Manescu Emilian Claudiu
Group 507

Generated Image Classification

Documentation

Architectures

The following model architectures were used:

CNN1 model

1. Conv2D with kernel size = 3 and n_filters = 16, ReLU activation
2. MaxPooling2D with size = 2
3. Conv2D with kernel size = 3 and n_filters = 32, ReLU activation
4. MaxPooling2D with size = 2
5. Conv2D with kernel_size = 3 and n_filters = 16, ReLU activation
6. MaxPooling2D with size = 2
7. Flatten layer
8. Dense(64), with ReLU activation + Dropout with probability 0.5
9. Dense(100), with softmax activation

CNN2 model

1. Conv2D with kernel size = 3 and n_filters = 32, ReLU activation + batch normalization
2. MaxPooling2D with size = 2
3. Dropout with probability 0.25
4. Conv2D with kernel size = 3 and n_filters = 64, ReLU activation + batch normalization
5. MaxPooling2D with size = 2
6. Dropout with probability 0.25
7. Conv2D with kernel size = 3 and n_filters = 128, ReLU activation + batch normalization
8. MaxPooling2D with size = 2
9. Dropout with probability 0.25
10. Flatten layer
11. Dense(256), ReLU activation + batch normalization + Dropout with probability 0.25
12. Dense(128), ReLU activation + batch normalization + Dropout with probability 0.25
13. Dense(100), with softmax activation

CNN3 model

1. Conv2D with kernel size = 3 and n_filters = 256, ReLU activation + batch normalization
2. MaxPooling2D with size = 2
3. Dropout with probability 0.25

4. Conv2D with kernel size = 3 and n_filters = 512, ReLU activation + batch normalization
5. MaxPooling2D with size = 2
6. Dropout with probability 0.25
7. Conv2D with kernel size = 3 and n_filters = 512, ReLU activation + batch normalization
8. MaxPooling2D with size = 2
9. Dropout with probability 0.25
10. Conv2D with kernel size = 3 and n_filters = 1024, ReLU activation + batch normalization
11. MaxPooling2D with size = 2
12. Dropout with probability 0.25
13. Flatten layer
14. Dense(1024), custom activation function (ReLU or Swish) + batch normalization
15. Dropout with probability 0.5
16. Dense(1024), custom activation function (ReLU or Swish) + batch normalization
17. Dropout with probability 0.5
18. Dense(512), custom activation function (ReLU or Swish) + batch normalization
19. Dropout with probability 0.5
20. Dense(100), with softmax activation

CNN4 model

1. Conv2D with kernel size = 3 and n_filters = 128, padding = same, ReLU activation
2. Conv2D with kernel size = 3 and n_filters = 128, padding = same, ReLU activation
3. Batch normalization layer
4. MaxPooling2D with size = 2
5. Dropout with probability 0.25
6. Conv2D with kernel size = 3 and n_filters = 256, padding = same, ReLU activation
7. Conv2D with kernel size = 3 and n_filters = 256, padding = same, ReLU activation
8. Batch normalization layer
9. MaxPooling2D with size = 2
10. Dropout with probability 0.25
11. Conv2D with kernel size = 3 and n_filters = 1024, padding = same, ReLU activation
12. Conv2D with kernel size = 3 and n_filters = 1024, padding = same, ReLU activation
13. Batch normalization layer
14. MaxPooling2D with size = 2
15. Dropout with probability 0.25
16. Flatten layer
17. Dense(1024), ReLU activation + batch normalization + Dropout with probability 0.25
18. Dense(1024), ReLU activation + batch normalization + Dropout with probability 0.25
19. Dense(1024), ReLU activation + batch normalization + Dropout with probability 0.25
20. Dense(100) with softmax activation

ResNet model

The ResNet model consists of:

- Identity block, in which the residual input and the output of the block have the same number of filters.
- Residual block, in which the residual input and the output of the block have a different number of filters. This means that the residual input must go through a convolutional layer to make sure that the number of filters is the same.

