**Neural Network Models for Object Recognition Using Classical ML (Track 1)**

**Slide 1: Introduction**  
Good morning, everyone. Today, I’m going to walk you through the results we achieved using classical machine learning techniques to recognize objects from images. Specifically, we worked with the CIFAR-10 dataset and applied Principal Component Analysis (PCA) and Support Vector Machines (SVMs). This work falls under Track 1 of our study.

**Slide 2: Project Overview**  
To give you a clear picture of what we’re aiming for, let me summarize our objective.  
Our main goal is to use traditional machine learning methods—rather than deep learning—to recognize objects in the CIFAR-10 dataset. Now, I know many of you might be thinking: “But isn’t deep learning the go-to solution for computer vision nowadays?”

And that’s a valid point. But that’s exactly why our approach is interesting.  
We wanted to see how well standard methods like PCA and SVM perform in this space. This gives us a baseline to understand what classical techniques can still offer—and how far deep learning has truly taken us.

It’s also important to remember that traditional methods still have their value. They’re useful when we have limited computational resources, when we need models that are easier to interpret, or when we just want a quick prototype.

**Slide 3: About the Dataset**  
Now, let’s talk about the dataset we used—CIFAR-10.  
It’s one of the most commonly used benchmark datasets in computer vision. It contains 60,000 color images, each sized at 32 by 32 pixels. These images fall into 10 categories: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

Out of the 60,000 images, 50,000 are used for training and 10,000 for testing. We followed this standard split in our experiments.

One thing that makes CIFAR-10 challenging is the image resolution. At just 32x32 pixels, the images have very little detail. This helps level the playing field between simple and complex models, making it a great dataset for comparison.

**Slide 4: Data Preparation**  
Preparing the data is a crucial step, and here’s what we did:

First, we **normalized** all image pixel values to a range between 0 and 1. This step ensures that all features are on the same scale and helps our algorithms work more efficiently.

Next, we **flattened** each image. Since CIFAR-10 images are 32x32 pixels with 3 color channels (RGB), flattening each one gave us a 3,072-dimensional vector.

However, working with such high-dimensional data can be problematic. It increases computational cost and the risk of overfitting. So we used **Principal Component Analysis** to reduce the dimensionality to just 100 features—capturing most of the important information while simplifying the model.

Finally, we split our training set using an **80/20 split**—80% for training and 20% for validation. This leads us to a key topic: the importance of validation.

**Slide 5: Why Validation Is Essential**  
Validation plays a key role in machine learning for several reasons:

First, it helps us **detect overfitting**. Sometimes a model performs really well on the training data but fails on new, unseen data. Validation data acts as a checkpoint, showing us whether our model is learning general patterns or just memorizing.

Second, it’s essential for **tuning hyperparameters**. Models like SVMs and PCA have settings—like the number of components or regularization strength—that we must choose manually. By testing different combinations on the validation set, we can find the most effective ones.

Third, validation helps us **estimate real-world performance**. It simulates how the model might behave on future data, giving us more confidence in its ability to generalize.

Finally, it’s useful for **comparing models**. If we try out multiple algorithms or approaches, the one that performs best on the validation set is usually the one we’ll go forward with.

**Slide 6: Classical ML Approach**  
Now let’s talk about the core of our classical machine learning method. For this track, we intentionally avoided deep learning techniques. Our goal was to set a strong baseline using traditional methods.

Our process had two main steps. First, we reduced the number of features using **Principal Component Analysis (PCA)**. This technique transforms our original 3,072-dimensional image data into a much smaller set of features—just 100 in our case—while still keeping most of the important information.

Next, we used **Support Vector Machines (SVM)** for classification, specifically with a **Radial Basis Function (RBF) kernel**. SVMs are powerful because they find the best boundary between different classes. The RBF kernel is great for handling non-linear data—like images—because it maps data into a space where even complex patterns can be separated more easily.

We didn’t just pick our parameters randomly. Through experimentation, we found that a **C value of 10** gave a good balance between minimizing errors and preventing overfitting. We set **gamma to 0.01**, which controls how far each training point influences the model. The RBF kernel proved to be a good fit for the complex patterns in our image data.

**Slide 7: Model Details & Reasoning**  
Let’s dive a bit deeper into our model setup and why we chose these settings.

By reducing the features to 100 using PCA, we managed to retain about **95% of the original data variance**. That’s a solid result—it means we captured most of the meaningful patterns while simplifying the data significantly.

