Multi-Class Classification of the CIFAR10 Dataset with a Convolutional Neural Network

Brandon Pratt, 12/3/2020,

STAT535 Final Project

Preprocessing

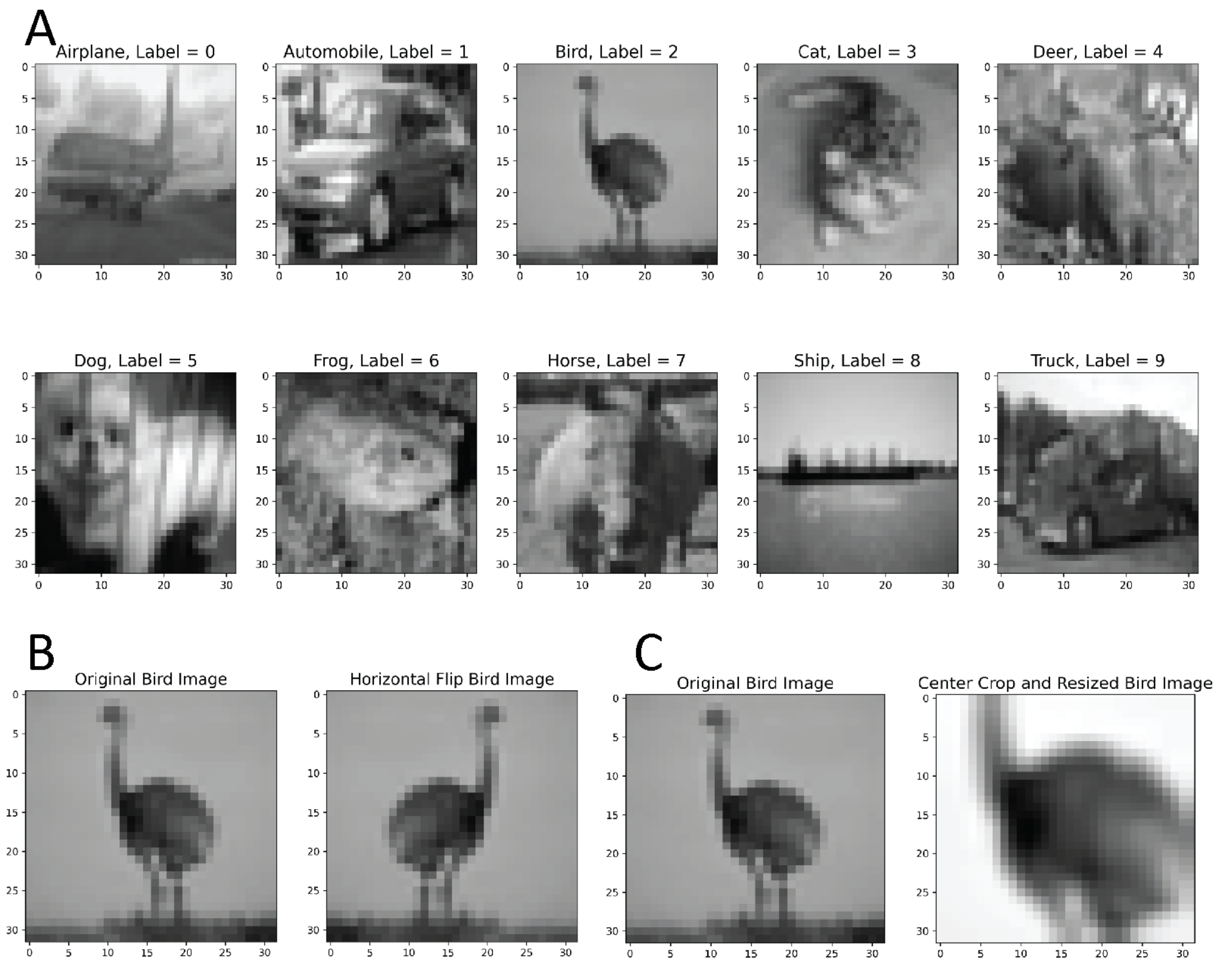

Figure 1: Preprocessing of RGB images from the training set. A) RGB images converted into grayscale. The images belonged to one of ten categories, including: airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. B) Example image flipped along its horizontal axis. C) Example center cropping and resizing of an image.

