CHAPTER 4

CUSTOMIZING CNN ARCHITECTURES FOR CLASSIFICATION OF DR STAGES

In the domain of DL, CNN is considered among the most impactful and frequently applied algorithms. Compared to earlier approaches, a significant advantage of CNNs is their capability to derive meaningful features from data without human intervention. CNNs are extensively utilised across various applications, including CV, speech processing, and face recognition, demonstrating their adaptability and effectiveness [79].

The architectural design of CNNs draws inspiration from the neuronal organisation in the human and animal brains. Notably, CNNs emulate the visual cortex of a cat's brain, where a complex arrangement of neurons is responsible for interpreting visual input. Unlike traditional FC networks, CNNs utilise shared parameters and localised connectivity to effectively exploit the two-dimensional structure of input data, such as images. This strategy considerably lowers parameter count, easing the training process and improving computational efficiency.

Similarly, cells in the visual cortex focus on localised regions rather than the entire scene, extracting spatial correlations. This mechanism is analogous to applying filters over specific areas of an input image, enabling efficient feature extraction. A typical CNN architecture, which resembles the MLP, consists of various convolutional layers followed by subsampling layers, with FC layers making up the final stages of the network.

4.1 CONVOLUTION OPERATION

In a CNN model, the input x for each layer is represented in three dimensions: height, width, and depth, where the height (m) equals the width. Depth corresponds to the number of channels that correspond to specific data attributes. To illustrate, the depth (r) is three in an RGB image, representing the red, green, and blue channels. Each convolutional layer comprises multiple kernels (filters), labeled as (k), that further exhibit three dimensions ($n \times n \times q$), similar to the input. However, the kernel dimensions must satisfy certain constraints: (n) is required to be lower than (m), and (q) is either less than or equal to (r).

Kernels establish the core of local connections and share standard parameters, including a bias term (b^k)and weights (W^k), to produce (k) feature maps (h^k) of size (m−n+1)×(m−n+1). These feature maps are generated by applying convolutional kernels to the input. The convolutional operation calculates the dot product between the input and the kernel weights, as illustrated in Equation 4.1. It is followed by an activation function being applied to the output of the convolution layer, adding nonlinearity and enabling the network to learn complex features.

$$h^k = f(W^k * x + b^k) \tag{4.1}$$

The following step involves down-sampling each feature map within the subsampling layers. This operation decreases the number of network parameters, accelerating training and reducing the likelihood of overfitting. A pooling function, such as max pooling or average pooling, is applied across all feature maps within neighbouring regions of size p×p, where p represents the kernel size.

Following this, FC layers process the extracted mid and low-level features to construct high-level abstractions, representing the network's final-stage layers. These layers then generate classification scores through the output layer, which typically employs methods such as SVM or a softmax function. Each score reflects the likelihood that an input sample is of a particular class.

The advantages are listed as follows: One of the main advantage is it reduce the number of trainable parameters which accelerates the training process and improves the model's ability to generalise to new data, decreasing the chance of overfitting. Sharing of weights between the input components allows the CNNs to learn spatial hierarchies of features, which is extremely appropriate for the image tasks.

Another advantage is the simultaneous learning of feature extraction and classification layers. This double-learning ensures a highly organised model output, with features fine-tuned to be particularly relevant to the classification task. Consequently, CNNs can produce more informative and accurate results, as the features are directly aligned with the classification classes. It is also much easier to implement giant-scale CNN architectures than those of other neural networks. They are modular, and the weights are shared, allowing them to scale adequately even when dealing with huge data. This makes CNNs particularly

useful for tasks requiring significant computational power, such as large image datasets or complex video processing applications.

4.2 CNN LAYERS

A CNN structure has multiple layers, often referred to as core components, each performing a specific role in the processing pipeline. The following section provides a comprehensive overview of each CNN layer's roles and mechanisms.

4.2.1 Convolutional Layer

Historically, before the emergence of DL and CNNs, convolution in image processing is primarily used to extract distinct image features like edges and corners and suppress noise. This is achieved by applying specialised kernels (filters) to the image. For instance, an edge detecting Sobel filter is applied, and a convolution with the image through a sliding window operation is achieved to ensure the overall coverage.

In the context of CNNs, the initial layers are designed to capture fundamental features like lines, edges, and surface textures. As the input advances through the network, the deeper layers extract more abstract and complex representations, such as entire objects or patterns. This layered hierarchy allows the network to use earlier basic features, integrating them to recognise increasingly sophisticated structures in subsequent layers.

A distinctive advantage of modern CNNs is the flexibility in filter dimensions, as filters are two-dimensional (2D) or even three-dimensional (3D), depending on the task. For instance, 3D filters are often employed in video analysis, where adding the temporal dimension adds complexity. One key feature of CNNs is that each filter's elements are considered as learnable network weights, which are updated during training via the back propagation algorithm.

The convolution operation, denoted mathematically by the symbol *, is the heart of CNNs. It is the fundamental mechanism by which CNNs extract spatial hierarchies and relationships within input data. The mathematical formulation of this operation is represented in Equation 4.2, which captures how the filter operates on the input image to generate a corresponding feature map.

$$s(t) = I(t) * K(a) = (I * K)(t) \tag{4.2}$$

Within this framework, s(t) signifies the output feature map, while I(t) represent the original image that is to be convolved with the filter K(a).

4.2.2 Activation Function

Activation functions serve as the crucial "enable" mechanism for neurons, determining whether they should be activated in response to the received input. In the context of CNN, various activation functions are frequently used, each fulfilling a distinct role in influencing the network's output. Commonly applied activation functions include sigmoid, tanh, ReLU, Leaky ReLU, and Randomised ReLU.

Among these, ReLU is particularly prevalent in medical imaging, as evidenced by its wide use according to the previous studies. This activation function has recently become the go-to choice for various DL models, notably those targeting medical image analyses. The ReLU activation function is defined mathematically by the Equation 4.3.

$$f(x) = max(0, x) \tag{4.3}$$

In Equation 4.3, x is the input to a neuron. Other activation functions often used in CNN besides ReLU are sigmoid (provided in Equation 4.4), tanh, and Leaky ReLU. Each of these functions has unique advantages based on the problem or task which the network is tackling.

$$Sigmoid = \frac{1}{1+ e^{-x}} \tag{4.4}$$

4.2.3 Pooling Layer

The pooling layer is important in reducing the network's computational burden by performing the down sampling of the feature representations. This is realised by reducing the image size instead of the channel number. Popular pooling layers in CNNs are maximum pooling, average pooling, and L2-normalised pooling. Among these, max pooling is the most widely utilized technique.

Max-pooling works by taking the maximum of a region on the feature map after convolution. This approach preserves the most important features by eliminating less useful information, thus improving the model's generalization capability and strengthening it to input data variation.

4.2.4 Fully Connected Layer

In the FC layer of a CNN where all neurons correspond to all the previous layer's outputs. This dense connection led to a high computational burden, especially in a complex network. To train for this task, CNNs use Stochastic Gradient Descent (SGD), which enables the network to capture meaningful correlations and features from the training samples.

A major benefit of CNN is that they effectively lower the dimensions of feature maps incrementally during the processing by using convolution and pooling operations. As a result, the most discriminative, high-level features are passed to the FC layer that provides probability scores for each target class. The FC layer is usually the last layer of a CNN model to process the features extracted by the preceding layers. Finally, it maps the input image to one of the previously given classes according to the learned features.

4.2.5 Regularisation in CNN

Overfitting is a significant issue in the domain of CNNs that hinders the models from good generalisation. This happens when a model is performing significantly well on the training data yet it is unable to approach equivalent performance on test data, characteristic of poor generalization to unseen data. This issue often occurs whenever the model does not generalize because of overfitting to training data, learning noise and irrelevant patterns instead of taking meaningful and generalizable characteristics. An underfitted model, in contrast, fails to learn enough patterns on the training data, meaning that it performs poorly on both the training and test datasets.

To ensure the best model performance, the aim is to construct a well-fitted model that performs relatively well on both the training and testing datasets, with a trade-off between underfitting and overfitting.

4.2.6 Dropout

Dropout is a popular technique in DL to improve generalisation. A randomly selected subset of neurons is turned off at every training epoch, meaning these neurons are not used in the forward or backward computations. This approach leads the model to learn multiple and independent features; otherwise, no neuron rely exclusively on others to predict. Dropouts can mitigate overfitting by inducing the feature learning to be distributed

over the entire network. Importantly, the whole network is exploited during testing, and all neurons contribute to the final prediction.

4.2.7 Batch Normalisation

BN is a strategy applied to improve the effectiveness of the output activations in DL models. One of the primary purposes of BN is to eliminate internal covariance shift, which is the phenomenon of changes occurring in the distribution of layer activations as a result of ongoing updates of weights. This problem is more evident in cases where the training data is derived from various sources, for example, image snaps taken in multiple periods of a day. When the internal covariances shift large enough, it causes low convergence, making it take longer for the model to learn effectively. BN tackles this issue by normalising the activations, stabilising the training procedure, and accelerating convergence. Therefore, it would lower the learning time as a whole and improve the learning efficiency.

Advantages of BN

1. **Preventing Vanishing Gradient**: BN keeps the gradient's norm through its propagation via the BN layer.
2. **Improved Weight Initialisation:** The method enhances the weight control initialization, to correct problems induced by poor initial weights.
3. **Faster Convergence:** BN significantly accelerates the network's convergence by promoting stability during training, which proves especially advantageous when handling large-scale datasets.
4. **Reduced Dependency on Hyperparameters:** BN reduces the model's sensitivity to hyperparameter choices, allowing for more efficient training and tuning.

4.3 DR DETECTION AND CLASSIFICATION USING CNN

Conventional diagnostic approaches for the DR stages, that depend on manual observation of retinal images by ophthalmologists, are usually time-consuming and susceptible to human-oriented error and subjective differences.

