected Layer:*

$$y_i = \sum_{j=1}^{N} w_{i,j} x_j + b_i$$

Where x is the input vector of size N , y is the output vector of size M , w is the weight matrix of size M ×N , b is the bias vector of size M , and i is the index of the output vector.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1799

➢ *Softmax Layer:*

$$y_i = \frac{\exp(x_i)}{\sum_{j=1}^{K} \exp(x_j)}$$

where x is the input vector of size K, y is the output vector of size K, and i is the index of the output vector. The output vector represents the probability distribution over the K classes.

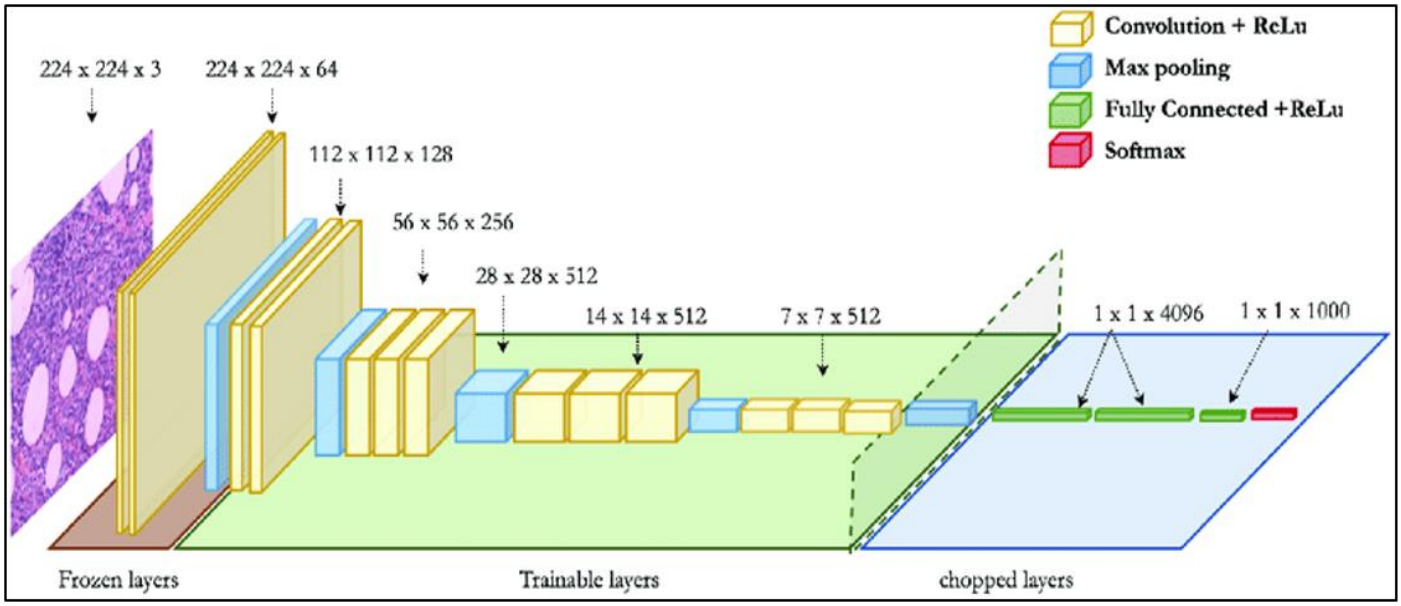

Fig 5 how to VGG16 classifier work

ResNet50V2: ResNet50V2 is a deep convolutional neural network that leans on residual learning to dodge the degrada- tion headache that comes with training super deep networks. That degradation mess is when a network's accuracy hits a wall and then tanks hard as you pile on more layers, even if you've got everything set up just right with initialization and regularization. ResNet50V2 sidesteps this by throwing in shortcut connections that hop over some layers and do identity mappings, letting the network focus on learning the leftovers—what's left to tweak—instead of trying to nail the whole output from scratch.

Figure 5 breaks down the core pieces of ResNet50V2. It's built with three convolutional layers teamed up with batch normalization and ReLU activation, then wraps up with an element-wise addition using that shortcut connection. The shortcut can either be a straight identity pass or a projection, depending on whether the input and output sizes match up. When it's a projection shortcut, it uses a 1x1 convolution with a stride of 2—shrinking the spatial size while beefing up the channel count. After the addition, the result runs through one more ReLU activation to keep things rolling.

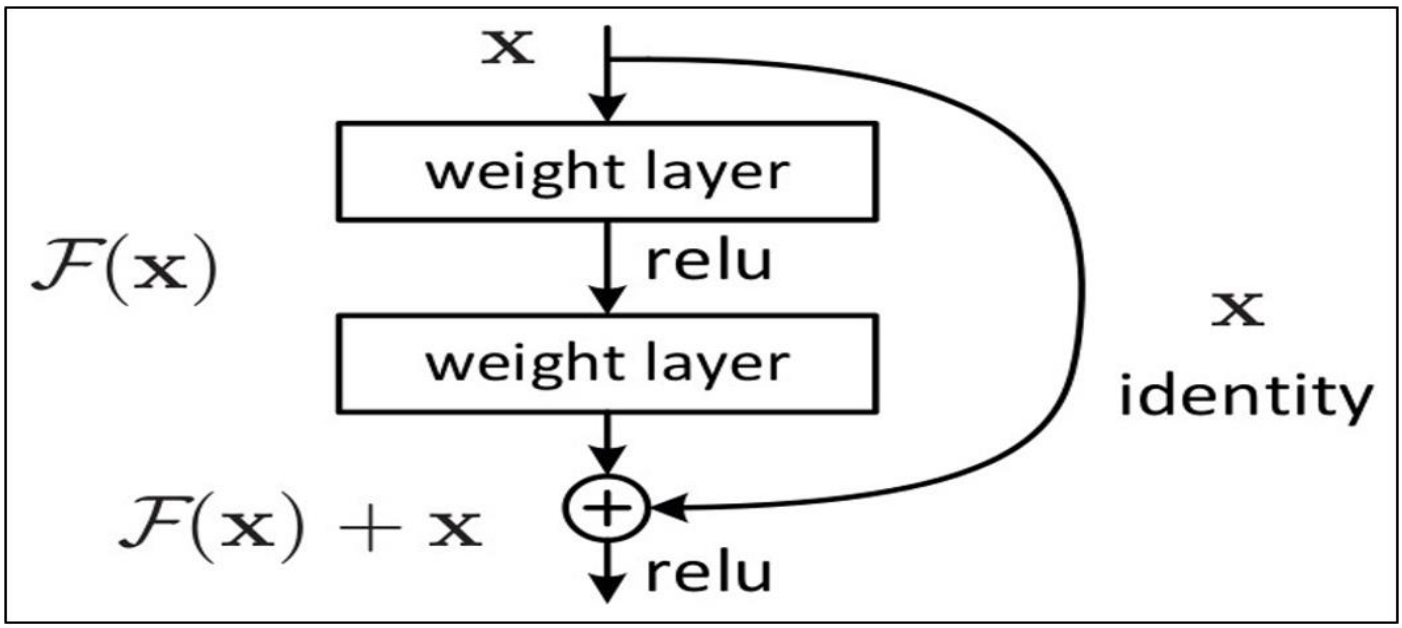

Fig 6 The basic building block of ResNet50V2.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1799

The ResNet50V2 network's got 50 layers and follows the layout you can check out in Table 1. It kicks off with a 7x7 convolution packing 64 filters and a stride of 2, then rolls into a 3x3 max pooling with another stride of 2. After that, it piles up four sets of residual blocks, each with its own filter count and number of blocks. The first set's rocking 64 filters across three blocks, the second jumps to 128 filters with four blocks, the third bumps up to 256 filters and six blocks, and the fourth tops out at 512 filters with three blocks. In each set, the first block gets a projection shortcut with a stride of 2, while the rest stick with identity shortcuts. Once the last residual block spits out its output, it flows into a global average pooling and wraps up with a fully connected layer using sigmoid activation to handle binary classification.

Table 2 The Architecture of Resnet50v2.

Layer name	Output size	ResNet50V2
conv1	112x112	7x7, 64, stride 2
pool1	56x56	3x3 max pool, stride 2
conv2 x	56x56	1x1, 64 3x3, 64 x 3 1x1, 256
conv3 x	28x28	1x1, 128 3x3, 128 x 4 1x1, 512
conv4 x	14x14	1x1, 256 3x3, 256 x 6 1x1, 1024
conv5 x	7x7	1x1, 512 3x3, 512 x 3 1x1, 2048
pool5	1x1	global average pool
fc	1	1-d sigmoid

We can train the ResNet50V2 model using stochastic gra- dient descent (SGD), tossing in some momentum and weight decay for good measure. The learning rate starts at 0.1 and gets slashed by 10 every 30 rounds. We set the batch size to 256 and let it run for 90 rounds total. Weight decay's dialed in at 0.0001, and momentum's sitting at 0.9. To kick things off, we use He normal initialization, with the bias zeroed out. To help it play nice with new data, we throw in some data augmentation tricks like random cropping, flipping, and color jittering.

The ResNet50V2 model pumps out top-tier results on all sorts of image classification jobs—like ImageNet, CIFAR-10, and CIFAR-100. It's also a solid pick for pulling features for computer vision tasks like spotting objects, breaking down scenes, or recognizing faces.

II. RELATED WORK

Image classification is a core gig in computer vision—it's all about slapping a label on an image based on what's in it. It's got a ton of uses, from picking out faces and helping with medical diagnoses to spotting objects and powering self-driving cars. The tricks for pulling this off range from old-school computer vision moves to the latest deep learning models [2].

Back in the day, classic computer vision relied on hand-picked features—think colors, textures, shapes, and key-points—to sum up an image. Those features got fed into classifiers like support vector machines (SVM), k-nearest neighbors (KNN), or decision trees to guess the label. But these methods had their hiccups: you needed serious know-how, they freaked out over noise or blocked views, and they struggled to grab the big-picture stuff. Plus, they often needed a mountain of labeled data to train, which isn't always around or easy to get [13].

Deep learning, though, flips the script by using neural networks to figure out both features and classifiers straight from the data. These networks are built with layers of neurons, each one tweaking the input a bit before passing it along. The early layers act like feature hunters, picking up basics like edges and corners from raw pixels. The later layers step up as classifiers, nailing down high-level stuff like faces or objects and spitting out the label. Deep learning's been killing it in image classification, especially since convolutional neural networks (CNNs) came into the mix [6].

CNNs are a special breed of neural network that lean on how images are laid out. They use convolutional layers that slap filters on the image, churning out feature maps. Each filter' s on the lookout for something specific—like an edge or a blob. Then come the pooling layers, which shrink down the feature maps with tricks like max-pooling or average-pooling, making the image tougher against shifts, spins, or zooms. These convolutional and pooling layers stack up into a deep setup, with each layer digging into more abstract and tricky features than the one before. The final layer's usually a fully connected one that seals the deal by picking the label [3].

CNNs have taken off big-time and keep getting better for image classification. Some heavy hitters include VGG16, MobileNet, and ResNet. VGG16 is a deep CNN with 16 lay-ers—13 convolutional and 3 fully connected. It uses tiny 3x3 filters and big strides (2) to nail high accuracy on ImageNet, a dataset with 1,000 classes and 1.2 million pics. MobileNet's

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1799

a leaner CNN that runs on depthwise separable convolutions, cutting down on parameters and crunch time—perfect for mobile or low-power gear. ResNet's a super deep CNN that uses residual connections—little shortcuts that skip layers and mix the previous layer's output into the next. That trick dodges the vanishing gradient mess that crops up when networks get too deep, letting ResNet hit peak performance on ImageNet with up to 152 layers [11].

Still, deep learning's got its downsides—like needing a ton of computing juice, being open to sneaky attacks, and not always explaining itself clearly. Plus, it often leans on centralized data, which can stir up privacy and security worries, especially in touchy areas like healthcare or finance. To tackle that, some folks are mixing blockchain tech into the deep learning game [1].

Blockchain's like a shared notebook that locks in transactions so they're safe and can't be messed with. It keeps data legit with crypto tricks like hash functions and digital signatures, and it uses consensus rules—like proof-of-work or proof-of-stake—to keep everyone on the same page. It's big in stuff like crypto cash, smart contracts, and tracking supply chains.

