Introduction

Blood cells play a critical role in the human body, carrying oxygen, fighting infections, aiding in the clotting process, and serving various other vital functions. Analyzing and correctly classifying different types of blood cells can offer profound insights into an individual's health and help early diagnose various diseases and conditions. With the advancement of digital imaging and computer vision, the possibility of automating blood cell analysis has become a reality.

The primary aim of this study is to evaluate the performance of four popular algorithms – Random Forest, K-Nearest Neighbors (KNN), Convolutional Neural Network (CNN), and Multilayer Perceptron (MLP) – in classifying the blood cell images into their respective categories. By comparing these algorithms, we aspire to discern which methodology is best suited for our existing dataset, for image-based blood cell classification tasks.

Data

Data description and exploration

The BloodMNIST is based on a dataset of individual normal cells, captured from individuals without infection, hematologic or oncologic disease and free of any pharmacologic treatment at the moment of blood collection. It contains a total of 17,092 images and is organized into 8 classes. We split the source dataset with a ratio of 7:1:2 into training, validation and test set. The source images have been resized into $3 \times 28 \times 28$(). The given dataset contains in total of 8 classes in y-train and y_test dataset.

Class distribution of in Training dataset

It shows imbalanced class distribution it might causes bias toward the majority class, so it will tend to make predictions in favor of the class that has more examples, as it learns that this is the most likely outcome. It also might cause poor generalization problem: The model may perform well on the training data, but it might generalize poorly to new, unseen data, especially for the minority class.

This is the visual summary of the pixel intensity distribution across an image or a set of images. This can help in understanding the common patterns or variations in data corresponding to each class.

We can see that the pixel intensities distribution of different has lots of differences, however the pattern of class 1 and class 4 is very similar, it would cause the confusion of the training model. By analyzing the graph, it can help us selecting suitable data processing decision.

Preprocessing description and justification

The preprocessing function we implemented followed following steps:

1. Data Augmentation: The function utilizes TensorFlow's data augmentation capabilities to artificially expand the training dataset. Specifically, the images undergo random rotations (up to 10% of the image's width or height), translations (up to 10% in both x and y directions), and random horizontal flips. These augmentations introduce variability and can help the model become more robust to variations in the input data, ultimately improving its generalization capabilities to avoid overfitting.

2. Flattening: Before scaling, the images are flattened, converting each image from a 2D array (or 3D if it's a color image) into a 1D array. This transformation facilitates the next scaling step by presenting the data in a format that scalers expect.

3. Scaling using MinMaxScaler: MinMaxScaler scales and translates each feature individually such that it is in the given range (default is between 0 and 1). Scaling pixel values is essential, especially for neural network-based models (CNN and MLP), as it ensures that all input features (in this case, pixels) have the same scale, leading to faster and more stable convergence during training.

4. Reshaping: After scaling, the data is reshaped back to its original format, ensuring it can be fed into models that expect input in the shape of images.

5. One-Hot Encoding of Labels: The target labels are one-hot encoded, converting them into a binary matrix representation. This step is particularly crucial for multi-class classification tasks when using neural networks, as the network's output layer typically has as many neurons as there are classes, each producing the probability for a particular class.

The choice of preprocessing steps often on the nature of this image dataset, the potential neutral network algorithm to provide a good balance between data enhancement and computational efficiency.

Methods

Theory of Convolutional Neural Network (CNN)

CNN is an artificial neural network specifically good to analyze image. It could perfectly fit the feature of images that has grid feature. The CNN model needs to be feed from input layer and has output layer as end. The whole process includes building fewer layers such as pooling layer, connected layer and active functions inserted.

Strengths and weakness of CNN

Strengths of CNN

CNNs also employ various techniques to prevent overfitting, such as dropout and data augmentation. It is also flexible to be conducted by image transformation for better classifications.(Ilesanmi, A. E., & Ilesanmi, T. O.)

Weaknesses of Using CNN

The layers algorism contains only hyperparameters but it needs further investigation to understand if the accuracy is not good. It might be a challenge to choose hyperparameters such as learning rate, batch size, etc. Proper tuning is often necessary. (Amato, G., & Falchi, F. ,2020)

Architecture and Hyperparameter of CNN

Architecture of CNN:

1.Input layer: The input layer expects 28x28 pixel images with 3 channels (RGB channels).

