Heart of Hearthstone

A Machine Learning Trilogy

Emil Møller Olsen, Filip Keinicke, Karl Johannes Hansen

All contributed equally

Picture created by Gemini

UNIVERSITY OF COPENHAGEN

Why Hearthstone?

INSPIRATION: magic the gathering.

ISSUES: Less cards,

PROBLEM: classifying cards? Making new cards? Clustering cards? Making new decks? Predicting if decks are good? Battle simulation with our decks? Using Win Rates to determine if decks and strategies are good.

There were a lot of Problems and we didn't really have any sense what we should focus on… So we tried almost all of them. We split up and each took on a different aspect of the game!

Data

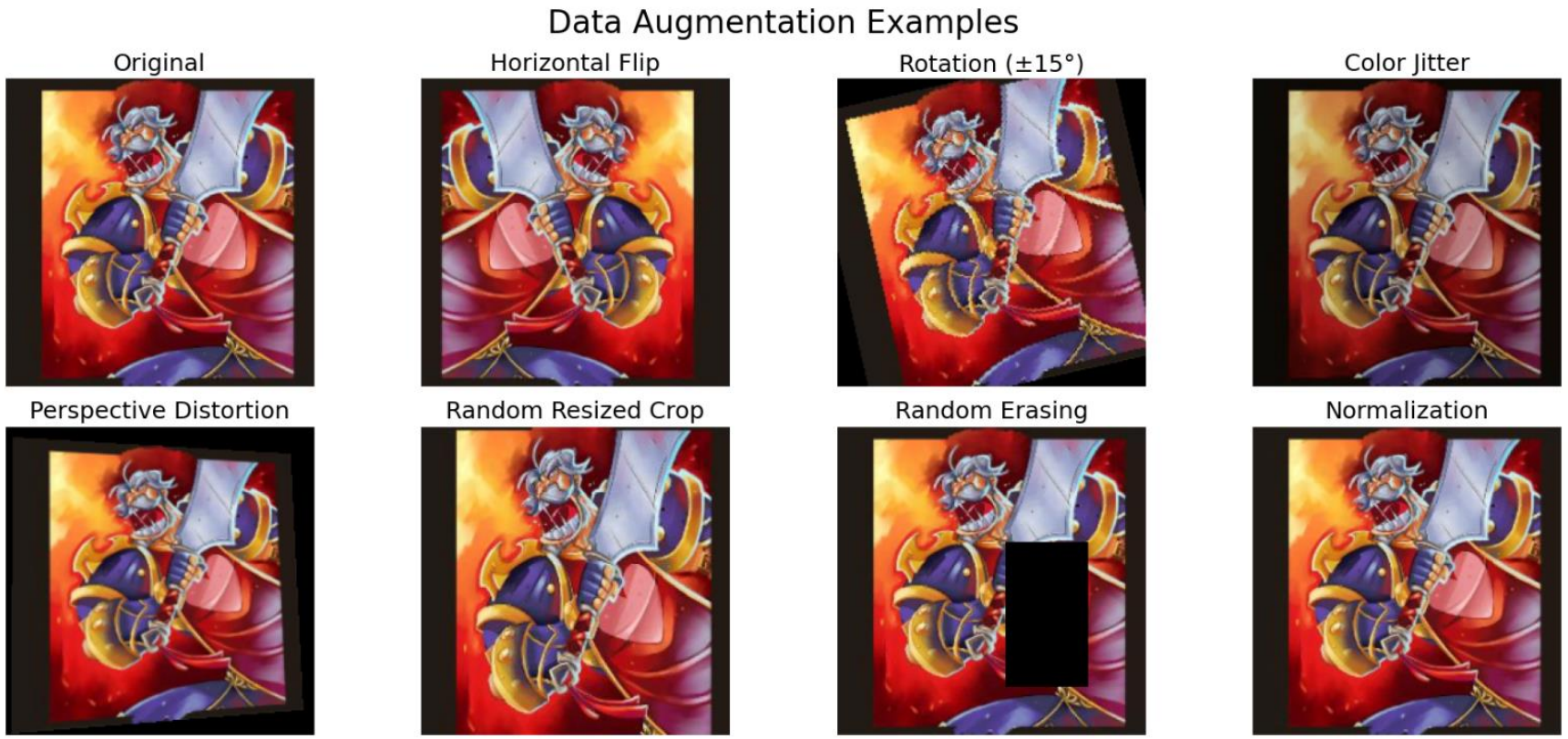
Pictures taken from https://hearthstone.blizzard.com/en-us/cards

