The Dynamics of CNNs—Applications and (Simple) Theory

Max Lovig
Yale University

Physics x Data Science – 2025

What Features Do CNN's Learn

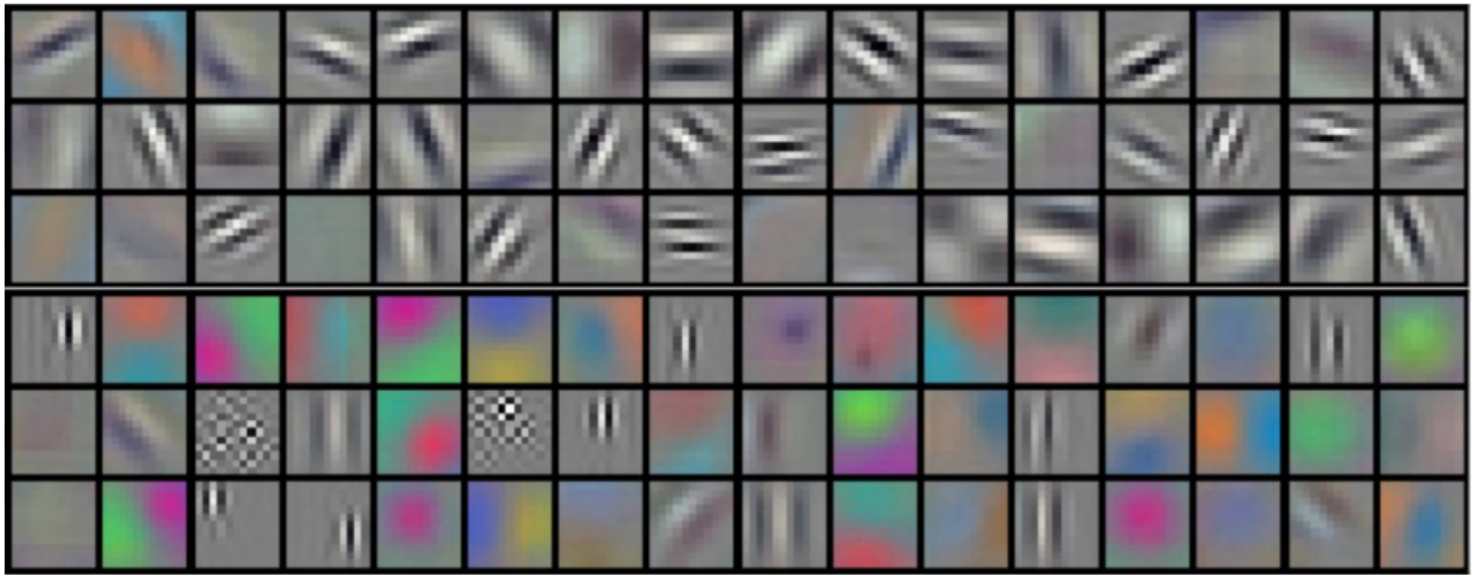


Figure: "96 convolutional kernels of size 11×11×3 learned by the first convolutional layer on the 224×224×3 input images." [KSH12]

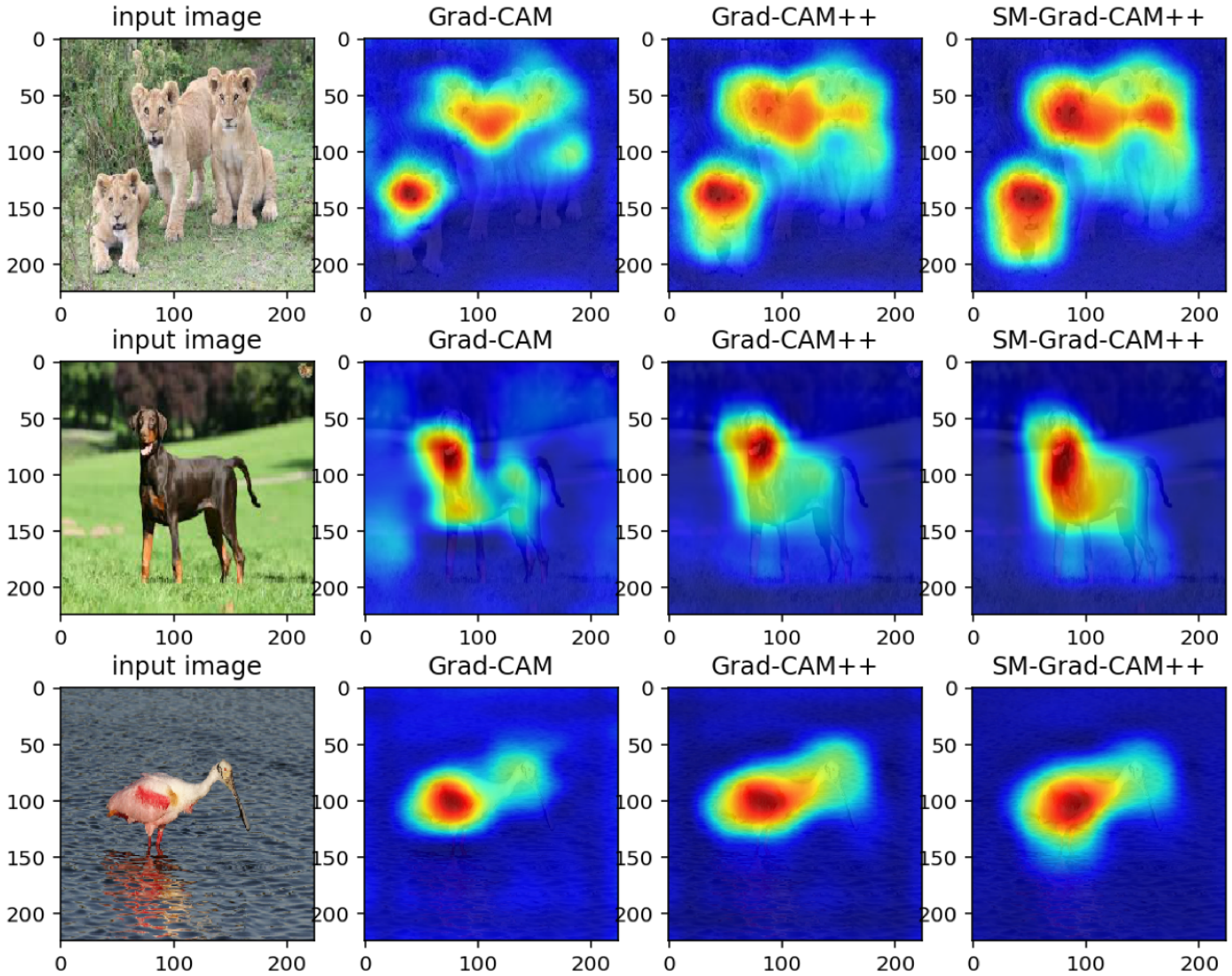


Figure: "Saliency maps generated using Grad-CAM, Grad-CAM++ and Smooth Grad-CAM++ respectively". [OSCW19]

How Do We Justify What Is Learned

Image Perturbation (RISE / LIME)[RSG16, PDS18]

Ablation Studies:

- Changing network features.
- Changing data input.
- Comparison between models and data features on final prediction methods.

Pros: Clean causal relationships.
Cons: Possibly very expensive, static in its comparison.

Motivation For Different Approach

1. Going beyond static comparisons
   - Is it possible to separate features learned at inference time?
   - Study the significance of features as they emerge.
2. Model agnostic, the method of analysis should not rely on any specific architecture.
3. Low computational overhead
   - I don't want to do additional inference for my visualizations.
   - Nor do I want to calculate gradients.
   - Wholistic picture.

