Photoreceptor-inspired CNN Model for Object Recognition

By

Nilou Ghazavi

Supervisor

Dr. Joel Zylberberg

Table of Contents

1. Background Information

The retina, a vital layer of our vision system, plays a crucial role in image formation, light and dark adaptation, and color vision. The retina consists of specialized sensor neurons and intricate neural circuits that play a vital role in the initial stages of image processing. These processes generate electrical signals that are then transmitted through the optic nerve to the brain, allowing for further processing and visual perception [1].

The retina contains two main types of photoreceptors: rods and cones. Rods are specialized for low-light vision, while cones are responsible for daylight and color vision. Human possess three types of cones that are sensitive to green, blue, and red light. The fovea, located at the center of the retina, is densely packed with cones and the peripheral regions of the retina are more sensitive to low-light conditions [1].

Adaptation is a crucial aspect of photoreceptor function with rods and cones exhibiting distinct characteristics. Rods adapt to slow changes in light, while cones adapt to rapid light fluctuations. This adaptive process enables the decomposition of images into separate components and occurs at the synapses between photoreceptors and bipolar cells. Different bipolar cells have different types of receptors for the neurotransmitter glutamate enabling them to respond differently to input from photoreceptors [1].

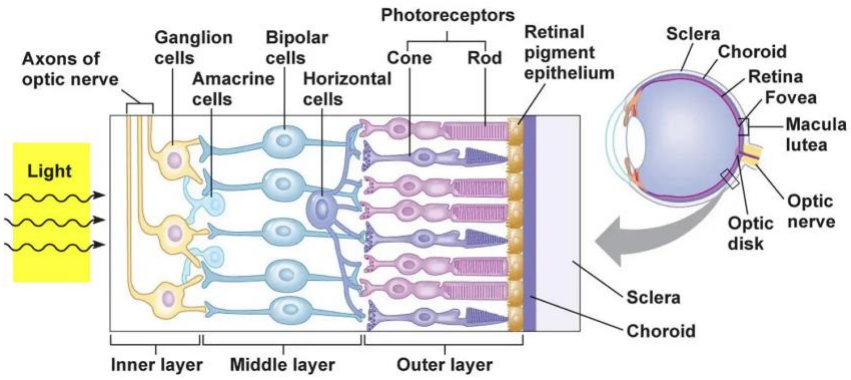


Figure 1.Primary Layers of the Retina [2]

Upon exposure to light, the membrane of photoreceptors undergoes hyperpolarization, following the temporal profile of the light stimulus and returns to its baseline state once the light stimulus is terminated. In darkness, both rods and cones release neurotransmitters as their membrane depolarize. However, when exposed to light, the ion channels in the cell membranes close, resulting in a sustained hyperpolarized state as long as the light is present [1].

Some bipolar cells are specialized in perceiving rapid fluctuations in visual signals, while others are tuned to slower changes. Some glutamate receptors in these cells quickly resensitize, while others do at a more gradual pace. So, these receptors enable the cells to either fire rapidly in succession or respond relatively slowly to the same level of stimulation. The activated receptors on these bipolar cells initiate an OFF pathway for detection of dark

images in relation to their surrounding background. On the other hand, certain bipolar cells possess inhibitory glutamate receptors, which prevent the cells from firing when exposed to the neurotransmitters. These inhibitory receptors activate the ON pathway for detection of light images against a darker background [1].

In the absence of light, both rods and cones in the retina maintain a membrane potential of approximately -40 mV. The amplitude of the electrical signals generated by the photoreceptors, reflects the brightness or intensity of the stimulus. The dynamics of this response encompass various aspects, including the time required to reach the peak response, the duration of the response, and the kinetics of onset and offset. When a photoreceptor is in darkness, even a small flash of flight elicits a significant and gradual change in its electrical potential. However, when the same small flash occurs in a bright background, the response is less pronounced (smaller amplitude) and faster (quicker change). This phenomenon is known as gain suppression and helps the photoreceptors to adapt to different lighting conditions [3].

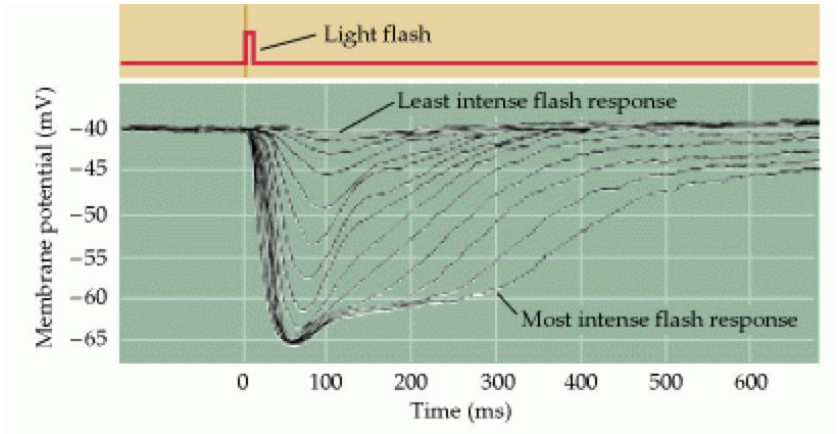


Figure 2. Photoreceptor responses from a single cone to a brief flash that was varied in intensity [3]

Adaptation enhances the encoding and processing of time-varying stimuli. For example, neurons may exhibit faster response dynamics to track rapidly changing sensory inputs. By adjusting the response characteristics, neural systems can prioritize important features, enhance the signal-to-noise ratio, and optimize the representation of relevant information [1].

2. Objective

Object detection is a complex and diverse field within computer vision with broad applicability. The use of convolutional neural networks (CNNs) has become popular in the development of models that can accurately identify and classify diverse image features. However, the performance of these models can be significantly impacted by dynamic lighting conditions. In real-world scenarios, lighting can vary greatly due to shadows and sunlight leading to changes in light intensity of 10-100-fold. Current CNN models do not account for this effect, resulting in poor performance in inconsistent lighting environments. The primary aim of this study is to optimize the ability of these models to detect and recognize objects accurately under a broad range of lighting conditions.

In order to account for fluctuations in light levels, we added a photoreceptor model layer as a front-end to a standard CNN model. This dynamic adaptation/CNN model is capable of classifying visual inputs with a wide range of lighting such as natural stimuli. This model imitates the behaviour of retinal photoreceptors and replicates the process of converting optical signals into electrical signals [4]. Photoreceptor adaptation depends on the temporal structure of the stimuli which affects the kinetics of the response [5].

To evaluate the performance of the photoreceptor-inspired CNN model, we generated a video stream using diverse range of images. To introduce variations in lighting conditions, we created some masks to simulate sudden large changes in the amount of light reflected such as those caused by shadows or changes in the intensity of the light source. These changes in lighting conditions were incorporated into some of the frames within the video stream.

The hypothesis is that the photoreceptor-inspired model can adjust for variation in light levels and subsequently achieve greater accuracy in object detection and object recognition. ‘

3. MNIST Dataset

3.1. Introduction

The MNIST dataset consisting of handwritten digits, was utilized to generate movies comprising 20 frames. To introduce intensity variations, binary masks with different intensity factors were applied to random frames within the movies. The purpose of these modifications was to simulate diverse scenarios and evaluate the Photoreceptor model’s ability to recognize patterns under dynamic lighting conditions.

3.2. Dataset

The MNIST dataset consists of grayscale handwritten digits representing 10 different classes: 0,1,2,3,4,5,6,7,8 and 9. Each class corresponds to a unique digit, allowing for classification tasks and pattern recognition analysis using the dataset.

To increase the difficulty of the classification task for the simple MNIST dataset, Gaussian noise was introduced to both train and test datasets. The model was evaluated using both clean and noisy images (refer to section 3.0 of the PRModel_MNIST notebook).

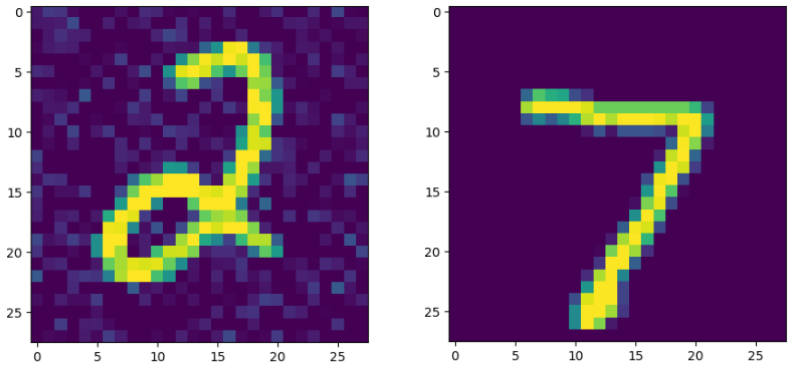


Figure 3. Comparison of a clean image (right) and an image with Gaussian noise added (left)

3.3. Data Preparation

The pixel values in these handwritten images range from 0 to 255, where 0 denotes a black pixel and 255 corresponds to a white pixel. To normalize the pixel values, each pixel in the images was divided by 255. Each image has a dimension of 28x28 pixels. The digits are centrally positioned within the images, with a bright region representing the digit and a dark background surrounding it. Each image was replicated 20 times to generate movies for testing and training the models. The dataset was divided into two subsets: training and testing. The training set consists of 60000 images, while the testing set consists of 10000 images.

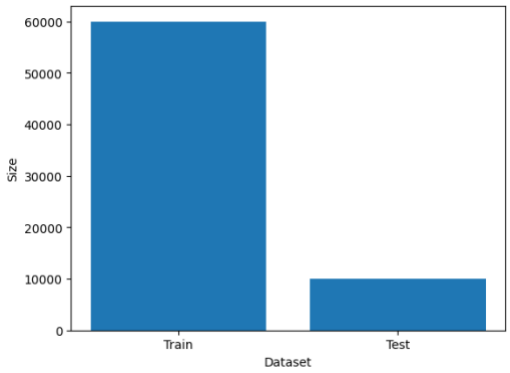


Figure 4. Distribution of data between train and test datasets

3.4. Masks

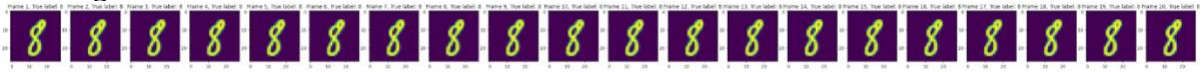
To assess the Photoreceptor model's performance in the presence of dynamic lighting conditions, binary masks were used to simulate real-world scenarios. By applying these masks, shadows were introduced onto the test dataset. Masked images were only used during training, to ensure that the model's performance is assessed on unseen samples (refer to section 6.0 of the PRModel_MNIST notebook).

The binary mask was applied to the images as follows:

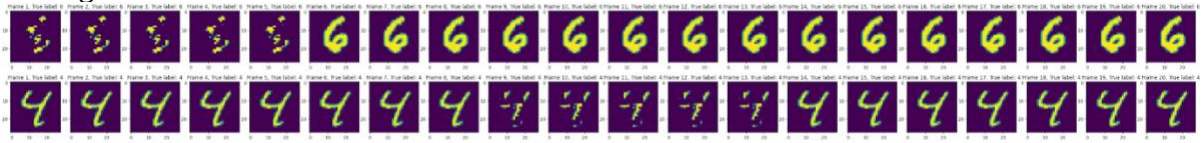
$$Modified\ Image = Image * (Mask * Intensity\ Factor) + Image * (1 - Mask)$$

In this equation, the mask represents a binary image and intensity represents a scalar value controlling the strength of the modification. In this case, the intensity values are less than 0.02 to simulate shadows in the image. Using a value greater than 1 would only make the digit appear brighter since the background pixels have a value of 0 so classifying digits would be easier for the CNN model.
Each sample of the dataset was subjected to the same mask. However, the duration and start frame of the mask varied for each sample (refer to section 7.0 of the PRModel_MNIST notebook).

Training Dataset



Testing Dataset



3.5. Data Generator

In order to compare the predicted and actual labels for each frame within a movie, a data generator function was used. Each sample represents a movie which consists of multiple frames. For each batch, the labels are repeated for each frame in the movie as the goal is frame-level prediction (refer to section 11.0 of the PRModel_MNIST notebook).

3.6. Dynamic Adaptive Model

In the Photoreceptor (PR) model, the response of the photoreceptor is determined by subtracting the resting membrane potential from the instantaneous membrane potential (Eq.1). This process allows photoreceptors to adjust and react to fluctuations in the optical signals over time [5]. The optical signal, s(t), which is the input to the PR model contains information about the intensity and spatial distribution of the light (Figure 5)

$$Eq.(1) \quad r(t) = V(t) - Vrest$$

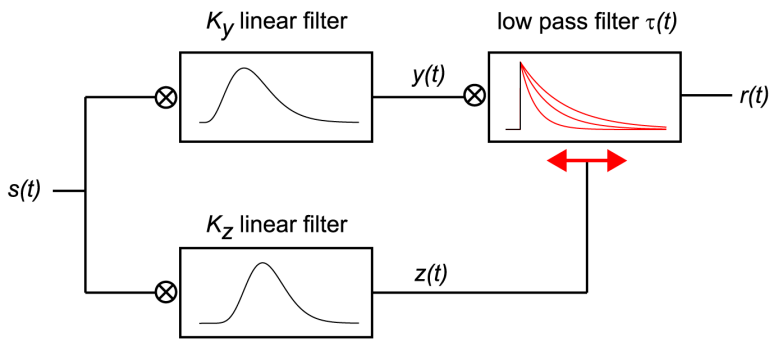


Figure 5. Dynamical Adaptation (DA) Model

The PR model involves performing convolutions between the input (the intensity of the pixels) and K_y (fast linear filter) and K_z (slow linear filter) (Figure 6) . These convolutions generate the outputs y(t) and z(t) (Eq.3 and Eq.4). By applying the K_y and K_z filters to the input, the model is capable of capturing and analyzing both fast and slow features or patterns.

$$Eq.(3)\ y(t) = \int_{-\infty}^{t} dt' Ky(t-t')s(t')$$

$$Eq.(4)\ z(t) = \int_{-\infty}^{t} dt' Kz(t-t')s(t')$$

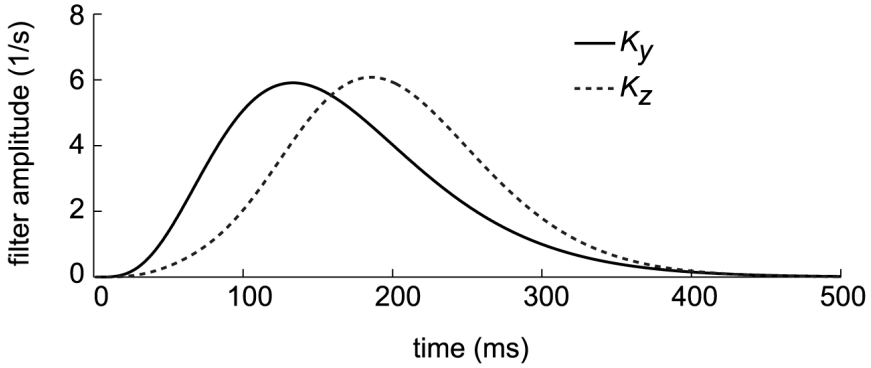


Figure 6. Kz and Ky linear filters

The filters K_y and K_z are obtained using Eq. (9) and Eq. (10) respectively. $K_y(t)$ serves as a simplified representation of the phototransduction cascade, a process that translates light into electrical signals within visual system. $K_z(t)$ integrates two distinct time scales: a rapid component τy and a slower component τz. In order to evenly allocate the influence of these two components, the perfectors γ and 1- γ are introduced. τy represents the time scale of the linear response, indicating how rapidly the system responds to changes. The term "n_y" defines "rise" behaviour, which refers to how the system's response fluctuates over time when it is subjected to a stimulus. For this study, $K_c(t)$ filter was removed to simplify the PR model [5].

$$Eq.(9)\quad Ky(t) = \frac{t^{ny}}{ny!\tau y^{ny+1}} e^{(-t/\tau y)}\ \theta(t)$$

$$Eq.(10)\ Kz(t) = \frac{t^{ny}}{ny!\tau y^{ny+1}} e^{(-t/\tau y)}\ \theta(t)\ \gamma + \frac{t^{nz}}{nz!\tau z^{nz+1}} e^{(-t/\tau z)}\ \theta(t)\ 1-\gamma)$$

The main Dynamic Adaptive equation is defined as:

$$Eq.(2)\ \tau r\frac{dr(t)}{dt} = \alpha y(t) - [1 + \beta z(t)]r(t)$$

When $\beta = 0$, then the equation becomes linear and the response is a low-pass filtered version of the input:

$$Eq.(5)\ r(t) = \int_{-\infty}^{t} \frac{dt'}{\tau r}\alpha y(t')\exp\left(-(t - t')/\tau r\right)$$

When $\beta \neq 0$, the z-term (z(t)) modulates both gain and dynamics, by dividing both sides of Eq. (2), the following equation is obtained:

$$Eq.(6)\frac{\tau r}{\left(1 + \beta z(t)\right)\partial r(t)} + r(t) = \left(\frac{\alpha}{1 + \beta z(t)}\right)y(t)$$

Where

$$\alpha\_effective = \frac{\alpha}{[1 + \beta z(t)]}$$

$$\tau r_{effective} = \frac{\tau r}{[1 + \beta z(t)]}$$

Eq. (2) can be solved by:

$$Eq.(7)\ \ r(t) = \int_{-\infty}^{t} \frac{dt'}{\tau r}\alpha y(t')\exp\left(-\int_{t'}^{t} \frac{dt'}{\tau r}[1 + \beta z(t)]\right.$$

By substituting the stimulus, s(t) into Eqs. (3,4,7), analytically or numerically, the model photoreceptor response can be obtained.

The general solution of the DA model can be rewritten as:

$$Eq.(12) PR\ Output = \frac{\zeta + \alpha * y(t)}{\kappa + \beta * z(t)}$$

Photoreceptor operates on a pixel-wise level and it takes into account the temporal structure. The following diagram shows, the intensity variation of two pixels (0,0) and (0,2) over 20 frames. The stimulus is convolved with two mono-lobed filters (Ky and Kz) to produce signals y(t) and z(t). To obtain the output of the PR model using Eq.12, the intensity change is attenuated as y-term is divided by z-term. In fact, in the presence of intensity variations, the PR model acts as a normalization layer, counteracting the impact of these fluctuations.