Both types of blocks consist of three convolutional layers and a skip connection. The convolutional layers are structured as follows:

1. Conv2D with kernel size 1x1 + batch normalization + activation (in this order)
2. Conv2D with kernel size 3x3 and same padding + batch normalization + activation
3. Conv2D with kernel size 1x1 + batch normalization

The model architecture is as follows:

1. ZeroPadding2D with padding = 3
2. Conv2D with kernel size = 7, 128 filters, stride = 2 + batch normalization + activation
3. Max pooling with size = 3 and stride = 2
4. First layer
    a. Residual block with number of filters = [128, 128, 256] and stride = 1
    b. Identify block with number of filters = [128, 128, 256] x 3
2. Second layer
    a. Residual block with number of filters = [128, 128, 512] and stride = 2
    b. Identity block with number of filters = [128, 128, 512] x 3
3. Third layer
    a. Residual block with number of filters = [256, 256, 1024] and stride = 2
    b. Identity block with number of filters = [256, 256, 1024] x 5
4. Fourth layer
    a. Residual block with number of filters = [512, 512, 2048] and stride = 2
    b. Identity block with number of filters = [512, 512, 2048] x 2
5. Average pooling 2D with pool size = 2
6. Output layer
    a. Flatten layer
    b. Dense(512), custom activation, custom kernel initializer + Batch norm + Dropout(0.5)
    c. Dense(512), custom activation, custom kernel initializer + Batch norm + Dropout(0.5)
    d. Dense(100), softmax activation, custom kernel initializer

Each residual and identity block has a custom activation function (ReLU or Swish), a custom kernel initializer (Glorot uniform or He normal), and stride equal to 1 or 2.

Preprocessing

Data preprocessing consists of the following:

1. Reading the image labels from the CSV files
2. Reading the images from the train, validation and test folders using OpenCV
3. Resizing the images to size 64x64
4. Converting the labels to categorical form (one-hot encoding of the labels)
5. The images are fed into the models using an ImageDataGenerator (from Keras), which ensures data augmentation and prevents overfitting. The following transformations are performed:
    a. Rotation range = 40, which specifies the maximum degree of rotation applied to the image
    b. Width shift range = 0.2, which specifies how much of the image is shifted to the left or right
    c. Height shift range = 0.2, which specifies how much of the images is shifted up or down
    d. Divide by 255 to ensure that pixel values are in the range [0, 1]
    e. Shear range = 0.2, which specifies how much of the image is stretched (one axis is fixed and the image is stretched along the other)
    f. Zoom range = 0.2, which specifies how much the image is zoomed in or out
    g. Horizontal flipping
    h. Nearest fill mode, where a missing pixel is replaced with the closest pixel
6. The test and validation image data generator only applies a normalization to the images (dividing by 255). No other transformations are applied.

The image data generator is used to generate a different generator for the training, validation and test set. The values are the same used for each of the models trained, with no variation.

Hyperparameters

The hyperparameter configurations tried for each model architecture are described below. I will also specify the number of epochs, the optimizer used, learning rates, and so on. I only used the categorical cross entropy loss and the accuracy metric. Every model fitting involves a checkpoint which saves the configuration of hyperparameters with the highest accuracy on the validation set.

CNN1

Given the simplicity of this model architecture (the first one that I tried), I only tested using two different optimizers: Adam and SGD, with the default values from Keras. I obtained values higher than the baseline for the Adam optimizer, but not high to warrant more tuning. I instead turned towards more complex architectures.

CNN2

For this model architecture, which is rather simplistic, I wanted to experiment with fixed and variable learning rates. Therefore, I trained two models to compare their results: one with a fixed learning rate (Adam with learning rate = 0.001), and one with a cosine decay scheduler with restarts. For the latter, I used an initial learning rate of 0.01 and a final learning rate value of 1e-5. The learning rate decays for a third of the total number of epochs, and then goes back to the initial value.

CNN3

For this architecture, I experimented more with the parameters of the learning rate scheduler and with the activation functions. Therefore, I tried two different activations (ReLU and Swish), and two different optimizers (Adam and SGD with Nesterov momentum, both using a weight decay of 0.001). I also added 10 warmup epochs to the scheduler, in which the learning rate goes from 0 to the initial value (in my experiments, this was equal to 0.01 or 0.001).

