SUITABILITY OF DEEP LEARNING ARCHITECTURES FOR FACIAL RECOGNITION

Betzalel Moskowitz
MSDS458-DL: Artificial Intelligence and Deep Learning
July 23, 2023

Abstract

Facial recognition software provides important added layers of security and makes confirming identities faster and more efficient. The technology has improved due to advances in deep learning architectures and computing hardware that yields higher performance and faster training of facial recognition models. Through experimentation with different deep learning architectures and hyperparameter settings, this paper demonstrates the power of convolutional neural network (CNN) architectures over ordinary fully connected layers in complex image processing tasks. Through testing on a proxy-dataset (CIFAR-10), several CNN architectures are shown to yield a 60% improvement in test accuracy relative to fully connected deep neural network (DNN) architectures, and has inspired many of the dominant facial recognition model architectures in use today.

Introduction

The advent of facial recognition (FR) software has had profound impact in countless domains. Virtually all modern smartphones support the convenient capabilities of confirming identity to unlock the device, downloading apps, or making payments using virtual wallets. Airports use the technology to verify identities at security checkpoints and customs and border control, allowing lines to move faster while requiring less human effort and obtaining higher accuracy. In a pilot study (*Where is Facial Recognition Used?*) the technology has reduced boarding times between 30-40%.

FR software has been used extensively by casinos to assist security personnel in identifying banned individuals, single out high-value players to engage and build relationships with and identifying self-excluded players to ensure that those players are rehabilitating properly (Swanger, 2021).

The United States government has also benefitted immensely from FR software – the U.S. Department of State commands the largest FR system in the world – their database consists of data of over 113 million American adults whose driver's license photos serve as the data source (Roberts, 2021). The FBI makes use of this data to identify people in past images or security feeds that may provide them with additional information to assist in investigations (Ritchie, 2016). While efforts exist to implement regulations to quell concerns over surveillance and privacy, the business value and capabilities provided by FR software are undeniable, and FR software is here to stay.

While facial recognition systems have been around since the 1960s, many of these systems required significant human involvement to manually identify the coordinates of specific facial features. These were used to calculate 20 facial features, which would be compared to distances in each image to calculate the differences between the measurements and return the most likely match from a closed database. While often unreliable, this method was also highly inefficient – a human could only process 40 images per hour, so building databases of faces was time-consuming and expensive (Nilsson 2009). Takeo Kanade demonstrated a face-matching system in 1970 that located anatomical features without human intervention, but later tests showed that the system did not consistently identify facial features accurately (Leeuw & Bergstra, 2007).

Thus, early work on facial recognition got off to a slow start since humans were selecting features themselves. The process required expensive and inefficient manual labor or was automated in a fashion that yielded unreliable results. Datasets were thus restricted in either or both volume and quality.

As time went by, the landscape became ripe for a new technology that allowed FR software to take off – neural networks. With the internet providing access to a sufficient quantity of images to train on, improved hardware (GPUs and TPUs) for efficiently training neural networks, and advances in research on network architectures, it was finally possible to use neural networks to build FR systems.

Neural networks are advantageous in this setting in that the neural network is designed to learn features on its own, so there is no need to spend the time manually taking measurements of features to pass into the model. The idea is that by passing images of people and their labels (in this case their identities) into a neural network, you can train the model such that it will learn the important features to discriminate between the people in the dataset.

Most modern FR systems in use today were built using neural networks. FaceNet, a model developed by researchers at Google, used a convolutional neural network (CNN) to obtain an accuracy score of 99.63%, at the time a new record (Schroff et al., 2015).

While facial recognition tasks are certainly more nuanced, it is not entirely different from many other image recognition tasks where the task is to distinguish between input data belonging to one class versus another. In an FR task, one might need to distinguish between one face and all of the other faces. Similarly, in an image classification task, one might be tasked with distinguishing between an airplane and other objects. Due to the conceptual similarities between FR and image recognition, this paper provides insights into important neural network architectures and topologies that perform well for more complex image recognition tasks. The paper will investigate these concepts through training models of different architectures and different hyperparameters to achieve strong performance on image classification using the CIFAR-10 dataset (Krizhevsky, 2009).

Literature Review

Due to its large sample size, completeness, structured nature, and minimal preprocessing required, the CIFAR-10 dataset has been researched extensively throughout literature and serves as a common benchmark for image recognition models. At the time of writing, 231 teams have submitted their models to compete in Kaggle's "Object Recognition in Images" competition, with the best submission achieving 95.53% accuracy on the test set (Kaggle). Papers with Code, a website that tracks performance from different research papers on machine learning tasks, lists over 200 papers written on CIFAR-10's classification task, with at least twenty papers achieving test accuracies over 99%. The two models tied for the highest performance (ViT-H/14 and DINOv2) achieved test accuracies of 99.5% (Papers with Code).

ViT-H/14, (developed by Alexey Dosovitskiy, Neil Houlsby, and others at Google), leverages attention by applying transformers directly to sequences of image patches (2020). Using this method, high performance was achieved by pre-training a visual transformer (ViT) on large quantities of data and transferring to smaller or medium sized datasets like CIFAR-10 (Dosovitskiy et al., 2020).

Like ViT-H/14, DINOv2 (Oquab et al., 2023) proposes an automatic pipeline to build carefully curated and diverse datasets and trains a pretrained ViT on one-billion parameters before distilling it into smaller models.

Most of the modern state-of-the-art methods seem to be using transformers (with hundreds of millions of parameters) and transfer learning, fine-tuning on the CIFAR-10 dataset with pretrained models have already been pretrained on amounts of data several orders of magnitude larger than the size of CIFAR-10.

Many of the remaining non-transformer or fine-tuned models are variants of ResNet (He et al., 2016), a specialized CNN with an architecture which passes input data forward, skipping an arbitrary number of layers before merging it with the output of the last layer it skipped. This allows deep learning models with large quantities of layers to overcome the degradation problem and achieve greater accuracy as it gets deeper into its layers. The ResNet model, which was trained on augmented data, achieved an error rate of 6.43% (93.57% test accuracy) on CIFAR-10 after being trained on 110 layers (He et al., 2016).