The preprocessing of the provided CIFAR-10 dataset of 40,000 images with 3 RGB channels involved transforming the images into grayscale, flipping the images along their horizontal axis, and center cropping and resizing of the images. Transforming the images into grayscale collapsed the number of input channels into the convolutional neural network from three channels to one channel. The original RGB images were also flipped horizontally and appended with the original RGB images for training. This was a logical transformation to perform because it maintained the global pixel structure of the original images, although, reversed, and added more training content for the convolutional neural network. The last augmentation to the original RGB images involved cropping the center of the images and resizing them to 32 x 32 pixels. This augmentation seemed logical as it should extract key features from the images, like animal faces, and enlarge them. These cropped and resized images were appended to the RGB images that were horizontally flipped and the original RGB images for training. The above transformations were confirmed through visualization (Fig. 1). Note that the CIFAR-10 dataset contains the following 10 classes: label 0 – airplanes, label 1 – automobiles, label 2– birds, label 3 – cats, label 4 – deer, label 5 – dogs, label 6 – frogs, label 7 – horses, label 8 – ships, and label 9 – trucks.

Predictor

A convolutional neural network (CNN) was the chosen predictor for classifying the CIFAR-10 RGB images into the 10 associated classes. The architecture of the CNN used for classification contained five layers, 2 convolutional layers and 3 linear layers. The input into the first convolutional layer was a batch of 25 32 x 32-pixel images. These images consisted of 1 or 3 channels (i.e. grayscale or original /augmented RGB images). In the layer, these batches of images underwent a 2D convolution with a 3x3 square kernel given by the following expression [ref 1]:

$$Out\left(N_i, C_{out\ j}\right) = bias\left(C_{out\ j}\right) + \sum_{k=0}^{C_{in}-1} weight\left(C_{out\ j}, k\right) * input(N_i, k) \quad [1]$$

Where N was the batch size, C_{in} was the number of input channels (equals one for grayscale images and 3 for RGB images), C_{out} was the number of output channels specified by the user (i.e. 8), and '*' denotes the 2D cross-correlation operator. The weights and bias are sampled from a uniform distribution with range -$\sqrt{k}$ to $\sqrt{k}$, where k is obtained by the following equation [ref 1]:

$$k = \frac{groups}{C_{in} * \prod_{i=0}^{1} kernel\_size[i]} \quad [2]$$

Where kernel_size was the specified kernel size (i.e. 3) and groups dictated the connections between input and output layers. The 2D convolution on these images with the 3x3 square kernel resulted in the height and width of these convolved images to decrease by 2 pixels. This was determined by the following equations [ref 1]:

$$H_{out} = \frac{H_{in} + 2*padding - dilation(kernel_{size} - 1) - 1}{stride} + 1 \quad [3]$$

$$W_{out} = \frac{W_{in} + 2*padding - dilation(kernel_{size} - 1) - 1}{stride} + 1 \quad [4]$$

Where the padding equaled 0, the dilation equaled 1, the kernel size equaled 3, and the stride equaled 1.

Within the same convolutional layer, another 2D convolution with a 3x3 square kernel was performed on the output of the prior 2D convolution (eqs. 1 & 2). The C_{in} and C_{out} for this second 2D convolution was 8 and 16 channels, respectively. Moreover, like with the prior 2D convolution, the height and width of the convolved images decreased by 2 pixels.

After these two 2D convolutions, the resulting convolved images were pooled using max pooling with a 2x2 kernel. This max pooling operation extracted the maximum value within the 2x2 kernel that translated across the twice convolved images. This max pooling procedure resulted in the 2D dimension of the convolved input data to be reduced by a factor of 2. Thus, the dimension of the associated input after max pooling was 14x14 pixels.

The output from max pooling was then passed through a rectified linear unit (ReLU). The ReLU set any value of the output less than 0 to 0, and any value equal to or greater than 0 to 1. The output of this non-linear activation retained the dimensions of the input (i.e. 14x14 pixels), but transformed the values of the input into 0's or 1's. This ReLU operation concluded convolutional layer 1. Thus, convolutional layer 1 consisted of two 2D convolutions with 3x3 square kernels, max pooling with a 2x2 square kernel, and a non-linear activation transformation of the input with a ReLU.