Dynamic Visualization

```python
class Net(nn.Module):
    def __init__(self, activation="relu"):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool1 = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.gap = nn.AdaptiveAvgPool2d((1,1))
        self.fc = nn.Linear(16, 10)

        # store activation function
        activations = {
            "relu": F.relu,
            "sigmoid": torch.sigmoid,
            "tanh": torch.tanh,
            "swish": lambda x: x * torch.sigmoid(x),
            "gelu": F.gelu,
            "softplus": F.softplus
        }
        self.act = activations[activation]

    def forward(self, x):
        x = self.act(self.conv1(x))
        x = self.pool1(x)
        x = self.act(self.conv2(x))
        x = self.gap(x)
        x = torch.flatten(x, 1)
        x = self.fc(x)
        return x

net = Net(activation="relu")
```

- Training on CIFAR-10.
- We will consider some different activations in some experiments.
- We focus on a very small network for today to demonstrate the types of visualizations.

Training Loop

```python
params = list(net.parameters())
epochs = 10
lr = .01
training_log = []
trained_images = 0   # acts as timestamp

for epoch in range(epochs):
    for batch_idx, (inputs, labels, ids) in enumerate(trainloader):
        outputs = net(inputs)
        loss = F.cross_entropy(outputs, labels)

        net.zero_grad()
        loss.backward()

        with torch.no_grad():
            for p in params:
                p -= lr * p.grad

        # --- record info ---
        batch_size = inputs.size(0)
        preds = outputs.detach().cpu()

        for idx, label, out in zip(ids, labels, preds):
            training_log.append({
                "id": int(idx.item()),                # true dataset ID
                "timestamp": trained_images,          # number of samples seen so far
                "label": int(label.item()),
                "output": out.tolist()
            })

        trained_images += batch_size
```

Barycenter Tracking Pt 2

```
groups = [ [] for _ in range(10)]
for e in training_log:
    groups[e["label"]].append(e)


window = 2000  # adjust as you like
smoothed = {}

for lbl, entries in enumerate(groups):
    # sort by timestamp (just in case)
    entries = sorted(entries, key=lambda x: x["timestamp"])

    timestamps = np.array([e["timestamp"] for e in entries])
    outputs = np.array([e["output"] for e in entries])  # shape: (N, num_classes)

    # --- moving average over time ---
    if len(outputs) >= window:
        kernel = np.ones(window) / window
        smooth_outputs = np.apply_along_axis(
            lambda m: np.convolve(m, kernel, mode='valid'),
            axis=0, arr=outputs
        )
    else:
        smooth_outputs = outputs  # too short, skip smoothing

    smoothed[lbl] = {
        "timestamps": timestamps[: len(smooth_outputs)],
        "avg_outputs": smooth_outputs
    }
```

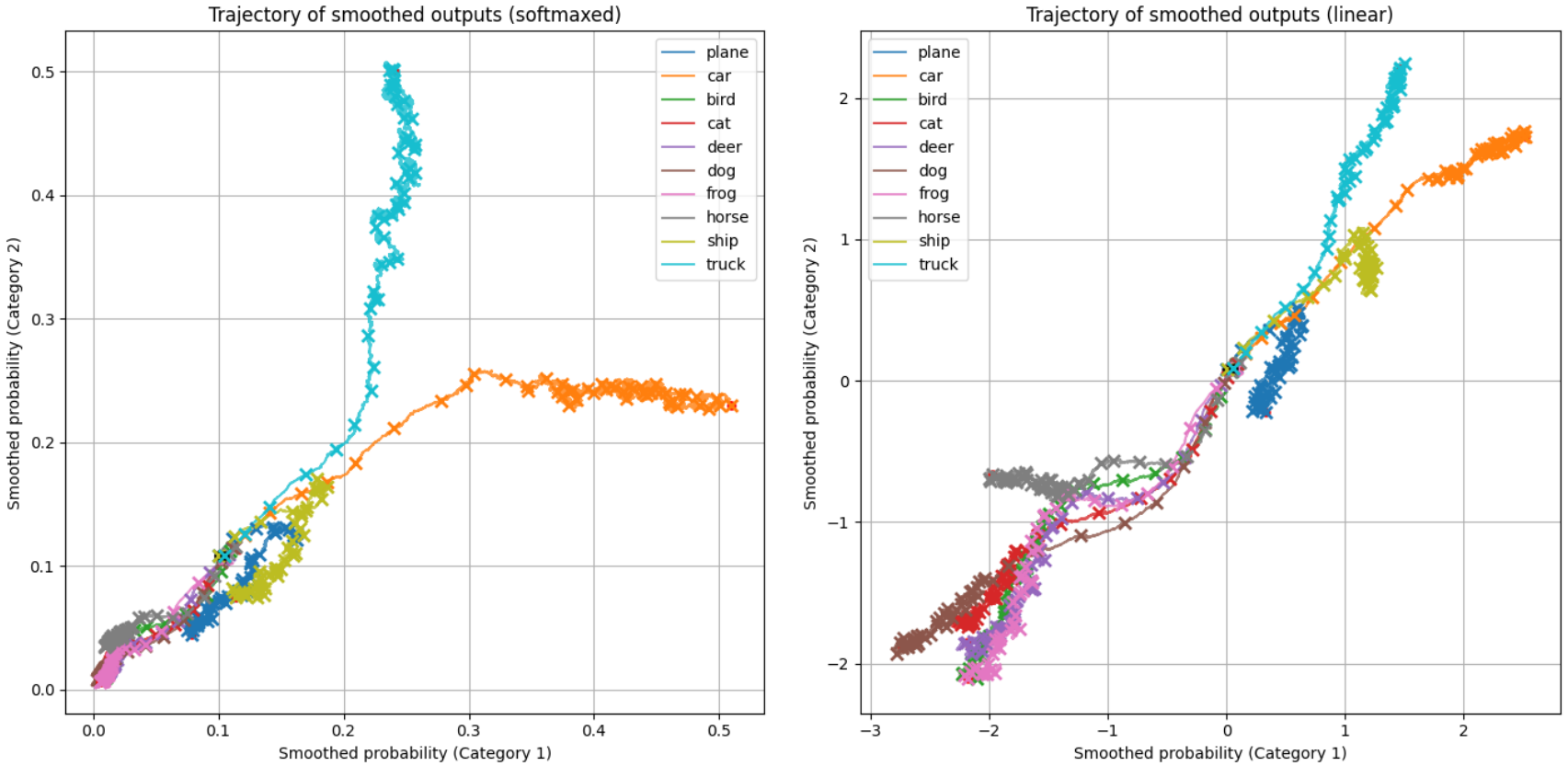


Figure: Left is the soft-maxed outputs while right is the read of the final linear head. Crosses represent each use of 1000 training points.

