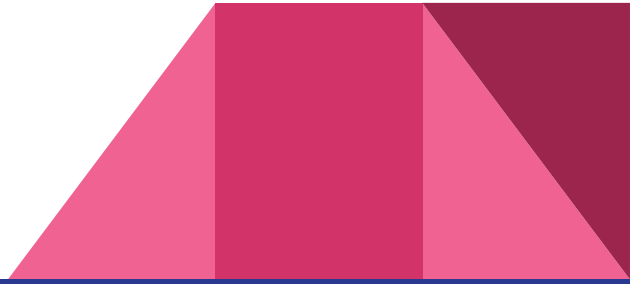
ASL Alphabet Classifiers

A comparative approach

Zoë Markovits, Daniel Grenell

Outline

1. The Problem
2. Other Work
3. The Data
4. ANN Experiments
5. CNN Experiments
6. Conclusions



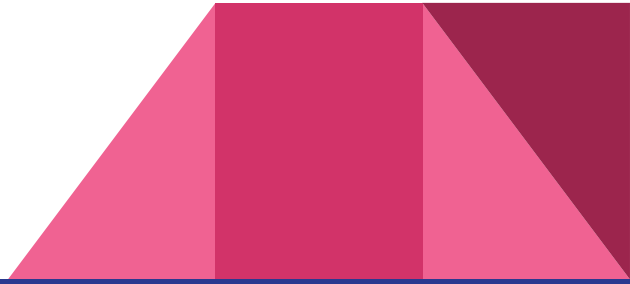
The Problem

Input: Static images of American Sign Language hand signs of the alphabet.

Output: A letter in the roman alphabet.

Why is this important?

Video communications have benefitted deaf and hard-of-hearing individuals greatly in human-human interactions. However, where hearing individuals can now communicate by voice with assistants like Siri and Alexa, the deaf and hard-of-hearing still communicate with computer systems primarily through text entry. The first step in building an image to text system for this group is to build a classifier like the one described here.



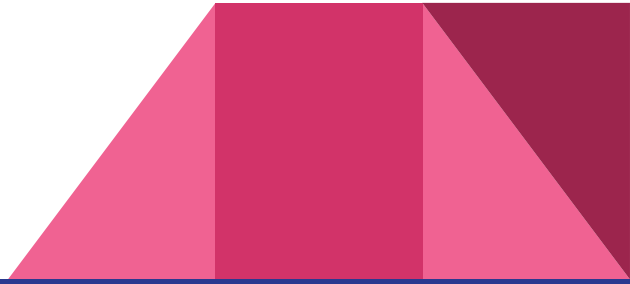
Other Work

Prior works examined different types of sign language recognition tasks:

- Isolated vs continuous sign recognition: a single gesture vs. a sentence of sign gestures
- Direct measurement vs vision based recognition: information gathered via data gloves and sensors vs. using cameras

Past studies used Hidden Markov Models, Multi Layered Perceptrons, and Convolutional Neural Networks

- Filters used: gabor, gaussian, discrete cosine transform, particle

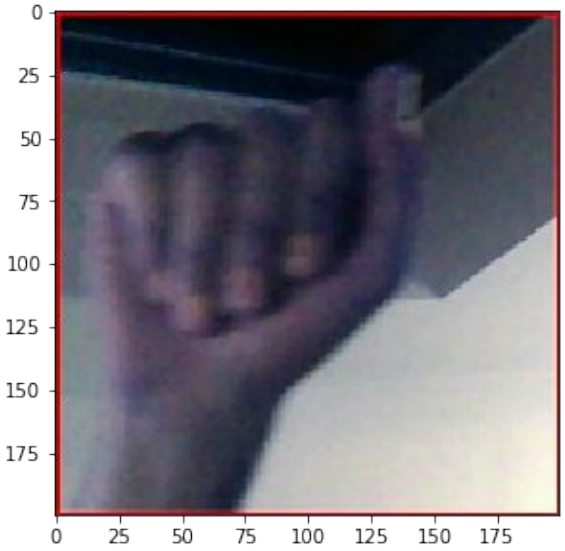


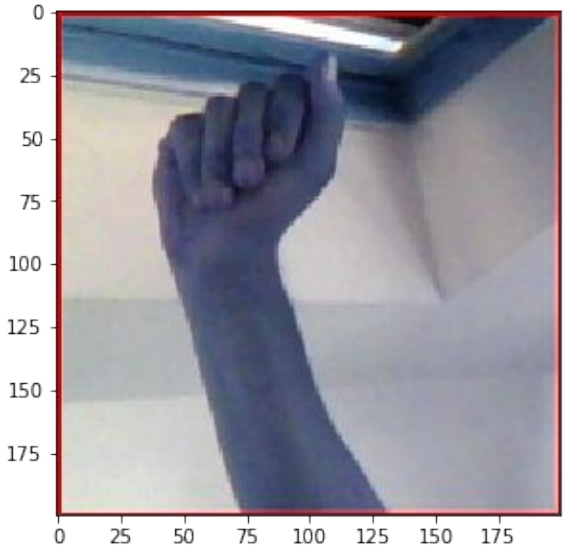
The Data

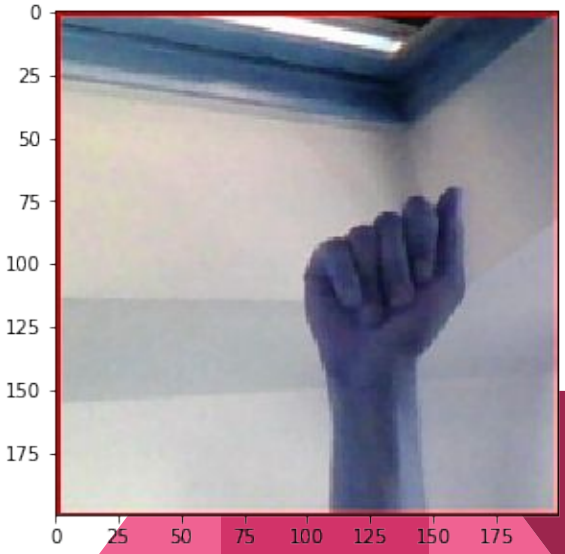
RGB images, 200x200

Generated in a classroom, near-neutral background, no occlusion, no faces

All images appear to depict the same hand







The Data

29 classes (the alphabet and three other signs)

Each class has 3000 samples

Samples were split into test, validation, and training sets

ANN

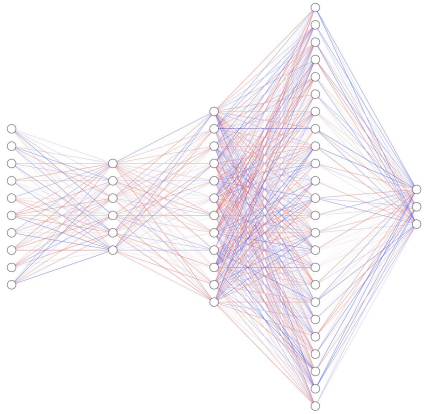
The literature has examples of simple feed-forward networks being used to categorize signs.

Can a multi-layer perceptron learn the ASL alphabet?

In total nine experiments were conducted.

	Adam	**Adamax**	**RMSProp**
B/W Images	raw_model1	raw_model2	raw_model3
Preprocessed Images	base_model1	base_model2	base_model3
Histograms	hist_model1	hist_model2	hist_model3

ANN



Models were trained for 20 epochs

ReLU activations

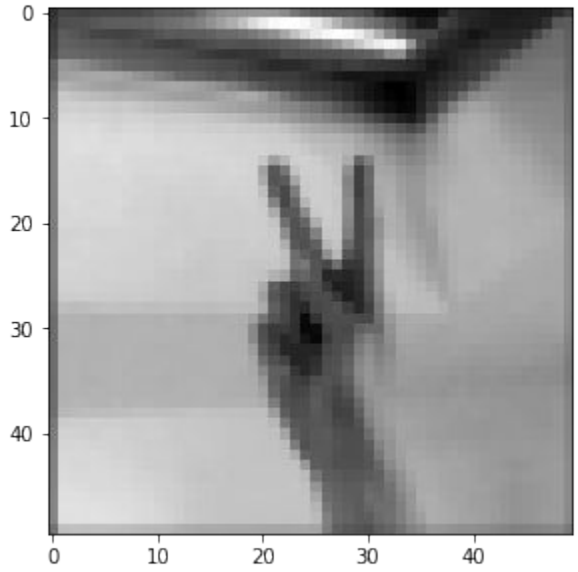
Accuracy was used for the performance metric

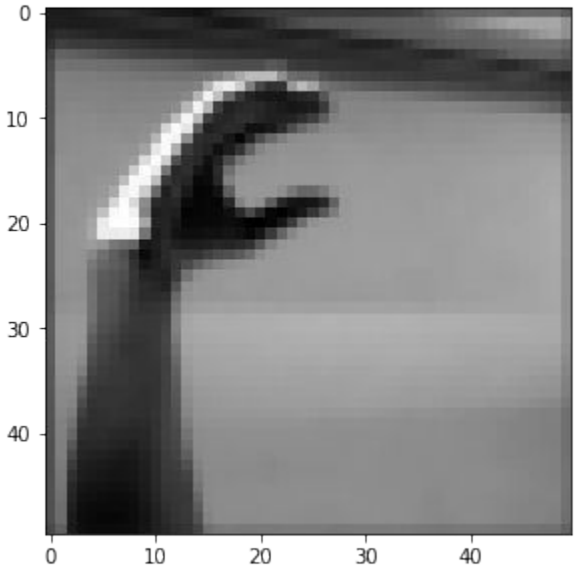
Layer (type)	Output Shape	Param #
flatten_4 (Flatten)	(None, 2500)	0
dense_13 (Dense)	(None, 128)	320128
dropout_10 (Dropout)	(None, 128)	0
dense_14 (Dense)	(None, 256)	33024
dropout_11 (Dropout)	(None, 256)	0
dense_15 (Dense)	(None, 512)	131584
dropout_12 (Dropout)	(None, 512)	0
dense_16 (Dense)	(None, 29)	14877

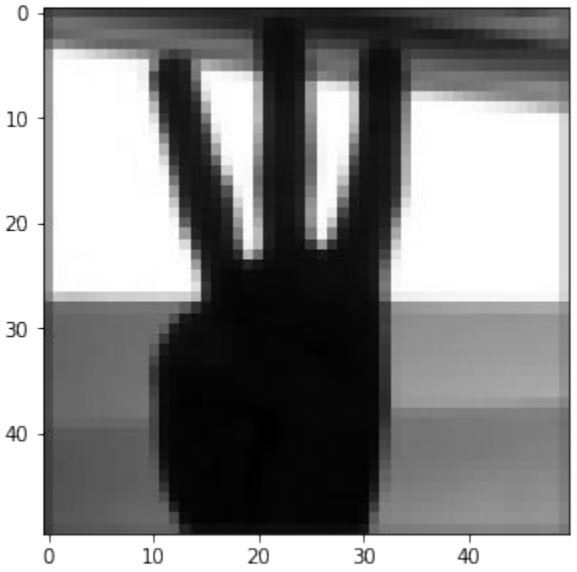
Total params: 499,613
Trainable params: 499,613
Non-trainable params: 0