The output from convolutional layer 1 was then passed to convolutional layer 2. Like convolutional layer 1, convolutional layer 2 consisted of two 2D convolutions with 3x3 square kernels, max pooling with a 2x2 square kernel, and a ReLU. In convolutional layer 2, the C_{in} and C_{out} for the first and second 2D convolutions was 16 and 32 channels, and 32 and 32 channels, respectively. The final output from convolutional layer 2 after the two 2D convolutions, max pooling, and non-linear activation was a transformed 5x5 pixel representation of the original input.

From here, the 5x5 pixel output was flattened and inputted into linear layer 1. The operation that linear layer 1 performed on the input is given by the following expression:

$$y = W * x + b \quad [5]$$

Where y was the desired output (i.e. 160 channels), x was the input features, b was the bias, and W was the weights. Importantly, the number of "in features" passed to linear layer 1 (i.e. x in eq. 5) was determined by multiplying the pixel dimensions (i.e. 5 x 5) by the number of output channels from convolutional layer 2 (i.e. 32 channels). Thus, there were 800 "in features". Furthermore, the specified number of output channels was arbitrarily chosen to be 160 channels. These 160 output channels were then passed through linear layer 2 and underwent the same operation as shown in eq. 5. The operation of linear layer 2 resulted in 84 output channels. These 84 channels were finally passed into linear layer 3 and were transformed into 10 output channels representing the 10 image classes.

After passing the images through these five layers of the CNN, each image was classified into one of the 10 categories described above. In particular, each image was classified into a category by first performing a log softmax on the output channels from linear layer 5. The log softmax was of the form [ref 2]:

$$f(x_i) = \log\left(\frac{e^{x_i}}{\sum_j e^{x_j}}\right) \quad [6]$$

Where Xi corresponded to one of the ten output channels from linear layer 5, and i = 1-10.

After applying the log softmax to the 10 output channels associated with an image from linear layer 5, the index of the maximum value across these 10 transformed channels was determined. The image was then classified into the category corresponding to this index.

Basic Training Algorithm

The basic training algorithm involved passing batches of 25 input RGB, grayscale, or RGB with augmentation images through the above CNN, calculating the cross-entropy loss, backpropagating the gradients into the CNN parameters, and using stochastic gradient descent (SGD) or Adam to update the weights within the CNN. In more detail, the first step of training required isolating batches of 25 images from the PyTorch tensor containing all the images used for training. The second step of training involved processing the input images through the CNN to obtain the probabilities of each image within the batch being in one of the ten image classes. From here, the error between the output of the CNN and the true image labels was computed as cross-entropy loss. This loss is of the form [ref 3]:

$$L(x, class) = -x[class] + \log\left(\sum_j e^{x_j}\right) \quad [7]$$

Where $x[class]$ was the true label of the image and $\log\left(\sum_j e^{x_j}\right)$ was the estimated label. Note that the cross-entropy loss combines the log softmax and negative log likelihood functions. This error and its corresponding gradients were then backpropagating through the CNN. The backpropagation was easy to do in PyTorch as it conveniently computed the gradients internally. Lastly, the weights of the CNN were updated using SGD or Adam. The expression of SGD is as follows:

$$weight_{i+1} = weight_i - learning\ rate * gradient \quad [8]$$

Where weight corresponded to the weights of the CNN, the learning rate was chosen to be 0.0001, i was the iteration number, and the gradient was computed through backpropagation. Like SGD, Adam is also a stochastic gradient optimization, but importantly involves adaptivity estimating the learning rate for different parameters as well as has momentum [ref 4]. The learning rate chosen for the Adam optimization was also 0.0001. Finally, the above training procedure was repeated for 100 epochs (i.e. training events) if the Adam optimizer was chosen or 300 epochs if the SGD optimizer was chosen. It should be noted that training was performed on a GPU provided by GoogleColabs.