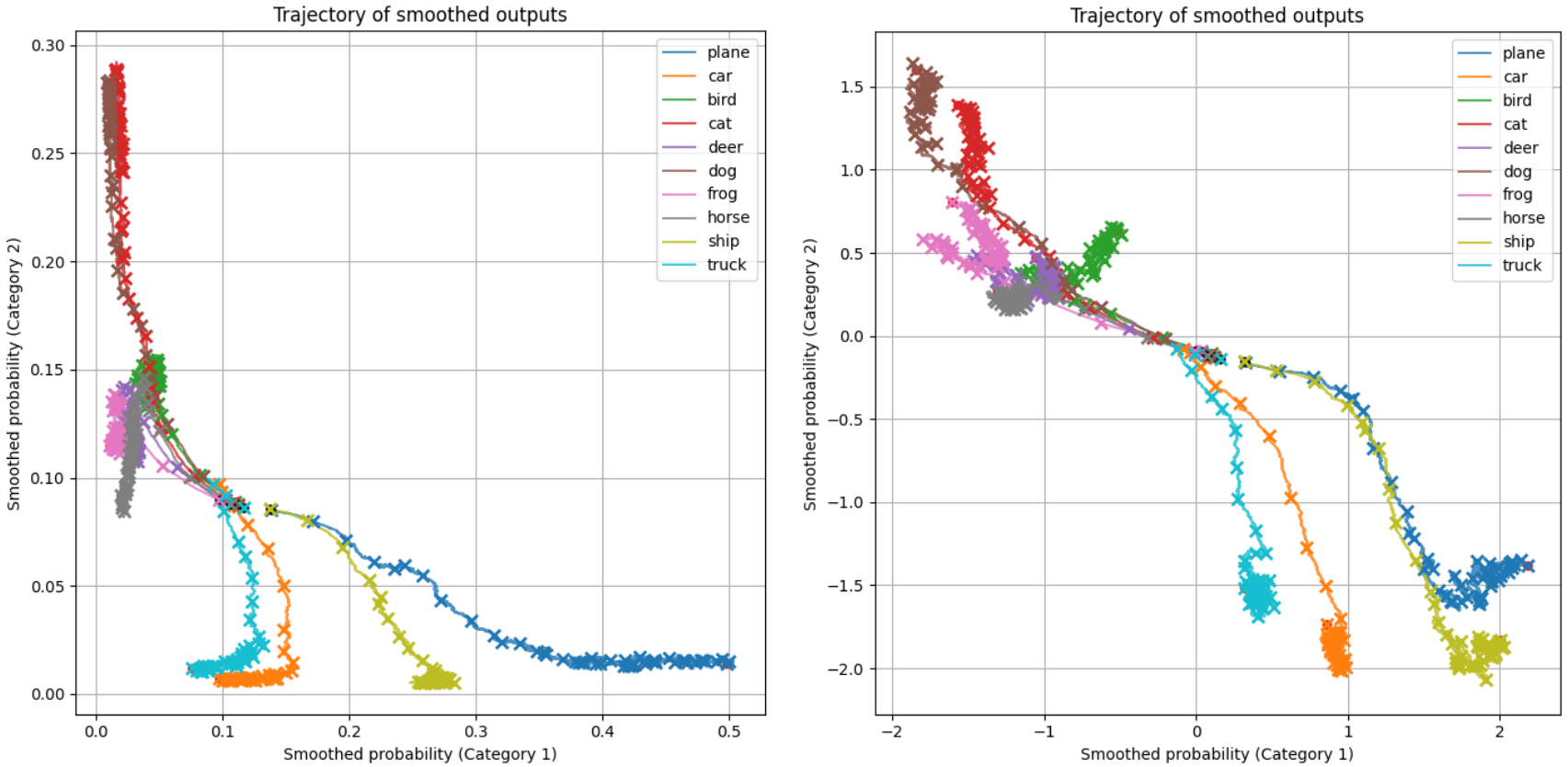


Figure: Estimated probability of planes versus cats, left is softmax and right is linear.

The Role Of Activations

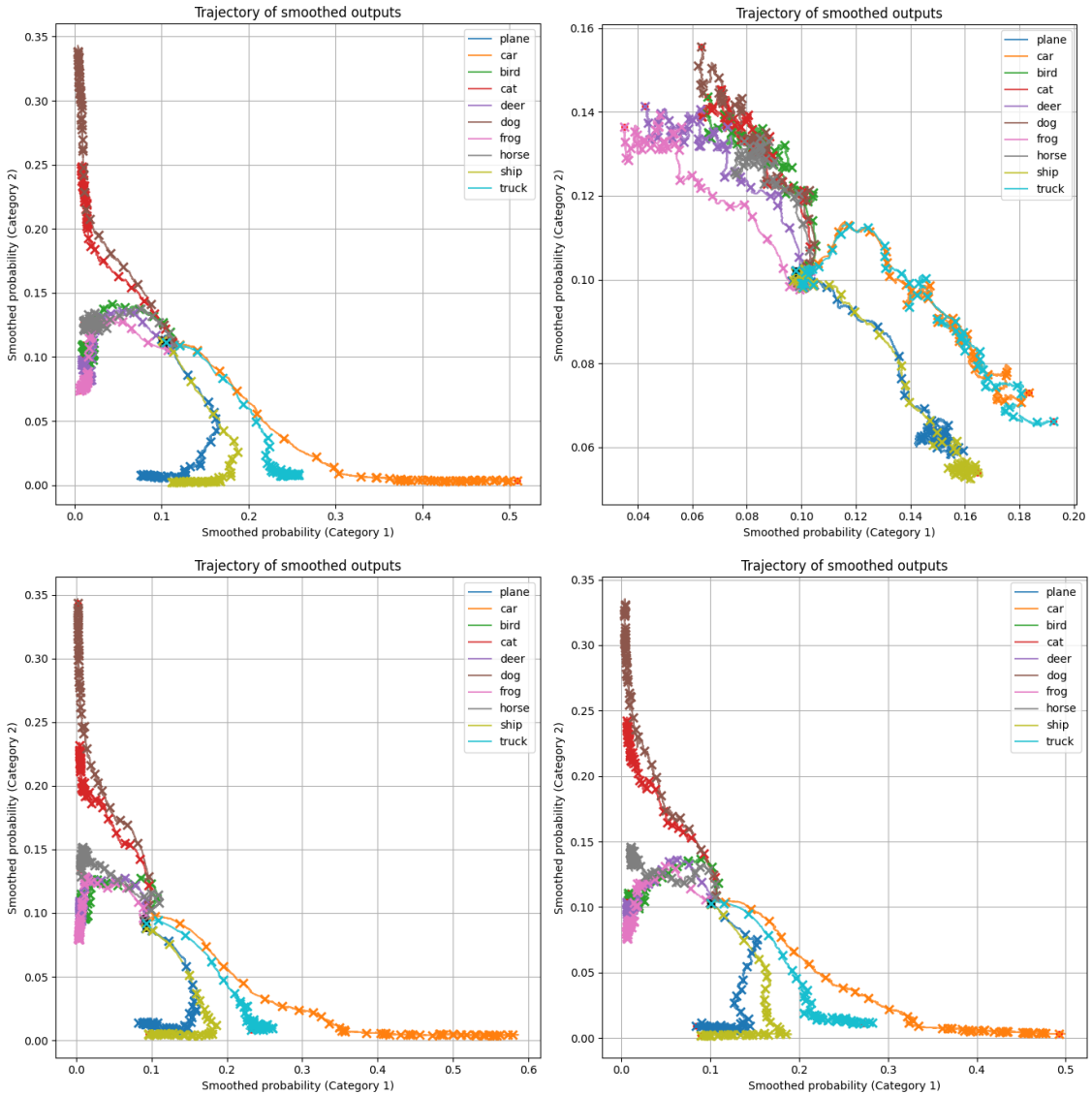


Figure: From upper left going clockwise: Relu, Sigmoid, Softplus, Swish.