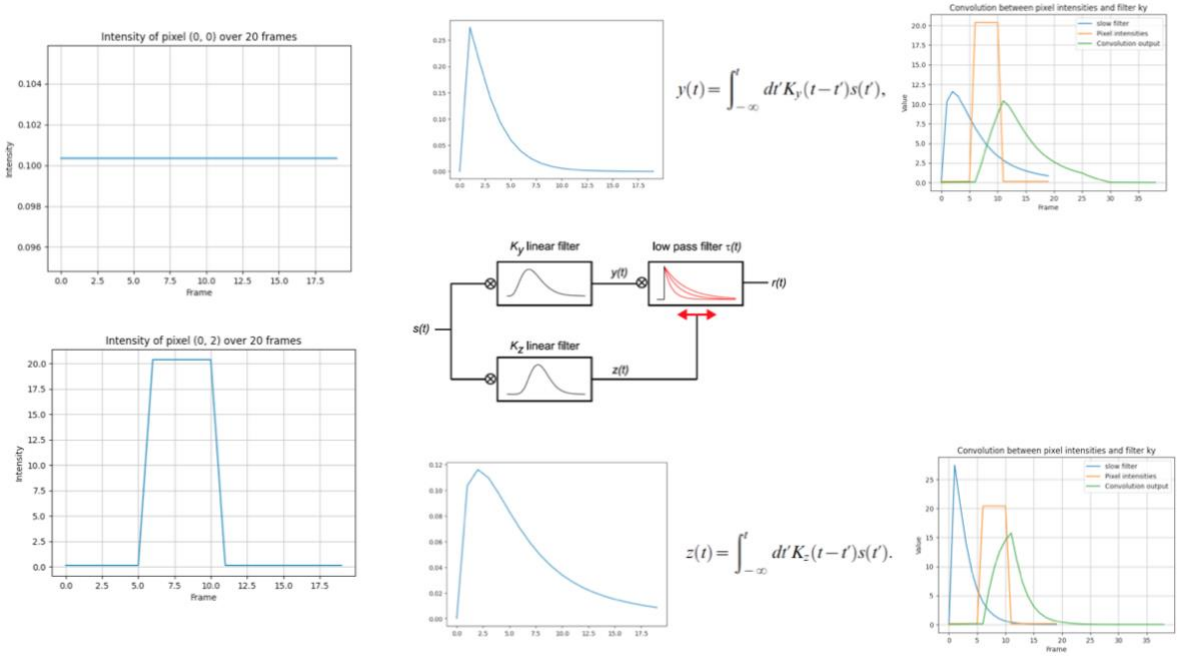


Figure 7. Convolution operation between input and mono-lobed filters (Ky and Kz)

3.7. Photoreceptor Model Parameters Optimization

Two different approaches were employed to identify the optimal values for the PR model. In the first method, the model was trained using masked images, enabling it to learn optimal values over time. In the second method, the root mean squared error between the PR model's output (a masked frame) and the input (an unmasked frame) was computed. As the objective was to effectively remove the mask using the PR model, the values corresponding to the minimum RMS error were considered as the optimal values.

For the second method, the initial range for the Ky and Kz filters' parameters (τ_y n_y, τ_z, n_z and γ) were obtained based on the shape of filters for various combinations of parameters.

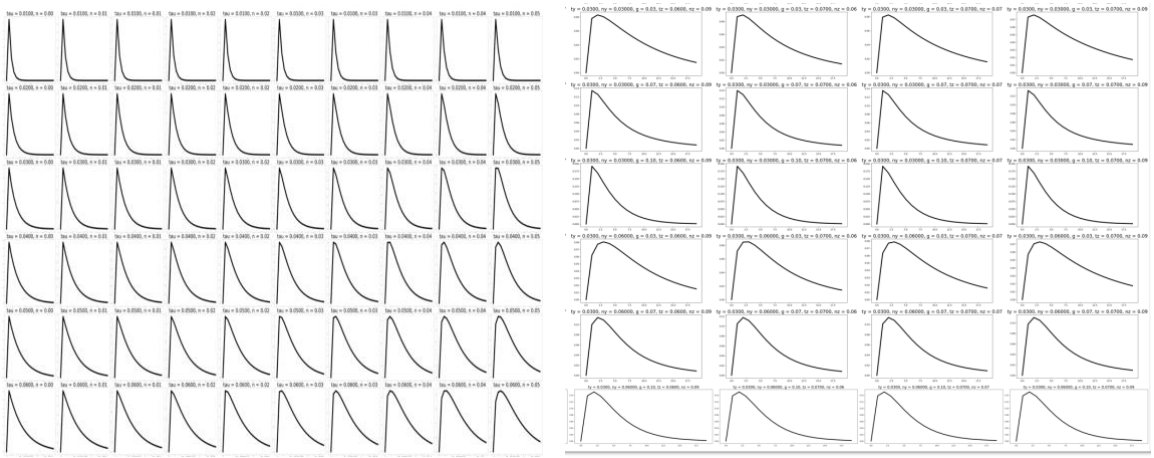
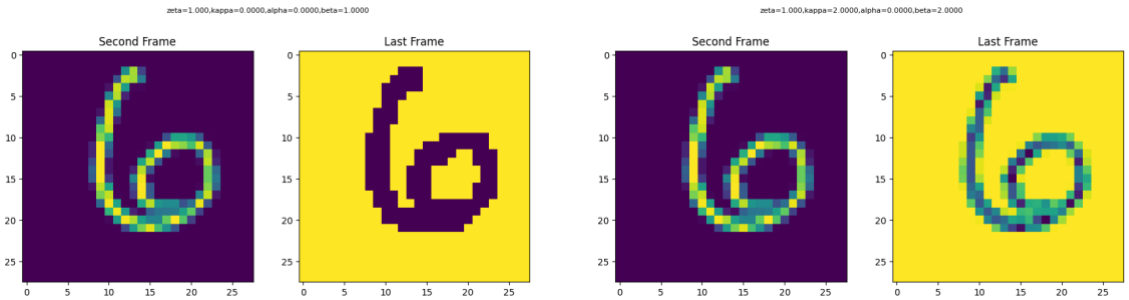


Figure 8. Shape of the fast filter (left) and slow filter (right) for different combinations of parameters

Subsequently, the weight values were manually adjusted using a nested loop. To reduce computational complexity, the 9 parameters were divided into two sets: Set 1 (τ_y, n_y, τ_z, n_z, and γ) and Set 2 (ζ, κ, α , and β). The parameters in Set 2 were initially set to values obtained from method 1, while the parameters in Set 1 were iteratively modified within the nested loop. This process was performed vice versa for parameters in Set 2 (refer to section 13.2 of the PRModel_MNIST notebook).

The following diagrams show pairs of the second and last frames obtained using four different combinations of PR parameters. The plot (Figure 10) illustrates the variation in RMS error for all combinations.



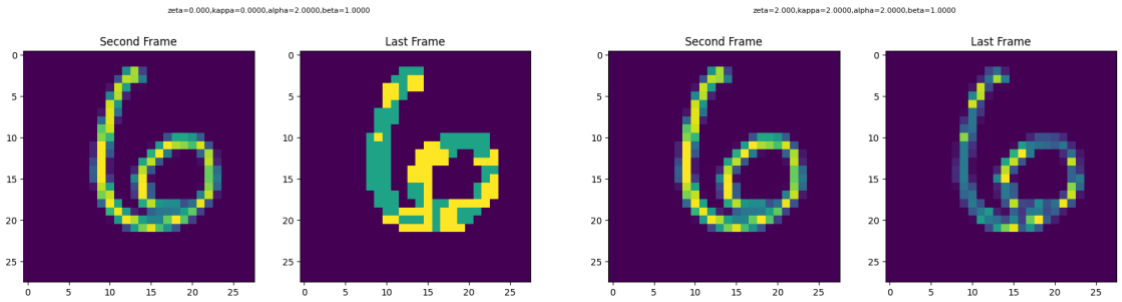


Figure 9.Comparison of the second frame of the input and the last frame of the PR model output

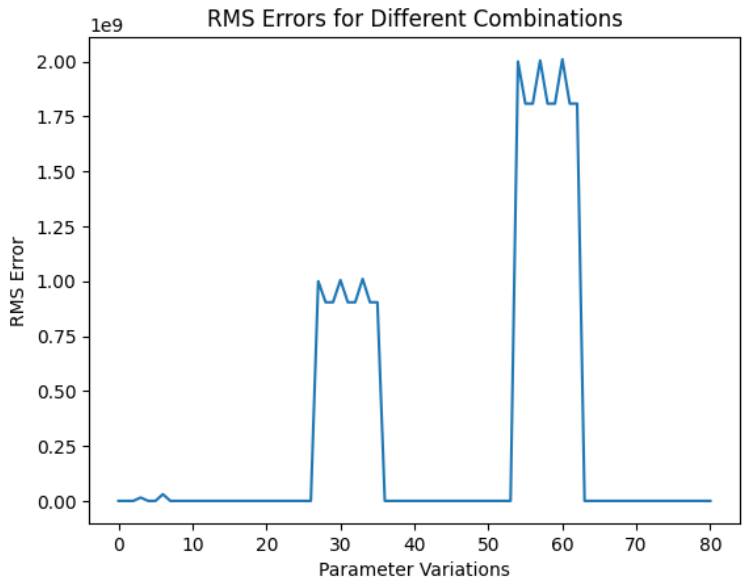


Figure 10.RMS errors for different combinations of the PR parameters

Based on the outcomes from both methods, it was concluded that the ideal parameter range for all variables should be defined as 0-1 and the multiplication factors remain unchanged throughout the training process.
After training the PR+CNN model using unmasked images, we obtained the corresponding weight values (Test: Optimal Values).

PR Parameters	Method 1: Optimal Values	Method 2: Optimal Values	Test : Optimal Values
ζ	0.02	0.0100	0.0200
κ	0.0483	1.0050	0.0309
α	0.05	2.0000	0.0500

β	0.02	0.0100	0.0200
γ	0.02	0.0200	0.0200
τ_y	0.02	0.0200	0.0200
n_y	0.0229	0.0200	0.0200
τ_z	0.0500	0.0500	0.0500
n_z	0.0500	0.0500	0.0500

Table 1. Optimal weight values

3.8. Models

In order to evaluate the performance of the Photoreceptor model, a convolution neural network (CNN) was used as the base model. Subsequently, the PR model was integrated into the CNN architecture, resulting the combined model (CNN+PR). Both models were evaluated using movies with 20 frames, including 5 masked images. The evaluation primarily focused on the predicted labels for the frames immediately after the intensity change.

3.8.1. Photoreceptor Model

To enable the PR model to handle time series data, the following lines were removed from the code:

```
# y_shift = tf.math.argmax(Ky,axis=1);y_shift = tf.cast(y_shift,tf.int32)
# z_shift = tf.math.argmax(Kz,axis=1);z_shift = tf.cast(z_shift,tf.int32)

# y_tf_reshape = slice_tensor(y_tf_reshape,y_shift)
# z_tf_reshape = slice_tensor(z_tf_reshape,z_shift)
```

3.8.2. Time-Distributed CNN Model

The Time-Distributed CNN model (refer to Section 12.4 of the PRModel_MNIST notebook) is a modified convolutional neural network (CNN) architecture for processing movies/sequence of images. The time independent CNN model was wrapped around a time distribution so that each convolutional layer is applied to each time step/frame of the movie. The architecture consists of two convolutional layers, each followed by layer normalization and max pooling operations. The model captures spatial information across the sequence. The

flatten feature maps are then subjected to dropout (50%) to prevent overfitting before being fed into dense layer with SoftMax activation function.

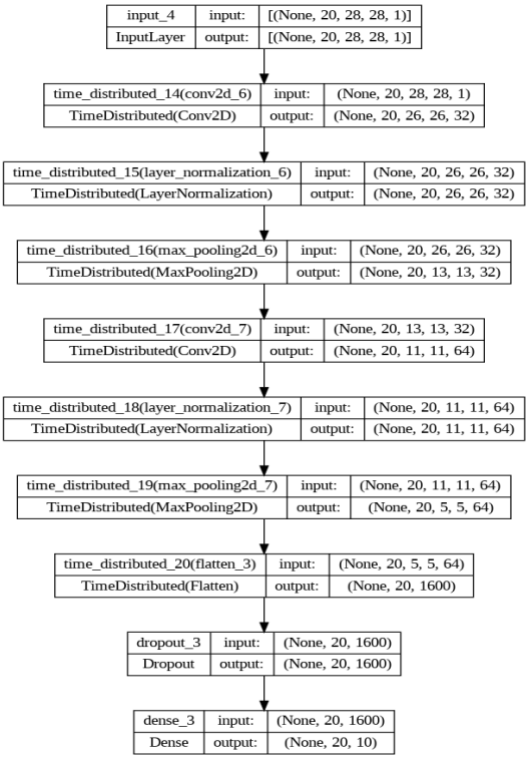


Figure 11. CNN model architecture

During gradient descent optimization using the Adam optimizer, the step size was set to 0.00001. After observing that the average accuracy reached a plateau after the 5th epoch, a training duration of 10 epochs was selected for this model.

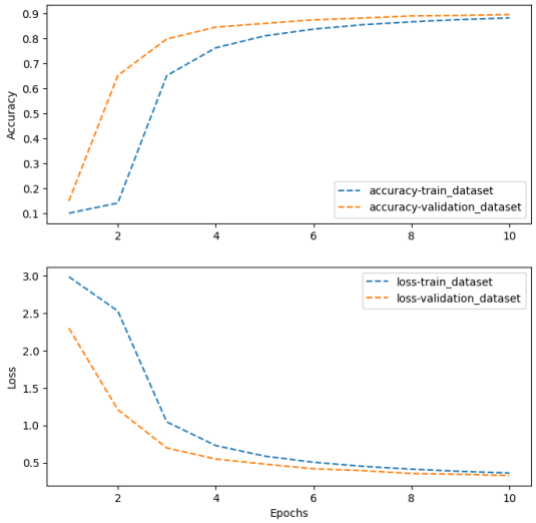


Figure 12. Validation and test accuracies

To calculate the true positive rate, the predicted and true labels for each movie were compared. The first two frames of the movie were excluded due to potential edge effects. A true positive rate was assigned when all labels matched between the predicted and true labels. This measurement allowed us to evaluate the accuracy of the model's predictions specifically after the intensity change in each movie.

With a decrease in intensity values (0.02, 0.002 and 0.0002), indicating the presence of shadows, the misclassification rate increased. The corresponding true positive rates were determined to be 58.21%, 55.06% and 56.84 % respectively.

3.8.3. Time-Distributed CNN+PR Model

In order to improve the performance of the CNN base model, the Photoreceptor (PR) model was incorporated (refer to Section 12.2 of the PRModel_MNIST notebook). The output of the PR model was then fed into the CNN model (Figure 13).

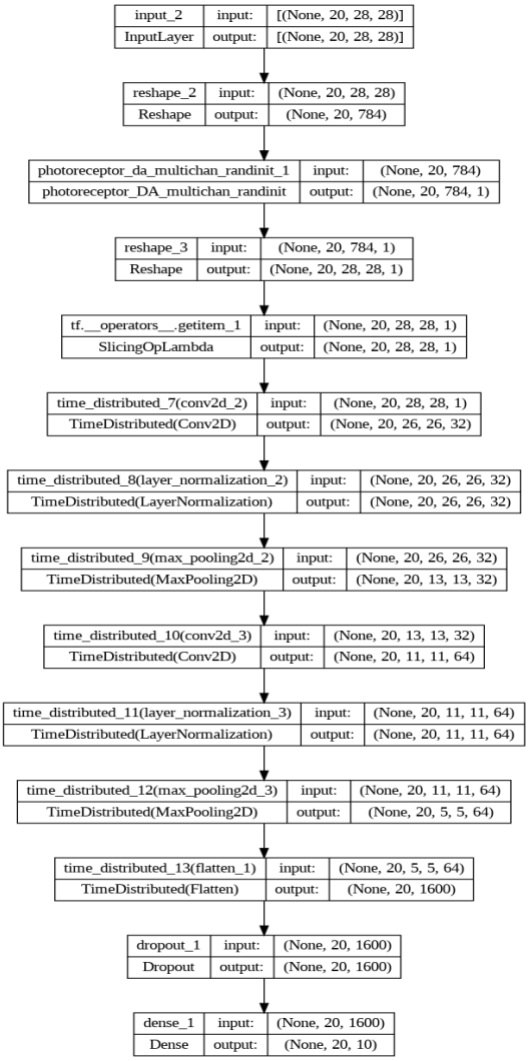
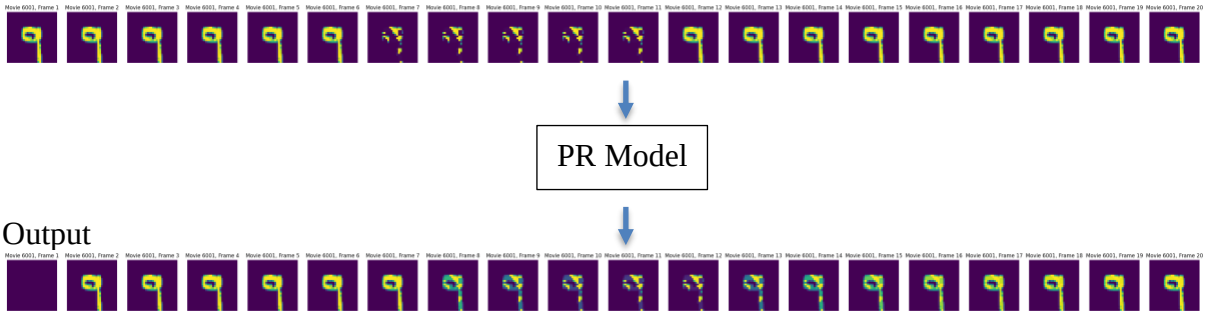


Figure 13. PR+CNN model architecture

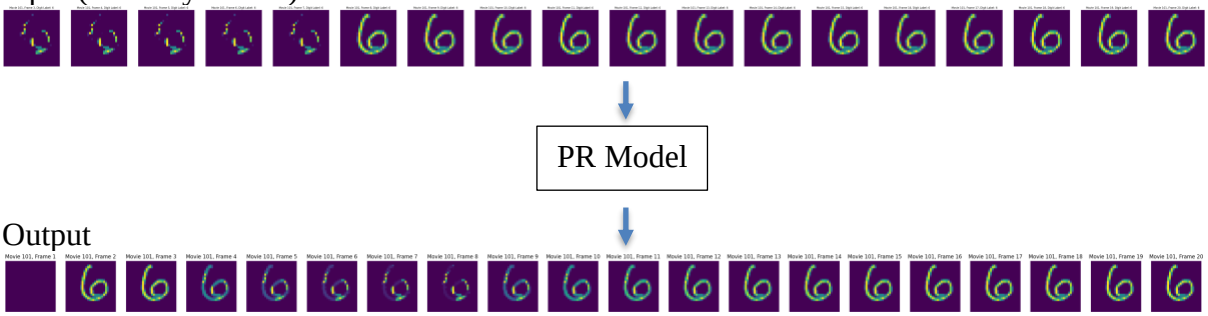
In order to maintain consistency during the evaluation process, the PR+CNN model underwent training using the same learning rate (0.00001) and number of epochs (10) as the base CNN model.

The following figures show the input to the PR model consisting of masked images with varying intensities, and the corresponding output generated by the PR model. With the removal or normalization of the mask through the PR layer, the output becomes detectable and recognizable.

Input (intensity=0.02)



Input (intensity=0.002)



In comparison to the CNN model, the true positive rate for the PR+CNN demonstrated substantial increase. The true positive rate reached values of 81.8%, 80.17% and 80.47% for intensities of 0.02, 0.002 and 0.0002 respectively.

3.9. Results

The combination of the PR and CNN model demonstrate a significant improvement in performance compared to the base CNN model. When the light intensity of the binary mask was set to 0.02 and 0.002, 0.0002, the performance demonstrated an improvement of 40.52 % , 45.60 % and 41.57% rise respectively.

