# From Pixels to Predictions: Building a Deep Learning Model - Transcript

## Introduction

Imagine teaching a machine to recognize the difference between a truck and a ship or a cat and a dog—all from tiny pixel images. In this presentation, I will walk you through the journey of building a deep learning model for image classification using the CIFAR-10 dataset (Krizhevsky and Hinton, 2009, pp. 5–6). From choosing the right framework to designing an efficient network architecture and fine-tuning its performance, this project was both a challenge and a learning experience. Along the way, I uncovered key insights into what makes neural networks work—and what holds them back (LeCun, Bengio, and Hinton, 2015). Let's dive into the foundation of this work.

---

## CIFAR-10

Meet CIFAR-10: a small yet mighty dataset that has become a playground for machine learning enthusiasts and researchers alike. With 60,000 colorful images, each just 32x32 pixels, it challenges models to classify objects across 10 distinct categories, including vehicles, animals, and more (Krizhevsky and Hinton, 2009, pp. 5–6).

What makes this dataset so fascinating is its simplicity paired with complexity. The small image size forces models to extract meaningful patterns, while subtle distinctions—like a cat versus a dog—test the network's ability to generalize. It is perfectly balanced, with each class contributing 6,000 images, pre-split into 50,000 for training and 10,000 for testing (Krizhevsky and Hinton, 2009, pp. 5–6).

In this project, CIFAR-10 was not just a dataset; it was a test of innovation. Preprocessing steps like normalization and data augmentation transformed this compact dataset into a canvas for exploration. The challenge of teaching a model to understand this pixelated world provided invaluable insights into feature extraction, model tuning, and evaluation (Shorten and Khoshgoftaar, 2019).

Though seemingly small, it has played a significant role in driving breakthroughs in image classification—and is the perfect starting point for this project's deep learning adventure (LeCun, Bengio, and Hinton, 2015).

---

**Framework Choice**

When considering which framework to use for this project, I compared three popular options: Keras, PyTorch, and TensorFlow with Keras. Each offered unique advantages and trade-offs. Keras, with its clean and simple API, was particularly appealing for its ease of use, making it a strong contender for rapid experimentation (Medavarapu, 2024). PyTorch stood out for its flexibility, thanks to dynamic computation graphs that allowed for real-time changes to the network's structure (Dai et al., 2022). This was especially useful for exploring more complex, custom models that might require non-standard layer configurations. However, when I started building my model with PyTorch, I found the syntax more confusing than expected, which made it harder to track down issues and understand why certain errors occurred.

Ultimately, I chose TensorFlow with Keras because it provided a balance between ease of use and powerful features. TensorFlow's robust backend allowed me to integrate data augmentation and implement learning rate schedules, while Keras's high-level interface

made designing and iterating on the model architecture more efficient (Wang et al., 2024). This approach provided the flexibility to develop a high-performing network while avoiding unnecessary focus on low-level implementation details, ensuring alignment with the project's objectives.

---

**An Explanation of Network Architecture**

The architecture I designed is structured to efficiently capture and process visual features from the dataset. It begins with three convolutional layers, which form the backbone of the feature extraction process (Krizhevsky, Sutskever, and Hinton, 2012). These layers are supported by pooling and Dropout, ensuring both dimensionality reduction and regularization to prevent overfitting (Srivastava et al., 2014).

The first layer uses 32 filters with a kernel size of 3x3. This layer detects fundamental visual elements such as edges and textures. It's followed by a MaxPooling layer, which halves the spatial dimensions, effectively focusing the network's attention on the most important features. A Dropout layer is applied, deactivating 20% of the neurons at random during training (Srivastava et al., 2014). This introduces randomness and forces the model to learn more robust representations.

The second layer increases the filter count to 64, allowing the network to capture more detailed patterns, such as shapes and contours. A kernel constraint, MaxNorm, is used to ensure weight stability, while another pooling layer continues to reduce the spatial dimensions, enabling the network to maintain computational efficiency (Brownlee, 2019). This setup ensures that the features learned in earlier layers are effectively refined and preserved.

The third convolutional layer expands the filters to 128. This layer focuses on abstract, high-level features that help differentiate the dataset's classes (Krizhevsky, Sutskever, and Hinton, 2012). It's followed by a Dropout layer at 30%, which further improves generalization (Srivastava et al., 2014). By progressively increasing the number of filters and applying regularization at each stage, the network becomes better at extracting and refining meaningful patterns.