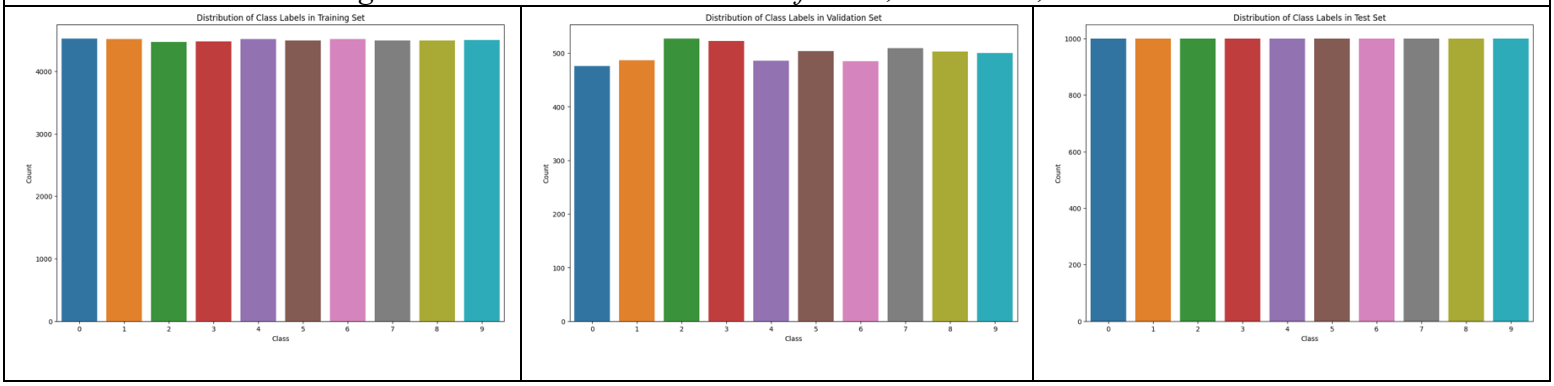
Methods

Data Exploration

The CIFAR-10 dataset, collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton is a labeled subset of the larger 80 million tiny images dataset (Krizhevsky, & Hinton, 2009). It provides 60,000 32 x 32 pixel color images in 10 classes, with 6,000 images of each class. The training set comprises 50,000 of those color images, while the other 10,000 belong to the test set. The test set consists of 10 samples of exactly 1,000 images representing each class. The images in the training set contain images in random order, resulting in training batches that may contain more images of one class than another. However, there are 5,000 images of each class between all five batches in the training set (Krizhevsky, & Hinton, 2009).

The 50,000 image training set was split into 45,000 images for training, and 5,000 images for validation using the same random sampling state to allow for each of the models to be trained and validated with the same data. It can be observed in Figure 1 that the distribution of class labels in each dataset is close to uniform, making for a balanced dataset.

Figure 1. Class Label Distribution of Train, Validation, and Test Sets



Data Preprocessing

For the models built without any convolutional layers, the inputs (images) were flattened by reshaping each image of size (32, 32, 3) 3D arrays to shape (3072,) vectors (1D arrays), allowing for each of the three RGB channels (red, green, and blue) in each of the 1024 pixels in the image to be fed into the input layer of the neural networks. Each of the pixel values in the image were represented by an integer value from 0 (no light) to 255 (maximum light), but these were scaled by dividing each value in each image by 255.

For the models built with convolutional layers, the inputs (images) of the split datasets were kept as (32, 32, 3) 3D arrays. Each of the pixel values in the image were represented by an integer value from 0 (no light) to 255 (maximum light), and these were also scaled by dividing each value in each image by 255.

The original labels for the MNIST dataset are integer digits from 0-9. Because SparseCategoricalCrossEntropy was used as the loss function, it was not necessary to one-hot encode the labels.

Implementation and Programming

The experiments were carried out via Google Colab notebooks, utilizing Google's T4 GPUs on a hosted Python runtime. All modeling was conducted using scikit-learn and Keras within Tensorflow version 2.12. All visualizations were created using Matplotlib or Seaborn. Additional Python libraries used included the Pandas and NumPy libraries for data manipulation.

In order to develop an intuition on how different architectures of deep neural networks impact performance on image processing tasks like CIFAR-10 (and FR by proxy), five general model topologies were developed:

- Experiments 1a-1m: DNN with two hidden layers (fully connected)
- Experiments 2a-2o: DNN with three hidden layers (fully connected)
- Experiments 3a, 3b: CNN with two convolution/max pooling layers
- Experiments 4a-4d: CNN with three convolution/max pooling layers

After seeing that the ResNet (He et al., 2016) architecture contained patterns of consecutive convolutional layers followed by a max pooling layer, I decided to experiment with an additional architecture:

- Experiments 5a-5b: CNN with three repetitions of two convolutional layers followed by a max pooling layer

Each model's hidden layers were trained using the ReLU activation function and a learning rate of 0.001. Each model contained an output layer with ten nodes and a SoftMax activation function. Sparse categorical cross-entropy was used as the loss function.

Each experiment was allowed to train on up to 200 epochs, but with an early stopping patience threshold of five. The early stopping occurred if the validation accuracy did not improve after five epochs, preserving the best model and avoiding additional overfitting.

In experiment 1, DNNs with two hidden layers were used. Experimentation was conducted to determine the effect that the number of hidden units had on training and performance. Experiments 1a-1f were conducted using 200, 500, 1000, 2000, 3000, and 2058 hidden units respectively in each of the two hidden layers (the 2058 was chosen to test out a rule-of-thumb that one should select the number of hidden units equal to 2/3 the input size (3072) plus the number of classes (10). The best model from 1a-1f was selected to then test out the effect of changing the batch size. Experiments 1g-1i tested out using batch sizes of 32, 64, and 256 (skipping 128, since it was already used for training the other models). In models 1j-1m, the best model from 1a-1i was used to test out the effectiveness of different regularization techniques: L2 regularization (with gamma=0.001), batch normalization, 10% dropout layers, and one where all three techniques were applied simultaneously.