Cards from XML file that contain information such as:

- Card ID
- Card name
- Mana cost
- Attack and Health (if applicable)
- Card type (Minion, Spell, Weapon, etc.)
- Class
- Rarity
- Card text and keywords (e.g., Taunt, Charge, Battlecry)

Cards - Image classification

- Downloaded card art as 256x256 .jpgs using Card ID
- 7926 images in total were downloaded split 80/20
- Augmentation to improve learning and generalization



Picture created by Gemini

ResNet18 - CNN Class Prediction Model

Image input
(256 x 256 x 3)

Image compression
(224 x 224 x 3)

7x7 convolution
(64 features)

Max Pooling

Resblock group 1
(64 filters)

Resblock group 2
(128 filters)

Resblock group 3
(256 filters)

Resblock group 4
(512 filters)

Global Average Pooling

Dropout (0.4)

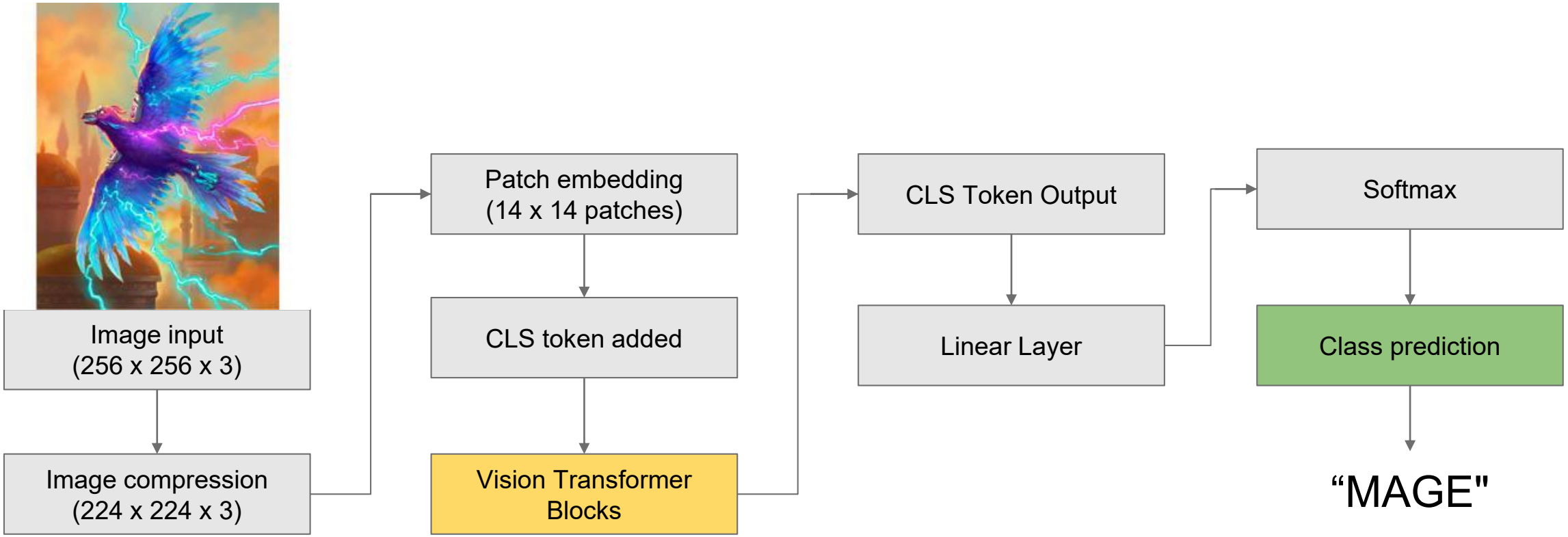
Dense layer (512 -> 12)

