Assignment 2 Report

Can Gözpınar [1]

1. Convolutional Neural Networks

This section contains my solutions to the questions in section 1 of the assignment 2 instructions.

1.1. Convolutional Filter Receptive Field

The dimensions of the receptive field for a node in Z_2 is 7×7. A node in Z_2 has a receptive field of size 3×3 which is the corresponding kernel size of the 2^{nd} convolutional layer. Then this corresponding 3×3 patch in Z_1 has the receptive field of 7×7.

It is effective to build convolutional networks deeper with more layers because it allows nodes at the final layers to have larger receptive fields. As a result of this, the convolutional networks can represent higher level features as the network depth increases. For example, earlier layers learn to recognize very basic things like edges whereas, deeper layers learn to recognize higher level things such as shapes. So in other words, the gradual increase in the receptive field helps convolutional neural networks with learning features at different levels of abstractions.

1.2. Run the Pytorch ConvNet

- There are a total of 4 layers. 2 of these are convolutional layers and the remaining 2 are fully connected layers. As a result they are not all convolutional layers, there are fully connected layers too.
- ReLU activation is used on the hidden nodes.
- Cross Entropy loss is being used to train the network.
- Loss is being minimized by updating the model parameters during the training. To do this backpropagation is being used to compute the partial gradients of the loss with respect to the model parameters. Then, ADAM optimizer is used to perform the updates on the model parameters.

Training accuracy for my model after the training is 0.6972222222222222. Validation accuracy is 0.5494505494505495. These two numbers tell me that the model has overfit the training data. In other words, the model has started to memorize the training data which hinders it generalization performance on the validation data. Also, these numbers show that the model has learned. Wee see this because there are 11 classes which would mean that a random guess would guess correctly with the probability of $1/11$ (accuracy of such a model model is $1/11$). We see that our model has significantly higher validation and training accuracy in comparison to a random model. Therefore it shows that the model has learned from the data.

1.3. Add Pooling Layers

After I applied max pooling my results improved. Both the training accuracy and the validation accuracy has increased. The difference between the training accuracy and the validation accuracy has decreased. This tells me that max pooling can reduce overfitting and improve model performance.

1.4. Regularize Your Network!

- *Dropout*: For $p = 0.5$ it resulted in lower performance. Both the training and the validation accuracies decreased (training accuracy: 0.45, validation accuracy: 0.42) but, the gap between these two accuracies became very small too which means as a regularizer it reduced overfitting too. For $p = 0.1$ it resulted in better performance that $p = 0.5$. The validation and the training accuracies (training accuracy: 0.64, validation accuracy: 0.58) became higher. Finally, I tested with $p = 0.3$ which resulted in training accuracy of 0.58 and validation accuracy of 0.55. Dropout reduces the gap between the training and the validation accuracies. It helps with reducing overfitting. It can improve models generalization at the expense of reduced training accuracies.

- *Weight Regularization*: Using SGD has resulted in significantly lower training accuracy and validation accuracy in comparison to using ADAM. Weight decay acted as a regularizer. Using stronger values for it lead to even stronger regularization which further reduced the training accuracy values and further reduced the difference between the two accuracies. I found out that using smaller weight decay values worked better for

[*]Equal contribution [1]Department of Computer Engineering. Correspondence to: Can Gözpınar <cgozpinar18@ku.edu.tr>.

COMP541 Deep Learning, Fall 2022.

my experiments.

- *Early Stopping*: Patience value of 10 resulted in validation accuracy of 0.53. Patience value of 100 resulted in validation accuracy of 0.62. Further increasing (e.g. patience = $\{120, 150\}$) it resulted in lower validation accuracies.

- *Learning Rate Scheduling*: Using StepLr as the learning rate schedular with step_size of 30, and gamma of 0.1 resulted in performance degradation. Using cosineAnnealing with eta_min set to 10 also resulted in performance degradation. After tuning StepLr's parameters a little bit the degradation was reduced. Learning rate schedular's usage helps with the overfitting issue by reducing the learning rate value towards the ends of the learning.

Early stopping was the most effective regularization technique. It is important to note that this was the case for my experiments and tuning them differently might yield different results.

1.5. Experiment with Your Architecture

Having a larger filter size such as 7 for the convolutional layers resulted in lower accuracy. Whereas, using smaller filter size such as 3 resulted in better accuracy. Increasing the filter size increases the number of parameters of the model. Also, larger filter size has larger receptive field as a result it does not extract lower level features as good as a smaller filter does. Having larger stride value resulted in bigger dimensionality reduction. It resulted in lower performance. Having greater depth introduced more parameters and increased the model training and inference times. It resulted in better performance since it allowed model to learn better representations. But increasing the depth too much resulted in overfitting. Having a pyramidal-shaped network results in better performance. This is due to the fact that, increasing the height and width gradually introduces extraction of better representations. For example, initial layers extract lower level features whereas layers towards the output represents higher level features which are in some way combination of these learned lower level features. This also introduces dimensionality reduction and these features extracted by the convolutional layers helps the performance of the final fully connected layers. On the other hand, having a flat architecture suffers performance loss. This happens because this type of architecture would not be able to extract features as successfully as the prior one. For example to keep the dimension the same using convolutional layers would require too much padding which would result in loss of information eventually on certain hidden nodes.

1.6. Optimize Your Architecture

Figure 1 shows the best performance I was able to achieve. It achieved best validation accuracy of 0.725. For achieving it I used learning rate of 10^{-3}, batch size of 128. I also used Pooling layers and dropout layers with dropout probability $p = 0.25$. One dropout layer was applied after flattening the activations, and the other dropout was applied before the final linear layer. I also used early stopping with patience value of 50. As an optimizer I used ADAM. I found that using early stopping, dropout and pooling layers have helped me with achieving better validation accuracy by preventing overfitting.

1.7. Test Your Final Architecture on Variations of the Data

I started with trying ColorJitter transformation. It resulted in validation accuracy to drop to 0.53 from the initial 0.72. I believe my model is not very invariant to changes such as brightness, color, and saturation. I believe it might have resulted in accuracy drop since the model might have learned about the color of the certain peoples skin and this transformation might be misleading the model. Furthermore, using GaussianBlur did not decrease the performance of the model a lot. But increasing the blur has lead to a accuracy loss. I believe the model is a little bit invariant to noise in the input image. Also, I tried RandomPerspective transformation. It resulted in significant drop of validation accuracy (0.54). This implies that my model is not invariant to perspective which is a very important property for computer vision. I tried RandomRotation transform with rotations in the range of $[-45°,\ +45°]$. The model has achieved 0.64 validation accuracy which indicated that the model is more invariant to rotations than it was to changes in perspectives. Finally, I tried to apply AffineTransformation. Using it to translate results have dropped accuracy on the validation set to 0.45. This shows that the model is not invariant to translations. This is an important issue since computer vision models should be invariant to translations as much as possible for good generalization performance. Applying different scaling to input images resulted in validation accuracy drop too. The accuracy has dropped down to 0.51. I am not suprised by the results since during the training of the model, these transformations were not applied as a preprocessing step. Moreover, the images in the dataset are mostly taken from the same angles and of same size. As a result, the lack of transformations to the images during the training phase combined with the similar structure of the unprocessed images results in a model that is not very invariant. It turns out that GaussianBlur was the transformatino that my model was most invariant to, and using AffineTransformation for translating the images lead to the images that are the most recognizable by my model. My model's invariance to transformations such as translation and scale tells me taht the features that my model has learned are not very generalizable features. This might be because the model has learned some features that are specific to the training dataset but not the task at hand in general.