Moreover, for the model Adam and cosine decay scheduler, I tried initializing the model with the weight values obtained after 200 epochs of training. This was my form of “fine-tuning”, since I assumed that retraining from that starting point would give me better results. This proved to be true (all metrics improved, but only by about 1%).

CNN4

This model architecture was more complex, had more parameters, but was not as efficient as the CNN3 architecture. Therefore, I decided to not perform hyperparameter tuning on this one, since the training time was too long and the performance of the CNN3 models was good enough. This model was trained using Adam optimizer with cosine decay scheduler (initial learning rate = 0.005 and final learning rate = 1e-5). I also added a weight decay of 0.001 in order to reduce overfitting, but the results were still not good enough.

ResNet

For the residual network architectures, I tried varying the following:

- The activation function (ReLu or Swish)
- The convolutional kernel initializer (He normal vs Glorot uniform)
- The optimizer (Adam vs SGD with Nesterov momentum), both using a cosine decay scheduler
- Label smoothing (only tried values 0, 0.1, and 0.2). In my experiments, 0.1 seemed to bring the best performance.

For these models, I added 10 warmup epochs for the learning rate as well (increasing it from 0 to 0.01). The final learning rate was 1e-5, the same as for the CNN3 models. All ResNet models have a weight decay value of 0.001.

Ensemble models

For the ensemble models, I tried two different model sets:

1. First, using all ResNet models in the table below, since they all had good relative performance, plus the following CNN3 models: CNN3 + Cosine Decay + Adam (1st and 2nd iterations), and CNN3 + Cosine Decay + Adam + Swish activation
2. Second, only using the ResNet model configuration from the table below

These two ensembles are the one used for my predictions for the Kaggle competition.

Results

The table below shows the results obtained on the validation set:

Model/Metric	Epochs	Accuracy	Precision	Recall	F1
CNN1 + Adam (lr = 0.001)	60	0.351	0.378	0.351	0.319
CNN1 + SGD (lr = 0.01)	60	0.259	0.263	0.259	0.215
CNN2 + Adam + lr = 0.001	60	0.557	0.616	0.557	0.539
CNN2 + Adam + Cosine Restarts	60	0.573	0.632	0.572	0.561
CNN3 + Adam + lr = 0.001	200	0.697	0.728	0.697	0.697
CNN3 + Cosine Decay + Adam, 1st iteration	200	0.714	0.737	0.713	0.716
CNN3 + Cosine Decay + Adam, 2nd iteration	200	0.722	0.745	0.721	0.724
CNN3 + Cosine Decay + Adam + Swish	200	0.736	0.756	0.736	0.738
CNN3 + Cosine Decay + SGD + ReLU	200	0.707	0.742	0.707	0.708
CNN4 + Cosine Decay + Adam	100	0.669	0.690	0.670	0.669
ResNet + Glorot + Adam	200	0.742	0.752	0.742	0.740
ResNet + Glorot + SGD	200	0.721	0.739	0.720	0.722
ResNet + He + Adam	200	0.739	0.759	0.738	0.740
ResNet + Glorot + Adam + Smoothing = 0.1	200	0.766	0.777	0.766	0.763
ResNet + Glorot + Adam + Smoothing = 0.2	200	0.756	0.766	0.757	0.753
ResNet + Glorot + Adam + Swish	200	0.744	0.766	0.745	0.745
Ensemble 1 (ResNet + CNN3 subset)	-	**0.803**	**0.813**	**0.803**	**0.802**
Ensemble 2 (only ResNet)	-	0.797	0.807	0.797	0.795

As can be seen in the table, from the single models, the ResNet models have obtained the best results. Notice the big improvement in all metrics when using label smoothing as compared to

the model without smoothing (**ResNet + Glorot + Adam** model has a 0.74 F1 score, while the same model trained with a label smoothing value of 0.1 has 0.763, a 0.2 improvement).

Also important to notice that a cosine decay scheduler is paramount to improving performance, as can be seen by comparing the simple CNN3 model, which was trained with a fixed training rate, and the same model trained with the learning rate scheduler with warmup.