PCA Embeddings

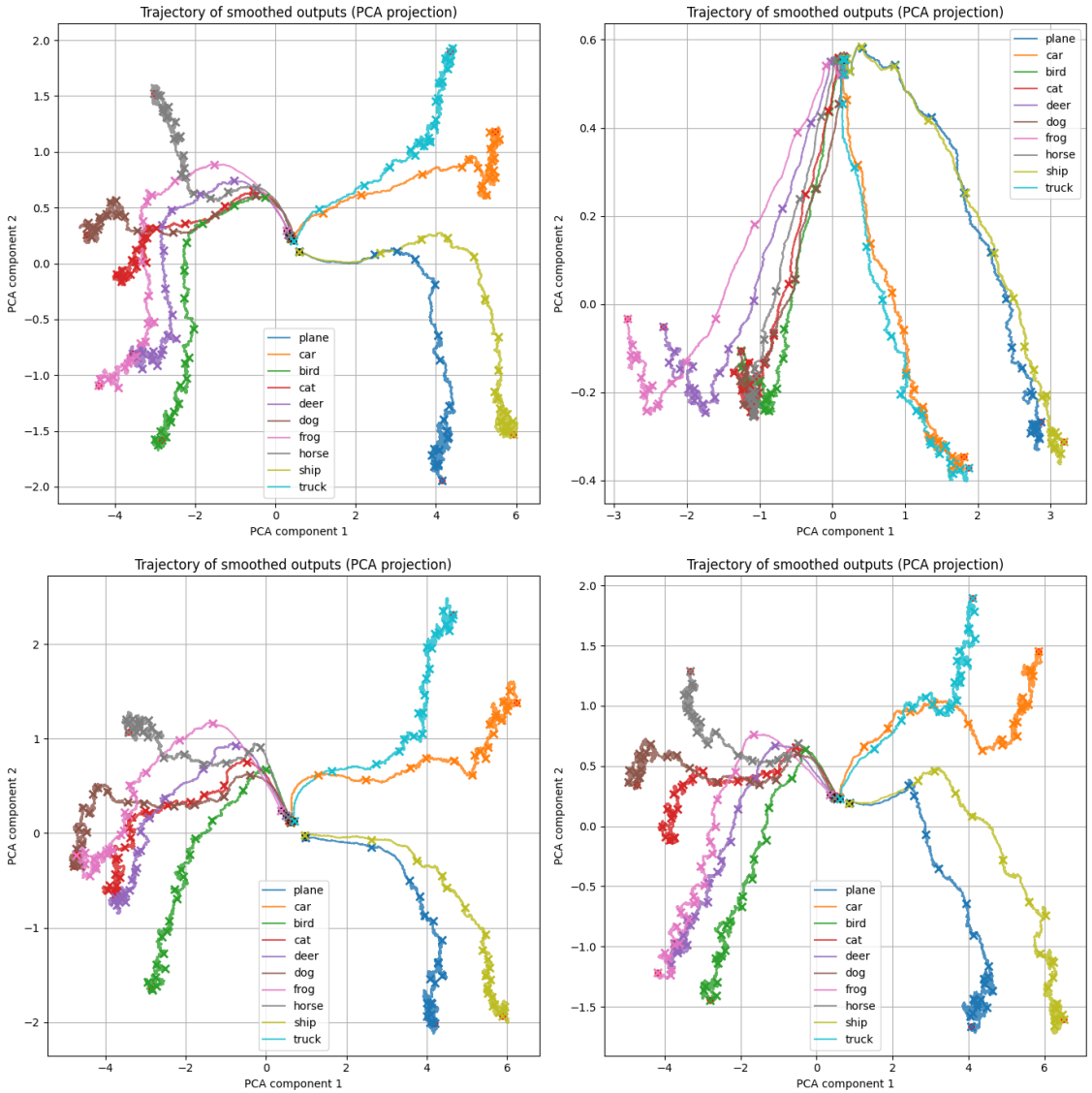


Figure: From upper left going clockwise: Relu, Sigmoid, Softplus, Swish.

Not Just Line Plots

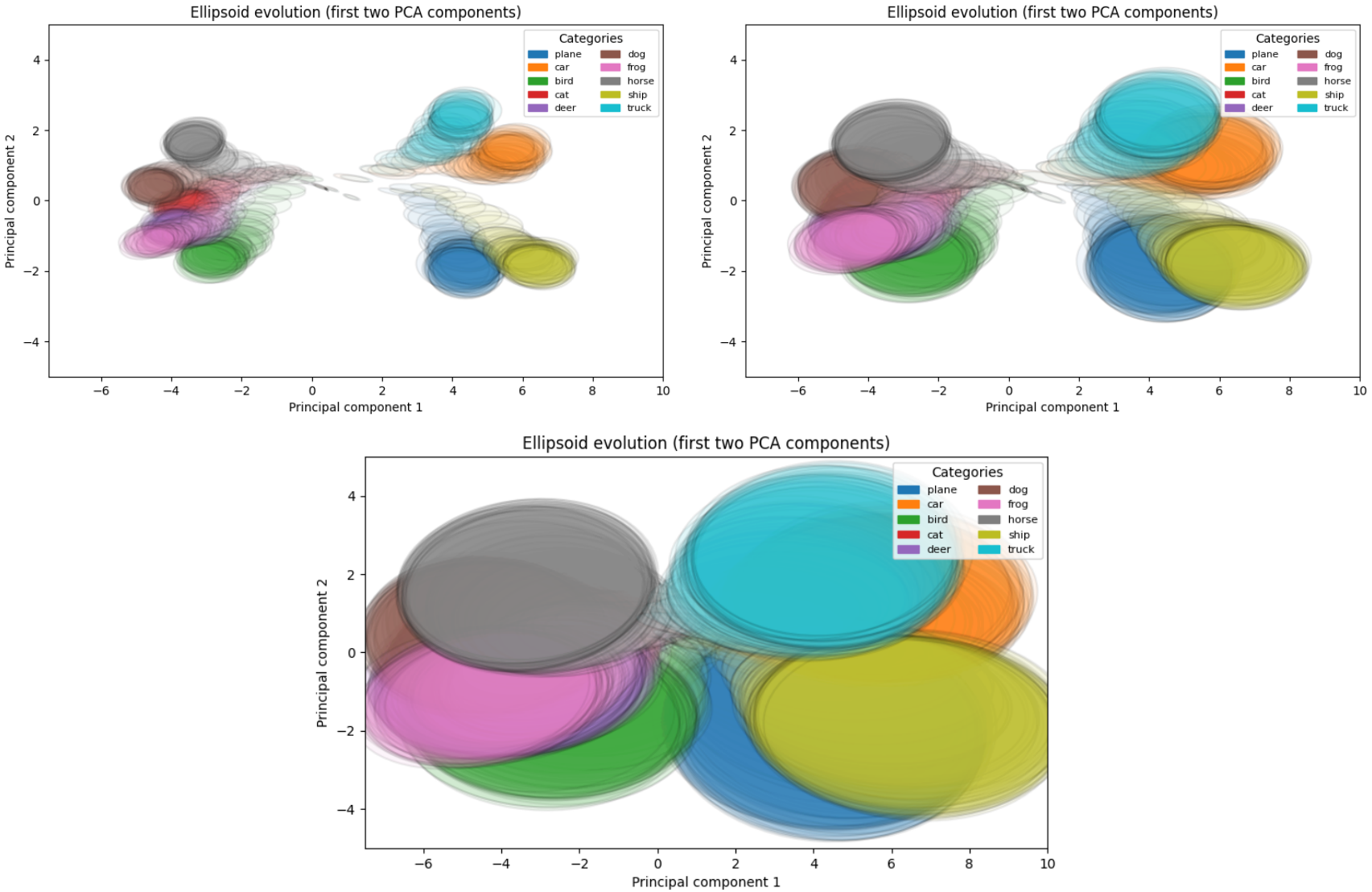


Figure: Relu network: Top row, $\sigma = 0.25, 0.5$; Bottom row, $\sigma = 1$.

