Automated Detection and Classification of Leukemia Using Deep Learning

Lee Kye Fung, May 2022 (link)

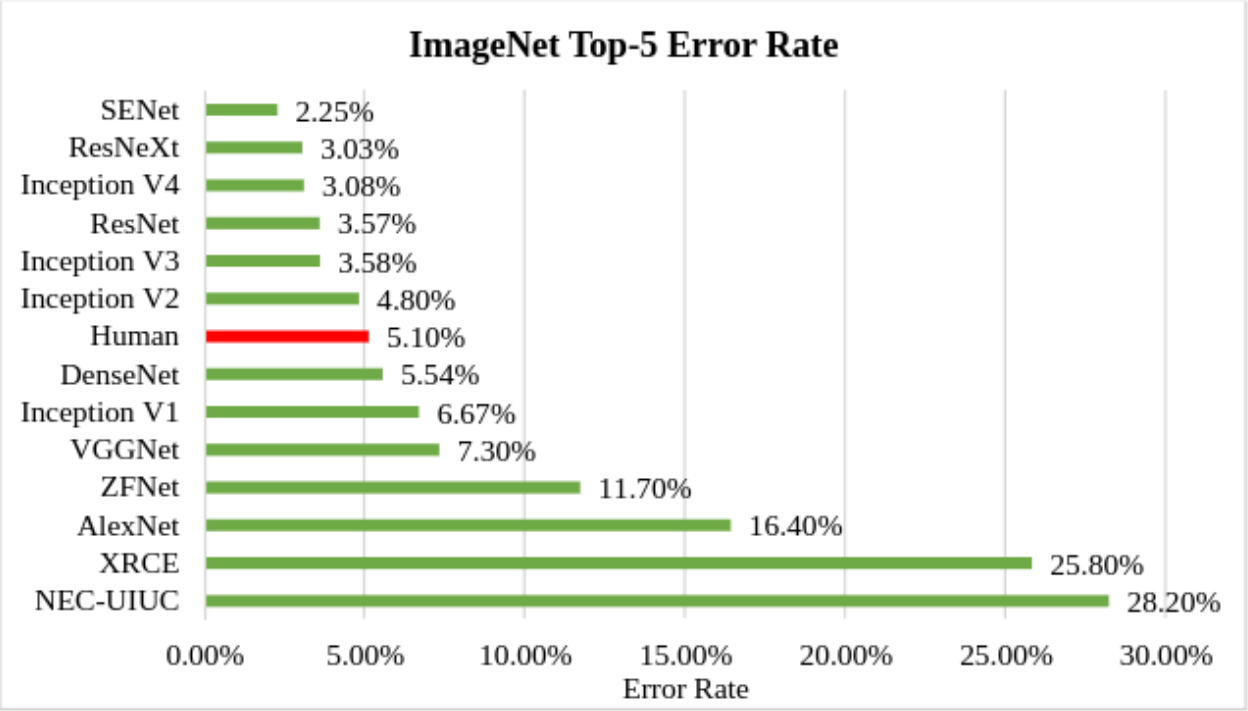


Figure 2.5: ImageNet Top-5 Error Rate of DL Models Compared to Human Errors (Adapted from Alzubaidi, 2021)

Based on this chart from Lee's 2022 paper, we can observe that SENet, ResNeXT, and Inception V4 have the lowest error rates compared to human error (2.25%, 3.03%, 3.08% respectively, compared to 5.10% human error.) We should thus consider using these models for testing/training our future cell classification model.

Lee (2022), for her case, used Categorical Cross-Entropy as a loss function, ADAM as an optimizer, and softmax as an activation method for 5 types of leukemia cells (ALL, AML, CLL, CML, and healthy cells.)

She also performed image augmentation to create more datapoints for training.

> *The images are augmented in such a way that the images are sheared, zoomed, rotated, and shifted for 2 times each, which increases the dataset by 11-fold. Then, the original and augmented images, are flipped horizontally and vertically, which again increases the size of the dataset by another 3-fold. Hence, after the image transformations, each image sample are increased by a factor of 33-fold, effectively increasing the overall dataset for each class. The code for this segment is listed in Code Listing 5 in Appendix A and the*

> *new number of samples for each class and its source is tabulated as shown in Tables 3.5 and 3.6.*

Lee (2022) makes the point that it is better to look at the F1-score of the model as both precision and recall is important for models such as this, and the F1-score is a score that combines both the precision and accuracy scores.