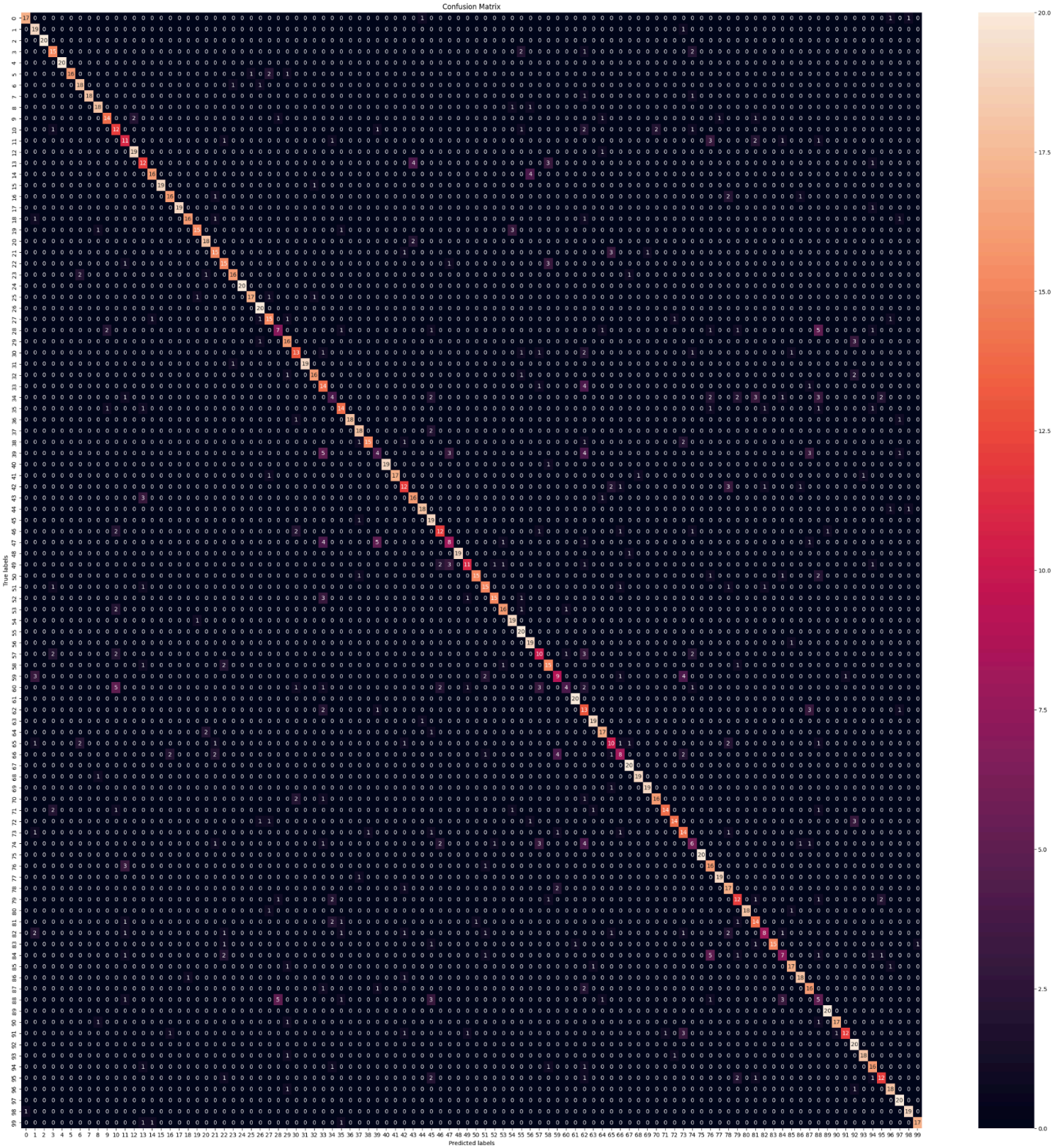
Moreover, by comparing the **ResNet + Glorot + Adam** (which uses the ReLU activation function) and **ResNet + Glorot + Adam + Swish**, we can see that changing the activation function brings a slight improvement for all metrics. However, I noticed that the training time increased considerably, so I was not able to test using both label smoothing and Swish activation, due to time constraints.