Why 100 components? It was a trade-off. We tried more, but it didn’t help much and only made things slower. Fewer components, on the other hand, led to worse performance.

As for the SVM with the RBF kernel, we chose it because **image classification is rarely linear**. The relationships between pixels and objects are often complex, and the RBF kernel allows the model to handle these complexities well.

It’s also worth noting that **classical ML works differently from deep learning**. We don’t train in batches or go through epochs. Instead, training with SVMs involves solving an optimization problem using **quadratic programming**—a very different process compared to gradient descent in neural networks.

**Slide 8: Accuracy Results**  
Now let’s look at how our model actually performed.

On the **validation set**, we achieved an accuracy of **54.85%**, and on the **test set**, we got **54.87%**. The similarity between the two results is a great sign—it shows that our model generalizes well and isn’t overfitting.

So, is 55% good? Well, if we were guessing randomly across 10 classes, we’d only get **10% accuracy**. So 55% is definitely better than chance—but of course, it's nowhere near what deep learning models can achieve on CIFAR-10, where accuracies of **90% or more** are common.

Still, our results highlight what classical methods can do, especially when computational resources are limited or quick results are needed.

**Slide 9: Confusion Matrix Insights**  
To better understand our model’s performance, we analyzed the **confusion matrix**. This matrix shows how often the model predicts each class correctly and where it makes mistakes.

From the matrix and its heatmap visualization, we could clearly see **which classes the model handles well** and which ones are more confusing. The diagonal values represent correct predictions, while off-diagonal ones show misclassifications.

This kind of breakdown is extremely helpful. It tells us not just how accurate the model is, but where exactly it struggles. For example, if two classes are frequently confused, we might look into improving feature extraction or tweaking the model for those specific cases.

**Slide 10: Class-by-Class Performance**  
Looking at class-level performance, we noticed some clear patterns.

Our model did **best with ships, trucks, and automobiles**. These objects have very distinctive shapes that stand out, even in low-resolution images.

But it **struggled with cats, dogs, and birds**. Why? Well, these are all animals that appear in many different poses and angles. In a tiny 32x32 image, it’s hard to see the fine details—like fur texture or facial features—that would help us tell them apart.

Cats and dogs, for instance, are visually similar—both are four-legged mammals with similar body shapes. And birds come in many forms and can appear flying or sitting, which adds to the complexity.

This unveils one of the main shortcomings of the classical methods such as PCA. Although PCA encode the statistical variation in the data, it is not always interested in visual aspects of importance to people, such as eyes, tails, or posture. It does not work to comprehend pictures as it does in contemporary deep learning.

**Slide 11: Classical ML vs. Deep Learning**  
So now we compare our classical machine learning approach and deep learning.

A number of features of classical ML which are worth mentioning include:

Speed: An SVM trained on PCA-dimension-reduced features is much faster to train, compared to deep convolutional neural networks. This attributes to the fact that classical techniques are excellent in terms of rapid experiments and prototyping.

Interpretability: Classical models are usually more interpretable. we may use the first factors in PCA as a way of seeing which features are most important and plot the decision boundaries of SVM. This type of transparency is significant when we are required to explicate the decision making process of a model.

Low Resource Demands: Our PCA + SVM pipeline does not need GPUs or high performance computing. It can be used on normal computers, and this is perfect where the teams or researchers have limited funds.

But then, this also has its limitations:

Classical ML does not have auto feature extraction capability. PCA does not necessarily represent visual patterns of a particular nature, such as shapes or textures, rather than statistically different patterns.

Here, deep learning, in particular, Convolutional Neural Networks (CNNs) is excellent. The CNNs learn features in a series: simple, low-level features such as edges and simple motives are learned in the lower layers and more complex features such as shapes or textures are learned in deeper layers. This enables CNNs to comprehend visual contents by far than the PCA-based methods.

**Slide 12: Key Takeaways**  
Throughout this project, we learned some valuable lessons that go beyond just numbers.

**First**, feature selection is critical. Reducing our input space from 3,072 to 100 dimensions using PCA made training feasible and helped avoid overfitting. Without PCA, the model would have struggled with both computation and performance.