By applying CNNs to retinal images, subtle and critical features associated with DR, such as MAs, HEMs, and EXs are extracted and those features are the key indicators

for distinguishing between different stages of DR, from mild non-proliferative DR to more advanced stages such as proliferative DR. The integration of CNNs into DR detection enhances diagnostic precision and assures that patients receive timely treatment [80 - 89]. Early detection through automated CNN systems helps prevent the progression of the disease to more severe stages, ultimately improving clinical outcomes and reducing the risk of vision loss. Furthermore, CNN-driven systems are scaled to handle large volumes of retinal images, offering significant potential for population-wide screenings and remote diagnosis, especially in underserved regions.

4.3.1 Need for Analysing Multiple CNN Architectures for DR Classification

The performance of a CNN in DR classification is significantly influenced by its architectural design. The architecture dictates how well the model learns efficiently from the data, accommodate different levels of complexity, and generalise its knowledge to new instances. Different CNN architectures have trade-offs in model complexity, computational requirements, and classification accuracy. Thus, it is important to evaluate and compare other CNN architectures and see how well they perform for the task of DR detection.

4.4 DEVELOPMENT OF CUSTOM CNN ARCHITECTURES

The potential of CNNs in automating the detection and grading of DR has been promising, especially for processing sophisticated medical imagery and images of patterns that are not always readily visible to the naked eye. Their capability to process and learn from large datasets enables the automation of DR screening, resulting in faster, more accurate, and less variable diagnostic outcomes. CNNs significantly reduce the workload on healthcare professionals by quickly identifying the presence and severity of DR, enabling more efficient use of medical resources.

4.4.1 Custom CNN Model_1 Architecture

The Custom CNN Model_1 architecture is a sequential model, one of the simplest neural network architectures available in Keras. In this structure, layers are positioned in a sequential order; every layer's output serves as the input for the next layer**.** It provides a straightforward, easy-to-use and intuitive model development framework. The Custom CNN Model_1 has an architectural design with four FC (dense) layers. The first layer of the model is 224x224x3 (height, width, channels), which is a typical image input shape, in this

case, an RGB image. The detailed Custom CNN Model_1 Architecture is shown in Figure 4.1.

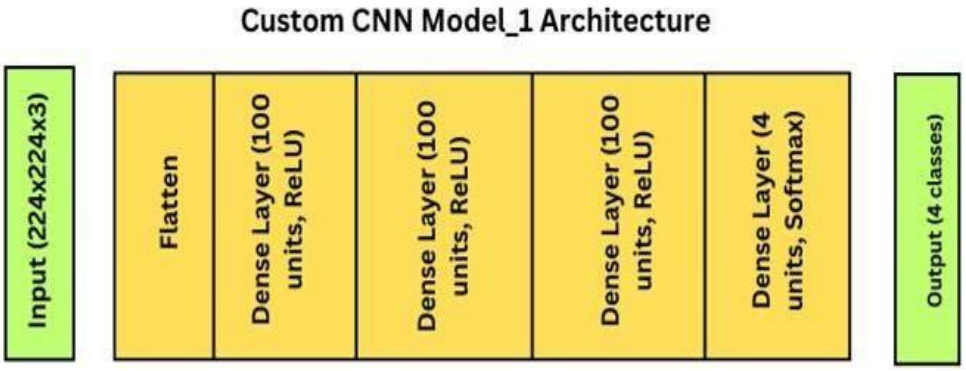

Figure 4.1 Custom CNN Model_1 Architecture

After flattening the input, the model applies four FC Layers, each followed by a ReLU activation function to introduce non-linearity. The final layer outputs a 4-dimensional vector using the softmax function, which is ideal for multiclass classification problems. The network structure of Custom CNN Model_1 Architecture is given in Table 4.1.

Table 4.1 Network Structure of Custom CNN Model_1 Architecture

Layer (Type)	Output Shape	Parameter
Flatten (Flatten)	(None, 150528)	0
Dense (Dense)	(None, 100)	15,052,900
Dense (Dense)	(None, 100)	10,100
Dense (Dense)	(None, 100)	10,100
Dense (Dense)	(None, 4)	404

The first layer is flattened, which is important when transitioning from the image to the dense space. In image processing, input images typically have multiple dimensions: height, width, and colour channels (in this case, 224x224 pixels with three channels for RGB). The flattened layer transforms the 2D feature map into a one-dimensional vector, enabling compatibility with the FC layers. This transformation is important, as thick layers require a 1D input vector rather than a multi-dimensional image array.

Flattening the image involves taking the entire set of pixel values from all three channels (RGB) and putting them into a single vector of size 224x224x3 = 150,528. This vector then becomes the input to the next layer, which performs computations based on the neurons' activations. A l l t he dense layers consists of 100 neurons, connecting all neurons to the preceding layer.

It aims to learn complicated, high-level relationships among the input features. This layer is applied with the ReLU activation function to obtain non-linearity in the model. This non-linear process helps the network learn and better represent information from the data.

This function ensures that negative values in the layer's output are squashed to zero. Restricting to the positive values for ReLU contributes to faster learning and better performance. It helps alleviate the vanishing gradient problem, which is commonly encountered in deep networks with activation functions like sigmoid or tanh. This potentially enables efficient and stable training of deep models.

For example, if a layer of size 100, 100 values get multiplied with each input (input as in the pixel value flattened), then added all together, and a layer of that combined sum is passed through ReLU. For the FC layer, the final output of the last FC layer is a 100-dimensional vector representing the learned features from the prior layers of the network.

The architecture also includes one FC network with 100 hidden units and ReLU activation. Adding the layer adds depth to the model and allows it to acquire more abstract representations of the input data. The extra layers enable hierarchical feature learning, where the initial layers are trained to learn features like edges and colours, and the later ones integrate such features to learn more abstract patterns like shapes, textures, or high-level objects, specifically for image classification.

Deeper models require proper regularisation and optimisation techniques such that they do not overfit. This is when the model memorise the training data rather than learn patterns applicable on new data. Softmax shift raw output scores (logits) to a probability distribution by exponentiating every output and dividing by the total of all the exponentiated outputs. This ensures that the sum of all output values equals one, with each value representing the probability that the input image belongs to a specific class. For example, if the model classifies images of DR stages, each output neuron could represent a specific

stage (e.g., normal, mild, moderate, severe), and the softmax function outputs probabilities for each stage.

The model takes a 224×224 RGB image as input, flattens it into a one-dimensional vector, and feeds it through three dense layers of 100 neurons each. All these layers use the ReLU activation function to add non-linearity so that the model excel in capturing the complex patterns.

4.4.2 Custom CNN Model_2 Architecture

The Custom CNN Model_2 architecture follows a standard CNN design pattern. It begins with convolutional layers to extract spatial features from the input image, followed by a flattening operation to transition to an FC layer. The detailed architecture of the custom CNN Model_2 is illustrated in Figure 4.2.

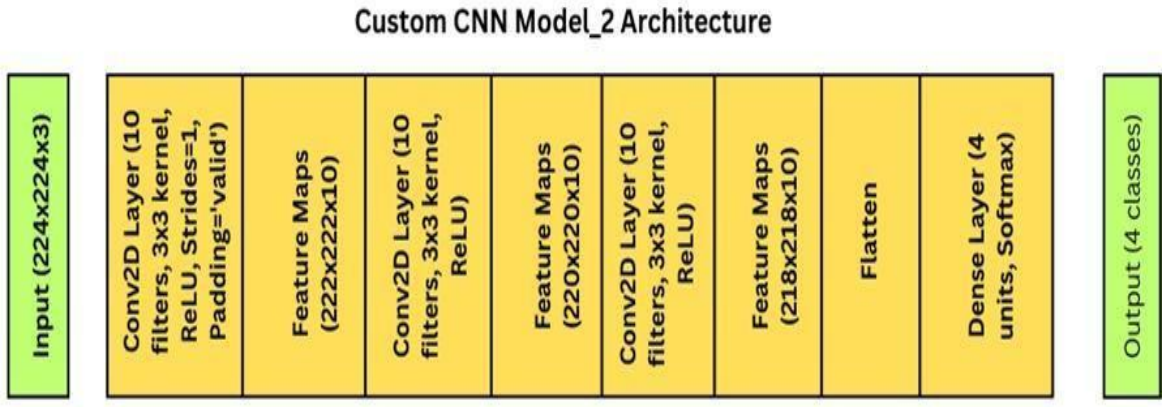

Figure 4.2 Custom CNN Model_2 Architecture

The architecture comprises three convolutional layers with a ReLU activation layer at the end. Following these layers, the output is reshaped and fed to a dense FC layer with four neurons and softmax activation. This is a common setup for an elementary image classification model whose predictions return the probability that the input image falls into each target class. The first layer is a 2D Convolution layer (Conv2D), i.e., convolution filters an image with a kernel.

- input_layer (224, 224, 3): This outlines the shape of the input image. In scenarios where the input is a colour image with a size of 224x224 pixels and 3 colour channels (RGB).

- Filters=10 indicates that the layer uses 10 different filters. These filters convolve with the image to give 10 distinct feature maps, each learning a unique feature of the image.
- kernel_size=3: This specifies the size of the filters. A kernel size 3x3 is commonly used in many architectures, as it captures localised spatial features while being computationally efficient.
- Strides=1 means the stride parameter controls how much the filter moves during convolution. A stride of 1 indicates the filter moves one pixel at a time, leading to an output feature map with nearly the exact size of the input image, depending on the padding used. Table 4.2 details about the architecture of the Custom CNN Model_2. padding='valid':

Table 4.2 Network Structure of Custom CNN Model_2

Layer (Type)	Output Shape	Param #
Conv2D (conv2d)	(None, 222, 222, 10)	280
Conv2D (conv2d_1)	(None, 220, 220, 10)	910
Conv2D (conv2d_2)	(None, 218, 218, 10)	910
Flatten (flatten_2)	(None, 475240)	0
Dense (dense_10)	(None, 4)	1,900,964

- Padding refers to adding pixels around the input image prior applying the convolution operation. Valid' means no padding is applied, resulting in a smaller output feature map relative to the input.
- activation='relu': Applies ReLU activation to the output of the convolution. ReLU is utilized to incorporate non-linearity into the model and also enables the model to acquire more advanced patterns.