In experiment 2, DNNs with three hidden layers were used. Experimentation was conducted again to determine the effect that the number of hidden units and batch size had on training and performance. The same process as in experiment 1a-1i was repeated in 2a-2i to examine this. In models 2j-2o, the best model from 2a-2i was used to test out the effectiveness of different regularization techniques: L2 regularization (with gamma=0.001), batch normalization, 10% dropout layers, 20% dropout layers, 30% dropout layers and one where L2 regularization, batch normalization, and the most performant dropout layers were applied simultaneously. Higher dropout percentages were considered in this experiment due to the presence of more hidden layers.

In experiment 3, CNNs with two convolutional layers and two max-pooling layers were trained. Experiment 3a and 3b both utilized 2-hidden layers in the fully connected layers with 256 hidden units, 3x3 kernel sizes, and convolutional layers consisting of 32 and 64 filters (ReLU activation functions), each followed by a 2x2 max pooling layer. 3a was conducted without any regularization techniques. 3b was conducted with L2 regularization, batch normalization, and 30% dropout layers.

Experiment 4 was similar to the experiment 3 but saw the addition of another convolutional layer (128 3x3 filters) followed by another 2x2 max pooling layer before the fully connected layers. There were 2 hidden layers in the fully connected layers with 256 hidden units. Experiment 4a contained no overfitting prevention techniques, but 4b, 4c, and 4d were conducted with L2 regularization (with gamma=0.001) and batch normalization. 4b, 4c, and 4d were also conducted with 30%, 20%, and 10% dropout layers respectively.

Experiment 5a and 5b experimented with an architecture that used a pattern of multiple convolutional layers of the same number of kernels (each with Batch normalization) followed by a max pooling layer and a dropout layer starting at 20%. This architecture was repeated twice more, each time increasing the number of filters by a factor of two and increasing the dropout layer percentage by 10%. The model was then flattened and fed into the fully connected layers with 2 hidden layers, each with 256 hidden units and followed by a 50% dropout layer. Experiments 5a and 5b only differ in the number of filters used in the convolutional layer – 5a has convolutional layers with filters (32, 32, 64, 64, 128, 128) and 5b has convolutional layers with filters (64, 64, 128, 128, 256, 256).

For each of the experiments run, performance metrics, and a confusion matrix were recorded to determine effects of different architectures and hyperparameters on training and performance.

Results

Architectures

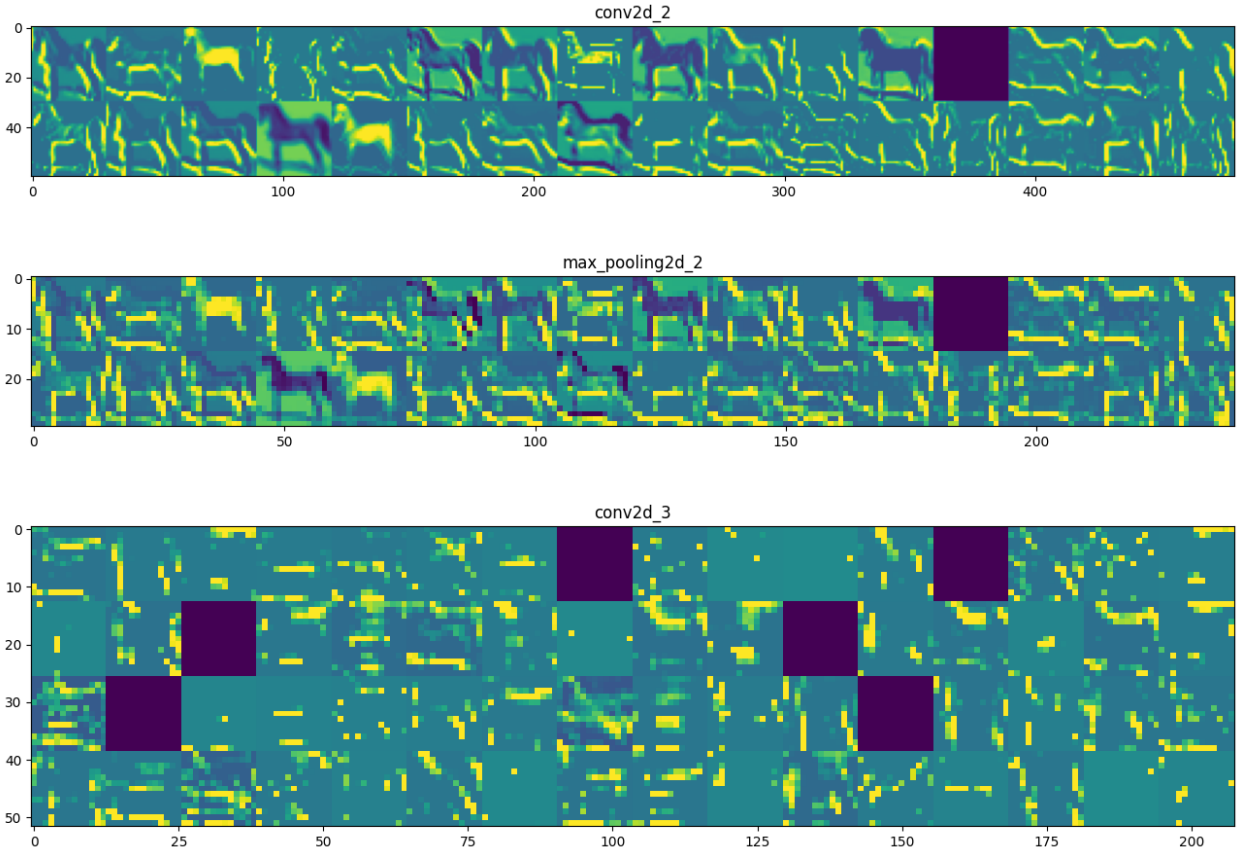
Figure A1 in the appendix displays the results of the experiments.