Also, notice that using ensembles increased the performance substantially (4% accuracy increase over the best single model used). Also good to notice that adding more models to the ensemble is more beneficial to the overall performance.

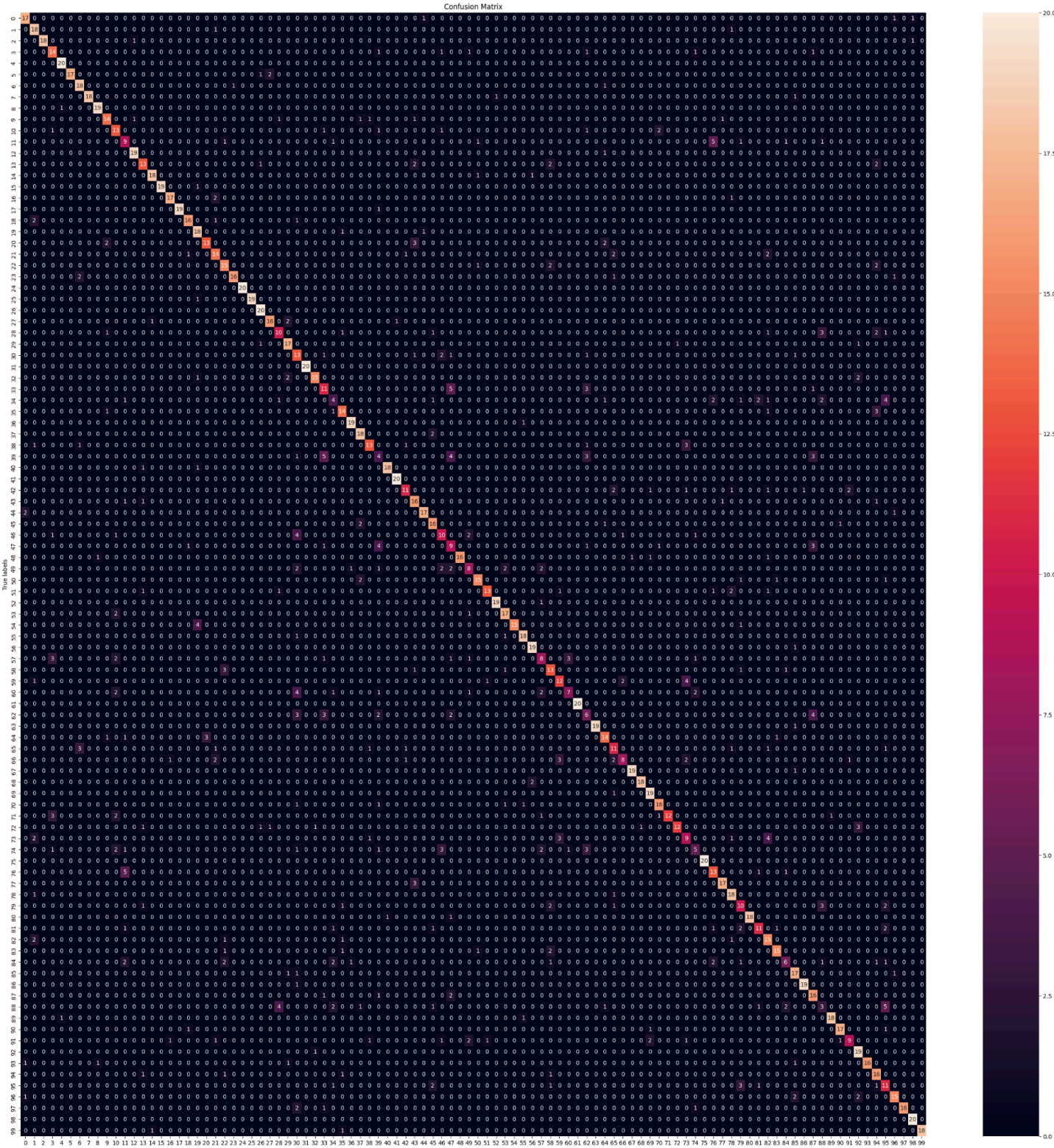
Confusion matrices

Below are some confusion matrices for some of the best performing models that I tried:

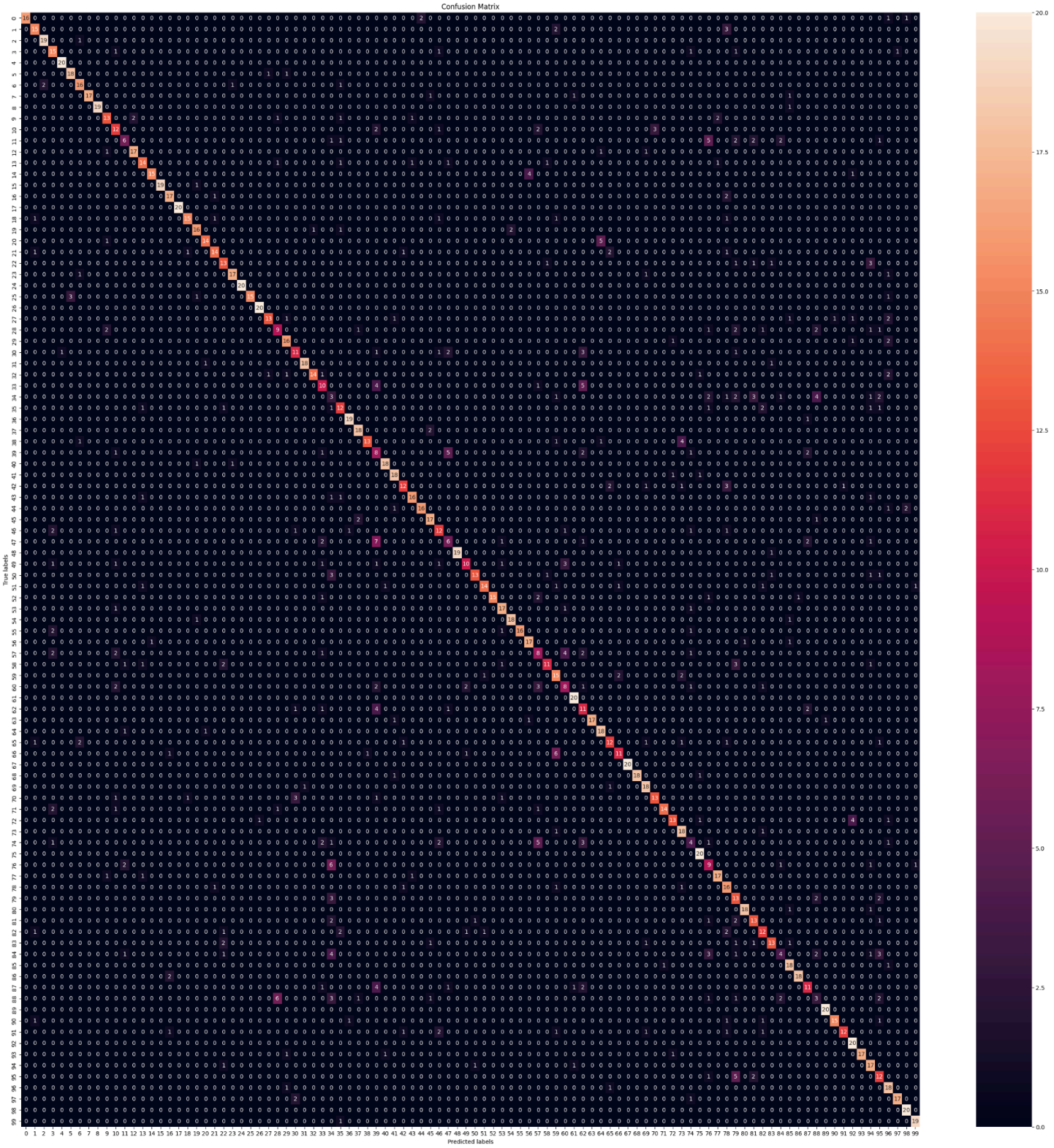
- **ResNet + Glorot + Adam + Smoothing = 0.0**