ANN: B/W Images

Images downscaled to 50x50 and converted to grayscale

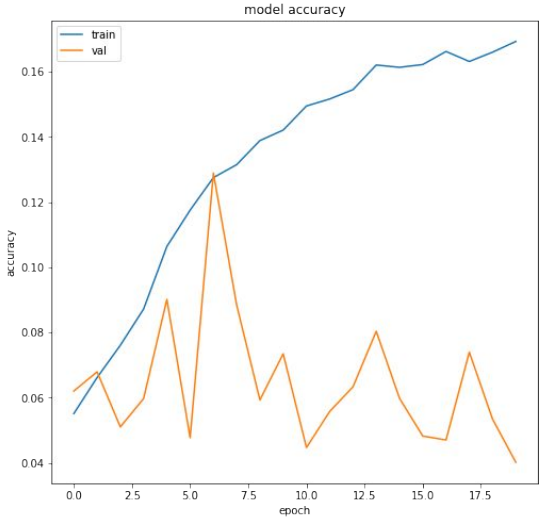


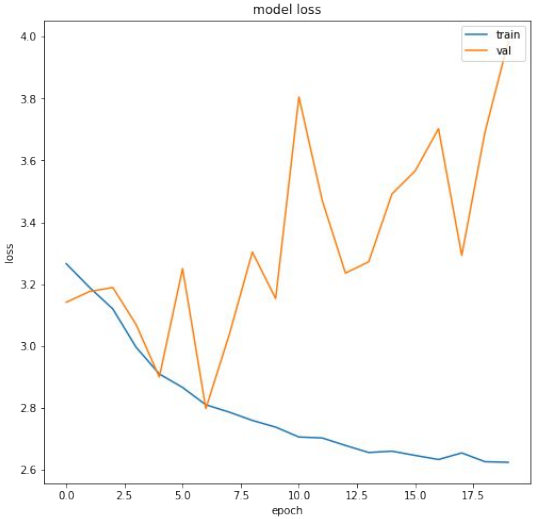


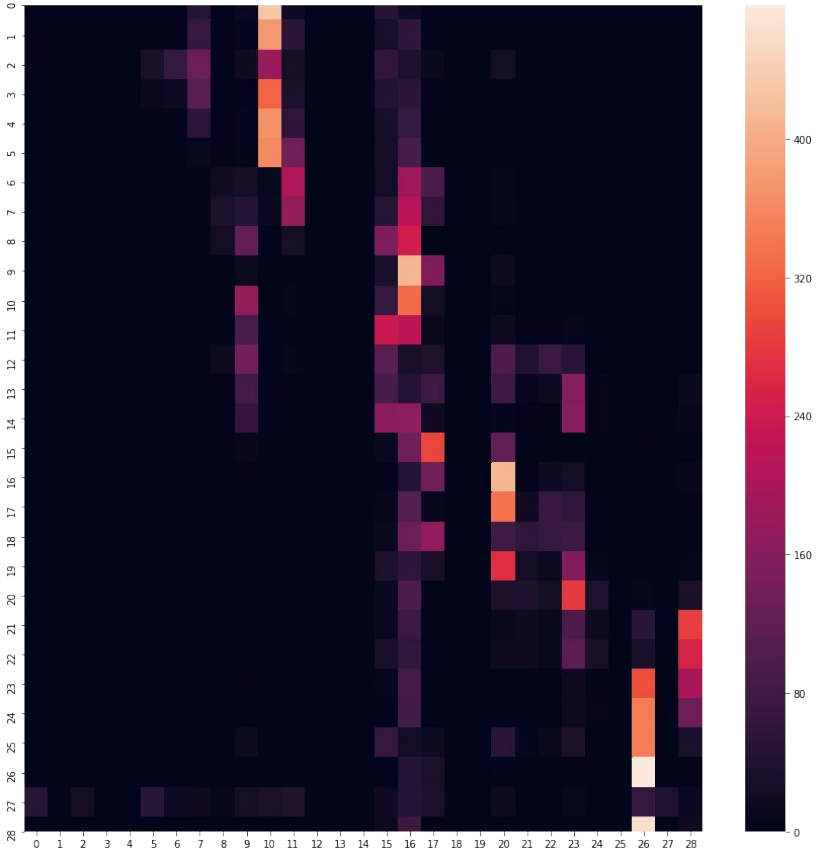


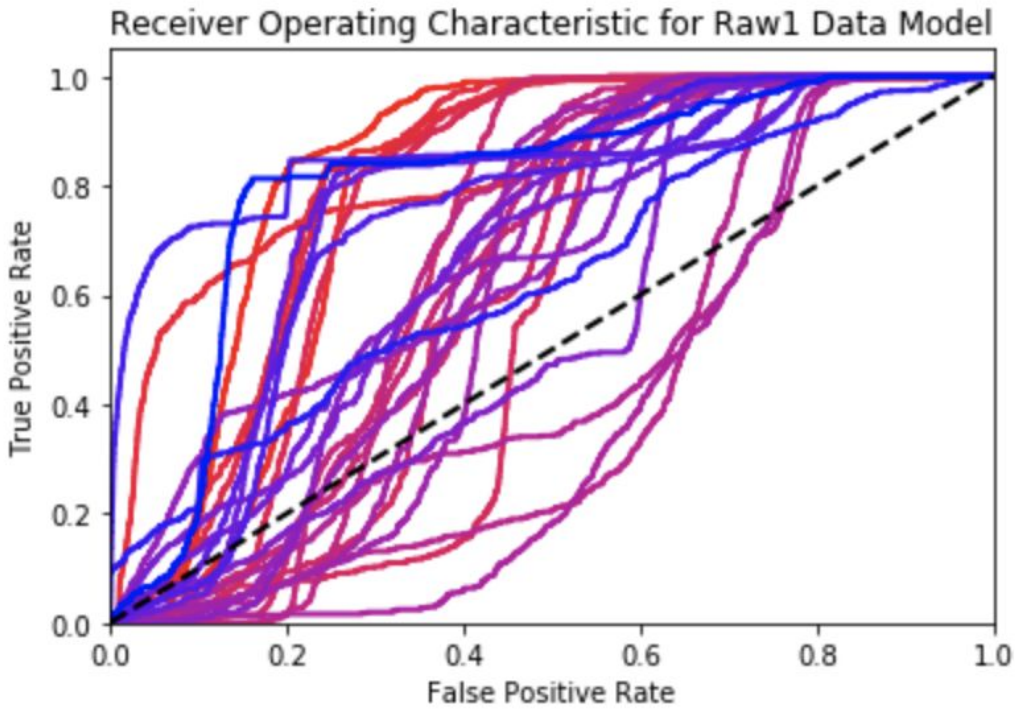
ANN: B/W Images

Adam Optimizer



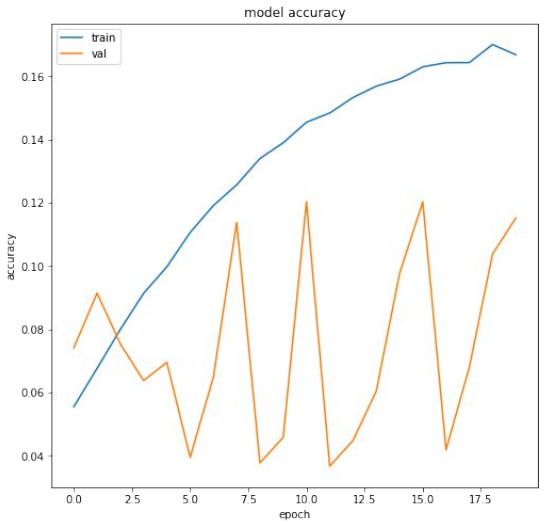


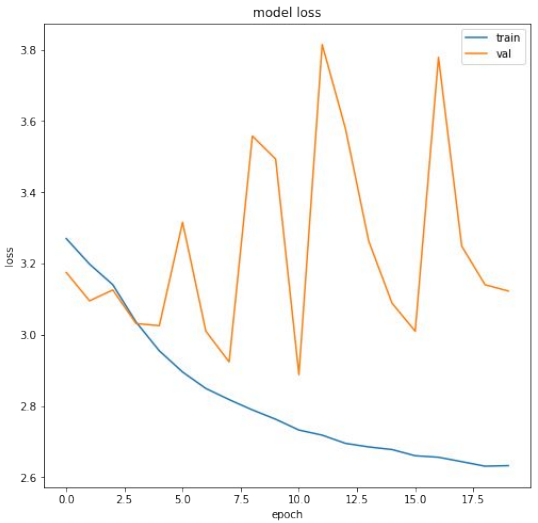


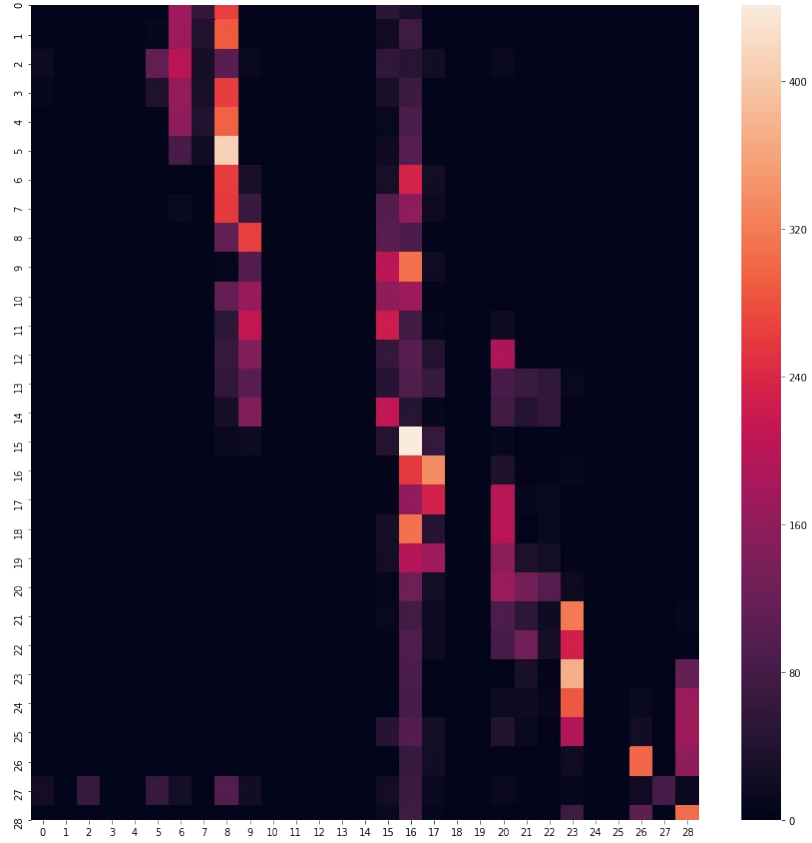


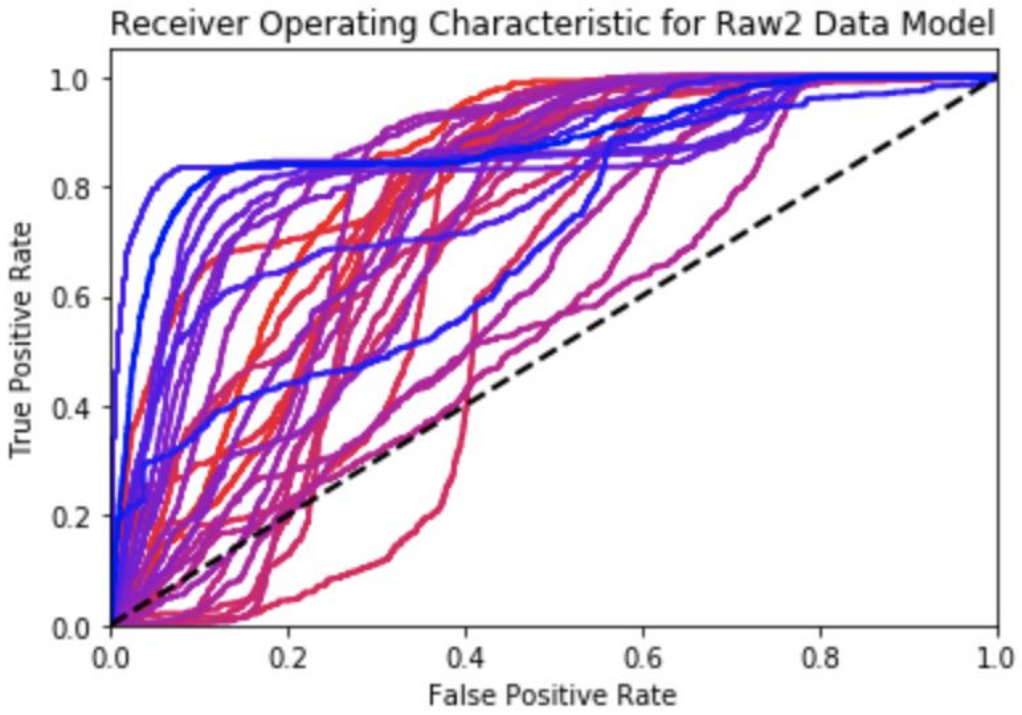
ANN: B/W Images

Adamax Optimizer



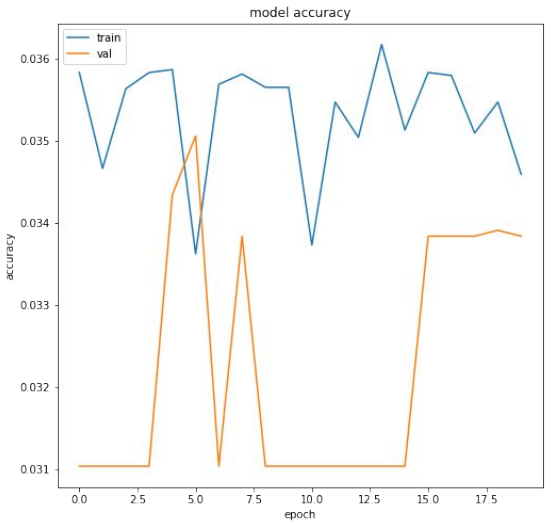


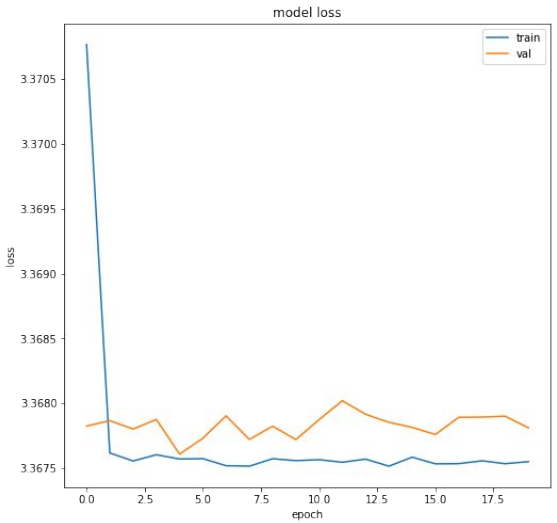


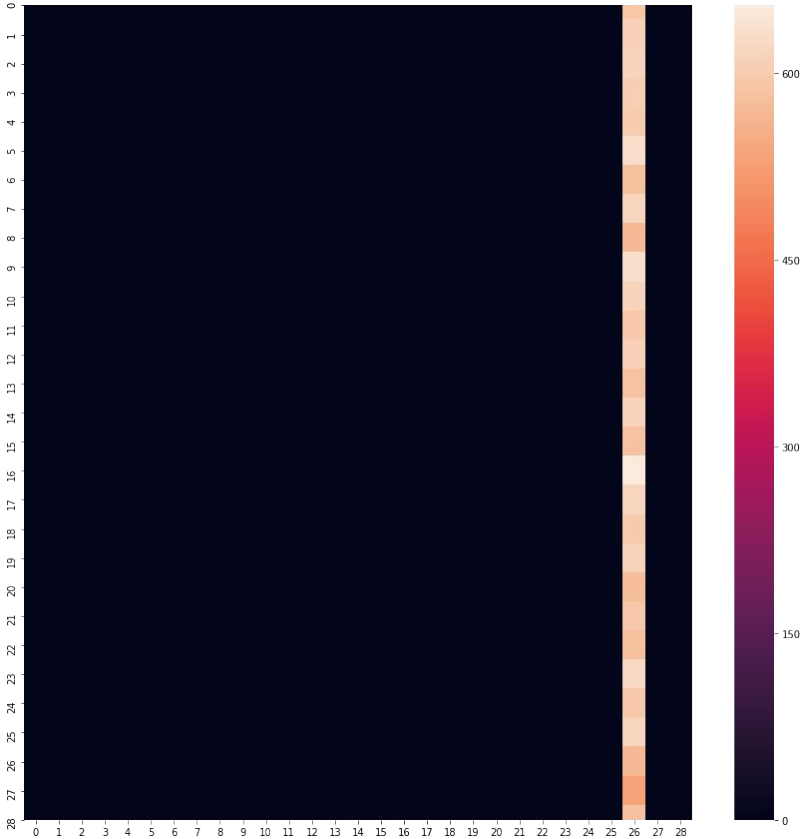


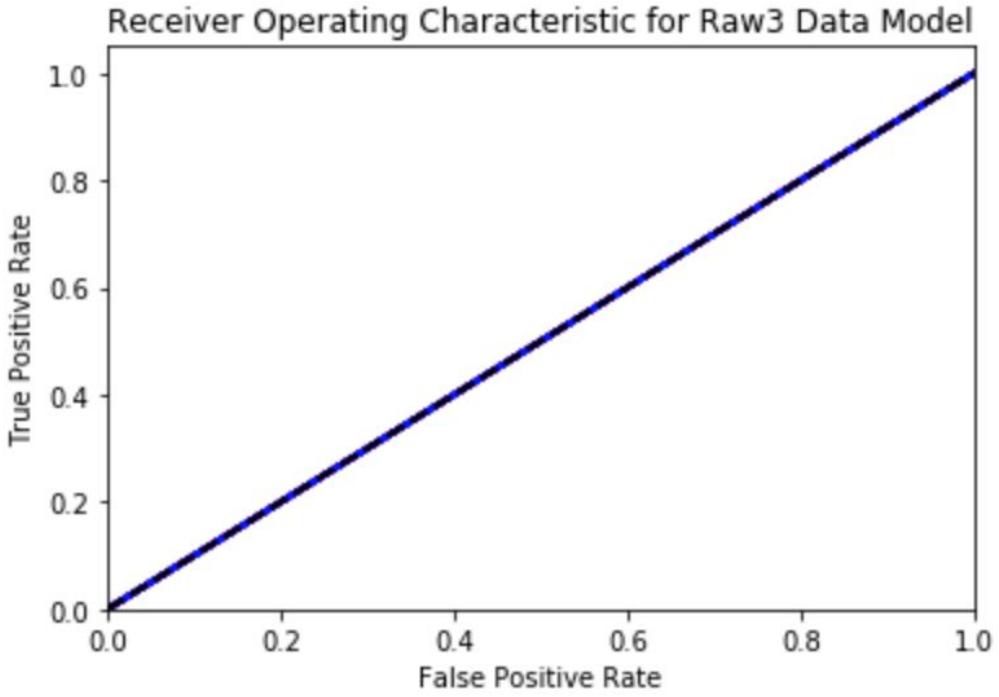
ANN: B/W Images

RMSProp Optimizer



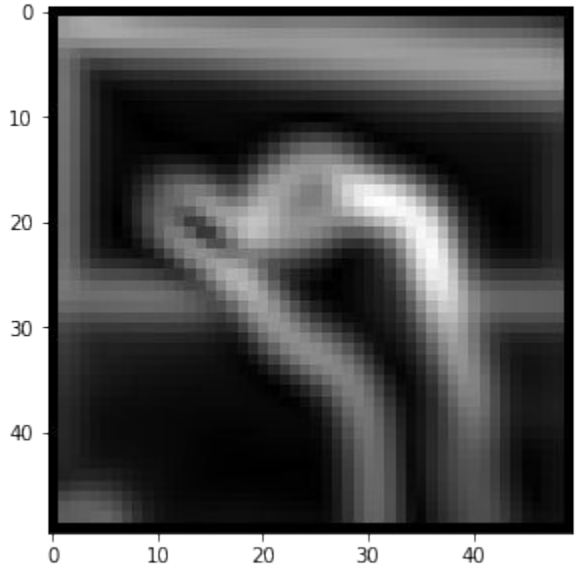


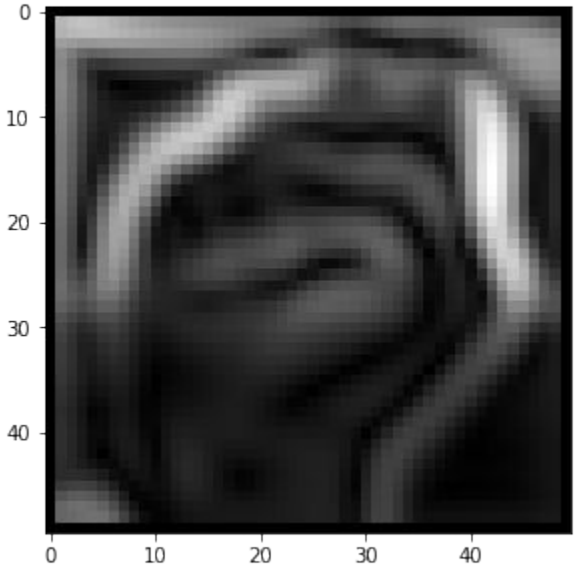


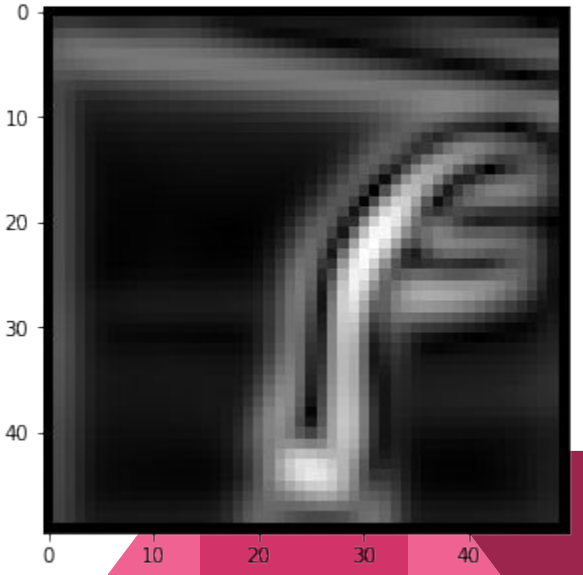