Precision measures how many of the “positive” predictions made by the model were correct. Recall measures how many of the positive class samples present in the dataset were correctly identified by the model. (v7labs, n.d.)

The conclusion made by Lee (2022) was that the best model to use for this training is SENet.

```
def train_model(model, criterion, optimizer, num_epochs=3):
        prev_acc=0

        print("Current K-Fold Cross Validation: {}".format(fold_var + 1))
        print("Previous validation accuracy: {:.5f}\n".format(prev_acc))

        for epoch in range(num_epochs):
                trn_acc = 0
                val_acc = 0
                trn_loss = 0
                val_loss = 0

                print('Epoch {}/{}'.format(epoch+1, num_epochs))
                print('-' * 10)

                for phase in ['train', 'validation']:
                        if phase == "train":
                                model.train()
                        else:
                                model.eval()

                        running_loss = 0.0
                        running_corrects = 0

                        for inputs, labels in dataloaders[phase]:
                                inputs = inputs.to(device)
                                labels = labels.unsqueeze(1)
                                labels = labels.to(device)
                                outputs = model(inputs.float())
                                loss = criterion(outputs, labels.float())

                                if phase == "train":
                                        optimizer.zero_grad()
                                        loss.backward()
```

```
                                        optimizer.step()

                                preds = torch.sigmoid(outputs) >= 0.5
                                running_loss += loss.item() *
inputs.size(0)
                                running_corrects += torch.sum(preds ==
(labels.data == 1))

                        if phase == "train":
                                epoch_loss = running_loss / len(train)
                                epoch_acc = running_corrects.double() /
len(train)
                                trn_loss = float(epoch_loss)
                                trn_acc = float(epoch_acc)
                        else:
                                epoch_loss = running_loss / len(valid)
                                epoch_acc = running_corrects.double() /
len(valid)
                                val_loss = float(epoch_loss)
                                val_acc = float(epoch_acc)

                        print("{} loss: {:.5f}, acc: {:.5f}".format(phase,
epoch_loss,
                                epoch_acc))

                        if phase == "validation":
                                if epoch_acc > prev_acc:
                                        save_path = #This is the location
where she saved the model.
                                        torch.save(model.state_dict(),
save_path)
                                        print("Epoch {}: val_accuracy
improved from {:.5f} to {:.5f}, saving model to
{}".format(str(epoch+1).zfill(5), prev_acc, epoch_acc, save_path))
                                        prev_acc = epoch_acc
                                else:
                                        print("Epoch {}: val_accuracy did
not improve from {:.5f}".format(str(epoch+1).zfill(5),
                                        prev_acc))

        try:
                str_history = pickle.load(open((#MODEL_LOCATION), "rb"))
                str_trn_loss = str_history['train loss']
                str_trn_acc = str_history['train accuracy']
                str_val_loss = str_history['validation loss']
                str_val_acc = str_history['validation accuracy']
        except Exception as e:
                str_trn_loss = []
                str_trn_acc = []
                str_val_loss = []
```

```
                str_val_acc = []

        str_trn_loss.append(trn_loss)
        str_trn_acc.append(trn_acc)
        str_val_loss.append(val_loss)
        str_val_acc.append(val_acc)

        history = {"train loss": str_trn_loss,
                                "train accuracy": str_trn_acc,
                                "validation loss": str_val_loss,
                                "validation accuracy": str_val_acc}

        with open(#MODEL_LOCATION, "wb") as wfile:
                pickle.dump(history, wfile)

        data = pickle.load(open(#MODEL_LOCATION), "rb")
        return data
```

Above is the function Lee used to train the SENet and ResNeXT models.

```
model_name = 'senet154'
senet = pretrainedmodels.__dict__[model_name](num_classes=1000,
pretrained="imagenet")

for param in senet.parameters():
        param.requires_grad = False

senet.last_linear = nn.Sequential(

nn.Linear(2048, 128),

nn.ReLU(inplace=True),

nn.Linear(128, 1)).to(device)
```

This is the code Lee used to perform transfer learning.

Machine learning in detection and classification of leukemia using C-NMC_Leukemia

Fatma M. Talaat & Samah A. Gamel, June 2023 (link)

Talaat and Gamel (2023) recommend a proposed leukemia classification technique: \

1. Image processing phase
2. Feature extraction phase

3. Classification phase

Image processing phase:

RGB color model, data augmentation with translation along the X and Y axis, mirrored along the vertical axis, and rotated left/right. They note that Gaussian blurring, shearing, and adding of salt + pepper noise resulted in better performance and model accuracy.