2. Transfer Learning with Deep Network

This section contains my solutions to the questions in section 2 of the assignment 2 instructions.

2.1. Train a Multilayer Perceptron

My model (Neural Network) is a fully-connected network with a single hidden layer of size 300. It's first layer flattens the image input image which is of shape [C, H, W] to a single dimension of shape [C*H*W]. Then this flattened input is passed through a linear layer which produces hidden features of shape 300. Then its outputs are passed through ReLU activations. Then these outputs are passed through another linear layer which produces the same number of neurons with the number of possible classes (number of unique celebrities in the dataset). The layers are initialized using Kaiming Uniform. As a preprocessing step I resized the images to 28×28 pixels, and converted them to torch Tensors. Its final performance on the test set is test_loss: 3.21 and test_accuracy: 0.47. Refer to figure 2 for further insight into its performance during training, and refer to figure 3 for observing final test performance. Note that during training around 400 epochs the model starts to overfit. Using regularization methods such as L2 or dropout might help with this issue.

2.2. AlexNet as a Fixed Feature Extractor

In my system a pre-trained AlexNet loaded from pytorch Hub is used as feature extractor. This AlexNet's parameters are always frozen entirely. I pass preprocessed input images through this AlexNet and take its outputs at the *conv4* layer. Then, these extracted outputs (features) are passed through my fully connected network. This fully-connected network is trained from scratch using the features extracted by the AlexNet's *conv4* layer. This fully-connected network first flattens the inputs so that it can be passed through linear layer. After flattening (after flattening its dimension becomes $[256 * 6 * 6]$), it is passed through a linear layer with output dimension of 4096. Then these are passed through ReLU activation function. Then its outputs are passed through another linear layer with output shape of 4096. Then another ReLU activation function is applied. Finally, these activations are passed through the final linear layer which the same number of neurons as the number of available classes in the dataset. The layers in my full-connected network are initialized using Kaiming Uniform. As a preprocessing step, I applied the preprocessing step that pre-trained AlexNet was using. To clarify further, I resize input images to 256×256 pixels, take a 224×224 crop at the center of the image, convert to torch Tensors, and Normalize using mean=[0.485, 0.456, 0.406] and std=[0.229, 0.224, 0.225]. It is important to note that during training only my fully-connected network was trained while the pre-trained AlexNet's layers were frozen (not updated). During inference time, the inputs are preprocessed as was described before and then passed through pre-trained AlexNet where its $conv4$ layers activations are fed into my fully-connected network to produce classification scores (logits). Its final performance on the test set is test_loss: 1.35 and test_accuracy: 0.628. Refer to figure 4 for further insight into its performance during training, and refer to figure 5 for observing final test performance. Note that we see model is starting to overfit. We understand this from the fact that the gap between the training and the validation curves widening. The network is starting to memorize training dataset which hurts its generalization performance. Regularization methods such as L2 regularization can be used to help with this issue.

2.3. Visualize Weights

These visualizations are interesting because especially the Model 1 Neuron 2 weights visualization resembles the features of a face. In that visualization one can clearly spot the outline of the head, eyes and mouth. Model2 Neuron 2 weight visualization resembles the outline of a head which is circular there. Overall, these visualization give an idea about what the model is looking for in a given input image. Note that having features that resemble a face is in accordance with our task since, we are trying to classify different faces. These visualizations are given in the figure 6

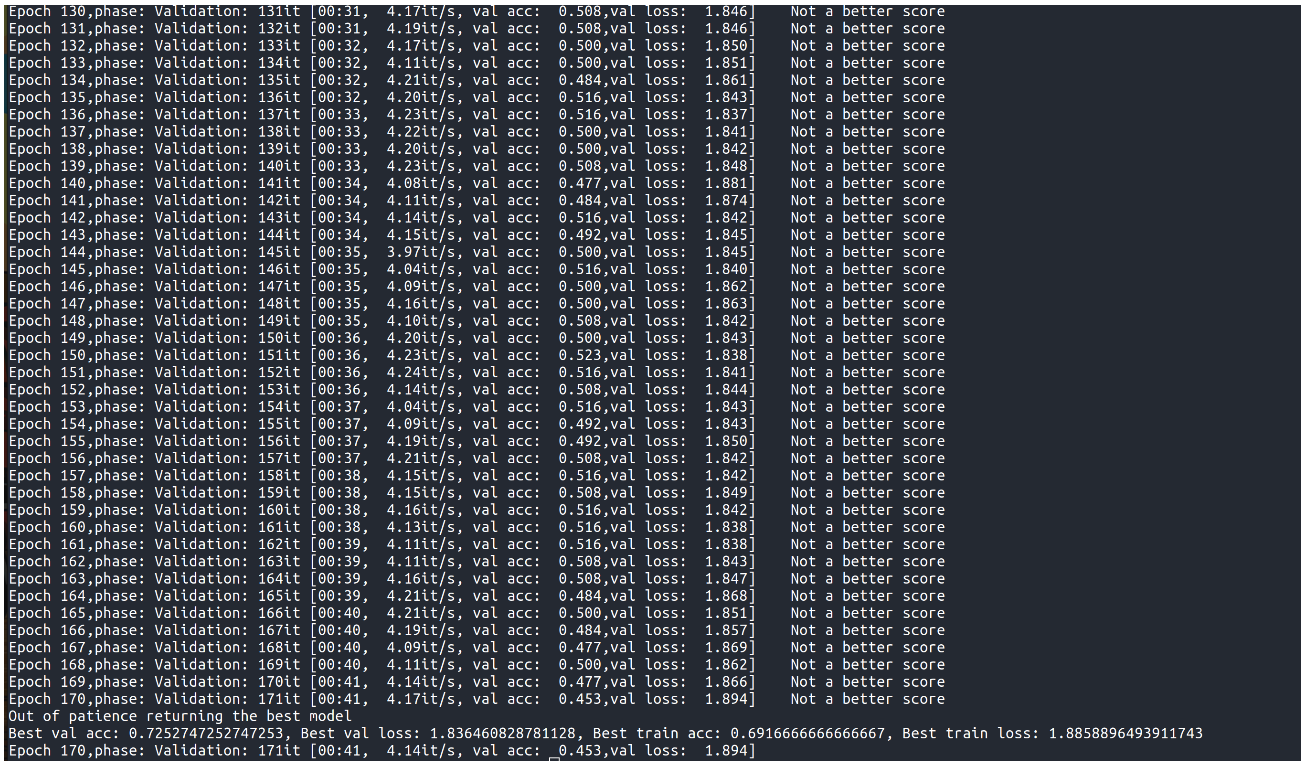
2.4. Finetuning AlexNet

Figure 7 shows the training and validation curves of the model used. Figure 8 shows a false prediction made by the model. For this prediction the face of Julianna Margulies's image was used as an input to the model. Whereas, the model predicted Ethan Hawke which was wrong. Figure 9 shows a case where the model has predicted correctly. In that case, the model was given the image of the face of Michael Weatherly as input, and the model has successfully predicted Michael Weatherly.