ANN: Preprocessed Images

Images were pre-processed before training to reduce extract salient features for the model to train on





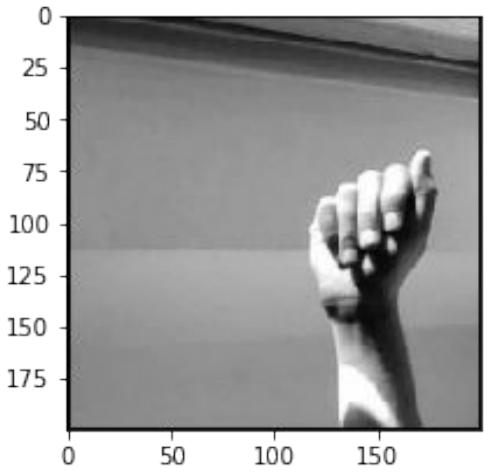


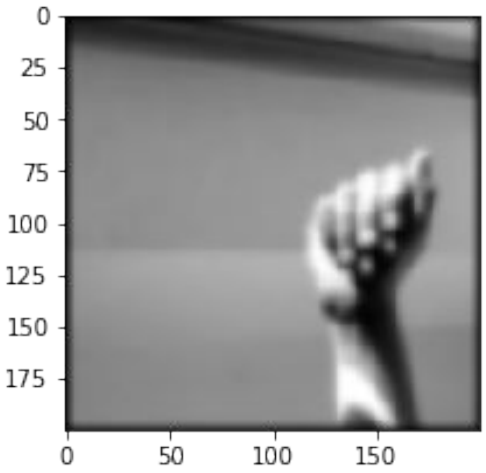
ANN: Preprocessed

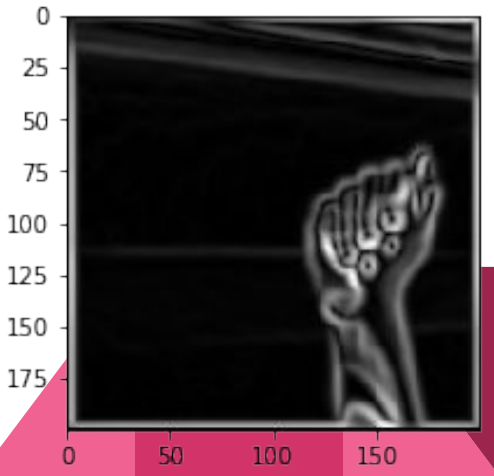
```
def image_preprocessing(img):
    img = ski.color.rgb2gray(img)
    img = ski.filters.gaussian(img, sigma = 2)
    img = ski.filters.sobel(img)
    return img
```

1. Grayscale conversion (as in the previous models)
2. Gaussian blur
    a. Multi-dimensional gaussian filter, sigma is the standard deviation of the gaussian
    b. Used here to reduce noise prior to edge detection
3. Sobel filter
    a. Convolve two filters over image, find their magnitude,
    b. This is a discrete approximation of the discrete gradient

