Kaggle Competition: Comparison of machine learning methods for hand drawn image classification

Team: Gradient Ascenders
Eisha Ahmed, eisha.ahmed@mail.mcgill.ca, 260583530
Mohd Safwan Ahmad Ansari, mohd.ansari@mail.mcgill.ca, 260880458
Ying Zhang, ying.zhang8@mail.mcgill.ca, 260815803

I. INTRODUCTION

Image classification is a common supervised machine learning problem, where the goal is to predict the class to which an image belongs. Here, we use a variant of Google's Quick Draw dataset consisting of human drawn images belonging to 31 different classes with the addition of noise (including an *empty* class consisting of only noise). We optimize and compare several machine learning approaches to classifying images, specifically linear support vector machines (SVMs), logistic regression, feed-forward neural networks, k-nearest neighbours, and convolutional neural networks. The best validation accuracy was obtained using convolutional neural networks, which were in turn used to predict the classes of images belonging to the test dataset for the Kaggle competition. The tuned convolutional neural network performed the best across all the models, having a validation accuracy of 80.5%. We demonstrate through image pre-processing and data augmentation, even smaller convolutional neural networks are capable of obtaining high classification accuracy across many classes.

II. FEATURE DESIGN

The original training dataset contains 10k 100×100 grey-scale images, consisting of human drawn images across 31 classes. To facilitate classification and feature extraction, we process the images to remove noise and crop the image to remove blank space (thus reducing computational complexity).

a) Image pre-processing: A key property that distinguishes noise from the drawn images is their relative size. To remove noise and crop the image, we first generated a binary array from each 100×100 images using a pixel threshold value of 127, from which a set of contours for each image was generated. The area of the bounding rectangle for each contour was computed, and all the pixels outside the largest bounding rectangle set to 0. Each image was then cropped such that the contents of the bounding rectangle were centered in the middle of a 45×45 pixel image. Finally, a new binary mask generated using Otsu's method, and the connected components (connectivity 8) of the resultant mask computed. Pixels in the image corresponding to connected components with a bounding rectangle of area below 40 were set to zero (see fig. 1).

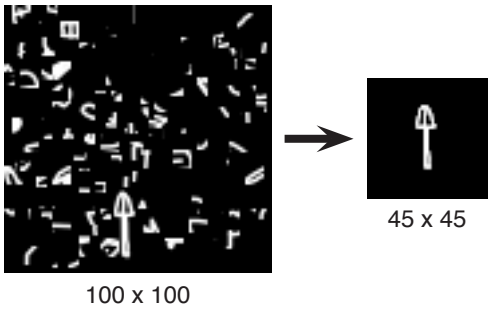

Fig. 1. Sample image from training data (shovel) before and after processing.

III. ALGORITHMS

A. Baseline Models

To establish a baseline for image classification, we test Support Vector Machines (SVMs), Logistic Regression, and k-Nearest Neighbours (kNN) to evaluate their capacity to learn and classify images from the pre-processed dataset. Algorithms are implemented using `sklearn`, and we analyze the classification accuracy across different hyper-parameters for each of these three algorithms.

B. Feed forward neural network

The Feed Forward Neural Network (FFNN) was applied as the requirement of this project. It is also called multiple layer perceptron. The model consists of several fully connected hidden layers with 31 neurons forming the final output layer. A self-implemented FFNN with one-hidden layer is provided in the attached code. To better analyze the performance of the FFNN with multiple hidden layers and different activation functions, we further implemented using the TensorFlow library. The results presented in this paper are from networks generated by this later script.

C. Convolutional neural network

Finally, we apply a machine learning model most commonly associated with image processing: convectional neural networks (CNNs). These are a class of feed-forward neural with a unique architecture, consisting of convolutional layers,

TABLE I

HYPER-PARAMETERS TESTED FOR LINEAR MODELS.

