Topic : A CNN for Image Classification using Transfer Learning.

Sandeep Kumar

Introduction

This project focuses on handwritten letter and character recognition using a pre-trained Convolutional Neural Network (CNN). The model is fine-tuned to achieve high accuracy on the given dataset. To enhance generalization and mitigate overfitting, various data augmentation techniques have been applied.

Dataset

For this project, I am using the **EMNIST (Extended MNIST) dataset**, a collection of handwritten uppercase and lowercase characters along with digits. It is derived from the NIST Special Database 19 and reformatted into a **28×28 grayscale image** structure, similar to the MNIST dataset.

Dataset Sizes:

- Training set size: 697,932 images
- Test set size: 116,323 images

Model

I am using **EfficientNet-B3** as the pre-trained model. EfficientNet is a family of models developed by Google that optimizes depth, width, and resolution to achieve a balance between computational efficiency and accuracy. The EfficientNet-B3 variant has approximately **12 million parameters** and a **300×300 input resolution**.

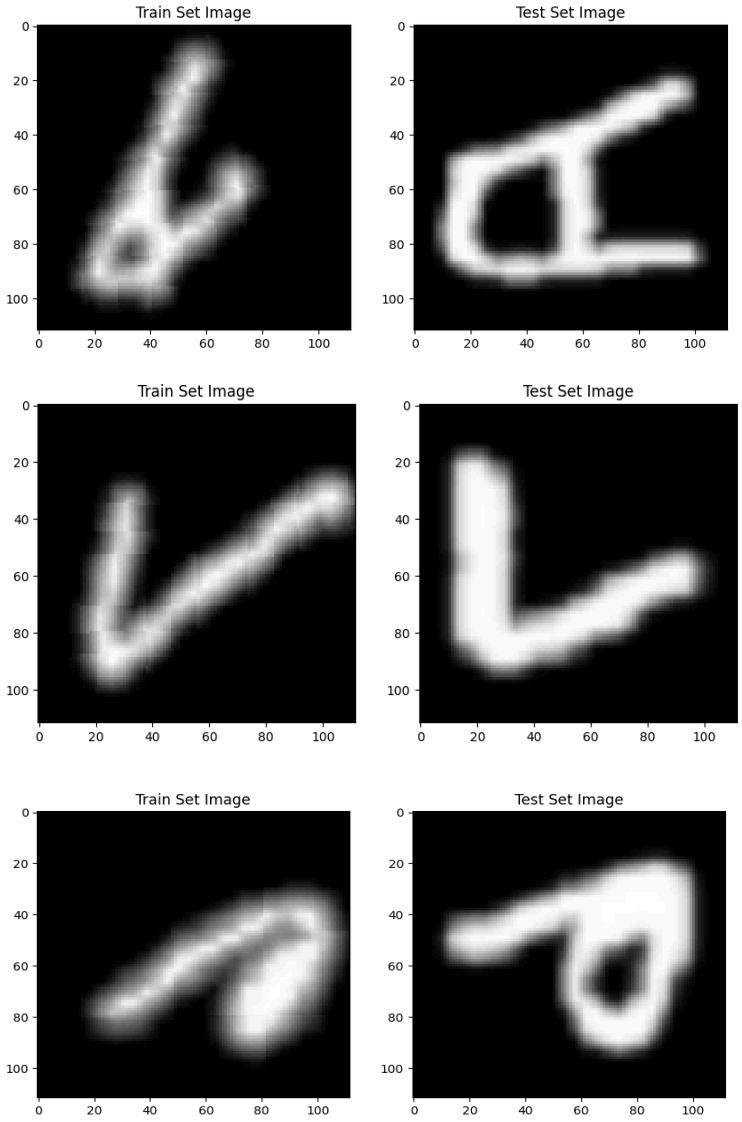
Modifications:

- The first convolutional layer was modified to accept **1-channel grayscale images** instead of the default 3-channel RGB input.
- The final classification layer was replaced to output **62 classes** (corresponding to the characters and digits in EMNIST).

Input Preprocessing

The original dataset consists of **28×28 pixel grayscale images**. To improve feature extraction, I **resized the images to 112×112** resolution before feeding them into the model.

Since the **"byclass" split** of EMNIST is highly unbalanced, I manually computed **class weights** to ensure balanced training. Additionally, I recalculated the dataset's **mean and standard deviation** after resizing, as these values changed due to the transformation.