This implies the negative output values are zeroed while the positive values are retained.

The number of filters used in each layer defines the number of features or patterns the model learns. By applying a series of convolutional layers, the model learns hierarchical

patterns: the first layer might detects basic patterns like edges, while the subsequent layers learn more complex patterns like textures or shapes.

Since the number of filters remains constant across all convolutional layers (10 filters), each layer produces an output with the same depth. The width and height of the output feature maps diminish depending on the padding and stride settings. With valid padding and a stride of 1, the output height and width are smaller than the input, but not by a large margin.

Thereafter the convolutional layers, the output is a multi-dimensional array (feature maps), but the dense layer requires a 1D vector as input. To achieve this, the flatten layer is used. The flatten layer reshapes the multi-dimensional array into a 1D vector, which is passed to the FC Layers (dense layers).

For example, if the output from the last convolutional layer has a shape of (224, 224, 10), the flatten operation would reshape this into a vector of size 224 * 224 * 10 = 501,760, where each value corresponds to a feature learned by the model.

The final layer is a dense layer, following flattening, the output is forwarded to an FC Layer with 4 neurons. The 4 neurons correspond to the 4 possible classes in the classification task. Since the task is a multiclass classification problem, the softmax activation function is applied at the output layer.

In summary, this CNN model is designed to classify input images sized at 224x224 pixels with 3 colour channels (RGB) into one of 4 possible classes. The model structured with three convolutional layers that use 10 filters of size 3x3 with ReLU activation. These convolutional layers are followed by a flatten operation to shift the feature maps into a 1D vector, passing through a dense layer with 4 neurons and softmax activation to output the final class probabilities.

4.4.3 Custom CNN Model_3 Architecture

The custom CNN Model_3 is designed so that feature extraction from the input image is hierarchical followed by a decision-making process within the dense layers. This model uses convolutional and pooling layers, typical for any CNN created for classifying images from the extracted features.

Figure 4.3 shows the detailed Custom CNN Model_3 Architecture. This model is intended for classifying input images into 4 classes. It starts with convolutional layers that capture the key features and is then proceeding to max-pooling layers to diminish the spatial dimension, which reduces computation in the subsequent layers.

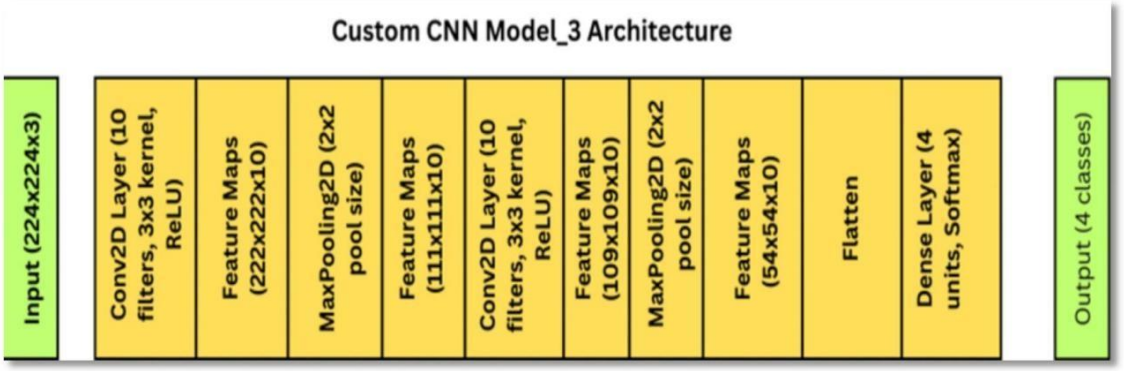

Figure 4.3 Custom CNN Model_3 Architecture

The output is then flattened and fed into a dense layer with 4 neurons, activated with a softmax activation layer. The network architecture for Custom CNN Model_3 Architecture is presented in Table 4.3.

Table 4.3 Network Structure of Custom CNN Model_3 Architecture

Layer (Type)	Output Shape	Param #
Conv2D (conv2d)	(None, 222, 222, 10)	280
Maxpooling2d_2	(None, 111, 111, 10)	0
Conv2D (conv2d_1)	(None, 109, 109, 10)	910
Maxpooling2d_3	(None, 54, 54, 10)	0
Flatten (flatten_2)	(None, 29160)	0
Dense (dense_10)	(None, 4)	116,644

The first layer is a 2D convolutional (Conv2D) that convolves the input image to learn spatial features. The convolutional operation applies the filters to the input image, producing feature maps. The shape of the output feature map relies on the kernel's size and the padding used (which is implicitly 'valid' in this case, meaning no padding is applied).

The input layer, filter, kernel size, and activation function in Model_3 are identical to those in the Custom CNN Model_2, with the addition of a max-pooling layer as the new component in Model_3 after the convolutional layer.

Pooling layers decrease the spatial dimensions (height and width) of the feature maps, which aids reduce the computation in the network and prevents overfitting by summarising important features. In max-pooling, a filter (often of size 2x2) moves across the feature map, selecting the maximum value in the area covered by the filter. This results in a down-sampled feature map with decreased spatial dimensions. For example, if the input feature map is 224x224, max-pooling with a 2x2 filter and a stride of 2 reduces the spatial dimensions to 112x112.

The second convolutional layer is like the first one. This layer extracts more complex and abstract features from the image by applying 10 filters to the output of the first max-pooling layer. The feature maps produced by this layer will likely capture more intricate patterns and structures in the image. The second max-pooling layer is applied after the second convolutional layer. Similar to the first max-pooling layer, this operation decreases the spatial dimensions of the feature maps, making the model more computationally efficient and abstracting important patterns.

After the convolutional and pooling layers, the output is a multidimensional array (feature maps). To feed this data into an FC layer, the feature maps must be flattened into a 1D vector, which is accomplished using the flatten layer. For example, if the output from the last max-pooling layer is a feature map of shape (56, 56, 10), the flatten layer converts this into a vector of size 56 * 56 * 10 = 31,360. Atlast the output layer which is a dense layer that takes the flattened vector and outputs the classification probabilities for each class.

4.4.4 Custom CNN Model_4 Architecture

The Model_4 architecture is a DCNN designed for image classification, specifically for categorizing images into four distinct classes. The model follows a hierarchical feature extraction process, where multiple convolutional layers progressively learn low-to-high-level features. Each convolutional block is followed by BN to stabilize training, and max-pooling layers to mitigate spatial dimensions while retaining important information.

The model utilizes mixed precision training for optimized computation and memory efficiency, assuring a balance between accuracy and performance. Additionally, the dense layers are explicitly set to float32 precision to prevent numerical instability. The Custom CNN Model_4 Architecture is given in Figure 4.4.

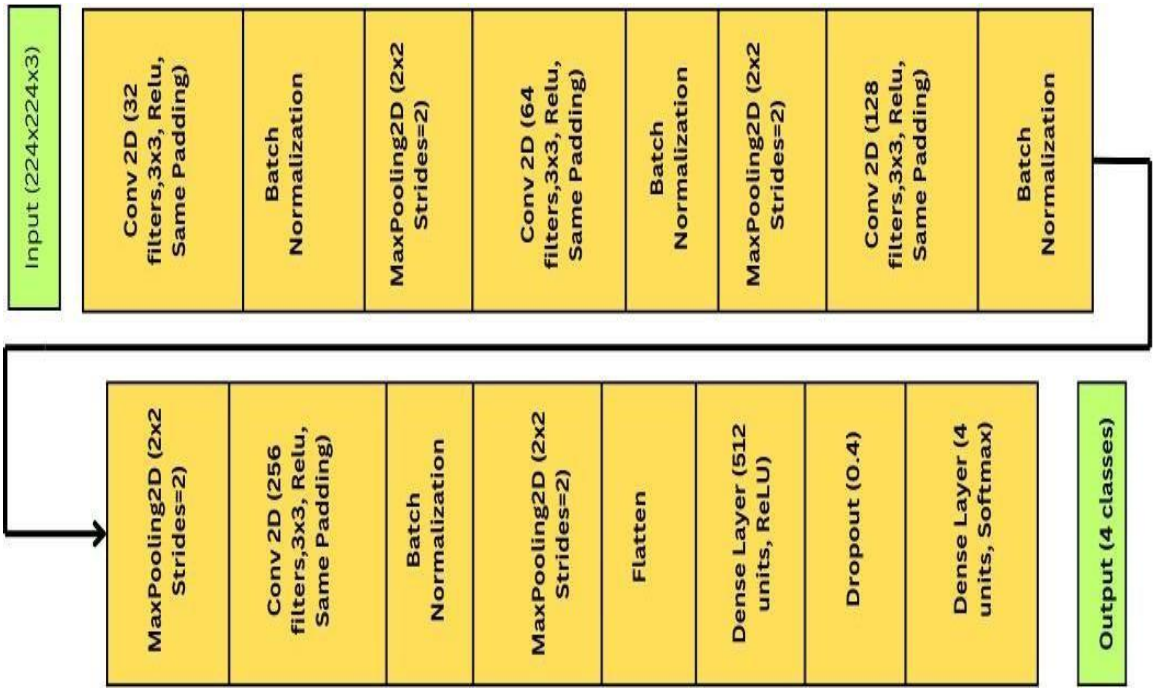

Figure 4.4 Custom CNN Model_4 Architecture

The architecture starts with a Conv2D layer that applies 32 filters of size 3×3, using ReLU activation to introduce non-linearity. The padding is set to 'same', assuring the spatial dimensions remain unchanged. BN is used to normalise activations, accelerate convergence and improve stability.

A max-pooling layer (2×2, stride 2) then downsamples the feature map from 224×224 to 112×112, which is computationally efficient. The second convolutional block follows the same structure but increases the number of filters to 64, allowing the model to detect more complex features. Another max-pooling layer further reduces the feature map size to 56×56. The network structure of the Custom CNN Model_4 Architecture is given in Table 4.4.