The different architectures of the models (fully connected DNNs vs CNNs) were responsible for producing the greatest variability in test accuracy.

It is immediately evident how poor the fully connected layers performed on complex image classification tasks. None of the models performed better than 53% test accuracy, and some models achieved as low as a 45% test accuracy. This was true for both the two-hidden layer architecture (experiment 1) and the three hidden layer architecture (experiment 2).

Experiments 3 and 4 showed exactly why CNNs are so widely used for image processing – these models achieved 70%-75% test accuracies, a significant improvement over the architectures in the previous two experiments. By looking at the outputs of the filters from the max pooling layers, it becomes evident why CNNs yield such an improvement over simple fully connected layers:

Figure 2: Grid of Images from the Filters in Experiment 3a



As can be seen, the CNNs are effective at exploiting the spatial structure of images. By looking at the image, one can see how it captures the local patterns and features within the image, enabling detection of edges, corners, and textures. The pooling layers, seen in the second row of grids, successfully down-sample the image while preserving the most important features. This helps to reduce the size, making the network more manageable and provides some degree of translation invariance. Additionally, having more layers help to capture features at different levels of abstraction. The deeper levels can detect simple features like edges and colors, while the deeper layers identify more complex and high-level features like shapes and objects. This hierarchical feature learning seems to emulate how humans perceive images.

The models in experiment 4 (trained with an extra convolutional layer and max pooling layer) seemed to perform slightly better than the ones trained in experiment 3, but only definitively when there were no regularization techniques used. When these techniques are used, both models' test accuracies improved by several percentage points.

Experiments 5a and 5b show the power of consecutive convolutional layers before max pooling layers. These models achieved test accuracies of 82.37% and 82.68% respectively. Model 5b had more filters in each layer than 5a and had the best accuracy of any model trained in the experiments. It seems as if the stacked convolutional layers allowed the model to learn a more

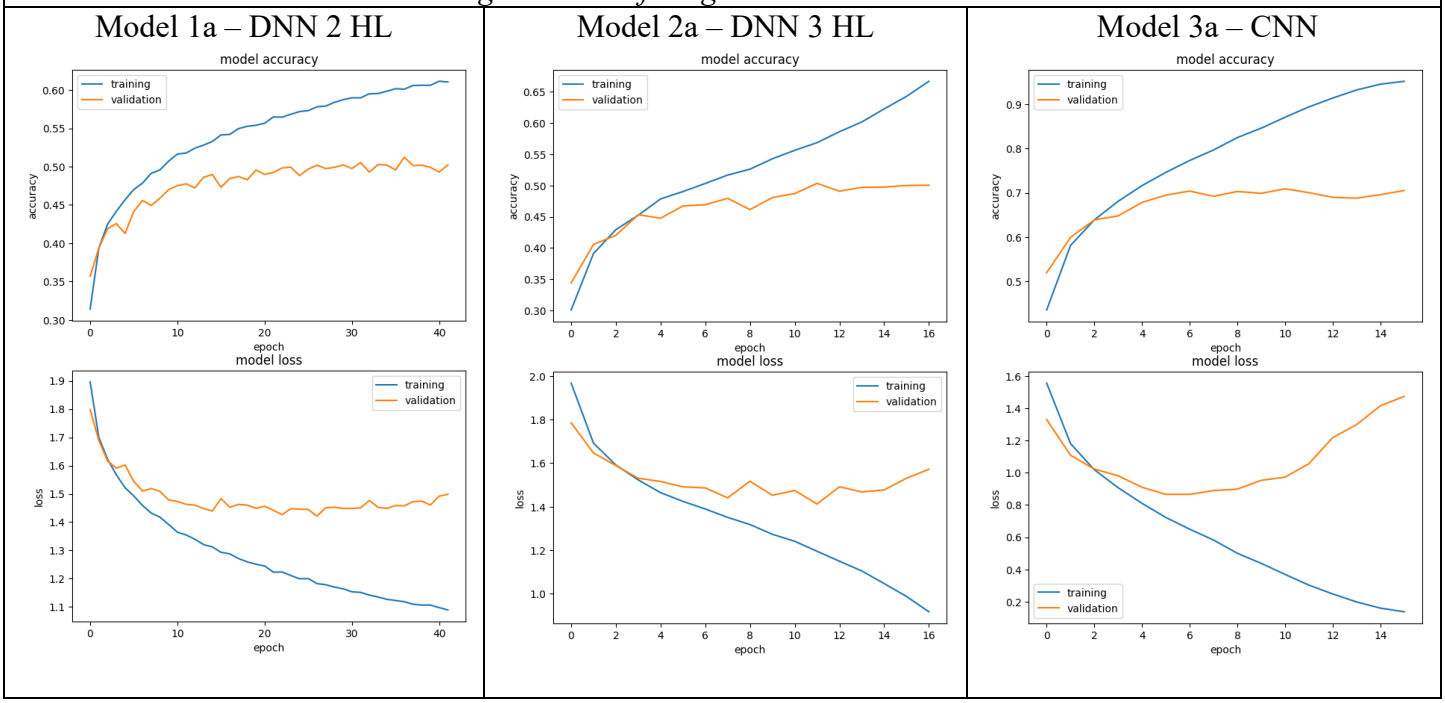
comprehensive number of features at different levels of abstraction throughout the model and allowed the model to capture more complex patterns.

Overfitting

Each of the models without any regularization techniques overfit heavily. Overfitting was prevalent among all architectures and would have been worse if early stopping with a patience of 5 wasn't used for all of the models. The overfitting can be visualized in Figure 3.

Figure 3 below shows that as the number of epochs increase across all model architectures during training, the gap between the training and validation loss and accuracy grows, indicating that the model will not translate as well in discriminating between classes in out of domain cases. This overfitting is especially evident in the CNN architecture.

