DL4CV - Assignment 2

Chirag - 12140520

October 9, 2024

1 Objective

Explore a new CNN design for personalized handwriting recognition. The dataset to be used is EMNIST. We will focus on: EMNIST ByClass: 814,255 characters. 62 unbalanced classes to solve the following tasks:

1. Custom Loss function Design and evaluation: Extension of cross entropy loss to add a penalty for all classes, not just the true class. Think of how you will give the penalty, when you will give the penalty, what will you do if some false class is predicted with very low probability.
2. Regularization Loss: Add a regularizer that promotes orthogonal filters at each layer.
3. Explore Focal Loss and SMOTE and other techniques to handle class imbalance in the dataset.
4. Neural Architecture Search: Since the input is only 28x28 and there are 62 classes, we can work with a smaller network. Our goal is to beat the performance [Accuracy, Precision, Recall, F1 Score] of ResNet with much smaller number of parameters. Report both micro-average and macro-average performance.
5. Write a report with insights obtained and well-documented code.

2 Approach

Architectures

For the first 3 components, I've used ResNet-18 architecture pre-trained on the ImageNet dataset. Out of the 8.4M parameters, about 130k are trainable, and the rest are frozen. This insures that the basic feature extraction of the ResNet architecture is reused instead of training it from scratch. The images are 28x28 grayscale images, so they are resized to 224x224 using Bilinear Interpolation, normalized, and the same channel is copied thrice because of the requirements of the architecture.

For 4th component (Neural Architecture Search), I've built the models following the ideas of VGG, ResNet, DenseNet and MobileNet. The resulting architectures have parameters ranging from $\tilde{2}0$k to 900k, resulting in a broad search space for the new models. The images should only be normalized (no need to reshape since the models require 28x28 grayscale images only).

Custom Loss Function

As specified in the problem, the loss function gives a penalty to all the classes, not just the true class. The way it is given is by:

$$\text{Loss}(p, y) = -\log(p_y) - \lambda \left(\sum_{i \neq y} \log(1 - p_i) \right)$$

where, p_y be the predicted probability of the true class y, p_i be the predicted probability of class i (for all classes), λ be the penalty weight (a hyperparameter).

Orthogonality Regularizer

Following the idea of the paper "Orthogonal Convolutional Neural Networks", the regularization loss is added to the loss function of the network. Additionally, L2 regularization was also tested once with Cross-entropy loss to compare the filters.

Handling Class Imbalance

To handle class imbalance, two types of changes were made: 1. Replacing Cross-entropy loss with Focal loss and 2. Oversampling the dataset using techniques like SMOTE and Borderline SMOTE. Since Borderline SMOTE samples more from the data-points that are harder to classify than the easier ones, experiments are also performed using this along with SMOTE.

3 Insights Gained

The experiments are done in 6 different settings for the first 3 components:

1. Cross-entropy loss + L2 Regularization
2. Custom Cross-entropy loss without Regularization
3. Softmax Focal loss without Regularization
4. Custom Cross-entropy loss + Orthogonal Regularization
5. Custom Cross-entropy loss with SMOTE
6. Custom Cross-entropy loss with Borderline SMOTE

Performance Comparison

Among all the six settings, setting 2 gave the highest test accuracy of $\tilde{0}.88$, followed by settings 5, 6, 1 and 3, with $\tilde{0}.84$ accuracy and least by setting 4, with $\tilde{0}.79$. The low accuracy of setting 4 explains the restrictiveness of Orthogonality on the filters. The Custom Cross-entropy loss function seems to work really well without any regularization.

Filter Comparison

The main purpose of using Orthogonal Regularization is to ensure the orthogonality of the filters. As you can see in the image below, the filters are indeed orthogonal in setting 4.

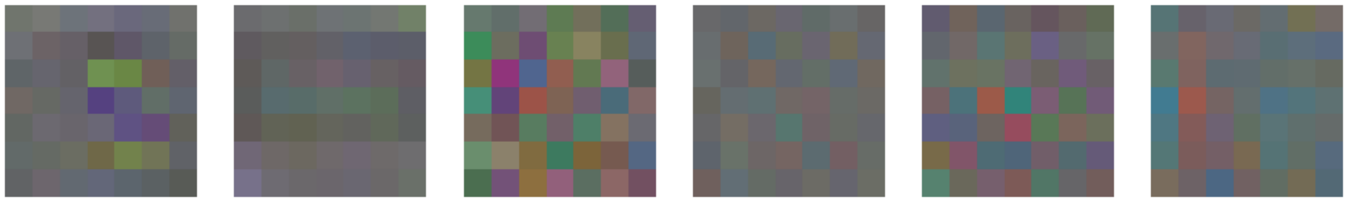

Figure 1: First 6 trainable filters of the network in Setting 4

As for setting 1, because of L2 regularization, all the values inside a filter are of nearly the same value with low magnitude, as shown below:

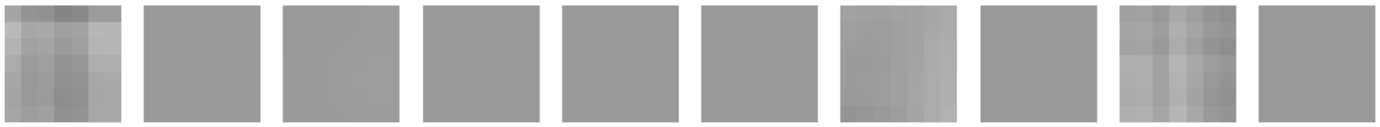

Figure 2: First 10 trainable filters of the network in Setting 1

In settings 2, 3, 5 and 6, the filters look very similar to each other but strikingly different from the above 2 settings.

Figure 3: First 6 trainable filters of the network in Setting 3

Class Imbalance Comparison

For class imbalance comparison, setting 6 seems to work better than other settings. This explains why balancing the data might help in reducing class imbalance.

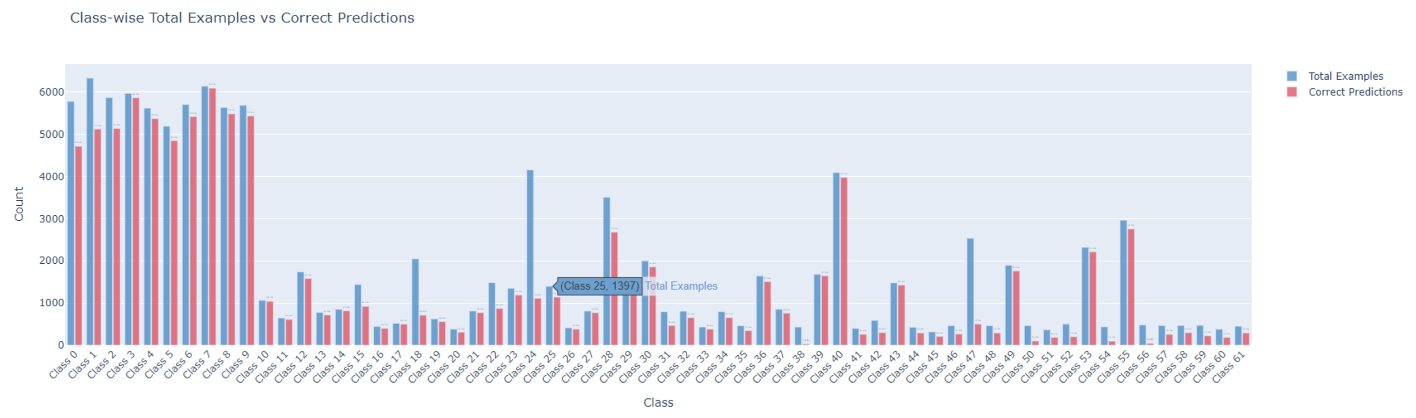

Figure 4: Setting 6

Setting 3, on the other hand, doesn't work well for the classes with very few examples, though it's only a qualitative comparison. A much better comparison can be done quantitatively in future.