Training Strategy

The training strategy involved using the above training algorithm (i.e. processing input images through the CNN, computing the loss and gradients, and updating the model weights) to learn the CNN parameter weights over 100 or 300 epochs during which the network performance was cross-validated. A total of seven CNNs were trained. The training of these CNNs differed in one or more of the following ways: 1) training set size, 2) type of input images (i.e. augmented or not), and 3) the optimizer used (i.e. Adam or SGD). There were thus seven trained CNNs that included: 1) CNN trained on 35,000 original RGB images with the SGD optimizer, 2) CNN trained on 35,000 original RGB images with the Adam optimizer, 3) CNN trained on 35,000 grayscale images with the SGD optimizer, 4) CNN trained on 35,000 grayscale images with the Adam optimizer, 5) CNN trained on 70,000 original and horizontally flipped RGB images (35,000 images each) with Adam optimizer, 6) CNN trained on 105,000 original, horizontally flipped, and center cropped RGB images (35,000 images each) with Adam optimizer, and 7) CNN trained on 79,900 original and horizontally flipped RGB images (39,950 images each) with Adam optimizer. Note that the seventh trained CNN was used for testing. The CNNs which had their weights updated by the

Adam optimizer were trained over 100 epochs, whereas those that had their weights updated by the SGD optimizer were trained over 300 epochs. This was because the optimal weights seemed to be learned faster by the Adam optimizer as compared to the SGD optimizer. Also, note that the initial weights of each CNN were predetermined by PyTorch.

In terms of training each of these CNNs, the training set was segmented into a batch size of 25 images, each of which were processed by the CNN. The cross-entropy loss associated with each of these batches was then recorded and the average loss across all batches was computed for each epoch. In addition to recording the average-cross entropy loss, the percent training accuracy was calculated and recorded for each epoch. The percent training accuracy was calculated as the number of correctly predicted training labels divided by the total number of training labels multiplied by 100. Both the average cross-entropy training loss and training accuracy were used to assess the performance of each CNN across epochs.

Furthermore, for each epoch, the CNNs were cross validated on withheld images from the original training set of 40,000 images. 8-fold cross-validation was performed on CNNs 1-4 described above. This involved segmenting the original training set (40,000) images into 8 different segments consisting of 5000 images used for validation and the other 35,000 images used for training. Thus, the 8-fold cross-validation was calculated as the average cross-entropy loss associated with the final epoch for each of the 8 validation sets. 1-fold cross-validation was conducted on CNNs 5-7 described above because of time constraints of the project. 1-fold cross validation involved computing the cross-entropy loss on the first 5000 withheld images of the original 40,000 image training set after the final epoch of CNN training. Importantly, the validation cross-entropy loss and accuracy (calculated in the same manner as the training loss and accuracy) were computed and stored for each epoch. Lastly, the predicted labels for the validation image set were stored after the final epoch across all CNNs. These predicted labels were then compared with the true labels by constructing a confusion matrix.