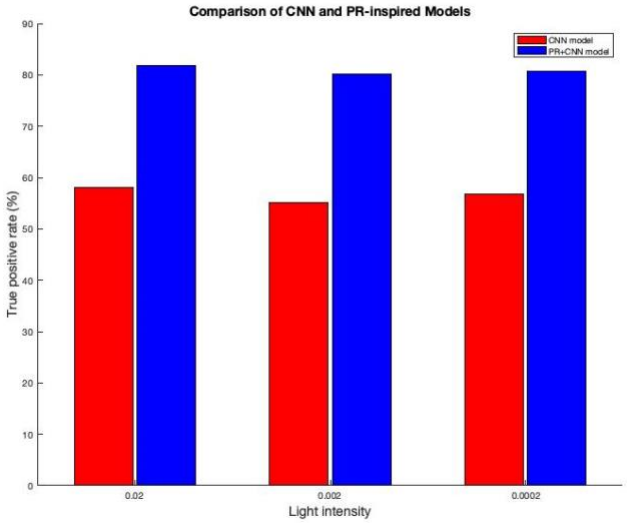
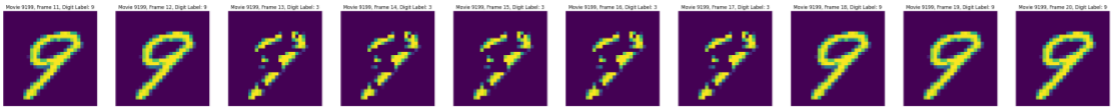


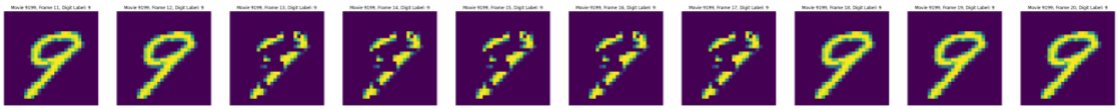
Figure 14. Comparison of true positive rate for CNN and PR+CNN models

For this test, masks were applied to 5 frames with their starting positions randomized. The following figures show that the CNN model initially classified the digit 9 as 9 prior to the mask. However, after the intensity change, the CNN model fails to recognize the digit and it was classified as 3. On the other hand, the utilization of the PR+CNN model resulted in accurate predicted labels both before and after the intensity change.

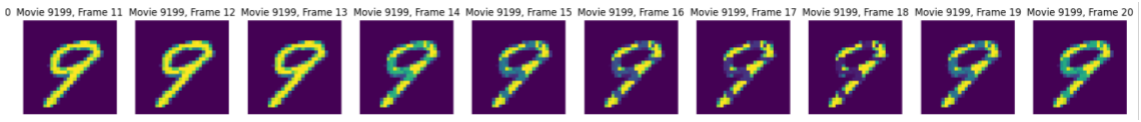
CNN Model: Predicted Labels



PR+CNN Model: Predicted Labels



PR Output



3.10. Discussion

It was observed that the optimal values for the PR parameters obtained from different methods varied, indicating the presence of multiple local minima. The collective combination of these parameters is crucial in effectively removing the mask.

Furthermore, it was noted that the model's performance degraded as the duration of the mask increased. To address this issue, increasing number of channels in the PR model to 20 enhanced the performance. In order to investigate whether the model learns new features through this increase in channel number, diagrams illustrating Ky and Kz filters for each channel were obtained. These diagrams (Figure 15 and Figure 16) show that there are variations in the gain and a shift of the peak by one frame.

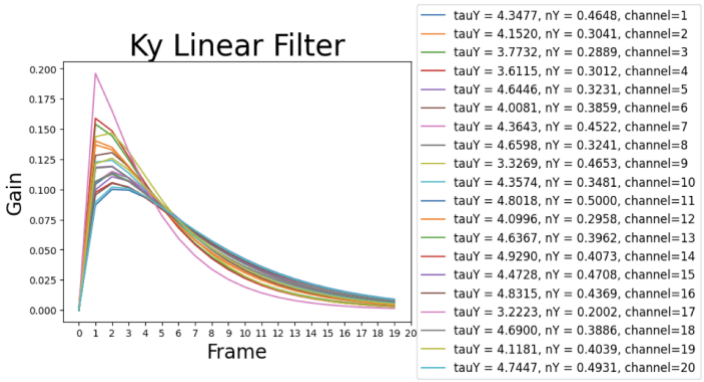


Figure 15. Ky filter for all 20 channels of the PR model

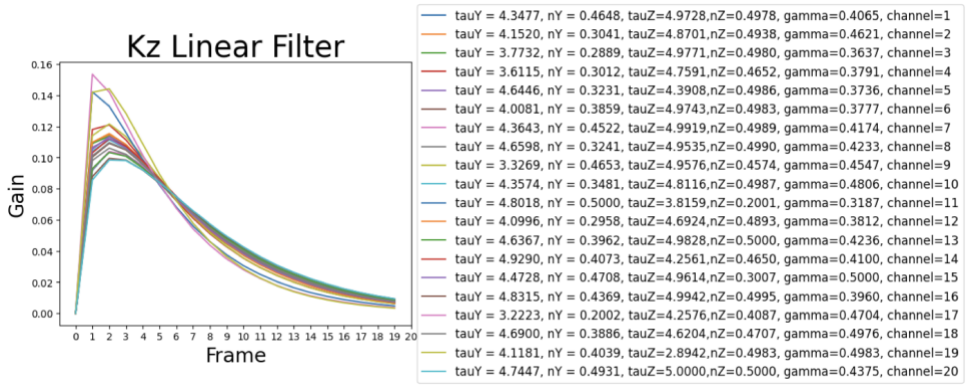


Figure 16. Ky filter for all 20 channels of the PR model

3.11. Technical Learnings

Initially, a random initializer was used to train the PR model and obtain the output. Subsequently, the weight values were manually updated within a nested loop to obtain optimal values. It was noticed that when the exact optimal values were used with a constant

initializer, the output was different. However, when a small degree of randomness (10^{-5}) was introduced and a random initializer was used, the output remained consistent.

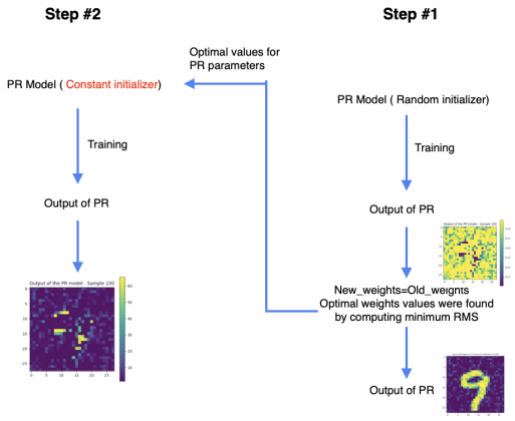


Figure 17. Comparison of constant vs random initializers

It was also observed that the order of the weight values was affected when a constant initializer was used.

		Random Initializer	Constant Initializer
Weight	0	zeta:0	**tauC:0**
Weight	1	kappa:0	zeta:0
Weight	2	alpha:0	zeta_mulFac:0
Weight	3	beta:0	kappa:0
Weight	4	gamma:0	kappa_mulFac:0
Weight	5	tauY:0	alpha:0
Weight	6	nY:0	alpha_mulFac:0
Weight	7	tauZ:0	beta:0
Weight	8	**tauC:0**	beta_mulFac:0
Weight	9	zeta_mulFac:0	gamma:0
Weight	10	kappa_mulFac:0	gamma_mulFac:0
Weight	11	alpha_mulFac:0	tauY:0
Weight	12	beta_mulFac:0	tauY_mulFac:0
Weight	13	gamma_mulFac:0	nY:0
Weight	14	ptauY_mulFac:0	nY_mulFac:0
Weight	15	nY_mulFac:0	tauZ:0
Weight	16	tauZ_mulFac:0	tauZ_mulFac:0
Weight	17	nZ:0	nZ:0
Weight	18	nZ_mulFac:0	nZ_mulFac:0
Weight	19	tauC_mulFac:0	tauC_mulFac:0
Weight	20	nC:0	nC:0
Weight	21	nC_mulFac:0	nC_mulFac:0

Table 2. Order of weight values using constant vs random initializers

Moreover, the model was evaluated with introduction of LayerNorm after the input. Surprisingly, this adjustment resulted in a significantly poor performance.

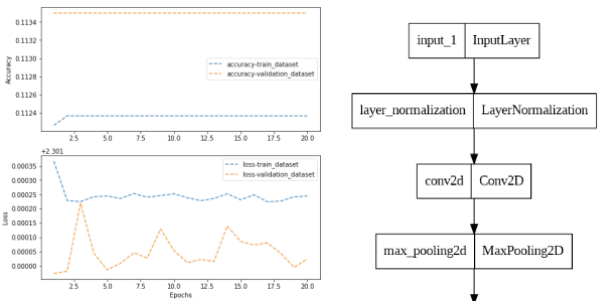


Figure 18. Accuracy plot for CNN model with LayerNorm after the input

3.12. Future Work

To further evaluate the models using MNSIT dataset, we can employ various types of masks other than binary, such as circular, rectangular, etc. Moreover, the duration of the movie can be extended beyond the current limit of 20 frames to assess the impact of the movie's duration on the performance of the model.

Also, as the initial blank frame from the output of the PR model has the potential to impact the results, the first few frames can be removed for evaluating the models. For this study, the comparison was conducted from frame #3 and the first two frames were excluded.

Due to computational constraints, it was not feasible to update all the PR parameters within the nested loop, so some parameters were set to constant values. Ideally, all parameters should be updated within the nested loop to ensure comprehensive optimization.

4. CIFAR-10 Dataset

4.1. Introduction

This study aims to evaluate the performance of the Photoreceptor model using natural movies. For this purpose, the CIFAR-10 dataset, consisting of natural color images, was used to create 20-frame movies. To modulate the intensity of specific regions within the image, circular masks were applied to the last five frames of each movie.

4.2. Dataset

The CIFAR-10 dataset comprises color images of ten classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Each class corresponds to a unique category, so different classification tasks and pattern identification can be conducted on the dataset.

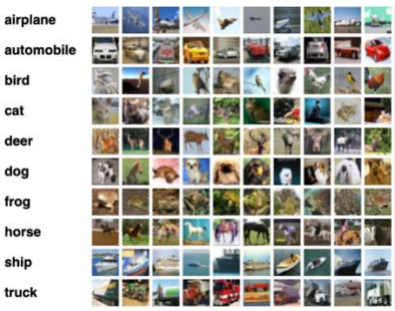


Figure 19. Classes of CIFAR-10 dataset

4.3. Data Preparation

The pixel values in the images within the CIFAR-10 dataset range from 0 to 255. In order to normalize these pixel values, each pixel in the images was divided by 255. Each image in the CIFAR-10 dataset has dimension of 32x32 pixels. The dataset was divided into two subsets: training and testing. The training set consists of 50000 images, while the testing set consists of 10000 images.

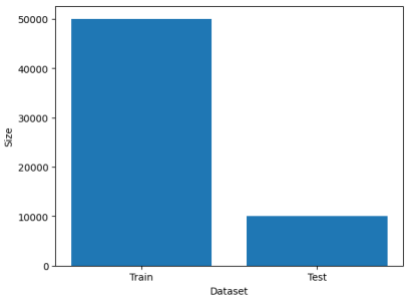


Figure 20. Distribution of data between train and test datasets

Light intensities ranging from 0.005 (dim light) to 100 (very bright light) were assigned to masks. Also, in order to provide the base models (ResNet50 and ResNet20) with a fair opportunity to learn the classification of images with masks, masked images were used in both the training and testing datasets (refer to section 3.0 of the PRModel_CIFAR10 notebook).

4.4. Masks

Circular masks with varying diameters and center coordinates were uniformly applied across all three-color channels. To selectively alter specific pixels within the images, the intensity factor (ranging from 0.005-100) was multiplied by the mask. The mask matrix contains pixels multiplied by the intensity factor, while the remaining areas are assigned to a value of 1. As a result, the intensity within the circular mask changes while the remaining areas of images remain unaffected (refer to section 5.0 of the PRModel_CIFAR10 notebook). The mask was obtained using the following equation:

$$Modified\ Image = Image * (Mask * Intensity\ Factor)$$

Testing Dataset

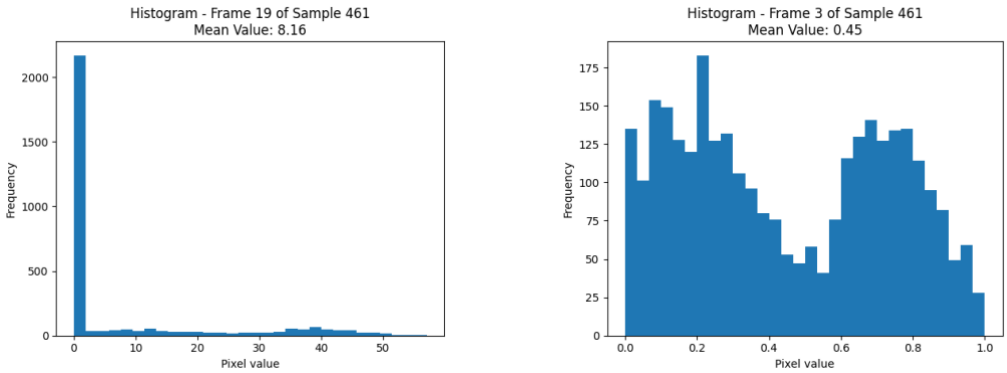
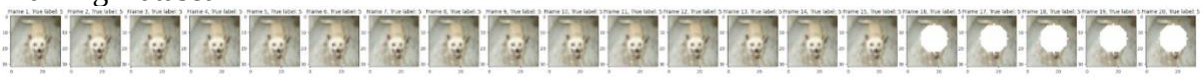


Figure 21. Histograms of a frame with a circular mask and an intensity factor of 62.29 (left) and a frame with no mask (right) from the testing dataset

Training Dataset



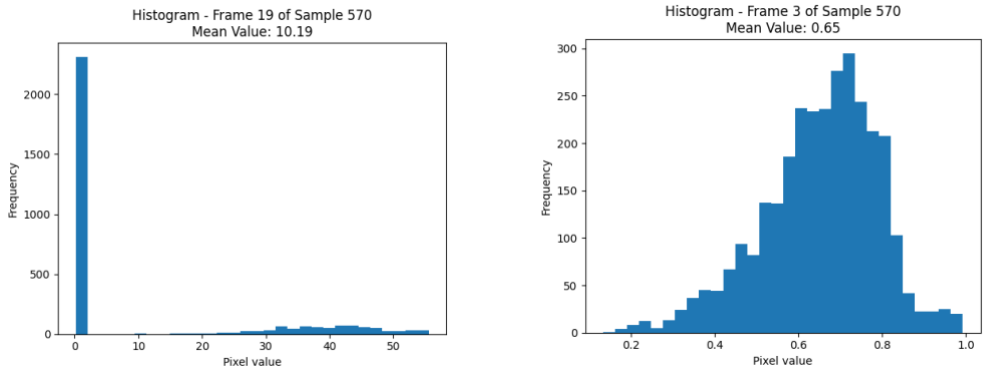


Figure 22. Histograms of a frame with a circular mask and an intensity factor of 71.63 (left) and a frame with no mask (right) from the testing dataset

4.5. Models

Due to the increased complexity of classifying color natural movies compared to grayscale handwritten digit movies, the residual neural networks were used to enhance the classification performance. Residual networks introduce skip connections to allow the model to learn residual functions. These connections enable the model to effectively address the problem of vanishing gradients in deep neural networks.

In this study, a ResNet50 model, previously trained on a different dataset, was employed for the classification of the CIFAR-10 dataset. ResNet 50 is composed of 50 residual layers including convolutional layers, pooling layers and fully connected layers. Additionally, a ResNet20 model, which was not pre-trained, was used for the classification of the same dataset. The Photoreceptor model was combined with both pre-trained (ResNet50) and non-pre-trained (ResNet20) models which shows its flexibility and compatibility with diverse architectures.

In order to assess the performance of the PR+ResNet50 and PR+ResNet20 models, several evaluation approaches were employed, including:

1. Comparing the integration of all color channels together and process them, versus processing each color channel separately.
2. Exploring the impact of applying a normalization layer before and after concatenating the color channels.
3. Evaluating the effectiveness of both time-distributed and non-time-distributed layer normalization techniques.

Also, the true positive rate was employed as an evaluation metric to evaluate the performance of these models. The true positive rate was determined by comparing the predicted labels with the actual labels for each frame in a movie. When all predicted labels matched the corresponding actual labels, it was considered a true positive.

4.5.1. Pre-trained Residual Neural Network (ResNet50)

The following diagram shows a time-distributed ResNet50-based architecture with additional trainable layers. By incorporating the concept of time-distribution, the model gains the ability to process sequential data (refer to section 9.1.1 of the PRModel_CIFAR10 notebook).
In Appendix of the report, you will find detailed information about the ResNet50 model including architectural components, layer configuration, shapes, etc.

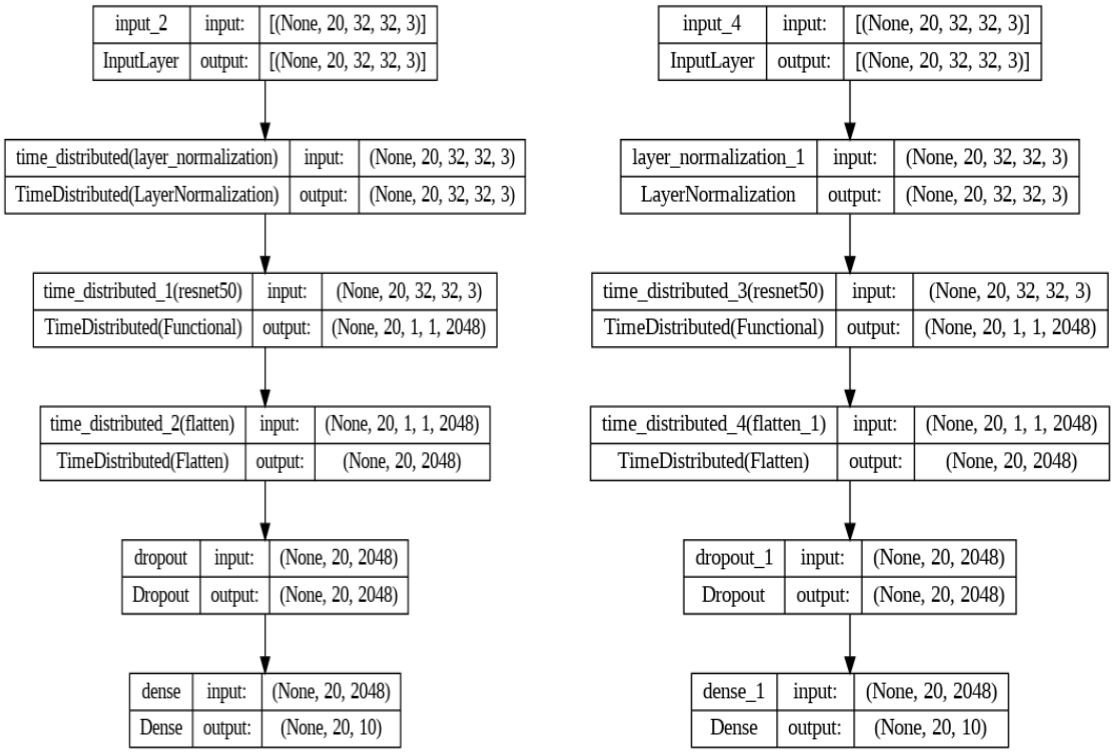


Figure 23. Time-distributed ResNet50 model architecture with time-distributed LayerNorm (left) and non-time distributed LayerNorm (right)

With the implementation of the time-distributed layer norm, the true positive rate was 55.45%. On the other hand, utilizing the non-time-distributed layer norm resulted in a higher true positive rate of 58.45%.