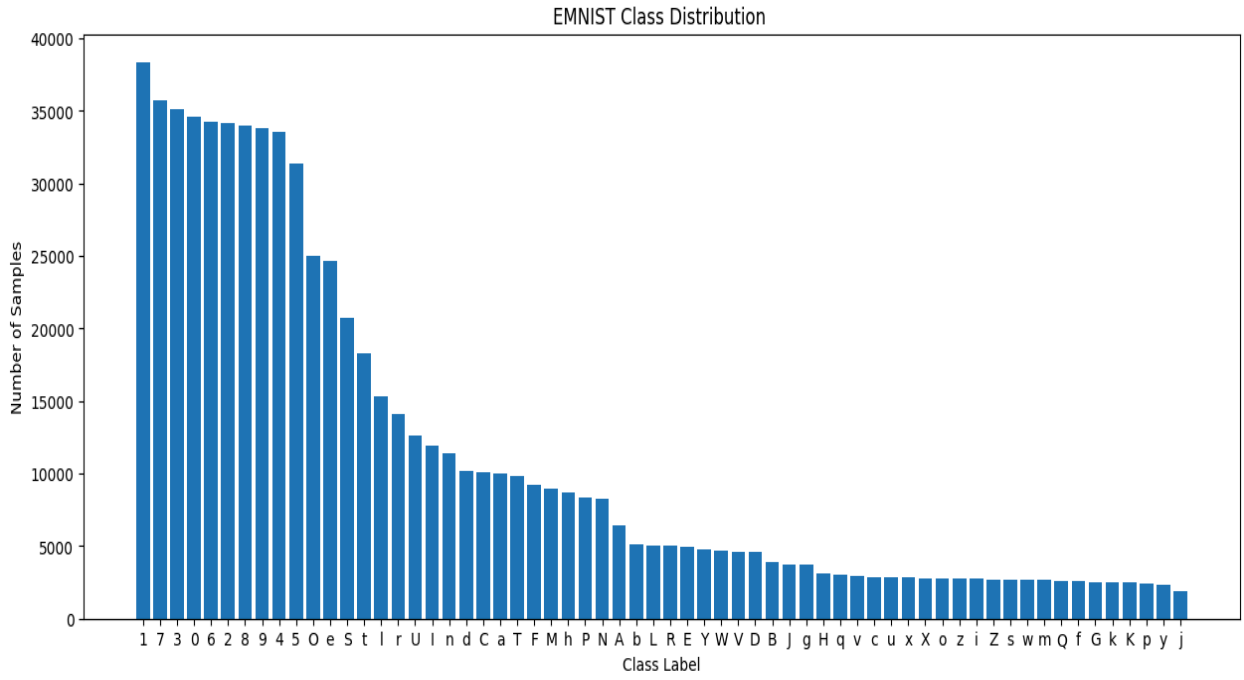
Approach

1.**Learning more about the dataset**

What I mean by learning more about the dataset is that I realized achieving high accuracy on the original EMNIST dataset, especially with the "byclass" split, is quite difficult. As I explored the dataset further and reviewed research papers, I found that it has several issues that make classification challenging.

One major issue is that many images contain errors and are pre-augmented by 90 degrees, which affects their natural orientation. Additionally, some letters look very similar, increasing the chances of misclassification. These challenges meant that standard augmentation techniques would not be effective, so I had to adjust my approach accordingly.

Another challenge I discovered is the dataset's imbalance. The most frequent label appears **33,374** times, while the least frequent label appears only **1,896** times, which could lead to biased model performance. To address this, I wrote a custom function to compute class weights ensuring that less frequent classes were not overshadowed by more common ones.



2.Computation

When I resized all the images, I encountered a significant computational challenge. The dataset size expanded dramatically from **562MB to 39.5GB**, leading to a **DataLoader bottleneck** that slowed down training. Managing this increased memory demand became a critical issue.

To address this, I needed to determine the optimal image resolution that provided the best balance between accuracy and training efficiency. I experimented with different image sizes by testing a **0.2% subset** of the training dataset along with the **full dataset** to measure both performance and training time.

Through these experiments, I found that a resolution of **112×112** offered the best trade-off between computational efficiency and model accuracy. Although this resolution still resulted in a **39.5GB dataset**, it was the most effective choice given the constraints.

3. Finding a Model that Fits

At first, I experimented with smaller models, but I found that they were quite slow. Moving to a larger model, however, wasn't as computationally expensive as I initially thought. Still, I had to ensure that the model didn't have too many parameters, so options like **VGG16, ResNet, EfficientNet-B4, and EfficientNet-B5** were ruled out.