Model	Hyper-parameters	Values (**Best value**)
SVM	Loss	HINGE, **Square Hinge**
	C	**0.001**, 0.01, 0.5, 1.0, 2.0
Logistic regression	C	**0.001**,0.01, 0.1, 1, 10, 100
kNN	Neighbours	3, **5**, 7, 9

pooling layers, and fully connected layers. Unlike standard FFNNs, CNN architecture captures information on the spatial relationship of input values (such as neighbouring pixels in an image), and can learn the best features for image classification, thus does not rely on prior image feature extraction. CNN architecture can have a significant impact on classification performance, including the number of convolutional layers, the number of filters per layer, filter kernel size, and the size and depth of dense layers. Further hyper-parameters like dropout rate in the hidden layers, the learning algorithm (e.g. Adams or stochastic gradient descent optimization), and learning rate can also be tuned to optimize performance. Here, CNNs are implemented using the TensorFlow Keras API.

IV. METHODOLOGY

The pre-processed dataset was shuffled, and 20% (2000) of the images were separated for validation used to tune the models (with the remaining 8000 images for training). The algorithm with the highest validation accuracy was then used to classify the testing dataset for submission to the Kaggle competition.

A. Baseline Models

Table I presents the list of hyper-parameters we considered for each of the baseline models. We used a grid of parameters with a discrete number of values for each, testing all combinations of hyper-parameters for those resulting in the greatest validation accuracy.

B. Feed forward neural network tuning

The hyper-parameters explored for FFNN are listed in Table II. The grid-search method was used to find the best hyper-parameters to tune the model, and cross-validation used to evaluate model performance. Table II also presents the list of hyper-parameters we considered in FFNN. All of these values were tried to find the best hyper-parameters. Batch normalization method was also applied in the end to see if it will improve the model.

C. CNN architecture and tuning

All explored CNNs take a 45×45 image as input, contain a single dense hidden layer, and have an output layer of 31 nodes (one for each image class). Each convolutional layer is followed by Relu activation, followed by a max pooling layer. Categorical cross-entropy is used as an error function during training.

Training data is augmented using translations (horizontal and vertical shifts, range of 0.1), horizontal flips, and rotations (15 degrees range). Data augmentation was implemented using `ImageDataGenerator` function from the Tensorflow Keras API. The ranges of hyper-parameters and architectures explored are summarized in Table III.

TABLE II

HYPER-PARAMETERS TESTED FOR FEED-FORWARD NEURAL NETWORK OPTIMIZATION.

Hyper-parameters	Range	Best Values
Number of hidden layers	1, 2, 3	2
Number of neurons	32, 64, 128, 256, 512, 1024	1024
Activation function	RELU, SIGMOID, tanh	RELU
Loss function	Adam, SGD	Adam
Learning rate	0.001, 0.002, 0.005, 0.01	0.001
Dropout rate	0.2, 0.4, 0.6, 0.8	0.8
L2-regularization	0.001, 0.002, 0.005, 0.01	0.005
Batch size	32, 64, 128, 256	256
Number of Epochs	20, 40	40

TABLE III

ARCHITECTURE AND HYPER-PARAMETERS TESTED IN CNN DESIGN.

Hyper-parameters	Tested values
Number of convolutional layers	1, 2, 3, 4
Number of filters per convolutional layer	8, 16, 32, 64, 128
Number of neurons in dense layer	64, 128, 256, 512, 1024
Dropout rate	0, 0.1, 0.2, 0.3, 0.4

a) Convolutional layer depth and filter count: We begin optimizing the architecture of the CNN by evaluating the effect of the number of layers on validation accuracy. Hidden-layer dropout rate (0.2), learning rate and algorithm (Adams optimization, learning rate 0.001), and training batch size (32 samples) are held constant across models.

b) Hidden layer size: After selecting the optimal convolutional layer configuration, the effect of number of nodes in the hidden layer (64, 128, 256, or 512 nodes) on validation accuracy was evaluated. Training hyper-parameters were fixed across models (as described above).

c) Dropout rate: Fixing the optimal network architecture, we next evaluate the effect of dropout rate in the hidden layer on validation accuracy, with dropout rates ranging from 0 to 0.4.

For simplicity, we use a shorthand notation to denote the CNN architecture. For example, [16C5-P2]-[32C-P2]-256 denotes a CNN with two convolutional layers (16 5×5 filters and 32 5×5 filters, where each layer is followed by 2×2 max-pooling) and one hidden dense layer with 256 nodes.