4.5.2. Color-Independent Model: ResNet 50+PR (R+G+B)

The input to the model is a movie comprising 20 frames, with each frame consisting of three-color channels. Initially, the Photoreceptor model processes each color channel individually. The outputs obtained from the PR model for each channel are then combined using a concatenation operation. This concatenation operation combines information from all color channels, enabling further processing within the ResNet50 architecture which expects inputs to have three color channels (refer to section 9.1.2 of the PRModel_CIFAR10 notebook).

To incorporate multiple channels in the PR model, it is necessary to add a 2D convolutional layer to reduce the number of channels to 3. This reduction step ensures compatibility with the pre-trained ResNet50 model.

After selectively modifying the intensity values of some of the pixels, the images were not normalized before being passed to the Photoreceptor model. Therefore, LayerNorm was applied to normalize the intensity values of the pixels after concatenating the color channels. The normalized output of LayerNorm was used as the input for the time-distributed ResNet50 model.

4.5.2.1. Time-Independent Layer Normalization after the Concatenation Layer

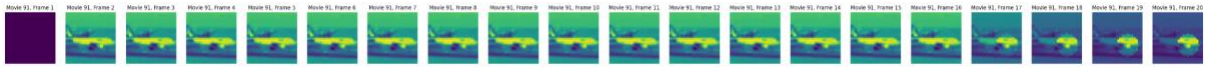
The following diagram shows the integration of the Photoreceptor model with the time-distributed ResNet50 model. In this case, LayerNorm was applied after concatenating the colors.

Figure 24. Time-distributed ResNet50+PR(R+G+B) model architecture with one channel (left) and 20 channels (right)

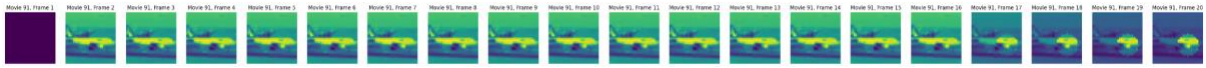
To visually capture the modifications occurring in each layer and to demonstrate the Photoreceptor model's effectiveness in removing masks for individual colors and the combination of all colors, the outputs of each channel were observed before and after the normalization layer.

Pre-Normalization

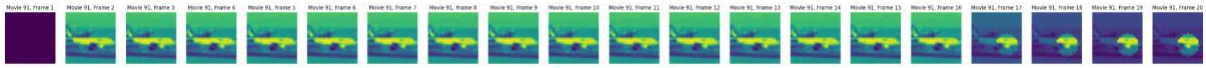
Channel #1=Red



Channel #2=Green



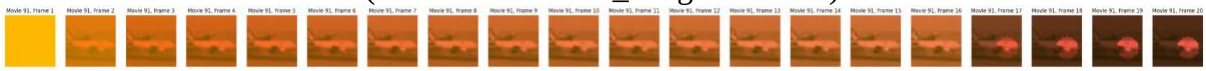
Channel #3=Blue



To improve the visualization of the output information, a manual normalization was applied to each frame of the output.

Combination of All Three Channels

After manual normalization (refer to normalize_image function)



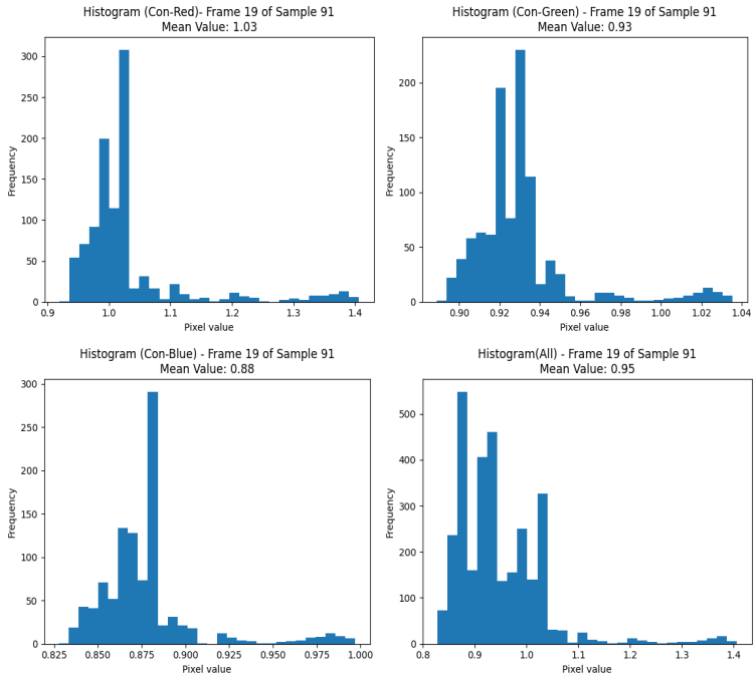
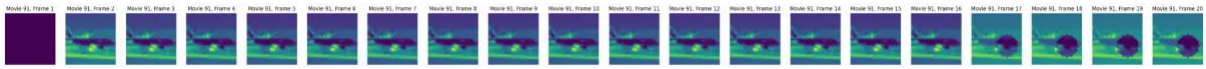


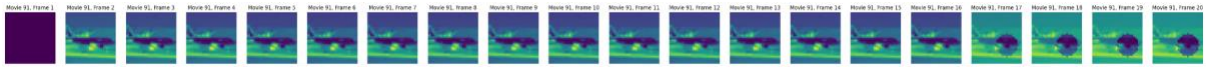
Figure 25. Histograms of a masked image for each color channel and all channels combined before the normalization layer

Post-Normalization

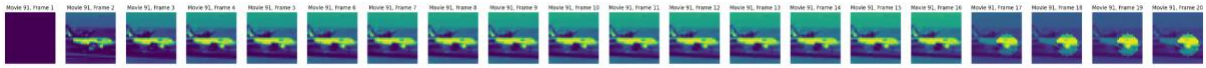
Channel #1=Red



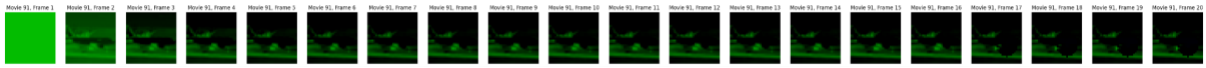
Channel #2=Green



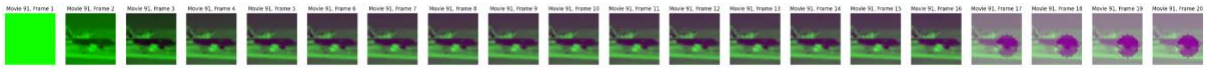
Channel #3=Blue



Combination of All Three Channels



After manual normalization



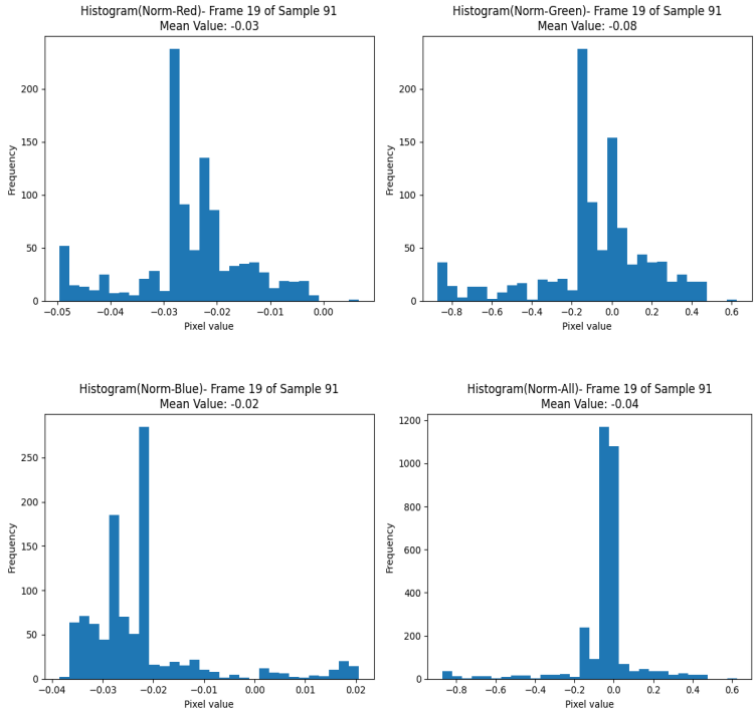


Figure 26. Histograms of a masked image for each color channel and all channels combined after the normalization layer

Due to the edge effect, the first frame of the output generated by the PR model appears as a blank frame. Consequently, when calculating the true positive rate, the comparison between predicted labels and true labels was performed for frames 2 to 20 (the first frame was excluded). As a result, the true positive rate for this model was determined to be 57.34%. In addition, when the number of channels in the PR model was increased to 20, the true positive rate was determined to be 56.66%. Despite the additional channels, the model's performance did not improve.

4.5.2.2. [Time-Distributed Layer Normalization after the Concatenation Layer]

****Verify the results of this model once again****

To achieve independent normalization for each color channel, Time-Distributed LayerNorm was applied along the width and height dimensions (-3,-2) after concatenating the color channels as shown in the following diagrams:

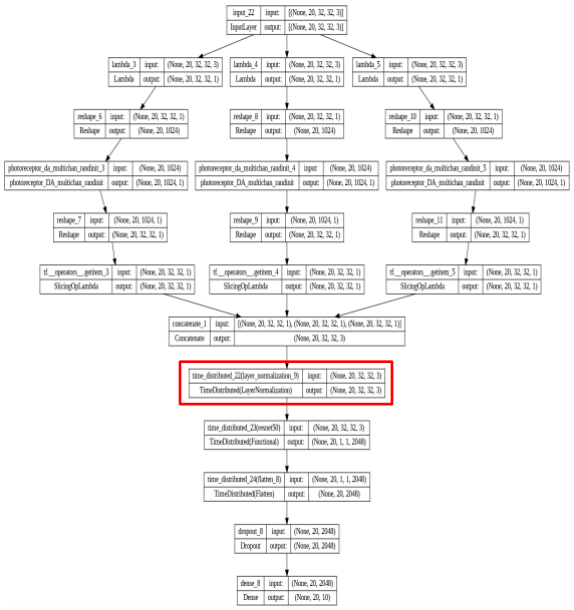
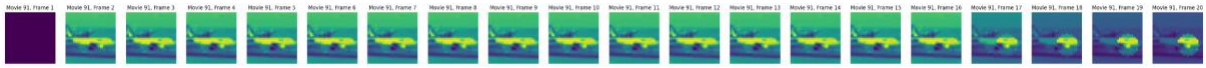


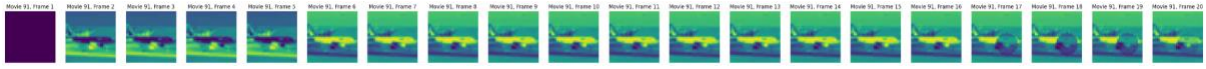
Figure 27. Time-distributed ResNet50+PR(R+G+B) model architecture with one channel

Pre-Normalization

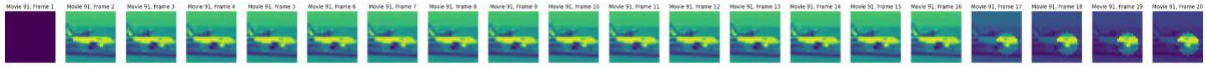
Channel #1=Red



Channel #2=Green

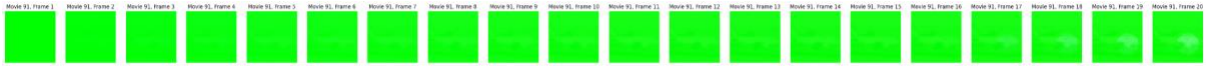


Channel #3=Blue



Combination of All Three Channels

After manual normalization



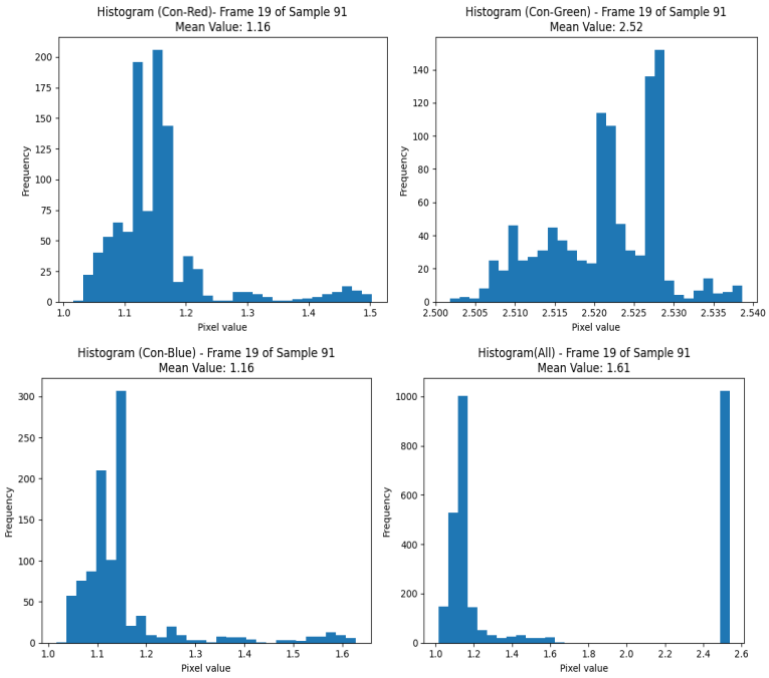
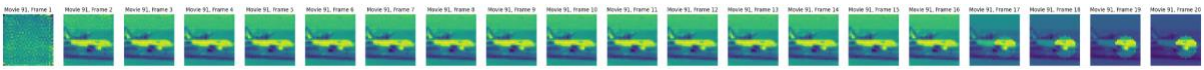


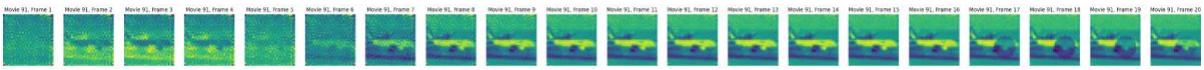
Figure 28. Histograms of a masked image for each color channel and all channels combined before the normalization layer

Post-Normalization

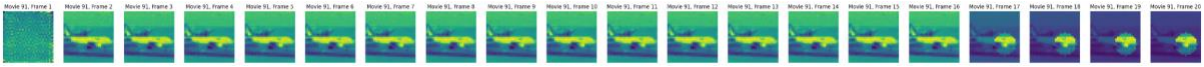
Channel #1=Red



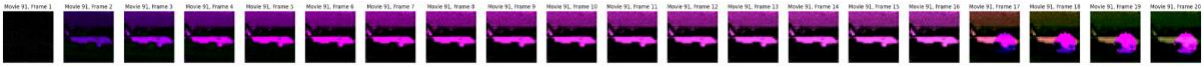
Channel #2=Green



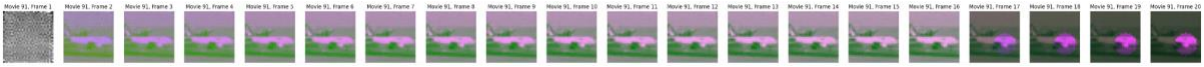
Channel #3=Blue

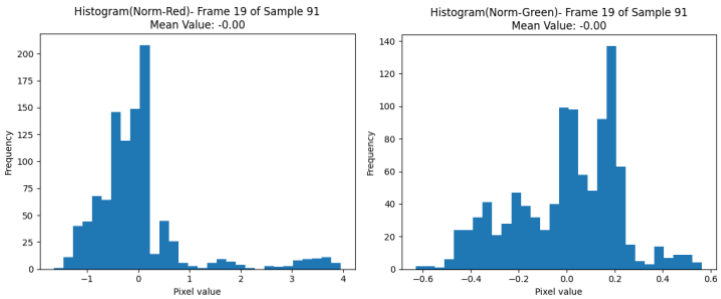


Combination of All Three Channels



After manual normalization





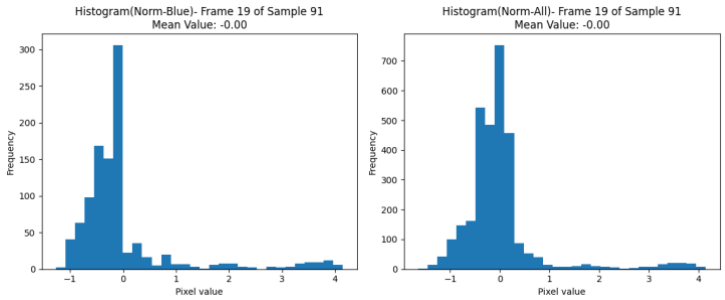


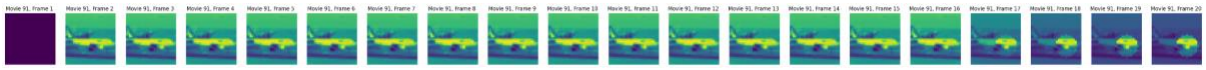
Figure 29. Histograms of a masked image for each color channel and all channels combined after the normalization layer

The true positive rate was determined to be 56.99%. In order to enhance the model's performance, the number of channels of the PR model was increased to 20. As a result, the true positive rate improved significantly to 66.14%

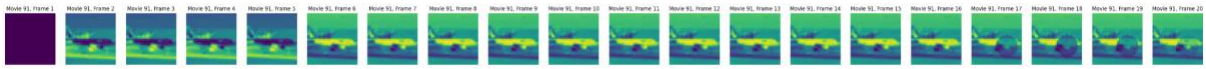
The following diagrams show the individual color channels of the output generated by the PR model both before and after the concatenation process. As expected, it can be observed that the output frames and histogram plots are identical, confirming the preservation of information throughout the concatenation step.