After testing different architectures, I settled on **EfficientNet-B2 and EfficientNet-B3**, as they provided the best balance between performance and efficiency. These models were not too large but still powerful enough to deliver high accuracy.

I also experimented with **pre-trained weights** by changing the input size to **3 channels** and resizing the images to match the standard input size of these models. However, this approach did not yield significant improvements, so I decided to stick with grayscale inputs while optimizing other aspects of training.

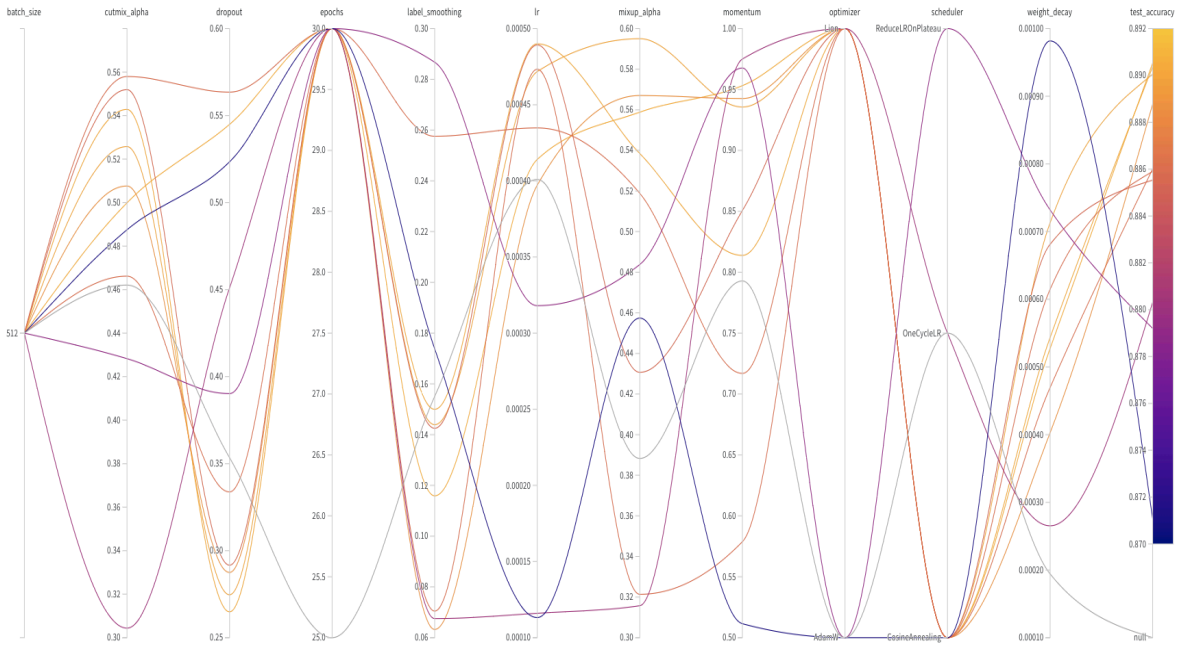
4. How to Find the Best Optimizer, Scheduler, and Hyperparameters

While discussing this project with a friend, he suggested using **Weights & Biases (WandB)** to track and experiment with different hyperparameters. Taking his advice, I started researching online and reading papers to identify the best options.

I narrowed it down to **four optimizers**: **Adam, AdamW, Lion, and SGD with Lookahead**. I also tested different learning rate schedulers to determine which combination worked best. To evaluate them efficiently, I ran experiments using **20% of the training dataset** and **100% of the test dataset**, training for **20 epochs** and repeating the process **10 times** to find the best hyperparameters and optimizer.

I kept refining this process, running multiple experiments and adjusting parameters to fully understand their impact. This iterative approach helped me fine-tune my model and select the best combination of optimizer, scheduler, and hyperparameters for the task.

This is a picture of one of many runs:



5.Training

During training, I noticed that the model was overfitting. To address this, I initially applied more data augmentation and added dropout. While this helped to some extent, it did not reduce overfitting as much as I had hoped.

To further tackle the issue, I incorporated **CutMix augmentation**, which significantly reduced overfitting. After implementing CutMix, I found that dropout and some other augmentations were no longer necessary. Removing them further improved the model's performance, leading to better generalization and overall accuracy.

Results

The benchmark for the EMNIST dataset using the "**byclass**" split is **88.43% validation accuracy**—this was the highest accuracy I could find. However, when I checked the benchmarks provided by the dataset publishers, their reported accuracy was significantly lower, falling within the **70% range**.

