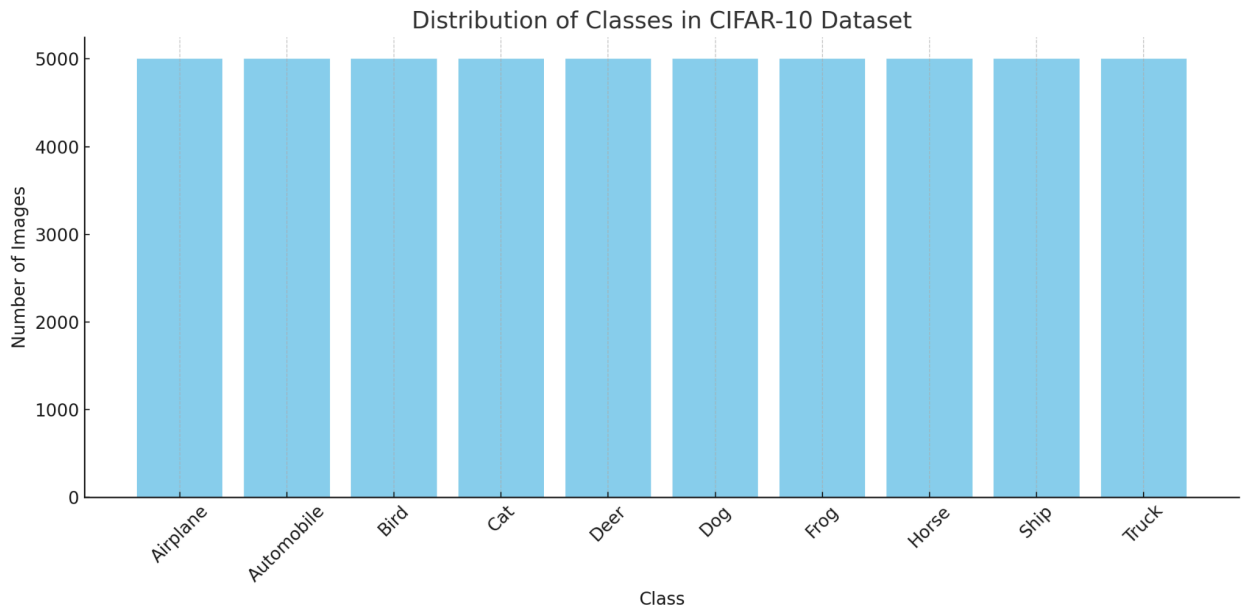
Exploring Traditional CNNs with Different Activation Functions and Convolutional Layers

For this task I have chosen the CIFAR-10 dataset. The CIFAR-10 dataset consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 test images. The CIFAR-10 dataset serves as a canvas for exploring and experimenting with ideas in the fields of feature extraction, dimensionality reduction, and transfer learning. Its image resolution, coupled with its diverse and balanced composition, presents an ideal set for both novice and seasoned practitioners to refine their skills and investigate new avenues in the domain of image analysis. Through the lens of CNNs, the CIFAR-10 dataset becomes more than just a collection of images; it evolves into a canvas where pixels harmonize to unveil patterns that escape the naked eye.



Here, a base CNN architecture was defined using TensorFlow/Keras. The base architecture consisted of convolutional layers, max-pooling layers, and fully connected layers. The model was compiled with the Adam optimizer and categorical cross-entropy loss. We introduced three different activation functions: ReLU, Leaky ReLU, and Sigmoid. Three CNN models were created, each using one of the activation functions in the convolutional layers. Each model was compiled and trained on the training data for 10 epochs. Different convolutional layer configurations were investigated, including varying the number of filters, kernel sizes, and strides. Models with these different configurations were defined and compiled. Each model was trained on the training data for 10 epochs. Performance metrics (accuracy, precision, recall, F1-score) were computed for each trained model using classification_report from sklearn.metrics. The results were stored for further analysis.

Firstly, we loaded the CIFAR-10 dataset from the batch files of it using Python's pickle library.
The loaded data was split into training and validation sets using train_test_split from sklearn.model_selection. The image data was normalized to have values between 0 and 1.
Labels were converted to one-hot encoded format using to_categorical from tensorflow.keras.utils. The data was converted to PyTorch tensors using torch.tensor, and data loaders were created using DataLoader from torch.utils.data.