Pre-Concatenation

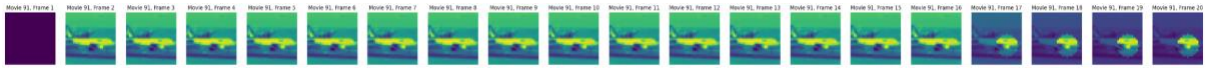
Channel #1=Red



Channel #2=Green



Channel #2=Blue



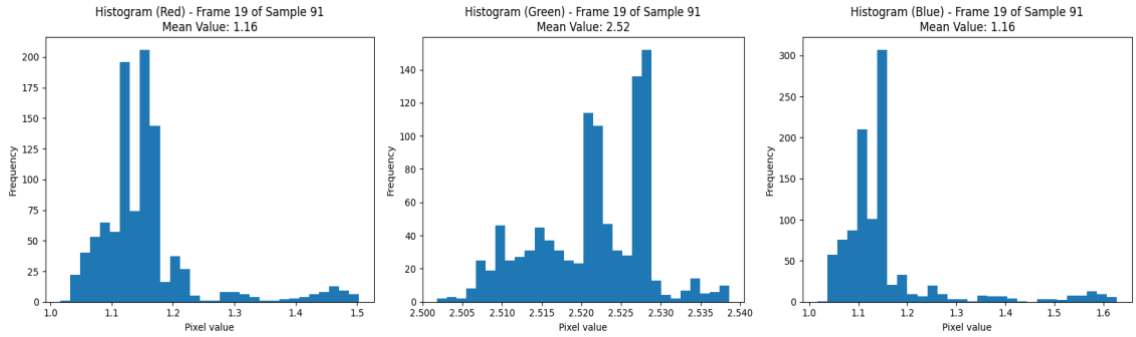
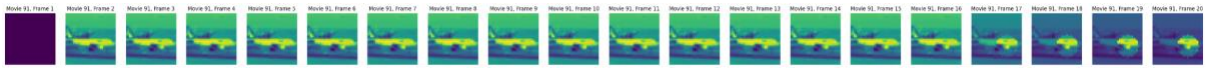


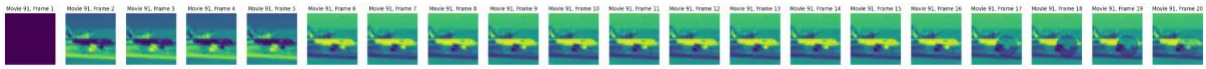
Figure 30. Histograms of a masked image for each color channel before concatenation

Post-Concatenation

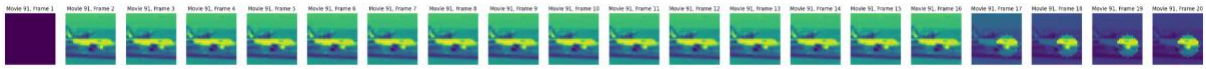
Channel #1=Red



Channel #2=Green



Channel #3=Blue



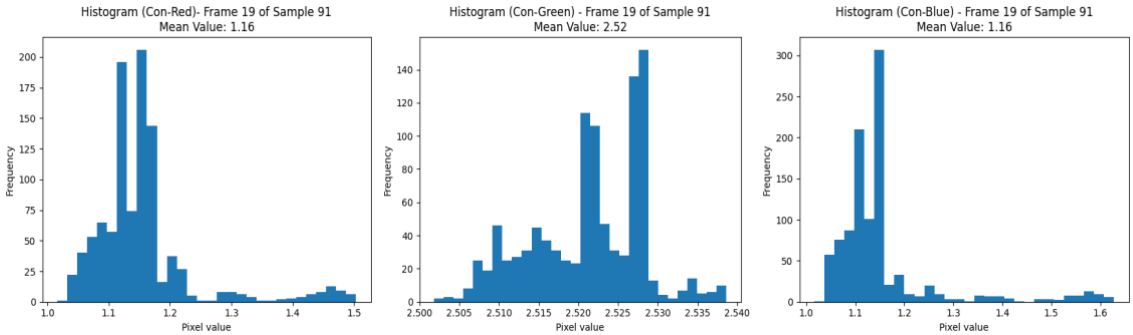


Figure 31. Histograms of a masked image for each color channel after concatenation

4.5.2.3. [Time-Distributed Layer Normalization Before the Concatenation Layer]

****Verify the results of this model once again****

Time-distributed LayerNorm was separately applied to each color along the height and width dimensions (-3,-2) before the concatenation layer as shown in the following diagrams:

Figure 32. Time-distributed ResNet50+PR(R+G+B) model architecture with one channel

Figure 33. Time-distributed ResNet50+PR(R+G+B) model architecture with 20 channels

The initial true positive rate was 57.11%. To improve the model's performance, the number of PR model channels increased from 1 to 20. Despite this modification, the performance did not show improvement and resulted in a true positive rate of 54.69%.

4.5.3. Integrated Color Model: ResNet 50+PR (RGB)

The Photoreceptor model used for grayscale movies, was modified to handle color movies. The part of the code that was for reshaping the input tensor was removed (refer to Functions_PRModel section):

```
    # y_tf_reshape = tf.reshape(y_tf,(-1,y_tf.shape[1],y_tf.shape[2],3,tau_z.shape[-1]))
    # z_tf_reshape = tf.reshape(z_tf,(-1,z_tf.shape[1],z_tf.shape[2],3,tau_z.shape[-1]))
```

The new shape is specified as (frames, width*height*color):

```
# reshape the input to PR (frame_num=20, width*height*colour=3072)
y1 = Reshape((inputs.shape [1],inputs.shape [-2]*inputs.shape [-
1]*inputs.shape [-3]))(inputs)
```

Also, another reshape layer was used to transform the output tensor to the shape of (sample, frames, width, height, color*channels):

```
y1 = Reshape((inputs.shape[1],inputs.shape[-2],inputs.shape[-
3],chan1_n*inputs.shape [-1]))(y1)
```

In this model, the three-color channels were combined with the photoreceptor (PR) channel numbers, so the PR model did not process each color channel separately (refer to section 9.1.3 of the PRModel_CIFAR10 notebook)

4.5.3.1. Time-Distributed Layer Normalization after the PR Layer

In the ResNet50+PR(RGB) model, all three channels are combined and the photoreceptor doesn't process each channel separately. In this case, time-distributed LayerNorm was applied to the output of the PR model as shown in the diagrams below:

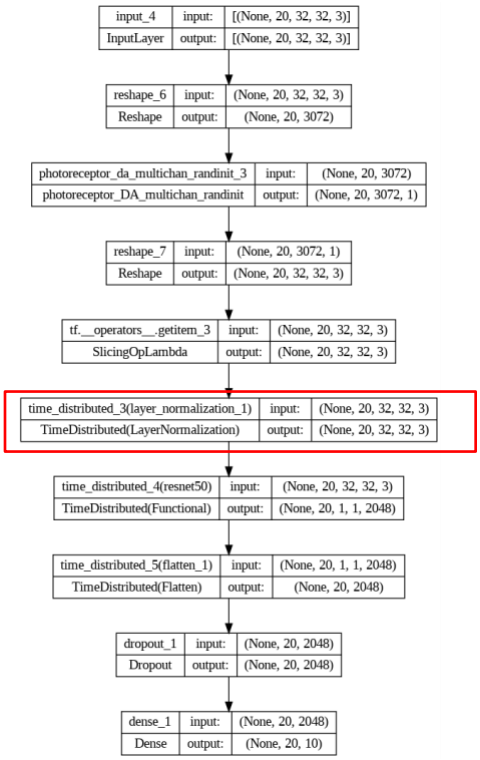
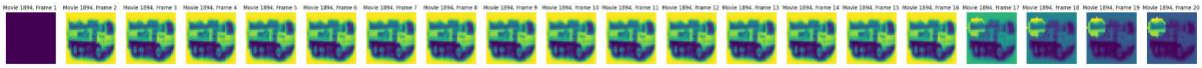


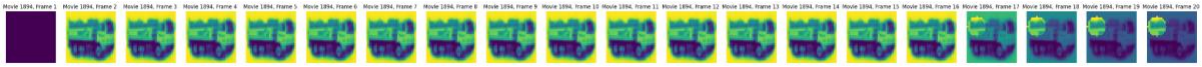
Figure 34. Time-distributed ResNet50+PR(RGB) model architecture with one channel (left)

Pre-Normalization

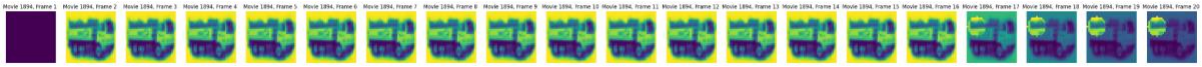
Channel #1=Red



Channel #2=Green



Channel #3=Blue



Combination of All Three Channels

After manual normalization

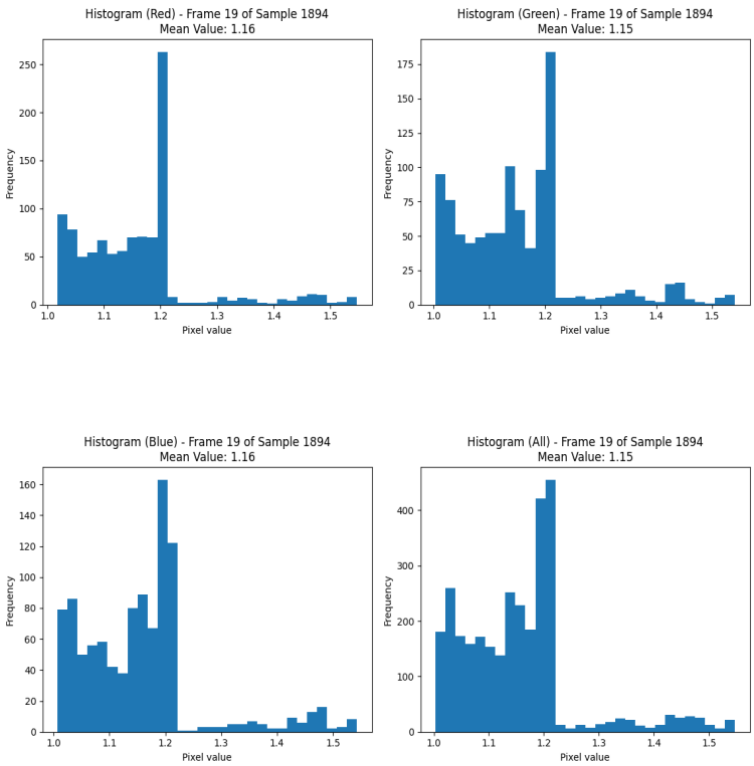
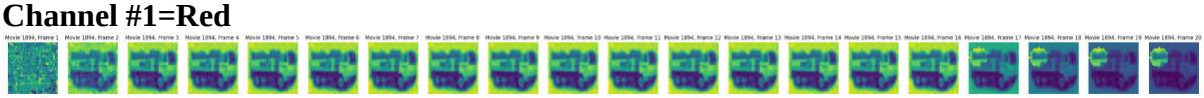


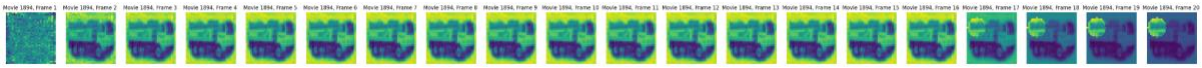
Figure 35. Histograms of a masked image for each color channel and all channels combined before the normalization layer

Post-Normalization

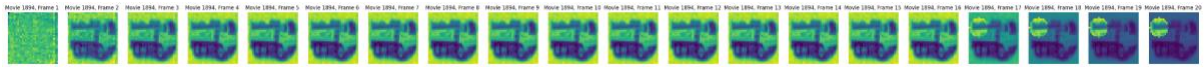
Channel #1=Red



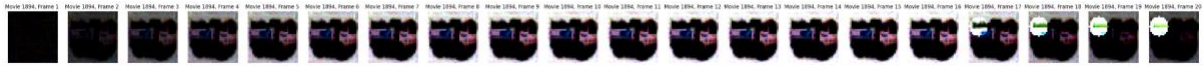
Channel #2=Green



Channel #3=Blue



Combination of All Three Channels



After manual normalization

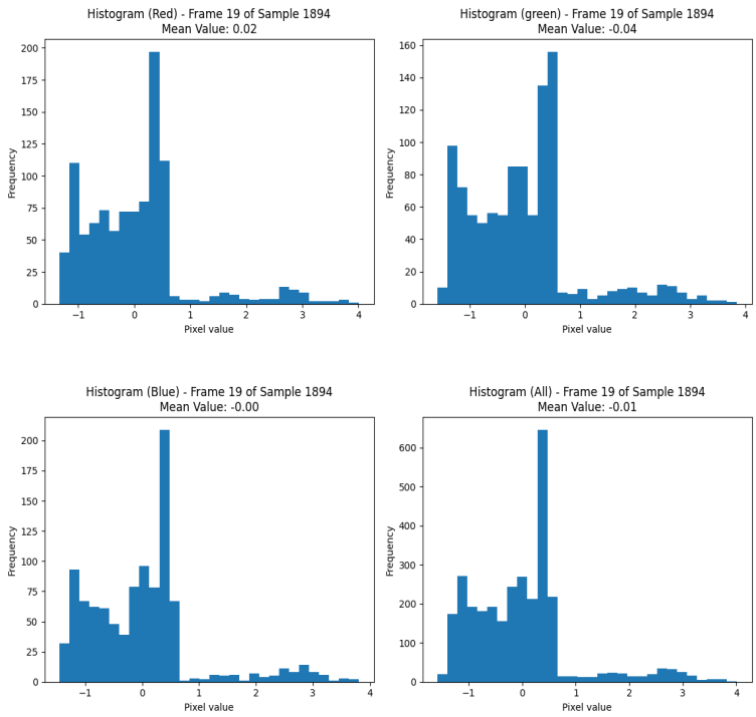


Figure 36. Histograms of a masked image for each color channel and all channels combined after the normalization layer

The initial true positive rate was 55.18 % . When the number of channels was increased to 20, the performance of the model did not improve and the true positive rate declined to 38.75%.

4.5.3.2. Time-Independent Layer Normalization after the PR Layer

The diagram below shows the architectural of the integration of the PR model with the Time-Distributed ResNet50 model. Time-distributed LayerNorm was applied to the output of the PR model as shown in the diagrams below:

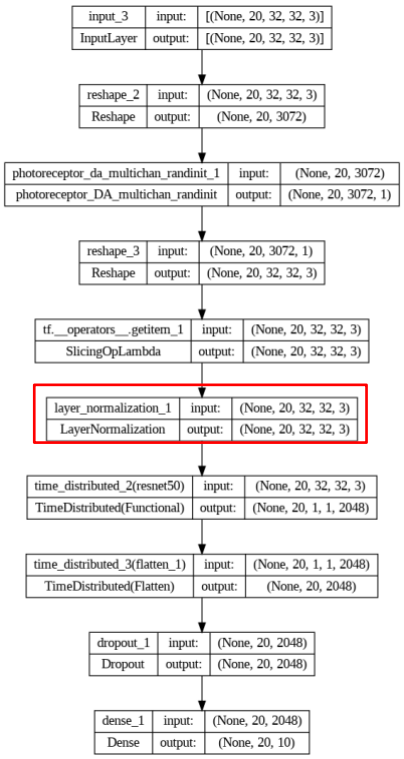
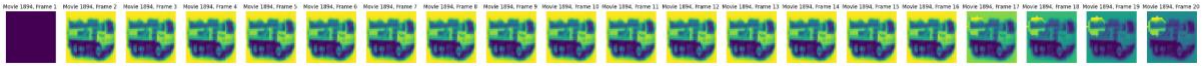


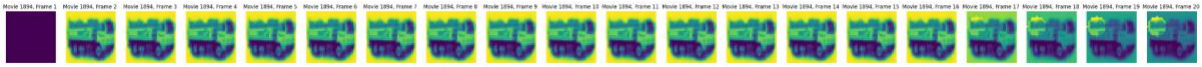
Figure 37. Time-distributed ResNet50+PR(RGB) model architecture with one channel

Pre-Normalization

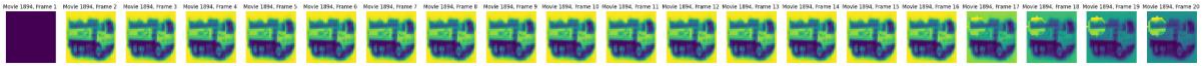
Channel #1=Red



Channel #2=Green



Channel #3=Blue



Combination of All Three Channels

After manual normalization

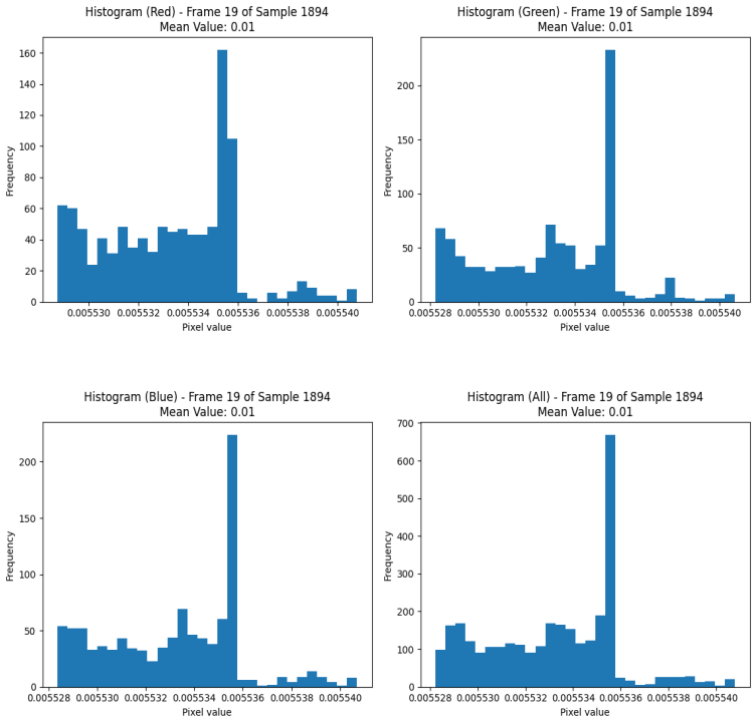
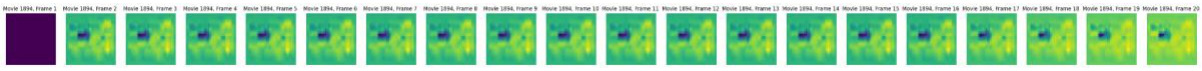


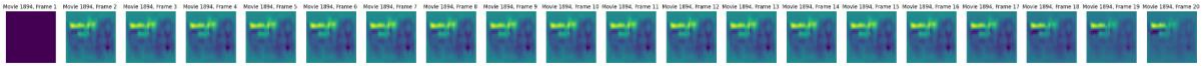
Figure 38. Histograms of a masked image for each color channel and all channels combined before the normalization layer

Post-Normalization

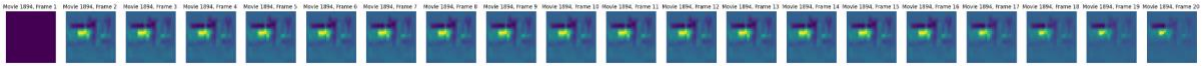
Channel #1=Red



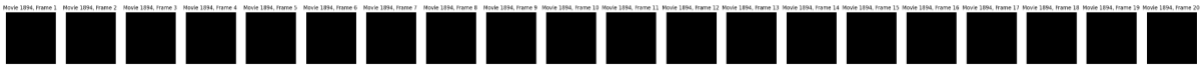
Channel #2=Green



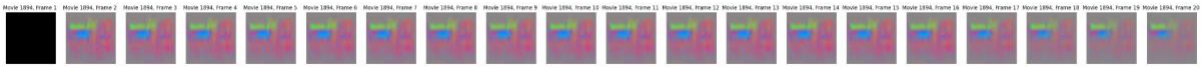
Channel #3=Blue



Combination of All Three Channels



After manual normalization



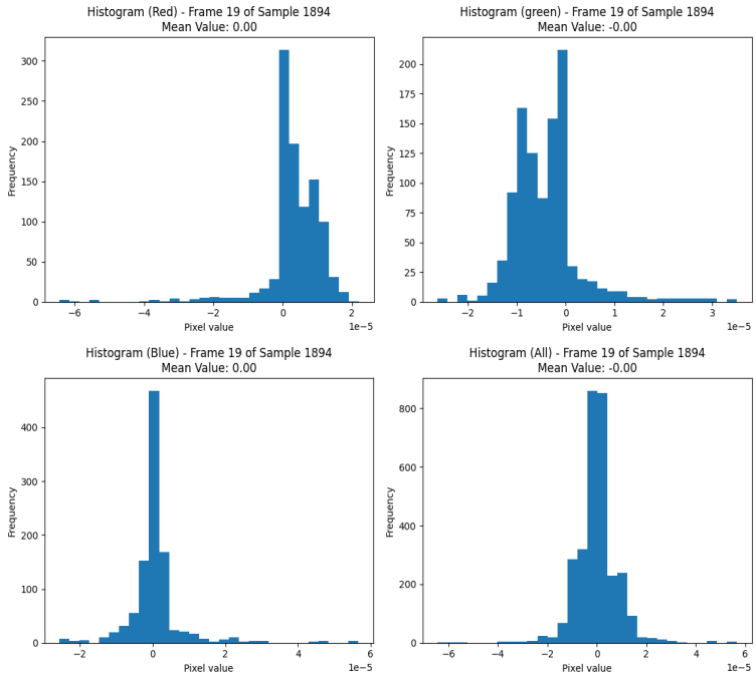


Figure 39. Histograms of a masked image for each color channel and all channels combined after the normalization layer

When utilizing a single channel in the PR model, the true positive rate was 46.84%. Also, some variations in the output images were noted when non-time-distributed LayerNorm was applied.