TABLE III
SUMMARY OF THE RESULTS FOR THE LINEAR AND OPIUM-BASED CLASSIFIERS ON THE EMNIST DATASET.

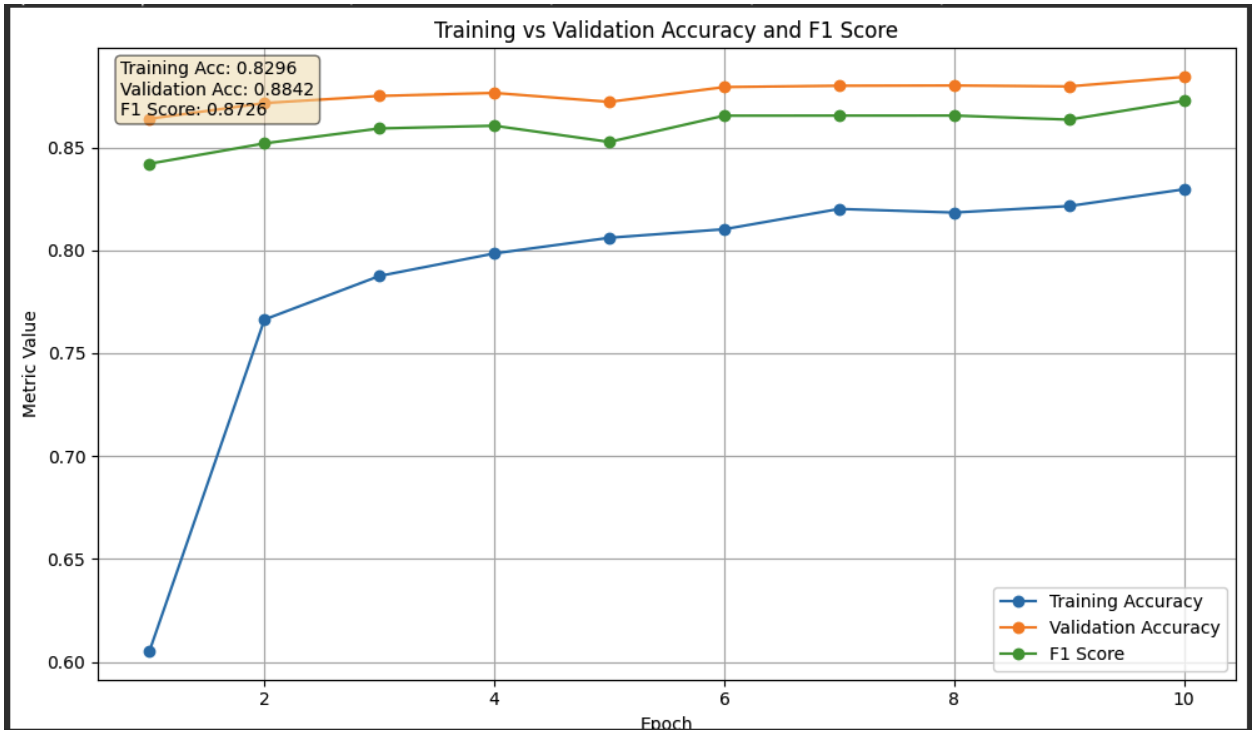
	Linear Classifier	OPIUM Classifier
Balanced	50.93%	78.02% ± 0.92%
By Merge	50.51%	72.57% ± 1.18%
By Class	51.80%	69.71% ± 1.47%
Letters	55.78%	85.15% ± 0.12%
EMNIST MNIST	85.11%	96.22% ± 0.14%
Digits	84.70%	95.90% ± 0.40%

I achieved **88.42% validation accuracy** using the **Lion optimizer** along with **CosineAnnealingWarmRestarts** and **SequentialLR** as the learning rate scheduler. For the model, I used **EfficientNet-B2** with **MixUp and CutMix** for augmentation. To handle class imbalance, I employed **KLDivLoss with computed class weights** as the loss function.

I experimented with training for more epochs in an attempt to push the accuracy higher, but after **5 epochs**, the model started oscillating, indicating instability. This suggests that beyond a certain point, the model struggles to converge properly, likely due to sensitivity in learning rate adjustments or augmentation variations.