2.5. Bonus: Gradient Visualization

Figure 10 shows the training and validation curves for the model that is being used for this task. Figure 11 shows the sample face image used for the gradient visualization. Here we also see that the model has successfully predicted Christa Miller for the corresponding image. Figures 12, 13, 14, 15 show the results for the Guided Backprop on the aforementioned image of the face of Christa Miller. These results give us an idea of what the neural network is looking for when it is evaluating if a given sample belongs to the class of Christa Miller. In other words, these results show us what kind of input is giving high classificaiton score for Christa Miller. We see that model has learned Christa Miller's face to be looking something like the resulted visualizations.

A. Appendix



```
Epoch 130,phase: Validation: 131it [00:31,  4.17it/s, val acc:  0.508,val loss:  1.846]    Not a better score
Epoch 131,phase: Validation: 132it [00:31,  4.19it/s, val acc:  0.508,val loss:  1.846]    Not a better score
Epoch 132,phase: Validation: 133it [00:32,  4.17it/s, val acc:  0.500,val loss:  1.850]    Not a better score
Epoch 133,phase: Validation: 134it [00:32,  4.11it/s, val acc:  0.500,val loss:  1.851]    Not a better score
Epoch 134,phase: Validation: 135it [00:32,  4.21it/s, val acc:  0.484,val loss:  1.861]    Not a better score
Epoch 135,phase: Validation: 136it [00:32,  4.20it/s, val acc:  0.516,val loss:  1.843]    Not a better score
Epoch 136,phase: Validation: 137it [00:33,  4.23it/s, val acc:  0.516,val loss:  1.837]    Not a better score
Epoch 137,phase: Validation: 138it [00:33,  4.22it/s, val acc:  0.500,val loss:  1.841]    Not a better score
Epoch 138,phase: Validation: 139it [00:33,  4.20it/s, val acc:  0.500,val loss:  1.842]    Not a better score
Epoch 139,phase: Validation: 140it [00:33,  4.23it/s, val acc:  0.508,val loss:  1.848]    Not a better score
Epoch 140,phase: Validation: 141it [00:34,  4.08it/s, val acc:  0.477,val loss:  1.881]    Not a better score
Epoch 141,phase: Validation: 142it [00:34,  4.11it/s, val acc:  0.484,val loss:  1.874]    Not a better score
Epoch 142,phase: Validation: 143it [00:34,  4.14it/s, val acc:  0.516,val loss:  1.842]    Not a better score
Epoch 143,phase: Validation: 144it [00:34,  4.15it/s, val acc:  0.492,val loss:  1.845]    Not a better score
Epoch 144,phase: Validation: 145it [00:35,  3.97it/s, val acc:  0.500,val loss:  1.845]    Not a better score
Epoch 145,phase: Validation: 146it [00:35,  4.04it/s, val acc:  0.516,val loss:  1.840]    Not a better score
Epoch 146,phase: Validation: 147it [00:35,  4.09it/s, val acc:  0.500,val loss:  1.862]    Not a better score
Epoch 147,phase: Validation: 148it [00:35,  4.16it/s, val acc:  0.500,val loss:  1.863]    Not a better score
Epoch 148,phase: Validation: 149it [00:35,  4.10it/s, val acc:  0.508,val loss:  1.842]    Not a better score
Epoch 149,phase: Validation: 150it [00:36,  4.20it/s, val acc:  0.500,val loss:  1.843]    Not a better score
Epoch 150,phase: Validation: 151it [00:36,  4.23it/s, val acc:  0.523,val loss:  1.838]    Not a better score
Epoch 151,phase: Validation: 152it [00:36,  4.24it/s, val acc:  0.516,val loss:  1.841]    Not a better score
Epoch 152,phase: Validation: 153it [00:36,  4.14it/s, val acc:  0.508,val loss:  1.844]    Not a better score
Epoch 153,phase: Validation: 154it [00:37,  4.04it/s, val acc:  0.516,val loss:  1.843]    Not a better score
Epoch 154,phase: Validation: 155it [00:37,  4.09it/s, val acc:  0.492,val loss:  1.843]    Not a better score
Epoch 155,phase: Validation: 156it [00:37,  4.19it/s, val acc:  0.492,val loss:  1.850]    Not a better score
Epoch 156,phase: Validation: 157it [00:37,  4.21it/s, val acc:  0.508,val loss:  1.842]    Not a better score
Epoch 157,phase: Validation: 158it [00:38,  4.15it/s, val acc:  0.516,val loss:  1.842]    Not a better score
Epoch 158,phase: Validation: 159it [00:38,  4.15it/s, val acc:  0.508,val loss:  1.849]    Not a better score
Epoch 159,phase: Validation: 160it [00:38,  4.16it/s, val acc:  0.516,val loss:  1.842]    Not a better score
Epoch 160,phase: Validation: 161it [00:38,  4.13it/s, val acc:  0.516,val loss:  1.838]    Not a better score
Epoch 161,phase: Validation: 162it [00:39,  4.11it/s, val acc:  0.516,val loss:  1.838]    Not a better score
Epoch 162,phase: Validation: 163it [00:39,  4.11it/s, val acc:  0.508,val loss:  1.843]    Not a better score
Epoch 163,phase: Validation: 164it [00:39,  4.16it/s, val acc:  0.508,val loss:  1.847]    Not a better score
Epoch 164,phase: Validation: 165it [00:39,  4.21it/s, val acc:  0.484,val loss:  1.868]    Not a better score
Epoch 165,phase: Validation: 166it [00:40,  4.21it/s, val acc:  0.500,val loss:  1.851]    Not a better score
Epoch 166,phase: Validation: 167it [00:40,  4.19it/s, val acc:  0.484,val loss:  1.857]    Not a better score
Epoch 167,phase: Validation: 168it [00:40,  4.09it/s, val acc:  0.477,val loss:  1.869]    Not a better score
Epoch 168,phase: Validation: 169it [00:40,  4.11it/s, val acc:  0.500,val loss:  1.862]    Not a better score
Epoch 169,phase: Validation: 170it [00:41,  4.14it/s, val acc:  0.477,val loss:  1.866]    Not a better score
Epoch 170,phase: Validation: 171it [00:41,  4.17it/s, val acc:  0.453,val loss:  1.894]    Not a better score
Out of patience returning the best model
Best val acc: 0.7252747252747253, Best val loss: 1.836460828781128, Best train acc: 0.6916666666666667, Best train loss: 1.8858896493911743
Epoch 170,phase: Validation: 171it [00:41,  4.14it/s, val acc:  0.453,val loss:  1.894]
```