Table 4.4 Network Structure of Custom CNN Model_4 Architecture

Layer (Type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 224, 224, 32)	896
batch_normalization (BatchNormalization)	(None, 224, 224, 32)	128
max_pooling2d_2 (MaxPooling2D)	(None, 112, 112, 32)	0
conv2d_6 (Conv2D)	(None, 112, 112, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 112, 112, 64)	256
max_pooling2d_3 (MaxPooling2D)	(None, 56, 56, 64)	0
conv2d_7 (Conv2D)	(None, 56, 56, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 56, 56, 128)	512
max_pooling2d_4 (MaxPooling2D)	(None, 28, 28, 128)	0
conv2d_8 (Conv2D)	(None, 28, 28, 256)	295,168
batch_normalization_3 (BatchNormalization)	(None, 28, 28, 256)	1,024
max_pooling2d_5 (MaxPooling2D)	(None, 14, 14, 256)	0
flatten_4 (Flatten)	(None, 50176)	0
dense_12 (Dense)	(None, 512)	25,690,624
dropout_1 (Dropout)	(None, 512)	0
dense_13 (Dense)	(None, 4)	2,052

The third convolutional block enhances the feature extraction by using 128 filters, searching for more complex patterns and object shapes. BN adds stability, and max-pooling decreases the spatial dimension to 28×28.

The fourth convolutional block follows a similar pattern but increases the filter count to 256, enabling the model to learn even finer details. A final (7×7) max pooling operation shrinks the feature map to 14×14 to have a more compact network and to preserve important features. These convolutional and pooling layers are the feature extraction backbone of the model, learning increasingly complex representations that aid in differentiating between classes.

The feature maps are passed through the FC layers after the convolutional and pooling layers. The 14×14×256 feature maps are flattened into a 1D vector as input to the classification part of the network. A dense layer with 150 nodes and ReLU activation allows the model to capture complex feature interactions.

A Dropout (0.4) layer is included to avoid overfitting by forcing 40% of the neurons to turn off randomly. Then, a dense layer with 4 neurons at the output end, with a softmax activation function. This model utilise CNN fundamentals effectively using multiple convolutional layers for feature extraction, subsequently FC Layers for classification. The incorporation of BN and dropout improves stability and generalisation.

The application of mixed precision training enhances performance, with clear float32 casting of dense layers ensuring accuracy in computations. In summary, Model_4 is an organised DCNN for image classification, providing a tradeoff between complexity, accuracy and efficiency.

4.4.5 Custom CNN Model_5 Architecture

The Custom CNN Model_5 is a DL architecture constructed with separable convolutional layers to provide computational efficiency with powerful feature extraction capabilities. It is reported that the separable convolution is popular in the leading-edge models such as MobileNet, Xception and EfficientNet and has a remarkable performance in terms of computational cost of the model. Lower Floating Point Operations (FLOPs) model learns faster, has less resource overhead, and is favourable for scaling to practical real-world applications in medical image analysis, autonomous systems, and mobile-based AI models.

The architecture of Model 5 assures an optimal balance between performance and efficiency. Separable convolutions significantly reduce computational complexity by breaking down standard convolution into depthwise and pointwise convolutions. Unlike conventional convolutions, which process spatial and depth information simultaneously, separable convolutions first apply a depthwise convolution (filtering input channels independently) and then a pointwise convolution (combining depthwise outputs using 1×1 convolutions). This separation drastically lowers the number of required operations while preserving important feature representations. As a result, the model requires fewer parameters, reducing memory footprint and training time without compromising accuracy. BN after each convolutional layer stabilises training by normalising activations, ensuring the network converges faster.

Furthermore, Model 5 guarantees quicker inference times and is more suitable for applications needing real-time processing. In general, this architecture balances accuracy and efficiency, and it is also a good choice in image classification and domains such as medical images, which require efficient and reliable models.

Advantages of Separable Convolution:

Parameter Efficiency: Compared to classical Conv2D layers, separableConv2D decreases the number of parameters noticeably. The model is lightweight and easy to train, but it maintains high classification accuracy.

Computational Efficiency: Separable convolutions factorize the convolution into two steps, depthwise and pointwise convolution, leading to a reduced number of operations and, therefore, a gain in training and inference time.

Less Overfitting: A less complex model is less sensitive to the data it is trained on, so it may perform better on an independent test set.

Better Feature Extraction: Separable convolutions can also efficiently represent spatial hierarchies in an image, while preserving crucial spatial configuration.

Easier for Deployment: Smaller models with increased average deployment can be achieved with lower computational throughput deployed at the edge or in real-time, for example, in medical imaging and mobile AI.

The detailed Custom CNN Model_5 Architecture is given in Figure 4.5. The Custom CNN Model_5 has four convolution blocks, which are made up of a SeparableConv2D layer, BN, and MaxPooling2D. These layers cooperate in calculating spatial and channel-wise features and ensure the stability of the training. The GAP is trying to minimise the number of parameters - specifically, the pin count - concerning an FC Layer, so it generalises better and does not overfit.

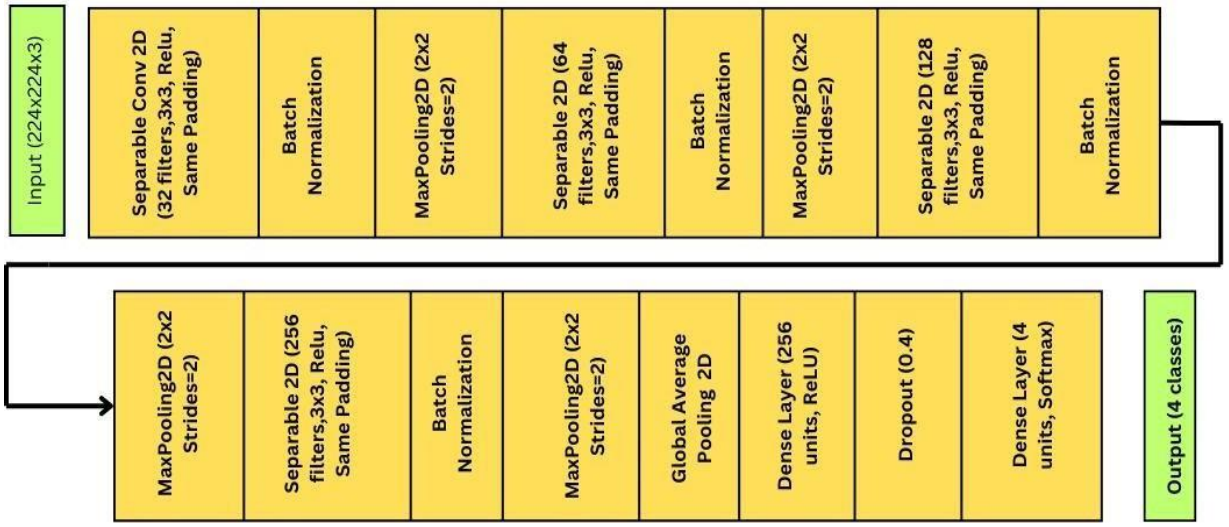

Figure 4.5 Custom CNN Model_5 Architecture

The main strength of GAP is its capability to replace large, FC Layers, reducing the risk of overfitting. Unlike flatten layers, which generate high-dimensional feature vectors, GAP computes the mean of feature maps, leading to a more compact representation and fewer trainable parameters.

Finally, two Dense layers with 256 ReLU neurons and 4 softmax neurons are applied, serving the deep feature learning and multi-class classification stages. Dropout (with 40% probability) provides some regularisation to further lower overfitting. Mixed precision training is also enabled, assuring better memory utilisation and faster training without sacrificing accuracy. Importantly, dense layers are explicitly set to float32 to avoid precision loss during training. The network structure of the Custom CNN Model_5 Architecture is given in Table 4.5.

Table 4.5 Network Structure of Custom CNN Model_5 Architecture

Layer (Type)	Output Shape	Parameters
SeparableConv2D (Conv Layer)	(None, 224, 224, 32)	155
BatchNormalization	(None, 224, 224, 32)	128
MaxPooling2D	(None, 112, 112, 32)	0
SeparableConv2D	(None, 112, 112, 64)	2,400
BatchNormalization	(None, 112, 112, 64)	256
MaxPooling2D	(None, 56, 56, 64)	0
SeparableConv2D	(None, 56, 56, 128)	8,896
BatchNormalization	(None, 56, 56, 128)	512
MaxPooling2D	(None, 28, 28, 128)	0
SeparableConv2D	(None, 28, 28, 256)	34,176
BatchNormalization	(None, 28, 28, 256)	1,024
MaxPooling2D	(None, 14, 14, 256)	0
GlobalAveragePooling2D	(None, 256)	0
Dense (FC Layer)	(None, 256)	65,792
Dropout (Regularisation)	(None, 256)	0
Dense (Output Layer - Softmax)	(None, 4)	1,028

This makes the model lighter and suitable for edge-device deployment with limited computational resources. The dense layer's dropout (0.4 rate) mitigates overfitting by randomly turning off neurons during training, ensuring the model generalises well to unseen data.

Additionally, using separable convolution, the number of trainable parameters in Model 5 is significantly reduced, with 113,407 parameters, compared to Model 4, which has 26,082,052 trainable parameters.

4.4.6 Structural Comparison of Custom CNN Architectures

A detailed comparison of the custom CNN network structures is provided in the table 4.6:

Table 4.6 Network Structure of Custom CNN Model Architectures

Layers	**Custom CNN Model_1**	**Custom CNN Model_2**	**Custom CNN Model_3**	**Custom CNN Model_4**	**Custom CNN Model_5**
Conv Layer	Nil	3	2	4	Nil
Separable Conv Layer	Nil	Nil	Nil	Nil	4
Batch Normalization	Nil	Nil	Nil	4	4
Max pooling	Nil	Nil	Nil	2	4
GAP/Flatten	1	1	1	1	1
Dense Layer (FC Layer)	4	1	1	1	2
Dropout	Nil	Nil	Nil	1	1

4.4.7 Inception-ResNet-v2 and custom CNN Model

A TL strategy employs the Inception-ResNet-v2 model, originally trained on the ImageNet dataset, with its final classification layers removed [90]. Additional layers are integrated to construct a hybrid DL architecture, as illustrated in Figure 4.6. The resulting framework consists of two principal components: a pre-trained Inception-ResNet-v2 backbone and a custom convolutional module, followed by FC layers. The custom module incorporates a naive inception-inspired block composed of four parallel convolutional layers with distinct filter sizes, each accompanied by batch normalization and dropout. Outputs from these parallel layers are concatenated and transformed into a one-dimensional representation through a GAP operation, which is subsequently directed to FC layers and a softmax classifier.

