Blink to Word with Convolutional Neural Networks (BWCNN)

Albara Ah Ramli
Dept. of computer science
UC Davis
arramli@ucdavis.edu

Vishal I B
Dept. of ECE
UC Davis
vib@ucdavis.edu

Rahul Krishnamoorthy
Dept. of ECE
UC Davis
rkrishnamoorthy@ucdavis.edu

Xin Liu
Dept. of computer science
UC Davis
rexliu@ucdavis.edu

***Abstract*—Amyotrophic lateral sclerosis or ALS is a progressive neurodegenerative disease that affects nerve cells in the brain and the spinal cord. An individual affected by this disease loses all motor functions. Even if a person loses all the motor functions, they can still blink their eyes. We present a system (BWCNN) to use the eye blinks as a mode of communication with the outside world. The system uses a Convolutional Neural Network (CNN) to predict the state of the eyes and this state is used to find the blinking pattern. Once the pattern is obtained, it is mapped with a collection of phrases with a label for each pattern. Several state-of-the-art ConvNet architectures such as ResNet, SqueezeNet, DenseNet, and InceptionV3 were implemented in this system to evaluate their performance in order to choose the best one for eye state recognition. By tuning the hyperparameters of the networks such as batch size and number of iterations, we analyzed the performance of different architectures based on the accuracy and the time taken for prediction. Even though the highest accuracy was obtained from ResNet architecture (99.26%) with 117 ms latency (response time), we used InceptionV3 architecture with 99.20% accuracy and 94 ms latency. The goal of our system is to strike a balance between the accuracy and the latency. The high accuracy makes our prediction better while small latency makes the prediction faster.**

***Index Terms*—CNN, Neural Network, Tranfser learning, Resnet, Inception, InceptionV3, DenseNet, Squeezenet, ALS.**

I. Introduction

There are several medical disorders that cause an individual to be paralyzed. In most cases, the patient loses the ability to communicate with the outside world even though their intelligence is still not affected. The specific problem here is to create a communication mechanism to aid paralyzed individuals to converse by blinking the eyes. The system needs to be robust and provide accurate results in real-time. Certain approaches have been used in the past to solve this problem. One approach [1] is to use IR sensors to estimate the state of the eyes to detect blinking and conversion to Morse code. This approach had multiple shortcomings such as the IR sensor getting irradiated by other sources resulting in false eye-blinks and prolong usage may involve an increased risk of cataract formation. A novel method [2] that can differentiate between short and long blinks was introduced in this paper. The system is intended to provide an alternate input modality to allow people with severe disabilities to access a computer. The images at the current and previous time stamps were taken and the bidirectional image is formed to detect motion in the scene to detect brightness changes in the scene to obtain eye candidates. This is used to detect the location of the eyes. Correlation scores for the user's eye at each frame of open and closed eyes were computed to detect blinks. This method recorded an accuracy of 95.6% but a task of spelling 2 words containing 8 letters took about 95 seconds. These image processing techniques are not robust and are prone to failure under limited lighting conditions. The real-time accuracy of these approaches is also low. Moreover, these approaches require the patient to learn the Morse coding scheme to communicate and interpret the messages. An efficient system for eye blink detection is presented in [3]. This method uses Haar-cascade classifiers for the detection of the face and the eye positions relative to the face. An eye blinking detection based on eyelids state (closed or open) is used for controlling android mobile phones. The application was used in different environments to study the effects of varying lighting conditions and distances from the eye. The performance of Haar-cascade classifiers is not completely invariant to the change in lighting conditions and hence there is a decay in performance. Artificial light was used and an overall accuracy of 98% was obtained. A low-cost implementation [4] of an eye-blink-based communication aid for ALS patients is presented in this paper. The position of the pupil is found by generating four rectangles around it. Template matching is used to track the eye and detect eye blinks using hierarchical optical flow. The optical-flow method used here is also dependent on various assumptions (1) the brightness of any feature point is constant over time and (2) nearby points in the image move in a similar manner. The implementation, though being a low-cost implementation, has an accuracy of 94.75% during the typing test. Though the accuracy is comparable, the algorithm takes a lot of time to type a phrase. It takes approximately 2 seconds for the algorithm to generate a single scan of the eye which is a lot considering it is for a single character. Multiple methods dealing with the detection of the eye state have been implemented in [5]. One of the methods is based on ideogram selection by moving a cursor on the screen. But this requires the user's gaze to be focused and not move around since that might lead to false positives. The second method is based on image binarization using an adaptive threshold

determined using the integral sum image for the selection of ideograms using voluntary eye blink detection which is defined as a longer blinking interval than a normal physiological one. Though this method seems user-friendly, as opposed to the first method, it requires a high learning curve as the user will have to learn the system in order to determine the duration of blinks necessary for the ideogram selection. Considering the stated limitations and accuracy of 91.04%, this method does not seem feasible. A vision-based human-computer interface [6] is presented in the paper. The interface detects eye-blinks and interprets them as control commands. A robust system that can detect faces and eyes using Haar-Cascade filters and can detect and interpret blinks using template matching with a fixed threshold is presented. An accuracy of 95.35% is achieved but the algorithm takes approximately 12 seconds to complete the process of detecting blinks which is considerably longer if communication needs to be carried out in real-time. This paper [7] presents a real-time detection and classification between eye blink, left and right wink. The process of blink detection has been divided into four parts. The first part is face localization in facial images acquired through a video camera. Following that, eye pair localization, pixels' motion analysis using optical flow technique, and classification of eye blinks were performed. This approach was accurate 96, 92 and 88% of the time for detection of eye blink, left wink and a right wink, respectively. The latency for the detection of a single blink was found to be 250ms. A wearable device [8] to detect eye blinks for alleviating dry eyes and computer vision syndrome was proposed in this paper. The prototypes sense infrared reflections from the cornea of the person of interest while blinking. Eye blinks are evaluated using discontinuity in infrared reflections. This method [8] captured 85.2% of all the blinks that occurred during testing. The IR sensors tend to show false readings when the orientations are altered and hence unreliable to use in realistic scenarios. Any facial movements such as laughing, talking and yawning can also induce errors.