Not Just Line Plots

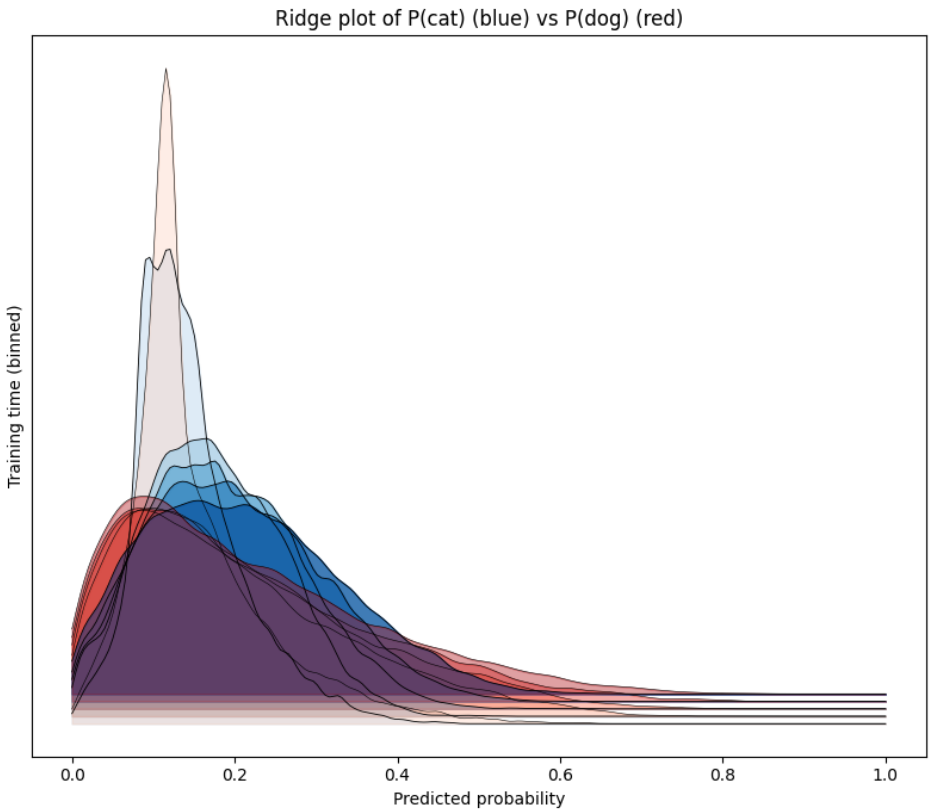


Figure: Relu Network: Cat In Blue Dog in Red; Histogram of cat category probabilities of cat (blue) versus dog (red) prediction.

Live Ablation

```
transform_green     = transforms.Compose([to_tensor, GreenOnly(), normalize])
transform_greenred  = transforms.Compose([to_tensor, GreenRed(), normalize])
transform_full      = transforms.Compose([to_tensor, normalize])

phases = [
    ("Green-only", transform_green),
    ("Green+Red", transform_greenred),
    ("Full RGB", transform_full)
]
```

Live Ablation

```
phase_epochs = epochs // len(phases)

for phase_idx, (phase_name, transform) in enumerate(phases):
    print(f"\n=== Starting phase {phase_idx+1}: {phase_name} ===")

    for epoch in range(phase_epochs):
        trainset = IndexedDataset(torchvision.datasets.CIFAR10(root='./data', train=True, transform=transform))
        trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size, shuffle=True, num_workers=0)

        for batch_idx, (inputs, labels, ids) in enumerate(trainloader):
            outputs = net(inputs)
            loss = F.cross_entropy(outputs, labels)

            net.zero_grad()
            loss.backward()
            with torch.no_grad():
                for p in params:
                    p -= lr * p.grad

            # track loss
            loss_history.append(loss.item())
            steps.append(step)
            step += 1

            # track outputs for later analysis
            preds = outputs.detach().cpu()
            for idx, label, out in zip(ids, labels, preds):
                training_log.append({
                    "id": int(idx.item()),
                    "timestamp": trained_images,
                    "label": int(label.item()),
                    "output": out.tolist()
                })
```

Live Ablation

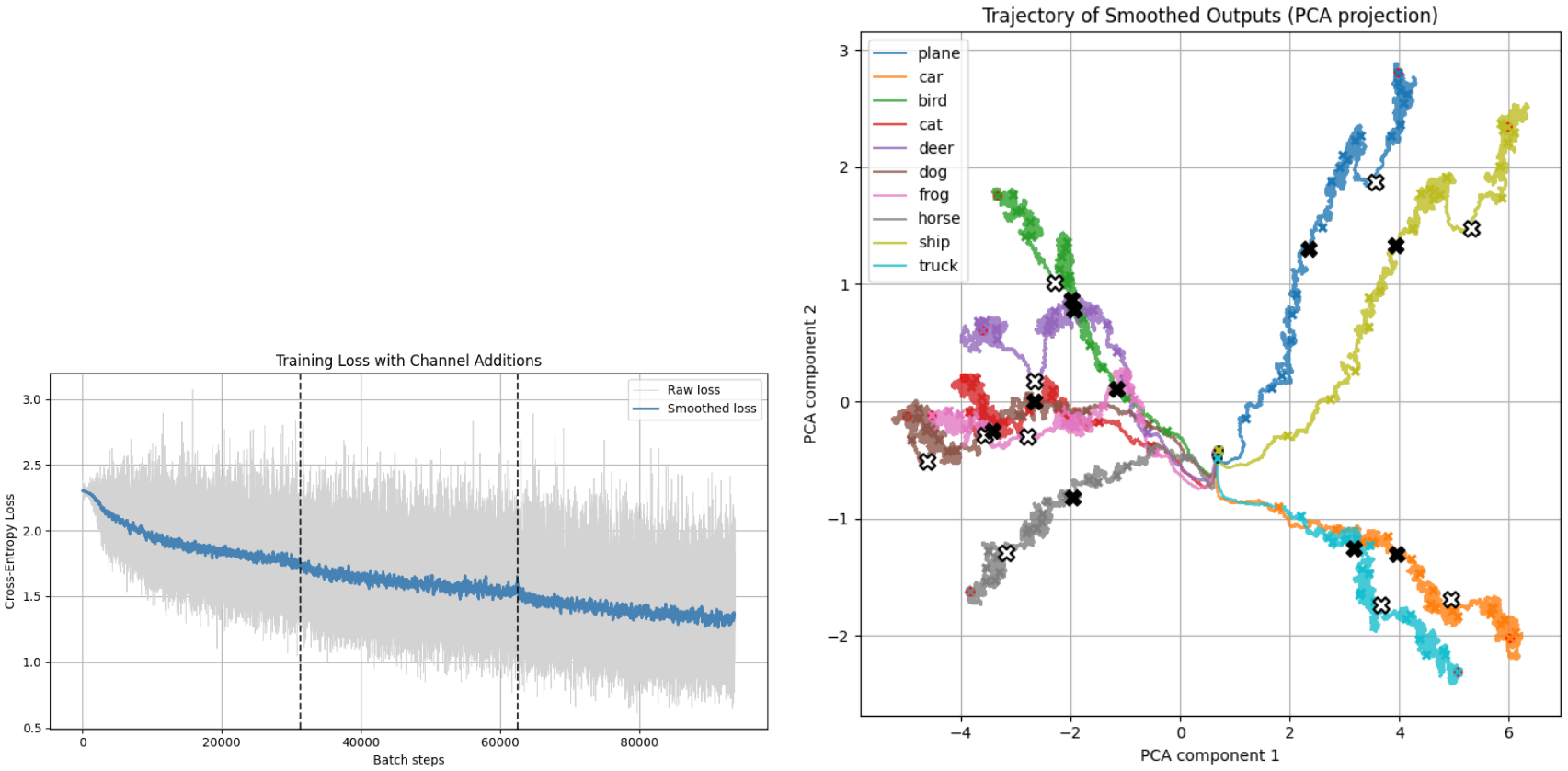


Figure: Left, live ablation loss curve when adding more color channels. Right, Dynamic barycenter of class prediction in a two dimensional PCA embedding.