Blockchain can give deep learning a boost in a few ways—like sharing data, training models, and checking results. Data sharing's about passing info between different players. Blockchain makes it secure and decentralized, letting everyone stash and grab data from the chain without a middleman. Model training's about tuning a neural network's settings with data. Blockchain lets folks team up by training local models on their own stuff, then swapping parameters or gradients through the chain. Result checking's about making sure the network's output holds up. Blockchain keeps it open and unchangeable, so anyone can double-check the outcome and where it came from by peeking at the records [5] [6].

A bunch of studies have dug into pairing blockchain with deep learning for image classification. Take Shafay et al.—they cooked up a hybrid permissioned blockchain and deep learning setup for classifying CT images of pneumonia patients. They used a residual neural network and spread the model weights across five hospitals via blockchain, showing it bumped up ac- curacy while keeping data private. Then there's Zou et al., who built a deep learning model for spotting diabetic retinopathy, mixing in blockchain and African vulture optimization (AVO). They used a TaylorAVO trick to pull the best image features and trained a SqueezeNet model on the blockchain network, proving it could nail diabetic retinopathy fast and right [12].

III. DATASETS USED

The BreaKHis database is packed with microscopic biopsy images of breast tumors, collected during a clinical study in Brazil from January to December 2014. Folks diagnosed with breast cancer were asked to join in, and all their data got scrubbed of personal details after getting the ethical green light. The biopsy samples were stained with hematoxylin and eosin, then run through the usual paraffin process for a solid histological breakdown. Pics were snapped at 40×, 100×, 200×, and 400× zooms using an Olympus microscope paired with a Samsung digital camera. The original shots got cropped and saved in RGB format, no compression applied.

The capture process involved taking images at different zoom levels within a specific area that pathologists flagged as the region of interest. The dataset's got a mix of images—some with transitional tissue, some without. At the end, they did a hands-on check to weed out any blurry or out-of-focus shots. In the BreaKHis database, the 400x magnification images give you a super close-up look at what's going on with breast tissue pathology. These pics are snapped using an Olympus

Table 3 Magnification and Digital Resolution of the Acquisition System

Visual Magnification	Objective Lens	Effective Pixel Size (m)
40×	4×	0.49
100×	10×	0.20
200×	20×	0.10
400×	40×	0.05

Table 4 Image Distribution by Magnification Factor and Class

Magnification	Benign	Malignant	Total
40×	625	1370	1995
100×	644	1437	2081
200×	623	1390	2013
400×	588	1232	1820
Total	**2480**	**5429**	**7909**
# Patients	**24**	**58**	**82**

BX-50 system microscope hooked up with a 3.3x relay lens and a Samsung digital color camera (SCC-131AN). That camera's rocking a 1/3 Sony Super-HAD interline transfer charge-coupled device, with pixels sized at 6.5 µm × 6.25 µm and a total count of 752 × 582 [11] [2]. During the capture process, pathologists scope out the tumors and zero in on a region of interest. They start by grabbing images at the lowest zoom (40×) to get the whole area in view. Then, they crank it up by hand to 100×, 200×, and finally 400×, snapping about the same number of shots at each level. This step-by-step zoom-in lets them check out the tissue from all angles and depths.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1799

Once the images are in the bag, they do a quick eyeball check to toss out any blurry ones. The keepers get saved in three-channel RGB format, no compression, with an 8-bit depth each. They come out at 700 × 460 pixels. The raw images might have some black borders or text notes hanging around, but those get scrubbed out during processing, leaving clean shots without any normalization or color tweaking [9].

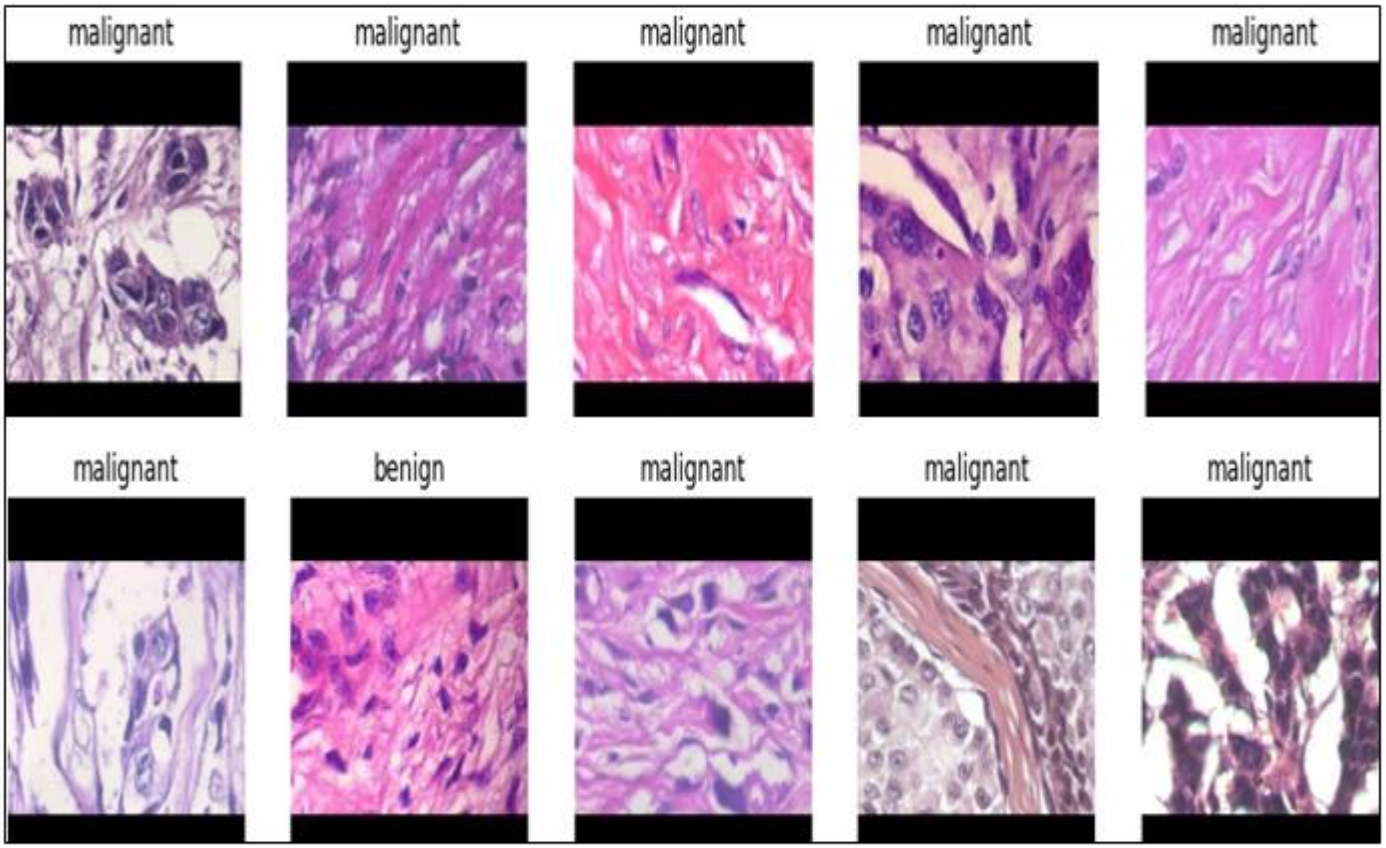

Fig 7 Sample of Dataset

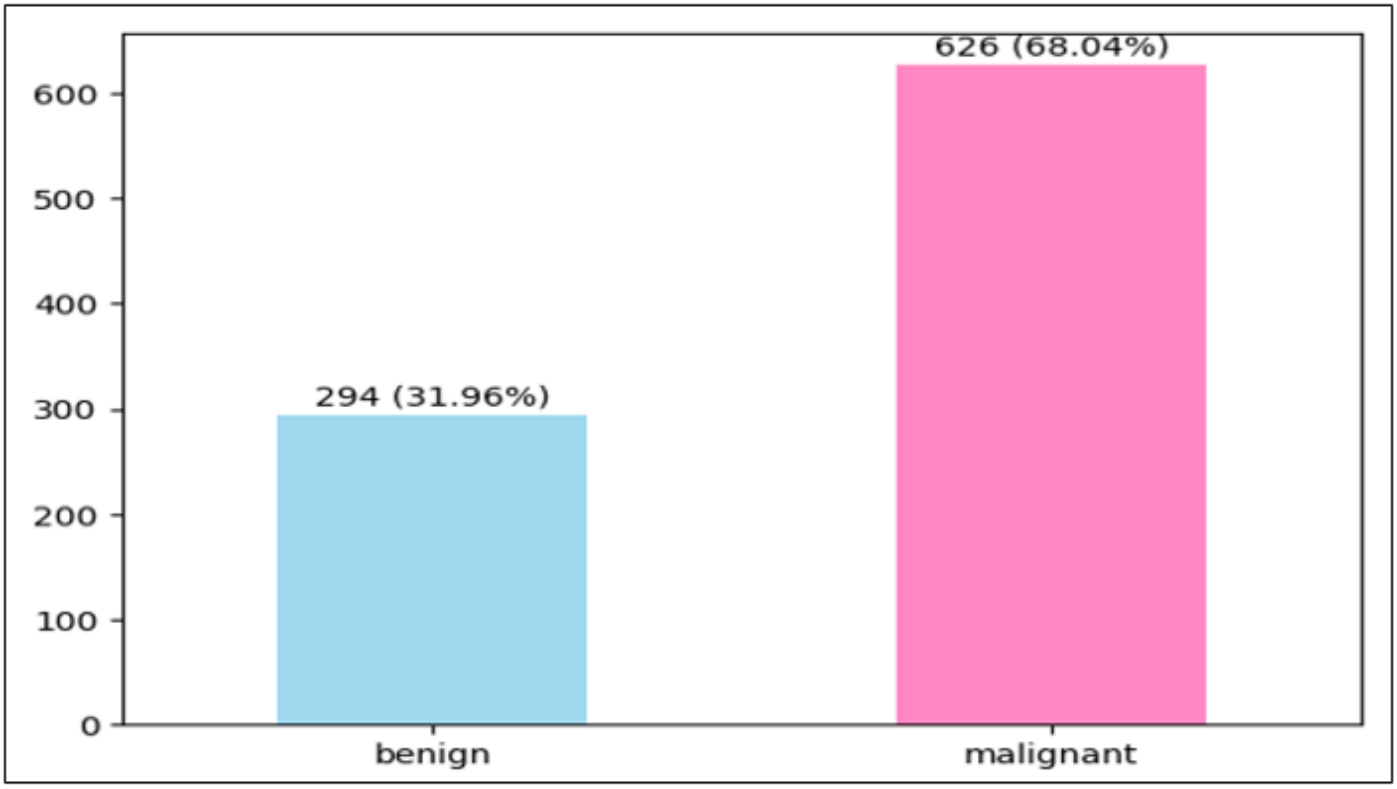

Fig 8 Training Dataset

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1799

IV. METHODOLOGY

A. Image Capture Process

The image grab starts when folks upload their medical pics to our website. Those images get sent over to Pinata—a decentralized storage spot—through a locked-down communication setup. Pinata whips up a one-of-a-kind hash ID for each image using the InterPlanetary File System (IPFS). That hash ID acts like a special tag for the image on the IPFS network.