4.5.4. Non-Pre-trained Model (ResNet20)

In addition to the pre-trained ResNet50 model, another variant of the residual neural network, ResNet20, was used in this study. Pre-trained residual neural networks are designed to accept input with only three channels (red, blue, green color channels). However, in this case, the number of photoreceptor channels was multiplied by the color channels, resulting in a total of 60 channels (with 20 channels for PR) in the last dimension. As ResNet20 is not pre-trained so it can handle inputs with varying shapes.

ResNet20 has two major versions: version 1 and version 2. Each version introduces certain architectural changes and improvements over the original ResNet design. These changes include modifications in skip connections, normalization technique, and the overall network structure. ResNet20, version 2 was used in this study.

Value of depth, which is the input to the model, is determined by the user. To obtain the depth value for each version and model using the following equations:

Depth_v1= n*6+2
Depth_v2=n*9+2

The following table shows the corresponding “n” values that should be used to calculate depth values.

Model	n v1(v2)	sec/epoch v1 (v2)	Depth-v1	Depth-v2
ResNet20	3 (2)	35 (NA)	20	20
ResNet32	5 (NA)	50 (NA)	32	–
ResNet44	7 (NA)	70 (NA)	45	–
ResNet56	9 (6)	90 (100)	57	56
ResNet110	18 (12)	165 (180)	111	110
ResNet164	27 (18)	–	165	164
ResNet1001	(111)	–	–	1001

Table 3. Depth values for different ResNet models

The ResNet20 architecture consists of the following layers:

1. Convolution Layer: The model starts with Conv2D layer with 16 filters and kernel size of 3. The stride is set to 1 allowing for feature extraction at the original resolution.
2. Batch Normalization Layer: Following the convolution layer, batch normalization layer was applied to achieve better stability.
3. ReLU Activation Layer: After batch normalization, a rectified linear unit (ReLU) activation layer is employed. ReLU introduces non-linearity to the network to help learning complex features from the input data.

The model proceeds with three stages (stage 0, stage 1 and stage 2) which downsample the input image and extract useful features. Feature map sizes for these stages are as follows:

- Stage 0: 32x32x64
- Stage 1: 16x16x128
- Stage 2: 8x8x256

After the third stage, batch normalization is applied one again. The normalization step helps in maintaining stable gradients during training. Another activation layer is applied after the normalization layer. The average maxpooling with a pool size of 8 is applied to reduce the spatial dimension of the data. consequently, the output size changes from (20,2048) to (20,2048/8=256). Then a flatten layer is used to transform the multidimensional output into a flat vector (refer to section 9.2 of the PRModel_CIFAR10 notebook).

In order to train all the ResNet model, a learning rate schedule was used to adjust the learning rate during training. This approach aims to find an optimal learning rate for the model to achieve better convergence and performance. In this case, learning rate decreased as follows:

Epochs	Learning Rate (lr)
0-19	10-3
20-49	10-5
50-99	10-6
100-139	10-7
>140	10-8

Table 4. Learning rates values using a learning rate scheduler

The following diagram shows a custom ResNet20-based architecture with TimeDistributed layers and additional trainable layers (refer to section 9.2.3 of the PRModel_CIFAR10 notebook). In Appendix of the report, you will find detailed information about the ResNet20 model including architectural components, layer configuration, shapes, etc.

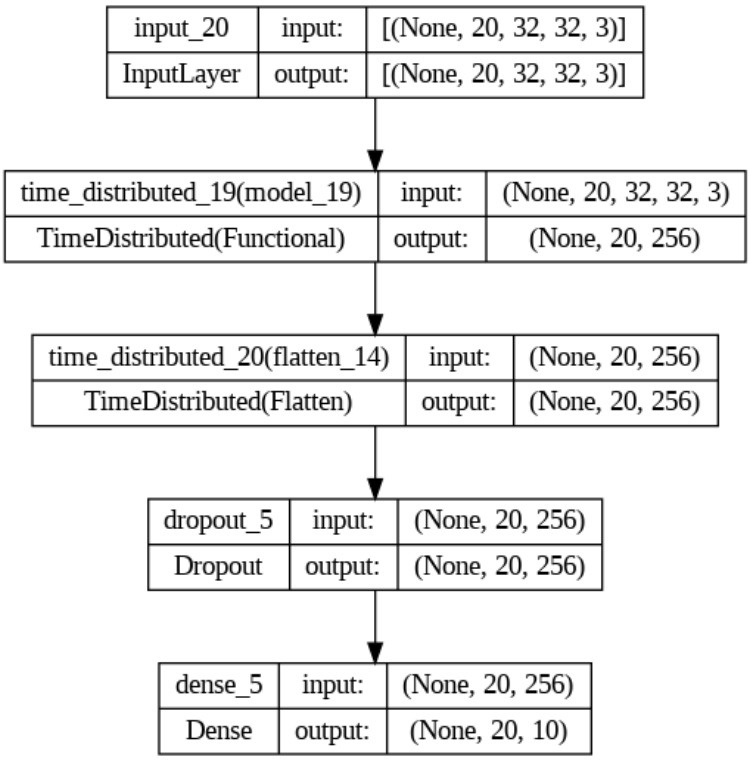


Figure 40. Time-distributed ResNet20 model architecture

The true positive rate, without the PR model, was determined to be 59.6%.

4.5.5. Color-Independent Model: ResNet 20+PR (R+G+B)

4.5.5.1. Time-Distributed Layer Normalization before the Concatenation Layer

The LayerNorm was removed from the original Resnet_v2 model. Instead, LayerNorm was integrated into the custom architecture of ResNet20 model (refer to section 9.2.2 of the PRModel_CIFAR10 notebook). In this case, each color channel was processed separately as shown below:

Figure 41. Time-distributed ResNet20+PR(R+G+B) model architecture

The model demonstrated a marginal enhancement in performance, and the true positive rate was 59.96%.

4.5.6. Integrated Color Model: ResNet 20+PR (RGB)

4.5.6.1. Time-Distributed Layer Normalization after the PR Layer

The LayerNorm was removed from the original Resnet_v2 model. Instead, LayerNorm was integrated into the custom architecture of ResNet20 model (refer to section 9.2.1 of the PRModel_CIFAR10 notebook). In this case, all color channels were combined as shown below:

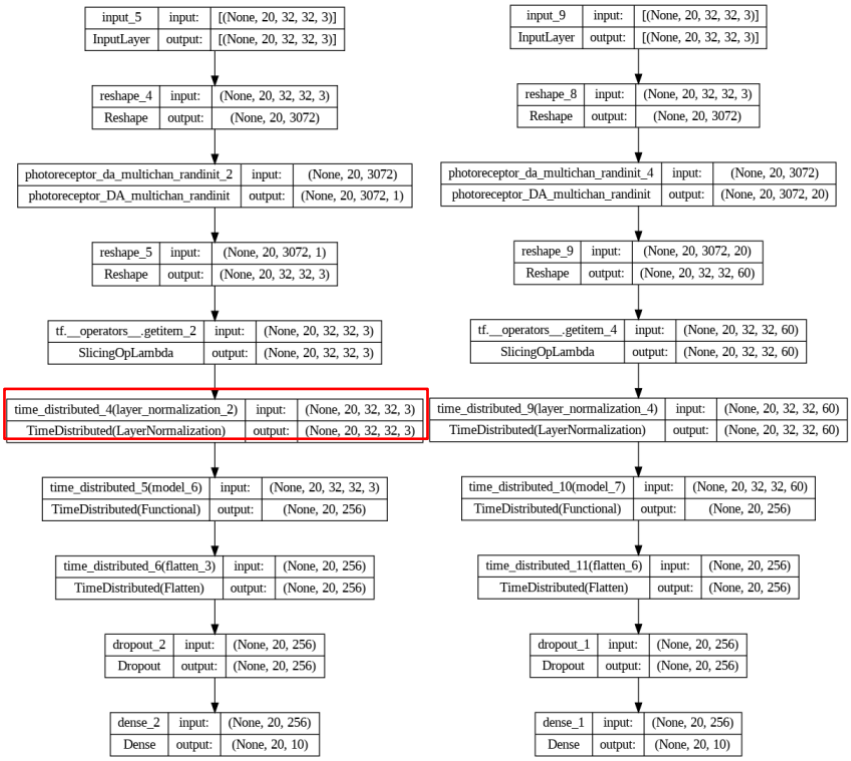
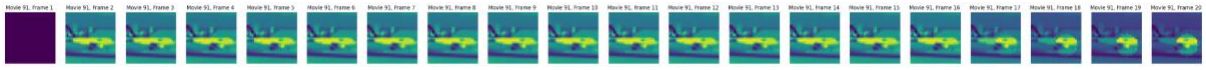


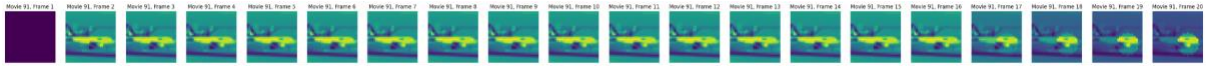
Figure 41. Time-distributed ResNet20+PR(RGB) model architecture with one channel (left) and 20 channels (right)

Pre-Normalization

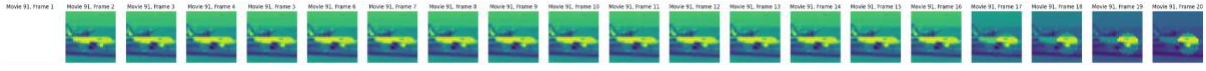
Channel #1=Red



Channel #2=Green



Channel #3=Blue



Combination of All Three Channels

After manual normalization

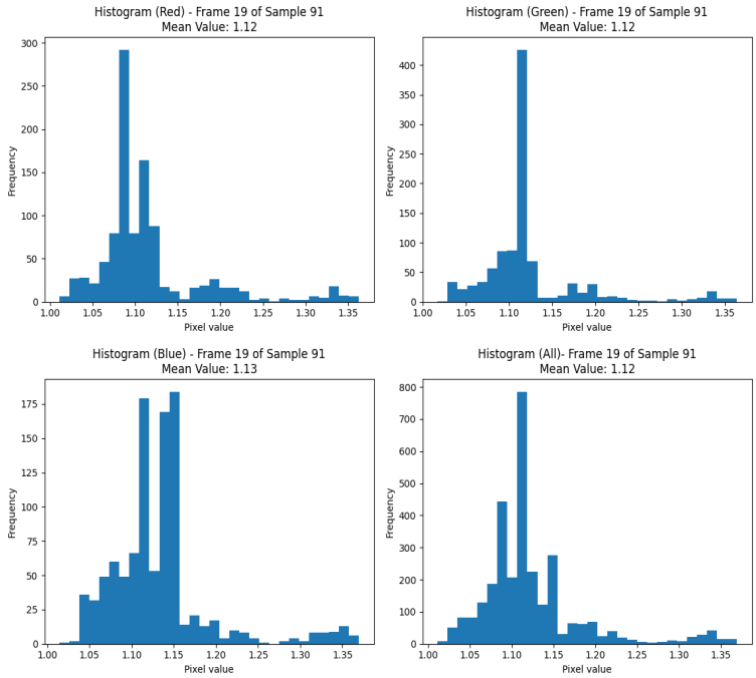
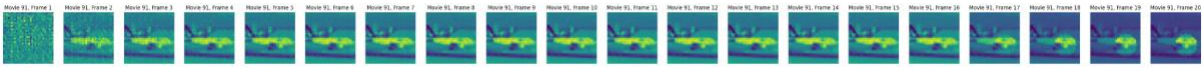


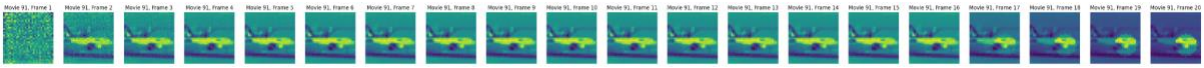
Figure 42. Histograms of a masked image for each color channel and all channels combined before the normalization layer

Post-Normalization

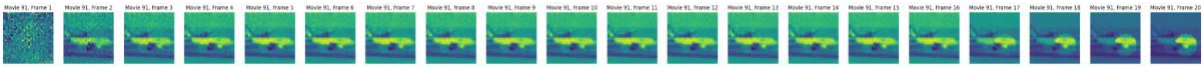
Channel #1=Red



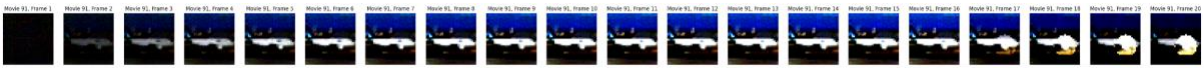
Channel #2=Green



Channel #1=Blue



Combination of All Three Channels



After manual normalization

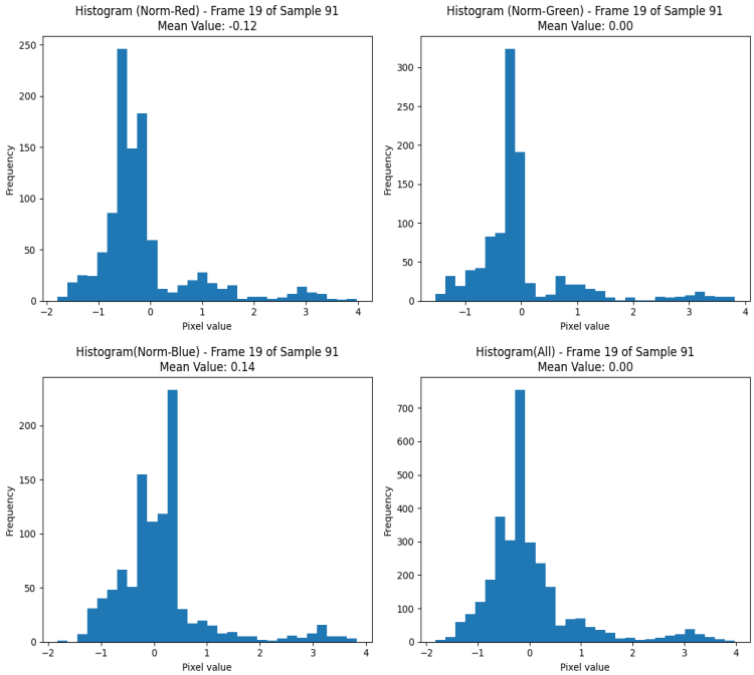


Figure 43. Histograms of a masked image for each color channel and all channels combined after the normalization layer

By comparing the true labels and predicted labels, it was observed that using one channel resulted in a true positive rate of 63.48%. Despite increasing the number of channels to 20, there was no improvement in the classification performance and the true positive rate was 62.88%

4.6. Photoreceptor Model Parameters Optimization

In order to find the optimal weight values, the models were trained using masked images, enabling them to learn optimal values over time. In the models where each color channel was processed independently, three sets of weight values were obtained. However, for the model that all channels were combined, there is only one set of weight values.

PR Parameter	**Multiplication Factor**	**ResNet20+PR (three colors combined)**	**ResNet50+PR (distinct color analysis)-2**		
			Ch 1(Red)	Ch2 (Blue)	Ch3 (Green)
ζ	1000	0.02001	0.03090	0.03015	0.04988
K	1000	0.02000	0.02023	0.02000	0.02004
α	100	0.05000	0.04971	0.05000	0.04348
β	10	0.05000	0.03343	0.04891	0.04988
γ	10	0.02000	0.02001	0.02000	0.02000
τ_y	100	0.04999	0.02000	0.02000	0.02000
n_y	10	0.03732	0.03573	0.02000	0.02000
τ_z	100	0.02000	0.04994	0.04684	0.03147
n_z	10	0.03944	0.04997	0.05000	0.05000

Table 5. Optimal weight values

4.7. Results

In order to evaluate the models, the true positive rate which is a measure based on the comparison between true and predicted labels for each frame was used. All the models were tested under uniform conditions including using the same learning rate (as per a defined learning rate schedule) and a consistent number of epochs (70).

The following plot shows a comparative analysis of the true positive rates among three models with the base ResNet20 using a single channel for the PR model:

1. ResNet20 with a Time-Distributed layer normalization

2. A combined model of Photoreceptor and ResNet20 (distinct color analysis) with a Time-Distributed layer normalization before concatenation.
3. A combined model of Photoreceptor and ResNet20 (three colors combined) with a Time-Distributed layer normalization after the PR layer.

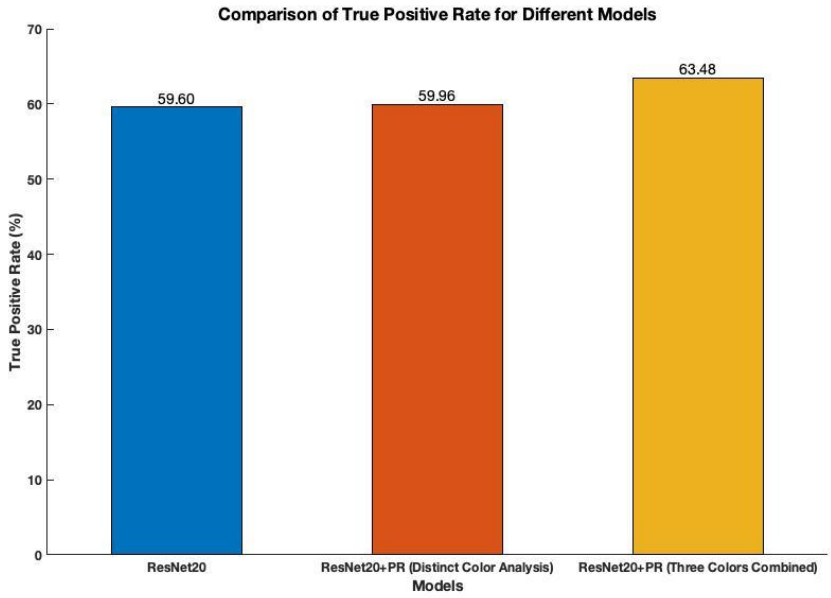


Figure 44. Comparison of true positive rate for ResNet50, ResNet50+PR(R+G+B) and ResNet50(RBG)

The findings indicate that the performance of the PR+ResNet20 (three colors combined) model is 6.51% better than the base ResNet20 model which lacks the dynamic adaptive layer.

The following plot shows a comparative analysis of the true positive rates among three models with the base ResNet50 using 20 channels for the PR model:

1. ResNet50 with a Time-Distributed layer normalization
2. A combined model of Photoreceptor and ResNet50 (distinct color analysis) with a Time-Distributed layer normalization before concatenation.
3. A combined model of Photoreceptor and ResNet50 (distinct color analysis) with a Time-Distributed layer normalization after concatenation.
4. A combined model of Photoreceptor and ResNet50 (three colors combined) with a Time-Distributed layer normalization after the PR layer.