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

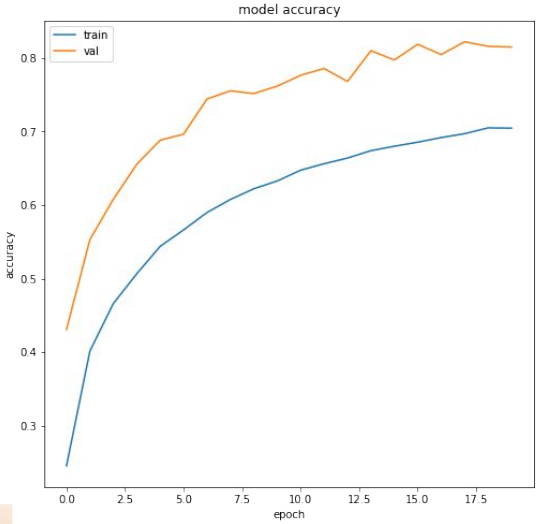


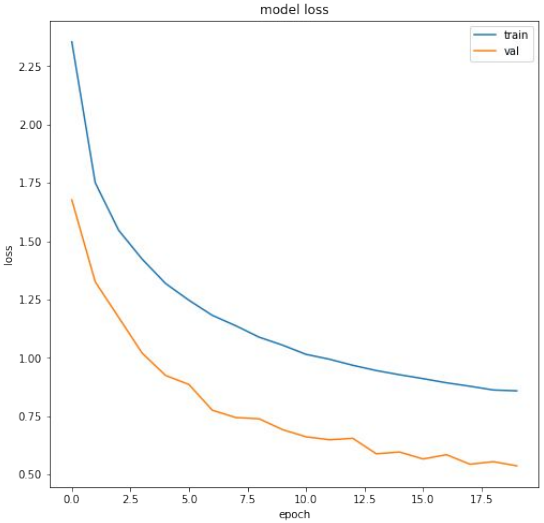


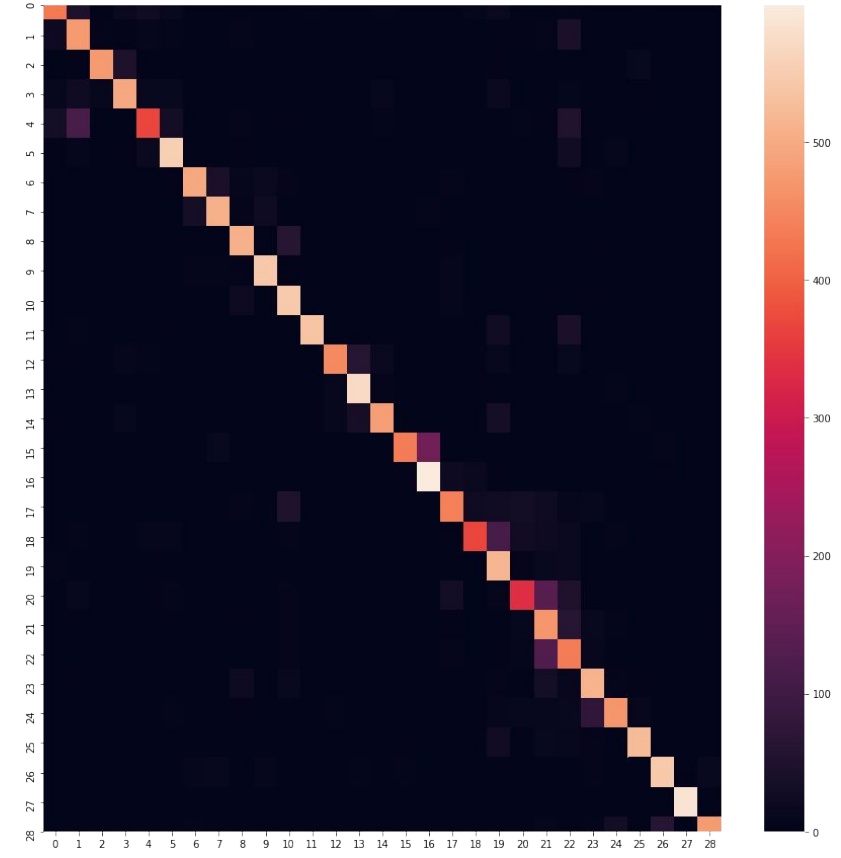


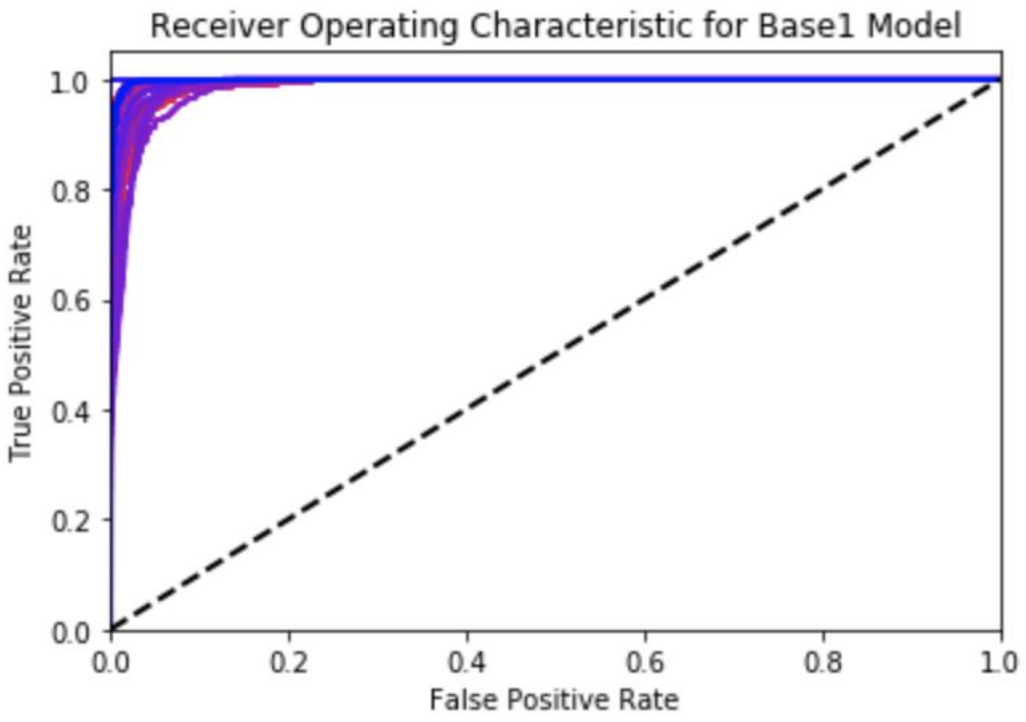
ANN: Preprocessed

Adam Optimizer



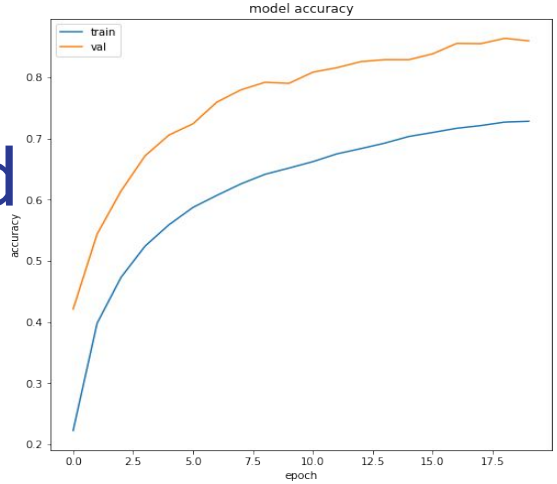


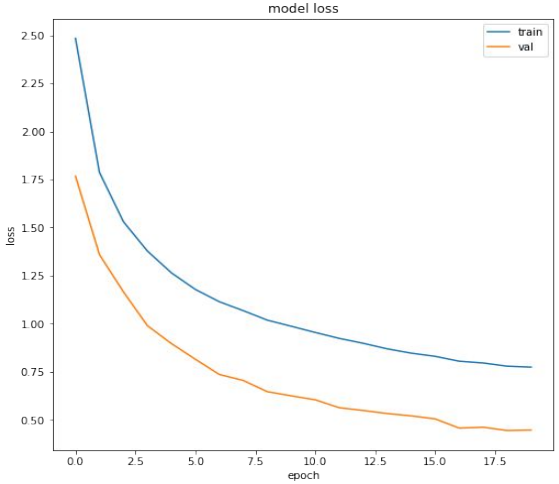


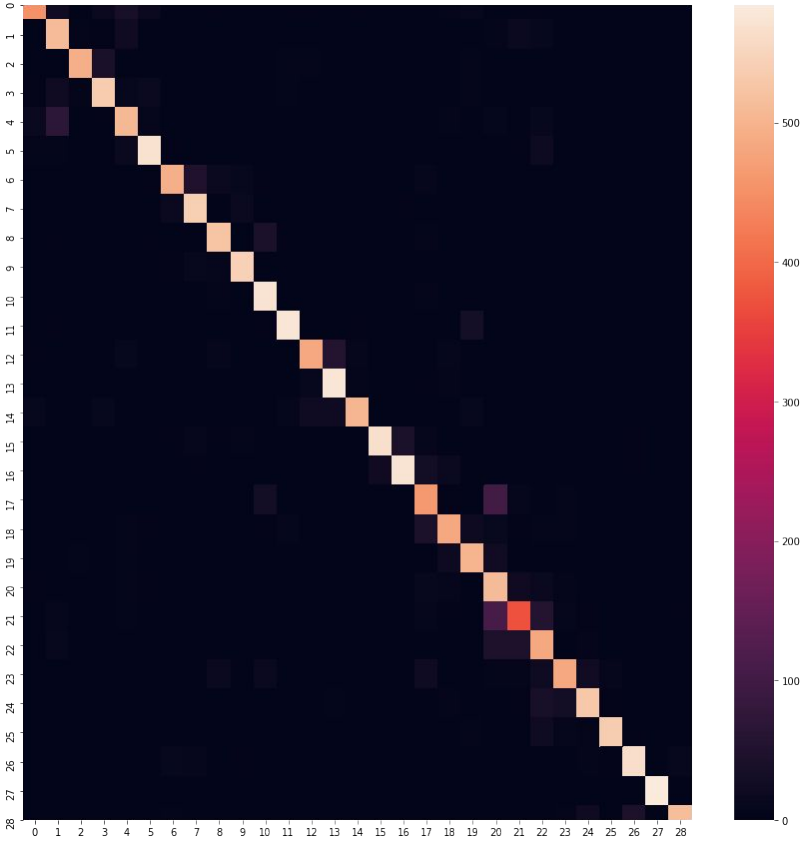


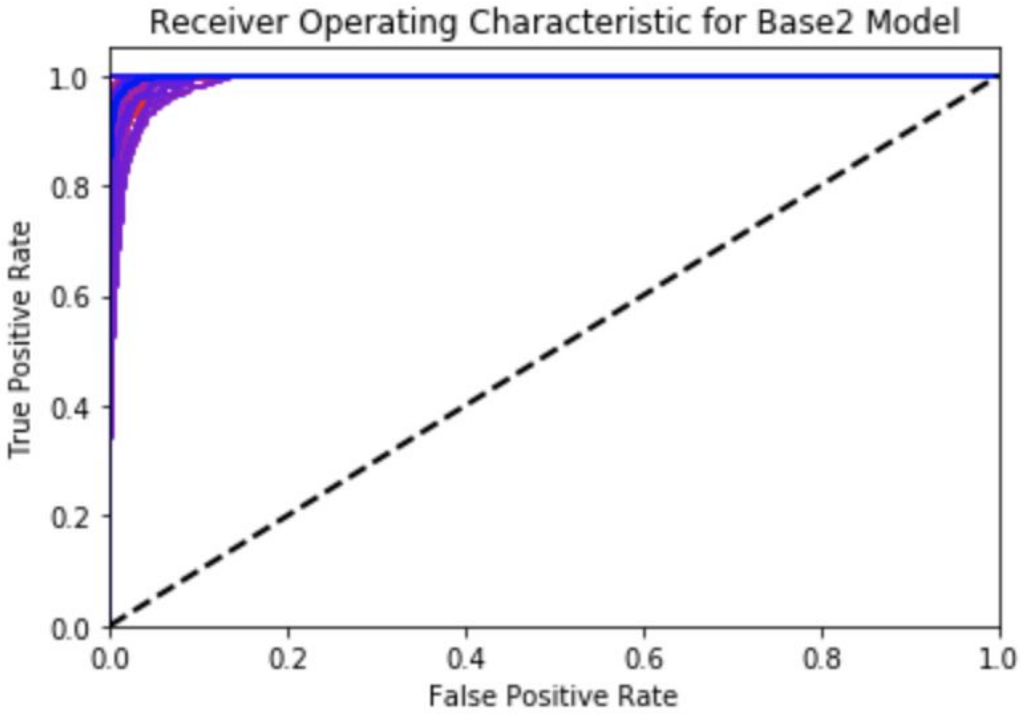
ANN: Preprocessed

Adamax Optimizer



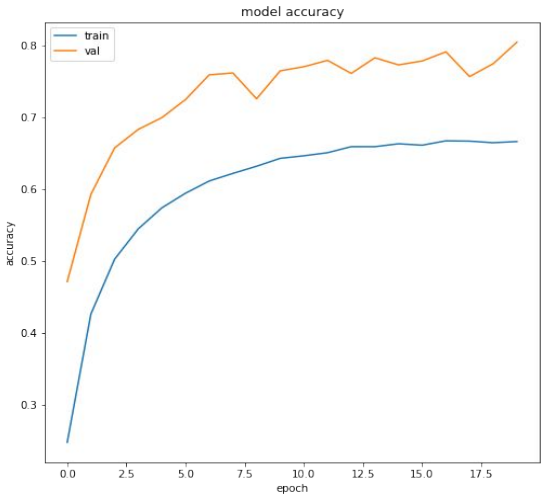


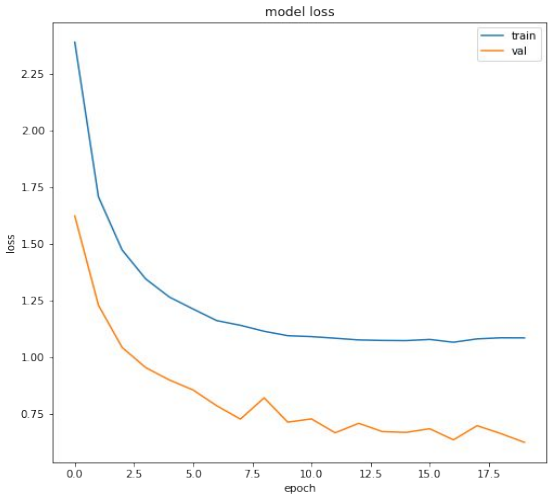


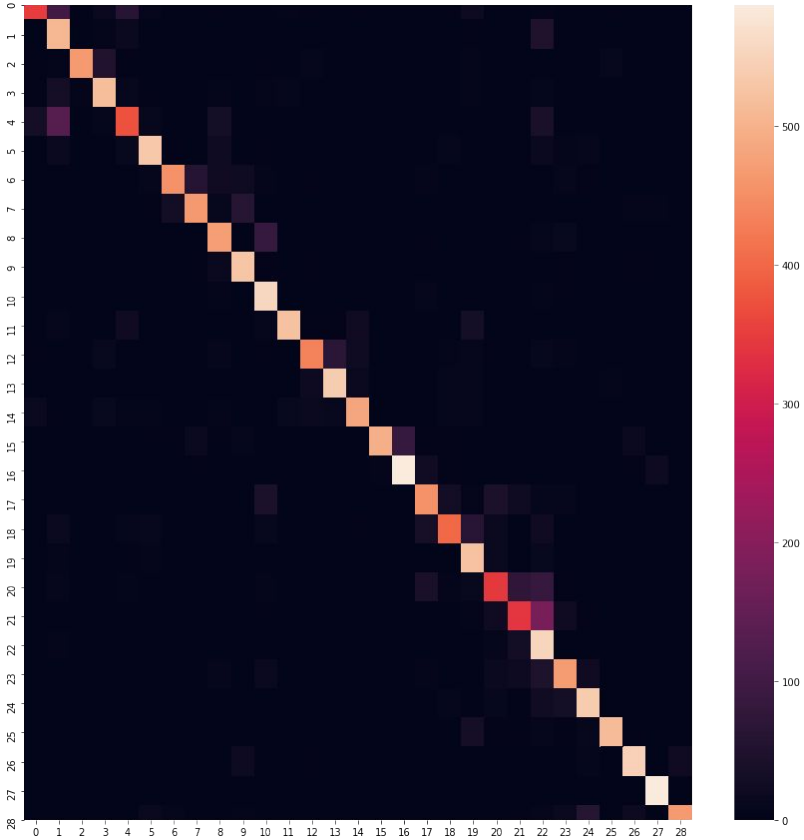


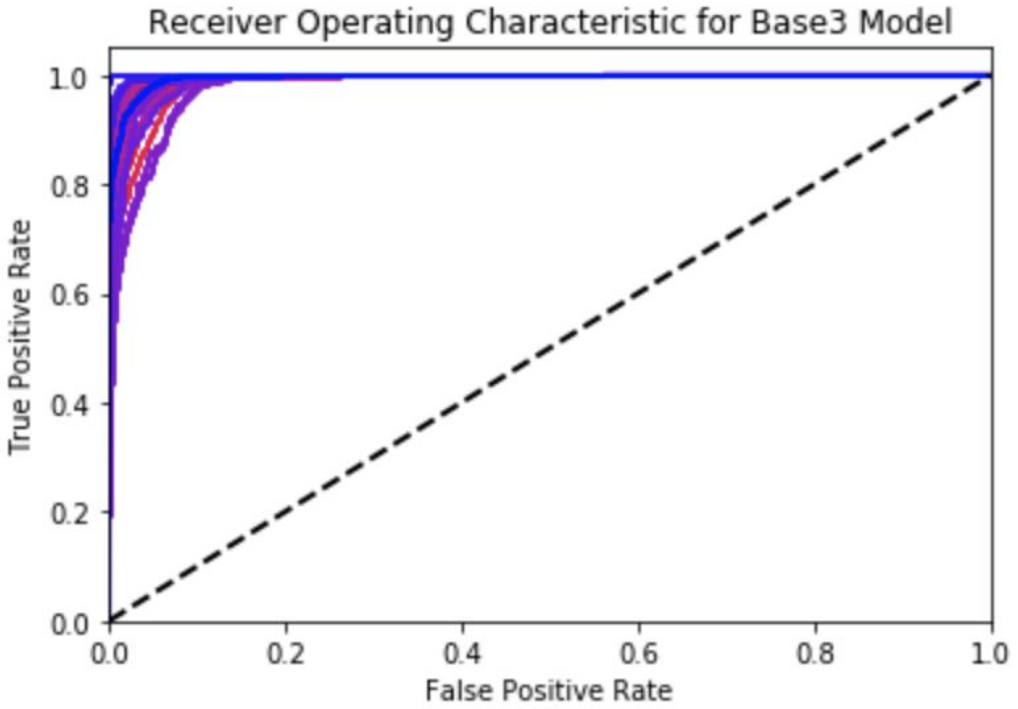
ANN: Preprocessed

RMSProp Optimizer









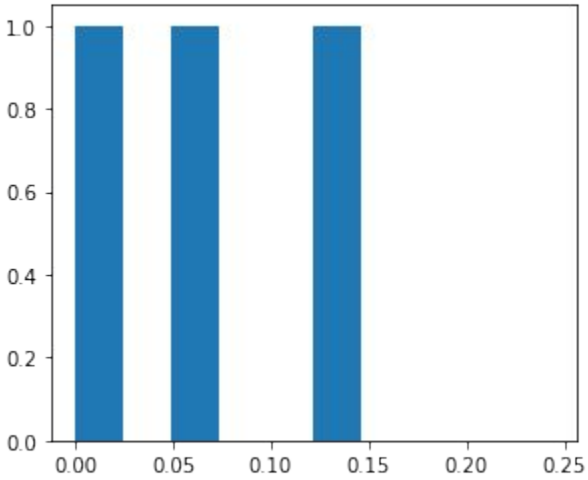
ANN: Histograms

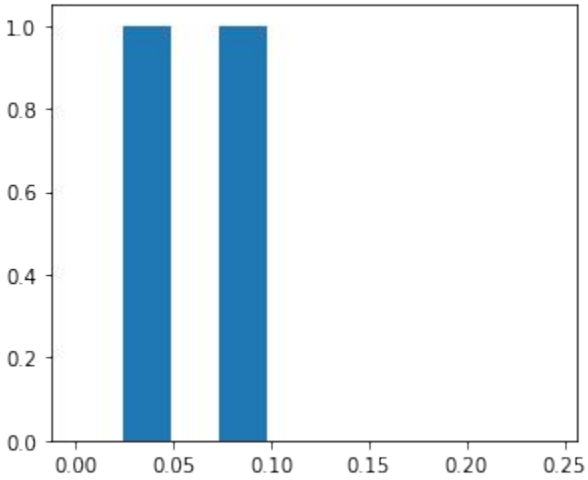
```
X_train_hist = [np.histogram(x, bins = bin_edges)[0] for x in X_train]
X_train_hist = np.array(X_train_hist)
X_train_hist = X_train_hist - X_train_hist.mean()
X_train_hist = X_train_hist / np.abs(X_train_hist).max()
```

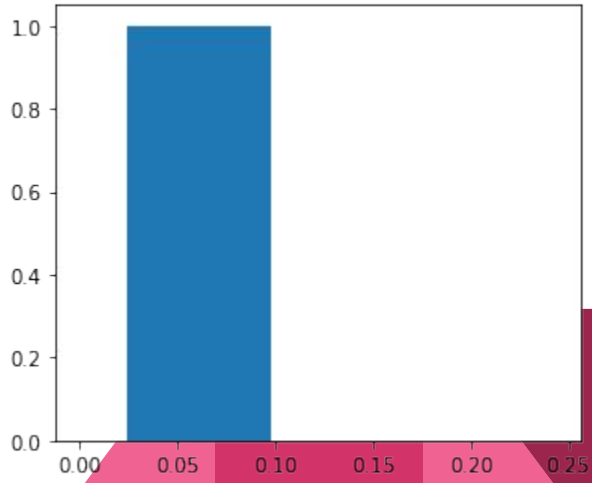
Bin edges were determined on the entire training set