We have created a vision-based system to tackle all the limitations of the previous methods. Our system uses an InceptionV3 architecture and we have achieved an accuracy of 99.20% compared with around 80% accuracy in the previously stated methods. It is a completely safe method unlike the method based on Infrared which causes cataracts. (Our system detects the state of eyes even under low lighting conditions, unlike the Haar-cascade based methods). We have a pre-defined dictionary of sentences corresponding to the blink pattern which we map in real-time. Since it uses predefined phrases instead of using Morse code or any other encoding patterns, the patient does not have to put a lot of effort to spell out the entire sentence. In order to sum up, we want to create a system that works almost flawlessly in real-time and is completely safe to use and this can be summed up using (1) and (2). P refers to the performance of the system in our case is the accuracy. S refers to the System itself. W refers to the weights that are going to be trained to achieve this performance. A refers to the architecture that is going to maximise the performance. AS refers to the set of all architectures that can be used for this purpose. tuning time is the prediction time for the model on the validation set and TC is the time constraint which is 100 ms for our case.

$$\max P(S, W, A); A \in AS \tag{1}$$

$$s.t\ \ prediction\ \ time \leq TC \tag{2}$$

II. Methods

The goal of this project is to detect if the person of interest blinks their eyes and to map the sequence of blinks to a particular entry in the dictionary of phrases. In order to achieve this, one has to detect the state of eyes (open or close). If an 'open' state is followed by a 'close' state in the subsequent frames, the system detects an eye blink. The project is divided into three phases as shown in Fig. 1. Phase one: preprocessing data by capturing and saving the stream of images. Phase 2: using the CNN model that we trained to predict the state of user's eyes which includes training a convolutional neural network on the labeled dataset to detect the state of user's eyes. Phase 3: using the output vector of the predictor, map the eyeblinks to a phrase in the dictionary.

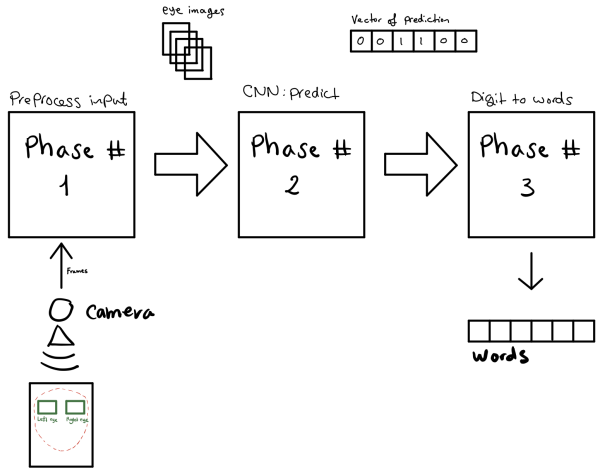

Fig. 1. The 3 phases of the BWCNN system.

In the phase one, the system preprocesses the data by capturing and saving the stream of frames. The frame is captured by the camera (PC, laptop, or IoT) and the stream of frames is saved as image files. In order to improve performance, the system only saves a designated eye area. This step is important to reduce the dimensionality of the saved images which will be fed as an input to the Neural Network in the next phase. By saving only the eyes area, the size is reduced from 10KB to 3KB. The size of the image can be further reduced from 3KB to 2KB by converting the images to grayscale.

In phase two, the system predicts the contents of the image. Based on the image obtained in the previous phase, the state of the eyes is predicted. In this phase, we deal with

the prediction of the image using a trained ConvNet. The fundamental challenge with deep learning is striking the right balance between generalization and optimization. The main question here is: given our dataset, what is the best CNN architecture to implement? Since this system runs in real-time, the latency (time taken for prediction) of the system is a very important factor. Thus, improving the accuracy of the prediction is not the only priority. The tradeoff between the accuracy and the latency (prediction time) will have to be taken into account. Moving forward, the question is, given our dataset, what is the best CNN architecture to implement in terms of accuracy and latency? In the coming subsection, we explain in detail the procedure we follow to choose the best architecture that we believe serves our goal. The input of this phase is one eye image and the output is a binary digit of zero or one. Zero represents the opened state and the one represents the closed state.

In the phase three, the system processes the output and maps it to the phrases. The system stores the output of the CNN as a vector of zeros and ones. Each blink is represented by 010. Based on the output vector, the number of blinks are calculated and is mapped with the phrases in the dictionary.

A. Phase 1: Preprocessing data by capturing and saving a stream of images

In this section, we present the method of obtaining the data and preprocessing it to be given as input to the ConvNet.

1) System Input: The system uses camera devices (PC, laptop, IoT) for capturing the frames. Regulars webcams are capable of capturing 30 frames per second. Choosing the right number of frames per second will directly affect the user experience. Since, The system runs in real-time, it is more effective to reduce the latency and not let the user wait for the results. At the same time using a small number of frames per second will lead to missing the eye blink and as a result, the CNN model mispredicts the state. Our experiment shows that 10 frames per second is a reasonable frame rate. So there is a frame being captured every 100ms, and in order for our system to avoid delay and operate in real-time, we need our model to predict each frame in less than 100ms. We further impose a constraint on the user that the closed state should be maintained for at least 200ms, so that two frames of that state are obtained for prediction. The first is the main prediction and the second is the redundant prediction which will be useful in case the user increases the blink rate or any other issue that leads to missing the current frame capture. Choosing the right architecture that has high accuracy and low latency is very critical. Thus, our trade-off here is choosing the right number of frames that serve the system without causing a huge delay. The system saves each frame as an Image of dimension 80X70 pixels and a size of 2KB and gray color. By converting them to grayscale, we reduced the frame size file from 3KB to 2KB.

B. Phase 2: Predict the contents of the image

In this section, we present the experiments to choose the best fitting neural network architecture for our application. Choosing the neural network architecture for a given data is a challenging task. The goal of our experiment here is to select the best-suited architecture among the four state-of-the-art architectures (SqueezNet, ResNet, InceptionV3 and DenseNet architecture). Not only the architecture but also the hyperparameters play a huge role in the model performance. The hyperparameter that we have chosen here is the batch size. We start by training the networks from scratch for different values of batch sizes and find the batch size that gives the best results. On finding the best batch size, we further explore the chances of improving the performance by using transfer learning with a pre-trained model and compare the results. By considering the trade-off between the accuracy and the time taken for prediction, we decide the best architecture that fits our system.