Experimental Results

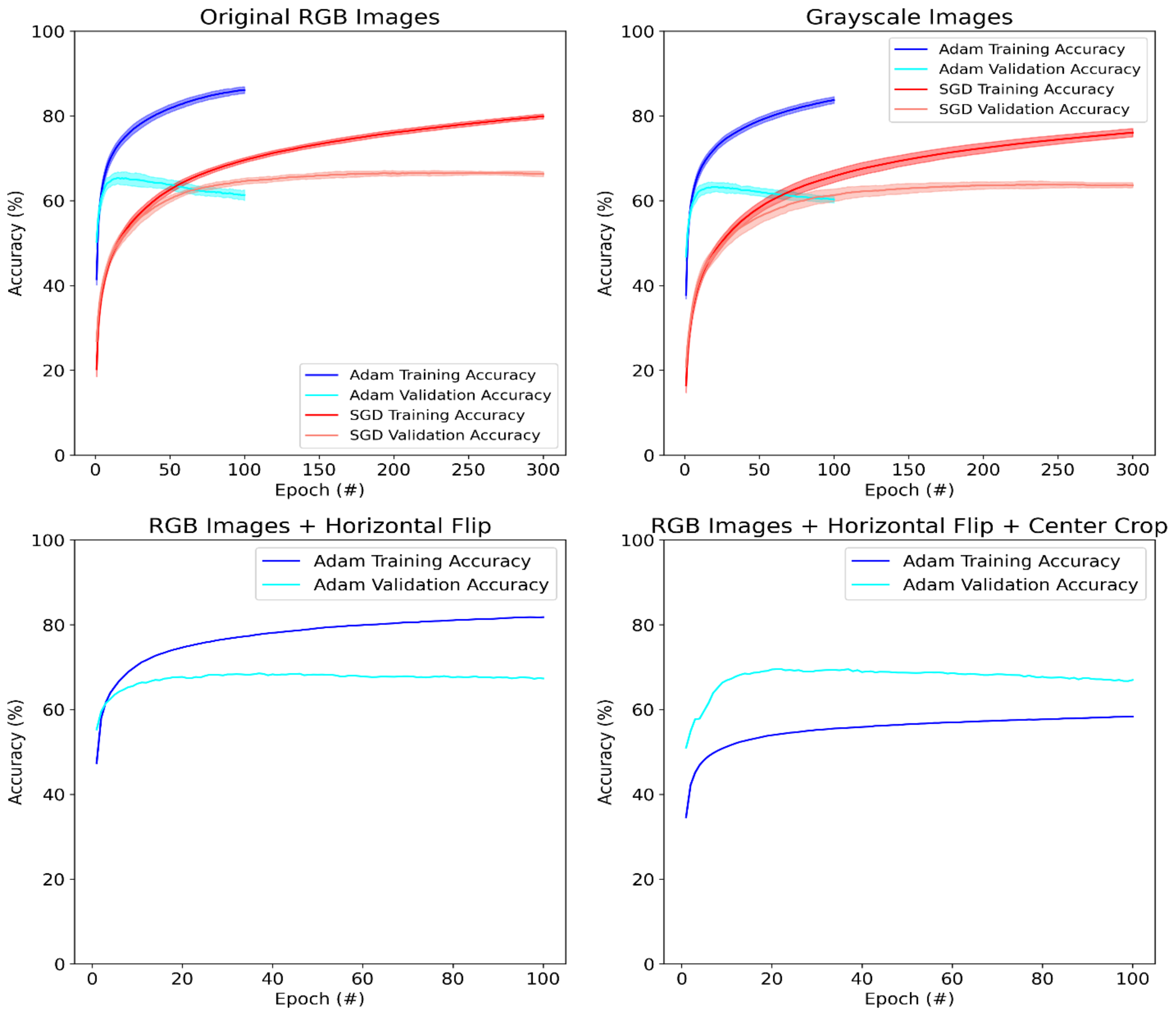

Figure 2: Training (Blue: Adam and Red: SGD) and validation accuracy (Cyan: Adam and Salmon: SGD) for CNNs trained with RGB images (top left), grayscale images (top right), RGB + horizontally flipped RGB images (bottom left), and RGB + horizontally flipped + center cropped RGB images (bottom right). Bounded error in top panels is the standard deviation of the 8 training and validation curves.