Images were mapped to their histograms

Histograms were zero centered and scaled to unit size

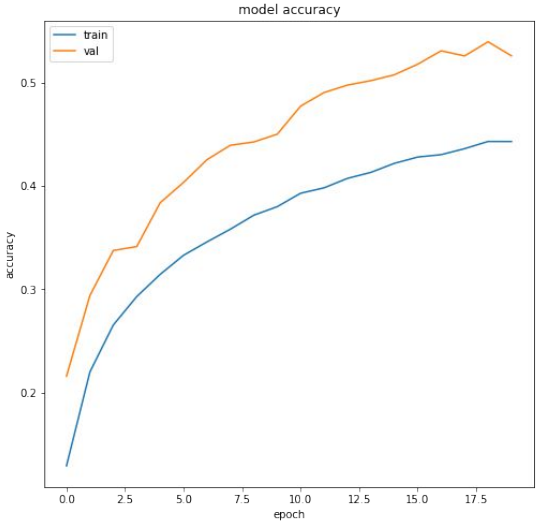


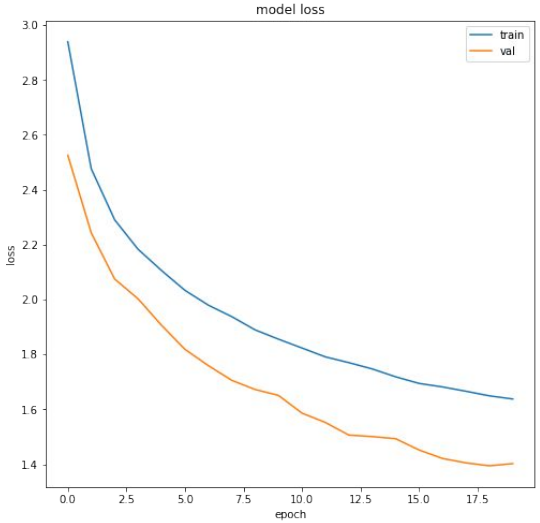


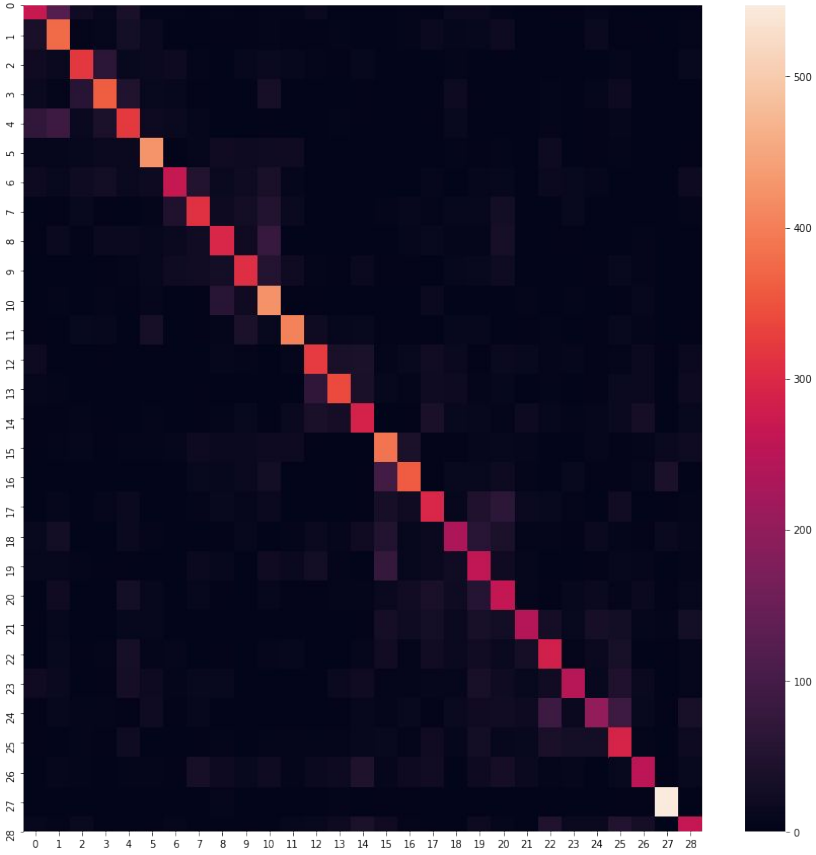


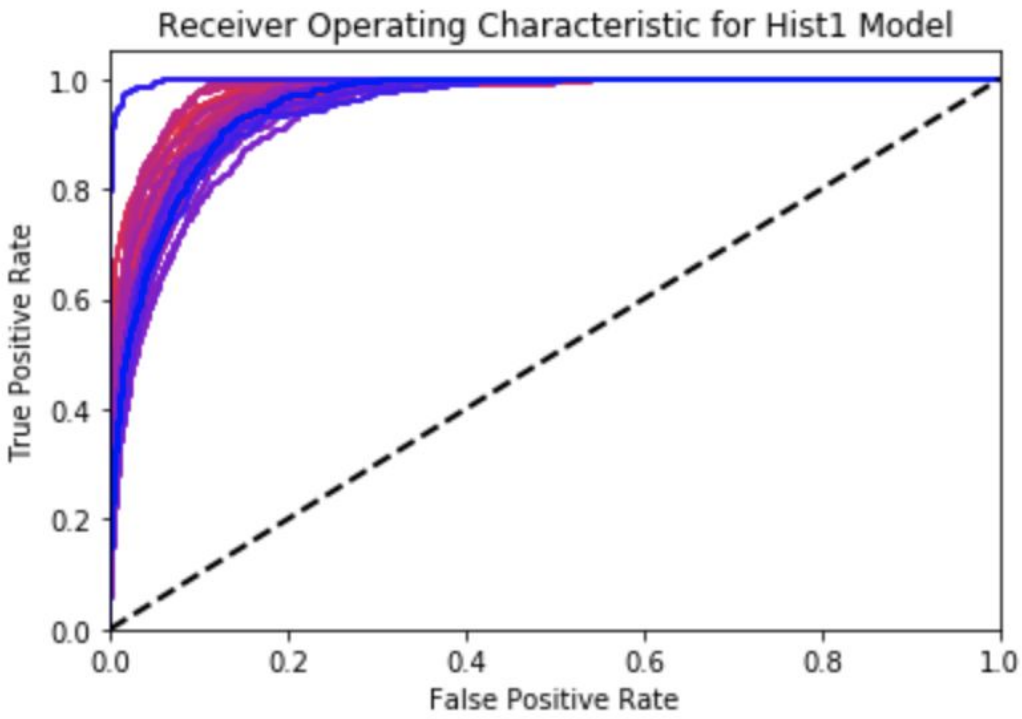
ANN: Histograms

Adam Optimizer



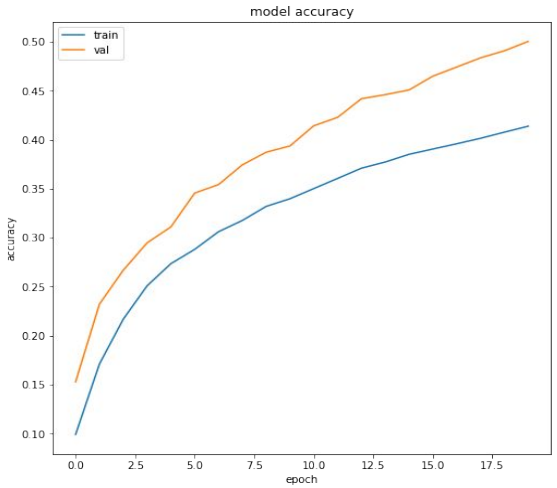


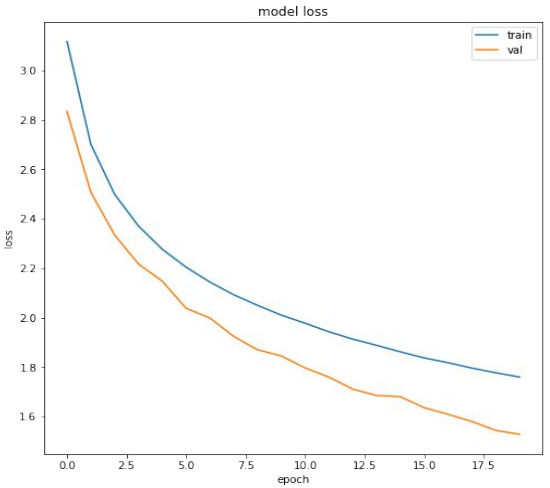


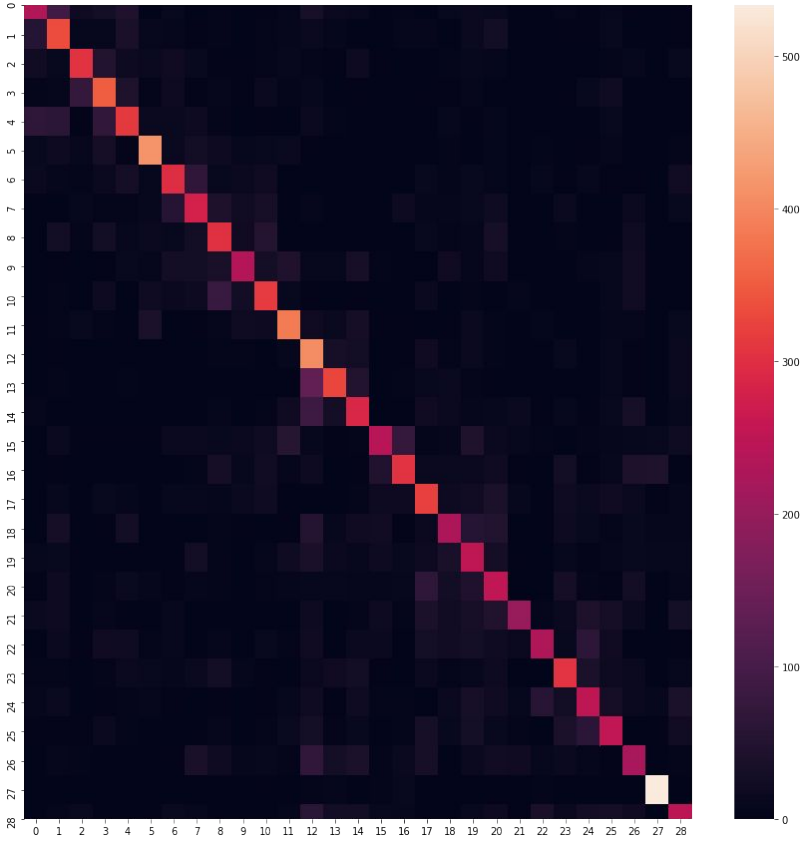


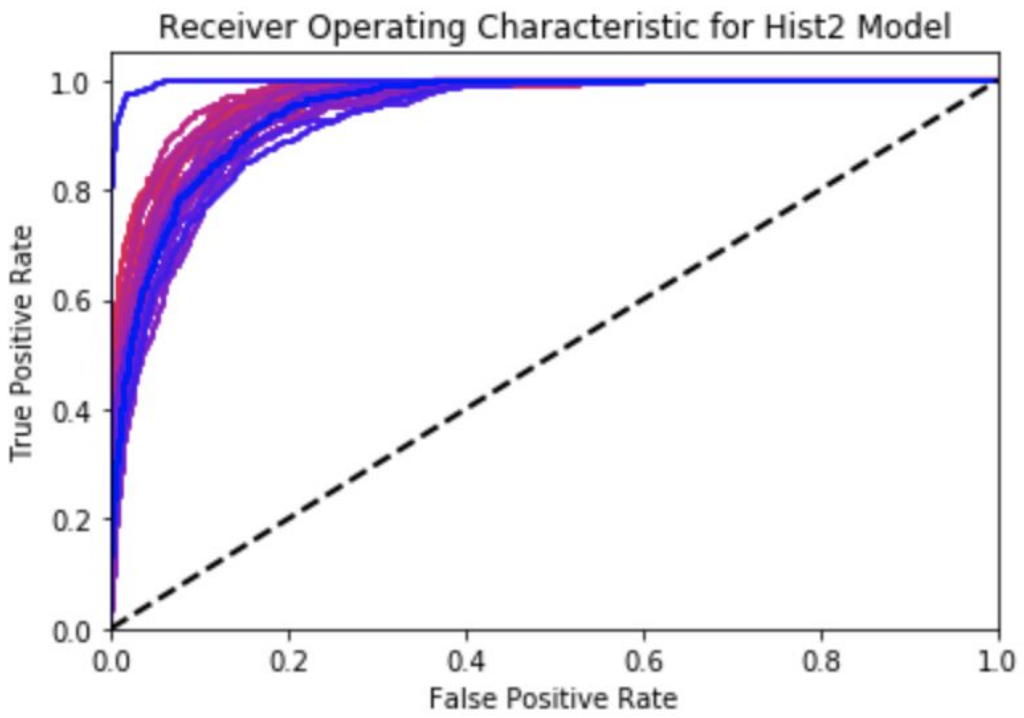
ANN: Histograms

Adamax Optimizer



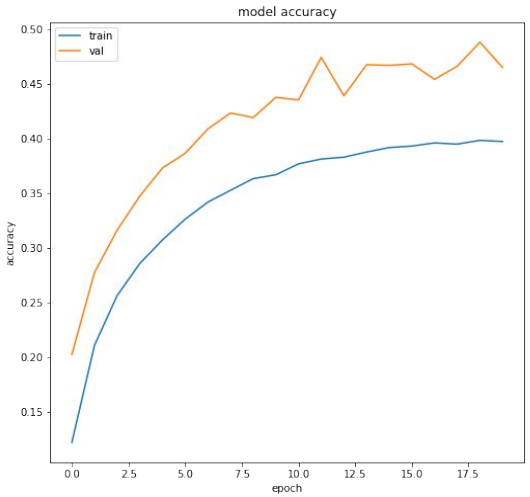


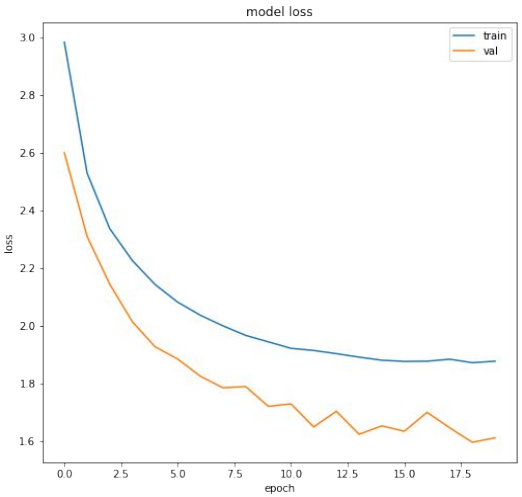


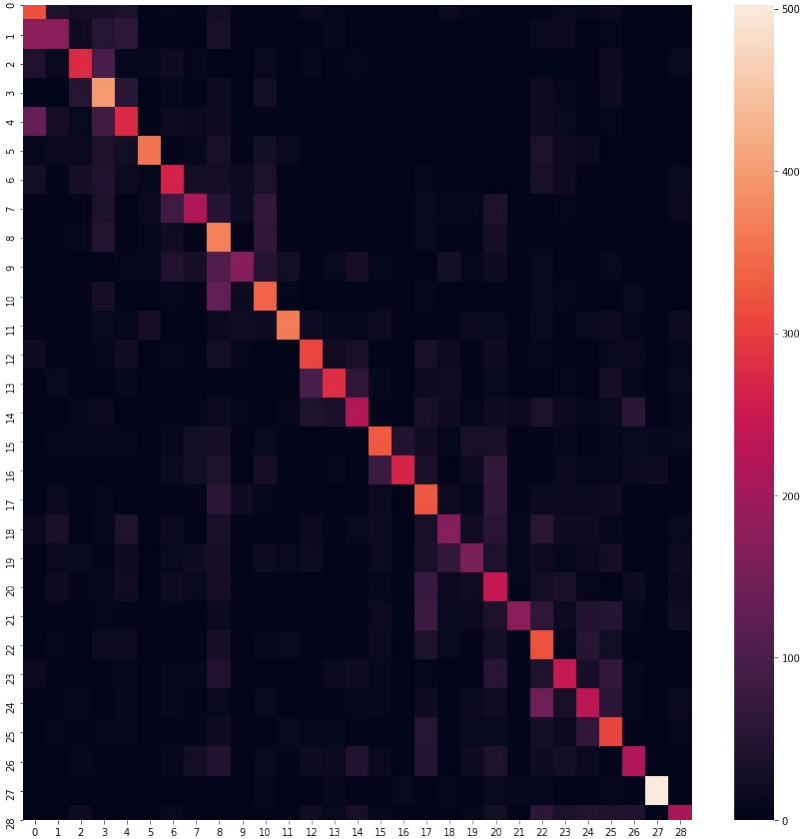


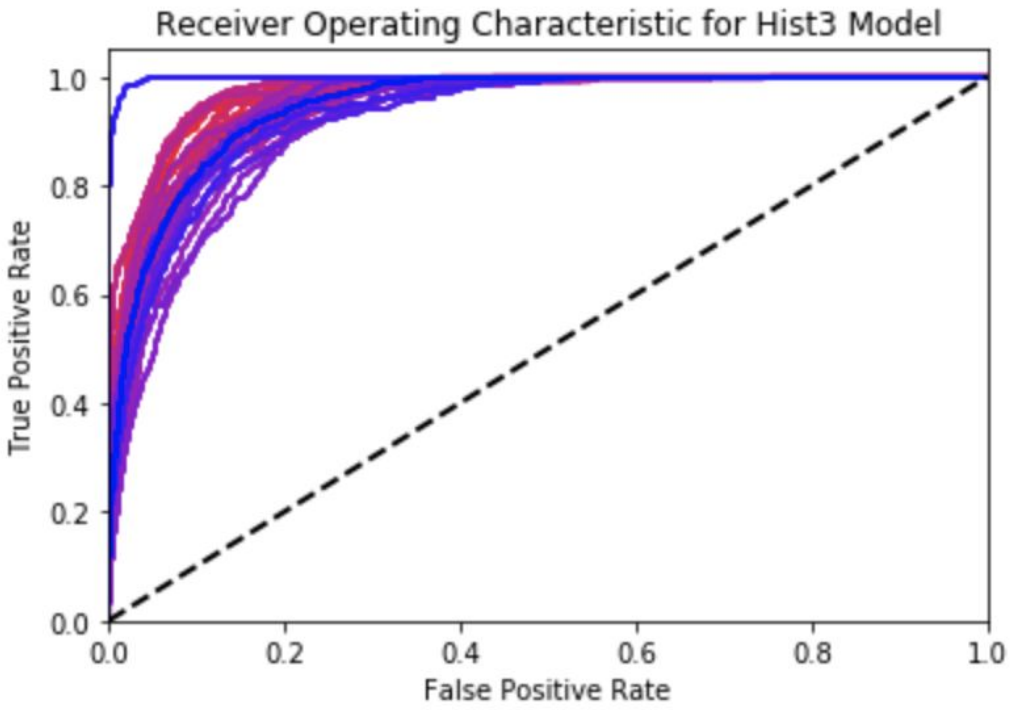
ANN: Histograms

RMSProp Optimizer





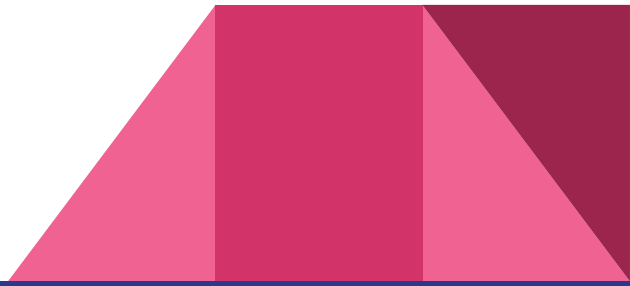




ANN

So how did the different models do?

	Adam	Adamax	RMSProp
B/W Images	3.5%	3.3%	3.5%
Preprocessed Images	81.5%	**86.5%**	80.6%
Histograms	52.7%	49.7%	46.2%



CNN: Architecture with Batch Normalization

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 60, 60, 64)	4864
batch_normalization_1 (Batch	(None, 60, 60, 64)	256
conv2d_6 (Conv2D)	(None, 56, 56, 64)	102464
max_pooling2d_3 (MaxPooling2	(None, 28, 28, 64)	0
batch_normalization_2 (Batch	(None, 28, 28, 64)	256