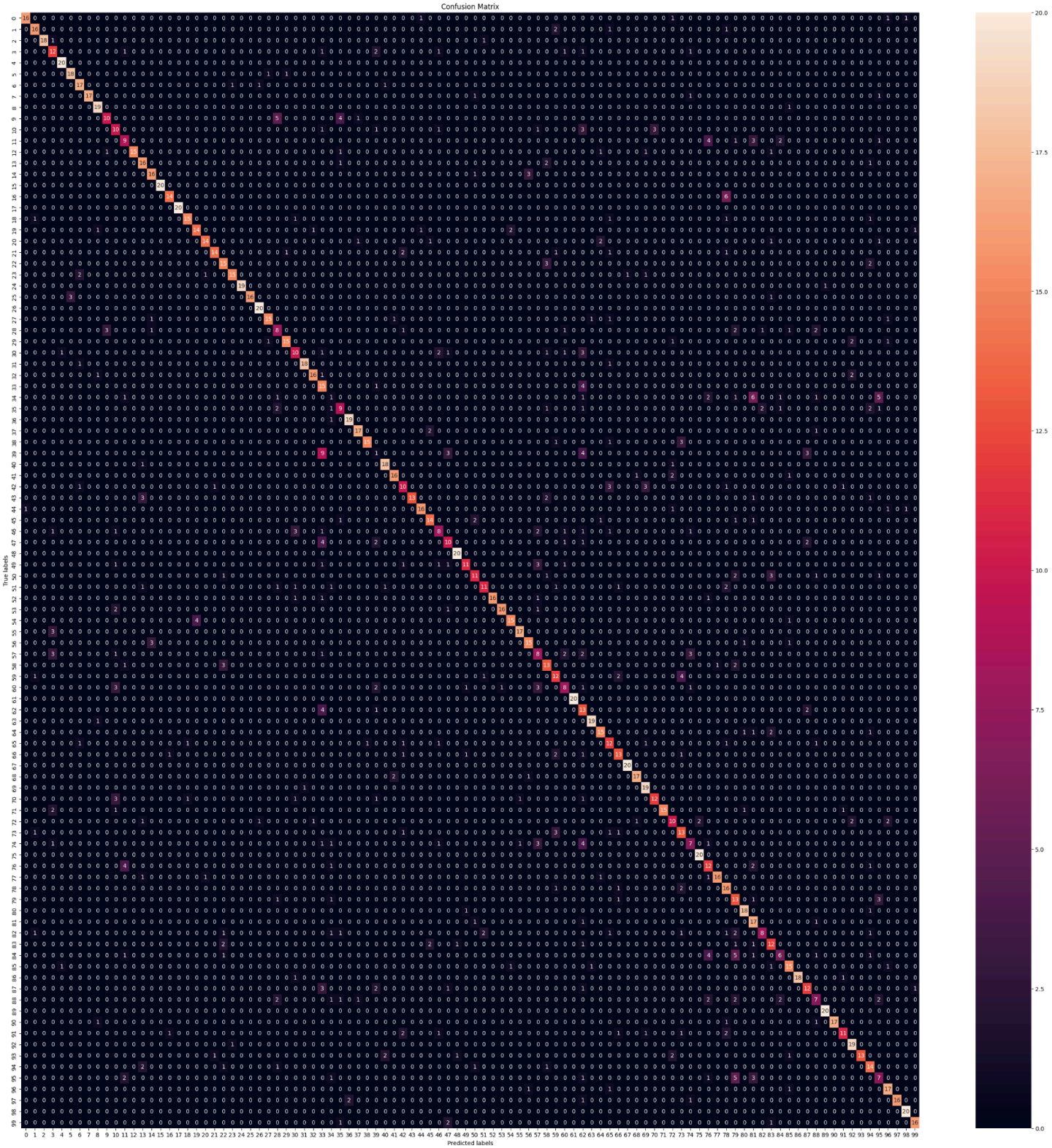
- **ResNet + Glorot + Adam:**



- **CNN3 + Cosine Decay + Adam + Swish:**



- **CNN3 + Cosine Decay + Adam, 2nd iteration:**



- **Ensemble 1:**