2. 2D convolution layer: It will learn 32 filters (or "kernels"), each of which is of size 3x3. The activation="relu" means it uses the Rectified Linear Unit (ReLU) activation

3. Max pooling layer, which down-samples the feature maps. It looks at 2x2 regions of the feature map and picks the maximum value from each region, effectively halving the height and width of the feature map.

4. Dense layer: This layer flattens the 2D feature maps into a 1D vector. It classifies the image into one of 10 classes using a dense output layer.

5. Output layer: This is a fully connected layer that outputs 10 values use softmax activation function

Hyperparameters:

The best hyperparameter combination is:

'optimizer': 'adam', 'num_filters_2': 128, 'num_filters_1': 32, 'epochs': 50, 'dropout_rate': 0.2, 'dense_units': 64, 'batch_size': 64

The explanation of each feature:

'adam' refers to the Adaptive Moment Estimation (Adam) optimizer

'num_filters_2': Number of Filters in a convolutional layer indicates how many distinct features the layer should try to learn from the input.

‘epochs': Epochs represent the number of times the learning algorithm will work through the entire training dataset. In this case, the dataset will be passed through the neural network 50 times.

'dropout_rate': Dropout Rate is the fraction of the input units to drop during training. It's a regularization technique to prevent overfitting. A dropout rate of 0.2 means that 20% of the input units will be randomly set to zero during training.

'dense_units': Dense Units refer to the number of neurons or nodes in a dense (or fully connected) layer. This indicates that there is a dense layer in the network with 64 neurons.

'batch_size': Batch Size is the number of training samples used in one iteration (or forward/backward pass). Using batches is more memory-efficient than using the entire dataset. A batch size of 64 means that 64 samples from the training dataset will be used to compute the gradient and update the weights in one iteration.

Theory of Multi-Layer Perceptron(MLP)

The Multi-Layer Perceptron (MLP) is a class of feedforward artificial neural network. At its core, an MLP is composed of one or more layers of neurons. Data is fed into the input layer, undergoes intermediate processing in the hidden layers, and the final output is generated in the output layer. The connections between neurons can be weighted, and these weights are adjusted during training to optimize the network's performance for a given task.

The fundamental architecture choices in MLP revolve around the number of hidden layers, the number of neurons in each layer, and the activation functions used in these neurons.

Strengths and weakness of MLP

Strengths of MLP

Direct Learning:

MLPs can learn directly from raw pixel values, eliminating the need for manual feature extraction or engineering. This can be beneficial in scenarios where it's challenging to handcraft features.

Flexibility:

MLPs can model complex, non-linear relationships, making them adaptable to a wide range of tasks, including those with intricate patterns in image data.(Pinkus, 1999).

Ease of Implementation:

With modern deep learning frameworks, implementing and training an MLP is straightforward. This makes it a good starting point for establishing a baseline performance on image datasets.

Weaknesses of Using MLP on Image Data:

Scalability Issues with Large Images:

For high-resolution images, the number of input neurons in an MLP can be very large, leading to an explosion in the number of parameters. This can make the network prone to overfitting and can also be computationally intensive.(Tolstikhin, et.al, 2021).

Overfitting:

Without proper regularization techniques (like dropout or weight decay), MLPs, especially large ones, are prone to overfitting, especially when the amount of training data is limited.

Exploding gradients:

Deep MLPs can suffer from issues like vanishing or exploding gradients, making them harder to train compared to more modern architectures like CNNs.

Architecture and Hyperparameter of MLP

Architecture of MLP

1.Layers:
Input Layer: Takes in the data. For images, it's like feeding in all the pixels.
Hidden Layers: Layers in between that process and transform the data.
Output Layer: Gives the final prediction, like classifying an image.

2. Neurons: Small processing units in each layer. They calculate a sum of their inputs, adjust it with a bias, and then apply a function to it (like turning a switch on/off).

3. Activation Functions: The "switch" that neurons use. Helps the network learn complex patterns. Examples include sigmoid and ReLU.

4. Weights: Think of these as the strength or importance of connections between neurons. They're adjusted during learning.

5. Learning: Uses an algorithm called backpropagation. The network makes a prediction, sees how wrong it is, and then tweaks the weights to be better next time.

6. Regularization: Techniques to prevent the network from memorizing the training data and to help it generalize better to new data.