Feature extraction phase:

In this, they state that a CNN is a common network design used in machine-learning applications, but don't further state more about what the process of this is.

Classification Phase

Here, they state the use of an OCNN (Optimized CNN) for classification, with **fuzzy optimization** used to tweak and optimize the hyperparameters.

Algorithm 1

OCNN Algorithm

- ***Input:***
  - HPT containing initial values for the hyperparameters of CNN.
- ***Output:***
  - The optimal values for the hyperparameters.
- ***Steps:***

1: Initialize gbest and lr (gbest=Vi and lr=lr0)
2: Collect data from HPT
3: Add a loop between values in HPT
4: calculate the Fitness Value (FVi) using Fuzzy Logic via the three parameters stored in HPT.
5: compare the value of FVi if its greater than gbest it will perform gbest= FVi and lr=lri
6: Update the HPT
7: Assign the new values to the HPT

Symbol	Meaning
gbest	global best value
lr	Leaming rate
Vi	Initial value of gbest
lr0	Initial value for lr
HPT	HyperParameter Table
FVi	Fitness Value

OCNN algorithm

This is the algorithm of the OCNN as determined by Talaat and Gamel.

For implementation, Talaat and Gamel collected lymphocyte images already pre-identified by experienced oncologists, and performed data augmentation on it to balance the training and validation sets, using the above mentioned augmentation steps.

They used the below method to evaluate the OCNN's performance in terms of precision, recall, accuracy, and specificity:

$$\text{Precision} = \text{TP} / \ (\text{TP} + \text{FP}) \tag{1}$$

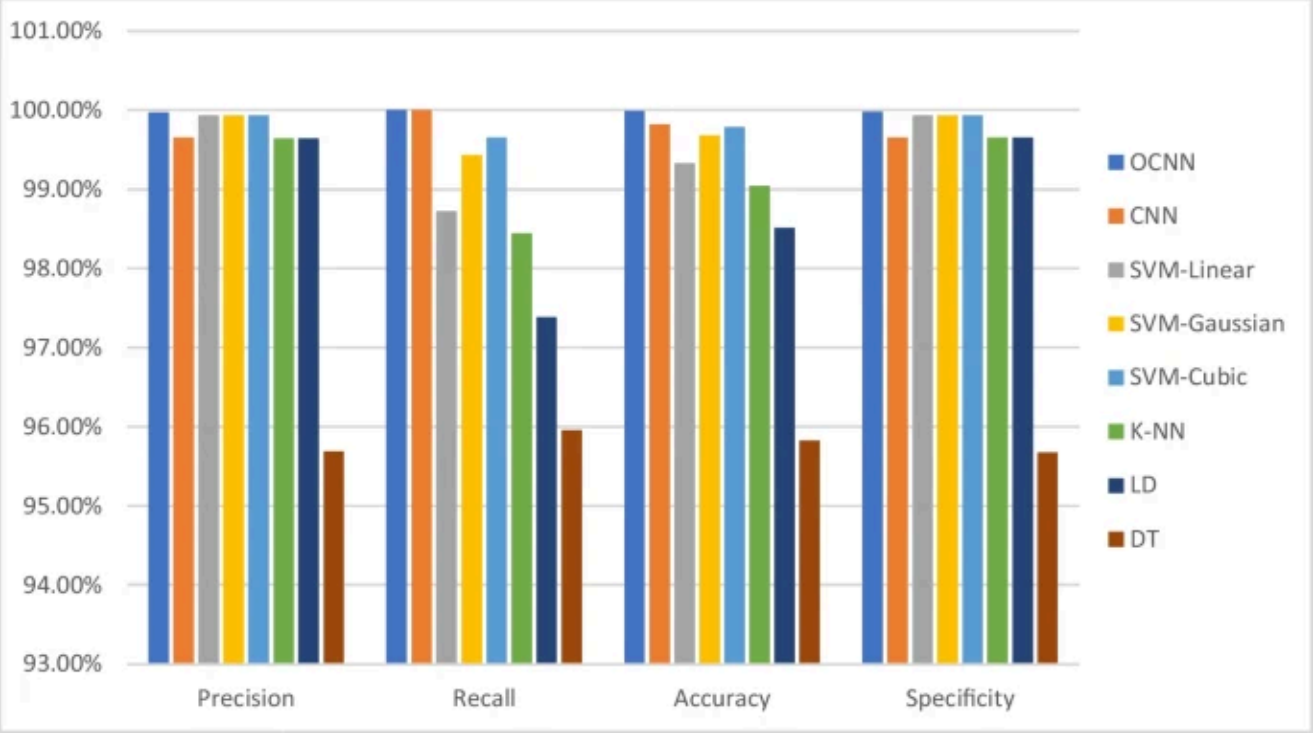
$$\text{Recall} = \text{TP} / \ (\text{TP} + \text{FN}) \tag{2}$$