1) Training Dataset: We used the eye dataset from Media Research Lab (MRL) which is available for public use. The dataset contains 84,898 pictures of eyes taken from thirty-seven individuals consisting of thirty-three men and four women. Each image in the dataset was collected from one of the following sensors: Intel RealSense RS 300 sensor with a resolution of 640 x 480, IDS Imaging sensor with a resolution of 1280 x 1024, and Aptina sensor with a resolution of 752 x 480. The original dataset contains 6 different classes: 'gender', 'glasses', 'eye state', 'reflections', 'lighting conditions' and, 'sensor resolution'. But we do not need all these classes for our purpose. Therefore, we divided the entire dataset into just two classes - 'Open' and 'Close'. We split the dataset into training and test sets. The training set consists of 80% of the images amounting to 67,919 images and the test set has 16,979 images which are 20% of the total number. Both the training and test set has an 'Open' and a 'Close' class. The training directory thus has two folders named 'Open' and 'Close' and likewise for the test directory too. The folder names of the images act as the labels for each image in that folder. You can download our dataset from here: http://albara.ramli.net/b2wcnn/imx.zip

2) Training experments: Training the model is crucial to obtain an accurate detection of the eye's state. Apart from the accuracy, another important factor to consider is the latency for detection i.e., the time taken to make an accurate classification. The robustness of a model is measured based on these two factors and it is important to choose the right dataset and architecture for this purpose.

Deep learning is a field that is researched extensively. With that being said, there are a lot of potential algorithms and architectures that can help with our goal. Each of them has its own set of advantages and limitations and all of them tend to act differently with unseen data. Apart from the architecture, the hyperparameters used for training such as batch sizes play a huge role in the accuracy of the model. To find the model which can provide the best accuracy with the least latency, we implemented various state of the art architectures such as SqueezeNet, ResNet, InceptionV3 and DenseNet. We trained them from scratch by randomly initializing the weights. We can make a fair comparison of these architectures if they are trained with the same batch size and for the same number of

epochs. We also used the weights from the trained model to train the same architecture with different batch sizes. Based on how each model performs on the test set, we can decide on which architecture suits our goal the best. The following are the experiments that were performed to analyze the neural networks:

a) Train ResNet architecture from scratch: There are a number of hyperparameters that can be tuned during the training. One of the hyperparameters is the batch size. We train the ResNet architecture for 100 epochs using 6 different batch sizes on our dataset and calculate the overall accuracy. The performance metric used to analyze each batch size is the overall accuracy. Comparing the performance, we select the three best batch sizes. For our experiment, we tried batch size numbers 1, 2, 4, 8, 16 and 32. The results (Table. III) show that using batch sizes 8, 16 and 32 provides the best accuracy among all the different batch sizes.

b) Run 500 epochs: Based on our observation of this experiment. After finding the best batch sizes, we wanted to further improve the performance of our network. Since our experiment stops only at 100 epochs, training the network for more number of epochs might improve the performance. Training the network for all the batch sizes is computationally expensive, so the residual network is trained only on the three selected batches for 500 epochs. The obtained results (Table. IV) show that there is no further improvement in terms of accuracy.

c) Transfer learning: Another way of improving network performance would be to use transfer learning. Here our pre-trained model of the ResNet is used as the source network. The best weights obtained from the three selected batch sizes 8, 16, and 32, are used to train the source network. After 100 epochs of training, the results (Table. V) show no improvement in the accuracy beyond what we have already obtained in the previous step. We also implemented transfer learning using a pre-trained model of ResNet50 as our source network and training it on the same set of three different batch sizes as before. Similar to the previous results, after 100 epochs the there was no significant improvement in the accuracy of the network. (Table. VI)

d) Train Inception, SqueezNet and DeneseNet architectures from scratch: After implementing the different variations of ResNet, the next approach towards finding the best network would be to try different architectures. Trying different batch sizes and testing every case is very expensive. Thus, our assumption is that the same set of 3 batch sizes from ResNet would be the best performing batch sizes in the other architectures as well. To investigate this assumption we run the same experiment again but with different architectures. DenseNet, Inception, and SqueezeNet are the three different architectures that were chosen for this purpose. (Table. VII, Table. VIII, Table. IX). The time taken for prediction for each frame is calculated for the best performing batch size for each architecture. This is a measure of the latency of the networks. The comparison is made between each architecture based on the accuracy and the latency.

3) Testing the Model: The first phase of our project is to obtain the dataset of images containing the two eye states, 'open' and 'close' and these images are preprocessed to make them apt for the contiguous steps. Following the preprocessing phase, we move on to the next phase, detection. In this step, given an input image, we predict the state of the eyes by using one of the above-stated pre-trained networks. The network classifies them as zeros (for open state) and ones (for closed state). The third and final phase involves detecting eye blinks based on the eye state predictions and mapping it to a sequence of words.

C. Phase 3: Mapping

The output of the neural network is a binary classification. The system stores the output of the network as a vector of zeros and ones. In this phase, the system normalizes all the changing state to one edge. For example, the vector 00000110000 becomes 010, where one represents the image of a closed eye and zero represents the image of an open eye. The change in state is considered as an edge and all the redundant states do not matter. Each blink is represented by 010. Based on the output vector which contains a representation of the blink sequence, the number of blinks is calculated and is mapped with the phrases in the dictionary. We have a previously defined dictionary of phrases. These are basic phrases that we use in everyday life and the dictionary can be further increased with more phrases. The dictionary maps every phrase with a certain vector of zeros and ones. (See Table. II)

III. Results

This section provides the performance results obtained from different architectures. Table. II discusses the results obtained from training ResNet for 100 epochs on different batch sizes (1,2,4,8,16,32). Fig. 2 and Fig. 3 shows the validation accuracy and loss curve for this.