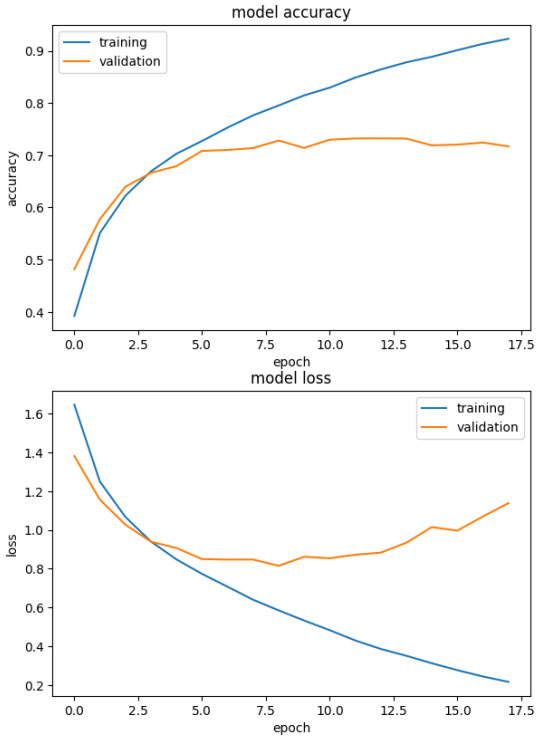
Figure 3. Overfitting Across Architectures



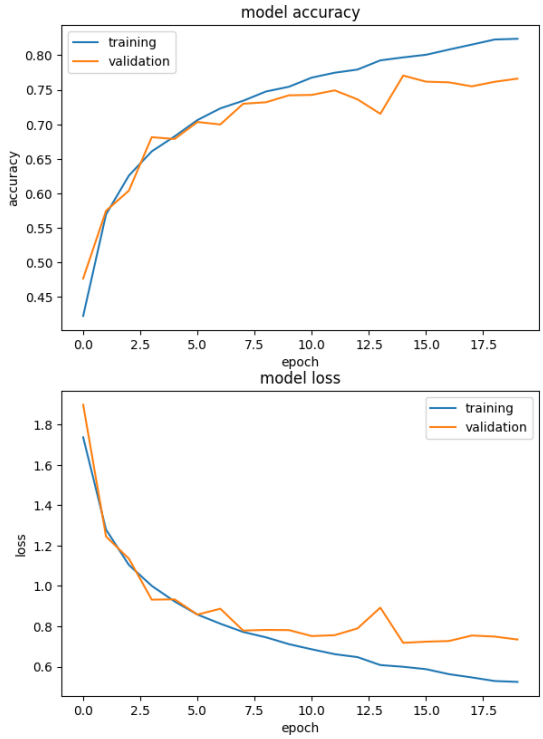
The regularization techniques seemed to make a big difference in reducing overfitting, especially among the CNN models. When tested individually, it appeared that L2 Normalization (gamma=0.001) was more effective than dropouts or batch normalization at reducing overfitting, but all experiments seemed to perform the best when all three regularization methods were used. While there was still significant overfitting occurring, the regularization techniques were able to achieve a few extra percentage points of test accuracy before early stopping kicked in. Figure 4 displays the difference that regularization makes when training a CNN with three convolutional layers.

Figure 4. Mitigating Overfitting Using Regularization

Model 4a – CNN with No Regularization Techniques



Model 4d – CNN with Batch Normalization, L2 Normalization, 10% Dropout



Hidden Nodes

In experimenting with different numbers of hidden nodes within the DNN, there did not appear to be a value that made a significant difference in performance. After conducting the experiments, it would have made more sense to change architectures first, instead of trying so many experiments with different numbers of hidden nodes, since most of the changes in performance were at most 1-2%. The small differences in performance in these scenarios are often due to random chance, which are not meaningful. This serves as yet another reminder that deep learning is data centric – there is often no one-size fits all method.

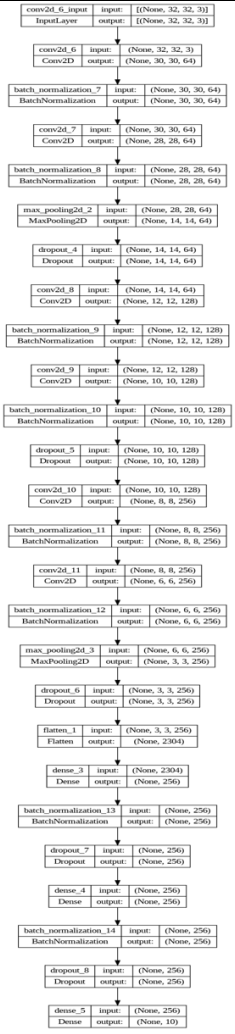
Batch Size

In both the two-hidden layer DNN and the three-hidden layer DNN, the performance was the worst when the batch size was 32, followed by 64, 256, and 128, which seemed to perform the best by several percentage points. It was also observed that each training epoch increased in speed as the batch size was raised.

Best Model

After conducting the experiments, the best model was model 5b. The model achieved an 82.68% test accuracy, 83% test precision, 83% test recall, and 83% test F1. Its architecture, training, confusion matrix, and tSNE plot could be seen below in Figures 5, 6, 7, and 8. The lack of clusters in Figure 8 suggests that the data cannot be easily clustered in lower dimensions.