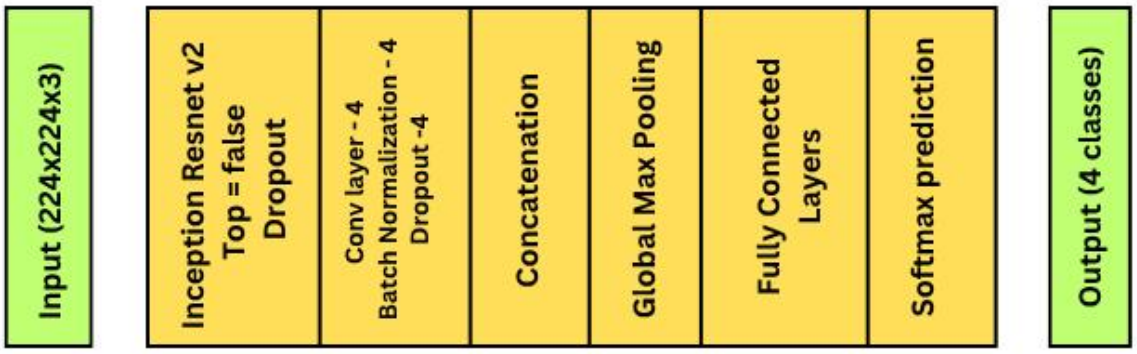

Figure 4.6 Inception-ResNet-v2 and custom CNN Architecture

The input image undergoes preprocessing before being propagated through the modified Inception-ResNet-v2 backbone. The custom block, positioned after the truncated backbone, utilizes four convolutional paths with differing filter sizes to eliminate the need for filter-specific tuning when adapting to new datasets.

This ensemble-style design ensures that diverse spatial features are captured without architectural adjustments. The concatenated feature maps are forwarded to succeeding layers for high-level representation learning, culminating in a softmax activation function that determines the four class labels.

4.4.8 Custom CNN Model

The network takes an input retinal image of size 448×448×3 and progressively extracts hierarchical features through a sequence of convolutional and pooling layers [91].

Figure 4.7 presents the detailed architecture of the existing CNN designed for DR prediction. The initial convolutional layers with 32 filters of size 4×4 capture low-level edge and texture information, followed by max-pooling operations to reduce spatial dimensions and computational complexity.

Subsequent convolutional blocks with 64, 128, 256, and 512 filters progressively learn more abstract and discriminative representations, allowing the network to effectively detect subtle retinal abnormalities.

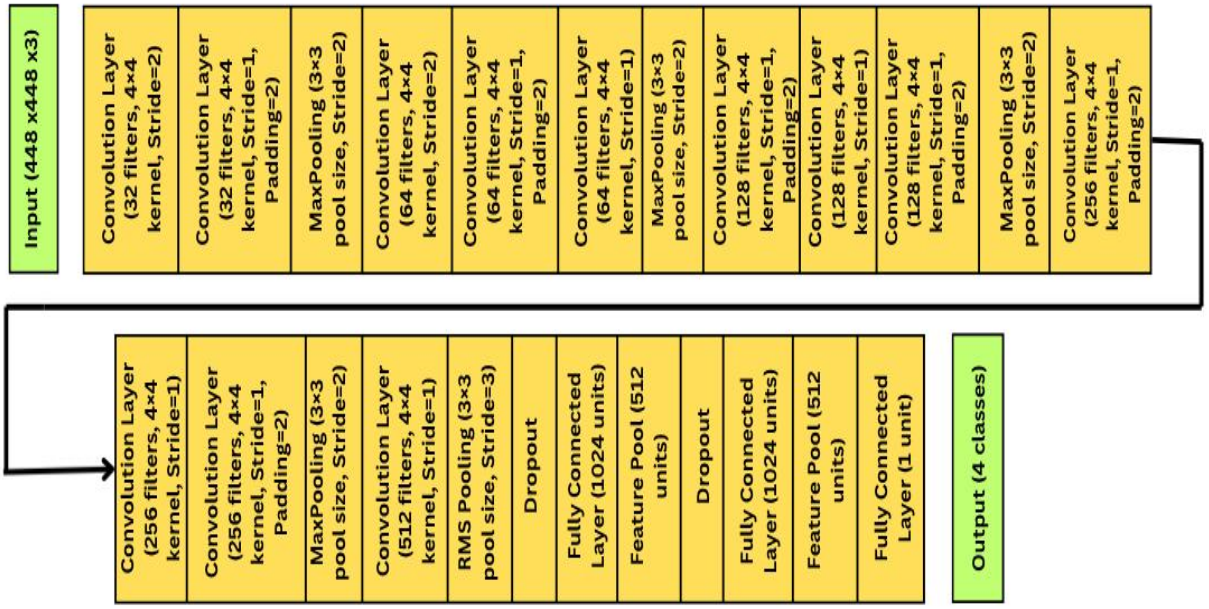

Figure 4.7 Custom CNN Model

Each convolutional stage uses a combination of strides and padding to control the output resolution, as shown in the Table 4.7. RMS pooling is applied after the final convolutional block to enhance robustness and reduce sensitivity to noise, while dropout layers are incorporated to prevent overfitting by randomly deactivating neurons during training.

The extracted features are then passed through fully connected layers (1024 and 512 neurons) that perform high-level feature integration. Finally, the output layer generates the classification score corresponding to the DR grade.

This hierarchical architecture enables the model to efficiently balance feature richness and computational efficiency, leading to improved performance in DR severity classification tasks.

4.5 DATASET

The APTOS 2019 Blindness Detection dataset contains 3,367 high-resolution coloured retinal fundus images divided into four severity levels: Mild, Moderate, Normal, and Severe DR. This ensures that an accurate assessment of the proposed DL models can be carried out for the potential diagnosis of DR.

The dataset is segmented into Training (70%), Validation (20%), and Testing (10%) as part of the robustness measure for evaluating the DL models for DR. Table 4.7 illustrates the split of images in the three sets—training (70%), Validation (20%), and Testing (10%).

The data set is divided into four levels of DR, i.e., Mild, Moderate, None, and Severe, ensuring balanced sampling for model training and testing. This well-organised split

evenly distributes the degrees of DR severity in training, validation, and test sets so that an efficient and generalizable DL model can be learned.

Table 4.7 Dataset Distribution across Training, Validation, and Test Set

Split	Total Images	Mild	Moderate	Normal	Severe
Training	2,356 (70%)	259	699	1,263	135
Validation	672 (20%)	74	199	361	38
Testing	339 (10%)	37	101	181	20
Total	3,367 (100%)	370	999	1,805	193

4.6 DATA AUGMENTATION (OVERSAMPLING)

Data augmentation plays a significant role in DL pipelines, especially image classification. It generates a synthetic increase of the training set with many transformations, thus enhancing the robustness of the model to variations in input images.

The methods of augmentation applied, rotation, zoom, and flipping, strengthen it and lower the potential for overfitting. Proper augmentation improves model accuracy and prevents overfitting, making the predictions more reliable. Additionally, normalisation confirms that pixel values are scaled appropriately for stable training.

One of the key transformations used is rotation. Rotation involves transforming an image by rotating it around its center by a specified angle θ. The transformation matrix for rotation is given in Equation 4.5.

$$M = \begin{bmatrix} \cos\theta & -\sin\theta & x \\ \sin\theta & \cos\theta & y \end{bmatrix} \quad (4.5)$$

For θ = 15°, the transformation applied is given in Equation 4.6:

$$M = \begin{bmatrix} cos(15°) & -sin(15°) & 0 \\ sin\,(15°) & cos(15°) & 0 \end{bmatrix} \quad (4.6)$$

This rotates the image by shifting pixels accordingly, helping the model recognise patterns in different orientations. The transform matrix also ensures the structure of the image is maintained when adjusting the pixels. By incorporating rotational invariance, the model learns features that are invariant under different orientations, improving the robustness as a result.

Another augmentation technique applied is flipping, which mirrors the image along a specified axis. The transformation formula is given in Equation 4.7:

$$I'=flip(I,flipCode) \quad (4.7)$$

where, flipCode = 1 performs a horizontal flip (left-right inversion).

Flipping is a kind of data augmentation that adds symmetrical variety, making models invariant to the images' left/right orientation. Zoom is one of the important augmentations performed in the dataset.

It cuts out the middle and scales it to the size of the original image. The procedure is expressed mathematically by Equations 4.8 and 4.9:

$$I'=resize(I[y1: y1+ h', x1 : x1+ w'],(w,h)) \quad (4.8)$$

$$w' = \frac{w}{zoom\ factor} \quad h' = \frac{h}{zoom\ factor'} \quad (4.9)$$

where, h' and w' are the new height and width after zooming. $y1,x1$ determine the starting point for cropping the zoomed region. Given a zoom factor of 1.3, the cropped size is given in Equation 4.10

$$h' = \frac{h}{1.3} \quad w' = \frac{w}{1.3} \quad (4.10)$$

The cropped image is then resized to (h, w) to maintain consistency. This forces the model to focus on finer details, helping it generalise to objects of varying sizes. Figure 4.8 shows the sample image after applying data augmentation techniques.