The extracted features are then flattened into a one-dimensional vector, which is passed to a dense layer with 256 neurons. This dense layer combines all learned features, making the final decisions more robust and informed. Finally, the output layer, using a softmax activation function, generates probabilities for each of the 10 classes in the dataset (Murugan, 2018). This structure balances complexity and efficiency, enabling the model to generalize well while remaining practical to train.

---

**The Role of Activation and Loss Functions**

To ensure efficient learning, I carefully selected activation and loss functions. ReLU (Rectified Linear Unit) was used in the convolutional and dense layers. This activation function is computationally efficient and helps mitigate the vanishing gradient problem. By outputting zero for negative values and scaling positive values linearly, ReLU enables faster training and the ability to learn complex patterns. It's particularly effective in deeper networks where other activation functions might struggle to maintain gradient strength (Asadi and Jiang, 2020).

For the output layer, I used the softmax activation function. Softmax transforms the network's raw outputs into a probability distribution, where the sum of all probabilities

equals 1. This makes it ideal for multi-class classification tasks. With softmax, the model can assign a confidence score to each class, allowing for more interpretable and reliable predictions (Franke and Degen, 2023).

The loss function I chose was categorical cross-entropy. This is a standard choice for multi-class classification problems. It measures the difference between the predicted probability distribution and the true one-hot encoded labels. By minimizing categorical cross-entropy, the model learns to assign higher probabilities to the correct classes, improving overall accuracy. Together, ReLU, softmax, and categorical cross-entropy provide a solid foundation for effective training (Mao, Mohri, and Zhong, 2023).

---

**Data Partitioning**

In any machine learning project, the way data is split plays a crucial role in determining the model's ability to generalize (Hebart and Vicente, 2016). Initially, I divided the dataset into an 80-20 split: 80 percent for training and 20 percent for validation. This approach provided a large enough training set to learn meaningful patterns while reserving a substantial portion of data for validation. Since CIFAR-10 is perfectly balanced across its ten categories, stratified splitting was unnecessary, as a random split naturally preserved class balance and ensured fair evaluation.. (Xu and Goodacre, 2018).

I also experimented with k-fold cross-validation. This technique trains the model on different subsets of the data, providing a more robust measure of generalization (Hebart and Vicente, 2016). While thorough, it proved to be computationally expensive, consuming all my free GPU credits on Google Colab without offering significant

improvements over the simpler split. Smaller validation sets were tested as well, but they led to instability and overfitting after just 18 epochs. In the end, the 80-20 split provided the most reliable and efficient approach, ensuring that the model had both enough training data and a representative validation set (Xu and Goodacre, 2018).

---

**Preprocessing Steps**

To ensure the data was ready for training, I applied several preprocessing steps. First, I normalized the pixel values, scaling them from their original range of 0–255 down to a range of 0.0–1.0. This consistent scaling stabilized the training process by ensuring that all input features were on the same scale. Normalization also reduced the risk of large gradients, helping the model converge more quickly and effectively (Huang et al., 2020).

Next, I transformed the class labels into one-hot encoded vectors. This step was necessary for compatibility with the categorical cross-entropy loss function. One-hot encoding allowed the network to output probabilities for each class and compare them directly to the true labels. Without this step, the model's loss calculations would have been less straightforward, making training less efficient (Mao, Mohri, and Zhong, 2023).

In addition to these standard preprocessing techniques, I incorporated data augmentation. This involved introducing random rotations, horizontal and vertical shifts, and horizontal flips. By artificially expanding the training data in this way, the model was exposed to more variations, helping it learn features that generalized better to unseen data (Shorten and Khoshgoftaar, 2019). These preprocessing steps collectively provided a stable foundation for training and improved the model's robustness .

**Training Process**

Training the network required careful parameter tuning. I started with 25 epochs, a learning rate of 0.1, and a batch size of 32. The batch size was chosen because it balanced memory efficiency and stable gradient updates. Smaller batches could have introduced more noise, while larger batches might have slowed convergence. With a batch size of 32, the model was able to update its weights efficiently and make consistent progress (Granziol, Zohren, and Roberts, 2022).

As the training proceeded, I increased the number of epochs to 50 to give the network more time to learn. However, early stopping triggered after 30 epochs, as the validation loss began fluctuating. This indicated that the high learning rate was causing instability, potentially preventing the network from converging fully. Early stopping is known to help mitigate overfitting by halting training when the model's performance on the validation set starts to degrade (Ji, Li, and Telgarsky, 2021)

To address this, I lowered the learning rate to 0.01 and kept a gradual decay schedule (Smith, 2017). By making the updates more gradual, the model became more stable. I then extended training to 75 epochs, providing the network with enough time to refine its weights and improve its performance. Eventually, early stopping halted training at 66 epochs, after it had previously stopped at 55, indicating that the model had reached a stable solution. This careful adjustment of parameters ensured that the training process was both efficient and effective.