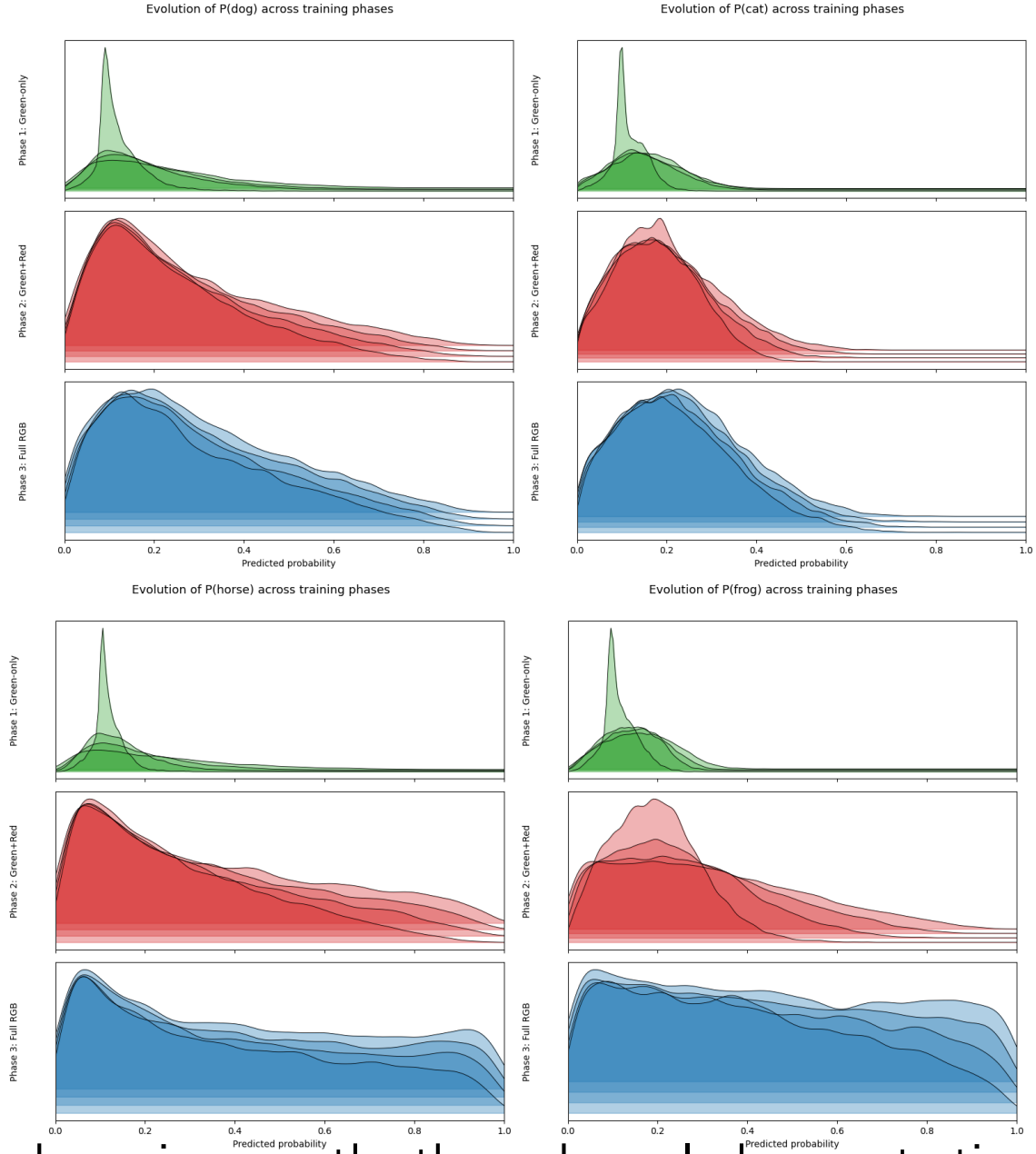


Figure: Training dynamics over the three channel phases, starting from upper left then going clockwise: Dog, Cat, Horse, Frog