conv2d_7 (Conv2D)	(None, 24, 24, 128)	204928
batch_normalization_3 (Batch	(None, 24, 24, 128)	512
conv2d_8 (Conv2D)	(None, 20, 20, 256)	819456
max_pooling2d_4 (MaxPooling2	(None, 10, 10, 256)	0
batch_normalization_4 (Batch	(None, 10, 10, 256)	1024
flatten_2 (Flatten)	(None, 25600)	0
dropout_2 (Dropout)	(None, 25600)	0
dense_3 (Dense)	(None, 512)	13107712
dense_4 (Dense)	(None, 29)	14877

CNN model architecture with batch normalization:

4 convolutional layers, 2 max pooling layers, 2 fully connected layers, ReLU activation function

Avoid overfitting by using Dropout and L2 regularization

CNN: Architecture without Batch Normalization

Layer (type)	Output Shape	Param #
conv2d_9 (Conv2D)	(None, 62, 62, 64)	1792
conv2d_10 (Conv2D)	(None, 60, 60, 64)	36928
max_pooling2d_5 (MaxPooling2	(None, 30, 30, 64)	0
conv2d_11 (Conv2D)	(None, 28, 28, 128)	73856
conv2d_12 (Conv2D)	(None, 26, 26, 256)	295168
max_pooling2d_6 (MaxPooling2	(None, 13, 13, 256)	0
flatten_3 (Flatten)	(None, 43264)	0
dropout_3 (Dropout)	(None, 43264)	0
dense_5 (Dense)	(None, 512)	22151680
dense_6 (Dense)	(None, 29)	14877

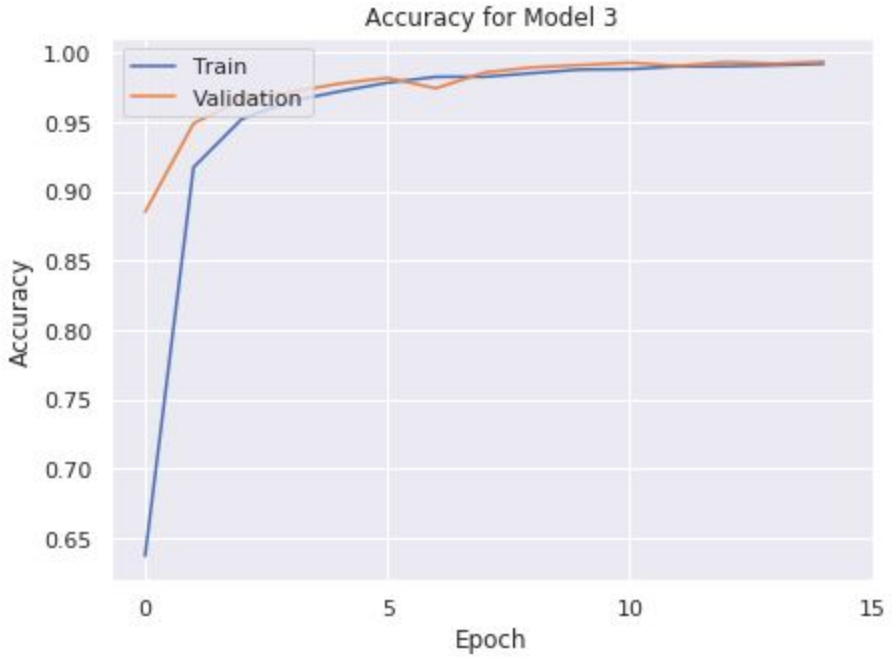
CNN model architecture without batch normalization:

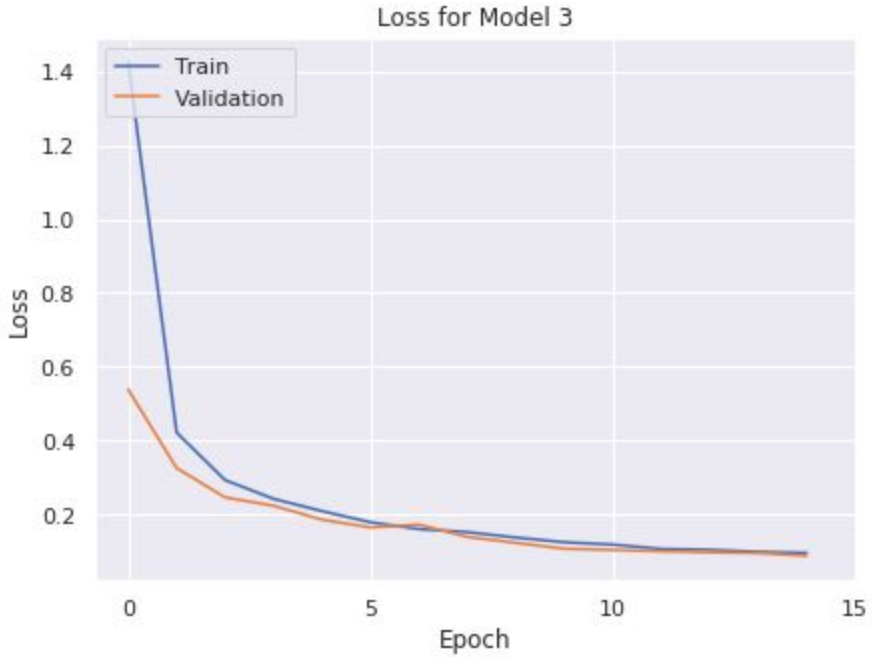
4 convolutional layers, 2 max pooling layers, 2 fully connected layers, ReLU activation function

Avoid overfitting by using Dropout and L2 regularization

CNN: Exploring Hyper-Parameters

Twelve models were run tuning three hyper-parameters: Dropout size 0.3, 0.4, and 0.5; kernel size 3 x 3 and 5 x 5; batch normalization or no batch normalization



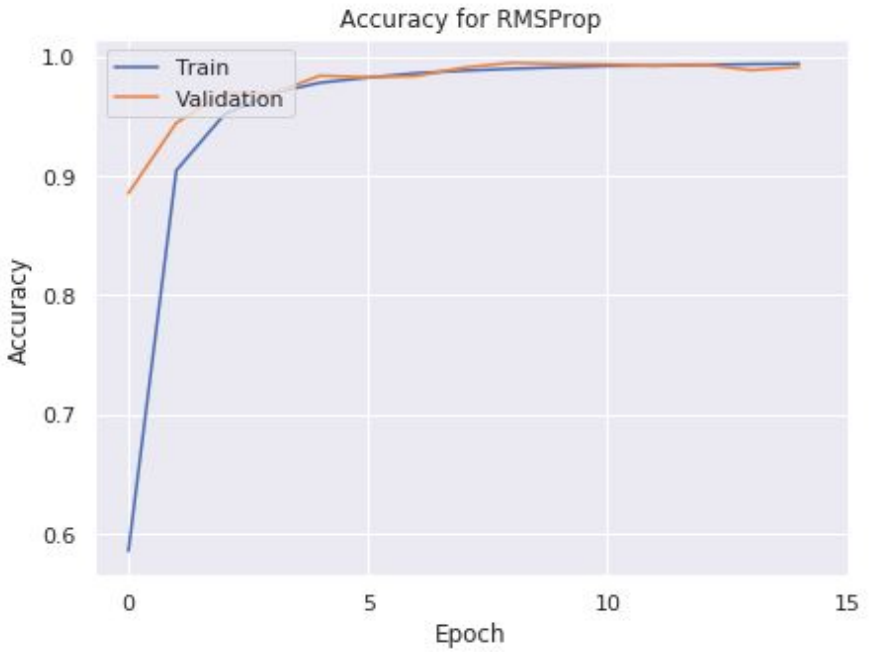


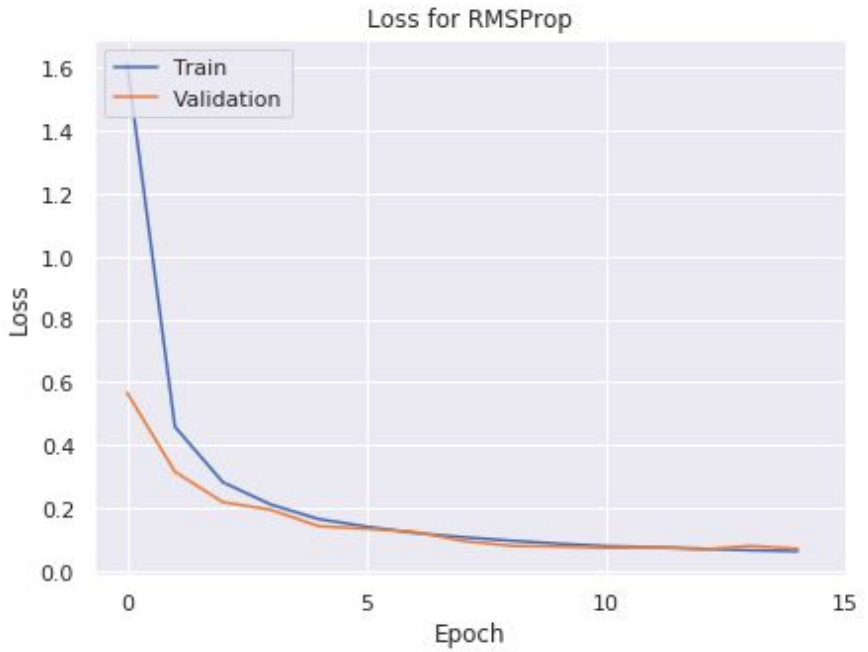
Best model of this experiment (Model 3) has Dropout = 0.3,

Kernel Size = 3 x 3, and No Batch Normalization

CNN: Exploring Optimizers

Use best model from previous step tuned with five different optimizers: Stochastic Gradient Descent, RMSProp, AdaGrad, ADAM, and AdaMax

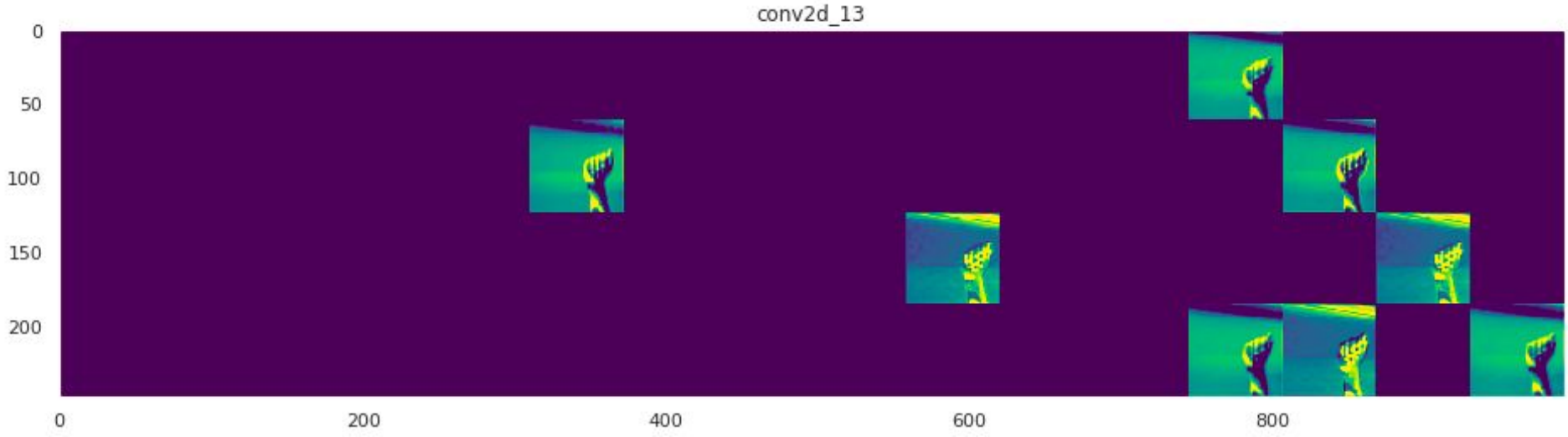




Best model of this experiment uses RMSProp optimizer

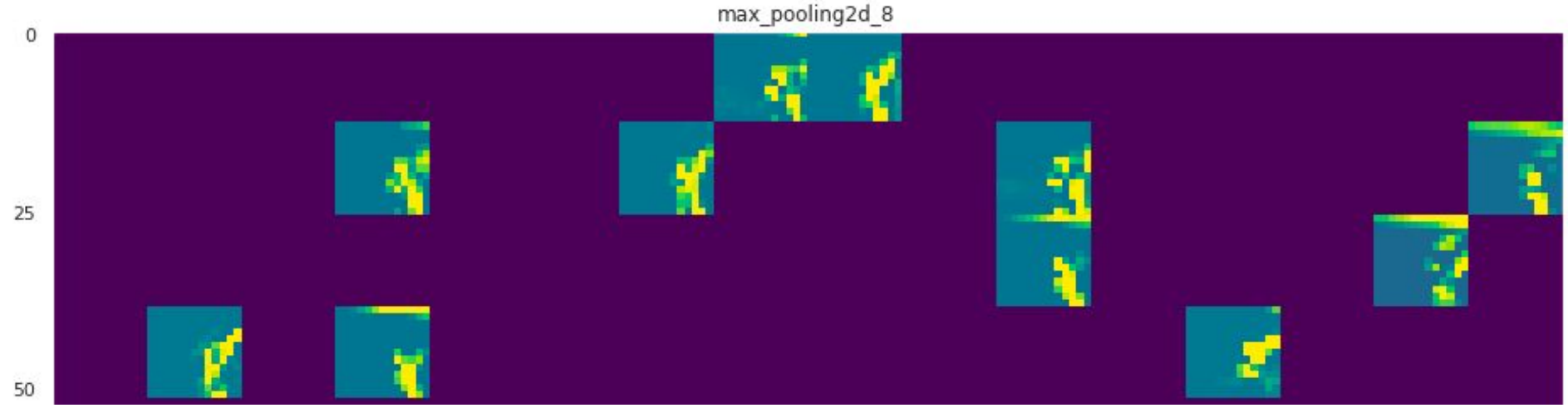
CNN: Visualizing Channels For Best Model