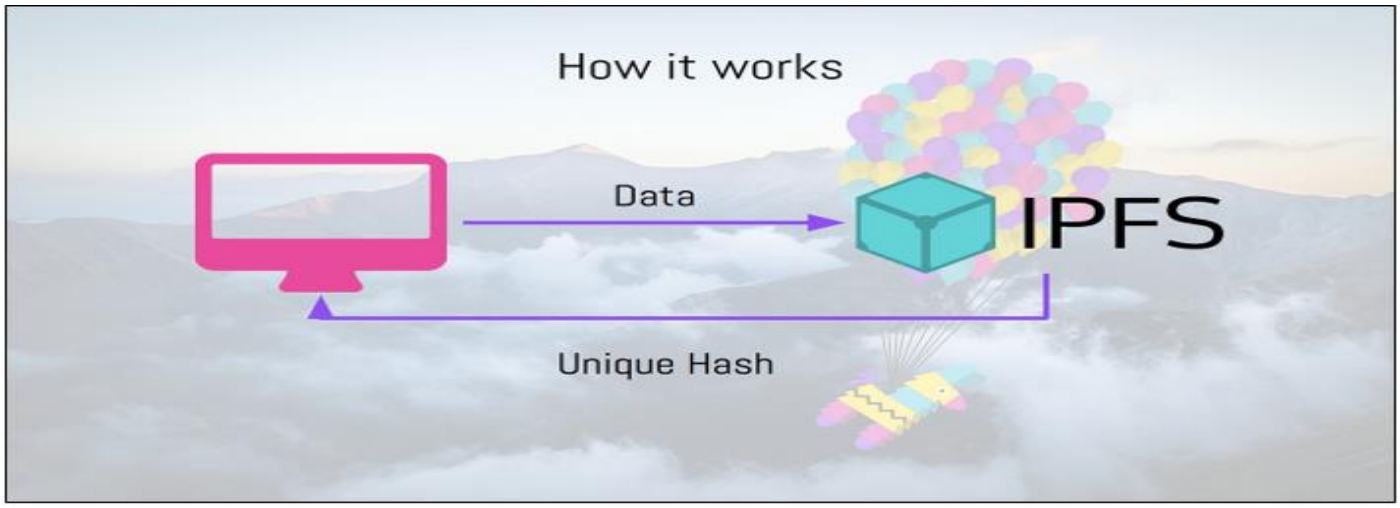

Fig 9 Image Hash Id Generate

B. Blockchain Integration using MetaMask

To keep the data legit and above board, we're leaning on blockchain tech to stash and double-check those hash IDs. MetaMask, this handy crypto wallet and Ethereum blockchain hookup, smooths out the process of dealing with image hash IDs. When someone drops an image into the system, a Meta- Mask transaction kicks off, tacking the hash ID onto a smart contract living on the Ethereum blockchain. That move gets locked into the blockchain, leaving a rock-solid, unchangeable trail of every image upload.

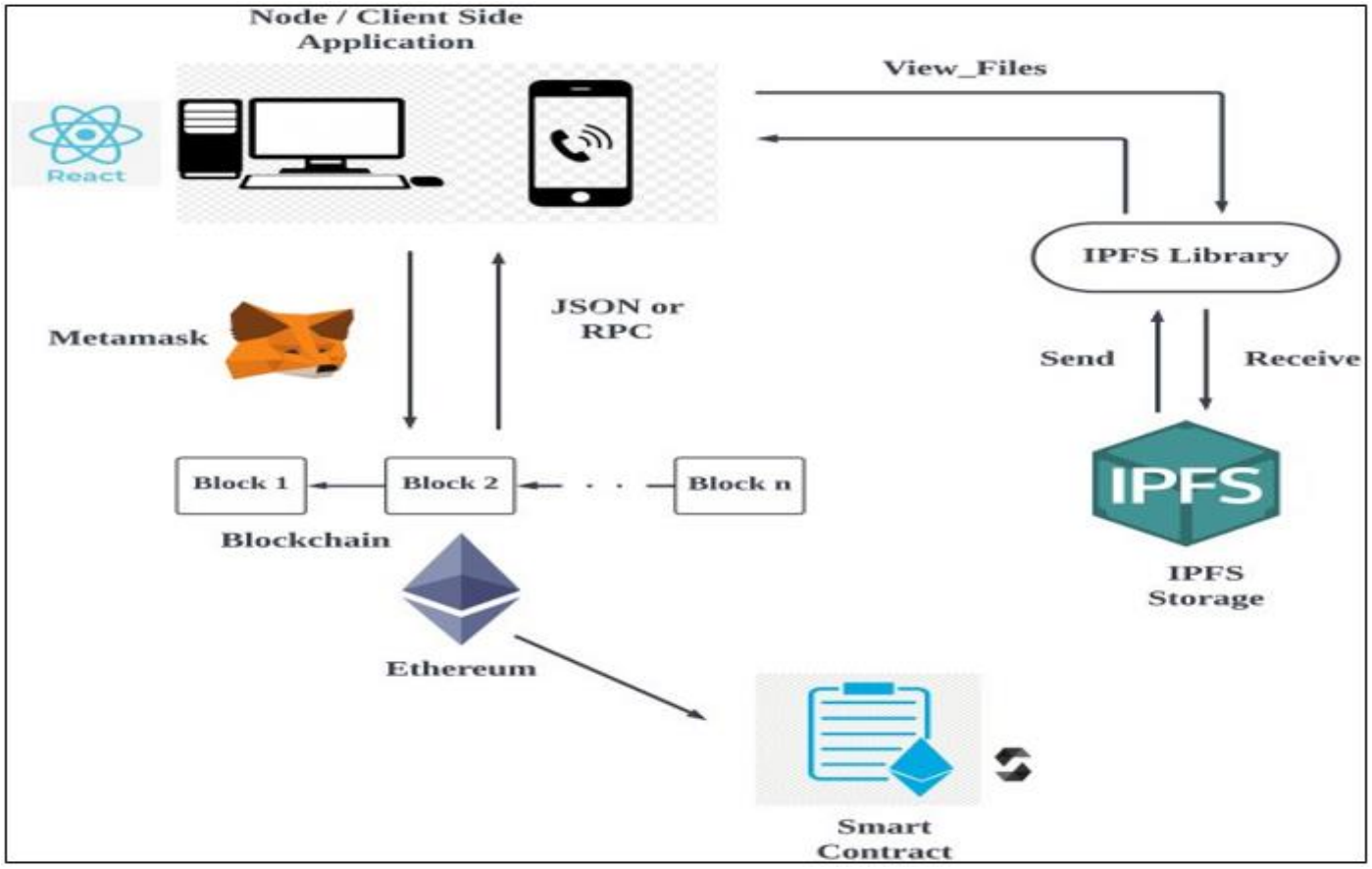

Fig 10 Add Block Using Metamask

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1799

C. Deep Learning Models for Image Classification

Our image classification setup rolls with three top-notch deep learning models: VGG16, MobileNet, and ResNet. VGG16 Model: We picked VGG16 because it's a beast at spotting tricky patterns, thanks to its deep setup. It comes pre-trained on a huge dataset, and we tweaked it on the BreakHis400x dataset to get it dialed in for the quirks of medical images.

MobileNet Model: MobileNet's our go-to for its speed and lightweight design, perfect for running on setups like cloud servers where resources might be tight. We trained it to nail that sweet spot between accuracy and not hogging too much computing power.

ResNet Model: ResNet's in the mix because of its deep residual learning trick, which keeps the vanishing gradient issue at bay. We got it trained up to pick out the subtle stuff in medical images so it classifies them spot-on.

Every one of these models got a thorough workout on the BreakHis400x dataset, which has [insert number] samples in the mix. We use the [insert optimization algorithm] for model training, with [insert hyperparameter details] [7].

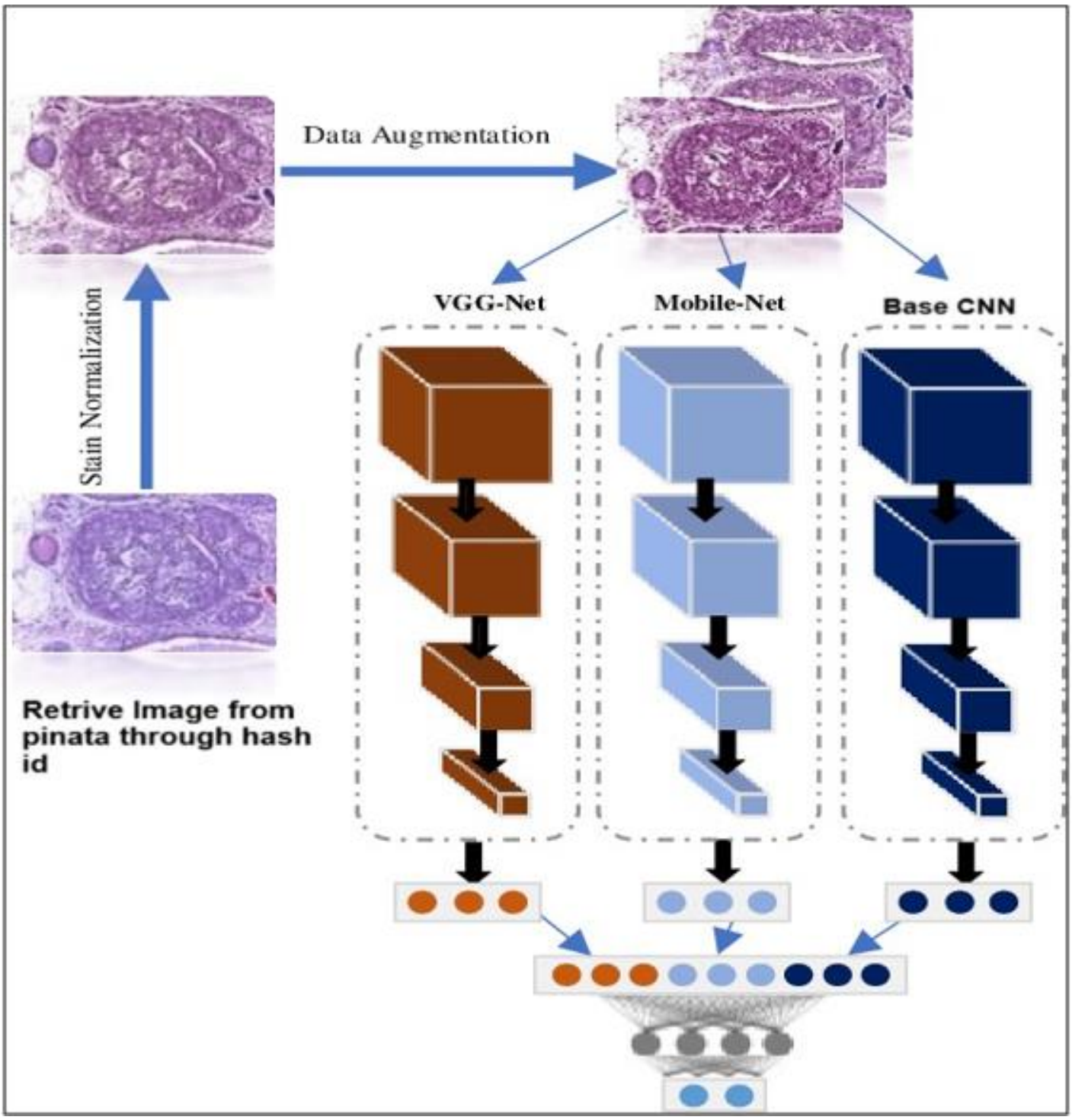

Fig 11 Deep Learning Models classifier work

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1799

To determine the most effective model for our medical image classification task, we evaluate the ensemble of these models using classification accuracy, precision, recall, and the F1 score.