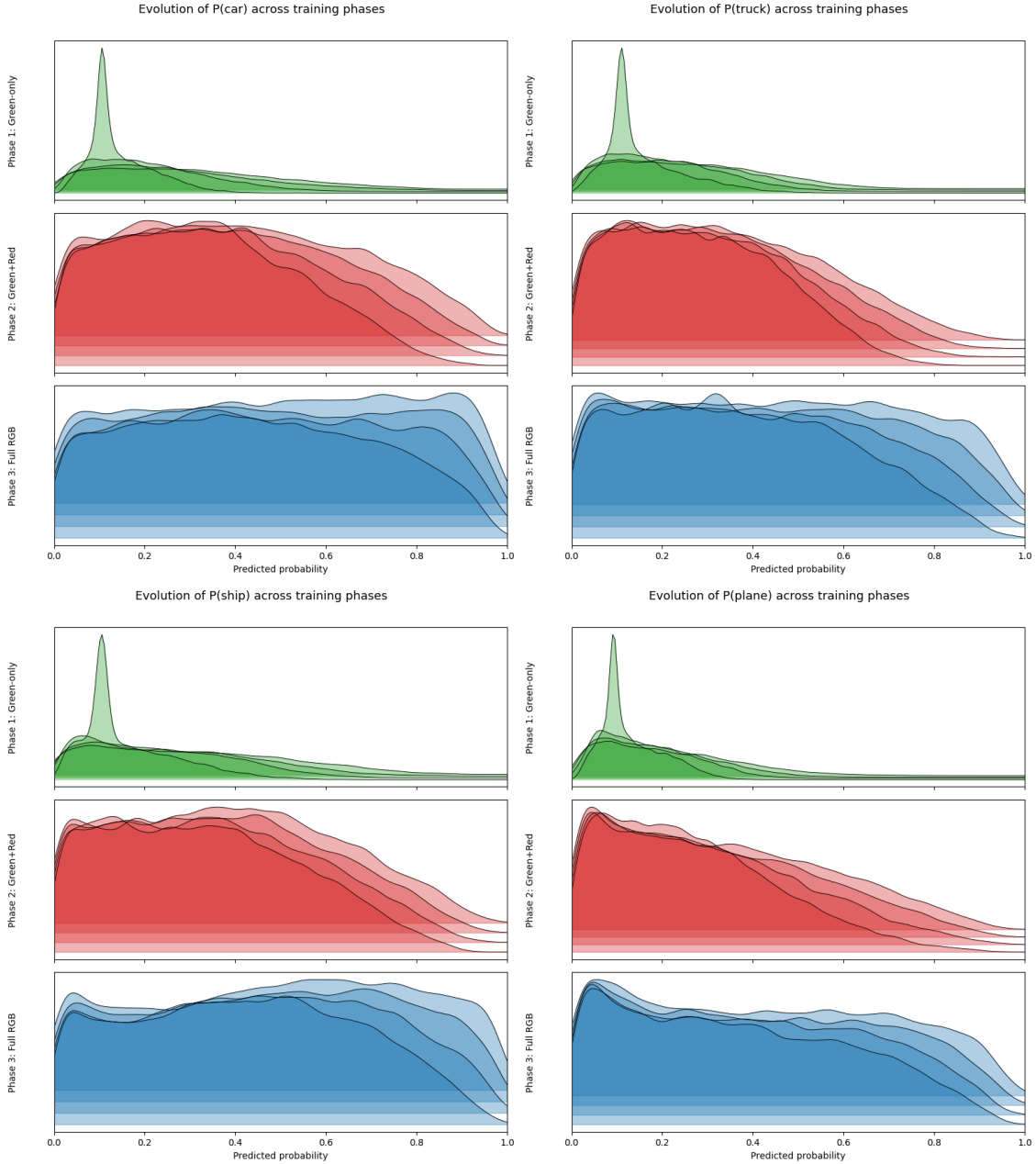


Figure: From upper left then going clock-wise: Car, Truck, Ship, Planes

Not Just Classification

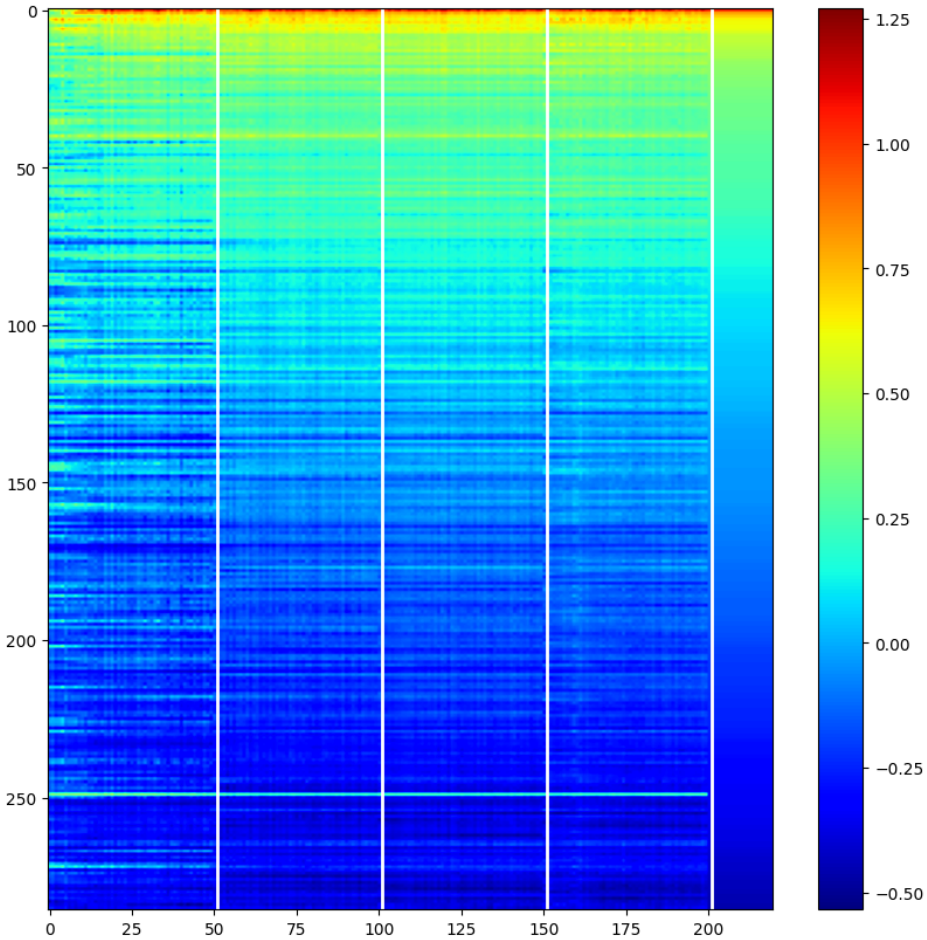


Figure: A dynamic visualization of the network predictions for a regression task. The ordering on the right-hand size (the continuous color band) is the true solutions. Going from left to right is adding more channels to the underlying image. As we go from left to right we we get "finer" corrections to model fit.

Challenges / Future Roadblocks

- Required book keeping when training, can't be done at time of inference.
- Even simple models live in > 3 dimensions, so what embedding reveals information and reduces noise.
- Applying corrections when feature start emerging, how to intervene for causal effect.
- To what extent can this be applied while training instead of post-training visualizations.

How Can We Try To Analyze This Theoretically?

Data Distribution

Data S is drawn from a mixture of Diracs over P atoms:

$$S \overset{\mathcal{L}}{=} \sum_{p \in [P]} \pi_p \delta_{\mu_p}, \quad \mu_p \in \mathbb{R}^{d^2}.$$

Define a local *patch*:

$$\wp_{i,j} = \left(S_{(i-1)\frac{d}{M}+a,(j-1)\frac{d}{M}+b} \right)_{a,b \in [d/M]}.$$

Two-Layer CNN Population Gradient Descent

Consider the very simple CNN student model,

$$\hat{f}(S) = \sum_{l \in [L]} v_l \sum_{i,j \in [M]} \sigma\left(\frac{1}{(d/M)^2} K_l^\top \wp_{ij}(S) \right)$$

and its associated teacher,

$$y(S) = \sum_{l \in [L^*]} v_l^* \sum_{i,j \in [M]} \phi\left(\frac{1}{(d/M)^2} (K_l^*)^\top \wp_{ij}(S) \right).$$

We consider population gradient descent limits on the loss:

$$\mathcal{L} = \mathbb{E}\left[\frac{1}{2} (\hat{f}(S) - y(S))^2 \right].$$

Population Gradient Flow

Gradient flow on the population loss gives:

$$\dot{K}_l = -\mathbb{E}\left[(\hat{f}(S) - y(S))v_l \sum_{i,j} \sigma'\left(\frac{\langle K_l, \wp_{ij}(S)\rangle}{(d/M)^2}\right) \frac{\wp_{ij}(S)}{(d/M)^2}\right],$$

$$\dot{v}_l = -\mathbb{E}\left[(\hat{f}(S) - y(S)) \sum_{i,j} \sigma\left(\frac{\langle K_l, \wp_{ij}(S)\rangle}{(d/M)^2}\right)\right].$$

Substituting $y(S)$ and simplifying:

$$\dot{K}_l = \sum_{l^*} v_l v_{l^*}^* \sum_{i,j,i^*,j^*} \mathbb{E}\left[\phi\left(\frac{\langle K_{l^*}^*, \wp_{i^*j^*}\rangle}{(d/M)^2}\right)\sigma'\left(\frac{\langle K_l, \wp_{ij}\rangle}{(d/M)^2}\right)\frac{\wp_{ij}(S)}{(d/M)^2}\right]$$
$$- \sum_{l'} v_l v_{l'} \sum_{i,j,i',j'} \mathbb{E}\left[\sigma\left(\frac{\langle K_{l'}, \wp_{i'j'}\rangle}{(d/M)^2}\right)\sigma'\left(\frac{\langle K_l, \wp_{ij}\rangle}{(d/M)^2}\right)\frac{\wp_{ij}(S)}{(d/M)^2}\right].$$

Similarly:

$$\dot{v}_l = \sum_{l^*}\sum_{i,j,i^*,j^*} \mathbb{E}\left[\phi\left(\frac{\langle K_{l^*}^*, \wp_{i^*j^*}\rangle}{(d/M)^2}\right)\sigma\left(\frac{\langle K_l, \wp_{ij}\rangle}{(d/M)^2}\right)\right]$$
$$- \sum_{l'}\sum_{i,j,i',j'} \mathbb{E}\left[\sigma\left(\frac{\langle K_{l'}, \wp_{i'j'}\rangle}{(d/M)^2}\right)\sigma\left(\frac{\langle K_l, \wp_{ij}\rangle}{(d/M)^2}\right)\right].$$

Order Parameter Reduction

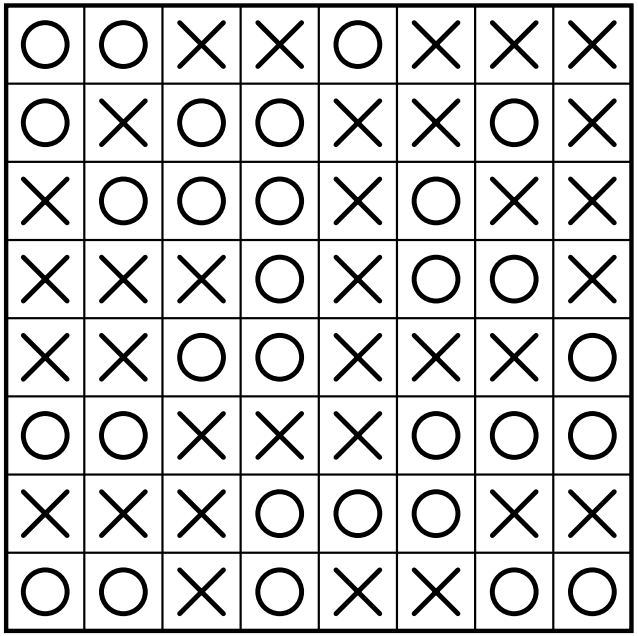
Define the order parameters:

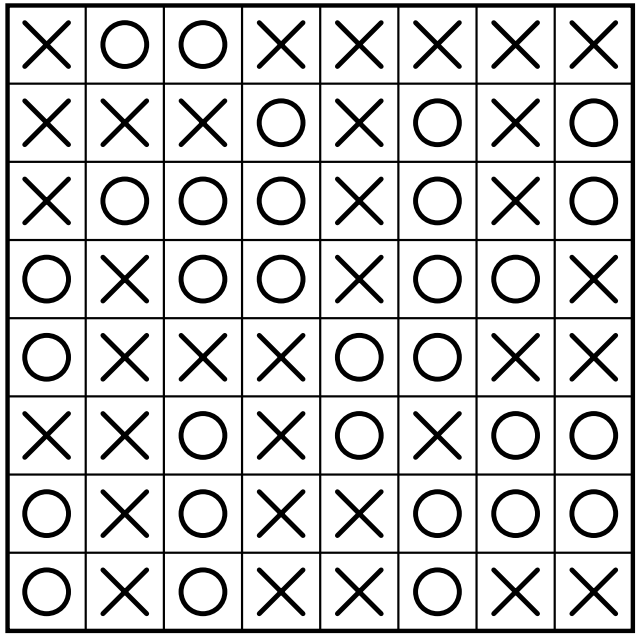
$$\mathbf{m}^*_{lij} = \frac{1}{(d/M)^2}\langle K^*_l, \wp_{ij}(S)\rangle, \quad \mathbf{m}_{lij} = \frac{1}{(d/M)^2}\langle K_l, \wp_{ij}(S)\rangle.$$

Then:

$$\begin{aligned}
\dot{K}_l &= \sum_{l^*} v_l v^*_{l^*} \sum_{i,j,i^*,j^*} \mathbb{E}\left[\phi(\mathbf{m}^*_{l^* i^* j^*})\sigma'(\mathbf{m}_{lij})\frac{\wp_{ij}(S)}{(d/M)^2}\right] \\
&\quad - \sum_{l'} v_l v_{l'} \sum_{i,j,i',j'} \mathbb{E}\left[\sigma(\mathbf{m}_{l' i' j'})\sigma'(\mathbf{m}_{lij})\frac{\wp_{ij}(S)}{(d/M)^2}\right], \\
\dot{v}_l &= \sum_{l^*} \sum_{i,j,i^*,j^*} \mathbb{E}\left[\phi(\mathbf{m}^*_{l^* i^* j^*})\sigma(\mathbf{m}_{lij})\right] - \sum_{l'} \sum_{i,j,i',j'} \mathbb{E}[\sigma(\mathbf{m}_{l' i' j'})\sigma(\mathbf{m}_{lij})]\,.
\end{aligned}$$

Consider the following images as our data points:





$Y(S) = \#\mathsf{X} - \#\mathsf{O} = 4 = 4$ $\quad Y(S) = \#\mathsf{X} - \#\mathsf{O} = 4 = 4$

$$\text{Shape} \sim \begin{cases} \times & \text{with probability } \rho \\ \circ & \text{with probability } 1-\rho \end{cases}, \qquad \rho \sim \mathbb{P}_{[0,1]}$$

Because of the iid form of the above mentioned data, we can easily close the PDE system in terms of order parameters.

$$\dot{\mathbf{m}}_{lr} = \sum_{l^* \in [L^*]} v_l v_{l^*}^* \sum_{p,q \in [P]} \mathbb{E}_\pi [\pi_p \pi_q] \phi(\mathbf{m}_{l^* p}^*) \sigma'(\mathbf{m}_{lq}) \mathbf{m}_{rq}^* \tag{1}$$

$$- \sum_{l' \in [L]} v_l v_{l'} \sum_{p,q \in [P]} \mathbb{E}_\pi [\pi_p \pi_q] \sigma(\mathbf{m}_{l' p}) \sigma'(\mathbf{m}_{lq}) \mathbf{m}_{rq}^* \tag{2}$$

$$\dot{v}_l = \sum_{l^* \in [L^*]} v_{l^*}^* \sum_{p,q \in [P]} \mathbb{E}_\pi [\pi_p \pi_q] \phi\left(\mathbf{m}_{l^* p}^*\right) \sigma\left(\mathbf{m}_{lq}\right) \tag{3}$$

$$- \sum_{l' \in [L]} v_{l'} \sum_{p,q \in [P]} \mathbb{E}_\pi [\pi_p \pi_q] \sigma\left(\mathbf{m}_{l' p}\right) \sigma\left(\mathbf{m}_{lq}\right). \tag{4}$$

For the shape counting problem we assume that $\phi(\mathbf{m}) = \mathbb{1}\{\mathbf{m} = 1\}$, $L^* = L = 2 = P$ and $v_* = [1, -1]$.

Simulations

Depending on the data composition of ρ and overlap between patches (if they are not strictly $\times$ or $\circ$) we can have interesting solution dynamics.

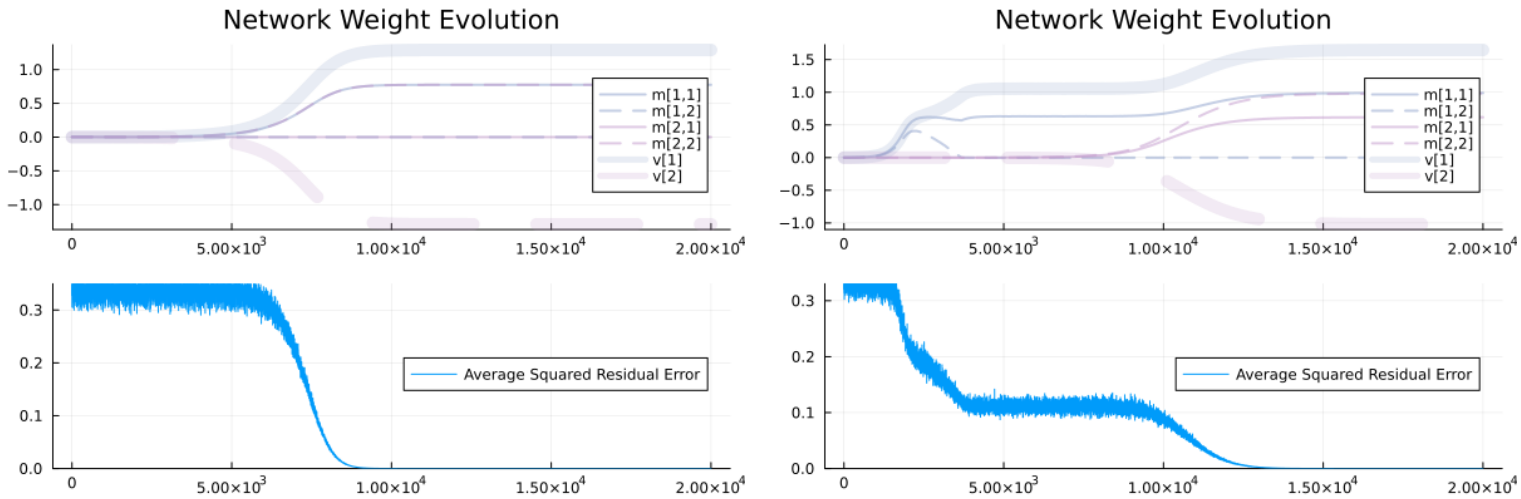


Figure: Simulation of order parameters for kernel-to-signal correlations with a evolving linear head initialized at $v = [\varepsilon, -\varepsilon]$ with $[1, -1] = v^*$ for the shape counting problem assuming 0.8 overlap with Relu activation. Left, balanced data set where $\rho \sim \mathrm{Unif}([0, 1])$; Right, unbalanced data set where $\rho \sim \mathrm{Direclet}([2, 1])$ has right skew.

Discussion

Questions & Discussion

References I

- Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, *Imagenet classification with deep convolutional neural networks*, Advances in Neural Information Processing Systems 25 (NIPS 2012), 2012, pp. 1097–1105.
- Daniel Omeiza, Skyler Speakman, Celia Cintas, and Komminist Weldermariam, *Smooth grad-cam++: An enhanced inference level visualization technique for deep convolutional neural network models*, 2019.
- Vitali Petsiuk, Abir Das, and Kate Saenko, *Rise: Randomized input sampling for explanation of black-box models*, 2018.
- Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin, *"why should i trust you?": Explaining the predictions of any classifier*, 2016.