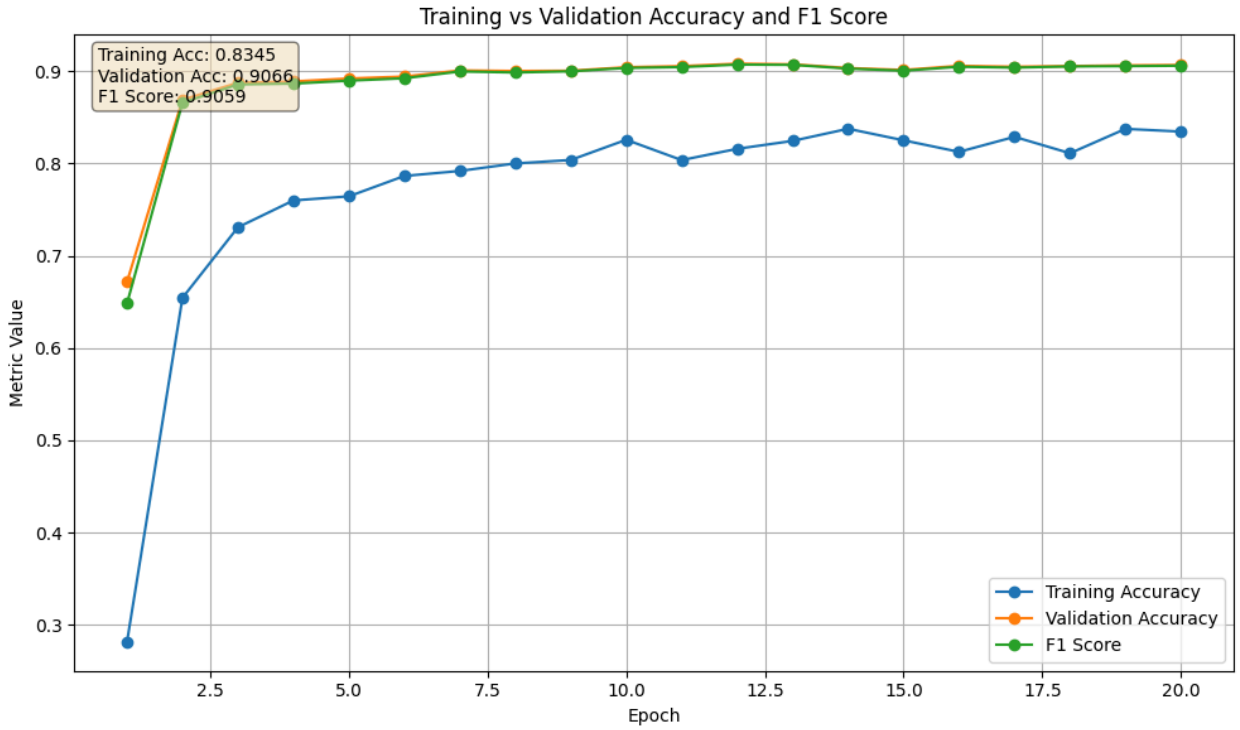
Learning_rate =0.0003

Weight_decay = 0.01



I also experimented with a different EMNIST split—**EMNIST Balanced**—and achieved a higher validation accuracy of **90.66%** using the same optimizers and hyperparameters. The benchmark accuracy for this split is **91.06%**, meaning my model performed very close to the best reported results.

This improvement suggests that the **Balanced** split is inherently easier to classify compared to the **ByClass** split, likely due to a more even distribution of samples. The combination of **Lion optimizer, CosineAnnealingWarmRestarts, SequentialLR, EfficientNet-B2, MixUp, CutMix, and KLDivLoss with class weights** proved to be highly effective in maximizing accuracy across both dataset splits.



Conclusion

Despite these results, there are still areas for improvement. One potential avenue is exploring more advanced **regularization techniques** or **adaptive augmentation strategies** to further improve accuracy, especially on the ByClass split. Additionally, experimenting with **larger or more complex models** such as **EfficientNet-B3 or ViTs (Vision Transformers)** could potentially yield better results. Another area for future work is optimizing training stability to **prevent oscillations in longer training runs**, possibly by fine-tuning the learning rate schedules further.

References

1. Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). **EMNIST: An Extension of MNIST to Handwritten Letters**. *The MARCS Institute for Brain, Behaviour and Development, Western Sydney University*.
2. Pranav, J. P., Viswanathan, K., Anandu, A. S., & Sethi, A. (2022). **WaveMix: A Resource-Efficient Neural Network for Image Analysis**. *Department of Electrical Engineering, Indian Institute of Technology Bombay, Mumbai, India*.
3. Yun, S., Han, D., Oh, S. J., Chun, S., Choe, J., & Yoo, Y. (2019). **CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features**. *Clova AI Research, NAVER Corp., Clova AI Research, LINE Plus Corp., Yonsei University*.
4. **Insights from friends**: Mustafa & Andera.