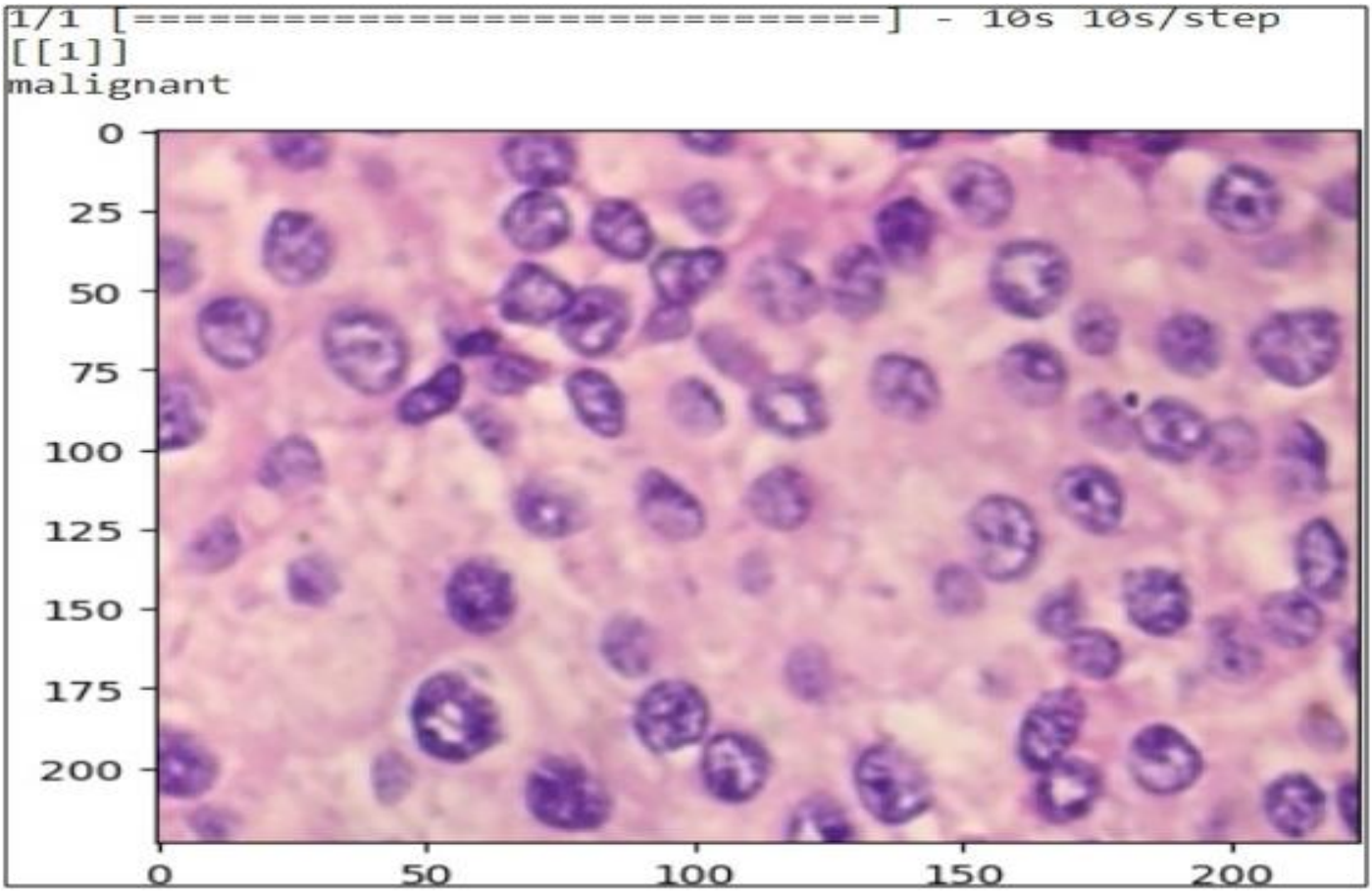

Fig 12 Classification Result

D. Server-Client Model for Result Display

We've set up a server-client setup to show off the classifica- tion results right on our website. The trained ensemble model lives on a cloud server, while the website plays the client role. When someone hunts for a specific image on the site, it fires off a request to the cloud server. The server digs up the matching image from the blockchain using its hash ID, runs it through the ensemble model for a real-time classification, and shoots the results back to the website. This whole deal ties together the deep learning models and the user interface like a charm, giving folks instant access to the classification scoop [14].

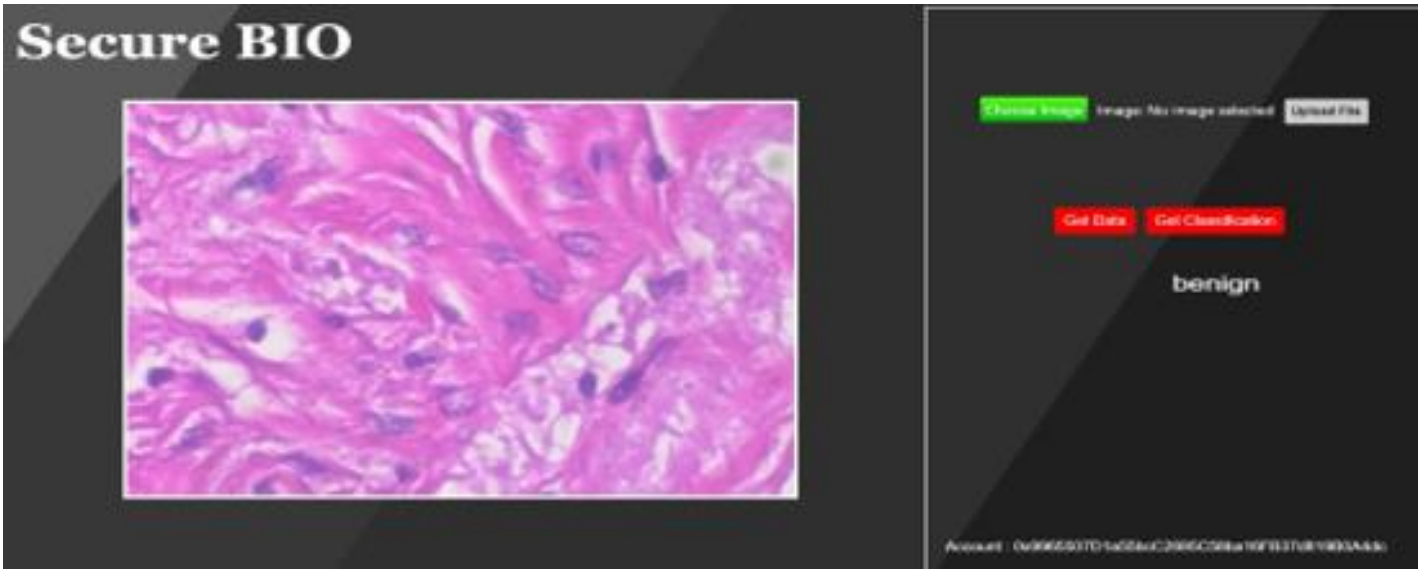

Fig 13 Classification Result on Website

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1799

V. RESULT ANALYSIS

In this part, we're laying out what we found from messing around with the BreakHis 400x dataset, which has 1,696 breast tumor images split into two groups: benign and ma- lignant. We're sharing the scoop from our tests with the breast cancer (BreakHis) dataset, packed with two kinds of im- ages—malignant and benign. We put five deep learning models through the wringer: baseCNN, MobileNetV3, EfficientNetB0, VGG16, and ResNet50v2. We sized them up using stuff like ROC-AUC, accuracy, and loss. We chopped the dataset into 54% for training, 13.6% for validation, and 32.4% for testing. To spice up the training set, we threw in some random flips and spins with data augmentation. We ran the show with the Adam optimizer, set the learning rate at 0.001, used binary cross-entropy for loss, and tracked AUC and accuracy as our go-to metrics. Plus, we tossed in early stopping and learning rate cuts to keep overfitting in check and help things come together smoother.

A. Base CNN

We rolled with a baseCNN model that starts with a random flip layer, then stacks up four convolutional blocks and a dense layer. The model architecture is shown in Table V.

We set the model up to train for up to 35 rounds, but it called it a day at round 25 because of an early stop. The figure lays out the training and validation curves for accuracy and loss. You can tell the model smashed it—hitting high accuracy and keeping loss nice and low on both the training and validation sets. That shows it really got the image details down pat without overcooking or half-baking the data.

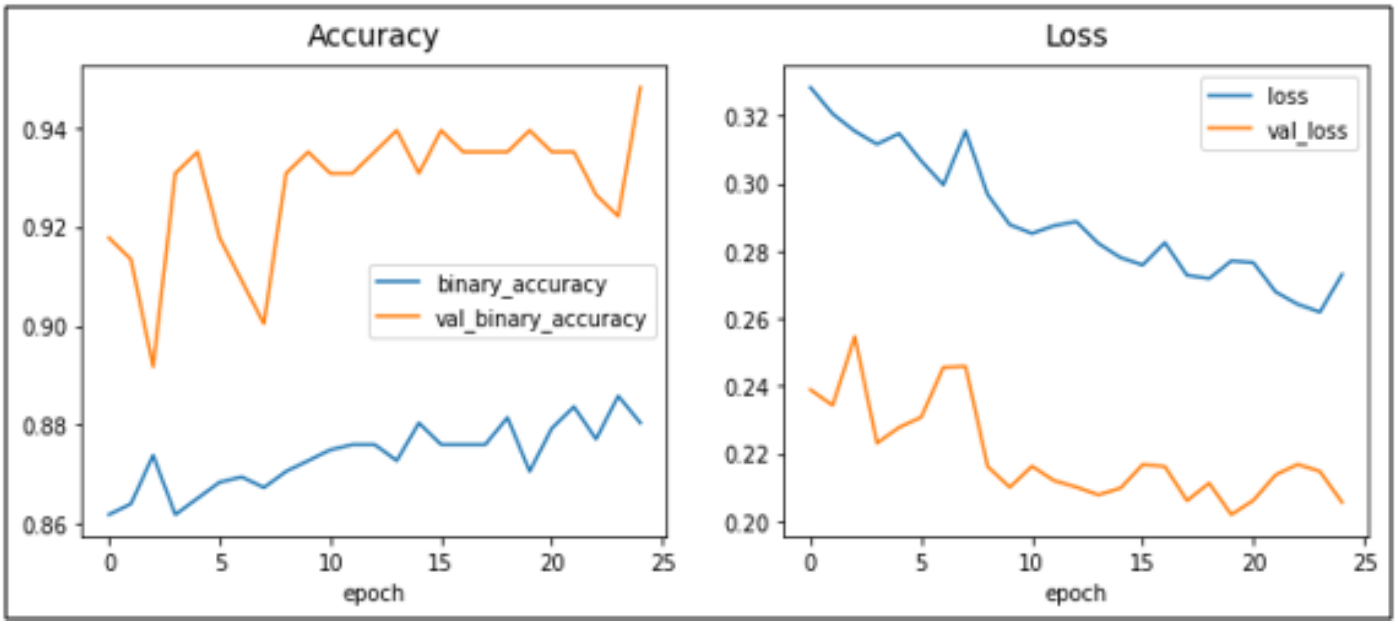

Fig 14 Training and validation curves for accuracy and loss

Table 5 Model Architecture of Basecnn

Layer (type)	Output Shape	Param #
random flip 6 (RandomFlip)	(None, 224, 224, 3)	0
random rotation 6 (RandomRotation)	(None, 224, 224, 3)	0
rescaling 5 (Rescaling)	(None, 224, 224, 3)	0
batch normalization 1 (BatchNormalization)	(None, 224, 224, 3)	12
conv2d 3 (Conv2D)	(None, 222, 222, 32)	896
max pooling2d 6 (MaxPooling2D)	(None, 111, 111, 32)	0
conv2d 4 (Conv2D)	(None, 109, 109, 64)	18496
max pooling2d 7 (MaxPooling2D)	(None, 54, 54, 64)	0
conv2d 5 (Conv2D)	(None, 52, 52, 64)	36928
max pooling2d 8 (MaxPooling2D)	(None, 26, 26, 64)	0
global average pooling2d 2 (GlobalAveragePooling2D)	(None, 64)	0
dropout 12 (Dropout)	(None, 64)	0
dense 24 (Dense)	(None, 256)	16640
dropout 13 (Dropout)	(None, 256)	0
dense 25 (Dense)	(None, 64)	16448

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1799

The model racked up a ROC-AUC of 0.93354, an accuracy of 0.86789, and a loss of 0.30512. These numbers prove it handled fresh, unseen data like a champ and should work great for new stuff too.

We also sketched out how the model did on some test images, like you can see in the figure. It nailed most of the calls, though it slipped a few times, mixing up benign and malignant tumors. That might've happened because the image details were too close or there was some static in the data throwing it off.