$$\text{Accuracy} = (\text{TP} + \text{TN}) / \ (\text{TP} + \text{TN} + \text{FP} + \text{FN}) \tag{3}$$

$$\text{Specificity} = \text{TN} / \ (\text{TN} + \text{FP}) \tag{4}$$

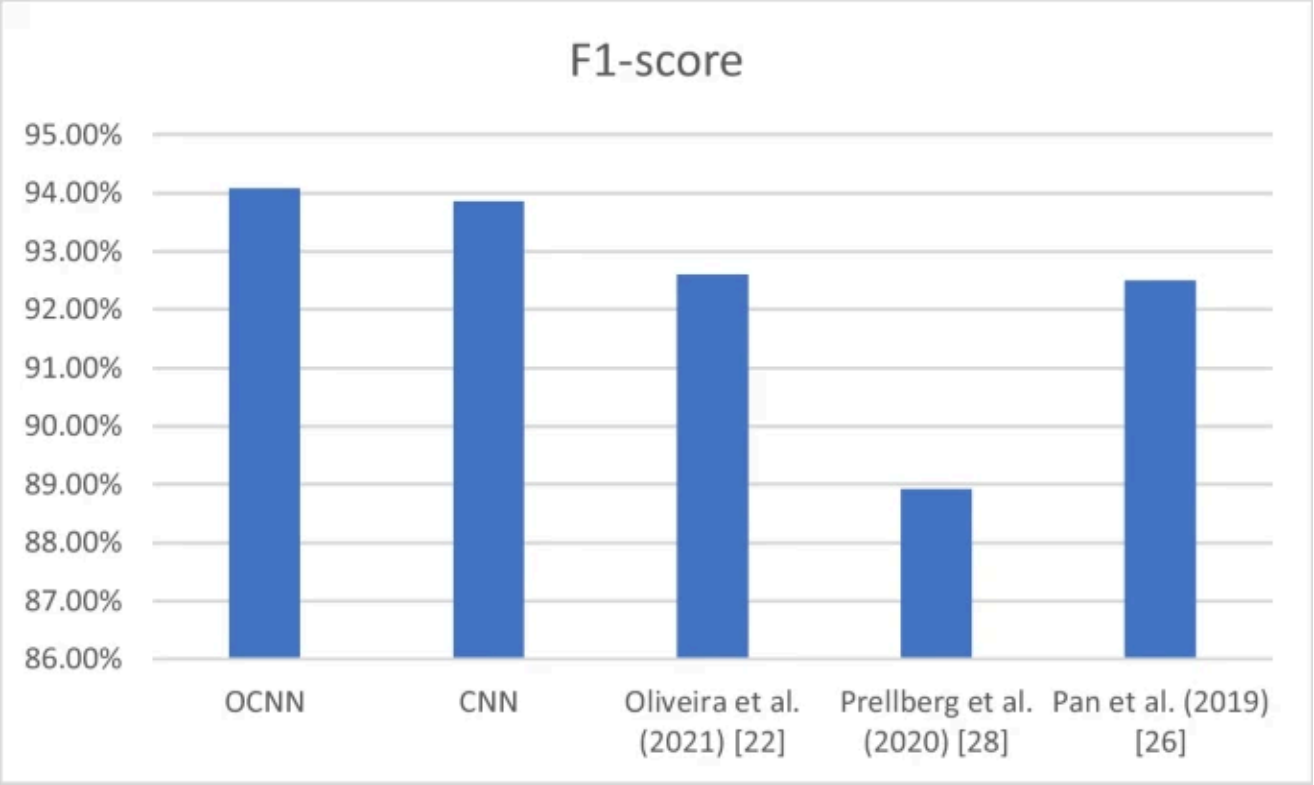
where **TP** is true positive; **TN** is true negative, **FP** is false positive, and **FN** is false negative.

Talaat and Gamel compare the performance of their OCNN to a normal CNN, Support Vector Machine (SVM), K-Nearest Neighbor (KNN), and Linear Discriminant (LD).