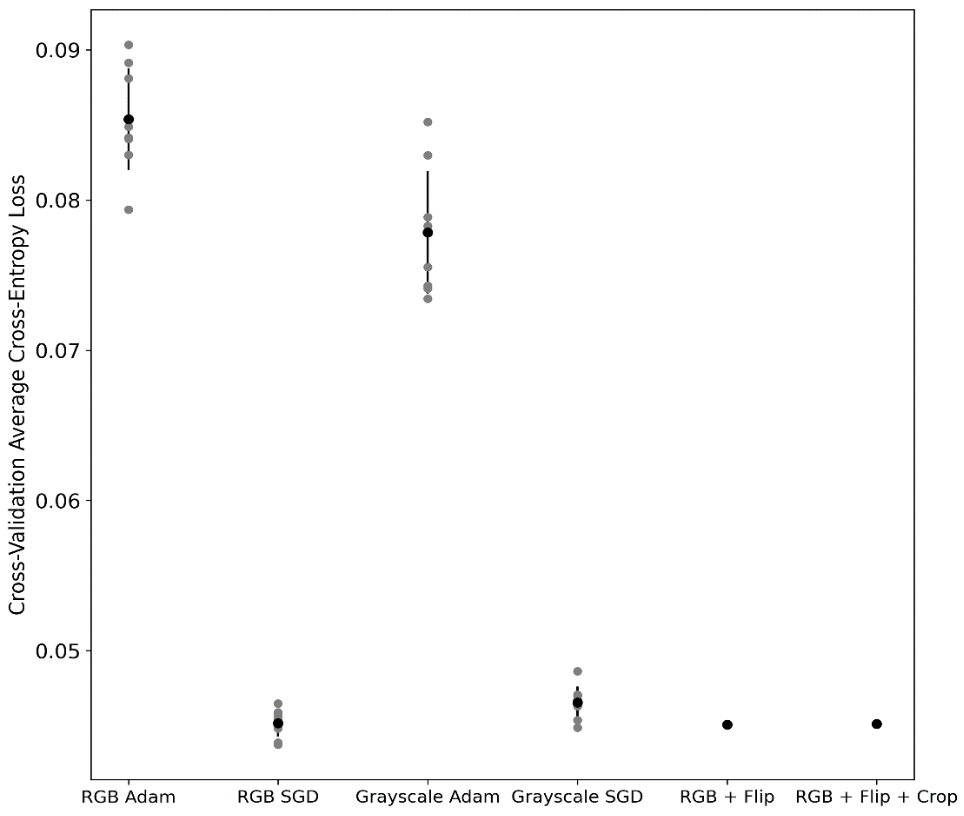

Figure 3: 8-fold or 1-fold cross-validation of the average cross-entropy loss across the CNNs trained on RGB, grayscale, or RGB plus augmented images and with either the Adam or SGD optimizer.

After visually confirming that each of the augmentations (i.e. grayscale, horizontal flipping of the RGB images, and center cropping of the RGB images) worked as expected (Fig. 1), the performance of each of the seven CNNs listed above was assessed. As mentioned above, CNN 1 and 2 were trained on the same unaugmented RGB images, but differed in which optimizer (i.e. Adam or SGD) performed the update step. Noticeably, the training accuracy of CNN 2, which used the Adam optimizer, increased more rapidly than CNN 1 and achieved a higher training accuracy of ~85% at 100 epochs as compared to the ~78% training accuracy obtained by CNN 1 at 300 epochs (Fig. 2 top left panel). Interestingly, the validation accuracy of CNN 2 declined after 20 epochs to an accuracy of about 62% at 100 epochs, whereas CNN 1 reached a steady state validation accuracy of about 64% at 300 epochs (Fig. 2 top left panel). This was consistent with CNN 1 having a lower 8-fold cross-validation average cross-entropy loss of 0.045 as compared to CNN 2's 8-fold cross-validation loss of ~0.085 (Fig. 3). Note that the 8-fold cross-validation average cross-entropy loss was used as the **classification error estimate** for CNN's 1-4.

Although, CNNs 3 and 4 were trained on grayscale versions of the RGB images, they performed similarly to CNNs 1 and 2 in terms of training and validation accuracy (Fig. 2 top right panel). Like CNN 2, CNN 4 also used the Adam optimizer and achieved a higher training accuracy of ~82% at 100 epochs as compared to the 75% training accuracy obtained by CNN 3 at 300 epochs. Furthermore, CNN 4's validation accuracy also declined after 20 epochs to a final accuracy of

60%, whereas CNN 3 reached a steady state validation accuracy of 63% by 300 epochs. The 8-fold cross-validation loss of CNN 3 was ~0.047, which was lower than that of CNN 4 in which had an 8-fold cross-validation loss of 0.078 (Fig. 3). Overall, the performance of CNNs 1 and 3, and 2 and 4 were similar, although, the performance of those trained on the unaugmented RGB images may be slightly better.