V. RESULTS

A. Baseline methods

The hyper-parameters chosen for each linear model is listed in Table I. The results for each parameter combination are described in Figure2. The training accuracy reached up to 100% for all the three models, and accuracy for validation set reach 6.10% for logistic regression and remains less than 5% for SVM and kNN. This clearly indicates over-fitting. Looking at the classification accuracy on the validation dataset, we can say that the linear models perform poorly for our dataset. It can be observed that there is no significance

TABLE IV

VALIDATION CLASSIFICATION ACCURACY ACROSS TUNED MODELS.

Model	Val. Accuracy (%)
SVM	5.10
Logistic Regression	6.10
kNN	4.20
Feed-forward NN	54.5
CNN	80.5

difference between the three models in their performance over both training and validation sets.

B. Feed forward neural network

The hyper-parameters for the best FFNN model are listed in Table II. The accuracy for training sets reach to 67.67%, and accuracy for validation set reach to 54.40%. Graphs for loss-epoch and accuracy-epoch are shown in Figure 3. Batch normalization method was also applied at the end, but did not result in improved accuracy. Looking at the classification accuracy for each class on the validation dataset, we observe particularly poor results for SQUIGGLE class, which was 0%. Since squiggles are largely random drawings, there is more variability between drawings and can thus make the class more difficult to classify. This case can be specially addressed in the future work.

C. CNN

The CNN architecture resulting in the optimal validation accuracy consisted of 3 convolutional layers (model 4, fig. 5c) and 128 dense layers (fig. 6, top), with a final network structure of [64C5-P2]-[128C5-P2]-[256C5-P2]-128 (see figure 6, bottom, for diagram of selected architecture). Next, we tuned the dropout rate of the hidden layer, obtaining a maximum validation accuracy of 79.4% with a dropout rate of 0.2 (fig. 6, bottom). The addition of batch-normalization following convolutional layers did not significantly impact validation accuracy in this model.

Finally, we loaded the pre-trained optimal CNN model and continued training for another 80 epochs using Adam's optimization, with a reduced learning rate of 0.0001. This resulted in a maximum validation accuracy of 80.45%. As demonstrated by the confusion matrix (fig 4), this CNN is capable of classifying images reasonably well across all classes.

D. Performance comparison across algorithms

Overall, the CNN (80.45% validation accuracy) performed significantly better than the baseline classifiers (SVM, logistic regression, and kNN - validation accuracy below 10%). The FFNN significantly better than the baseline classifiers (54.5% accuracy), however was unable to achieve the CNN classification accuracy due to the loss of spatial information in the model architecture (see table IV).

VI. DISCUSSION

CNN provided the best result in this project compared to FFNN, SVM and baseline functions. Linear models over fit performed poorly mainly due to their inability to largely