Grid Search Setup for an MLP:

Hyperparameters in the Grid:

model__n_units:This defines the number of neurons (or units) in each layer.Two architectures are considered:One with 2 hidden layers having 32 and 16 neurons respectively.Another with 2 hidden layers having 128 and 64 neurons respectively.

model__dropout_rates:Dropout is a regularization technique for neural networks. It randomly "drops" or deactivates a fraction of neurons during training.wo dropout rates are considered for both layers:10% (0.1) for both layers.20% (0.2) for both layers.

model__optimizer:The optimizer is responsible for updating the model's weights based on the error of the predictions.Two optimizers are being considered:Adam: A popular optimizer that adapts the learning rate during training.RMSprop: Another adaptive learning rate optimizer.

model__activation: The activation function introduces non-linearity to the model.Only the 'ReLU' (Rectified Linear Unit) activation function is being considered here.

batch_size:Represents the number of samples that will be propagated through the network at once.Two batch sizes are considered:32 samples at a time.128 samples at a time.

epochs:An epoch is one complete forward and backward pass of all training samples.Two durations for training are considered:20 epochs.40 epochs.

{'batch_size': 128, 'epochs': 40, 'model__activation': 'relu', 'model__dropout_rates': [0.1, 0.1], 'model__n_units': [128, 64], 'model__optimizer': 'adam'}

Ensemble Method selected: K-nearest neighbors(KNN) and Random Forest

It is required to preprocess the dimension of dataset in 2 dimensions since both methods of random forest and KNN has issues on processing the high dimension value that traditionally used for high-dimensional data like images. Reshaping the data into 2D can still give a baseline performance.

Theory of KNN:

K-nearest neighbors' algorithm (KNN) is a well-known classification method which fit the target mission of the project. Each image can be represented as a high-dimensional vector based on pixel intensities. For our current task that we are evaluate the colored images.

Strength and weakness of KNN:

Strength of KNN: KNN is non-parametric algorithm, can capture these non-linearities without any explicit modeling.

The model parameters are easy to compromise with less parameters. Some image classes might have prototypical representations, meaning certain images are very representative of their class. (Guo, G. et.al., 2003). KNN can easily identify and match new images with these prototypes. As new labeled images are acquired, they can be simply added to the dataset without the need for retraining a model, making KNN adaptable. (Amato, G., & Falchi, F. ,2010)

Weakness of KNN: Except dimension problem. Make decision of K values and scaling method needs to be considered carefully and the grid search of each parameters consume time. Besides, there might be overfitting problem. It can be evaluated by confusion matrix.

Architecture and hyperparameters

1. Use randomgrid to detect the k value.
2. Set k value from 1 to 15. Set weight method as uniform and distance.
3. The method calculating distance is set as 'euclidean' and 'manhattan'.

Hyperparameters

The best combination hyperparameter is n_neighbors=15, weights='uniform', metric='manhattan'.

The algorithm will consider the 15 nearest data points (neighbors) from the training set when making a prediction for a new data

The 'uniform' weight option means that all neighbors have equal influence on the prediction of a new data.

The “manhattan” is a mathematical way to calculate the distance between new data.

Theory of Random Forest:

The Random Forest model is an ensemble of decision trees. It works by aggregating the outputs of individual decision trees to make a final decision. The fundamental architecture choices in Random Forest revolve around how each tree is constructed and how many trees are in the forest. It is robust to error and provide high performance.

By averaging out biases and reducing variance, they provide a balanced trade-off which results in better generalization to unseen data.

Strength and weaknesses of the random forest:

Strength of random forest

Performance: Due to the ensemble nature of Random Forests, they often yield a good performance. By averaging out biases and reducing variance, they provide a balanced trade-off which results in better generalization to unseen data.

Overfitting: The bootstrapping (sampling with replacement) and random subspace (random selection of features) techniques inherently make Random Forests resistant to overfitting. By aggregating results from multiple trees, the model smoothes out individual errors or biases.

Weaknesses of random forest

Interpretability: Though individual decision trees can be visualized and interpreted, a forest is more complex, and understanding how the collective decision is made can be challenging.(Belgiu & Drăguţ, 2016)

Runtime and Memory: Training can be slow, especially with a large number of trees. The model also requires storing all trees, which can be memory intensive.

Architecture and hyperparameter of random forest

Architecture:

Ensemble Learning: The main idea behind ensemble learning is to combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability and robustness over a single estimator.

Bootstrap Aggregating (Bagging): Random Forests utilize the bagging method to draw random bootstrap samples of the training data. For each of these samples, a decision tree is constructed. This results in a 'forest' of trees.