**Second**, classical machine learning provides a **solid baseline**. Our ~55% accuracy might not be cutting-edge, but it gives us a reference point. Any model we develop after this should aim to beat it.

**Third**, we saw that while **SVMs are theoretically strong**, they can struggle with high-dimensional image data. Even after using PCA, the features may still not be ideal for classification.

And **finally**, we saw the classic trade-off: **speed vs. accuracy**. Classical ML is fast and resource-friendly but less accurate. Deep learning is more powerful—but slower and more resource-intensive. Choosing the right approach depends on your specific goals and constraints.

**Slide 13: Practical Applications**  
So what does this all mean in practice?

In situations where computing power is limited—like on embedded devices or in edge computing—**classical methods may still be very useful**. They don’t need expensive hardware and can be deployed quickly.

Also, for **rapid prototyping**, classical ML is a great starting point. If you’re exploring a new dataset or problem, it’s often helpful to test a classical approach first. This gives you a sense of how challenging the task is and whether investing in deep learning is worth it.

However, when accuracy is the top priority—as in production systems—**deep learning becomes the better choice**. The extra time and resources it demands are usually justified by the performance gains.

**Slide 14: Final Thoughts**  
To wrap up: our experiment using classical machine learning on the CIFAR-10 dataset gave us a strong foundation. Using PCA for feature reduction and SVM for classification, we reached an accuracy of around **55%**.

This shows that **yes, classical methods can still be applied** to image recognition tasks. They’re fast, easy to interpret, and require minimal resources—making them excellent for early-stage work or constrained environments.

But when it comes to high-stakes, high-accuracy applications, **deep learning remains the clear winner**. CNNs, with their ability to learn layered, meaningful features, are far better suited for handling the complexities of image data.

In the end, it’s not about choosing one over the other. It’s about **understanding the strengths and limitations of both approaches**, and knowing when to use which tool.

**Slide 15:**

Thank you for listening—and I’d be happy to answer any questions about our methods, results, or the broader comparison between classical and deep learning techniques.

## **References:**

* + Krizhevsky, A. (2009) *Learning Multiple Layers of Features from Tiny Images*. CIFAR-10 dataset creators. Available at: <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf> (Accessed: July 14, 2025).
  + Momotov, A. (2025) *Image Classification with SVM and PCA on CIFAR‑10*. GitHub. Available at: <https://github.com/Alex-Momotov/image_classification> (Accessed: July 14, 2025). [Wikipedia+1GitHub+1](https://en.wikipedia.org/wiki/CIFAR-10?utm_source=chatgpt.com)[GitHub](https://github.com/Alex-Momotov/image_classification?utm_source=chatgpt.com)
  + Miczek, D. (2025) *Implementing PCA and Dual PCA on CIFAR‑10*. Available at: <https://dmicz.github.io/machine-learning/cifar10-pca/> (Accessed: July 14, 2025). [Dennis Miczek](https://dmicz.github.io/machine-learning/cifar10-pca/?utm_source=chatgpt.com)
  + LemuelKL (2025) *CIFAR10‑HOG‑PCA‑SVM: Grayscaling, HOG, PCA, RBF SVM classifier achieving ~62%*. GitHub. Available at: <https://github.com/LemuelKL/CIFAR10-HOG-PCA-SVM> (Accessed: July 14, 2025). [GitHub](https://github.com/LemuelKL/CIFAR10-HOG-PCA-SVM?utm_source=chatgpt.com)
  + Fournier, Q. and Aloise, D. (2021) ‘Empirical comparison between autoencoders and traditional dimensionality reduction methods’, *arXiv preprint arXiv:2103.04874*. Available at: <https://arxiv.org/abs/2103.04874> (Accessed: July 14, 2025). [arXiv](https://arxiv.org/abs/2103.04874?utm_source=chatgpt.com)
  + Tang, Y. (2013) ‘Deep Learning using Linear Support Vector Machines’, *arXiv preprint arXiv:1306.0239*. Available at: <https://arxiv.org/abs/1306.0239> (Accessed: July 14, 2025). [arXiv](https://arxiv.org/abs/1306.0239?utm_source=chatgpt.com)
  + Wikipedia contributors (2025) *Support vector machine*. Available at: <https://en.wikipedia.org/wiki/Support_vector_machine> (Accessed: July 14, 2025).