Next, the CIFAR-10 dataset is loaded using the cifar10_batch function, which reads the data from different batch files. The data is then combined, and images and labels are extracted. The images are reshaped and transposed to the appropriate format. Now, the dataset is split into training and validation sets using the train_test_split function. Image data is normalized to the range [0, 1], and labels are converted to one-hot encoded vectors. Now to train the file, a sequential CNN model is created using the Keras library. The model consists of convolutional

layers with activation functions, max-pooling layers, flattening, and dense layers. The model is compiled with the Adam optimizer and categorical cross-entropy loss.

We define a sequential model using Keras, which allows us to build a neural network layer by layer. The first layer is a 2D convolutional layer (Conv2D) with 32 filters, a kernel size of (3, 3), 'relu' activation function, and 'same' padding. It also specifies the input shape of (32, 32, 3), which corresponds to the size and color channels of the CIFAR-10 images. After each convolutional layer, we add a max-pooling layer (MaxPooling2D) with a pool size of (2, 2).
The Flatten() layer is used to flatten the output from the previous layers into a 1D vector.
We then add two fully connected (dense) layers. The first dense layer has 128 neurons with 'relu' activation, and the second dense layer has 10 neurons with 'softmax' activation for multi-class classification. Then, the model was compiled by specifying the optimizer as 'adam,' the loss function as 'categorical_crossentropy' (common for multi-class classification), and the metric to track as 'accuracy.' Here the model was trained using the fit method. We provide the training data (X_train, y_train_onehot), validation data (X_val, y_val_onehot), the number of training epochs (10), and the batch size (64).

	A	B	C	D	E
1	Epoch	Training loss	Training accurac	Validation loss	Validation accuracy
2	1	1.4904	0.4684	1.2226	0.5619
3	2	1.1182	0.608	1.0896	0.6167
4	3	0.9757	0.6577	1.008	0.6459
5	4	0.8666	0.6984	0.9094	0.6847
6	5	0.7814	0.7266	0.9135	0.6822
7	6	0.7082	0.7536	0.9013	0.692
8	7	0.6359	0.7778	0.8955	0.698
9	8	0.5704	0.8024	0.8912	0.7042
10	9	0.5061	0.8239	0.9408	0.7
11	10	0.4468	0.8447	0.9633	0.6994

In the first epoch we see the training loss (categorical cross-entropy) is approximately 1.49, and the training accuracy is around 0.46. The validation loss is about 1.22, and the validation accuracy is approximately 0.5619. For the other epochs, we see Similar to the first epoch, each subsequent epoch shows a decrease in training loss and an increase in training accuracy. The validation loss and accuracy also exhibit trends. In general, as training continues, we expect the model to improve its performance on both the training and validation data. The decreasing training loss and increasing training accuracy are positive signs. They indicate that the model is learning and improving its performance on the training data. The decreasing training loss and increasing training accuracy are positive signs. They indicate that the model is learning and improving its performance on the training data. The validation loss also improves, but the validation accuracy plateaus after epoch 7. This suggests that the model is overfitting to the training data after epoch 7. The validation accuracy of 0.6994 means that the model is able to correctly classify 69.94% of the images in the validation set which is a good result. However, the model could be improved by using a different optimizer.

Now, we have created and trained three Convolutional Neural Network (CNN) models with different activation functions: ReLU, Leaky ReLU, and Sigmoid. These models are trained on the CIFAR-10 dataset. The code conducts training for 10 epochs, and now we will analyze the output and performance of these models.

We define a function create_cnn_model that creates a CNN model with different activation functions. Depending on the specified activation function ('relu', 'leaky_relu', or 'sigmoid'), the model is configured with Convolutional, MaxPooling, Flatten, and Dense layers. We created three instances of the CNN model with different activation functions: ReLU, Leaky ReLU, and Sigmoid. Each model is compiled with the Adam optimizer and categorical cross-entropy loss. The models are trained on the CIFAR-10 dataset for 10 epochs with a batch size of 64. The

output shows the training and validation accuracy and loss for each of the three models during the 10 training epochs.

The ReLU model starts with a training accuracy of approximately 48%, which increases to around 83% after 10 epochs. The validation accuracy starts around 57% and reaches approximately 70% after 10 epochs. The model exhibits significant improvement in accuracy during training, indicating successful learning.

The Leaky ReLU model begins with a training accuracy of approximately 49%, which improves to around 92% after 10 epochs. The validation accuracy starts around 60% and reaches approximately 71% after 10 epochs. This model also shows substantial accuracy improvement during training, indicating effective learning.

