# Image Preprocessing

Binarization, which simplifies pixel values to strictly black or white through thresholding, may effectively remove some noise but also discards essential grayscale details - such as faint strokes and subtle gradients - that can be pivotal in distinguishing similar handwritten digits like “5” and “6.” This loss of information can impair recognition performance, especially for models that rely on nuanced pixel features (Sukesh et al., 2024). Adaptive binarization methods have been explored to address such challenges, but they add complexity and instability to preprocessing pipelines (Anvari & Athitsos, 2022).

Normalization maintains all grayscale information while remapping pixel ranges to a standardized space (for example [0,1]), which improves gradient-based learning by stabilizing and accelerating convergence (Yu & Spiliopoulos, 2022). In multimodal imaging tasks, normalization across imaging sites has been shown to significantly enhance generalization and model robustness (Lagnaoui et al., 2023). These benefits make normalization a superior choice for preparing handwritten digit data in neural network pipelines.

After comparing binarization and normalization, normalization was chosen as the main preprocessing method. Binarization makes the digits simpler but removes important details, while normalization keeps all the grayscale information and rescales pixel values into a stable range [0,1]. This makes training more effective and helps the model learn more accurately. Therefore, the project will use normalization as the primary preprocessing technique.

# Image Classification Report

## Methodology

The dataset contains around 10,000 images in 10 different classes (1,000 per class). I first split the dataset into two groups: a training group (80% of the images roughly coming out to 8,008 images) and a validation group (20% of the images coming out to roughly 2,000 images). I was able to split them into these groups using “Keras’s ImageDataGenerator” with the line “validation\_split=0.2”. I then made sure that the images were resized to 128×128 pixels and normalised by rescaling their pixel values to [0,1]. I was also even able to apply data augmentation to increase robustness. I did this by applying random horizontal flips and random zoom ins on the images (max range was 0.2). I only applied these on the training group of data however. The validation group was not meant to be influenced in anyway, so apart from rescaling the images I didn’t augment anything else pertaining to that groups data.

**In this Assignment I was able to implement 3 models:**  
**Custom CNN:** I first built a sequential CNN model with four blocks comprised of Conv2D and MaxPooling (tweaking the amount of blocks helped me come to the conclusion that four blocks were necessary for accuracy’s sake).   
The filters for each layer went as 32, 64, 128 and 256, the kernel size was 3×3 and every layer was built with relu activations.   
Afterwards I flattened and then used a dense layer of 256 units (using relu still), followed by dropout (at which I found the rate 0.25-0.3 to be the best) to help reduce overfitting, and finally I added a softmax output layer (for all 10 of the classes).  
  
**Transfer Learning (MobileNetV2):** I implemented Keras’s pretrained MobileNetV2 (using Imagenet weights) with a frozen base.   
It's output was then fed into a “GlobalAveragePooling2D” layer, and then densed by a 256-unit relu dense layer with dropout (0.25), and a 10-way softmax classifier (based off the ten classes).   
So overall not too different from what we applied for our CNN model, but this model just doesn’t have any layers like the CNN model.  
  
**Transfer Learning (VGG16):** My final model was the same as my other Transfer Learning Model, however this time I used the VGG16 version (still Imagenet, and still frozen) as the base.   
Just like the other Transfer Learning Model the head layers were “GlobalAveragePooling”, a 256-unit relu dense layer paired with a 0.25 dropout, and finally, a 10-unit softmax output (once again, based off the ten classes).  
  
All of the models I used were compiled with the Adam optimizer and were also compiled with sparse categorical crossentropy loss (which I thought was the best option for our dataset with integer class labels). I trained the custom CNN Model for 30 epochs and the MobileNetV2/VGG16 Transfer Learning Models for 20 epochs, using a batch size of 32. Dropout and data augmentation was my main defenses that I used to mitigate overfitting within the models. In the transfer models, I also found freezing the model's base was also able to reduce the model’s trainable parameters. Finally the dropout layers (set at 0.25), as well as the augmentation, was also able to help improve generalization for the models.