Figure 1. Final Test Performance of the Model for Section 1.6

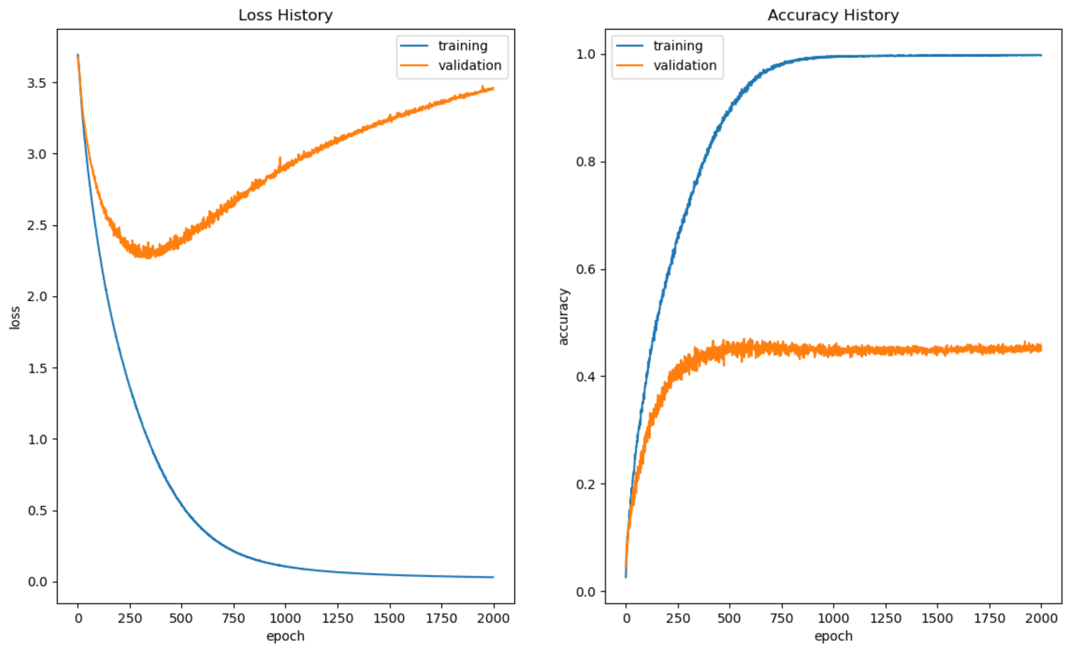
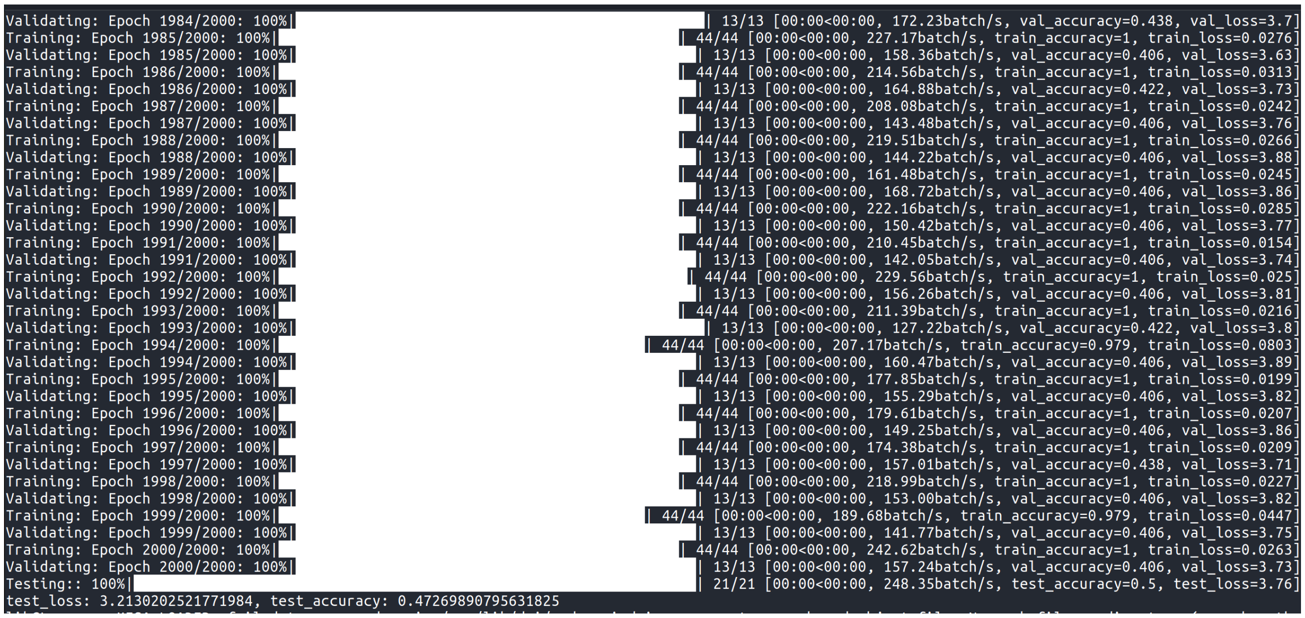