TABLE I
FINAL RESULTS

Architectur	*ResNet*	*DenseNet*	*SqueezeNet*	*InceptionV3*
Batch size	*16*	*8*	*16*	*16*
Last epoch were models improved out of 100	*55*	*55*	*1*	*22*
# of layers	*107*	*242*	*26*	*189*
Total params	*23,591,810*	*7,039,554*	*723,522*	*21,806,882*
Trainable params	*6,955,906*	*6,955,906*	*723,522*	*21,772,450*
Non-trainable params	*83,648*	*83,648*	*0*	*34,432*
model size	*283MB*	*85MB*	*8MB*	*262MB*
accuracy	99.26%	99.24%	49.40%	99.20%
Avg latency (ms) to predict an image	117.28	146.09	13.64	94.1

The results show the accuracy obtained for each batch and the last epoch where there was an improvement in the

accuracy. It can be seen that training the network for 100 epochs on a batch size of 16 yields the best accuracy of 99.26%. It can also be seen that there is no improvement in accuracy after 55 epochs. Fig. 4 and Fig. 5 shows the validation accuracy and loss curve for batch size 16 for ResNet.

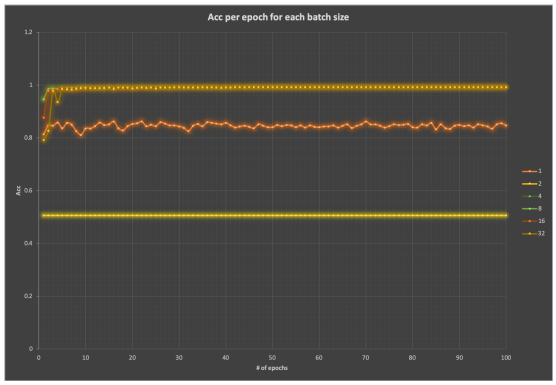

Fig. 2. ResNet validation accuracy curves for batch sizes 1, 4, 8, 16, 32

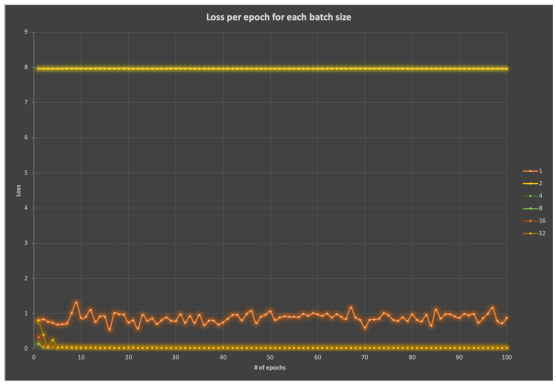

Fig. 3. ResNet validation loss curves for batch sizes 1, 4, 8, 16, 32

Table. III shows these results for 500 epochs for the three best batch sizes (8,16,32). Fig. 14 and Fig. 15 show the validation accuracy and loss curves for this. Our trained ResNet model was trained again using transfer learning, by initializing the weights obtained from our model. Fig. 10 and Fig. 11 shows the validation accuracy and loss curve for this. We also used transfer learning to train ResNet50 for our dataset by initializing it with our weights. These results are shown in Table. IV and Table. V respectively while Fig. 12 and Fig. 13 shows the validation accuracy and loss curve for this. Table. VI, Table. VII and Table. VIII present the results for different architectures. Table. VI encompasses the same category of results for the DenseNet architecture for the three batch sizes (8,16,32) that were experimentally determined to be best for ResNet. Fig. 16 and Fig. 17 shows the accuracy and loss curve for this. DenseNet has the best accuracy of 99.24% when trained for 100 epochs on a batch size of 8. Fig. 6 and Fig. 7 shows the validation accuracy and loss curve for DenseNet for all batch sizes. The results obtained from training the SqueezeNet and InceptionV3 architectures are presented in table 6. SqueezeNet seems to be the least performing among all the architectures with an accuracy of 49.40% for 100 epochs and a batch size of 16, and also there is not a change in the accuracy after the first epoch which is evident from Fig. 20 and Fig. 21. The final comparison between the best results obtained from the different architectures can be seen in Table. I. We can see that InceptionV3, DenseNet and ResNet has simmilar accuracies but the InceptionV3 model has the lowest prediciton time so we implement the InceptionV3 to predit the state of the system. Apart from the accuracy of

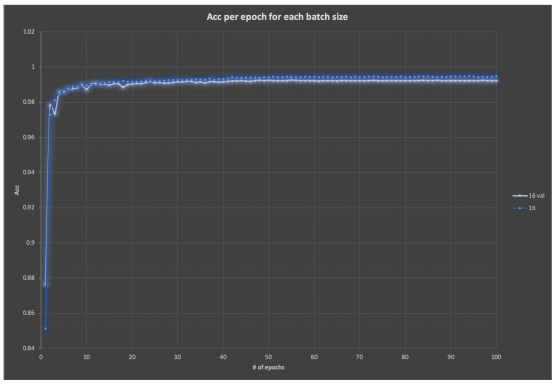

Fig. 4. ResNet traininng and validation accuracy curves for batch sizes 16 (best accuracy)

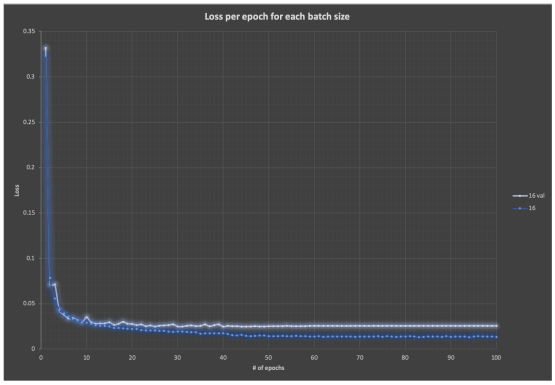

Fig. 5. ResNet traininng and validation loss curves for batch sizes 16 (best accuracy)

the networks this table also contains the time (in milliseconds) required for the prediction of each frame of image for different architectures for our application. We can see that ResNet has the best accuracy of the lot, but has a high latency of 117.28ms. InceptionV3 has an accuracy of 99.20% which is close to ResNet but with a lower latency of 94.1ms. Fig. 8 and Fig. 9 shows the validation accuracy and loss curve for InceptionV3 for the batch size of 16 while Fig. 18 and Fig. 19 shows the same for all batch sizes. The trade-off between accuracy and latency need to be considered to select a network for any application. Thus, the overall accuracy of the system is 99.20% and the latency is 94ms.