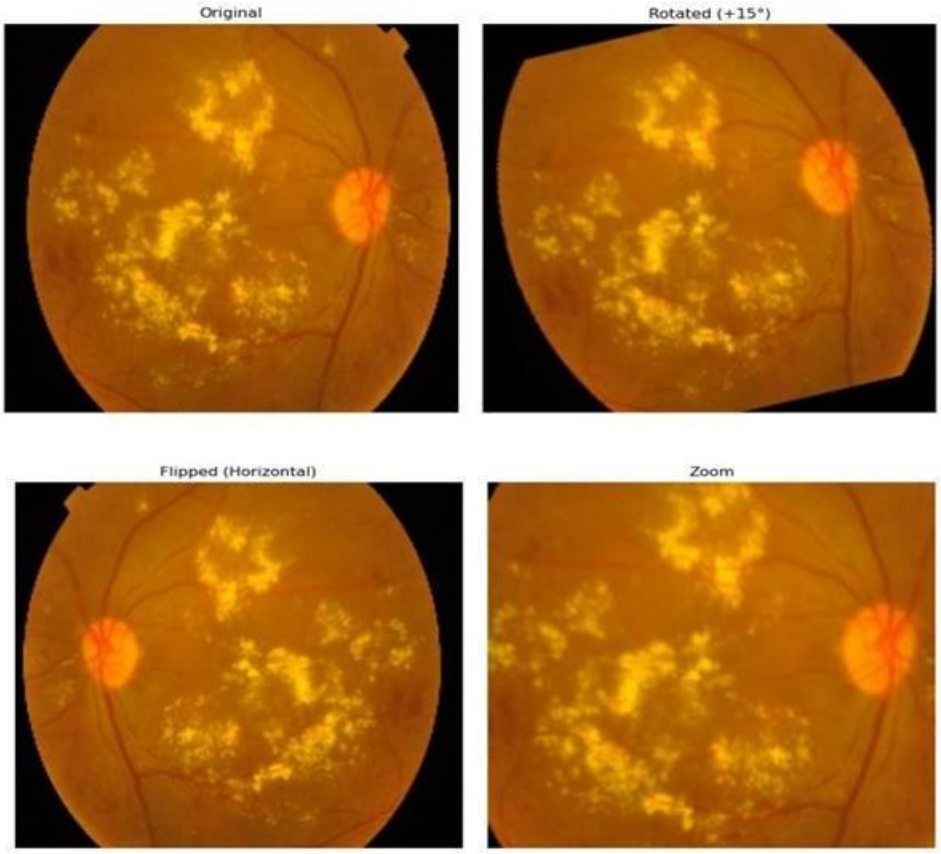

Figure 4.8 Fundus image with Data Augmentation Techniques

.

Zooming augmentation assures that even small features within an image contribute to the learning process, making the model more versatile in detecting patterns at different scalesUsing these augmentation techniques enriches the dataset with strong variations, resisting overfitting as well as improving the model's generalisation ability. Such conversions are of critical importance to guarantee that DL models are best able to operate on real-world data, thereby making them more reliable for real-world applications.

After applying augmentation, the class distribution becomes balanced, as shown in Table 4.8.

Table 4.8 Number of Training Images after Augmentation for each DR Stage

DR Stages	Number of Training Images After Augmentation
Normal	1263
Mild	1263
Moderate	1263
Severe	1263

4.7 FINAL NORMALIZATION & DATA PREPARATION

Normalisation is applied to standardise pixel values before giving the images to a DL model. In most cases, pixel values range from 0 to 255, and normalisation scales them to the interval [0,1] which enhances numerical stability while training the model. The formula for normalisation is given in Equation 4.11.

$$Inorm = \frac{I}{255} \tag{4.11}$$

where, I is the original pixel intensity (0 to 255), and Inorm is the normalised pixel value (0 to 1).

After normalisation, images are reshaped to a specific dimension (e.g., 224×224 pixels) to ensure consistency across the dataset.

The final dataset is ready for model training with balanced classes and standardised pixel values.

Proper image preprocessing enhances model performance by assuring a well-structured, balanced, and clean dataset.

The steps outlined above, splitting, corrupt image removal, class balancing, augmentation, and normalisation, contribute to a more robust and efficient DL pipeline.

After applying data augmentation, the balanced dataset looks like the one in Table 4.9.

Table 4.9 Balanced dataset

Split	Total Images	Mild	Moderate	Normal	Severe	Total
Training	2,356 (70%)	1,263	1,263	1,263	1,263	5052
Validation	672 (20%)	74	199	361	38	672
Testing	339 (10%)	37	101	181	20	339
Total	3,367 (100%)	1374	1563	1,805	1321	6063

4.8 HYPERPARAMETER CONFIGURATION FOR CUSTOM CNN MODELS

After preprocessing techniques, the hyperparameter setup for CNN Models aims to enhance the understanding of how various design choices influence the effectiveness of CNNs in detecting and classifying different stages of DR.

The custom CNN models are trained using a batch size of 8, ensuring a balanced compromise between computational efficiency and model convergence. The input image size is set to 224×224, a standard resolution that preserves sufficient detail for feature extraction while maintaining a manageable computational load. This image size is widely used in DL applications because it permits the model to extract fine grained patterns within the data.

The Adam optimiser is employed with a learning rate of 0.0001 for optimisation.

Adam is a widely preferred optimiser due to its capacity for learning adaptation during training, which results in quicker convergence as well as better generalisation. The loss function used is sparse categorical cross-entropy, particularly suitable for DR classification tasks where class labels are provided as integers rather than one-hot encoded vectors. This loss function ensures efficient gradient updates, allowing the model to accurately differentiate among classes.

4.9 RESULTS

All the five models are trained over 10 epochs, which ensures they have enough iterations to extract meaningful patterns from the dataset without overtraining and overfitting the dataset. The hyperparameters are consistently used across all custom CNN models to maintain uniform training conditions and enable fair performance comparisons.

Figure 4.9 shows the performance of five different custom CNN architectures across ten epochs. The key metric for performance evaluation is the training loss, which quantifies how effectively the model predicts the target values. A lower loss indicates better model performance.

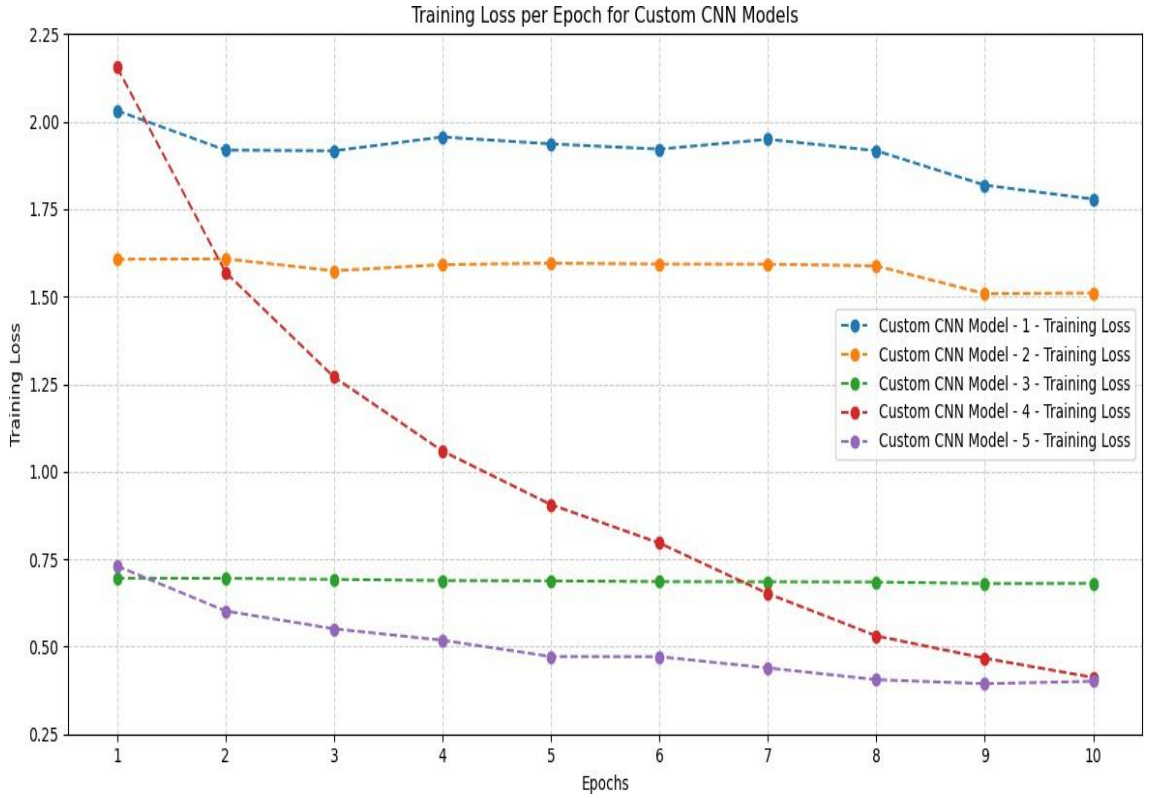

Figure 4.9 Training loss of Five Custom CNN Models

Model_1 starts with a high loss of 2.0313, indicating initial difficulties in learning. Over the epochs, its loss fluctuates slightly but follows a decreasing trend. By the 5th epoch, it reduces to 1.9363, showing gradual improvement. The model continues refining its parameters as training progresses, reaching 1.8185 in epoch nine and dropping to 1.7786 in the final epoch. While the reduction is steady, the model still maintains a relatively high loss compared to others, suggesting room for further optimisation.

Model_2, the loss is initially 1.6067, which stays somewhat consistent as the model trains. Its loss does not drop to extreme values like the other models; instead, it shows a moderate decline, reaching 1.5738 after the third epoch. The model is still slowly improving and attains 1.5084 at the 9^{th} epoch and 1.5104 at the last epoch. Although it denotes a stable training, there is a slight drop, which indicates that the model is converging more slowly than others.

Model_3 has the smallest loss of all the models throughout, and starts from 0.6957, declining with a bit of variability. Meaning: The value is already solid at 0.3929 (By the 4th epoch, it chases 0.6890, that too for a couple of epochs, showing strong stability in learning). The model fluctuates very little and has fully converged around 0.6847 in epoch 8. Its loss at the last epoch is 0.6811, indicating that Model_3 converges efficiently without significant fluctuations. This suggests that the model should generalise well and possibly need less fine-tuning in training.