Figure 2. Accuracy History and Loss History for Section 2.1



```
Validating: Epoch 1984/2000: 100%| | 13/13 [00:00<00:00, 172.23batch/s, val_accuracy=0.438, val_loss=3.7]
Training: Epoch 1985/2000: 100%| | 44/44 [00:00<00:00, 227.17batch/s, train_accuracy=1, train_loss=0.0276]
Validating: Epoch 1985/2000: 100%| | 13/13 [00:00<00:00, 158.36batch/s, val_accuracy=0.406, val_loss=3.63]
Training: Epoch 1986/2000: 100%| | 44/44 [00:00<00:00, 214.56batch/s, train_accuracy=1, train_loss=0.0313]
Validating: Epoch 1986/2000: 100%| | 13/13 [00:00<00:00, 164.88batch/s, val_accuracy=0.422, val_loss=3.73]
Training: Epoch 1987/2000: 100%| | 44/44 [00:00<00:00, 208.08batch/s, train_accuracy=1, train_loss=0.0242]
Validating: Epoch 1987/2000: 100%| | 13/13 [00:00<00:00, 143.48batch/s, val_accuracy=0.406, val_loss=3.76]
Training: Epoch 1988/2000: 100%| | 44/44 [00:00<00:00, 219.51batch/s, train_accuracy=1, train_loss=0.0266]
Validating: Epoch 1988/2000: 100%| | 13/13 [00:00<00:00, 144.22batch/s, val_accuracy=0.406, val_loss=3.88]
Training: Epoch 1989/2000: 100%| | 44/44 [00:00<00:00, 161.48batch/s, train_accuracy=1, train_loss=0.0245]
Validating: Epoch 1989/2000: 100%| | 13/13 [00:00<00:00, 168.72batch/s, val_accuracy=0.406, val_loss=3.86]
Training: Epoch 1990/2000: 100%| | 44/44 [00:00<00:00, 222.16batch/s, train_accuracy=1, train_loss=0.0285]
Validating: Epoch 1990/2000: 100%| | 13/13 [00:00<00:00, 150.42batch/s, val_accuracy=0.406, val_loss=3.77]
Training: Epoch 1991/2000: 100%| | 44/44 [00:00<00:00, 210.45batch/s, train_accuracy=1, train_loss=0.0154]
Validating: Epoch 1991/2000: 100%| | 13/13 [00:00<00:00, 142.05batch/s, val_accuracy=0.406, val_loss=3.74]
Training: Epoch 1992/2000: 100%| | 44/44 [00:00<00:00, 229.56batch/s, train_accuracy=1, train_loss=0.025]
Validating: Epoch 1992/2000: 100%| | 13/13 [00:00<00:00, 156.26batch/s, val_accuracy=0.406, val_loss=3.81]
Training: Epoch 1993/2000: 100%| | 44/44 [00:00<00:00, 211.39batch/s, train_accuracy=1, train_loss=0.0216]
Validating: Epoch 1993/2000: 100%| | 13/13 [00:00<00:00, 127.22batch/s, val_accuracy=0.422, val_loss=3.8]
Training: Epoch 1994/2000: 100%| | 44/44 [00:00<00:00, 207.17batch/s, train_accuracy=0.979, train_loss=0.0803]
Validating: Epoch 1994/2000: 100%| | 13/13 [00:00<00:00, 160.47batch/s, val_accuracy=0.406, val_loss=3.89]
Training: Epoch 1995/2000: 100%| | 44/44 [00:00<00:00, 177.85batch/s, train_accuracy=1, train_loss=0.0199]
Validating: Epoch 1995/2000: 100%| | 13/13 [00:00<00:00, 155.29batch/s, val_accuracy=0.406, val_loss=3.82]
Training: Epoch 1996/2000: 100%| | 44/44 [00:00<00:00, 179.61batch/s, train_accuracy=1, train_loss=0.0207]
Validating: Epoch 1996/2000: 100%| | 13/13 [00:00<00:00, 149.25batch/s, val_accuracy=0.406, val_loss=3.86]
Training: Epoch 1997/2000: 100%| | 44/44 [00:00<00:00, 174.38batch/s, train_accuracy=1, train_loss=0.0209]
Validating: Epoch 1997/2000: 100%| | 13/13 [00:00<00:00, 157.01batch/s, val_accuracy=0.438, val_loss=3.71]
Training: Epoch 1998/2000: 100%| | 44/44 [00:00<00:00, 218.99batch/s, train_accuracy=1, train_loss=0.0227]
Validating: Epoch 1998/2000: 100%| | 13/13 [00:00<00:00, 153.00batch/s, val_accuracy=0.406, val_loss=3.82]
Training: Epoch 1999/2000: 100%| | 44/44 [00:00<00:00, 189.68batch/s, train_accuracy=0.979, train_loss=0.0447]
Validating: Epoch 1999/2000: 100%| | 13/13 [00:00<00:00, 141.77batch/s, val_accuracy=0.406, val_loss=3.75]
Training: Epoch 2000/2000: 100%| | 44/44 [00:00<00:00, 242.62batch/s, train_accuracy=1, train_loss=0.0263]
Validating: Epoch 2000/2000: 100%| | 13/13 [00:00<00:00, 157.24batch/s, val_accuracy=0.406, val_loss=3.73]
Testing:: 100%| | 21/21 [00:00<00:00, 248.35batch/s, test_accuracy=0.5, test_loss=3.76]
test_loss: 3.2130202521771984, test_accuracy: 0.47269890795631825
```

Figure 3. Final Test Performance of the Model for Section 2.1

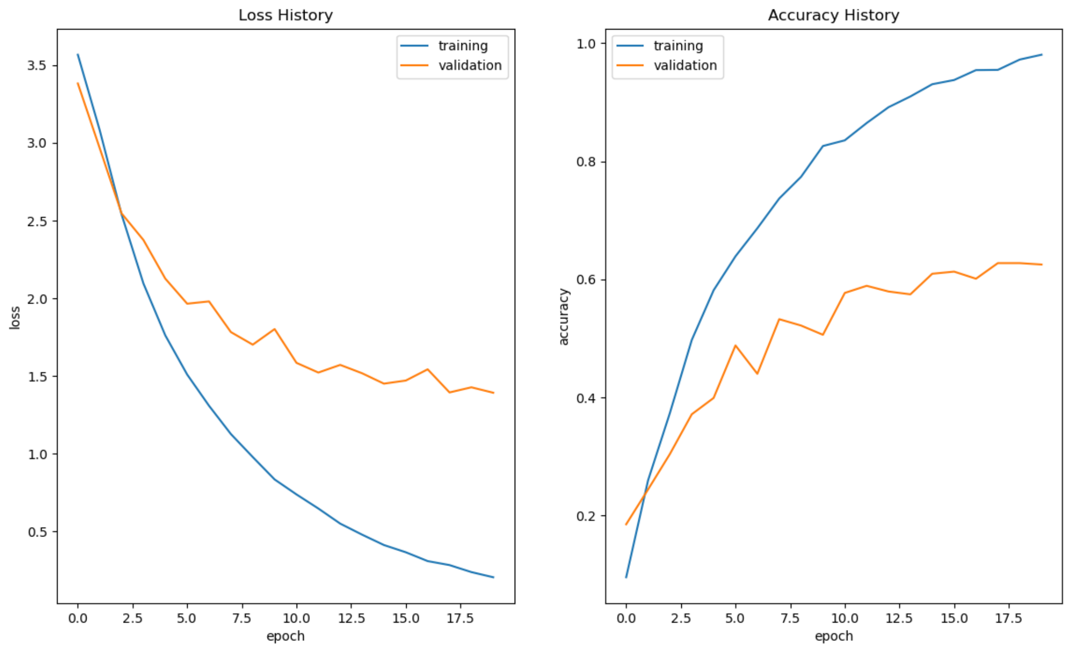
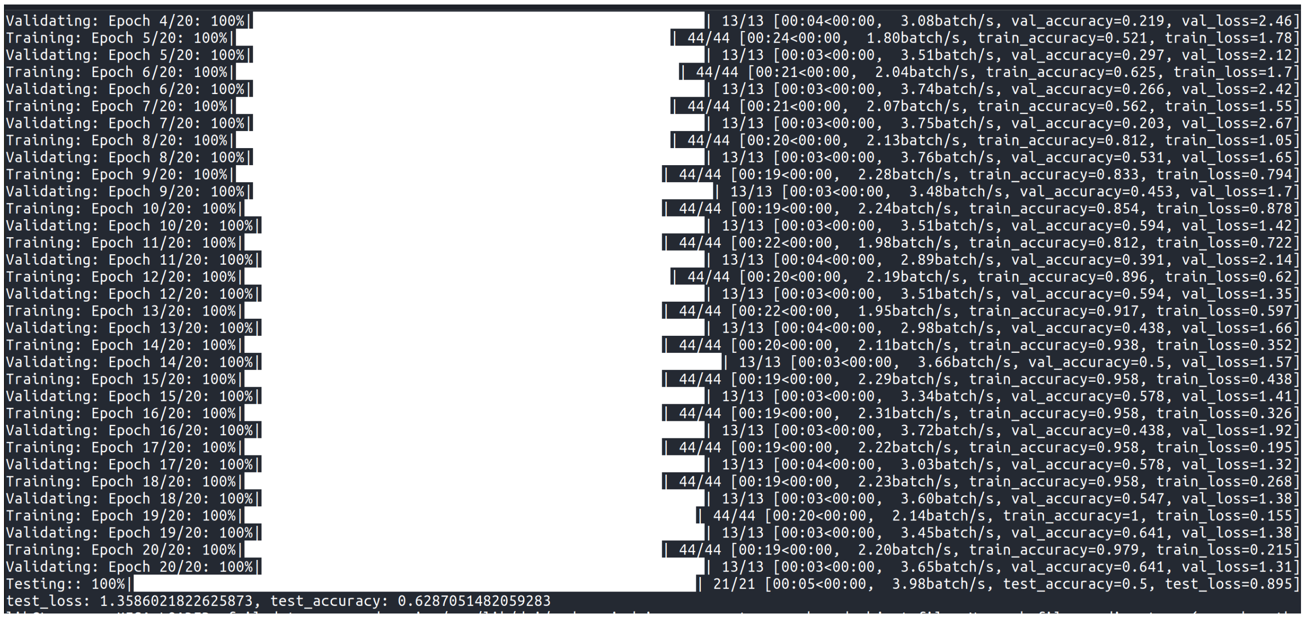