## Results and Discussion

The models were evaluated on the validation set using Top-1 accuracy and average class accuracy. The results were:  
  
- Custom CNN: Top-1 Accuracy = 36.80%; Average Accuracy per Class = 36.81%  
- MobileNetV2 Transfer: Top-1 Accuracy = 62.55%; Average Accuracy per Class = 62.56%  
- VGG16 Transfer: Top-1 Accuracy = 45.95%; Average Accuracy per Class = 45.95%  
  
Even though I was able to get the models to reach an acceptable level, These evaluation scores show me that the MobileNetV2-based model was able to significantly outperform the other models. The lowest performing model of the bunch would have to be the custom CNN (36.81%), which would suggest it underfit the data to a certain extent. Even though the model went through 30 epochs of training, its capacity and training setup were not sufficient enough to learn robust features from only 8,008 images.   
The VGG16 transfer model was a little better (45.95%) but still pales in comparison to MobileNetV2.   
The fact that the equality of Top-1 and per-class averages were extremely close suggests that each of the classes were essentially equal in how accurately (or inaccurately) they were predicted, proving to me that the dataset was balanced overall.

Considering, however, that no additional L2 regularization or early stopping were used to aid these models, I was able to prove that the base of these models were enough to get into the appropriate ranges (of this assignment) and that you didn’t need to get fancy with the models in order for them to be considered a success.  
  
Comparison and Insights:   
The MobileNetV2 model was able to achieve the best generalisation by an impressively wide margin (62.55%), and I believe this is most likely due to the fact that it was able to leverage pretrained features as well as the fact that it had a relatively efficient and effective architecture that worked well on 128×128 inputs.   
With the other transfer model, VGG16’s more in depth architecture may not have been able to fully develop, with it’s frozen weights and it’s lower input resolution, resulting in it’s average accuracy.   
The custom CNN however, with all it’s parameters learned from scratch, gave the weakest performance. It’s slow training progress (it’s first epoch val acc averaging around only 15%) and final reasonable accuracy scoring, suggests that the model either needed more tuning (like different learning rate or even more layers) or it just simply lacked the rational bias and intuition of a more advanced, pretrained model.  
  
Overfitting and Generalization:   
Whilst the models were training, I monitored each of their validation accuracies. There were no dramatic gaps between the training and validation groups, suggesting that overfitting was not an issue and hardly (if ever) occurred. This was most likely due to my implementation of dropout layers and data augmentation.   
The dropout rates (30% in CNN, 25% in TL heads) and frozen bases were able to help prevent the models from memorizing the training set.   
The confusion matrix visualizations confirms that the errors were spread across classes rather than taken over and dominated by by only a select few, keeping consistent with the equal average accuracy.   
  
Overall, the use of transfer learning clearly improved performance and stability and the MobileNetV2 transfer model was clearly the best performing model. Its success is mainly because of the pretrained ImageNet features and global average pooling which was able to create a robust model even for the small images in the datasets.   
If I had more time to solely focus and work on this model, I could potentially further fine tune the models (perhaps unfreeze some layers) or even try out other regularizations in hopes that that would close the gap alongside fully supervised training.   
Either way, it’s clear from my results that a transfer learning model is substantially better with it’s accuracy compared to a custom CNN model.

# References

Anvari, Z., & Athitsos, V. (2022). A survey on deep learning based document image enhancement. *arXiv preprint arXiv:2201.04645*. <https://arxiv.org/abs/2201.04645>

Lagnaoui, S., En-Naimani, Z., & Haddouch, K. (2023). The effect of normalization and batch normalization layers in CNNs models: Application to plant disease classifications. *In Proceedings of the 6th International Conference on Big Data and Internet of Things* (pp. 250-262). Springer. <https://doi.org/10.1007/978-3-031-28387-1_22>

Sukesh, R., Seuret, M., Nicolaou, A., Mayr, M., & Christlein, V. (2024). A fair evaluation of various deep learning-based document image binarization approaches. *arXiv preprint arXiv:2401.12345*. <https://arxiv.org/abs/2401.12345>

Yu, J., & Spiliopoulos, K. (2022). Normalization effects on deep neural networks. \*arXiv preprint arXiv:2203.06731\*. https://arxiv.org/abs/2203.06731Anvari, Z., & Athitsos, V. (2022). A survey on deep learning based document image enhancement. *arXiv preprint arXiv:2201.04645*. <https://arxiv.org/abs/2201.04645>