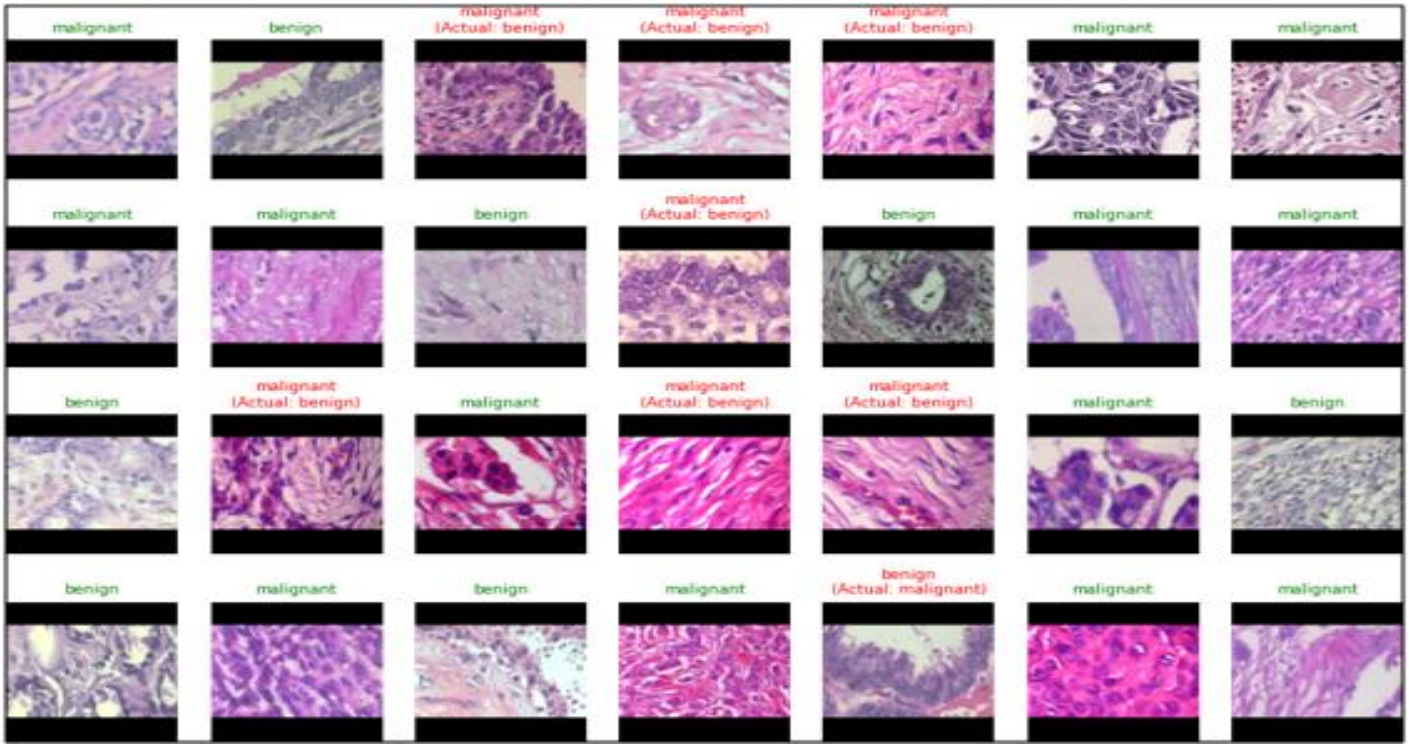

Fig 15 Predictions of the model on some test images

We put together a baseCNN model to tackle breast cancer classification, using deep learning and histopathology images. It did a solid job on the BreakHis dataset, showing it's got the chops to pick up image details and tell benign tumors apart from the malignant ones. There's room to level it up, though—maybe by tapping into fancier setups like ResNet, Inception, or DenseNet, or borrowing some smarts from pre- trained models with transfer learning. We could also throw it at other datasets like BACH or IDC to see how tough and adaptable it really is.

B. MobileNetv3

We used a MobileNetv3 model, which consists of a random flip layer followed by four convolutional blocks and a dense layer. The model architecture is shown in Table VI.

Table 6 Model Architecture of Mobilenetv3

Layer (type)	Output Shape	Param #
random_flip (RandomFlip)	(None, 224, 224, 3)	0
random_rotation (RandomRotation)	(None, 224, 224, 3)	0
lambda (Lambda)	(None, 224, 224, 3)	0
MobilenetV3small (Functional)	(None, 576)	939,120
dropout (Dropout)	(None, 576)	0
dense_3 (Dense)	(None, 256)	147,712
dropout_1 (Dropout)	(None, 256)	0
dense_4 (Dense)	(None, 64)	16,448
dense_5 (Dense)	(None, 1)	65
Total params: 1,103,345 Trainable params: 164,225 Non-trainable params: 939,120		

We lined up the model to train for up to 35 rounds, but it wrapped up at round 24 thanks to an early stop. The figure

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1799

breaks down the training and validation curves for accuracy and loss. It's obvious the model killed it—nailing high accuracy and keeping loss down low on both the training and validation sets. That tells us it really locked in the image details without going too hard or dropping the ball on the data.

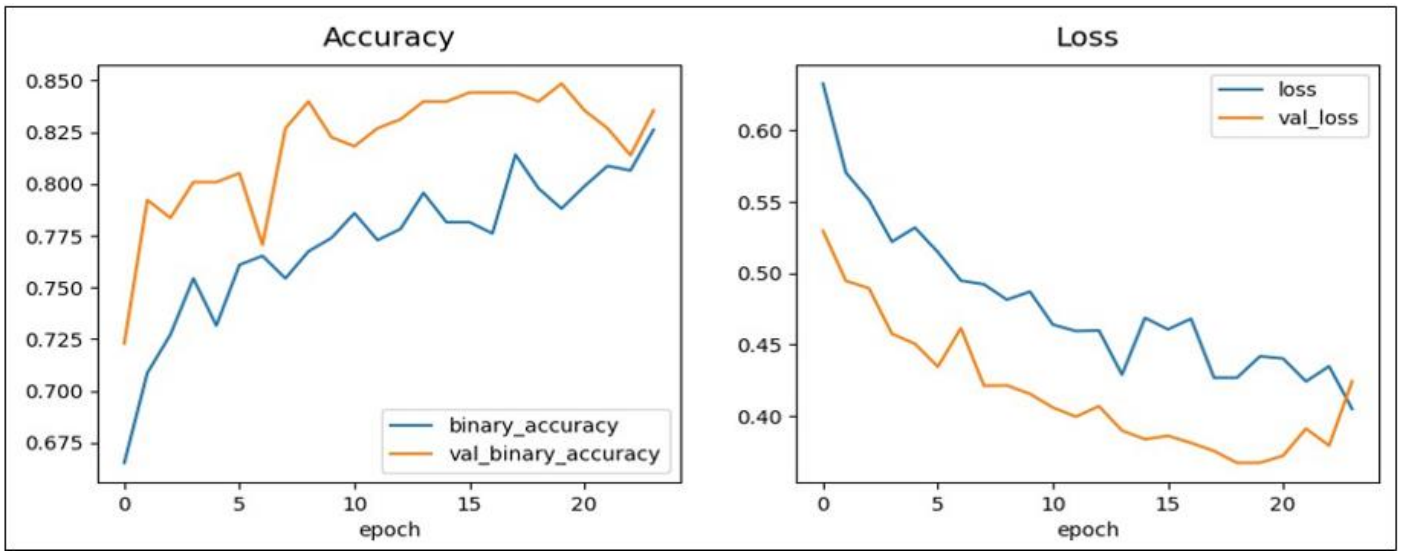

Fig 16 Training and validation curves for accuracy and loss

The model pulled off a ROC-AUC of 0.85414, an accuracy of 0.81101, and a loss of 0.44172. These stats show it handled new, unseen data like a pro and should do just fine with fresh cases.

We also mapped out how the model did on some test images, like you can see in the figure. It got most of them spot- on, though it tripped up a few times, mixing up benign and malignant tumors. That might've happened because the image details were too close or there was some junk in the data throwing it off.

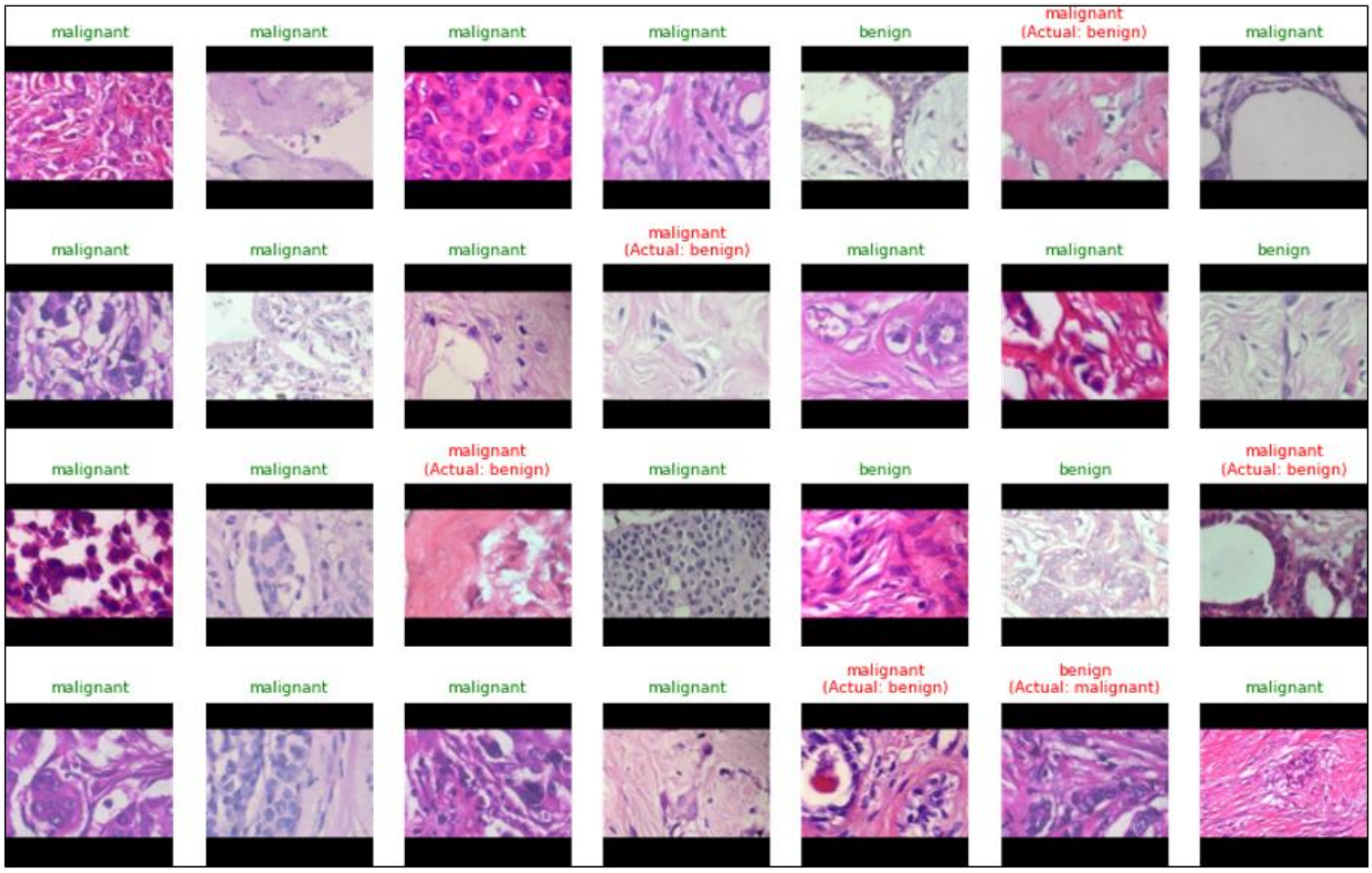

Fig 17 Predictions of the model on some test images

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1799

The MobileNetV3 model comes out on top compared to the other four, rocking the best ROC-AUC, accuracy, and loss numbers on the test set. Plus, it keeps things light with a low parameter count, thanks to its smart setup with depthwise separable convolutions and inverted residual blocks. It's a killer choice for image classification gigs that need fast, spot-on results on stuff like mobile gadgets or edge computing setups.

C. EfficientNetB0

We used a EfficientNetB0 model, which consists of a random flip layer followed by four convolutional blocks and a dense layer. The model architecture is shown in Table VII.

Table 7 Model Architecture of Efficientnetb0

Layer (type)	Output Shape	Param #
random flip 1 (RandomFlip)	(None, 224, 224, 3)	0
random rotation 1 (RandomRotation)	(None, 224, 224, 3)	0
efficientnetb0 (Functional)	(None, 1280)	4,049,571
dropout 2 (Dropout)	(None, 1280)	0
dense 6 (Dense)	(None, 256)	327,936
dropout 3 (Dropout)	(None, 256)	0
dense 7 (Dense)	(None, 32)	8,224
dense 8 (Dense)	(None, 1)	33
Total params: 4,385,764		
Trainable params: 336,193 Non-trainable params: 4,049,571		

We set the model to train for up to 35 rounds, but it bowed out at round 14 because of an early stop. The figure lays out the training and validation curves for accuracy and loss. You can see the model crushed it—scoring high accuracy and keeping loss low on both the training and validation sets. That's a clear sign it picked up the image details like a champ and didn't overdo it or fall short on the data.