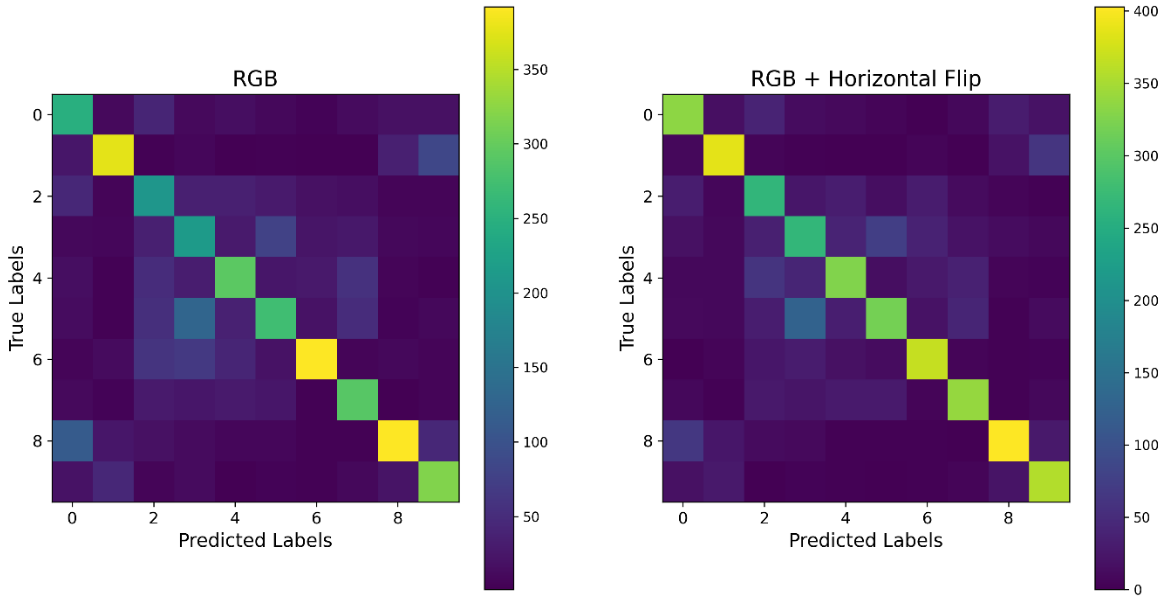

Figure 4: Confusion matrix of the CNN trained on the original RGB images (left panel) and that trained on the original plus horizontally flipped RGB images (right panel). These matrices demonstrate the misclassification of one label as another.

The CNNs that were trained on the original plus augmented RGB images (i.e. CNN 5: original + horizontally flipped RGB images, and CNN 6: original + horizontally flipped + Center Cropped RGB images) outperformed CNNs 1-4 in terms of validation accuracy. Both CNN 5 and 6 reached a final validation accuracy of ~68% (Fig. 2 lower left and right panels, respectively). Notably, the training accuracy of CNN 5 reached a higher value of 82% compared to the 60% training accuracy achieved by CNN 6. This difference was most likely due to CNN 6 having 35,000 more training images than CNN 5. Moreover, CNN 5 and 6 had a similar 1-fold cross-validation average cross-entropy loss (**estimate of the classification error**) of ~0.043 (Fig. 3). Due to its higher validation accuracy and lower estimated classification error than CNNs 1-4 and the fewer training images than CNN 6, the augmentation used in CNN 5 was chosen to be that for the final CNN used for testing. CNN 5's performance was further compared to that of CNN 2 (trained on unaugmented RGB images) by determining which labels were misclassified. It was evident that CNN 2 and 5 classified the majority of images used for validation into their correct categories, but often misclassified dogs (label 5) as cats (label 3), automobiles (label 1) as trucks (label 9), deer (label 4) as cats, and vice versa (Fig. 4). Importantly, the augmentation conducted in CNN 5 helped to better classifying the images into their proper categories.