**Validation Strategy**

Throughout the training process, I used the validation set to monitor the model's performance. After each epoch, I evaluated validation accuracy and loss. This continuous monitoring allowed me to identify when the model started to overfit or when its generalization capabilities began to plateau (Mahsereci et al., 2017).

Early stopping played a key role in my validation strategy. By setting a patience level, I ensured that if the validation loss did not improve for five consecutive epochs, training would stop early. This not only prevented overfitting but also saved time and resources. Additionally, by restoring the best weights observed during training, I guaranteed that the final model represented its peak performance on the validation set. These validation strategies were essential in refining the model's training process (Vilares Ferro et al., 2024).

---

**Testing and Results**

The model achieved a test accuracy of 75.62%, correctly classifying three-quarters of the images. However, accuracy alone does not tell the full story. To really understand how well the model was doing, I looked at the F1 score. It's a metric that balances two things: precision, which tells us how accurate the predictions were, and recall, which shows how well the model picked up all the important instances. This provided a deeper and more meaningful evaluation (Tsoi et al., 2020).

For example, Class 1 performed well with an F1 score of 0.88, driven by a precision of 0.87 and recall of 0.90, showing the model's strong ability to reliably identify this category. On the other hand, dogs and cats, which are Classes 3 and 5, had lower F1

scores. This suggests the model had a harder time with these categories, which makes sense since they look quite similar. (Bénédict et al., 2021).

The overall weighted F1 score of 0.7539 reflects balanced performance across categories and reveals areas for growth. By focusing on F1, I identified key improvements, such as refining the architecture or applying targeted data augmentation, to strengthen future iterations (Ahmadian et al., 2024).

---

**Critical Analysis**

While the model performed reasonably well, it also revealed some challenges and limitations. Certain classes consistently scored higher in precision and recall, while some showed weaker results. This imbalance suggested that additional targeted data collection or more specialized augmentation could improve the model's generalization in these areas (Buda et al., 2018).

Overfitting remained a concern despite the use of early stopping and Dropout layers. The validation loss occasionally fluctuated, indicating that the model might still be memorizing patterns. Applying stronger regularization, such as weight decay, or experimenting with more advanced architectures could mitigate these issues (Srivastava et al., 2014).

Finally, the relatively simple architecture of the network limited its ability to capture complex patterns. Exploring deeper networks or integrating pre-trained models could enhance performance and lead to more consistent results (He et al., 2016).

---

**9. Insights and Reflections**

This project was not just about designing a neural network—it was a journey full of challenges and valuable lessons. Running out of GPU limits while experimenting with k-fold cross-validation and complex architectures forced me to rethink my approach. I realized that simplicity is often better; overcomplicating models can waste resources without significant improvements (Bergman et al., 2024). Another challenge was troubleshooting accuracy and loss plots, which initially only showed data for 5 epochs instead of the full 75. This slowed progress but reinforced the importance of debugging and paying attention to every detail (Schneider et al., 2021).

Despite these obstacles, I gained a deeper understanding of how architectural choices and regularization techniques influence performance. Early stopping and validation monitoring helped avoid overfitting, while experimenting with learning rates and data augmentation taught me how to optimize training effectively (Bergman et al., 2024).

Ultimately, this project showed me the importance of balancing theoretical knowledge with practical problem-solving, and the lessons learned will guide me in future projects, reminding me that progress comes from persistence and adaptability. These lessons will guide me in future projects, reminding me that progress often comes from persistence and adaptability.

---

**Application to Real-World Problems**

The lessons from this project extend far beyond the classroom. Imagine a medical AI identifying critical abnormalities on an X-ray, where missing a detail could cost lives. Insights into precision and recall gained here play a key role in training such systems

(Chen et al., 2024). Or picture an autonomous car safely navigating a rainy night. The data augmentation strategies explored here help models adapt to unpredictable scenarios like these (Tong et al., 2023).

By scaling these methods to larger datasets or incorporating pre-trained models, we can develop more intelligent, adaptive AI systems. This project laid the foundation for solving real-world problems, one pixel at a time.

---

**Conclusion**

Building this model was a journey of empowering a machine to learn from data and make meaningful decisions. Working with CIFAR-10's tiny images pushed the limits of creativity and problem-solving, from selecting the best framework to crafting an architecture that balances power and simplicity (Krizhevsky, 2009). Along the way, I learned how to turn challenges—like overfitting and class imbalances—into valuable learning opportunities. While 75.62% accuracy is a solid achievement, the real victory lies in the insights gained and the doors opened to more ambitious challenges ahead. This is just the beginning of what's possible with deep learning!