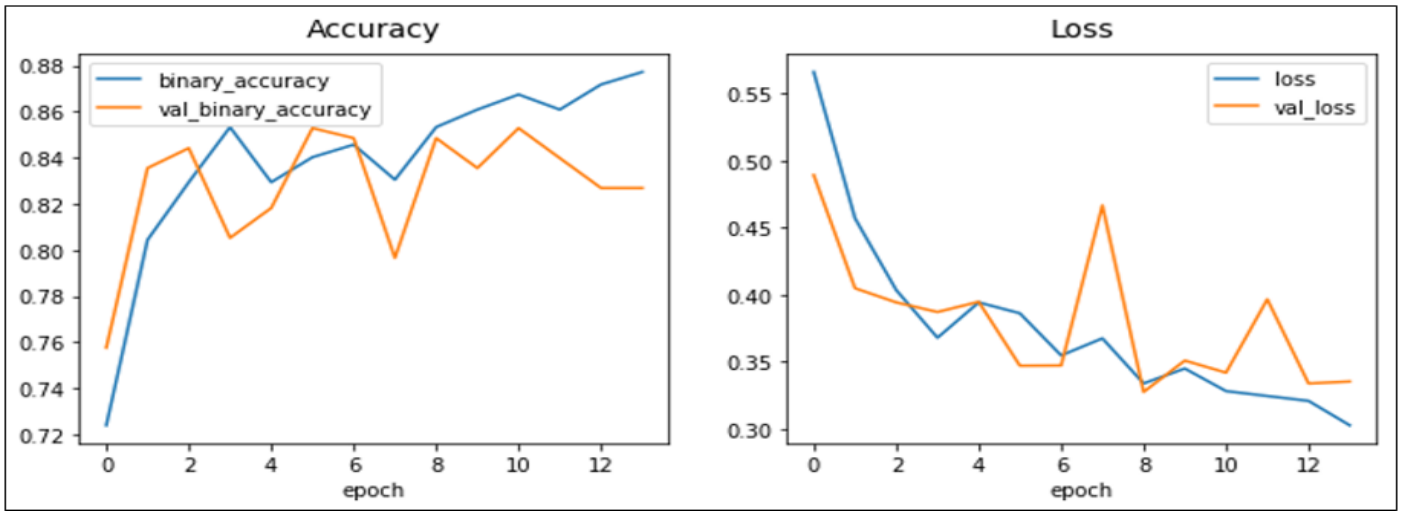

Fig 18 Training and validation curves for accuracy and loss

The model clocked in with a ROC-AUC of 0.90801, an accuracy of 0.82936, and a loss of 0.37056. These numbers show it did a great job on data it hadn't seen before and should work fine for new stuff too.

We also sketched out how the model handled some test images, like you can check out in the figure. It nailed most of the classifications, though it slipped up a few times, getting benign and malignant tumors mixed up. That might be because the image details were too similar or there was some fuzz in the data messing things up.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1799

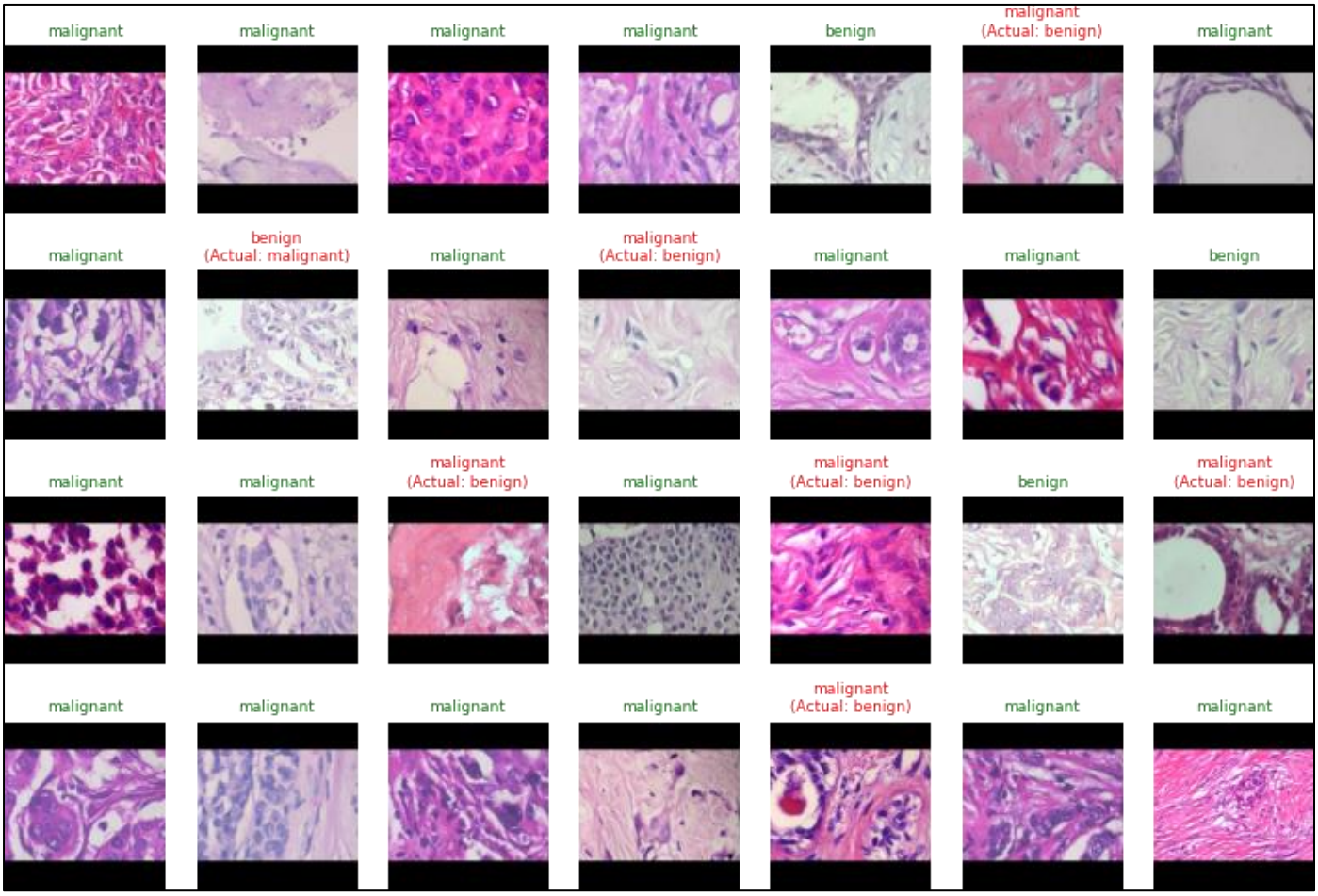

Fig 19 Predictions of the model on some test images

The EfficientNetB0 model runs pretty close to the MobileNetV3 model, though it's packing a few more parameters. It's got this smart compound scaling trick that juggles the network's depth, width, and resolution to keep things balanced. It's a solid pick for jobs where you need to strike a deal between performance and complexity, or where you can throw extra resources at it to scale up the network.

D. VGG16

We used a VGG16 model, which consists of a random flip layer followed by four convolutional blocks and a dense layer. The model architecture is shown in Table VIII.

We geared up the model to train for up to 35 rounds, but it called it quits at round 17 thanks to an early stop. The figure breaks down the training and validation curves for accuracy and loss. It's clear the model knocked it out of the park—hitting high accuracy and keeping loss nice and low on both the training and validation sets. That shows it really got the hang of the image details without going overboard or slacking on the data.

Table 8 Model Architecture of Vgg16

Layer (type)	Output Shape	Param #
- - random flip 2 (RandomFlip)	(None, 224, 224, 3)	0
- - random rotation 2 (RandomRotation)	(None, 224, 224, 3)	0
- lambda 1 (Lambda)	(None, 224, 224, 3)	0
vgg16 (Functional)	(None, 512)	14,714,688
- dropout 4 (Dropout)	(None, 512)	0
- dense 9 (Dense)	(None, 256)	131,328
- dropout 5 (Dropout)	(None, 256)	0
- dense 10 (Dense)	(None, 32)	8,224
dense 11 (Dense)	(None, 1)	33
Total params: 14,854,273		
Trainable params: 139,585 Non-trainable params: 14,714,688		

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1799

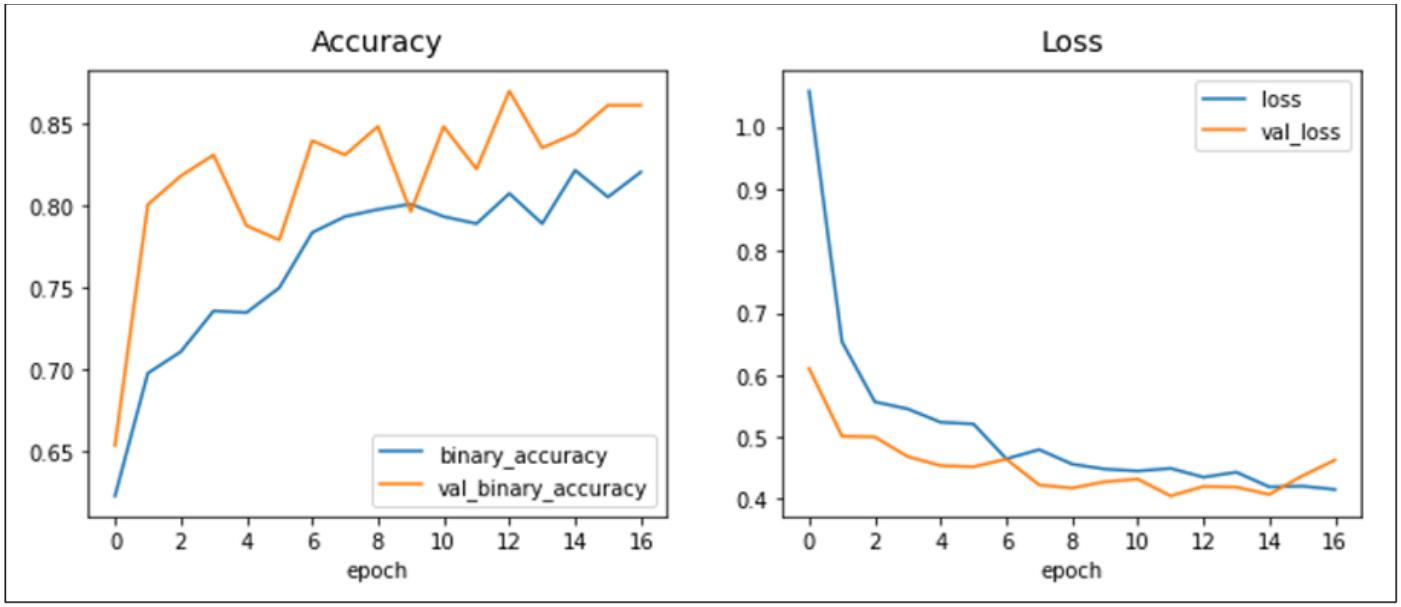

Fig 20 Training and validation curves for accuracy and loss

The model scored a ROC-AUC of 0.87348, an accuracy of 0.79633, and a loss of 0.44824. These numbers tell us it handled fresh, unseen data pretty well and should hold up for new situations.

We also mapped out how the model did with some test images, like you can see in the figure. It got most of them right, but stumbled a bit here and there, mixing up benign and malignant tumors. That might've happened because the image details were too close or there was some static messing with the data.