Figure 4. Accuracy History and Loss History for section 2.2



```
Validating: Epoch 4/20: 100%|| 13/13 [00:04<00:00,  3.08batch/s, val_accuracy=0.219, val_loss=2.46]
Training: Epoch 5/20: 100%|| 44/44 [00:24<00:00,  1.80batch/s, train_accuracy=0.521, train_loss=1.78]
Validating: Epoch 5/20: 100%|| 13/13 [00:03<00:00,  3.51batch/s, val_accuracy=0.297, val_loss=2.12]
Training: Epoch 6/20: 100%|| 44/44 [00:21<00:00,  2.04batch/s, train_accuracy=0.625, train_loss=1.7]
Validating: Epoch 6/20: 100%|| 13/13 [00:03<00:00,  3.74batch/s, val_accuracy=0.266, val_loss=2.42]
Training: Epoch 7/20: 100%|| 44/44 [00:21<00:00,  2.07batch/s, train_accuracy=0.562, train_loss=1.55]
Validating: Epoch 7/20: 100%|| 13/13 [00:03<00:00,  3.75batch/s, val_accuracy=0.203, val_loss=2.67]
Training: Epoch 8/20: 100%|| 44/44 [00:20<00:00,  2.13batch/s, train_accuracy=0.812, train_loss=1.05]
Validating: Epoch 8/20: 100%|| 13/13 [00:03<00:00,  3.76batch/s, val_accuracy=0.531, val_loss=1.65]
Training: Epoch 9/20: 100%|| 44/44 [00:19<00:00,  2.28batch/s, train_accuracy=0.833, train_loss=0.794]
Validating: Epoch 9/20: 100%|| 13/13 [00:03<00:00,  3.48batch/s, val_accuracy=0.453, val_loss=1.7]
Training: Epoch 10/20: 100%|| 44/44 [00:19<00:00,  2.24batch/s, train_accuracy=0.854, train_loss=0.878]
Validating: Epoch 10/20: 100%|| 13/13 [00:03<00:00,  3.51batch/s, val_accuracy=0.594, val_loss=1.42]
Training: Epoch 11/20: 100%|| 44/44 [00:22<00:00,  1.98batch/s, train_accuracy=0.812, train_loss=0.722]
Validating: Epoch 11/20: 100%|| 13/13 [00:04<00:00,  2.89batch/s, val_accuracy=0.391, val_loss=2.14]
Training: Epoch 12/20: 100%|| 44/44 [00:20<00:00,  2.19batch/s, train_accuracy=0.896, train_loss=0.62]
Validating: Epoch 12/20: 100%|| 13/13 [00:03<00:00,  3.51batch/s, val_accuracy=0.594, val_loss=1.35]
Training: Epoch 13/20: 100%|| 44/44 [00:22<00:00,  1.95batch/s, train_accuracy=0.917, train_loss=0.597]
Validating: Epoch 13/20: 100%|| 13/13 [00:04<00:00,  2.98batch/s, val_accuracy=0.438, val_loss=1.66]
Training: Epoch 14/20: 100%|| 44/44 [00:20<00:00,  2.11batch/s, train_accuracy=0.938, train_loss=0.352]
Validating: Epoch 14/20: 100%|| 13/13 [00:03<00:00,  3.66batch/s, val_accuracy=0.5, val_loss=1.57]
Training: Epoch 15/20: 100%|| 44/44 [00:19<00:00,  2.29batch/s, train_accuracy=0.958, train_loss=0.438]
Validating: Epoch 15/20: 100%|| 13/13 [00:03<00:00,  3.34batch/s, val_accuracy=0.578, val_loss=1.41]
Training: Epoch 16/20: 100%|| 44/44 [00:19<00:00,  2.31batch/s, train_accuracy=0.958, train_loss=0.326]
Validating: Epoch 16/20: 100%|| 13/13 [00:03<00:00,  3.72batch/s, val_accuracy=0.438, val_loss=1.92]
Training: Epoch 17/20: 100%|| 44/44 [00:19<00:00,  2.22batch/s, train_accuracy=0.958, train_loss=0.195]
Validating: Epoch 17/20: 100%|| 13/13 [00:04<00:00,  3.03batch/s, val_accuracy=0.578, val_loss=1.32]
Training: Epoch 18/20: 100%|| 44/44 [00:19<00:00,  2.23batch/s, train_accuracy=0.958, train_loss=0.268]
Validating: Epoch 18/20: 100%|| 13/13 [00:03<00:00,  3.60batch/s, val_accuracy=0.547, val_loss=1.38]
Training: Epoch 19/20: 100%|| 44/44 [00:20<00:00,  2.14batch/s, train_accuracy=1, train_loss=0.155]
Validating: Epoch 19/20: 100%|| 13/13 [00:03<00:00,  3.45batch/s, val_accuracy=0.641, val_loss=1.38]
Training: Epoch 20/20: 100%|| 44/44 [00:19<00:00,  2.20batch/s, train_accuracy=0.979, train_loss=0.215]
Validating: Epoch 20/20: 100%|| 13/13 [00:03<00:00,  3.65batch/s, val_accuracy=0.641, val_loss=1.31]
Testing:: 100%|| 21/21 [00:05<00:00,  3.98batch/s, test_accuracy=0.5, test_loss=0.895]
test_loss: 1.3586021822625873, test_accuracy: 0.6287051482059283
```