First layer: First convolutional layer

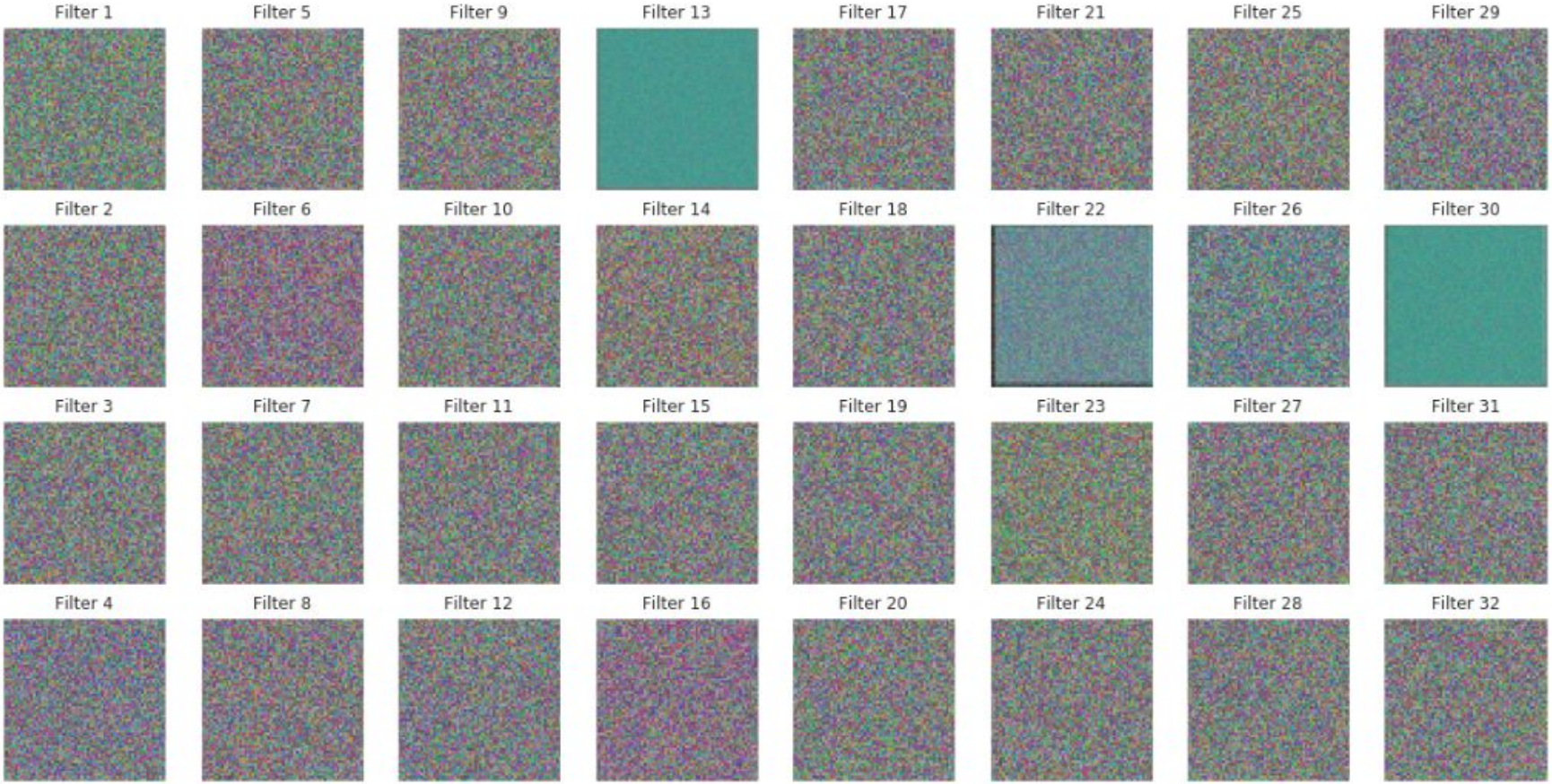


CNN: Visualizing Channels For Best Model

Last layer: Second max pooling layer



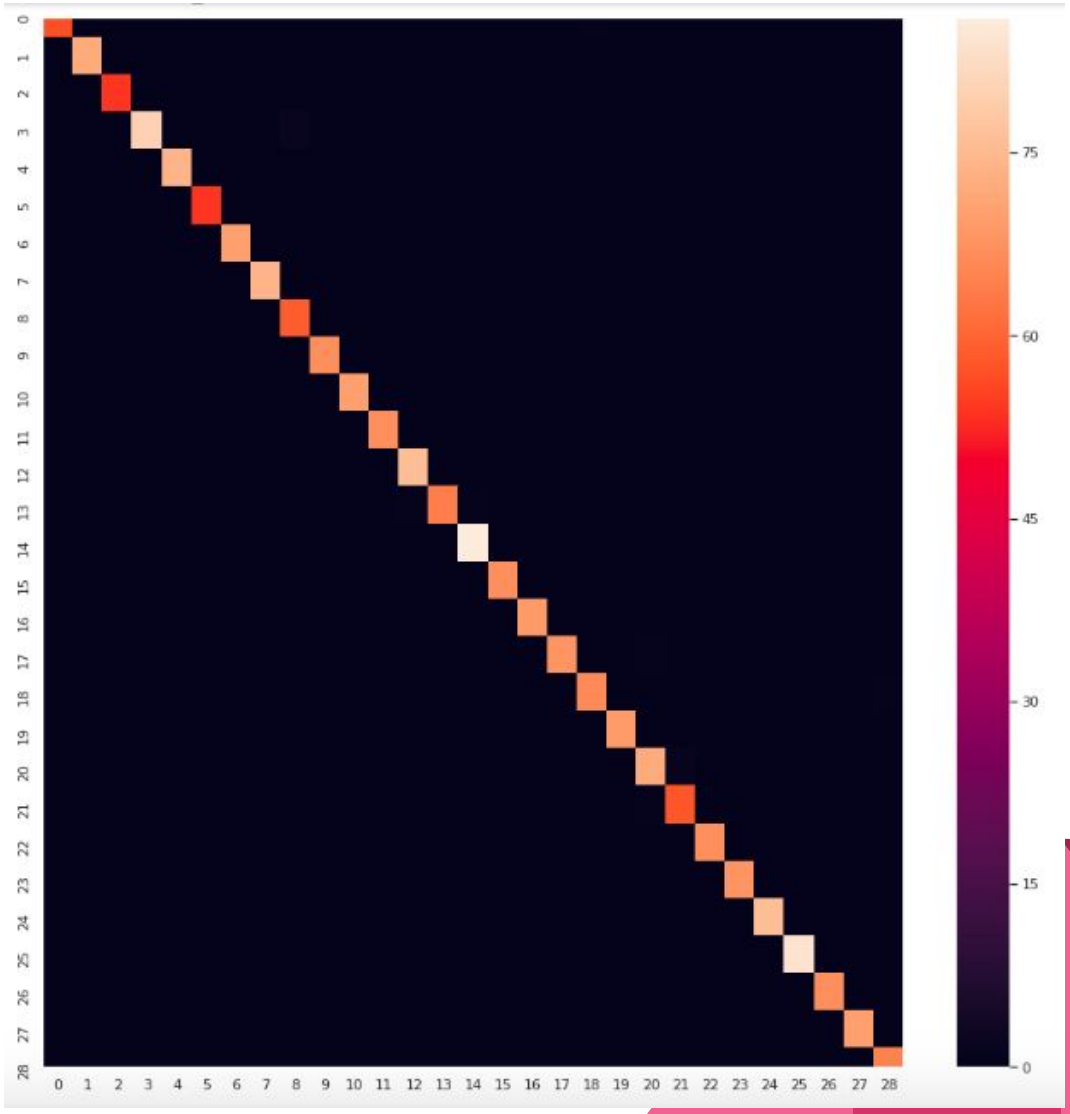
CNN: Visualizing Filters For Best Model



CNN: Confusion Matrix For Best Model

Best model run with test set data:

98.95% accuracy

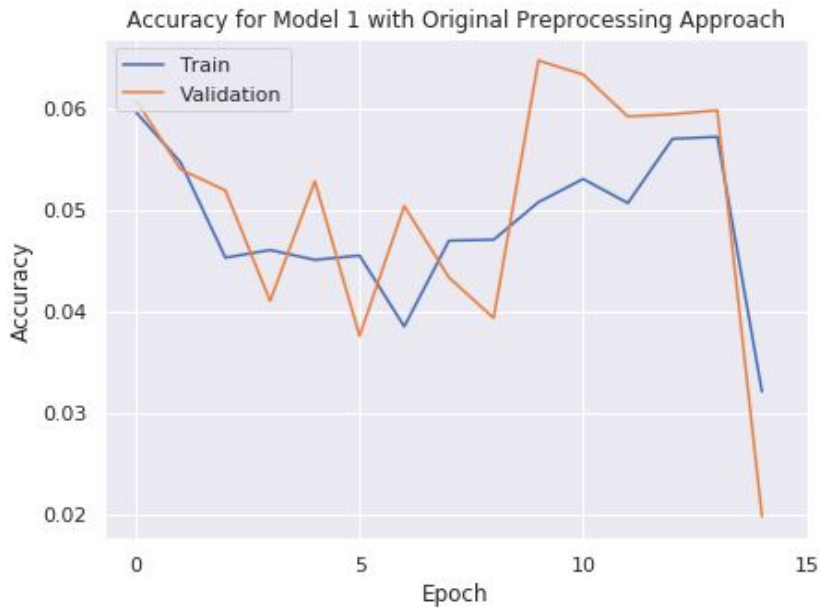


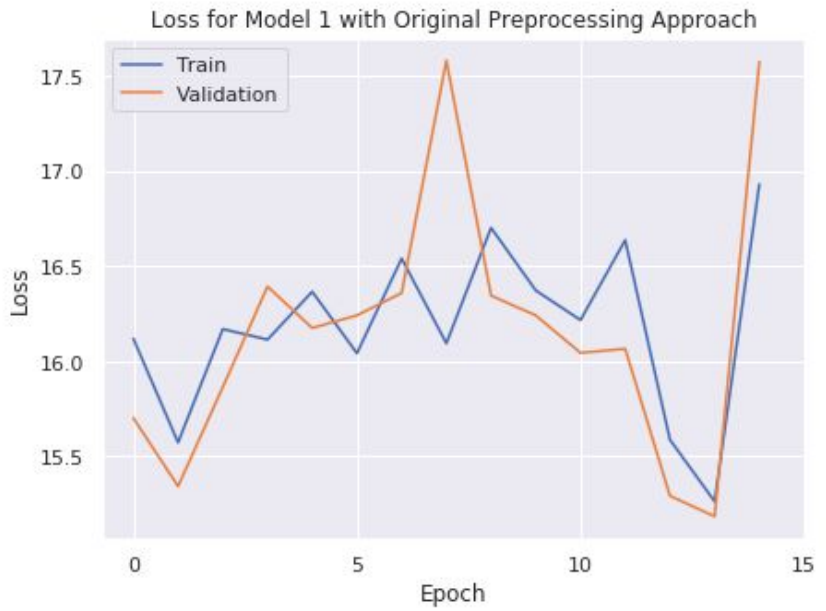
CNN: Exploring Different Preprocessing

We preprocess our data by downscaling, converting to grayscale, and adding Gaussian and Sobel filters.

We run the new preprocessed data on our worst model to see if it improves performance.

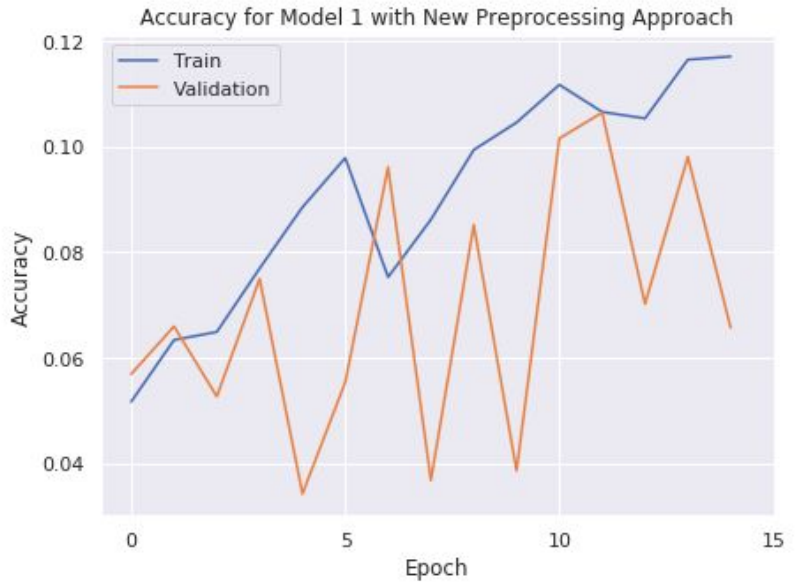
Our worst model has a Dropout 0.3, Kernel Size 3 x 3, and Batch Normalization:

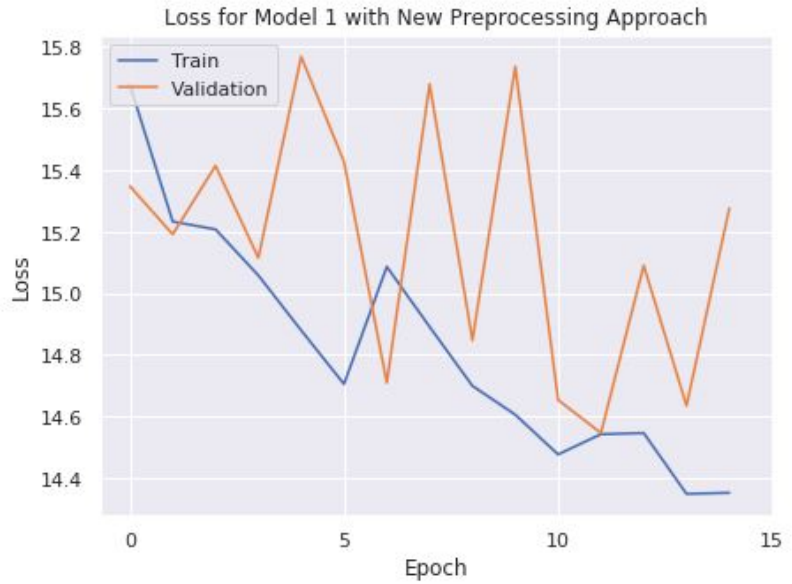




CNN: Exploring Different Preprocessing

After Preprocessing:





Conclusions

ANN

1. Preprocessing helps

   … but there's a limit

2. Adamax is the best optimizer

3. Can achieve 86.5% accuracy

CNN

1. Preprocessing doesn't help

2. RMSProp is the best optimizer

   Batch Norm messes up training

   3>5

3. Can achieve 99% accuracy.

The bottom line: ANNs can be made that are better than expected, but CNNs are clearly better in this domain.

References

1. Admasu, Y F, and K Raimond. “Ethiopian Sign Language Recognition Using Artificial Neural Network.” 2010 10th International Conference on Intelligent Systems Design and Applications, 2010, pp. 995–1000.

2. Ameen, Salem, and Sunil Vadera. “A Convolutional Neural Network to Classify American Sign Language Fingerspelling from Depth and Colour Images.” Expert Systems, vol. 34, no. 3, 2017, p. n/a.
3. Đogic, Sabaheta. “Sign Language Recognition Using Neural Networks.” TEM Journal, vol. 3, no. 4, 2014, pp. 296–301.

4. Islam, Sanzidul, et al. “A Potent Model to Recognize Bangla Sign Language Digits Using Convolutional Neural Network.” Procedia Computer Science, vol. 143, 2018, pp. 611–618.
5. Lim, Kian, et al. “Isolated Sign Language Recognition Using Convolutional Neural Network Hand Modelling and Hand Energy Image.” Multimedia Tools and Applications, vol. 78, no. 14, 2019, pp. 19917–19944.

6. Rao, G. Anantha, et al. “Neural Network Classifier for Continuous Sign Language Recognition with Selfie Video.” Far East Journal of Electronics and Communication, vol. 17, no. 1, 2017, pp. 49-71

7. Singh, Shekhar, et at. “Sign Language to Number by Neural Network.” International Journal of Computer Applications, vol. 40, no. 10, 2012, pp. 38-45

Thank you!