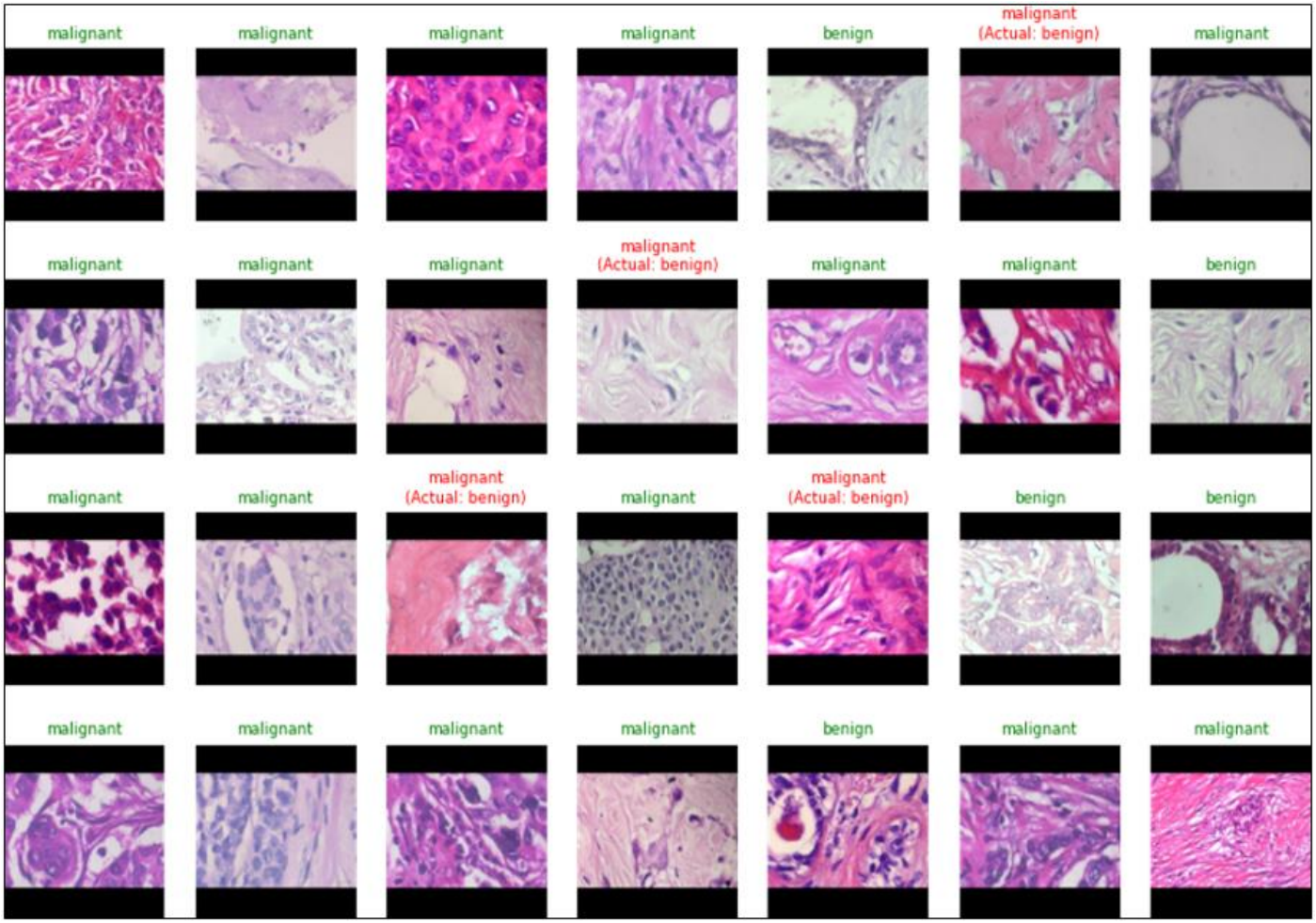

Fig 21 Predictions of the model on some test images

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1799

The VGG16 model is the heaviest of the five when it comes to parameters, but it's also the weakest performer after the baseCNN model. It's got a straightforward, no- fuss design—stacked with a ton of convolutional and pooling layers, then topped off with fully connected ones. Thing is, it might be overfitting and swinging too wild because it struggles to roll with new data. It could use a boost from tricks like dropout or batch normalization, or even fancier stuff like skip connections or attention tweaks to tighten things up.

E. ResNet50v2

We used a ResNet50v2 model, which consists of a random flip layer followed by four convolutional blocks and a dense layer. The model architecture is shown in Table IX.

Table 9 Model Architecture of Resnet50v2

Layer (type)	Output Shape	Param #
random flip 3 (RandomFlip)	(None, 224, 224, 3)	0
random rotation 3 (RandomRotation)	(None, 224, 224, 3)	0
lambda 2 (Lambda)	(None, 224, 224, 3)	0
resnet50v2 (Functional)	(None, 2048)	23,564,800
dropout 6 (Dropout)	(None, 2048)	0
dense 12 (Dense)	(None, 256)	524,544
dropout 7 (Dropout)	(None, 256)	0
dense 13 (Dense)	(None, 32)	8,224
dense 14 (Dense)	(None, 1)	33
Total params: 24,097,601		
Trainable params: 532,801 Non-trainable params: 23,564,800		

We set the model up to train for a max of 35 rounds, but it tapped out at round 15 because of an early stop. The figure lays out the training and validation curves for accuracy and loss. You can tell the model crushed it—scoring high accuracy and keeping loss low on both the training and validation sets. That's a sign it picked up the image details like a pro and didn't overdo or slack off on the data.

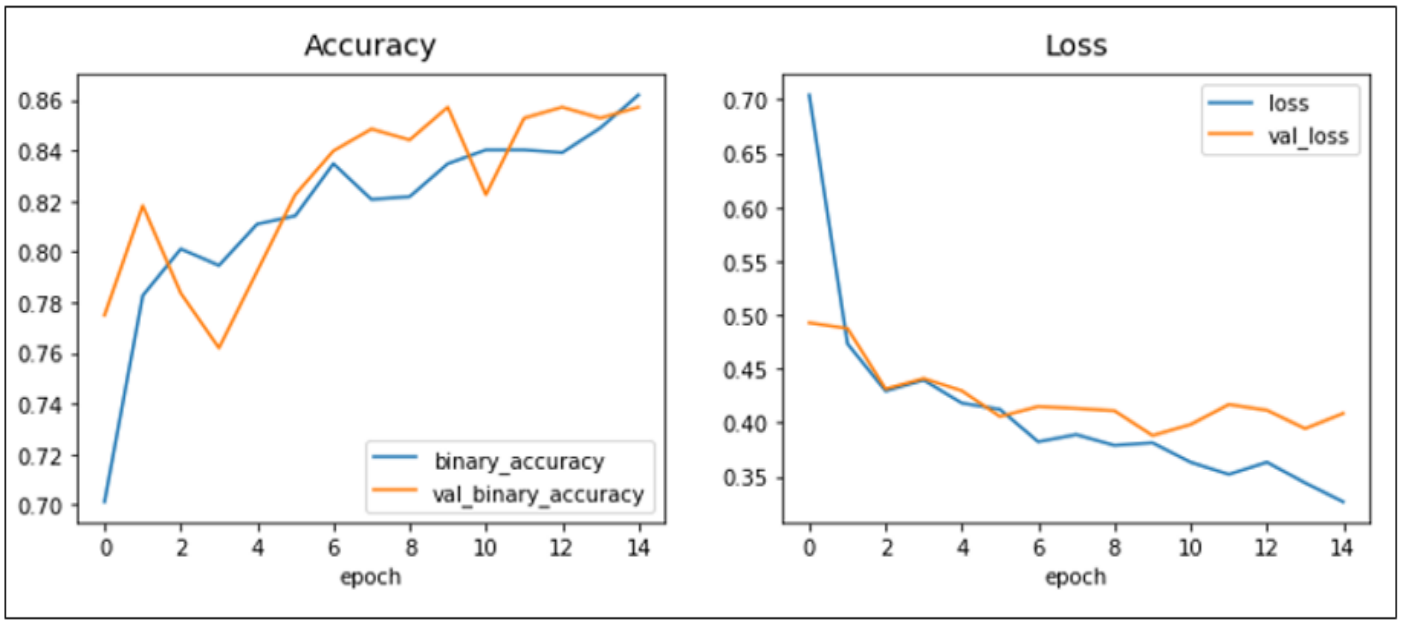

Fig 22 Training and validation curves for accuracy and loss

The model pulled off a ROC-AUC of 0.88756, an accuracy of 0.82752, and a loss of 0.39751. These numbers show it did a solid job handling data it hadn't seen before, making it a good fit for tackling new stuff.

We also threw together some plots of the model's predic- tions on a handful of test images, like you can see in the figure. It nailed most of the classifications, though it tripped up a few times, mixing up benign and malignant tumors. That might've happened because the image details were too close to call or there was some fuzz in the data.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1799

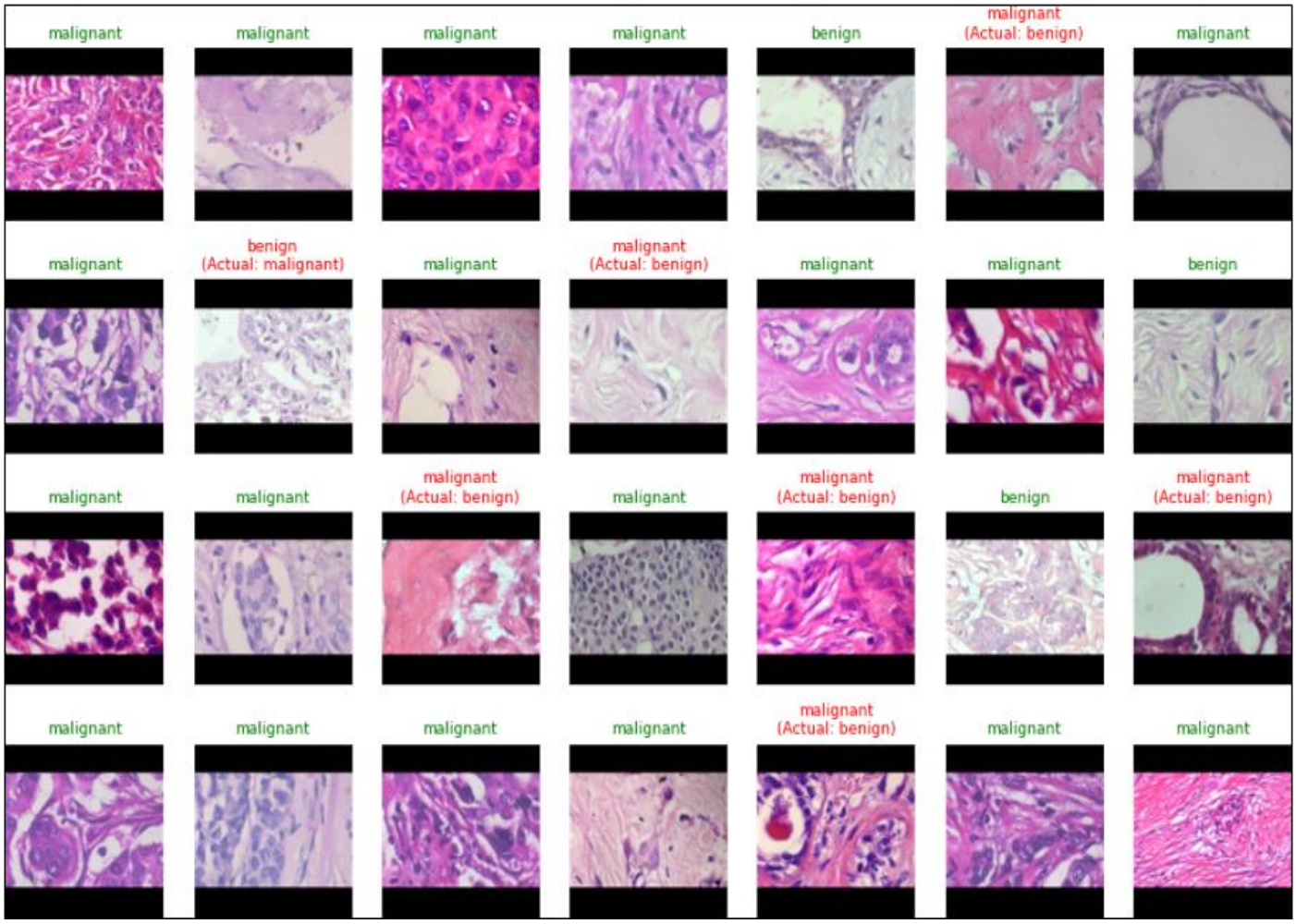

Fig 23 Predictions of the model on some test images

The ResNet50v2 model is the lightest of the five when it comes to parameters and does a pretty solid job overall. It's built with this cool residual learning setup, which lets it pick up on the input's identity mappings without tripping over the vanishing gradient issue. It's a great pick for gigs that need dependable image classification or could use the extra muscle of a deeper, layered network.

F. Model Evaluation

We dug deep into testing out a bunch of deep-learning mod- els for a particular job. The lineup we worked with includes MobileNetV3, EfficientNetB0, VGG16, and ResNet50V2. We put these models through their paces—training them up and then checking how they held up on a test dataset, looking at big-deal measures like loss, ROC-AUC, and accuracy. The table below sums up how each model stacked up on the test data.