Figure 5. Final Test Performance of the Model for Section 2.2

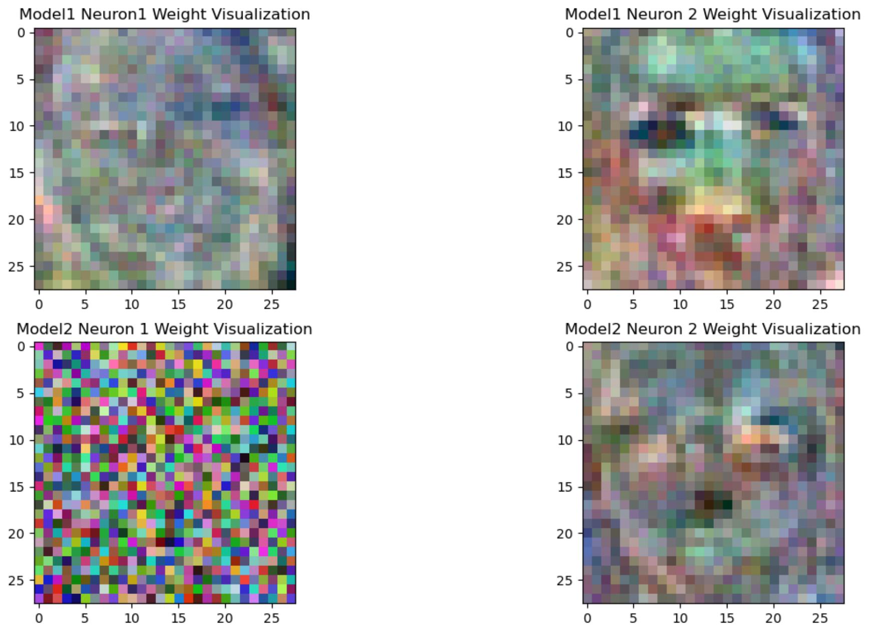


Figure 6. Results of Different Weight Visualizations for Section 2.3

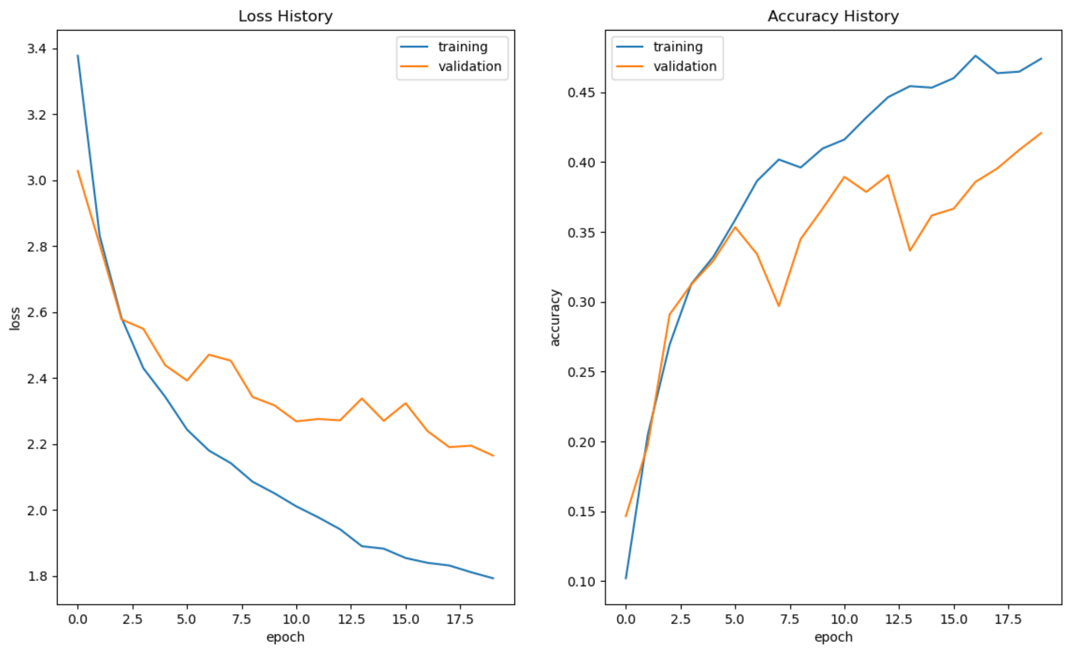


Figure 7. Accuracy History and Loss History for Section 2.4

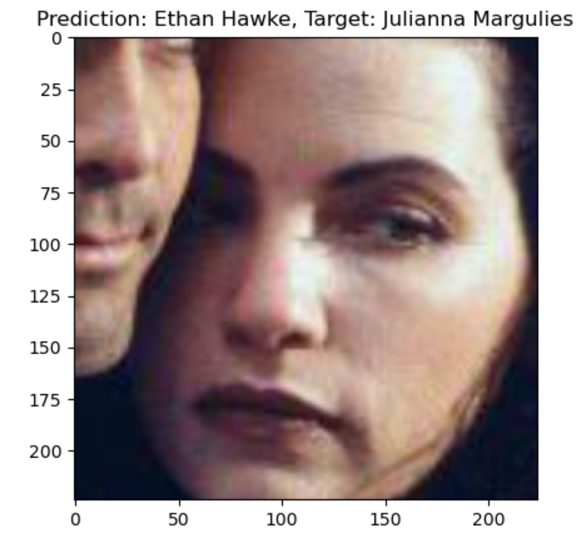


Figure 8. False Prediction for Sample Input for Section 2.4

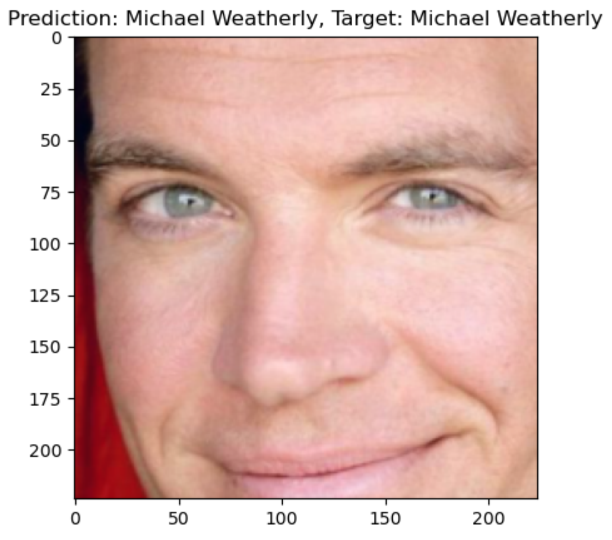


Figure 9. Correct Prediction for Sample Input for Section 2.4

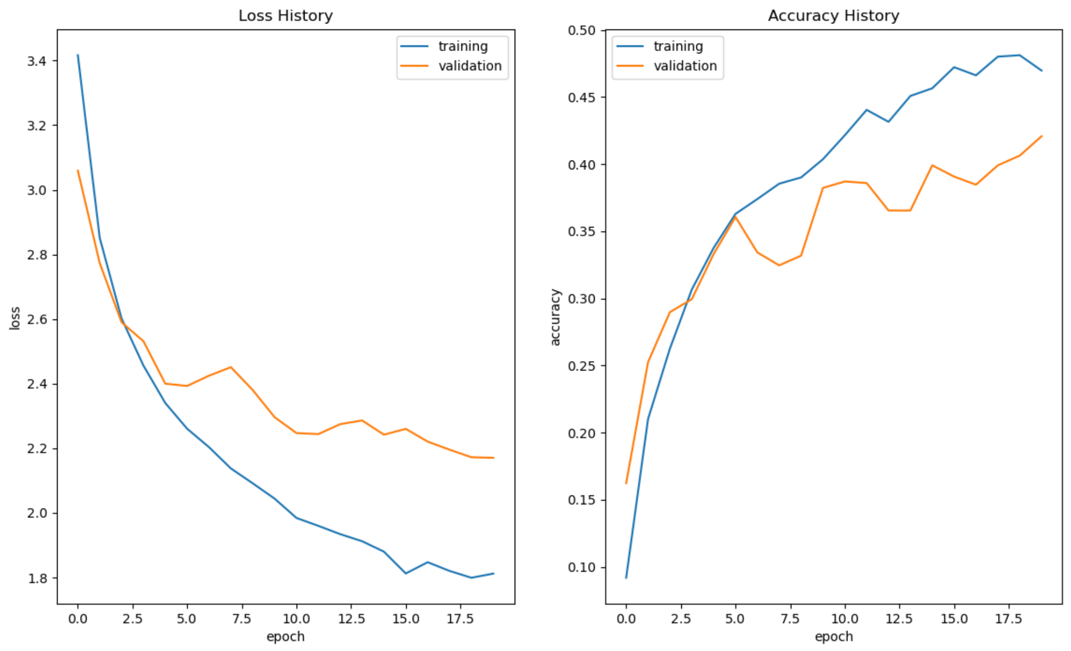


Figure 10. Accuracy History and Loss History for Section 2.5