IV. Discussion

We trained different architectures with different hyperparameters to identify which combination gives the best accuracy with the lowest latency. This is our basis for comparison to find the right model for our application. To make it clear, our priority is to obtain the highest accuracy on the test set and if the accuracies are comparable, we can compare the latencies and choose the one with the lowest latency. For the sake of conducting a clear comparative analysis, we are comparing the results of each architecture for a constant batch size of 8. SqueezeNet received the lowest accuracy with the least number of parameters. The DenseNet, ResNet and, Inception-v3 acquired accuracies in the same range of 99.20% and above. Since they have similar accuracies, it makes sense to compare their latencies since our application must work in real time. On making this comparison, we found that the Inception -v3 acquired the least latency, making it the appropriate model for this project.

ResNet which is short for Residual network, created by the Mircrosoft team, gave us the best accuracy and had the greatest number of parameters. It also had the largest number of layers. Since it has a very deep network, it also must deal with the vanishing gradient problem. ResNet tackles this problem by creating several "identity shortcut connections" that skips one or more layers which means that there is a path from one layer to another. Since there is a direct connection between one layer to another [9], during forward propagation, integrity of information can be protected by directly sending the input to the next layer through shortcut paths. Also, during backpropagation, since there are shortcut paths, the gradients have to move through lesser layers than usual which reduces the vanishing gradient problem significantly. So, based on our understanding of ResNet, we believe that the shortcut paths allowed to improve the accuracy of prediction. We introduced transfer learning for the ResNet architecture, in the hopes of improving the accuracy and latency. We used the weights trained on the official ResNet50. We see that transfer learning does not help us in improving the testing accuracy and latency, but it did help us in converging to optimality much more easily than by randomly initializing the weights.

DenseNet breaks away from the stereotype of deepening network layers and widening network architecture to improve network performance. Through feature reuse and Bypass setting, it can not only greatly reduce the number of parameters in the network, but also alleviate the emergence of the gradient vanishing problem [10]. DenseNet has several advantages. Firstly, it could reduce gradient vanishing problems. Secondly, it could enhance the spread of features. Thirdly, feature reuse is encouraged in this network architecture. Finally, it also could reduce the number of entries. In this network, there is a direct connection between any two layers, which means that the input of each layer of the network is the union of the output of all the previous layers, and the feature graph learned by this layer will be directly passed to all the subsequent layers as input. This explains how the model reaches very high accuracy with this architecture. They have designed the output channel in such a way that the convolution layer is very small. We think that this acts as a bottleneck and thus increases the time for prediction.

Since latency is an important concern for our project too, we worked on the Squeezenet architecture in the aim of reducing the prediction time required to less than 100ms. The main idea behind SqueezeNet is to use 1x1 pointwise filters replacing 3x3 filters. The building brick of SqueezeNet is called the 'fire module' which contains a Squeeze layer and an expand layer. SqueezeNet stacks a bunch of fire modules along with a few pooling layers. The Squeeze layer reduces the depth to a small number while the expand layer increases it [11]. They both act together to keep the feature map to be of the same size at the end. We believe that the squeezeNet did not receive a high accuracy because it does not have a deep network and has a very low number of parameters to train on.

All mainstream Deep learning networks have large number of layers. Large number of layers tend to increase the accuracy, but this improved accuracy is accompanied by several disadvantages. Firstly, very deep networks are prone to overfitting and if the training set is limited, it is difficult to generalize to any application. Also, Larger the network is, Greater the computational complexity. So, it is difficult to implement a network practically. Thirdly, we also have the vanishing gradient problem which we must deal with for very large networks. It gets extremely difficult to optimize the cost function. Inception network works in increasing the depth and width of the network while also reducing the number of parameters to be trained. This is the reason why Inception has a prediction time lesser than all the other architectures. Inception version 3 transfers all the features of the previous versions along with its own distinctive characteristics. In the inception -v1, the sparse CNN is approximated with a dense construction. Since only a small number of neurons are effective, the kernel size is made smaller to make the same computations but in a less expensive manner. It uses convolutions of varied sizes and scales to capture the different features. It uses the scales which can reduce the computational cost [12]. This is another reason why the Inception gives out a lower latency. Another salient point is that the inception network has a bottleneck layer (1x1 convolutions) which can reduce the dimensionality immensely just saving up on computation power and time. Inception -v2 is an improvement to the first version where they introduced factorization. By factorization what they mean is that they have reduced the kernel size even more to make them computationally cheap. They have altered the 7x7 convolutions into two 3x3 convolutions since 7x7 convolutions are 2.78 times computationally expensive than 3x3. After that they have changed the 3x3 convolutions into a pair of 1x3 convolutions and 3x1 convolutions. This makes the computation 33% faster [13]. This feature is present in the final version of inception too and we believe this is another reason for the low latency that we achieved in our system. Adding to this feature, Inception -v3 also has batch normalization in the auxiliary classifiers, RMSprop optimizer and Label smoothing which would have

contributed to the fast training and good accuracy.

V. Conclusion

In this paper, we designed a system for ALS patients which could convert eye blink to word with CNN. The system uses a Convolutional Neural Network (CNN) to predict the state of the eyes of the person of interest and this was used to find the blinking pattern. We compared several CNN architectures and discussed hyperparameter selection in model training. For the evaluation, we tested our system using 16,979 facial images in our testing dataset and found that our proposed prediction model was efficient and effective. Results demonstrate that overall prediction accuracy is 99.20% and an average prediction time is 94ms with InceptionV3. In our future work, we plan to improve the eye detection and prediction speed. Maybe we could use a single network for detection and prediction to reduce the response time. We also plan to add more complex language models to better serve those who need this system.