We can see from the results that OCNN outperforms all the other options in terms of performance, indicating that the optimization of the hyperparameters with fuzzy logic improved its performance.

Talaat and Gamel also compared the F1-score of the OCNN compared to previous methods used by previous authors.



It can be seen here that the F1-score of the OCNN is higher compared to the other options, as well.

Thus, it is worth looking into how to use fuzzy logic to further optimize the hyperparameters of a CNN.

Deep learning enhances acute lymphoblastic leukemia diagnosis and classification using bone marrow images

Elsayed et al., December 2023 (link)

In this article, Elsayed et al. reviews several pre-existing studies to determine the proficiency and adaptability of deep learning methodologies in detecting leukemia. In this case, instead of looking at the blood cells, Elsayed et al. focuses on the initial diagnostic process by examining bone marrow, which he states have been overlooked in deep learning models despite being the gold standard for leukemia diagnosis.

Elsayed et al. picked 10 studies to review after a screening process, all of which were published between 2013 to 2023. For my review, I'll note the models produced most recently (2020 onwards.)

Authors (Year)	Validation (IV/EV)	Best Models(s)	F1 (%)
Kumar et al. (2020)	IV (Train-Test)	Dense CNN (DCNN)	96.89

Authors (Year)	Validation (IV/EV)	Best Models(s)	F1 (%)
Huang et al. (2020)	IV (Train-Test)	CNN (DenseNet121) with Transfer Learning Technique	ACC (99%)
Zhou et al. (2021)	IV (Train-Test) EV (BM samples)	Ensemble of CNNs (ResNext101_32x8d, ResNext50_32x4d)	95 (EV)
Ikechukwu et al. (2022)	IV (Train-Test)	CNN (i-Net)	99.19
Kavitha et al. (2022)	IV (Train-Test)	CNN with Cat-Swarm Optimization	99.89
Yang et al. (2023)	IV (Train-Test) EV (SN-AM)	Hybrid of CNN and ViT (MobileViTv2) with MultiPathGAN	ACC(96% IV 99.72 EV)
Devi et al. (2023)	IV (Train-Test)	CNN (Convolutional Leaky RELU) with Cat-Boosting Algorithm) CNN (Convolutional Leaky RELU) with XG-Boosting Algorithm	100 97.2

We can see that the best model that Elsayed et al. discovered from all the models is the CNN with a Cat-Boosting Algorithm.

Elsayed et al. also discovered the strengths and weaknesses of each model as shown in the table below:

Authors (Year)	Outcome	Strengths	Limitations
Kumar et al. (2020)	Detection and classification of B-ALL and MM using a Dense CNN with fewer parameters.	Outperforms ML models Feature extraction capability Real-world application	Lack of external validation Limited dataset Limited interpretability
Huang et al. (2020)	Distinguishing between different types of leukemia using a CNN with transfer learning.	Multiple leukemia types Feasibility for small datasets Minimize need for segmentation	Lack of external validation Misclassification of leukemias Limited interpretability
Zhou et al. (2021)	Diagnosis of ALL in real clinical scenarios using an ensemble of CNN models.	Real-world & external validation Large dataset Mimics	Single-center data Prospective validation needed

Authors (Year)	Outcome	Strengths	Limitations
		hematologist workflow	
Ikechukwu et al. (2022)	Detection and classification of B-ALL and MM using a CNN with tuned hyperparameters.	Simplified architecture Feature selection capability Real-time applicability	Lack of external validation Limited dataset Limited interpretability
Kavitha et al. (2022)	Detection and classification of B-ALL and MM using a CNN with Cat-Swarm Optimization algorithms.	Outperforms ML models Optimized hyperparameters Real-time applicability	Lack of external validation Limited dataset Computational complexity
Yang et al. (2023)	Diagnosis and classification of leukemias using MobileViTv2 classifier and MultiPathGAN.	External validation Lightweight hybrid network Flexibility and adaptability	Sensitivity to data quality Lack of real-world scenarios
Devi et al. (2023)	Classification of B-ALL and MM using CNNs with boosting algorithms.	Feature selection Use of boosting algorithms Reduced pre-processing	Lack of external validation Limited dataset Complex segmentation

Looking at these weaknesses and strengths, we can make better decisions as to which models we should explore for usage in our case. In this case, it is likely best not to follow Huang et al.'s model as it misclassifies the leukemias. Kavitha et al.'s model is stated to outperform ML models, so it could be good to explore that model.