Figure 5. Architecture of Best Model




```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 30, 30, 64)        1792

 batch_normalization_7 (Batc  (None, 30, 30, 64)       256
 hNormalization)

 conv2d_7 (Conv2D)           (None, 28, 28, 64)        36928

 batch_normalization_8 (Batc  (None, 28, 28, 64)       256
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 14, 14, 64)       0
 2D)

 dropout_4 (Dropout)         (None, 14, 14, 64)        0

 conv2d_8 (Conv2D)           (None, 12, 12, 128)       73856

 batch_normalization_9 (Batc  (None, 12, 12, 128)      512
 hNormalization)

 conv2d_9 (Conv2D)           (None, 10, 10, 128)       147584

 batch_normalization_10 (Bat  (None, 10, 10, 128)      512
 chNormalization)

 dropout_5 (Dropout)         (None, 10, 10, 128)       0

 conv2d_10 (Conv2D)          (None, 8, 8, 256)         295168

 batch_normalization_11 (Bat  (None, 8, 8, 256)        1024
 chNormalization)

 conv2d_11 (Conv2D)          (None, 6, 6, 256)         590080

 batch_normalization_12 (Bat  (None, 6, 6, 256)        1024
 chNormalization)

 max_pooling2d_3 (MaxPooling  (None, 3, 3, 256)        0
 2D)

 dropout_6 (Dropout)         (None, 3, 3, 256)         0

 flatten_1 (Flatten)         (None, 2304)              0

 dense_3 (Dense)             (None, 256)               590080

 batch_normalization_13 (Bat  (None, 256)              1024
 chNormalization)

 dropout_7 (Dropout)         (None, 256)               0

 dense_4 (Dense)             (None, 256)               65792

 batch_normalization_14 (Bat  (None, 256)              1024
 chNormalization)

 dropout_8 (Dropout)         (None, 256)               0

 dense_5 (Dense)             (None, 10)                2570

=================================================================
Total params: 1,809,482
Trainable params: 1,806,666
Non-trainable params: 2,816
_________________________________________________________________
```

Figure 6. Model Training

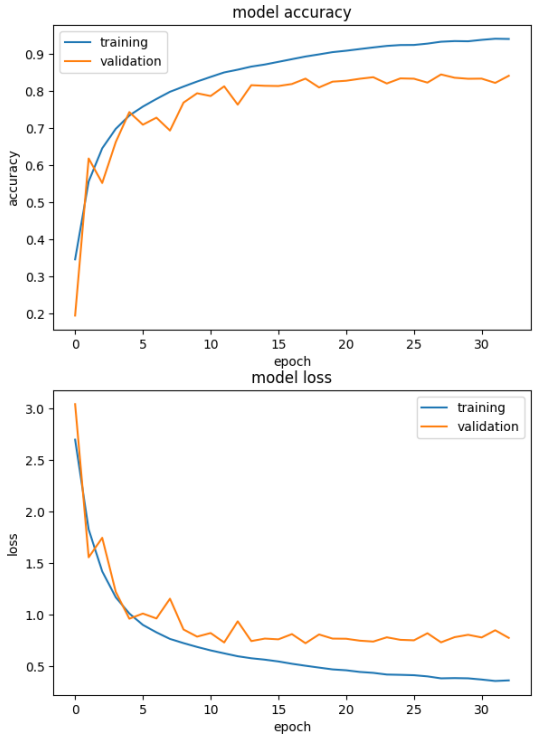


Figure 7. Model Confusion Matrix

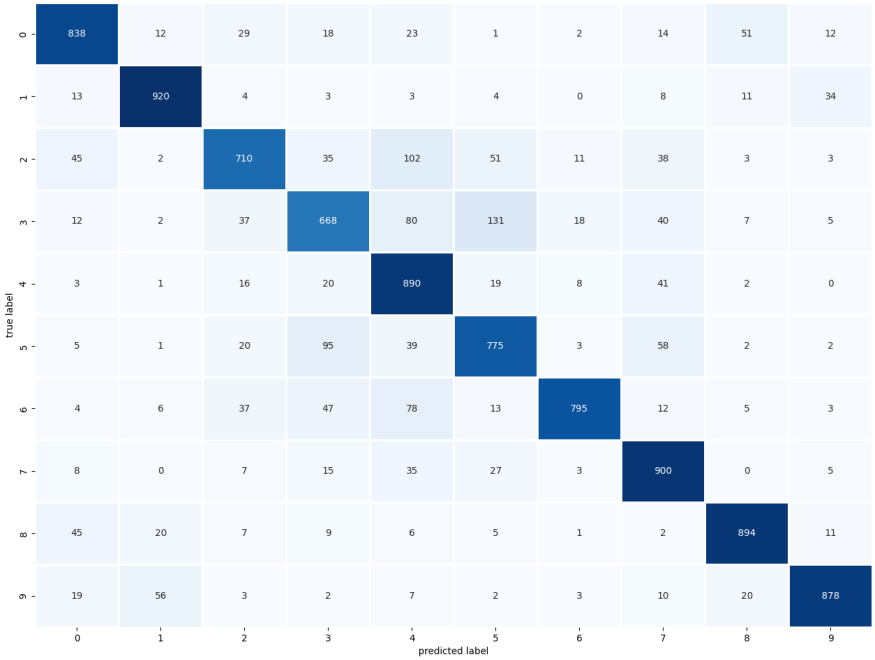
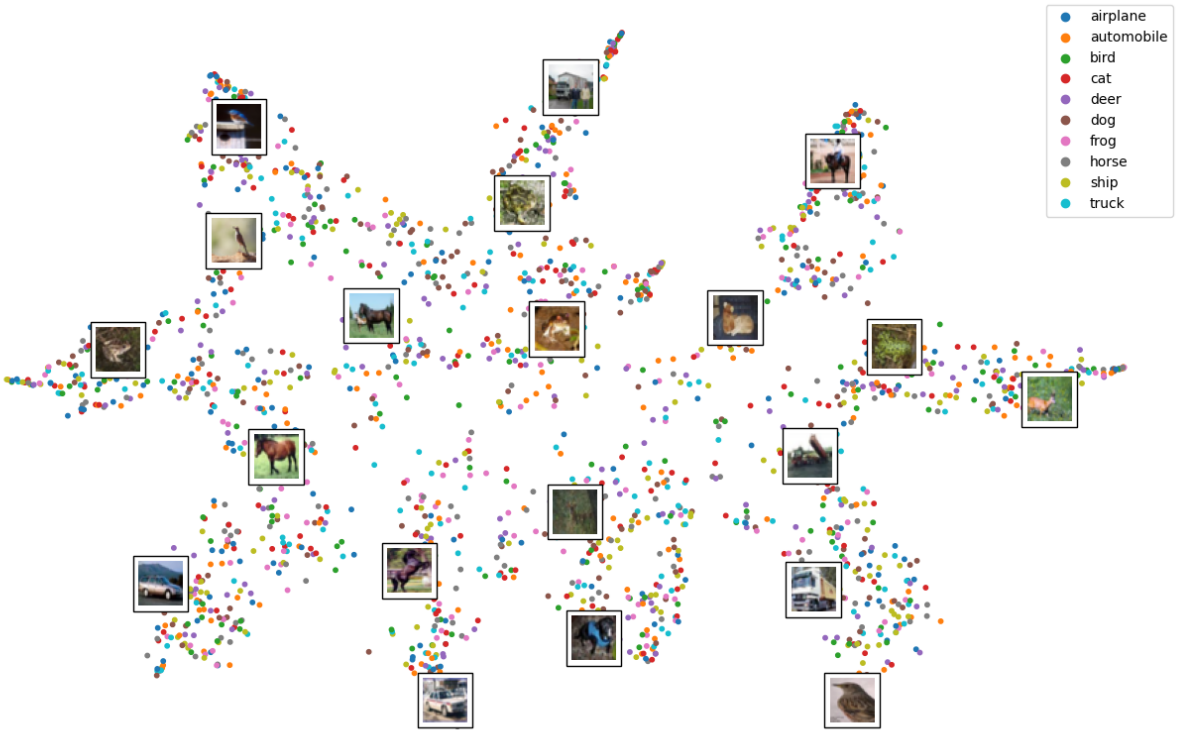