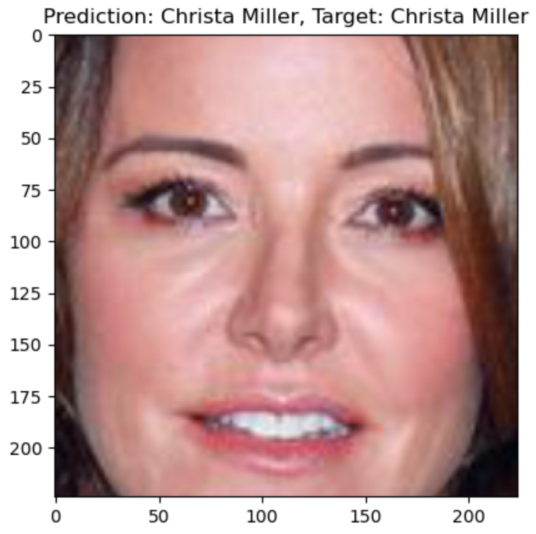


Figure 11. Sample Face Image used for Gradient Visualization and the Model's Prediction for Section 2.5

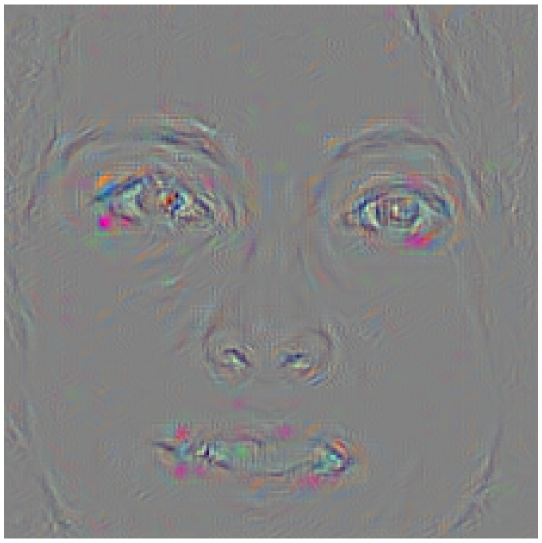


Figure 12. Corresponding Guided Backprop Color Result for Section 2.5

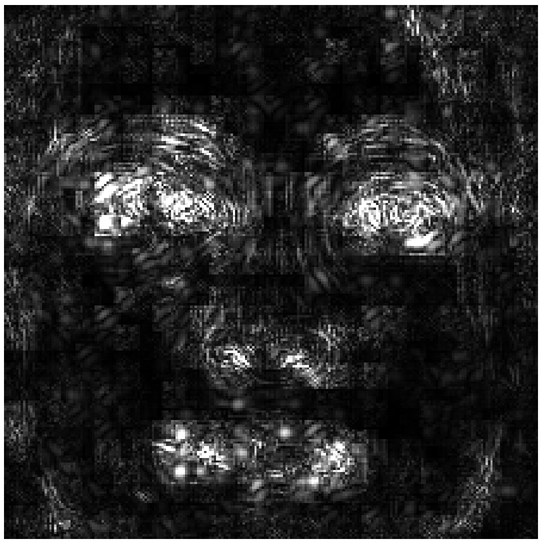


Figure 13. Corresponding Guided Backprop Gray Result for Section 2.5

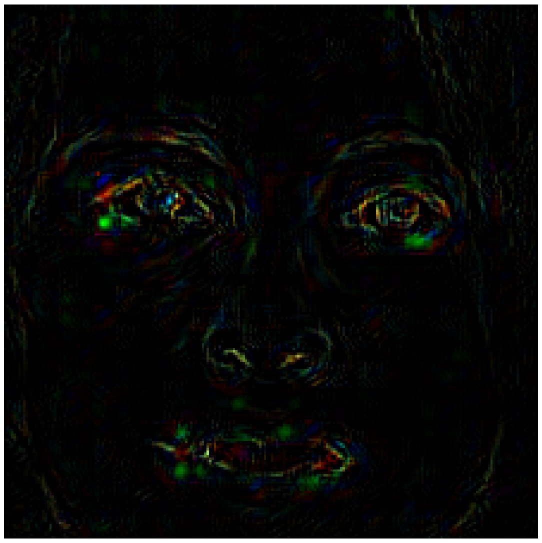


Figure 14. Corresponding Guided Backprop Negative Saliency Result for Section 2.5

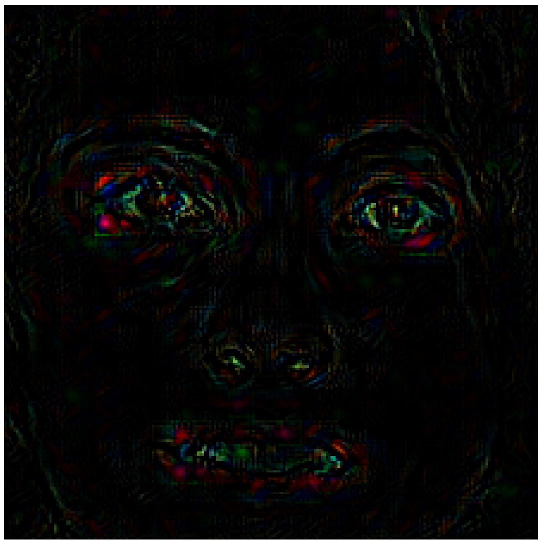


Figure 15. Corresponding Guided Backprop Positive Saliency Result for Section 2.5