**References:**

Ahmadian, R., Ghatee, M., and Wahlström, J., (2024). *Superior Scoring Rules for Probabilistic Evaluation of Single-Label Multi-Class Classification Tasks.* arXiv preprint arXiv:2407.17697. Available at: https://arxiv.org/abs/2407.17697 (Accessed: 20 January 2025)

Asadi, B. and Jiang, H., (2020). *On Approximation Capabilities of ReLU Activation and Softmax Output Layer in Neural Networks*. arXiv preprint arXiv:2002.04060. Available at: https://arxiv.org/abs/2002.04060 (Accessed: 20 January 2025).

Bénédict, G., Koops, V., Odijk, D., and de Rijke, M., (2021). *sigmoidF1: A Smooth F1 Score Surrogate Loss for Multilabel Classification*. arXiv preprint arXiv:2108.10566. Available at: https://arxiv.org/abs/2108.10566 (Accessed: 20 January 2025)

Bergman, E., Purucker, L., and Hutter, F., (2024). *Don't Waste Your Time: Early Stopping Cross-Validation*. arXiv preprint arXiv:2405.03389. Available at: https://arxiv.org/abs/2405.03389 (Accessed: 20 January 2025).

Brownlee, J., (2019). *A Gentle Introduction to Pooling Layers for Convolutional Neural Networks. Machine Learning Mastery*. Available at: https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/ (Accessed: 20 January 2025).

Buda, M., Maki, A., and Mazurowski, M.A., (2018). *A systematic study of the class imbalance problem in convolutional neural networks.* Neural Networks, 106, pp.249–259. [online] Available at: https://arxiv.org/abs/1710.05381 (Accessed: 20 January 2025).

Chen, H., Dan, L., Lu, Y., Chen, M., and Zhang, J., (2024). *An improved data augmentation approach and its application in medical named entity recognition*. BMC Medical Informatics and Decision Making, 24, Article number: 221. Available at: https://bmcmedinformdecismak.biomedcentral.com/articles/10.1186/s12911-024-02624-x (Accessed: 20 January 2025).

Dai, H., Peng, X., Shi, X., He, L., Xiong, Q. and Jin, H., (2022). *Reveal training performance mystery between TensorFlow and PyTorch in the single GPU environment*. Science China Information Sciences, 65(11), Article 112103. Available at: https://link.springer.com/article/10.1007/s11432-020-3182-1 (Accessed: 20 January 2025).

Franke, M. and Degen, J., (2023). *The Softmax Function: Properties, Motivation, and Interpretation*. Available at: https://alpslab.stanford.edu/papers/FrankeDegen_submitted.pdf (Accessed: 20 January 2025).

Granziol, D., Zohren, S., & Roberts, S., (2022). *Learning rates as a function of batch size: A random matrix theory approach to neural network training*. Journal of Machine Learning Research, 23(173), pp.1–65. Available at: https://jmlr.org/papers/v23/20-1258.html (Accessed: 20 January 2025).

He, K., Zhang, X., Ren, S., and Sun, J., (2016). *Deep residual learning for image recognition*. In: Proceedings of the IEEE Conference on Computer Vision and Pattern

Recognition (CVPR), pp.770–778. Available at: https://arxiv.org/abs/1512.03385 (Accessed: 20 January 2025).

Hebart, M.N. and Vicente, R., (2016). *An Efficient Data Partitioning to Improve Classification Performance While Keeping Parameters Interpretable*. PLOS ONE, 11(8), p.e0161788. Available at: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0161788 (Accessed: 20 January 2025)

Huang, L., Yang, D., Luo, Z., Xu, D. and Lu, H., (2020). *Normalization Techniques in Training Deep Neural Networks: A Survey*. arXiv preprint arXiv:2009.12836. Available at: https://arxiv.org/pdf/2009.12836 (Accessed: 20 January 2025).

Ji, Z., Li, J.D., and Telgarsky, M., (2021). Early-stopped neural networks are consistent. *Advances in Neural Information Processing Systems*, 34, pp. 1–13. Available at: https://proceedings.neurips.cc/paper/2021/file/0e1ebad68af7f0ae4830b7ac92bc3c6f-Paper.pdf (Accessed: 20 January 2025).

Krizhevsky, A. & Hinton, G., (2009). *Learning multiple layers of features from tiny images*. Available at: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf (Accessed 20 January 2025)

Krizhevsky, A., Sutskever, I., & Hinton, G.E., (2012). *ImageNet classification with deep convolutional neural networks*. Advances in Neural Information Processing Systems, 25, pp.1097–1105. Available at: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf (Accessed: 20 January 2025).