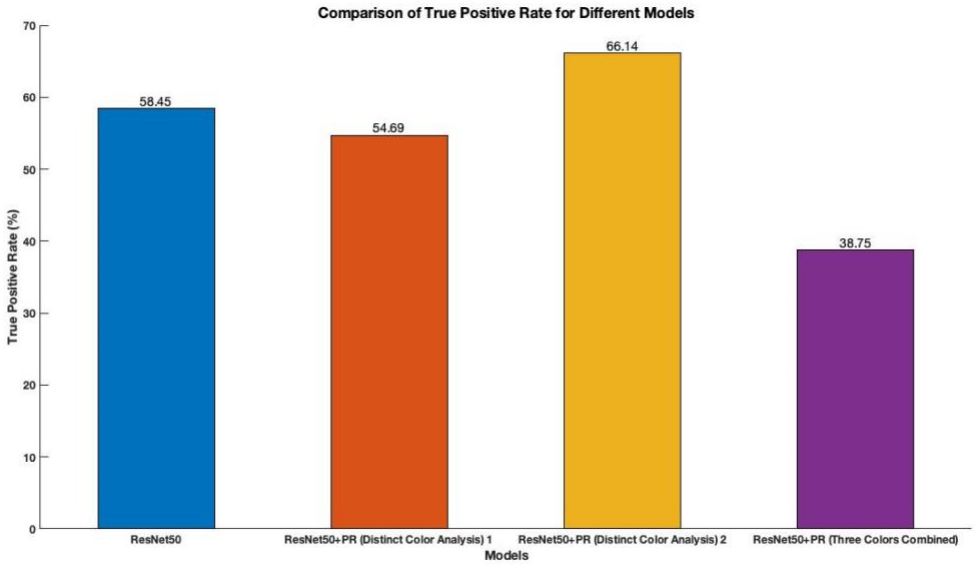


Figure 45. Comparison of true positive rate for ResNet50, ResNet50+PR(R+G+B) and ResNet50(RBG)

The findings indicate that the performance of the PR+ResNet50 (Distinct color analysis) model is 13.16% better than the base ResNet50 model which lacks the dynamic adaptive layer.

The following diagrams show that the ResNet50 base model misclassified the frog immediately after a change in intensity. It can be observed that changing the intensity of the circular mask negatively impacts the accuracy of image classification.

4.8. Discussion

In the case where all color channels are combined (ResNet20+PR(RGB) & ResNet50+PR(RGB)), time-distributed layer norm should be used after the PR layer. Interestingly, the performance of the ResNet20+PR(RGB) model was better than ResNet50+PR(RGB)) model. On the other hand, for the models where each color channel was processed separately, better results were achieved by employing time-distributed LayerNorm after concatenating color channels.

The following summary tables show the true positive rate for all the models:

ResNet50		**ResNet50+PR(R+B+G)**								**ResNet50+PR(RBG)**			
Time-distributed-LayerNorm	Time-independent LayerNorm	Time-distributed LayerNorm				Time-independent LayerNorm				Time-dis LN		Time-ind LN	
55.45	58.45	After Concat Layer		Before Concat Layer		After Concat Layer		Before Concat Layer		Ch=1	Ch=20	Ch=1	Ch=20
		Ch=1	Ch=20	Ch=1	Ch=20	Ch=1	Ch=20	Ch=1	Ch=20	55.18	38.75	46.84	NA
		56.99	66.14	57.11	54.69	57.34	56.66	NA	NA				

Table 6. True Positive Rates using the ResNet50 base model

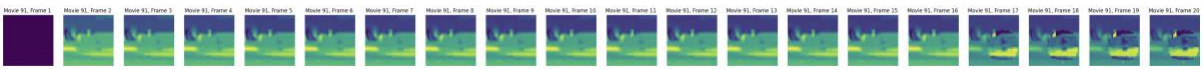
ResNet20		**ResNet20+PR(R+B+G)**								**ResNet20+PR(RBG)**			
Time-distributed-LayerNorm	Time-independent LayerNorm	Time-distributed LayerNorm				Time-independent LayerNorm				Time-dis LN		Time-ind LN	
59.6	NA	After Concat Layer		Before Concat Layer		After Concat Layer		Before Concat Layer		Ch=1	Ch=20	Ch=1	Ch=20
		Ch=1	Ch=20	Ch=1	Ch=20	Ch=1	Ch=20	Ch=1	Ch=20	63.48	62.88	NA	Na
		NA	NA	59.96	NA	NA	NA	NA	NA				

Table 7. True Positive Rates using the ResNet20 base model

4.9. Technical Learnings

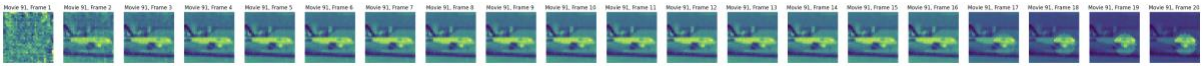
Applying non-time-distributed LayerNorm to the output of the PR in both the ResNet20+PR(RGB) and ResNet50+PR(RGB) models without specifying the axis resulted in incorrect normalization. This misalignment can be seen in the following diagram:

Red Channel



However, the issue was resolved by employing time-distributed LayerNorm:

Red Channel



4.10. Future Work

To further test the models with the CIFAR-10 dataset, several potential tasks can be done to continue and expand upon this research project:

1. Optimize the photoreceptor parameters using the Root Mean Square (RMS) error approach and then exclude masked images from the training phase.
2. Conduct an extended evaluation by running the models over a longer period of time.
3. Evaluate the performance of other pre-trained models, such as VGG16 and EfficientNet (refer to….).
4. Apply masks to randomly selected frames (refer to section 6.0 from the PRModel_CIFAR10 notebook).
5. After modifying the duration of the mask, increase the number of channels in the PR model to assess the potential impact on the its performance.
6. Evaluate the impact of removing the first few frames from the output of the PR model to eliminate the edge effect and determine whether excluding them improves the model's performance.

5. Conclusion

By using well-known datasets such as MNIST and CIFAR-10, we successfully demonstrated the significant enhancement in classification capability of CNN models under dynamic lighting conditions by incorporating the dynamic adaptive (DA) layer into the CNN models. This is a preliminary investigation, highlighting the effectiveness of the PR layer. To further evaluate the PR model's performance, future studies should involve testing with more realistic natural movies as input.

This research will make a significant contribution to the field of computer vision and enhance our understanding of how to develop more robust object detection models that can operate effectively under dynamic lighting conditions. These models can be used in different applications such as smart glasses for visually impaired people and autonomous vehicles.

6. References

[1] Kolb, H., Fernandez, E., & Nelson, R.(2003). The Organization of the Retina and Visual System. Webvision: The Organization of the Retina and Visual System.

[2] Stanford . (n.d.). Retrieved July 5, 2023, from https://stanford.edu/class/ee267/lectures/lecture5.pdf

[3] BIO254:Phototransduction - OpenWetWare. (n.d.). BIO254:Phototransduction - OpenWetWare.https://openwetware.org/wiki/BIO254:Phototransduction#:~:text=Visual%20phototransduction%20occurs%20in%20the,the%20photoreceptor%20onto%20downstream%20neurons

[4] Idrees. S., Field, G.D., Rieke. F., & Zylberberg. J. (2023). A new photoreceptor-inspired (CNN) Layer Enables Deep Learning Models of Retina to Generalize Across Lighting Conditions.

[5] Clark, D. A., Benichou, R., Meister, M., & Azeredo da Silveira, R. (2013). Dynamical Adaptation in Photoreceptors. PLoS Computational Biology, 9(11).

7. APPENDIX

7.1. ResNet20 Summary

Model:ResNet 20

Layer (type)	Output Shape	Param #	Connected to
input_15 (InputLayer)	[(None, 32, 32, 60)]	0	[]
layer_normalization_6 (LayerNormalization)	**(None, 32, 32, 60)**	**120**	**['input_15[0][0]']**
conv2d_132 (Conv2D)	(None, 32, 32, 16)	8656	['layer_normalization_6[0][0]']
batch_normalization_114 (BatchNormalization)	(None, 32, 32, 16)	64	['conv2d_132[0][0]']
activation_114 (Activation)	(None, 32, 32, 16)	0	['batch_normalization_114[0][0]']
conv2d_133 (Conv2D)	(None, 32, 32, 16)	272	['activation_114[0][0]']
batch_normalization_115 (BatchNormalization)	(None, 32, 32, 16)	64	['conv2d_133[0][0]']
activation_115 (Activation)	(None, 32, 32, 16)	0	['batch_normalization_115[0][0]']
conv2d_134 (Conv2D)	(None, 32, 32, 16)	2320	['activation_115[0][0]']
batch_normalization_116 (BatchNormalization)	(None, 32, 32, 16)	64	['conv2d_134[0][0]']
activation_116 (Activation)	(None, 32, 32, 16)	0	['batch_normalization_116[0][0]']
conv2d_136 (Conv2D)	(None, 32, 32, 64)	1088	['activation_114[0][0]']
conv2d_135 (Conv2D)	(None, 32, 32, 64)	1088	['activation_116[0][0]']
add_36 (Add)	(None, 32, 32, 64)	0	['conv2d_136[0][0]', 'conv2d_135[0][0]']
batch_normalization_117 (BatchNormalization)	(None, 32, 32, 64)	256	['add_36[0][0]']
activation_117 (Activation)	(None, 32, 32, 64)	0	['batch_normalization_117[0][0]']
conv2d_137 (Conv2D)	(None, 32, 32, 16)	1040	['activation_117[0][0]']
batch_normalization_118 (BatchNormalization)	(None, 32, 32, 16)	64	['conv2d_137[0][0]']

activation_118 (Activation) (None, 32, 32, 16) 0 ['batch_normalization_118[0][0]']

conv2d_138 (Conv2D) (None, 32, 32, 16) 2320 ['activation_118[0][0]']

batch_normalization_119 (Batch Normalization) (None, 32, 32, 16) 64 ['conv2d_138[0][0]']

activation_119 (Activation) (None, 32, 32, 16) 0 ['batch_normalization_119[0][0]']

conv2d_139 (Conv2D) (None, 32, 32, 64) 1088 ['activation_119[0][0]']

add_37 (Add) (None, 32, 32, 64) 0 ['add_36[0][0]', 'conv2d_139[0][0]']

batch_normalization_120 (Batch Normalization) (None, 32, 32, 64) 256 ['add_37[0][0]']

activation_120 (Activation) (None, 32, 32, 64) 0 ['batch_normalization_120[0][0]']

conv2d_140 (Conv2D) (None, 16, 16, 64) 4160 ['activation_120[0][0]']

batch_normalization_121 (Batch Normalization) (None, 16, 16, 64) 256 ['conv2d_140[0][0]']

activation_121 (Activation) (None, 16, 16, 64) 0 ['batch_normalization_121[0][0]']

conv2d_141 (Conv2D) (None, 16, 16, 64) 36928 ['activation_121[0][0]']

batch_normalization_122 (Batch Normalization) (None, 16, 16, 64) 256 ['conv2d_141[0][0]']

activation_122 (Activation) (None, 16, 16, 64) 0 ['batch_normalization_122[0][0]']

conv2d_143 (Conv2D) (None, 16, 16, 128) 8320 ['add_37[0][0]']

conv2d_142 (Conv2D) (None, 16, 16, 128) 8320 ['activation_122[0][0]']

add_38 (Add) (None, 16, 16, 128) 0 ['conv2d_143[0][0]', 'conv2d_142[0][0]']

batch_normalization_123 (Batch Normalization) (None, 16, 16, 128) 512 ['add_38[0][0]']

activation_123 (Activation) (None, 16, 16, 128) 0 ['batch_normalization_123[0][0]']

conv2d_144 (Conv2D) (None, 16, 16, 64) 8256 ['activation_123[0][0]']

batch_normalization_124 (Batch Normalization) (None, 16, 16, 64) 256 ['conv2d_144[0][0]']

activation_124 (Activation) (None, 16, 16, 64) 0 ['batch_normalization_124[0][0]']

conv2d_145 (Conv2D) (None, 16, 16, 64) 36928 ['activation_124[0][0]']

batch_normalization_125 (Batch Normalization) (None, 16, 16, 64) 256 ['conv2d_145[0][0]']

activation_125 (Activation) (None, 16, 16, 64) 0 ['batch_normalization_125[0][0]']

conv2d_146 (Conv2D) (None, 16, 16, 128) 8320 ['activation_125[0][0]']

add_39 (Add) (None, 16, 16, 128) 0 ['add_38[0][0]',
'conv2d_146[0][0]']

batch_normalization_126 (Batch (None, 16, 16, 128) 512 ['add_39[0][0]']
Normalization)

activation_126 (Activation) (None, 16, 16, 128) 0 ['batch_normalization_126[0][0]']

conv2d_147 (Conv2D) (None, 8, 8, 128) 16512 ['activation_126[0][0]']

batch_normalization_127 (Batch (None, 8, 8, 128) 512 ['conv2d_147[0][0]']
Normalization)

activation_127 (Activation) (None, 8, 8, 128) 0 ['batch_normalization_127[0][0]']

conv2d_148 (Conv2D) (None, 8, 8, 128) 147584 ['activation_127[0][0]']

batch_normalization_128 (Batch (None, 8, 8, 128) 512 ['conv2d_148[0][0]']
Normalization)

activation_128 (Activation) (None, 8, 8, 128) 0 ['batch_normalization_128[0][0]']

conv2d_150 (Conv2D) (None, 8, 8, 256) 33024 ['add_39[0][0]']

conv2d_149 (Conv2D) (None, 8, 8, 256) 33024 ['activation_128[0][0]']

add_40 (Add) (None, 8, 8, 256) 0 ['conv2d_150[0][0]',
'conv2d_149[0][0]']

batch_normalization_129 (Batch (None, 8, 8, 256) 1024 ['add_40[0][0]']
Normalization)

activation_129 (Activation) (None, 8, 8, 256) 0 ['batch_normalization_129[0][0]']

conv2d_151 (Conv2D) (None, 8, 8, 128) 32896 ['activation_129[0][0]']

batch_normalization_130 (Batch (None, 8, 8, 128) 512 ['conv2d_151[0][0]']
Normalization)

activation_130 (Activation) (None, 8, 8, 128) 0 ['batch_normalization_130[0][0]']

conv2d_152 (Conv2D) (None, 8, 8, 128) 147584 ['activation_130[0][0]']

batch_normalization_131 (Batch (None, 8, 8, 128) 512 ['conv2d_152[0][0]']
Normalization)

activation_131 (Activation) (None, 8, 8, 128) 0 ['batch_normalization_131[0][0]']

conv2d_153 (Conv2D) (None, 8, 8, 256) 33024 ['activation_131[0][0]']

add_41 (Add) (None, 8, 8, 256) 0 ['add_40[0][0]',
'conv2d_153[0][0]']

batch_normalization_132 (Batch (None, 8, 8, 256) 1024 ['add_41[0][0]']
Normalization)

activation_132 (Activation) (None, 8, 8, 256) 0 ['batch_normalization_132[0][0]']

average_pooling2d_6 (AveragePo (None, 1, 1, 256) 0 ['activation_132[0][0]']
oling2D)

flatten_9 (Flatten) (None, 256) 0 ['average_pooling2d_6[0][0]']

==========

Total params: 579,848
Trainable params: 576,360
Non-trainable params: 3,488

7.2. ResNet50 Summary

Model: ResNet50
Model: "resnet50"

Layer (type) Output Shape Param # Connected to

==========

input_3 (InputLayer) [(None, 32, 32, 3)] 0 []

conv1_pad (ZeroPadding2D) (None, 38, 38, 3) 0 ['input_3[0][0]']

conv1_conv (Conv2D) (None, 16, 16, 64) 9472 ['conv1_pad[0][0]']

conv1_bn (BatchNormalization) (None, 16, 16, 64) 256 ['conv1_conv[0][0]']

conv1_relu (Activation) (None, 16, 16, 64) 0 ['conv1_bn[0][0]']

pool1_pad (ZeroPadding2D) (None, 18, 18, 64) 0 ['conv1_relu[0][0]']

pool1_pool (MaxPooling2D) (None, 8, 8, 64) 0 ['pool1_pad[0][0]']

conv2_block1_1_conv (Conv2D) (None, 8, 8, 64) 4160 ['pool1_pool[0][0]']

conv2_block1_1_bn (BatchNormal (None, 8, 8, 64) 256 ['conv2_block1_1_conv[0][0]']
ization)

conv2_block1_1_relu (Activatio (None, 8, 8, 64) 0 ['conv2_block1_1_bn[0][0]']
n)

conv2_block1_2_conv (Conv2D) (None, 8, 8, 64) 36928 ['conv2_block1_1_relu[0][0]']

conv2_block1_2_bn (BatchNormal (None, 8, 8, 64) 256 ['conv2_block1_2_conv[0][0]']
ization)

conv2_block1_2_relu (Activatio (None, 8, 8, 64) 0 ['conv2_block1_2_bn[0][0]']
n)

conv2_block1_0_conv (Conv2D) (None, 8, 8, 256) 16640 ['pool1_pool[0][0]']

conv2_block1_3_conv (Conv2D) (None, 8, 8, 256) 16640 ['conv2_block1_2_relu[0][0]']

conv2_block1_0_bn (BatchNormal (None, 8, 8, 256) 1024 ['conv2_block1_0_conv[0][0]']
ization)

conv2_block1_3_bn (BatchNormal (None, 8, 8, 256) 1024 ['conv2_block1_3_conv[0][0]']
ization)

conv2_block1_add (Add) (None, 8, 8, 256) 0 ['conv2_block1_0_bn[0][0]',
 'conv2_block1_3_bn[0][0]']

conv2_block1_out (Activation) (None, 8, 8, 256) 0 ['conv2_block1_add[0][0]']

conv2_block2_1_conv (Conv2D) (None, 8, 8, 64) 16448 ['conv2_block1_out[0][0]']

conv2_block2_1_bn (BatchNormal (None, 8, 8, 64) 256 ['conv2_block2_1_conv[0][0]']
ization)

conv2_block2_1_relu (Activatio (None, 8, 8, 64) 0 ['conv2_block2_1_bn[0][0]']
n)

conv2_block2_2_conv (Conv2D) (None, 8, 8, 64) 36928 ['conv2_block2_1_relu[0][0]']

conv2_block2_2_bn (BatchNormal (None, 8, 8, 64) 256 ['conv2_block2_2_conv[0][0]']
ization)

conv2_block2_2_relu (Activatio (None, 8, 8, 64) 0 ['conv2_block2_2_bn[0][0]']
n)

conv2_block2_3_conv (Conv2D) (None, 8, 8, 256) 16640 ['conv2_block2_2_relu[0][0]']

conv2_block2_3_bn (BatchNormal (None, 8, 8, 256) 1024 ['conv2_block2_3_conv[0][0]']
ization)

conv2_block2_add (Add) (None, 8, 8, 256) 0 ['conv2_block1_out[0][0]',
 'conv2_block2_3_bn[0][0]']

conv2_block2_out (Activation) (None, 8, 8, 256) 0 ['conv2_block2_add[0][0]']

conv2_block3_1_conv (Conv2D) (None, 8, 8, 64) 16448 ['conv2_block2_out[0][0]']

conv2_block3_1_bn (BatchNormal (None, 8, 8, 64) 256 ['conv2_block3_1_conv[0][0]']
ization)

conv2_block3_1_relu (Activatio (None, 8, 8, 64) 0 ['conv2_block3_1_bn[0][0]']
n)