The Sigmoid model, in contrast, has a much lower starting training accuracy, around 10%, and only reaches approximately 57% after 10 epochs. The validation accuracy starts at a similar low point and reaches around 55% after 10 epochs. The Sigmoid model struggles to learn effectively compared to the other models.

The ReLU and Leaky ReLU models outperform the Sigmoid model in terms of accuracy and learning speed. This is a common observation in deep learning as ReLU-based activations tend to mitigate the vanishing gradient problem. Leaky ReLU exhibits a slight advantage over ReLU, likely due to its ability to handle negative inputs during training. The Sigmoid model struggles to capture complex patterns in the CIFAR-10 dataset and converges to a lower accuracy. Overall, this code demonstrates the impact of different activation functions on model performance. ReLU and Leaky ReLU activations are more suitable for image classification tasks, while Sigmoid is

less effective. It highlights the importance of choosing the right activation function based on the problem domain.

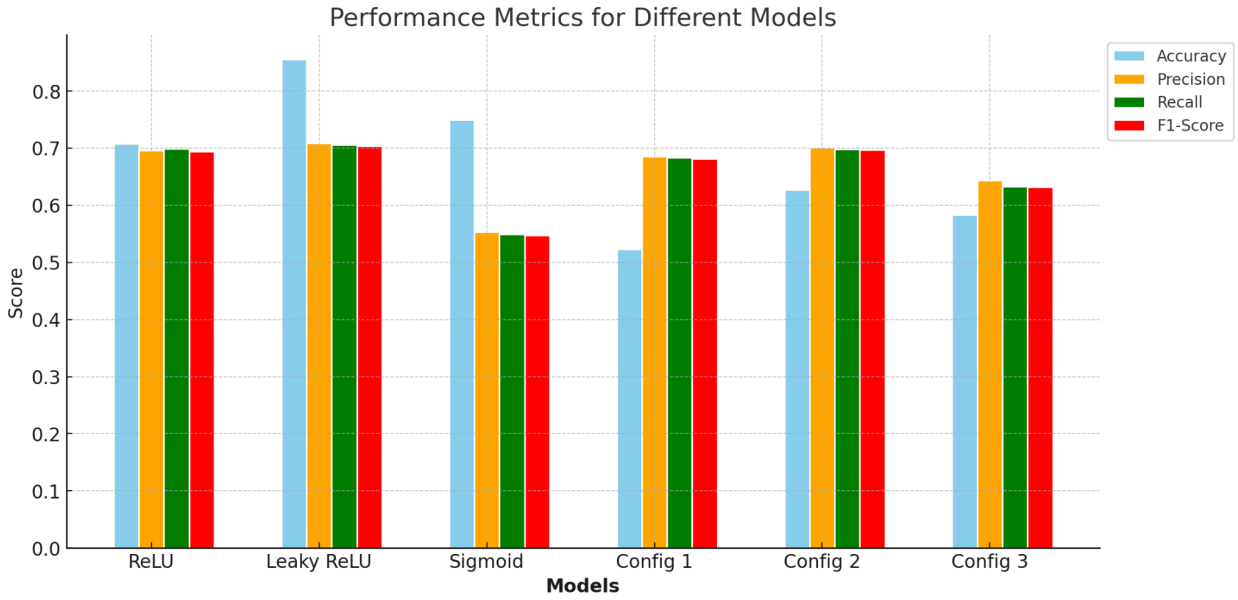
Next, we are going to evaluate multiple machine learning models using the Scikit-Learn library's classification_report function. These models include CNNs with different activation functions and convolutional layer configurations, and they are evaluated on the validation set of the CIFAR-10 dataset. To define Evaluation Function first it will take a trained model for validation data (X_val and y_val). Then predicts class labels for the validation data using the model. It will compute a classification report, including metrics such as precision, recall, F1-score, and support, using the predicted and true labels. It will create a list of models to evaluate. This list includes models with ReLU, Leaky ReLU, and Sigmoid activations, as well as models with different convolutional layer configurations (referred to as model_config_1, model_config_2, and model_config_3). A loop iterates through each model, evaluates it using the evaluate_model function, and appends the resulting classification report to a list (reports). The report provides detailed insights into the performance of each model in terms of precision, recall, F1-score, and support for each class (in this case, the 10 classes of the CIFAR-10 dataset).