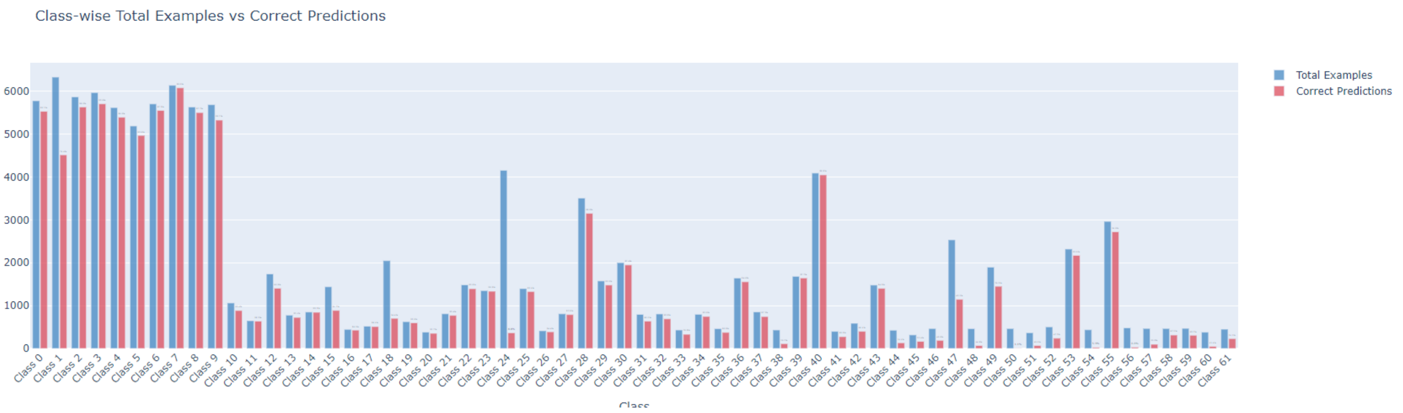

Figure 5: Setting 3

Setting 4, having the least overall accuracy, works worst in the class imbalance scenario as well.

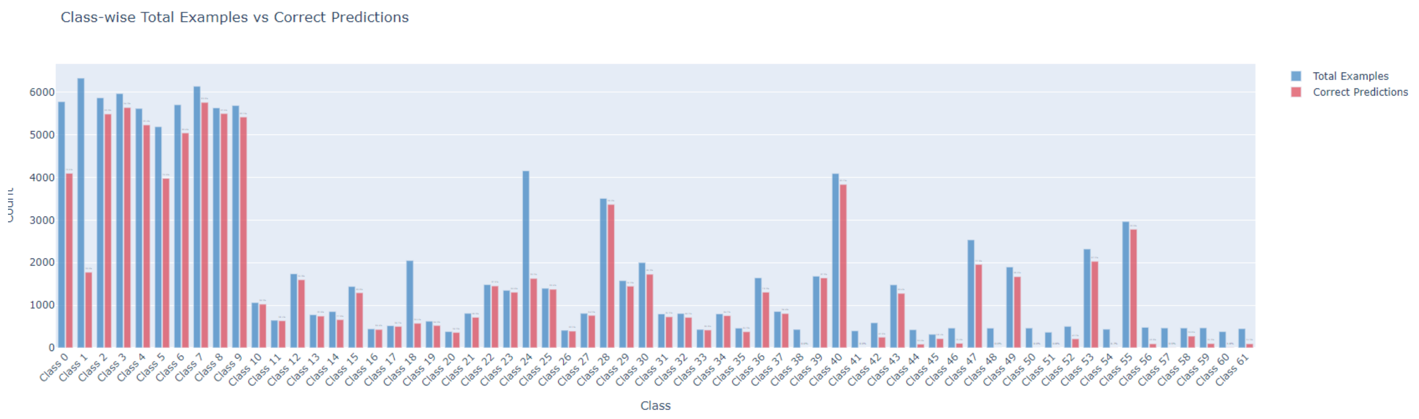

Figure 6: Setting 4

4 Neural Architecture Search

As mentioned above, 4 types of architectures are designed (inspired by VGG, Resnet, DenseNet and MobileNet. Out of these architectures, architecture derived from DenseNet with $\tilde{4}60$k parameters and trained on the original training dataset seems to slightly outperform the best-performer setting i.e., setting 2 in macro average F1-score. Here are the numbers for reference:

- Setting 2
  - Accuracy: 0.88
  - Precision (Micro): 0.88, Recall (Micro): 0.88, F1-Score (Micro): 0.88
  - Precision (Macro): 0.82, Recall (Macro): 0.76, F1-Score (Macro): 0.76
- DenseNet-inspired custom architecture
  - Accuracy: 0.88
  - Precision (Micro): 0.88, Recall (Micro): 0.88, F1-Score (Micro): 0.88
  - Precision (Macro): 0.81, Recall (Macro): 0.76, F1-Score (Macro): 0.77

5 Conclusion

The above insights give a clear indication towards the effectiveness of Custom Cross-entropy loss over Cross-entropy loss in terms of performance, of Borderline SMOTE over Focal Loss in terms of class imbalance and of Orthogonal Regularization in ensuring the orthogonality of filters. Because of the lack of computational resources, some more experiments that could be performed, were not done (say by testing Orthogonal Regularization with Focal loss etc.).

6 End Note

This work implements and analyses the behaviour of various loss functions, architectures, regularizers and oversampling techniques. It, however, doesn't fully explain why such behaviour is observed in these settings. All the experiments are done in similar settings (same learning rate and hyperparameters). The work can be used as a starting point to further extend this analysis of CNNs and related techniques.