Random Subspaces: Unlike traditional decision trees, when splitting a node during the construction of the tree, random forests select the best feature from a random subset of features. This injects randomness into the tree-building process and ensures that the trees are diverse.

Aggregation: For a classification problem like the one in the provided code, each tree in the forest 'votes' for a class, and the class receiving the most votes becomes the model's prediction.

Grid search:

The chosen search method is Grid Search, which is a systematic way of working through multiple combinations of hyperparameters, cross-validating as it goes to determine which combination gives the best performance. In our case, we use 3-fold cross-validation (cv=3), meaning the training set is split into 3 parts: 2 are used for training, and 1 is used for validation. This process is repeated 3 times, so every part gets used for validation once.

Hyperparameters in the Grid:

1. bootstrap:Determines whether bootstrap samples are used when building trees.False means the whole dataset is used to build each tree.

2. max_depth:The maximum depth of the tree.20 indicates that trees can grow up to a depth of 20 nodes.

3. min_samples_leaf:The minimum number of samples required to be at a leaf node.1 means a leaf node can have as few as one sample.

4. min_samples_split:The minimum number of samples required to split an internal node.

5. 5 means that at least 5 samples are required at a node for it to be split further.

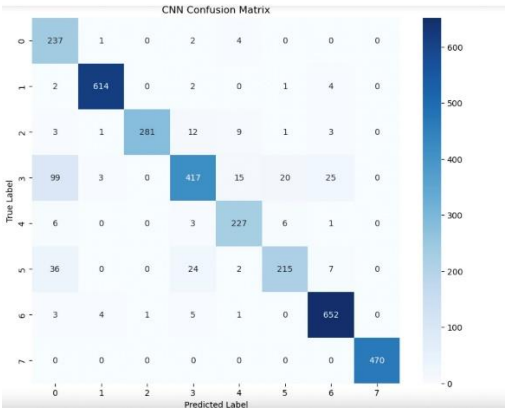
6. n_estimators:The number of trees in the forest.200 means the forest will be composed of 200 trees.

Best combination: 'bootstrap': False, 'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 200

Results and Discussion

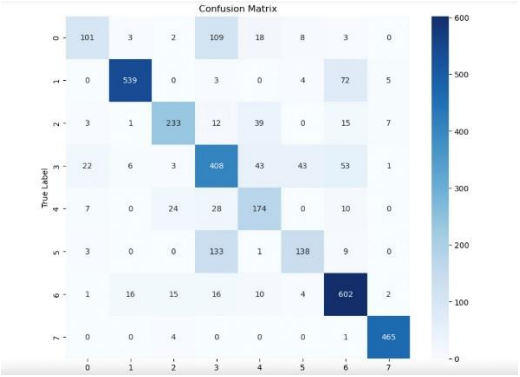
Comparative Analysis: With 4 different of machine learning algorithms available, it becomes imperative to understand the strengths and weaknesses of each in relation to specific tasks. By examine algorithms against the same dataset, we can derive insights about their applicability, and efficiency in image classification tasks.

	precision	recall	f1-score	support
0	0.61	0.97	0.75	244
1	0.99	0.99	0.99	623
2	1.00	0.91	0.95	310
3	0.90	0.72	0.80	579
4	0.88	0.93	0.91	243
5	0.88	0.76	0.82	284
6	0.94	0.98	0.96	666
7	1.00	1.00	1.00	470
accuracy			0.91	3419
macro avg	0.90	0.91	0.90	3419
weighted avg	0.92	0.91	0.91	3419



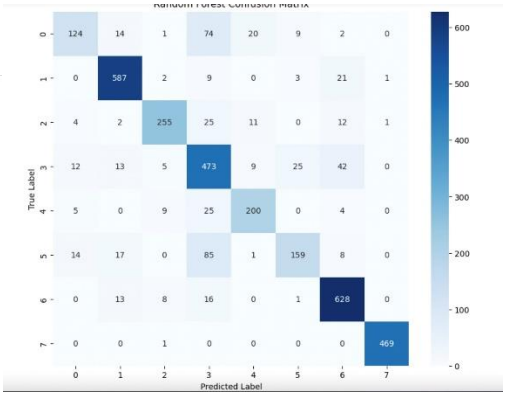
CNN

	precision	recall	f1-score	support
0	0.74	0.41	0.53	244
1	0.95	0.87	0.91	623
2	0.83	0.75	0.79	310
3	0.58	0.70	0.63	579
4	0.61	0.72	0.66	243
5	0.70	0.49	0.57	284
6	0.79	0.90	0.84	666
7	0.97	0.99	0.98	470
accuracy			0.78	3419
macro avg	0.77	0.73	0.74	3419
weighted avg	0.79	0.78	0.77	3419