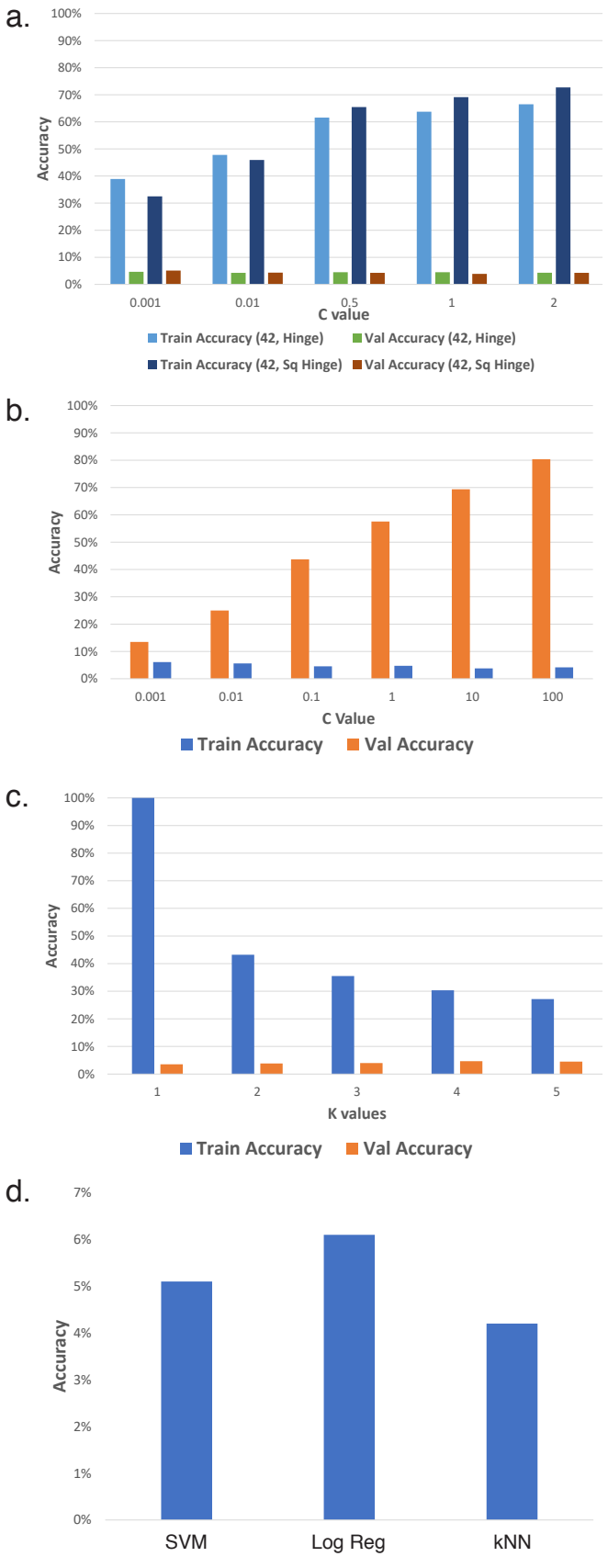

Fig. 2. Linear Models training and validation accuracy over different parameters and comparison. (a) Accuracy from SVM hyper-parameter tuning. (b) Accuracy from logistic regression hyper-parameter tuning. (c) Accuracy of kNN classifications for different values of k. (d) Comparison of validation accuracy across baseline methods.

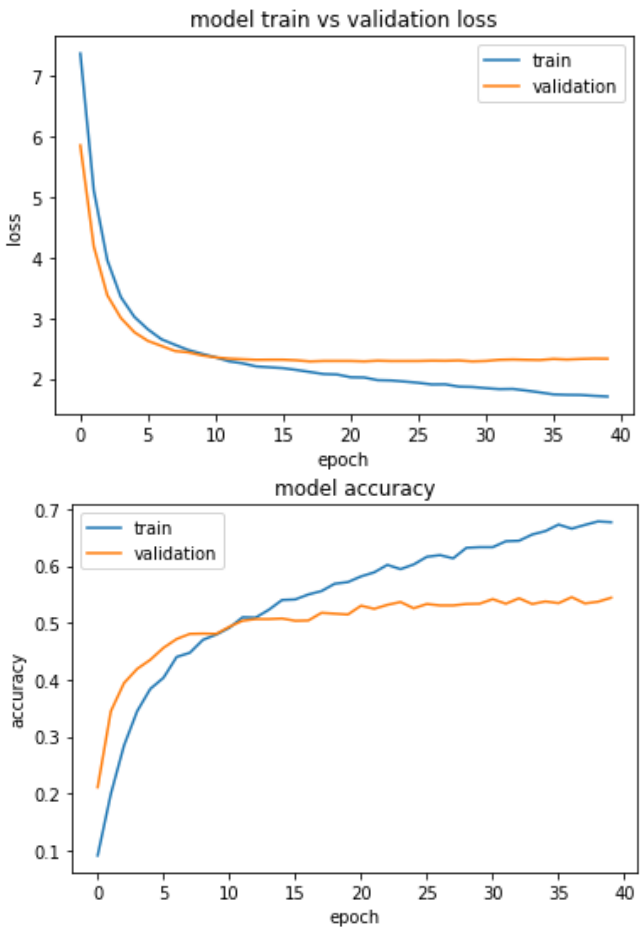

Fig. 3. FFNN training and validation loss and accuracy over 40 epochs.