VI. Syetem Source code, demo and dataset

a) : The system source code:
https://github.com/albararamli/b2wcnn

b) : The system dataset:
http://albara.ramli.net/b2wcnn/imx.zip

c) : Demo of the system:
https://www.youtube.com/watch?v=a2W6Ec3uCAM

References

[1] K. Mukherjee and D. Chatterjee, "Augmentative and Alternative Communication device based on eye-blink detection and conversion to Morse-code to aid paralyzed individuals," in 2015 International Conference on Communication, Information and Computing Technology (ICCICT), Mumbai, India, 2015, pp. 1–5.

[2] K. Grauman, M. Betke, J. Gips, and G. R. Bradski, "Communication via eye blinks - detection and duration analysis in real time," in Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, 2001, vol. 1, pp. I–I.

[3] A. Mohammed and Ms. Anwer, "Efficient Eye Blink Detection Method for disabled-helping domain," Int. J. Adv. Comput. Sci. Appl., vol. 5, Jun. 2014.

[4] M.-C. Su, C. Yeh, S. Lin, P. Wang, and S. Hou, "An implementation of an eye-blink-based communication aid for people with severe disabilities," presented at the ICALIP 2008 - 2008 International Conference on Audio, Language and Image Processing, Proceedings, 2008, pp. 351–356.

[5] A. Pasarica, R. G. Bozomitu, V. Cehan, and C. Rotariu, "Eye blinking detection to perform selection for an eye tracking system used in assistive technology," in 2016 IEEE 22nd International Symposium for Design and Technology in Electronic Packaging (SIITME), Oradea, Romania, 2016, pp. 213–216.

[6] A. Królak and P. Strumiłło, "Eye-blink detection system for human–computer interaction," Univers. Access Inf. Soc., vol. 11, no. 4, pp. 409–419, Nov. 2012.

[7] Singh, Hari and Singh, Jaswinder. (2018). Real-time eye blink and wink detection for object selection in HCI systems. Journal on Multimodal User Interfaces. 12. 10.1007/s12193-018-0261-7.

[8] A. Dementyev and C. Holz, "DualBlink: A Wearable Device to Continuously Detect, Track, and Actuate Blinking For Alleviating Dry Eyes and Computer Vision Syndrome," Proc. ACM Interact. Mob. Wearable Ubiquitous Technol., vol. 1, no. 1, pp. 1–19, Mar. 2017.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In CVPR, 2016. 2, 5, 6, 7.

[10] G. Huang, Z. Liu, K. Q. Weinberger, and L. Maaten. Densely connected convolutional networks. In CVPR, 2017. 2, 4, 6.

[11] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. arXiv preprint arXiv:1709.01507, 2017. 1, 2, 5, 7.

[12] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In CVPR, 2015. 1, 2, 4.

[13] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojn3. Rethinking the inception architecture for computer vision. In CVPR, 2016. 2, 4, 6.

VII. Author contributions

a) : Albara Ramli worked in designing ResNet from scratch for different batch sizes, and trained it for both 100 and 500 epochs. He also implemented the phase 1 and phase 3.

b) : Rahul Krishnamoorthy and Albara Ramli together worked on Transfer learning. They used the trained weights from the trained ResNet model trained previously to initialize the weights here but again trained from scratch. They also worked on transfer learning by getting the trained model from ResNet50 and use it for this application. They also worked together in designing the squeezenet.

c) : Vishal IB and Albara Ramli together worked on implementing the DenseNet architecture from scratch.

d) : Rex Liu and Albara Ramli together worked on implementing the InceptionV3 architecture from scratch.

APPENDIX

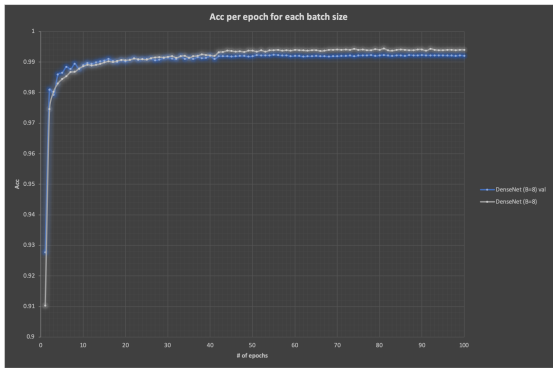

Fig. 6. DenseNet traininng and validation accuracy curves for batch sizes 8 (best accuracy)

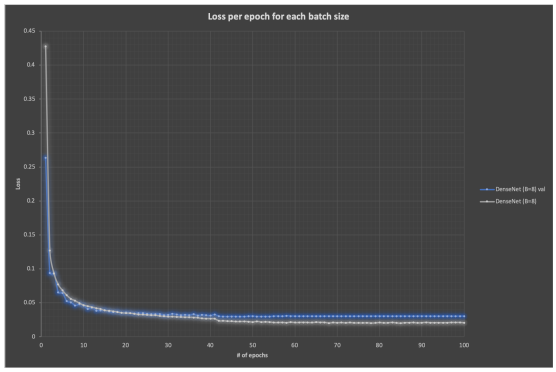

Fig. 7. DenseNet traininng and validation loss curves for batch sizes 8 (best accuracy)

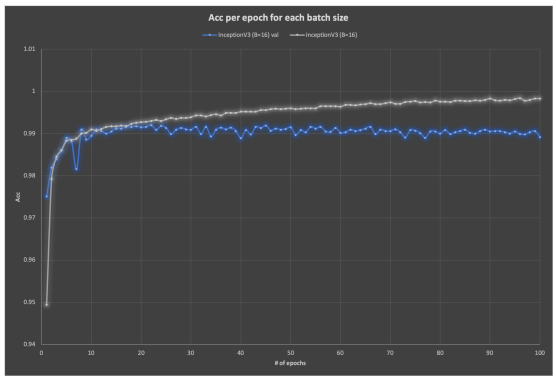

Fig. 8. Inception traininng and validation accuracy curves for batch sizes 16 (best accuracy)