MLP

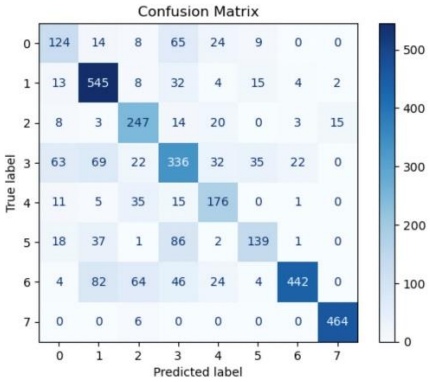
	precision	recall	f1-score	support
0	0.78	0.51	0.62	244
1	0.91	0.94	0.93	623
2	0.91	0.82	0.86	310
3	0.67	0.82	0.74	579
4	0.83	0.82	0.83	243
5	0.81	0.56	0.66	284
6	0.88	0.94	0.91	666
7	1.00	1.00	1.00	470
accuracy			0.85	3419
macro avg	0.85	0.80	0.82	3419
weighted avg	0.85	0.85	0.84	3419



Randon forest

Accuracy: 0.7233109096226967

	precision	recall	f1-score	support
0	0.51	0.51	0.51	244
1	0.72	0.87	0.79	623
2	0.63	0.80	0.70	310
3	0.57	0.58	0.57	579
4	0.62	0.72	0.67	243
5	0.69	0.49	0.57	284
6	0.93	0.66	0.78	666
7	0.96	0.99	0.98	470
accuracy			0.72	3419
macro avg	0.71	0.70	0.70	3419
weighted avg	0.74	0.72	0.72	3419



KNN

From the images above: The CNN accuracy score is 0.91, the MLP accuracy is 0.85, the random forest accuracy score is 0.85, the KNN accuracy score is 0.72.

By studying the performance of different algorithms on the blood cell image dataset, we can glean valuable information about the suitability of these algorithms for image-based datasets in general and medical imaging in particular.

Though the concept of each model is different, the accuracy of each model is different, the confusion matrix of each model has the obvious similarities.

The color could indicate the following information:

Label 1, label 6, label 7 are the easiest labels to be classified no matter which model is applied. The reason could be: label1 and label 6 has enough volume. Label 7 has very significant appearance. It could be side proved that: from the sample graphs we load at the beginning, the cells in the images could be identified that has some features. There are two sections of color in the cell. There two sections look like a specific shape like “8”. Their pixel values look much simply than others. Besides, the label 5 looks likely to be detected as label 3. CNN could better solve the specific problem of confusion between label 5 and label 3.

Overall, we suggest that CNN is optimal choice when investigate image classification task. The large volume of data could improve the performance of modeling.

Hyperparameter Set	optimizer	num_filters_2	num_filters_1	epochs	dropout_rate	dense_units	batch_size	runtime	Accuracy
Best	adam	128	32	50	0.2	64	64	1h 23min	91%
Set 1	sgd	64	16	100	0.5	128	32	2h 10min	87.7%
Set 2	rmsprop	256	64	75	0.3	32	128	1h 55min	88.9%

Runtime
The runtime for Best model is also the shortest, possibly due to the efficient 'adam' optimizer and balanced architecture that neither underfits nor overfits the training data. 'Set 1' took the longest time because of the highest number of epochs and a smaller batch size, leading to more updates and longer training times.

Dropout Rate
The moderate dropout rate in the Best model = set seems to have provided a good balance to prevent overfitting, while in 'Set 1', the high dropout rate might have led to underfitting or insufficient learning, reflected in its lower accuracy.

Model	Hyperparameters	Run Time
CNN	'optimizer': 'adam' 'num_filters_2': 128 'num_filters_1': 32 'epochs': 50 'dropout_rate': 0.2 'dense_units': 64 'batch_size': 64	1h 23min
Random Forest	'bootstrap': False 'max_depth': 20 'min_samples_leaf': 1 'min_samples_split': 5 'n_estimators': 200	58min
MLP	'optimizer': 'adam' 'num_filters_2': 128 'num_filters_1': 32 'epochs': 50 'dropout_rate': 0.2 'dense_units': 64 'batch_size': 64	2h 10min
KNN	'n_neighbors': 15 'weights': 'uniform' 'metric': 'Manhattan'	1h 47min

CNN (Convolutional Neural Network)
Results: The CNN model, with hyperparameters like adam optimizer and dropout rate of 0.2, has a runtime of 1h 23min and is among the fastest models to train.