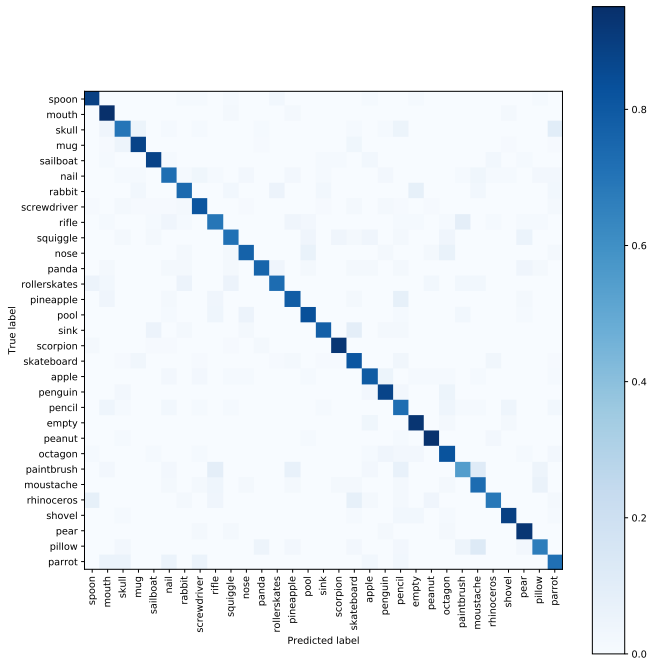

Fig. 4. Normalized confusion matrix for optimized CNN classification on validation dataset.

due to noise in the drawings and thus the variability between them. This also explains the poor performance over validation set compared to training set The pre-processing used simplified the classification problem without resulting in the loss of information, which in turn greatly improved validation accuracy. The dataset, which consisted of 10,000 images, was pre-processed to remove noise and reduce the input dimension from 100×100 to 45×45 (reducing the number of input pixels by 80%).

The use of data-augmentation in our model during training also greatly improved validation accuracy by reducing overfitting and increasing the amount of training data. When generating augmented training data, we considered the real situation for drawing a sketch. We only consider translations, small rotations, and horizontal flipping for our model, as these best capture possible variants of possible human drawings across all the classes. Augmentation such as vertical flipping is not used because it can be poorly representative of certain classes (e.g. a skull is unlikely to be drawn upside down in this dataset), thus such transformations would make the problem excessively difficult. Deeply understanding the problem is crucial when applying machine learning to solve a particular problem.

There are multiple avenues for future work and improvement. Alternate image pre-processing and feature extraction methods can be explored to improve classification accuracy, to which the linear methods are particularly sensitive. Different learning algorithms, including decision trees and ensemble models, can also be explored. With more computational resources, a greater range of FFNN and CNN architectures and hyper-parameters can be explored. Alternative training methods, such as training CNNs using auto-encoders may also prove effective. Beyond CNNs, capsule neural networks could also be explored to improve image classification. This variant of CNNs can better model the relationship between features in the image, without the information loss resulting from max-pooling.

VII. STATEMENT OF CONTRIBUTIONS

Eisha performed dataset pre-processing, implemented and tuned the convolutional neural network architecture, and edited the report. Safwan implemented and tuned baseline classifiers (SVM, logistic regression, and kNN). Ying implemented and tuned the feed-forward neural network (MLP), and provided inputs into CNN design and dataset pre-processing. Each author wrote sections of the report and generated figures corresponding to their respective implemented methods. We hereby state that all the work presented in this report is that of the authors.

APPENDIX

We present additional figures summarizing results from CNN architecture optimization and hyper-parameter tuning (fig. 5, 6).