Figure 8. t-SNE Plot



Conclusion

The superior performance of CNN architectures on image processing tasks like CIFAR-10 suggest that CNNs are useful for solving image classification tasks like facial recognition. Adequate regularization using a combination of dropout layers, batch normalization and L2 regularization help to eliminate overfitting to allow the model to generalize to data outside of the training set. Far more performant than DNNs, CNNs have served as the foundation of some of the leading facial recognition models such as FaceNet. These types of software have been found to achieve near human-level performance. When applied to facial recognition tasks, the CNN architectures can be used to gain unprecedented accuracy capable of delivering convenience, extra security layers, and business value to companies, governments, and the ordinary layman who prefers to unlock their phone without using a passcode.

References

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

Kaggle. (n.d.). *Digit recognizer*. Kaggle. https://www.kaggle.com/competitions/cifar-10/leaderboard

Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images.

Leeuw, K. de, & Bergstra, J. A. (2007). *The History of Information Security: A Comprehensive Handbook*. Elsevier.

Nilsson, N. J. (2009). *The Quest for Artificial Intelligence*. Cambridge University Press.

Oquab, M., Darcet, T., Moutakanni, T., Vo, H., Szafraniec, M., Khalidov, V., ... & Bojanowski, P. (2023). Dinov2: Learning robust visual features without supervision. arXiv preprint arXiv:2304.07193.

Papers with Code. (n.d.). *Image Classification on CIFAR-10 | Papers With Code*. Papers with Code: The Latest in Machine Learning. https://paperswithcode.com/sota/image-classification-on-cifar-10

Ritchie, R. K. A. K. (2016, December 14). *The trouble with facial recognition technology (in the real world)*. Phys.org. https://phys.org/news/2016-12-facial-recognition-technology-real-world.html

Roberts, J. J. (2021, June 15). *Here's how many adult faces are scanned from facial recognition databases*. Fortune. https://fortune.com/2016/10/18/facial-recognition-database/

Schroff, F., Kalenichenko, D., & Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 815-823).

Swanger, C. (2021, November 19). *How integrated resorts and casinos are leveraging facial recognition software for increased security*. eConnect. https://www.econnectglobal.com/blog/how-integrated-resorts-and-casinos-are-leveraging-facial-recognition-software-for-increased-security

Where is Facial Recognition Used?. Thales Group. (n.d.). https://www.thalesgroup.com/en/markets/digital-identity-and-security/government/inspired/where-facial-recognition-used