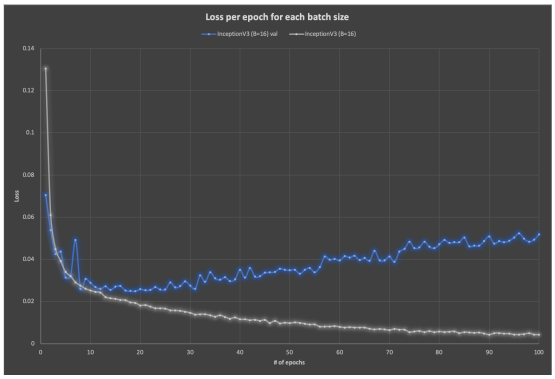

Fig. 9. Inception traininng and validation loss curves for batch sizes 16 (best accuracy)

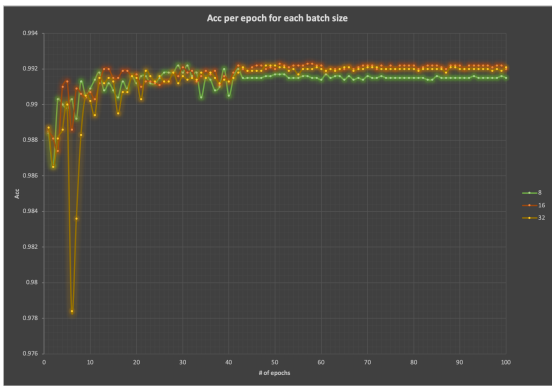

Fig. 10. ResNet traininng and validation acc curves for ResNet transfer learning to itself from pretrained of batch size 16 to batch sizes 8, 16, and 32

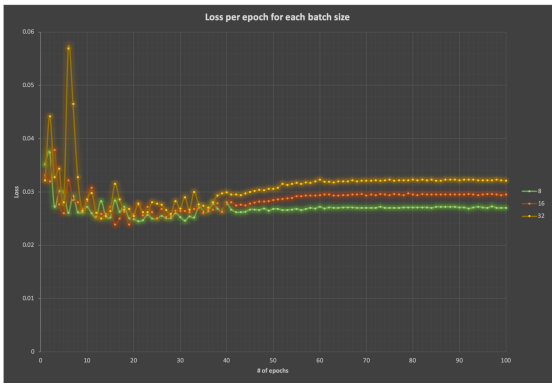

Fig. 11. ResNet traininng and validation loss curves for ResNet transfer learning to itself from pretrained of batch size 16 to batch sizes 8, 16, and 32

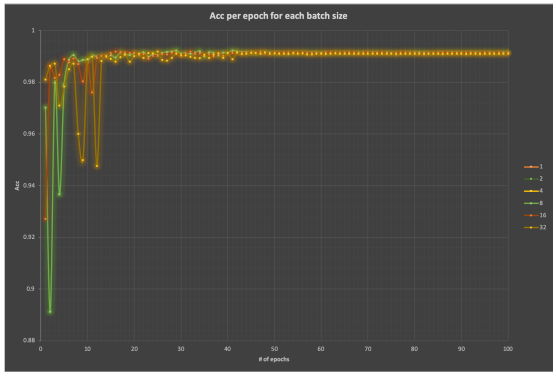

Fig. 12. ResNet traininng and validation acc curves for ResNet transfer learning from official pretrained of batch size 16 to batch sizes 8, 16, and 32

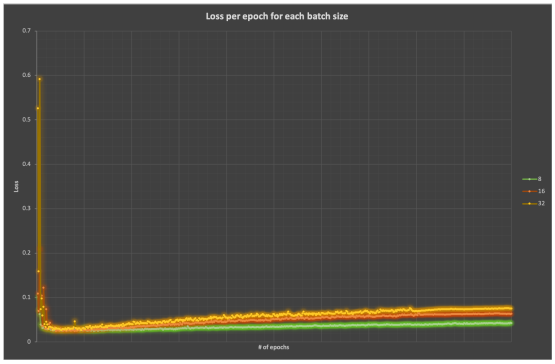

Fig. 15. ResNet traininng and validation loss curves for ResNet for 500 epochs to batch sizes 8, 16, and 32

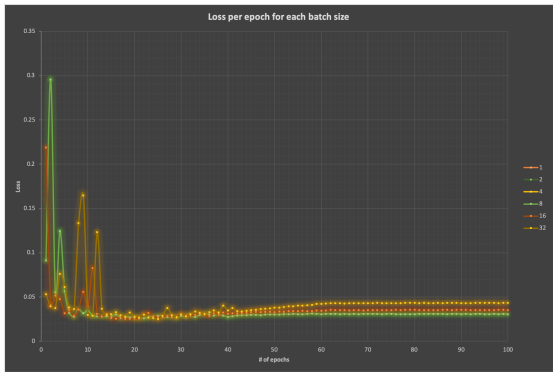

Fig. 13. ResNet traininng and validation loss curves for ResNet transfer learning to itself from official pretrained of batch size 16 to batch sizes 8, 16, and 32

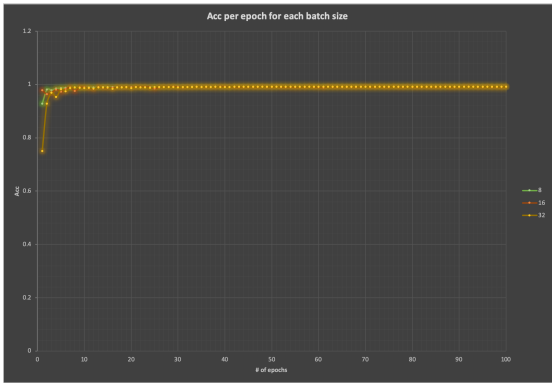

Fig. 16. DenseNet traininng and validation acc curves of batch sizes 8, 16, and 32

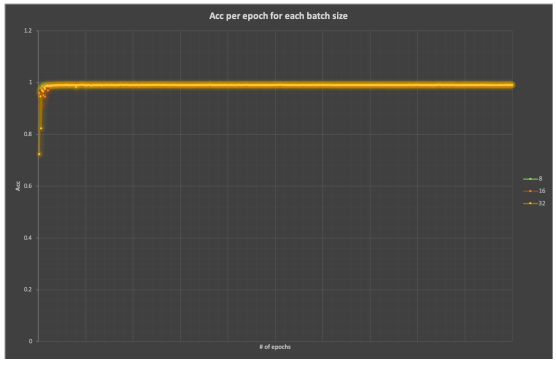

Fig. 14. ResNet traininng and validation acc curves for ResNet for 500 epochs to batch sizes 8, 16, and 32