Elsayed et al. goes into further detail about a few models, as well:

Devi et al.'s CNN used CatBoost and XGBoost (CLR-CXG, in which CLR stands for **C**onvolutional **L**eaky **R**eLU, and CXG stands for **C**atBoost and **XG**Boost) algorithms for gradient boosting: the features extracted by the CNN were provided as inputs to the boosting algorithms, which refine the classification decisions. A novel element pointed out by Elsayed et al. is the incorporation of the Leaky ReLU's activation function, which elevates the architecture's capabilities.

What is CatBoost? (source)
CatBoost stands for **Cat**egorical **Boost**ing, and is a classifier specialized in categorical data. We can use it in place of a CNN, or in Devi et al.'s case, as a second model for their data.

What is XGBoost? (source)
XGBoost stands for e**X**treme Gradient **Boost**ing, is a gradient boosted decision tree ML library. It builds upon Supervised Machine Learning, Decision Trees, and Gradient Boosting.

What is Gradient Boosting? (source)
Gradient boosting is the idea of 'boosting' or improving a single model by combining it with other models to create a collectively strong model. The process of adding weak models is formalized as a gradient descent algorithm over an objective function.

Kavitha et al.'s deep CNN utilizes a novel CAT (Cat Swarm Optimization) algorithm for hyperparameter tuning. The process has three phases: data prep, data augmentation, and classification. The CAT algorithm enhances the model's performance by combining seeking and tracing modes to optimize the network's parameters.

What is a Cat Swarm Optimization? (source)
This is an optimization method proposed in the field of Swarm Intelligence (SI), based on how creatures perform tasks efficiently in groups. Other forms of SI are Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), and Grey Wolf Optimizer (GWO)

Cat Swarm Optimization (CSO) is inspired by the behavior of cats in the real world. In this case, there are two different states of being for any cat:
1. Seeking mode: Cats are inactive in this state, i.e. resting, looking around.
2. Tracing mode: Cats are active in this state, i.e., they change their current position.

How it Works:

1. **Initialization:** A population of 'cats' is randomly distributed to the solution space.
2. **Seeking Mode:**
    - Each cat evaluates its current position and explores nearby positions.
    - The cat decides whether to move to a new position based on a set of rules/
3. **Tracing Mode:**
    - Each cat moves towards a target position (often the best-known position so far)
    - The movement is influenced by factors like distance to the target, and the cat's speed.
4. **Update:** The positions of all cats are updated based on their movements in seeking or tracking mode.
5. **Iteration:** Steps 2-4 are repeated until a set number of iterations, or a stopping criterion is met.

The model created by Kumar et al. leverages deep learning techniques to automate classification, and employs a dense convolutional neural network (DCNN) using an Adam optimizer with a sigmoid cross-entropy loss function. Unlike what we are doing, and what Huang et al. achieves, it seems that Kumar et al. does not use transfer learning, which could result in a longer training time.

Huang et al. used Inception-V3, ResNet50, and DenseNet121 for classification, using transfer learning to optimize the model performance. This is most similar to what we've been doing these past testing periods. Huang et al. achieved best performance with DenseNet121, achieving a 95.3% after transfer learning. Elsayed et al. notes that Huang et al.'s models has challenges such as distinguishing specific cell types, which could potentially be a problem as well.

Yang et al. used their MultiPathGAN model to perform a novel method called "stain domain augmentation", which normalized stain styles and expanded their dataset. A lightweight model called MobileViTv2, which combined the strengths of CNN and vision transformers (ViTs) was developed for this classification. MobileViTv2 achieved an average accuracy of 94.28% on their test set, and outperformed CNNs and ViTs despite only having 9.8 million parameters. (For example, ResNet has over 23 million parameters.) However, the application of the device is focused on diagnosing broad disease categories as opposed to specific subtypes.

Zhou et al. used raw images without preprocessing to train their CNN architecture. They utilized different configurations of ResNet and ResNeXt for white blood cell detection and classification, and the ensemble of CNNs comprising of ResNeXt101_32x8d, ResNeXt50_32x4d, and ResNet50 was their top performing configuration, with 82.02% F1 score. This is unique in the fact that there was no preprocessing done on any images, and it also replicated the workflow of hematologists, which allows for easier real-world application.

In conclusion, perhaps we should look into things like CatBoost in order to continue to get better results on our classification tasks. Looking into CatSwarm also seems interesting, as Kavitha et.al achieved very high F1 scores in their application of CatSwarm.