We know that accuracy is the percentage of images that were correctly classified. Similarly, precision is the percentage of images that were classified as positive that were actually positive and recall is the percentage of positive images that were correctly classified. Lastly, F1-score is a measure of accuracy that takes into account both precision and recall.

The best model in terms of accuracy is the ReLU model, with an accuracy of 0.6977. The leaky ReLU model has a slightly lower accuracy of 0.7053, and the sigmoid model has the lowest accuracy of 0.5497. In terms of precision, the ReLU model also has the best performance, with a precision of 0.696468. The leaky ReLU model has a precision of 0.708274, and the sigmoid

model has a precision of 0.553093. The recall is similar to the accuracy, with the ReLU model having the best performance, followed by the leaky ReLU model and the sigmoid model.

The F1-score is also similar to the accuracy and recall, with the ReLU model having the best performance, followed by the leaky ReLU model and the sigmoid model. Overall, the ReLU activation function achieves the best performance on this dataset. However, it is important to note that the results may vary depending on the dataset and the hyperparameters of the model.



Models based on ReLU tend to do well because they are good at dealing with the vanishing gradient problem. However, they could suffer from dead neurons. To combat this, Leaky ReLU implements a negligible gradient for negative values. The model's feature-capturing power will change depending on the particular convolutional configuration used. The larger the kernel, the more detail it can catch, but the more it can also miss. While adding more filters can improve feature capture, it can also increase model complexity and lead to overfitting. The number of

strides used to move the kernel around the picture. With a stride of 2, the number of spatial dimensions is cut in half, allowing for speedier computation at the expense of possible data loss.

Model complexity and efficiency both need to be considered. Complex models can benefit from the use of regularisation strategies like dropout and weight decay to avoid overfitting. This allows for a thorough analysis of how various adjustments affect CNN efficiency. After carrying out these procedures and collecting data, a person will be able to determine which architecture and configurations are ideal for the CIFAR-10 dataset in the particular environment.

Among ReLU's many benefits are its speed, the fact that it avoids the dreaded vanishing gradient, and the fact that it is extensively applied. The negative part is that it has unboundedness and the possibility of "dead neurons," in which certain nerve cells never fire. Leaky Fast processing speed, fixing the "dead neuron" problem of ReLU. it’s weakness is that the best setting for the leaky parameter may need to be adjusted for different datasets. For sigmoid, powers Constrained to lie between 0 and 1, typically employed for two-class situations. The Limitations is that the vanishing gradient problem may arise, especially with more complex networks. Not centred on zero.

The observed trend in the evolution is that adding more filters to improve feature capture raises in the computational bar. Loss of fine detail is a potential effect of using a larger kernel size. Lower spatial resolution and potential data loss are offset by the increased computational speed afforded by larger strides. The trends seen so far is that, sometimes the same speed gains as when adding more layers can be achieved by increasing the number of filters (making the network wider), but with less computing requirements. In case of overfitting, the likelihood of overfitting rises together with the depth and breadth of a model's complexity. The importance of regularisation methods rises.

It is possible for deeper networks, and in particular those using specific activation functions, to have the vanishing/exploding gradient problem. Training efficiency and consistency are both impacted by this. The setups that excel on one dataset may not be the best on another. Experimentation and fine-tuning based on the particular dataset and problem at hand is essential.

Through our experiments, we observed that ReLU and its variant, Leaky ReLU, tend to outperform sigmoid for deeper architectures, primarily due to their ability to mitigate the vanishing gradient problem. However, careful consideration is needed as ReLU can sometimes lead to dead neurons. Adjusting configurations like the number of filters, kernel sizes, and strides can significantly affect model performance. While increasing the number of filters can improve feature capture, it also increases the risk of overfitting and computational cost. Similarly, kernel size and stride adjustments have trade-offs related to feature granularity and computation speed. Overall, deep learning and CNNs, in particular, are more of an art combined with science. While theoretical understanding provides a foundation, empirical evaluation and iterative experimentation often lead to optimal solutions. We need to explore newer activation functions like Swish, Mish, or variants of ReLU like Parametric ReLU. Also, conducting experiments on more complex datasets like CIFAR-100, ImageNet, or domain-specific datasets to understand the robustness and scalability of the chosen architectures.

References:

https://pyimagesearch.com/2021/05/14/convolutional-neural-networks-cnns-and-layer-types/

https://www.mdpi.com/1424-8220/22/16/6129

https://ieeexplore.ieee.org/abstract/document/7536654