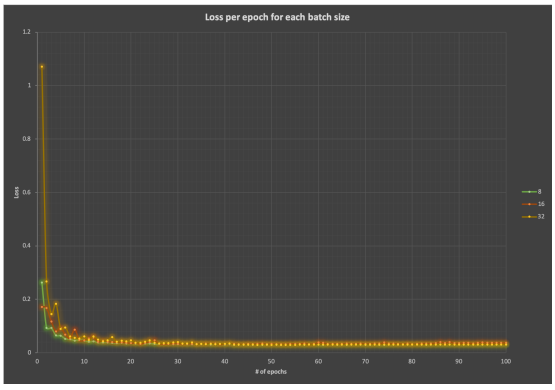

Fig. 17. DenseNet traininng and validation loss curves of batch sizes 8, 16, and 32

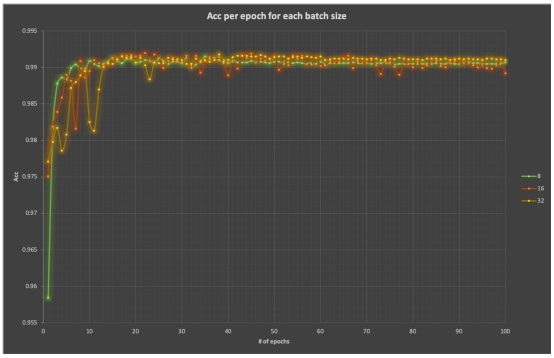

Fig. 18. Inception traininng and validation acc curves of batch sizes 8, 16, and 32

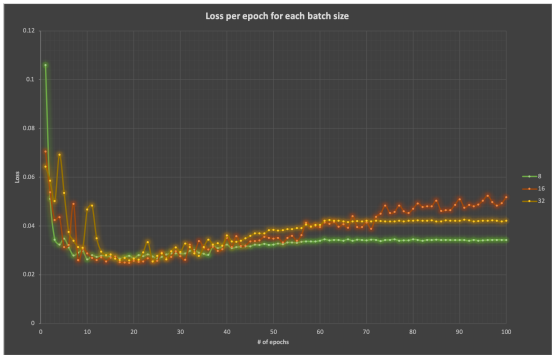

Fig. 19. Inception traininng and validation loss curves of batch sizes 8, 16, and 32

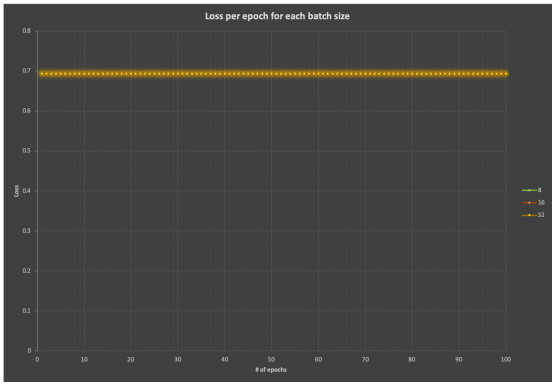

Fig. 21. SqueezNet traininng and validation loss curves of batch sizes 8, 16, and 32

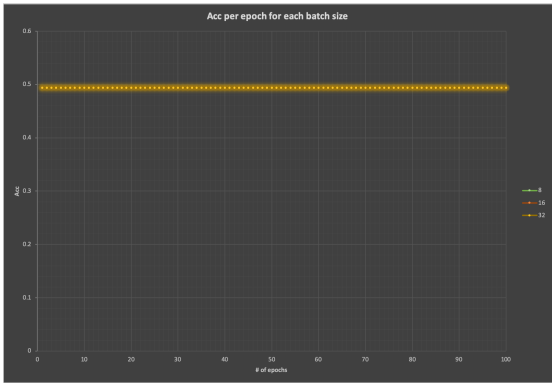

Fig. 20. SqueezNet traininng and validation acc curves of batch sizes 8, 16, and 32

TABLE II
DICTIONARY

# of blinks	Word	Vector
1	Yes	010
2	No	01010
3	Hi	0101010
4	I am	010101010
5	Good	01010101010
6	Thanks	0101010101010
7	How are you?	010101010101010

TABLE III
RESNET

Batch Size	No. of Epochs	Accuracy (%)	Last Improved Epoch
1	100	86.25	70
2	100	50.59	1
4	100	99.15	44
8	100	99.21	32
16	100	99.26	55
32	100	99.22	48

TABLE IV
RESNET (BEST 3 BATCH SIZES FOR 500 EPOCHS)

Batch Size	No. of Epochs	Accuracy (%)	Last Improved Epoch
8	500	99.20	60
16	500	99.21	33
32	500	99.19	46

TABLE V
RESNET (TRANSFER LEARNING FROM BATCH SIZE 16 FOR 100 EPOCHS)

Batch Size	No. of Epochs	Accuracy (%)	Last Improved Epoch
8	100	99.22	31
16	100	99.23	51
32	100	99.22	49

TABLE VI
RESNET (TRANSFER LEARNING FROM OFFICIAL RESNET50 FOR 100 EPOCHS)

Batch Size	No. of Epochs	Accuracy (%)	Last Improved Epoch
8	100	99.22	29
16	100	99.17	16
32	100	99.17	25

TABLE VII
DENSENET

Batch Size	No. of Epochs	Accuracy (%)	Last Improved Epoch
8	100	99.24	55
16	100	99.18	70
32	100	99.21	52

TABLE VIII
SQUEEZENET

Batch Size	No. of Epochs	Accuracy (%)	Last Improved Epoch
8	100	49.40	1
16	100	49.40	1
32	100	49.40	1

TABLE IX
INCEPTIONV3

Batch Size	No. of Epochs	Accuracy (%)	Last Improved Epoch
8	100	99.14	35
16	100	99.20	22
32	100	99.17	38