Random Forest
Results: The Random Forest, with 200 estimators and a maximum depth of 20, took 58min to run.
Theoretical Properties: Random Forest is an ensemble of decision trees. It reduces overfitting by introducing randomness into its construction.

MLP (Multi-layer Perceptron)
Results: The MLP, which is a type of deep feed-forward artificial neural network, had a runtime of 2h 10min, making it the slowest among the models listed. MLP training time might be long due to increase number of estimators.

KNN (k-Nearest Neighbors)
Results: With 15 neighbors, uniform weights, and the Manhattan metric, the KNN model took 1h 47min.

Theoretical Properties: KNN is a lazy learner, meaning it doesn't learn a discriminative function from the training data but memorizes the dataset instead.

The MLP and CNN model has use grid research which sort all the combination of hyperparameters, which could cause train time much large compared to Random Forest and KNN use randomizedsearchCV which randomly select combination of parameter, also the ‘epoch’ and ‘max_depth’ those quantitative contains seems to have larger impact. Therefore, it aligns with our expectation.

Conclusion

Limitations:

Processing capacities. It is time consuming to do each model with different hyperparameters. It could not be avoided since we could not identify the concept in the “black box” clearly.

The actual accuracy rates of each model are not high as expectations. It needs further investigations in the future. At this moment, we start two models which are specific useful for two dimensions dataset. In the future, it could be possible that we could looking for method which are effective on high dimensional dataset. Currently, we didn’t try more dimensional reduction methods such as PCA. It is possible that different way of dimensional reduction could applied. Last, the images has imbalance distribution which might be sorted to the same amounts.

Reflection:

Completing the assignment was a valuable learning experience that significantly enriched my understanding of the subject. One of the primary takeaways was the importance of data preprocessing and feature engineering in machine learning workflows. The quality and relevance of the input data directly impact the performance of a model, and by strategically preprocessing the data and engineering meaningful features, we can substantially enhance the model's predictive power.

Another crucial learning point was the significance of model evaluation and hyperparameter tuning. Understanding that there's no one-size-fits-all in machine learning, it's essential to experiment with various hyperparameters and regularly evaluate models to ensure their generalization capability on unseen data. These insights emphasized that while the choice of algorithm is essential, the steps before and after model training – data preprocessing, feature engineering, and model evaluation – are equally crucial to the success of a machine learning project.

Reference

Guo, G., Wang, H., Bell, D., Bi, Y., Greer, K. (2003). KNN Model-Based Approach in Classification. In: Meersman, R., Tari, Z., Schmidt, D.C. (eds) On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE. OTM 2003. Lecture Notes in Computer Science, vol 2888. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-39964-3_62

Ilesanmi, A. E., & Ilesanmi, T. O. (2021). Methods for image denoising using convolutional neural network: a review. *Complex & Intelligent Systems*. https://doi.org/10.1007/s40747-021-00428-4

Amato, G., & Falchi, F. (2010). kNN based image classification relying on local feature similarity. *Proceedings of the Third International Conference on SImilarity Search and APplications - SISAP '10*. https://doi.org/10.1145/1862344.1862360

Pinkus, A. (1999). Approximation theory of the MLP model in neural networks. *Acta Numerica, 8*, 143-195. doi:10.1017/S0962492900002919

Tolstikhin, I. O., Houlsby, N., Kolesnikov, A., Beyer, L., Zhai, X., Unterthiner, T., ... & Dosovitskiy, A. (2021). Mlp-mixer: An all-mlp architecture for vision. *Advances in neural information processing systems*, *34*, 24261-24272.

Belgiu, M., & Drăguţ, L. (2016). Random forest in remote sensing: A review of applications and future directions. *ISPRS Journal of Photogrammetry and Remote Sensing*, *114*(114), 24–31. https://doi.org/10.1016/j.isprsjprs.2016.01.011

Parmar, A., Katariya, R., & Patel, V. (2019). A review on random forest: An ensemble classifier. In *International conference on intelligent data communication technologies and internet of things (ICICI) 2018* (pp. 758-763). Springer International Publishing.