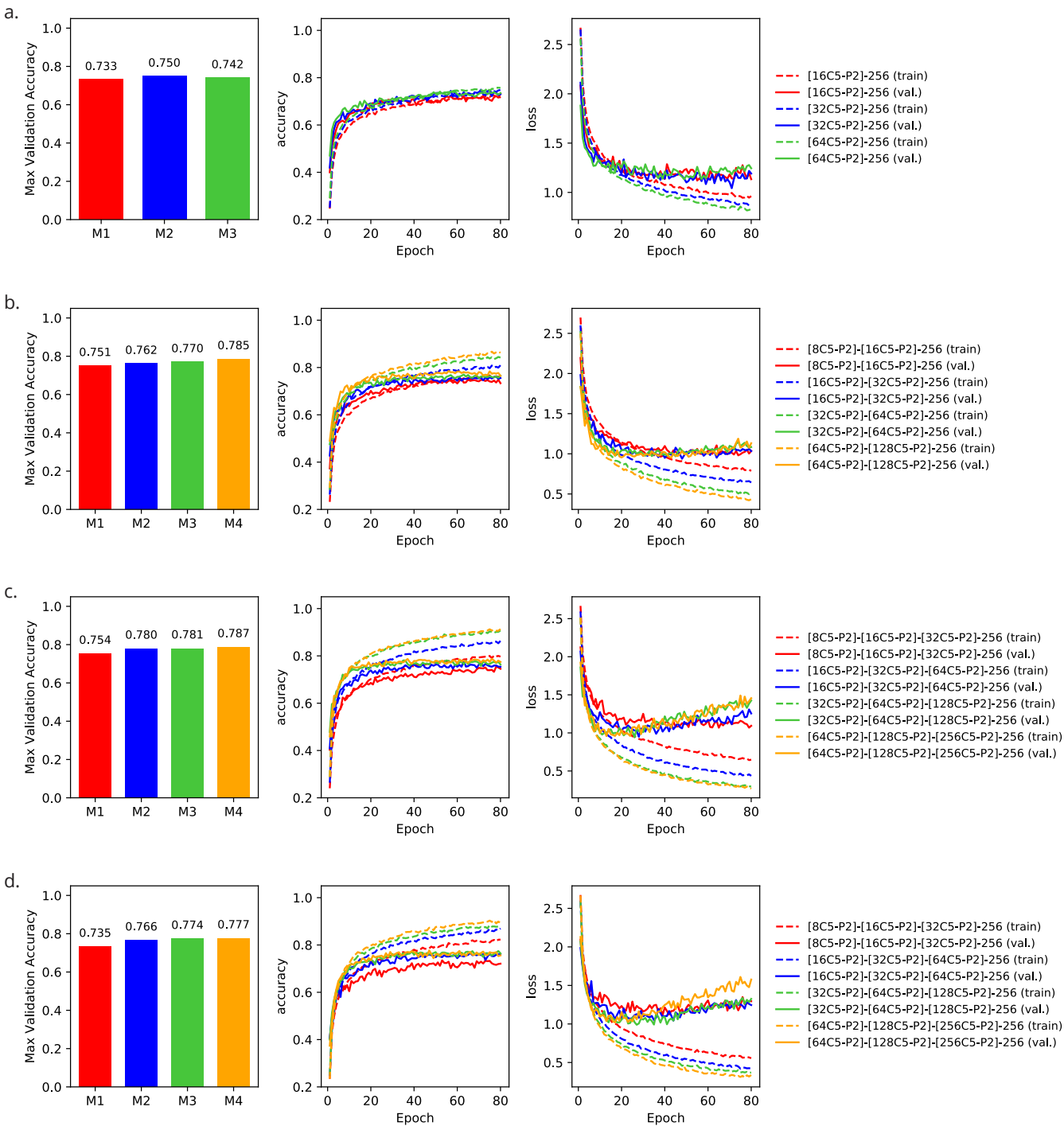

Fig. 5. Training and validation results from CNNs with varying number of convolutional layers and filters over 80 epochs (batch size 32, Adams Optimization with learning rate of 0.001, hidden layer dropout rate 0.2). (a) One convolutional layer, (b) two convolutional layers, (c) three convolutional layers, and (d) four convolutional layers. Bar charts (left) report the maximum validation accuracy obtained for each model across epochs, and plots summarize training and validation accuracy and loss across epochs.