Model_4 begins with the worst loss of 2.1555; it is a difficult beginning. It decreases considerably, down to 1.5694 in epoch 2 and falling even further to 1.0588 at epoch 4. The model keeps learning relatively fast, with 0.7960 at epoch 6. It attains a loss of 0.4120 at the last epoch, one of the most significant decreases in all the models. This indicates that Model_4 can learn efficiently and generalise well across the training.

Model_5 begins with a 0.7304 loss value; it is less than other models. It decreases monotonically to 0.4718 by epoch 5, and the model keeps improving, to 0.4392 in epoch 7 and 0.3941 in epoch 9. Model_5 has the best final loss, equal to 0.4011 at the last epoch. It demonstrates strong convergence and learning capability.

Among all models, Model_3 and Model_5 achieve the lowest final losses, indicating strong learning capabilities. Model_4 demonstrates the most significant improvement,

suggesting a high adaptability during training. Model_1 and Model_2, though improving, maintain relatively higher losses, indicating potential areas for optimisation.

Validation Loss is an important step applied to quantify ML model behavior during training. It is estimated by making predictions from the model against another validation data set that isn't used for training. While training loss defines how effectively the model performs on the training data, validation loss is applied to measure if the model can generalize new data. The performance of five different Custom CNN architectures is evaluated based on the validation loss across ten epochs, as shown in Figure 4.10.

Model_1 exhibits high validation loss throughout training. It starts at 2.2556 and peaks at 2.4182 in epoch 7, indicating instability in generalisation. Although the loss slightly decreases towards the final epoch, settling at 2.3366, it remains significantly higher than other models. This suggests that Model_1 struggles to learn effectively and may be overfitting.

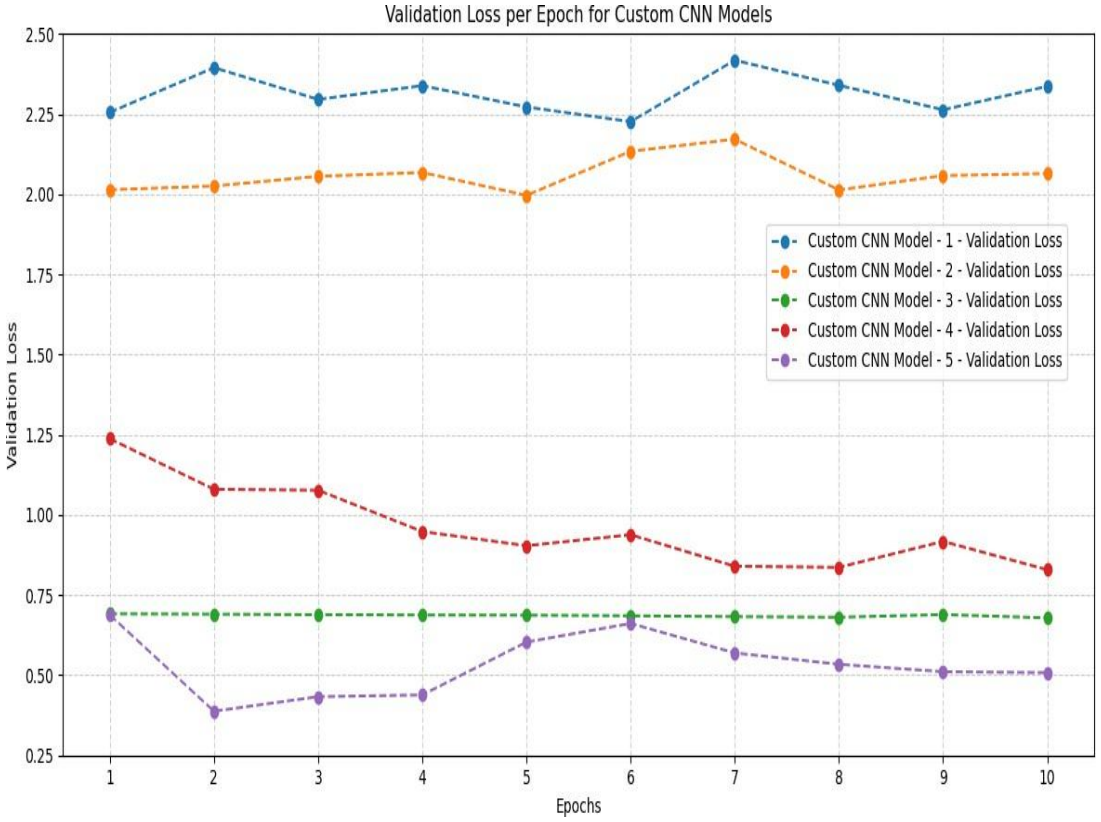

Figure 4.10 Validation Loss of Five Custom CNN models

Model_2 starts with a validation loss of 2.0142 and fluctuates over the epochs. The loss increases to 2.1717 in epoch 7 but stabilises slightly towards the final epoch at 2.0645. Despite maintaining a lower loss than Model_1, Model_2 does not show consistent improvement, indicating that further optimisation is necessary.

Model_3 achieves the lowest validation loss among most models, starting at 0.6915 and gradually decreasing to 0.6787 by the final epoch. The small and consistent reductions indicate stable learning with minimal overfitting. Model 3 generalises well compared to Models 1 and 2.

Model_4 starts with a validation loss of 1.2374, which is relatively high but decreases steadily to 0.8289 in the final epoch. The consistent downward trend suggests that the model learns effectively over time. However, compared to Model_3 and Model_5, the loss remains higher, indicating that it does not generalise well.

Model_5 outperforms all other models in validation loss reduction. It starts with a low validation loss of 0.6880 and continues to decrease, reaching 0.5080 in the final epoch. Although it fluctuates slightly, Model_5 consistently maintains the lowest loss, demonstrating the best generalisation capability.

The accuracy of a ML model is a critical performance metric, reflecting the fraction of samples accurately classified by the model. In DL, especially for complex tasks like image classification, a model's accuracy across epochs provides insight into how effectively the model learns from the data and generalises to new, unseen instances.

Figure 4.11 elaborates on accuracy trends observed across five distinct custom CNN models, measured over ten training epochs. These models are compared based on their performance and ability to enhance classification accuracy over time.

Model_1 begins with a relatively low accuracy of 0.5311 and fluctuates over the epochs. It reaches its highest accuracy of 0.5863 in the final epoch, showing some improvement. However, the overall progression remains slow, and its final accuracy is significantly lower than other models, indicating that Model_1 struggles to learn effectively.

Model_2 starts at 0.6187 and remains relatively stable throughout training. Although it slightly increases to 0.6443 at epoch 9, it does not exhibit a strong upward trend. In the final epoch, the accuracy slightly drops to 0.6399, suggesting that the model does not

improve significantly as training progresses.

Model_3 begins with the lowest initial accuracy, 0.4900, but steadily improves. By epoch 6, the accuracy jumps to 0.5821 and continues to increase, reaching 0.5904 at epoch 9. However, it slightly declines to 0.5721 in the final epoch. Despite its improvements, Model_3 does not achieve competitive performance compared to other models.

Model_4 shows a significant upward trend in accuracy. Starting at 0.4887, it consistently improves, surpassing 0.7 by epoch 4 and reaching 0.8298 in the final epoch. This indicates strong learning capability and practical training, outperforming Models 1, 2, and 3.

Model_5 demonstrates the best performance among all models. It begins with the highest initial accuracy of 0.7238 and continues to improve steadily. By epoch 5, it surpasses 0.8; in the final epoch, it reaches 0.8611, the highest among all models. This confirms that Model_5 generalises the best and achieves the most consistent improvements.

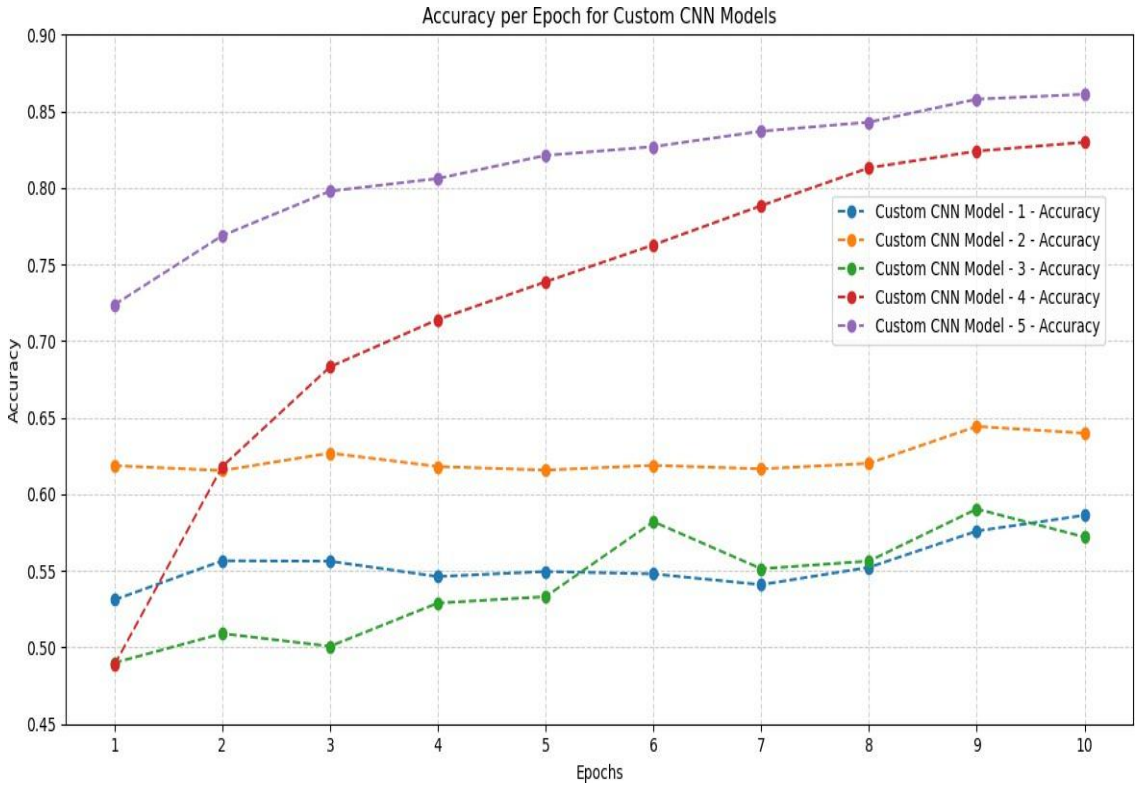

Figure 4.11 Training Accuracy of Five Custom CNN models

Validation accuracy offers critical insights into a model's capability to generalise to previously unobserved data, which is vital for evaluating its robustness and practical applicability. Figure 4.12 analyses the validation accuracy of five different Custom CNN models over ten epochs, highlighting their performance trends and comparing their ability to generalise effectively.