Table 10 Performance Metrics on Test Dataset

Model	Loss	ROC-AUC	Accuracy
MobileNetV3	0.4417	0.8541	0.8110
EfficientNetB0	0.3706	0.9080	0.8294
VGG16	0.4482	0.8735	0.7963
ResNet50V2	0.4359	0.8763	0.7982

This table gives you a quick rundown of how the models did, covering stuff like loss, ROC-AUC, and accuracy. We whipped up some visuals to make the training and test results easier to wrap your head around. On the left side of the figure, you've got a look at how the training ROC-AUC and loss played out over the epochs, while the right side has a bar chart stacking up ROC-AUC and loss for each model side by side.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1799

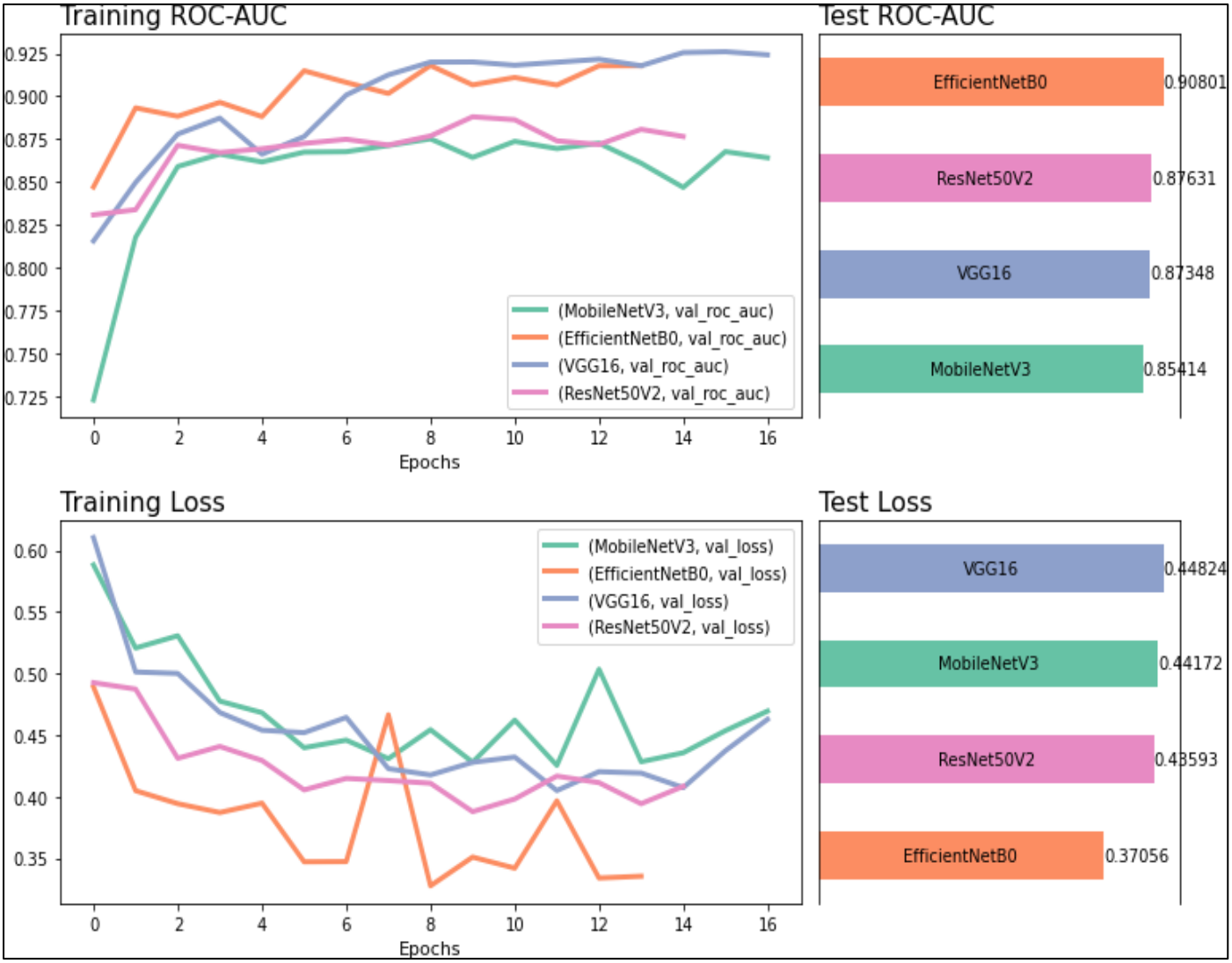

Fig 24 Training and Test Performance Visualization

VI. CONCLUSION

To wrap it up, our research project pulls together blockchain tech, some brainy image-recognition tools called convolutional neural networks (CNNs), and a web-based image uploader to build a rock-solid setup for safe, fast image classification in decentralized healthcare. The heart of our system has a few key pieces: an easy-to-use website where folks can drop their images, the Pinata platform that uses the InterPlanetary File System (IPFS) to keep data locked in tight, and blockchain magic via MetaMask transactions that stamps each image with a unique code—making sure no one can mess with where it came from.

We threw three big-name CNN models into the mix—VGG16, MobileNet, and ResNet—and let them duke it out to see who's best at sorting images. We're looking at stuff like how spot-on they are, how quick they crunch the numbers, and whether they can handle growing bigger. The whole point? Figure out which one's the champ for our dataset, giving us some solid clues about the best way to classify images in healthcare setups that don't lean on a central boss.

We've got a client-server vibe going to keep things smooth between users and the system. The website dishes out real- time results right on the screen, so you get instant feedback on what the images are saying—like a quick diagnosis in your pocket. It's not just about making it user-friendly; it's about speeding up decisions in healthcare when time matters.

Bringing blockchain into the game locks down data security and keeps everything crystal clear with a tamper-proof record of every image's story. This setup tackles the big need for safe, slick image classification in healthcare systems spread out across the map. By mixing high-tech goodies like blockchain and deep learning, we're tossing our hat into the ring of folks figuring out how these worlds collide.

Looking down the road, we're hoping to blaze a trail for decentralized healthcare that leans on cutting-edge tools to nail diagnoses and keep patient info under lock and key. Getting blockchain and CNNs to play nice sets the stage for some game-changing healthcare moves, showing off what this field can really do. We're carving out a path to a future where these systems use smart tech to level up patient care and diagnostic firepower.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1799

Down the line, we could tweak little details like learning speed, batch sizes, training rounds, and callbacks to squeeze even more juice out of the models. To beef up the training data, we might mess around with different optimizers, loss setups, metrics, or tricks like data augmentation. We could also play with pruning, shrinking, or compressing the networks to lighten the load—cutting down on parameters, steps, or memory so the models run leaner and meaner. Plus, we might test-drive some sleeker model designs like SqueezeNet, ShuffleNet, or NASNet to keep things efficient.

ACKNOWLEDGMENT

We want to give a heartfelt thanks to everyone—people and places alike—who chipped in to get this research paper across the finish line. Without their help, pointers, and cheerleading, this whole thing wouldn't have happened.

A massive shoutout goes to our advisors and mentors: Dr. Diwakar Singh (Head of the CSE Department at BUIT), Dr. Amit Jha (Assistant Professor at BUIT), Mr. Madhav Chaturvedi (Professor at BUIT), and Mrs. Neha Lidoriya (Pro- fessor at BUIT). They've been rock-solid with their advice, smarts, and big-picture ideas all through this journey. Their guidance was key in steering us and sharpening our focus.

We can't thank the participants enough—they gave up their time and shared their data with us, no questions asked. What they brought to the table seriously beefed up our dataset and gave us some eye-opening takes on breast cancer.

Big props to the University Institute of Technology Barkat- ullah (BUIT), Bhopal, too. They opened the doors to all the gear and spaces we needed to run our tests and dig into the numbers. That kind of backup made the techy stuff so much smoother.

And we've got to give a nod to all the researchers and writers who've poured their hearts into stuff like the BreaKHis database, blockchain, and machine learning. Their hard work set the stage for everything we've built here.

REFERENCES

[1]. Dheeb Albashish, Rizik Al-Sayyed, Azizi Abdullah, Moham- mad Hashem Ryalat, and Nedaa Ahmad Almansour. Deep cnn model based on vgg16 for breast cancer classification. In 2021 International conference on information technology (ICIT), pages 805–810. IEEE, 2021.

[2]. Yassir Benhammou, Boujemaa Achchab, Francisco Herrera, and Siham Tabik. Breakhis based breast cancer automatic diagnosis using deep learning: Taxonomy, survey and insights. Neurocomputing, 375:9–24, 2020.

[3]. Manuel Cossio. Ethereum, ipfs and neural compression to decentralize and protect patient data in computational pathology. 2022.

[4]. Rafael Jesus de Arau´jo Vasconcelos. Smart contracts: A study about its challenges from a developer experience. 2022.

[5]. Vikas Hassija, Siddharth Batra, Vinay Chamola, Tanmay Anand, Poonam Goyal, Navneet Goyal, and Mohsen Guizani. A blockchain and deep neural networks-based secure framework for enhanced crop protection. Ad Hoc Networks, 119:102537, 2021.

[6]. Nilesh Kumar Jadav, Tejal Rathod, Rajesh Gupta, Sudeep Tanwar, Neeraj Kumar, and Ahmed Alkhayyat. Blockchain and artificial intelligence-empowered smart agriculture framework for maximizing human life expectancy. Computers and Electrical Engineering, 105:108486, 2023.

[7]. Harleen Kaur, M Afshar Alam, Roshan Jameel, Ashish Kumar Mourya, and Victor Chang. A proposed solution and future direction for blockchain-based heterogeneous medicare data in cloud environment. Journal of medical systems, 42:1–11, 2018.

[8]. Abhinav Kumar, Sanjay Kumar Singh, K Lakshmanan, Sonal Saxena, and Sameer Shrivastava. A novel cloud-assisted secure deep feature classification framework for cancer histopathology images. ACM Trans- actions on Internet Technology (TOIT), 21(2):1–22, 2021.

[9]. Abhinav Kumar, Sanjay Kumar Singh, Sonal Saxena, K Lakshmanan, Arun Kumar Sangaiah, Himanshu Chauhan, Sameer Shrivastava, and Raj Kumar Singh. Deep feature learning for histopathological image classification of canine mammary tumors and human breast cancer. Information Sciences, 508:405–421, 2020.

[10]. Lae¨titia Launet, Yuandou Wang, Adria´n Colomer, Jorge Igual, Cris- tian Pulgar´ın-Ospina, Spiros Koulouzis, Riccardo Bianchi, Andre´s Mosquera-Zamudio, Carlos Monteagudo, Valery Naranjo, et al. Fed- erating medical deep learning models from private jupyter notebooks to distributed institutions. Applied Sciences, 13(2):919, 2023.

[11]. Ch VNU Bharathi Murthy, M Lawanya Shri, Seifedine Kadry, and Sangsoon Lim. Blockchain based cloud computing: Architecture and research challenges. IEEE access, 8:205190–205205, 2020.

[12]. Abdulla All Noman, Mustafizur Rahaman, Tahmid Hasan Pranto, and Rashedur M Rahman. Blockchain for medical collaboration: A federated learning-based approach for multi-class respiratory disease classification. Healthcare Analytics, 3:100135, 2023.

[13]. Muhammad Shafay, Raja Wasim Ahmad, Khaled Salah, Ibrar Yaqoob, Raja Jayaraman, and Mohammed Omar. Blockchain for deep learning: review and open challenges. Cluster Computing, 26(1):197–221, 2023.

[14]. AAA Shareef, PL Yannawar, ASH Abdul-Qawy, and MG Almusharref. Share and retrieve images securely using blockchain technology.

[15]. Shubham Thakur and Vijay Kumar Chahar. Storage and verification of medical records using blockchain, decentralized storage, and nfts. In International Conference on Paradigms of Communication, Computing and Data Analytics, pages 753–768. Springer, 2023.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25mar1799

[16]. Md Taufiqul Haque Khan Tusar and Roban Khan Anik. Automated detection of acute lymphoblastic leukemia subtypes from microscopic blood smear images using deep neural networks. arXiv preprint arXiv:2208.08992, 2022.
[17]. Md Taufiqul Haque Khan Tusar and Roban Khan Anik. Automated detection of acute lymphoblastic leukemia subtypes from microscopic blood smear images using deep neural networks. arXiv preprint arXiv:2208.08992, 2022.