LeCun, Y., Bengio, Y., & Hinton, G. (2015). *Deep learning*. Nature, 521(7553), pp. 436–444. Available at: https://doi.org/10.1038/nature14539 (Accessed: 20 January 2025)

Mahsereci, M., Balles, L., Lassner, C., and Hennig, P., (2017). *Early Stopping Without a Validation Set. arXiv preprint arXiv:1703.09580.* Available at: https://arxiv.org/abs/1703.09580 (Accessed: 20 January 2025).

Mao, A., Mohri, M. and Zhong, Y., (2023). *Cross-Entropy Loss Functions: Theoretical Analysis and Applications*. arXiv preprint arXiv:2304.07288. Available at: https://arxiv.org/abs/2304.07288 (Accessed: 20 January 2025).

Medavarapu, S.S., (2024). *Advancements in Deep Learning: A Review of Keras and TensorFlow Frameworks*. Journal of Scientific and Engineering Research, 11(5), pp. 282–286. Available at: https://jsaer.com/download/vol-11-iss-5-2024/JSAER2024-11-5-282-286.pdf (Accessed: 20 January 2025).

Murugan, P., (2018). *Implementation of Deep Convolutional Neural Network in Multi-class Categorical Image Classification*. arXiv preprint arXiv:1801.01397. Available at: https://arxiv.org/abs/1801.01397 (Accessed: 20 January 2025)

Schneider, F., Dangel, F., and Hennig, P., (2021). *Cockpit: A Practical Debugging Tool for the Training of Deep Neural Networks. In: Proceedings of the 35th Conference on Neural Information Processing Systems (NeurIPS 2021).* Available at: https://proceedings.neurips.cc/paper/2021/hash/ae3539867aaeec609a4260c6feb725f4-Abstract.html (Accessed: 20 January 2025).

Shorten, C. & Khoshgoftaar, T.M., (2019). *A survey on Image Data Augmentation for Deep Learning.* Journal of Big Data, 6(1), p.60. Available at: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0197-0 (Accessed: 20 January 2025).

Smith, L.N., (2017). *Cyclical learning rates for training neural networks*. *In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 4641–4650. [online] Available at: https://arxiv.org/abs/1506.01186 (Accessed: 20 January 2025)

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R., (2014). *Dropout: A simple way to prevent neural networks from overfitting*. Journal of Machine Learning Research, 15(56), pp.1929–1958. Available at: http://jmlr.org/papers/v15/srivastava14a.html (Accessed: 20 January 2025).

Tong, W., Xie, J., Li, T., Deng, H., Geng, X., Zhou, R., Yang, D., Dai, B., Lu, L., Li, H., (2023). *3D Data Augmentation for Driving Scenes on Camera*. In: Lecture Notes in Computer Science, 13807, pp. 45–56. Available at: https://link.springer.com/chapter/10.1007/978-981-97-8508-7_4 (Accessed: 20 January 2025).

Tsoi, N., Candon, K., Li, D., Milkessa, Y., and Vázquez, M., (2020). Bridging the Gap: Unifying the Training and Evaluation of Neural Network Binary Classifiers. *arXiv preprint arXiv:2009.01367*. Available at: https://arxiv.org/abs/2009.01367 (Accessed: 20 January 2025).

Vilares Ferro, M., Doval Mosquera, Y., Ribadas Pena, F.J., and Darriba Bilbao, V.M., (2024). Early Stopping by Correlating Online Indicators in Neural Networks. *arXiv preprint arXiv:2402.02513*. Available at: https://arxiv.org/abs/2402.02513 (Accessed: 20 January 2025)

Wang, Z., Liu, K., Li, J., Zhu, Y. and Zhang, Y., (2024). *Various Frameworks and Libraries of Machine Learning and Deep Learning: A Survey.* Archives of Computational Methods in Engineering, 31, pp. 1–24. [online] Available at: https://link.springer.com/article/10.1007/s11831-018-09312-w (Accessed: 20 January 2025).

Xu, Y., Goodacre, R., (2018(. *On Splitting Training and Validation Set: A Comparative Study of Cross-Validation, Bootstrap and Systematic Sampling for Estimating the Generalization Performance of Supervised Learning*. Journal of Analysis and Testing, 2(3), pp.249–262. Available at: https://link.springer.com/article/10.1007/s41664-018-0068-2 (Accessed: 20 January 2025).

Link to code:

Machine-Learning-module/CNN.ipynb at main · dzervenes/Machine-Learning-module · GitHub