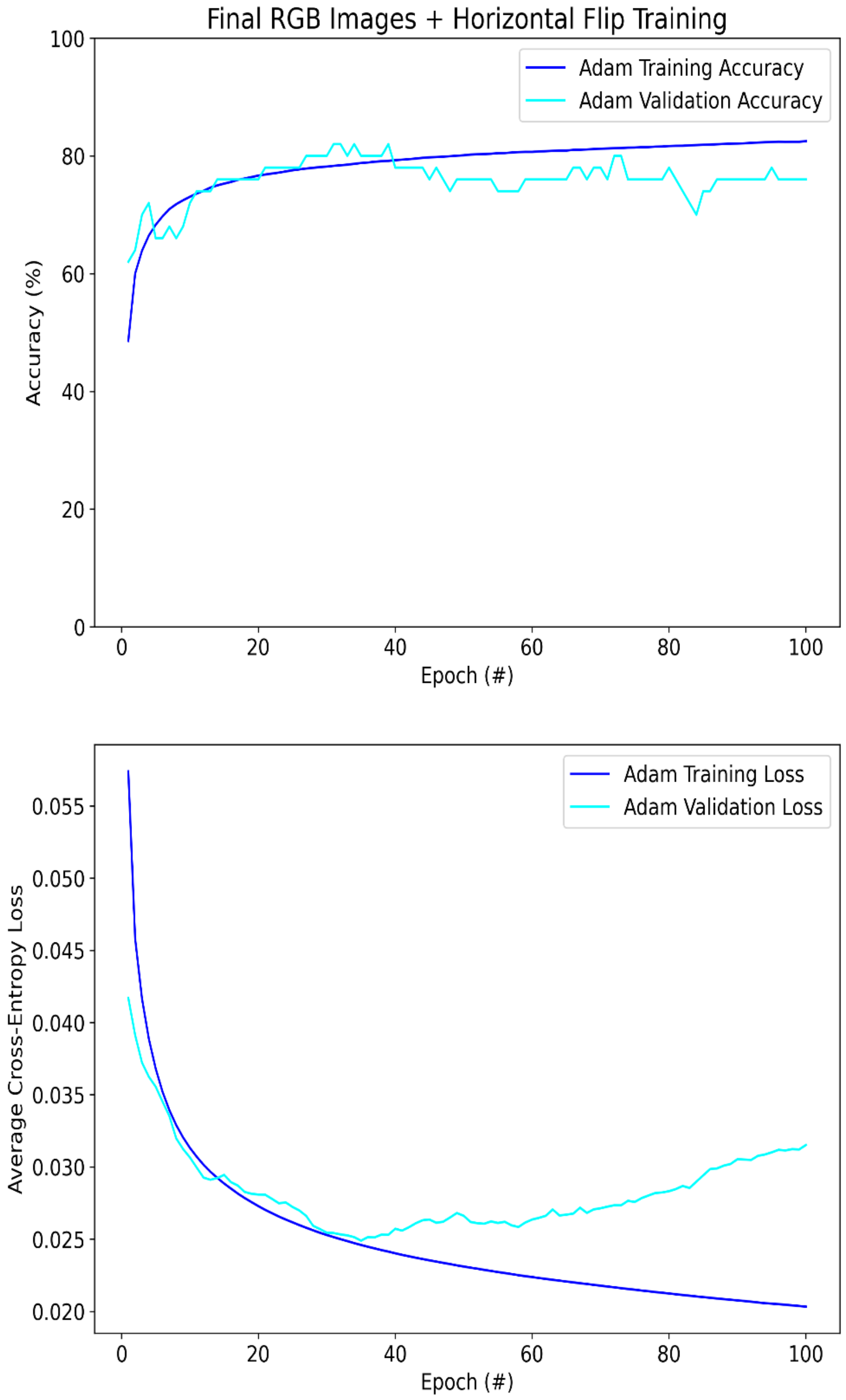

Figure 5: Training and validation accuracy (top panel: blue and cyan), and average cross-entropy training and validation loss (bottom panel) of CNN used for testing. This CNN trained on 79,900 images consisting of original and horizontally flipped RGB images, and also used the Adam optimizer.

Given the performance of CNN 5, the CNN used for testing, CNN 7, mimicked CNN 5, but differed in that it trained on more images from the provided training set (Fig. 5). Note that because the validation of CNN 7 was done on fewer images, the estimated cross-validation classification error was not as accurate. Regardless, CNN 7 achieved a validation accuracy of 75% and a training accuracy of 82% at 100 epochs (Fig. 5 top panel). Moreover, CNN 7 obtained a final training loss of ~0.020 and a validation loss of ~0.0325 at 100 epochs (Fig. 5 bottom panel). CNN 7 was estimated to have a classification error of 30%, and thus was the highest performing CNN.

Overall, the trained CNNs correctly predicted the class of each image well above chance. Moreover, augmentation appeared to be crucial in improving classification accuracy. In addition, the Adam optimizer achieved a higher training accuracy and a comparable validation accuracy as to that obtained by the SGD optimizer. This is important as the Adam optimizer allows for faster training session compared to the SGD optimizer without sacrificing the classification performance of the CNN. Finally, the CNNs were shown to be within the classical training regime because the training accuracy did not reach 100%.

References

1) PyTorch. CONV2D: https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html.
2) PyTorch.LOGSOFTMAX:https://pytorch.org/docs/stable/generated/torch.nn.LogSoftmax.html.
3) PyTorch.CROSSENTROPYLOSS:https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html.
4) Kingma D. and Lei Ba J. (2017). Adam: A method for stochastic optimization. arXiv: 1412.6980v9.