Appendix

Figure A1. Experiment Results

Experiment	Description	Hidden Layers	Number of Nodes	Kernel Size	Filters	Overfitting Prevention Technique	Batch Size	Epochs	Training Time (seconds)	Train Time per Epoch	Train Accuracy	Validation Accuracy	Test Accuracy	Test Precision	Test Recall	Test F1
1a	2-Hidden Layer Fully Connected Layers	2	200	NA	NA	None	128	42	73.22	1.74	0.5987	0.5018	0.5178	0.52	0.52	0.51
1b	2-Hidden Layer Fully Connected Layers	2	500	NA	NA	None	128	23	45.83	1.99	0.5996	0.5132	0.5125	0.52	0.51	0.51
1c	2-Hidden Layer Fully Connected Layers	2	1000	NA	NA	None	128	26	55.34	2.13	0.6044	0.506	0.5159	0.52	0.52	0.51
1d	2-Hidden Layer Fully Connected Layers	2	2000	NA	NA	None	128	22	64.85	2.95	0.5923	0.5	0.5085	0.51	0.51	0.51
1e	2-Hidden Layer Fully Connected Layers	2	3000	NA	NA	None	128	32	125.58	3.92	0.5796	0.4936	0.5025	0.5	0.5	0.5
1f	2-Hidden Layer Fully Connected Layers	2	2058	NA	NA	None	128	24	65.08	2.71	0.6118	0.5018	0.5175	0.51	0.52	0.51
1g	2-Hidden Layer Fully Connected Layers	2	200	NA	NA	None	32	24	129.63	5.4	0.5368	0.4754	0.4835	0.48	0.48	0.48
1h	2-Hidden Layer Fully Connected Layers	2	200	NA	NA	None	64	17	56.37	3.32	0.5581	0.485	0.4993	0.5	0.5	0.5
1i	2-Hidden Layer Fully Connected Layers	2	200	NA	NA	None	256	19	27.67	1.46	0.5583	0.4926	0.5042	0.51	0.5	0.5
1j	2-Hidden Layer Fully Connected Layers	2	200	NA	NA	L2 Regularization (gamma = 0.001)	128	35	58.19	1.66	0.6267	0.5176	0.5194	0.52	0.52	0.52
1k	2-Hidden Layer Fully Connected Layers	2	200	NA	NA	Batch Normalization	128	20	43.3	2.17	0.5494	0.4818	0.4872	0.51	0.49	0.48
1l	2-Hidden Layer Fully Connected Layers	2	200	NA	NA	10% Dropout	128	35	65.55	1.87	0.5458	0.481	0.4929	0.49	0.49	0.49
1m	2-Hidden Layer Fully Connected Layers	2	200	NA	NA	L2 Regularization (gamma = 0.001), Batch Normalization, 10% Dropout	128	32	72.399	2.26	0.5746	0.5096	0.5123	0.51	0.51	0.51
2a	3-Hidden Layer Fully Connected Layers	3	200	NA	NA	None	128	25	54.98	2.2	0.6135	0.5004	0.515	0.51	0.52	0.51
2b	3-Hidden Layer Fully Connected Layers	3	500	NA	NA	None	128	24	49.58	2.07	0.6186	0.5028	0.5086	0.52	0.51	0.51
2c	3-Hidden Layer Fully Connected Layers	3	1000	NA	NA	None	128	14	36.37	2.6	0.5594	0.4994	0.5027	0.51	0.5	0.49
2d	3-Hidden Layer Fully Connected Layers	3	2000	NA	NA	None	128	17	55.64	3.27	0.6135	0.5034	0.5159	0.52	0.52	0.51
2e	3-Hidden Layer Fully Connected Layers	3	3000	NA	NA	None	128	26	139.07	5.35	0.5914	0.4916	0.5036	0.5	0.5	0.5
2f	3-Hidden Layer Fully Connected Layers	3	2058	NA	NA	None	128	28	88.99	3.18	0.51978	0.487	0.5103	0.51	0.51	0.51
2g	3-Hidden Layer Fully Connected Layers	3	2000	NA	NA	None	32	23	197.66	8.59	0.5247	0.4566	0.4699	0.48	0.47	0.47
2h	3-Hidden Layer Fully Connected Layers	3	2000	NA	NA	None	64	17	86.25	5.07	0.5475	0.476	0.4697	0.48	0.47	0.47
2i	3-Hidden Layer Fully Connected Layers	3	2000	NA	NA	None	256	24	62.74	2.61	0.5856	0.5002	0.5083	0.5	0.51	0.5
2j	3-Hidden Layer Fully Connected Layers	3	2000	NA	NA	L2 Regularization (gamma = 0.001)	128	28	87.89	3.14	0.5796	0.5016	0.5156	0.52	0.52	0.51
2k	3-Hidden Layer Fully Connected Layers	3	2000	NA	NA	Batch Normalization	128	22	78.09	3.55	0.5859	0.5088	0.5058	0.51	0.51	0.5
2l	3-Hidden Layer Fully Connected Layers	3	2000	NA	NA	10% Dropout	128	24	84.88	3.54	0.5973	0.49	0.4974	0.5	0.5	0.5
2m	3-Hidden Layer Fully Connected Layers	3	2000	NA	NA	20% Dropout	128	34	109.17	3.21	0.561	0.4874	0.4925	0.5	0.49	0.49
2n	3-Hidden Layer Fully Connected Layers	3	2000	NA	NA	30% Dropout	128	32	100.56	3.14	0.4889	0.4422	0.4534	0.46	0.45	0.45
2o	3-Hidden Layer Fully Connected Layers	3	2000	NA	NA	L2 Regularization (gamma = 0.001), Batch Normalization, 10% Dropout	128	35	232.58	6.65	0.6294	0.5176	0.5253	0.54	0.53	0.52
3a	CNN with 2 Convolution/Max Pooling Layers	2	256	3x3	32, 64	None	128	16	38.14	2.38	0.8143	0.704	0.7039	0.71	0.7	0.7
3b	CNN with 2 Convolution/Max Pooling Layers	2	256	3x3	32, 64	L2 Regularization (gamma = 0.001), Batch Normalization, 30% Dropout	128	41	120.96	2.95	0.9188	0.7628	0.7532	0.75	0.75	0.75
4a	CNN with 3 Convolution/Max Pooling Layers	2	256	3x3	32, 64, 128	None	128	18	56.67	3.15	0.8244	0.728	0.7218	0.73	0.72	0.72
4b	CNN with 3 Convolution/Max Pooling Layers	2	256	3x3	32, 64, 128	L2 Regularization (gamma = 0.001), Batch Normalization, 30% Dropout	128	31	108.99	3.52	0.8056	0.7502	0.7448	0.75	0.74	0.74
4c	CNN with 3 Convolution/Max Pooling Layers	2	256	3x3	32, 64, 128	L2 Regularization (gamma = 0.001), Batch Normalization, 20% Dropout	128	22	89.575	4.07	0.8234	0.7586	0.7488	0.75	0.75	0.75
4d	CNN with 3 Convolution/Max Pooling Layers	2	256	3x3	32, 64, 128	L2 Regularization (gamma = 0.001), Batch Normalization, 10% Dropout	128	20	74.06	3.7	0.8697	0.7706	0.7552	0.76	0.76	0.76
5a	CNN with 6 Convolutional Layers and 3 Max Pooling Layers	2	256	3x3	32, 32, 64, 64, 128, 128	L2 Regularization (gamma = 0.001), Batch Normalization, Increasing Dropouts	128	35	264.47	7.56	0.9284	0.828	0.8237	0.82	0.82	0.82
5b	CNN with 6 Convolutional Layers and 3 Max Pooling Layers	2	256	3x3	64, 64, 128, 128, 256, 256	L2 Regularization (gamma = 0.001), Batch Normalization, Increasing Dropouts	128	33	411.77	12.48	0.9416	0.8336	0.8268	0.83	0.83	0.83