Model_1 starts with a validation accuracy of 0.4666, but instead of improving, it fluctuates and eventually decreases to 0.4569 in the final epoch. This suggests that Model_1 struggles with generalisation and performs poorly on unseen data.

Model_2 begins with a validation accuracy of 0.5257 and remains relatively stable throughout training. Although it reaches a peak of 0.5321 at epoch 7, it does not show significant improvement and finishes at 0.5143 in the final epoch. The minimal increase indicates that Model_2 does not generalise effectively.

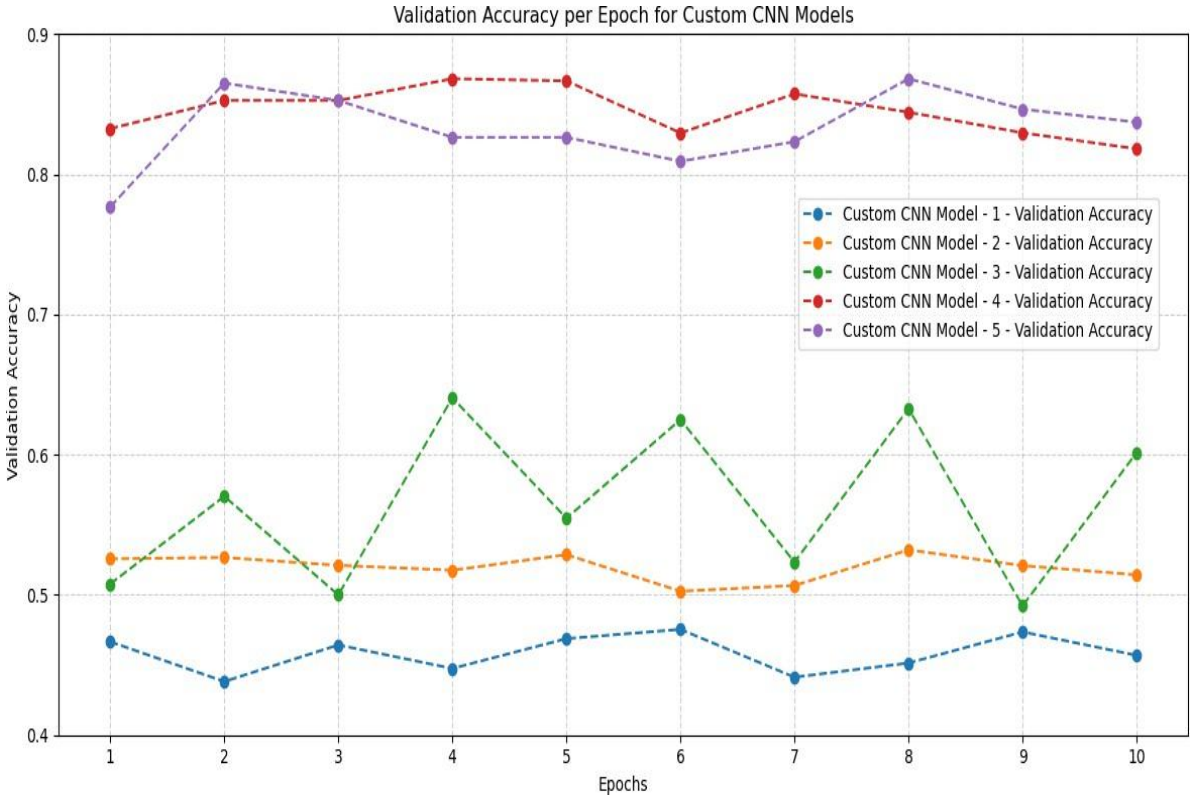

Figure 4.12 Validation Accuracy of Five Custom CNN models

Model_3 shows inconsistency in validation accuracy. It begins at 0.5078, grows to

0.6406 at epoch 4, and oscillates before stabilising at 0.6016 at epoch 7. Though it shows improvement, the instability shows that Model_3 is possibly unstable in real-life scenarios.

Model_4's initial validation accuracy is 0.8326. It has consistently increased, and at epoch 4, it became 0.8682. While validation accuracy decreased slightly in the last epochs,it is still high, showing generalisation.

Model_5 achieves the highest validation accuracy of all models. It's initialised with 0.7767 and gradually gets better, reaching the peak of 0.8682 in epoch 8. It is 0.8372 at the last epoch. Of the five, the steady ascending and the high final accuracy indicate that Model_5 is the most generalizable among the five models.

Table 4.10 presents the comparative performance metrics of five proposed Custom CNN Models and two Existing CNN Models. These measures give valuable information on the model's performance in instance classification, in tasks where the detection of positive instances and the reduction of false positives and false negatives are important. It seeks to provide a systematic evaluation of each model's merits and limitations and evaluate their application in practice.

Custom CNN Model 1 shows poor performance among these five CNN models with an accuracy of 45 %, a precision of 41 %, a recall of 45 %, and an F1-score of 42 % while the Inception-ResNet-v2 and custom CNN Model slightly outperformed with an accuracy of 48% , a precision of 50 %, a recall of 48 %, and an F1-score of 49 %. A poor classifier's capacity causes the low accuracy and recall to classify the instances, resulting in many false negatives. Therefore, the low recall also indicates that the model does not perfectly lower the rate of false positives. Hence, it results in untrustworthiness in the classification. The low performance quality might result from flawed model architecture, ineffective feature extraction or unsuitable hyperparameter tuning.

Custom CNN Model 2 performs slightly better than Custom CNN Model 1 and Inception-ResNet-v2 and custom CNN Model, with an accuracy of 51%, a precision of 46%, a recall of 48%, and an F1-score of 47%. Although the accuracy and recall values indicate some improvement in correctly identifying instances, they still reflect moderate performance with room for enhancement. The precision of 46% suggests that the model

slightly reduces false positives compared to Custom CNN Model 1, but still lacks robustness in classification.

Table 4.10 Performance metrics for five custom CNN models with Existing CNN Model

Custom CNN Models	Accuracy (%)	Precision (%)	Recall (%)	F1score (%)
Inception-ResNet-v2 and custom CNN Model [90]	48	50	48	49
Custom CNN Model [91]	73	48	51	48
Custom CNN Model - 1	45	41	45	42
Custom CNN Model - 2	51	46	48	47
Custom CNN Model - 3	60	49	57	53
Custom CNN Model - 4	84	81	80	79
Custom CNN Model - 5	88	86	84	84

The marginal gain of Custom CNN Model 2 over Custom CNN Model 1 suggests more thoughtful architecture design or hyperparameter tuning. However, the model also has difficulty generalising well across classes.

Custom CNN Model 3 demonstrates a noticeable improvement compared to the previous models, with an accuracy of 60%, a precision of 49%, a recall of 57%, and an F1-score of 53%.. Custom CNN Model 3 achieves a better trade-off between precision and recall, meaning it captures more helpful information from the dataset. The F1-score of the Custom CNN model 3 is 53%, indicating a better tradeoff between accuracy and recall, resulting in improved classifications. Still far from ideal, the third model is much more equipped to deal with the task than the first two, and serves as a good stepping stone to further development.

Compared to Custom CNN Model [91] with an accuracy of 73%, a precision of 48%, a recall of 51%, and an F1-score of 48% , Custom CNN Model 4 does show strong performance, with 84% accuracy, 81% precision, 80% recall, and an F1-score of 79%. The high scores tell us that the model minimises both false positives and false negatives; hence, it is a good classification model. The Custom CNN model 4 has a high precision, which means it has better decision boundaries. The balanced F1-score provides evidence that the Custom CNN model 4 can correctly classify the instances. This improvement can be attributed to a more complex architecture, stronger feature abstraction ability, and a more effective training method.

Custom CNN Model 5 outperforms all the CNN models including two existing CNN models with an accuracy of 88%, precision of 86%, recall of 84% and F1-score of 84%. These measures show strong generalisation performance and low classification error, even under this comparison, and thus, it is the best model among all models. The high precision and recall reflect its ability to reliably detect the instances of interest and avoid false positives and negatives. This degree of performance is evidence that Custom CNN Model 5 possesses the best architecture and hyperparameters for such a task and is therefore the most appropriate candidate for deployment in real-world classification problems.

4.10 DISCUSSION

This chapter provides a comparative study of different Custom CNN-based models regarding their classification performance. The performance metrics for models are accuracy, precision, recall, and F1-score.

Custom CNN Model 5 is selected among the models as it had the best results among the five models and existing custom CNN models, with an accuracy of 88% and a reasonable precision-recall trade-off. Due to its feature-optimised architecture and optimal hyperparameters, it realised a powerful classification performance and is accepted as the most appropriate model for deployment.

Custom CNN Model 1, on the other hand, is the least accurate, with 45% accuracy and 42% F1, suggesting the inability to differentiate between classes. The second model performed somewhat better, but it also suffered from generalisation. Custom CNN Model 3 dramatically improved accuracy and recall, proving the step-by-step architectural

improvements. Custom CNN Model 4 performed an excellent job in reducing false positives and negatives, reaching a high accuracy of 84%.

Overall, the study evidences that advancing CNN architectures by optimising hyperparameters is key in enhancing learning performance. The classification accuracy increases from Custom CNN Model 1 to Custom CNN Model 5, indicating the significant influence of feature extraction and network.