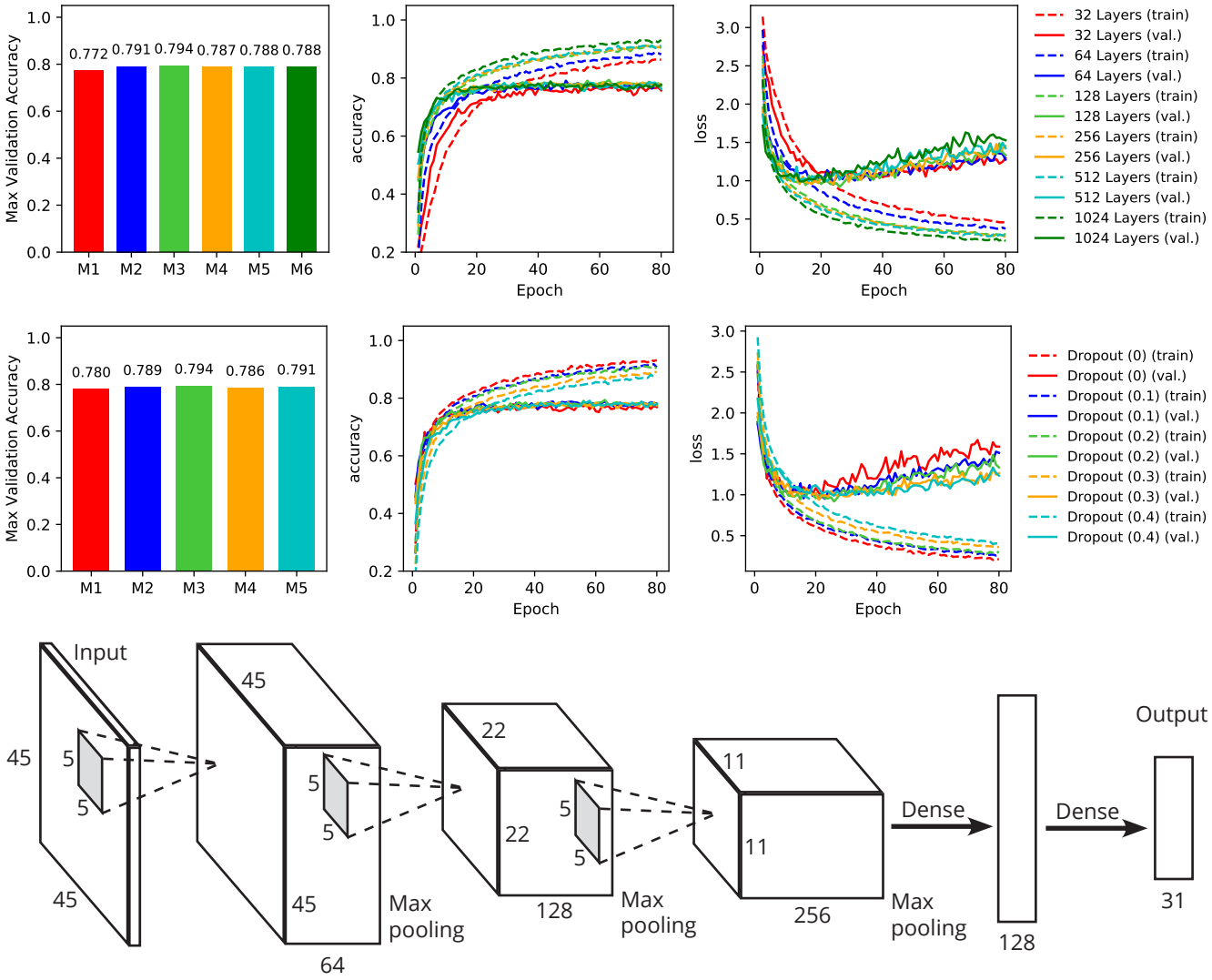

Fig. 6. **TOP:** CNN training accuracy and loss with varying sizes of the dense hidden layer. Here, the network architecture was [64C5-P2]-[128C5-P2]-X, where X is the number of nodes in the hidden dense layer (32, 64, 128, 256, 512, or 1024). **MIDDLE:** CNN training accuracy and loss with varying dropout rates in the dense hidden layer. Here, the network architecture modeled was [64C5-P2]-[128C5-P2]-128. The greatest validation accuracy was obtained with a dropout rate of 0.2. Bar charts (left) report the maximum validation accuracy obtained for each model across epochs, and plots summarize training and validation accuracy and loss across epochs. **BOTTOM:** Architecture of optimized CNN.