conv2_block3_2_conv (Conv2D) (None, 8, 8, 64) 36928 ['conv2_block3_1_relu[0][0]']

conv2_block3_2_bn (BatchNormal (None, 8, 8, 64) 256 ['conv2_block3_2_conv[0][0]']
ization)

conv2_block3_2_relu (Activatio (None, 8, 8, 64) 0 ['conv2_block3_2_bn[0][0]']
n)

conv2_block3_3_conv (Conv2D) (None, 8, 8, 256) 16640 ['conv2_block3_2_relu[0][0]']

conv2_block3_3_bn (BatchNormal (None, 8, 8, 256) 1024 ['conv2_block3_3_conv[0][0]']
ization)

conv2_block3_add (Add) (None, 8, 8, 256) 0 ['conv2_block2_out[0][0]',
'conv2_block3_3_bn[0][0]']

conv2_block3_out (Activation) (None, 8, 8, 256) 0 ['conv2_block3_add[0][0]']

conv3_block1_1_conv (Conv2D) (None, 4, 4, 128) 32896 ['conv2_block3_out[0][0]']

conv3_block1_1_bn (BatchNormal (None, 4, 4, 128) 512 ['conv3_block1_1_conv[0][0]']
ization)

conv3_block1_1_relu (Activatio (None, 4, 4, 128) 0 ['conv3_block1_1_bn[0][0]']
n)

conv3_block1_2_conv (Conv2D) (None, 4, 4, 128) 147584 ['conv3_block1_1_relu[0][0]']

conv3_block1_2_bn (BatchNormal (None, 4, 4, 128) 512 ['conv3_block1_2_conv[0][0]']
ization)

conv3_block1_2_relu (Activatio (None, 4, 4, 128) 0 ['conv3_block1_2_bn[0][0]']
n)

conv3_block1_0_conv (Conv2D) (None, 4, 4, 512) 131584 ['conv2_block3_out[0][0]']

conv3_block1_3_conv (Conv2D) (None, 4, 4, 512) 66048 ['conv3_block1_2_relu[0][0]']

conv3_block1_0_bn (BatchNormal (None, 4, 4, 512) 2048 ['conv3_block1_0_conv[0][0]']
ization)

conv3_block1_3_bn (BatchNormal (None, 4, 4, 512) 2048 ['conv3_block1_3_conv[0][0]']
ization)

conv3_block1_add (Add) (None, 4, 4, 512) 0 ['conv3_block1_0_bn[0][0]',
'conv3_block1_3_bn[0][0]']

conv3_block1_out (Activation) (None, 4, 4, 512) 0 ['conv3_block1_add[0][0]']

conv3_block2_1_conv (Conv2D) (None, 4, 4, 128) 65664 ['conv3_block1_out[0][0]']

conv3_block2_1_bn (BatchNormal (None, 4, 4, 128) 512 ['conv3_block2_1_conv[0][0]']
ization)

conv3_block2_1_relu (Activatio (None, 4, 4, 128) 0 ['conv3_block2_1_bn[0][0]']
n)

conv3_block2_2_conv (Conv2D) (None, 4, 4, 128) 147584 ['conv3_block2_1_relu[0][0]']

conv3_block2_2_bn (BatchNormal (None, 4, 4, 128) 512 ['conv3_block2_2_conv[0][0]']

ization)

conv3_block2_2_relu (Activatio (None, 4, 4, 128) 0 ['conv3_block2_2_bn[0][0]']
n)

conv3_block2_3_conv (Conv2D) (None, 4, 4, 512) 66048 ['conv3_block2_2_relu[0][0]']

conv3_block2_3_bn (BatchNormal (None, 4, 4, 512) 2048 ['conv3_block2_3_conv[0][0]']
ization)

conv3_block2_add (Add) (None, 4, 4, 512) 0 ['conv3_block1_out[0][0]',
'conv3_block2_3_bn[0][0]']

conv3_block2_out (Activation) (None, 4, 4, 512) 0 ['conv3_block2_add[0][0]']

conv3_block3_1_conv (Conv2D) (None, 4, 4, 128) 65664 ['conv3_block2_out[0][0]']

conv3_block3_1_bn (BatchNormal (None, 4, 4, 128) 512 ['conv3_block3_1_conv[0][0]']
ization)

conv3_block3_1_relu (Activatio (None, 4, 4, 128) 0 ['conv3_block3_1_bn[0][0]']
n)

conv3_block3_2_conv (Conv2D) (None, 4, 4, 128) 147584 ['conv3_block3_1_relu[0][0]']

conv3_block3_2_bn (BatchNormal (None, 4, 4, 128) 512 ['conv3_block3_2_conv[0][0]']
ization)

conv3_block3_2_relu (Activatio (None, 4, 4, 128) 0 ['conv3_block3_2_bn[0][0]']
n)

conv3_block3_3_conv (Conv2D) (None, 4, 4, 512) 66048 ['conv3_block3_2_relu[0][0]']

conv3_block3_3_bn (BatchNormal (None, 4, 4, 512) 2048 ['conv3_block3_3_conv[0][0]']
ization)

conv3_block3_add (Add) (None, 4, 4, 512) 0 ['conv3_block2_out[0][0]',
'conv3_block3_3_bn[0][0]']

conv3_block3_out (Activation) (None, 4, 4, 512) 0 ['conv3_block3_add[0][0]']

conv3_block4_1_conv (Conv2D) (None, 4, 4, 128) 65664 ['conv3_block3_out[0][0]']

conv3_block4_1_bn (BatchNormal (None, 4, 4, 128) 512 ['conv3_block4_1_conv[0][0]']
ization)

conv3_block4_1_relu (Activatio (None, 4, 4, 128) 0 ['conv3_block4_1_bn[0][0]']
n)

conv3_block4_2_conv (Conv2D) (None, 4, 4, 128) 147584 ['conv3_block4_1_relu[0][0]']

conv3_block4_2_bn (BatchNormal (None, 4, 4, 128) 512 ['conv3_block4_2_conv[0][0]']
ization)

conv3_block4_2_relu (Activatio (None, 4, 4, 128) 0 ['conv3_block4_2_bn[0][0]']
n)

conv3_block4_3_conv (Conv2D) (None, 4, 4, 512) 66048 ['conv3_block4_2_relu[0][0]']

conv3_block4_3_bn (BatchNormalization) (None, 4, 4, 512) 2048 ['conv3_block4_3_conv[0][0]']

conv3_block4_add (Add) (None, 4, 4, 512) 0 ['conv3_block3_out[0][0]', 'conv3_block4_3_bn[0][0]']

conv3_block4_out (Activation) (None, 4, 4, 512) 0 ['conv3_block4_add[0][0]']

conv4_block1_1_conv (Conv2D) (None, 2, 2, 256) 131328 ['conv3_block4_out[0][0]']

conv4_block1_1_bn (BatchNormalization) (None, 2, 2, 256) 1024 ['conv4_block1_1_conv[0][0]']

conv4_block1_1_relu (Activation) (None, 2, 2, 256) 0 ['conv4_block1_1_bn[0][0]']

conv4_block1_2_conv (Conv2D) (None, 2, 2, 256) 590080 ['conv4_block1_1_relu[0][0]']

conv4_block1_2_bn (BatchNormalization) (None, 2, 2, 256) 1024 ['conv4_block1_2_conv[0][0]']

conv4_block1_2_relu (Activation) (None, 2, 2, 256) 0 ['conv4_block1_2_bn[0][0]']

conv4_block1_0_conv (Conv2D) (None, 2, 2, 1024) 525312 ['conv3_block4_out[0][0]']

conv4_block1_3_conv (Conv2D) (None, 2, 2, 1024) 263168 ['conv4_block1_2_relu[0][0]']

conv4_block1_0_bn (BatchNormalization) (None, 2, 2, 1024) 4096 ['conv4_block1_0_conv[0][0]']

conv4_block1_3_bn (BatchNormalization) (None, 2, 2, 1024) 4096 ['conv4_block1_3_conv[0][0]']

conv4_block1_add (Add) (None, 2, 2, 1024) 0 ['conv4_block1_0_bn[0][0]', 'conv4_block1_3_bn[0][0]']

conv4_block1_out (Activation) (None, 2, 2, 1024) 0 ['conv4_block1_add[0][0]']

conv4_block2_1_conv (Conv2D) (None, 2, 2, 256) 262400 ['conv4_block1_out[0][0]']

conv4_block2_1_bn (BatchNormalization) (None, 2, 2, 256) 1024 ['conv4_block2_1_conv[0][0]']

conv4_block2_1_relu (Activation) (None, 2, 2, 256) 0 ['conv4_block2_1_bn[0][0]']

conv4_block2_2_conv (Conv2D) (None, 2, 2, 256) 590080 ['conv4_block2_1_relu[0][0]']

conv4_block2_2_bn (BatchNormalization) (None, 2, 2, 256) 1024 ['conv4_block2_2_conv[0][0]']

conv4_block2_2_relu (Activation) (None, 2, 2, 256) 0 ['conv4_block2_2_bn[0][0]']

n)

conv4_block2_3_conv (Conv2D) (None, 2, 2, 1024) 263168 ['conv4_block2_2_relu[0][0]']

conv4_block2_3_bn (BatchNormal (None, 2, 2, 1024) 4096 ['conv4_block2_3_conv[0][0]']
ization)

conv4_block2_add (Add) (None, 2, 2, 1024) 0 ['conv4_block1_out[0][0]',
'conv4_block2_3_bn[0][0]']

conv4_block2_out (Activation) (None, 2, 2, 1024) 0 ['conv4_block2_add[0][0]']

conv4_block3_1_conv (Conv2D) (None, 2, 2, 256) 262400 ['conv4_block2_out[0][0]']

conv4_block3_1_bn (BatchNormal (None, 2, 2, 256) 1024 ['conv4_block3_1_conv[0][0]']
ization)

conv4_block3_1_relu (Activatio (None, 2, 2, 256) 0 ['conv4_block3_1_bn[0][0]']
n)

conv4_block3_2_conv (Conv2D) (None, 2, 2, 256) 590080 ['conv4_block3_1_relu[0][0]']

conv4_block3_2_bn (BatchNormal (None, 2, 2, 256) 1024 ['conv4_block3_2_conv[0][0]']
ization)

conv4_block3_2_relu (Activatio (None, 2, 2, 256) 0 ['conv4_block3_2_bn[0][0]']
n)

conv4_block3_3_conv (Conv2D) (None, 2, 2, 1024) 263168 ['conv4_block3_2_relu[0][0]']

conv4_block3_3_bn (BatchNormal (None, 2, 2, 1024) 4096 ['conv4_block3_3_conv[0][0]']
ization)

conv4_block3_add (Add) (None, 2, 2, 1024) 0 ['conv4_block2_out[0][0]',
'conv4_block3_3_bn[0][0]']

conv4_block3_out (Activation) (None, 2, 2, 1024) 0 ['conv4_block3_add[0][0]']

conv4_block4_1_conv (Conv2D) (None, 2, 2, 256) 262400 ['conv4_block3_out[0][0]']

conv4_block4_1_bn (BatchNormal (None, 2, 2, 256) 1024 ['conv4_block4_1_conv[0][0]']
ization)

conv4_block4_1_relu (Activatio (None, 2, 2, 256) 0 ['conv4_block4_1_bn[0][0]']
n)

conv4_block4_2_conv (Conv2D) (None, 2, 2, 256) 590080 ['conv4_block4_1_relu[0][0]']

conv4_block4_2_bn (BatchNormal (None, 2, 2, 256) 1024 ['conv4_block4_2_conv[0][0]']
ization)

conv4_block4_2_relu (Activatio (None, 2, 2, 256) 0 ['conv4_block4_2_bn[0][0]']
n)

conv4_block4_3_conv (Conv2D) (None, 2, 2, 1024) 263168 ['conv4_block4_2_relu[0][0]']

conv4_block4_3_bn (BatchNormal (None, 2, 2, 1024) 4096 ['conv4_block4_3_conv[0][0]']
ization)

conv4_block4_add (Add) (None, 2, 2, 1024) 0 ['conv4_block3_out[0][0]',
'conv4_block4_3_bn[0][0]']

conv4_block4_out (Activation) (None, 2, 2, 1024) 0 ['conv4_block4_add[0][0]']

conv4_block5_1_conv (Conv2D) (None, 2, 2, 256) 262400 ['conv4_block4_out[0][0]']

conv4_block5_1_bn (BatchNormal (None, 2, 2, 256) 1024 ['conv4_block5_1_conv[0][0]']
ization)

conv4_block5_1_relu (Activatio (None, 2, 2, 256) 0 ['conv4_block5_1_bn[0][0]']
n)

conv4_block5_2_conv (Conv2D) (None, 2, 2, 256) 590080 ['conv4_block5_1_relu[0][0]']

conv4_block5_2_bn (BatchNormal (None, 2, 2, 256) 1024 ['conv4_block5_2_conv[0][0]']
ization)

conv4_block5_2_relu (Activatio (None, 2, 2, 256) 0 ['conv4_block5_2_bn[0][0]']
n)

conv4_block5_3_conv (Conv2D) (None, 2, 2, 1024) 263168 ['conv4_block5_2_relu[0][0]']

conv4_block5_3_bn (BatchNormal (None, 2, 2, 1024) 4096 ['conv4_block5_3_conv[0][0]']
ization)

conv4_block5_add (Add) (None, 2, 2, 1024) 0 ['conv4_block4_out[0][0]',
'conv4_block5_3_bn[0][0]']

conv4_block5_out (Activation) (None, 2, 2, 1024) 0 ['conv4_block5_add[0][0]']

conv4_block6_1_conv (Conv2D) (None, 2, 2, 256) 262400 ['conv4_block5_out[0][0]']

conv4_block6_1_bn (BatchNormal (None, 2, 2, 256) 1024 ['conv4_block6_1_conv[0][0]']
ization)

conv4_block6_1_relu (Activatio (None, 2, 2, 256) 0 ['conv4_block6_1_bn[0][0]']
n)

conv4_block6_2_conv (Conv2D) (None, 2, 2, 256) 590080 ['conv4_block6_1_relu[0][0]']

conv4_block6_2_bn (BatchNormal (None, 2, 2, 256) 1024 ['conv4_block6_2_conv[0][0]']
ization)

conv4_block6_2_relu (Activatio (None, 2, 2, 256) 0 ['conv4_block6_2_bn[0][0]']
n)

conv4_block6_3_conv (Conv2D) (None, 2, 2, 1024) 263168 ['conv4_block6_2_relu[0][0]']

conv4_block6_3_bn (BatchNormal (None, 2, 2, 1024) 4096 ['conv4_block6_3_conv[0][0]']
ization)

conv4_block6_add (Add) (None, 2, 2, 1024) 0 ['conv4_block5_out[0][0]',
'conv4_block6_3_bn[0][0]']

conv4_block6_out (Activation) (None, 2, 2, 1024) 0 ['conv4_block6_add[0][0]']

conv5_block1_1_conv (Conv2D) (None, 1, 1, 512) 524800 ['conv4_block6_out[0][0]']

conv5_block1_1_bn (BatchNormal (None, 1, 1, 512) 2048 ['conv5_block1_1_conv[0][0]']
ization)

conv5_block1_1_relu (Activatio (None, 1, 1, 512) 0 ['conv5_block1_1_bn[0][0]']
n)

conv5_block1_2_conv (Conv2D) (None, 1, 1, 512) 2359808 ['conv5_block1_1_relu[0][0]']

conv5_block1_2_bn (BatchNormal (None, 1, 1, 512) 2048 ['conv5_block1_2_conv[0][0]']
ization)

conv5_block1_2_relu (Activatio (None, 1, 1, 512) 0 ['conv5_block1_2_bn[0][0]']
n)

conv5_block1_0_conv (Conv2D) (None, 1, 1, 2048) 2099200 ['conv4_block6_out[0][0]']

conv5_block1_3_conv (Conv2D) (None, 1, 1, 2048) 1050624 ['conv5_block1_2_relu[0][0]']

conv5_block1_0_bn (BatchNormal (None, 1, 1, 2048) 8192 ['conv5_block1_0_conv[0][0]']
ization)

conv5_block1_3_bn (BatchNormal (None, 1, 1, 2048) 8192 ['conv5_block1_3_conv[0][0]']
ization)

conv5_block1_add (Add) (None, 1, 1, 2048) 0 ['conv5_block1_0_bn[0][0]',
'conv5_block1_3_bn[0][0]']

conv5_block1_out (Activation) (None, 1, 1, 2048) 0 ['conv5_block1_add[0][0]']

conv5_block2_1_conv (Conv2D) (None, 1, 1, 512) 1049088 ['conv5_block1_out[0][0]']

conv5_block2_1_bn (BatchNormal (None, 1, 1, 512) 2048 ['conv5_block2_1_conv[0][0]']
ization)

conv5_block2_1_relu (Activatio (None, 1, 1, 512) 0 ['conv5_block2_1_bn[0][0]']
n)

conv5_block2_2_conv (Conv2D) (None, 1, 1, 512) 2359808 ['conv5_block2_1_relu[0][0]']

conv5_block2_2_bn (BatchNormal (None, 1, 1, 512) 2048 ['conv5_block2_2_conv[0][0]']
ization)

conv5_block2_2_relu (Activatio (None, 1, 1, 512) 0 ['conv5_block2_2_bn[0][0]']
n)

conv5_block2_3_conv (Conv2D) (None, 1, 1, 2048) 1050624 ['conv5_block2_2_relu[0][0]']

conv5_block2_3_bn (BatchNormal (None, 1, 1, 2048) 8192 ['conv5_block2_3_conv[0][0]']
ization)

```
conv5_block2_add (Add)         (None, 1, 1, 2048)   0           ['conv5_block1_out[0][0]',
                                                                 'conv5_block2_3_bn[0][0]']

conv5_block2_out (Activation)  (None, 1, 1, 2048)   0           ['conv5_block2_add[0][0]']

conv5_block3_1_conv (Conv2D)   (None, 1, 1, 512)    1049088     ['conv5_block2_out[0][0]']

conv5_block3_1_bn (BatchNormal  (None, 1, 1, 512)   2048        ['conv5_block3_1_conv[0][0]']
ization)

conv5_block3_1_relu (Activatio  (None, 1, 1, 512)   0           ['conv5_block3_1_bn[0][0]']
n)

conv5_block3_2_conv (Conv2D)   (None, 1, 1, 512)    2359808     ['conv5_block3_1_relu[0][0]']

conv5_block3_2_bn (BatchNormal  (None, 1, 1, 512)   2048        ['conv5_block3_2_conv[0][0]']
ization)

conv5_block3_2_relu (Activatio  (None, 1, 1, 512)   0           ['conv5_block3_2_bn[0][0]']
n)

conv5_block3_3_conv (Conv2D)   (None, 1, 1, 2048)   1050624     ['conv5_block3_2_relu[0][0]']

conv5_block3_3_bn (BatchNormal  (None, 1, 1, 2048)  8192        ['conv5_block3_3_conv[0][0]']
ization)

conv5_block3_add (Add)         (None, 1, 1, 2048)   0           ['conv5_block2_out[0][0]',
                                                                 'conv5_block3_3_bn[0][0]']

conv5_block3_out (Activation)  (None, 1, 1, 2048)   0           ['conv5_block3_add[0][0]']

==================================================================================================
Total params: 23,587,712
Trainable params: 23,534,592
Non-trainable params: 53,120
__________________________________________________________________________________________________
```