Softmax

Class prediction

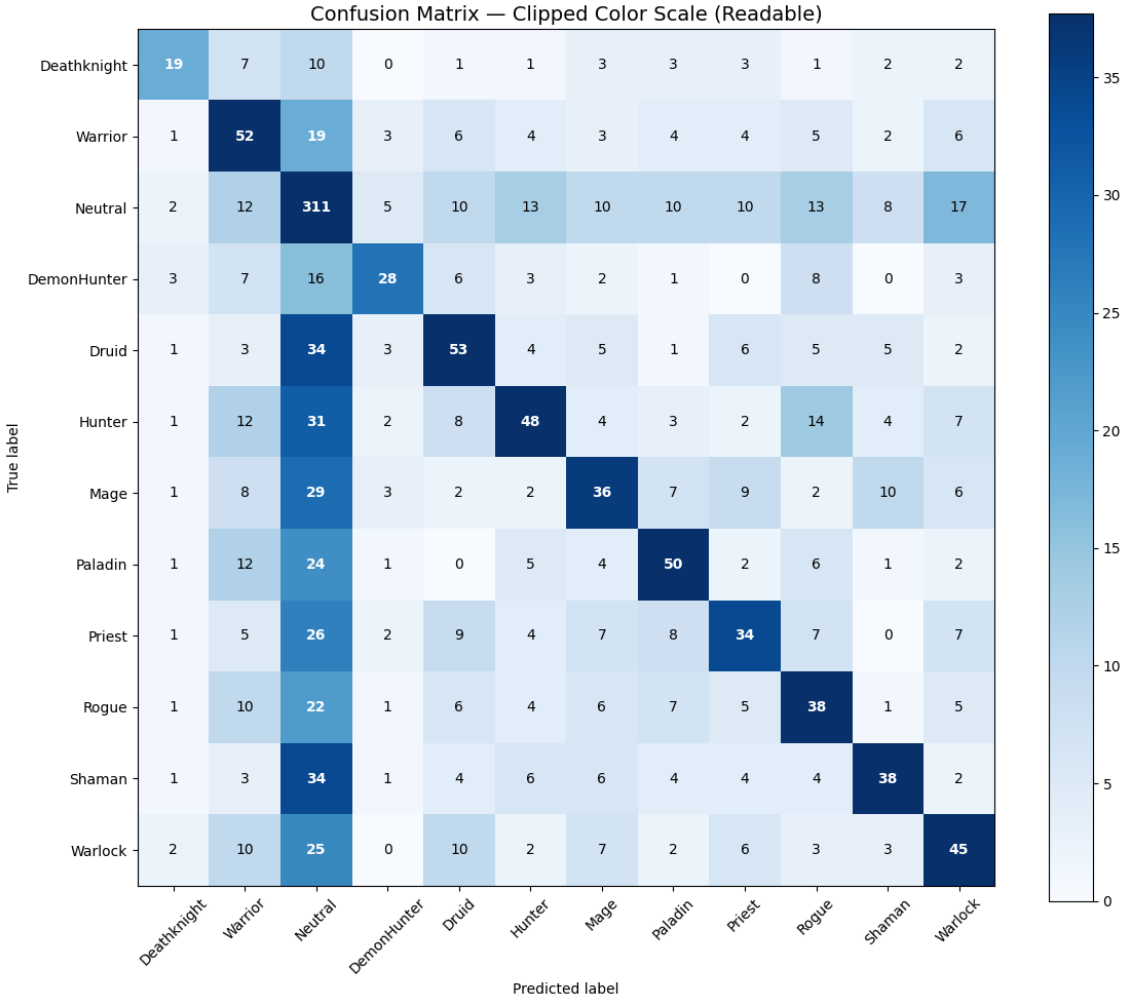
“MAGE"

DINOv2 - Vision Transformer Class Prediction Model

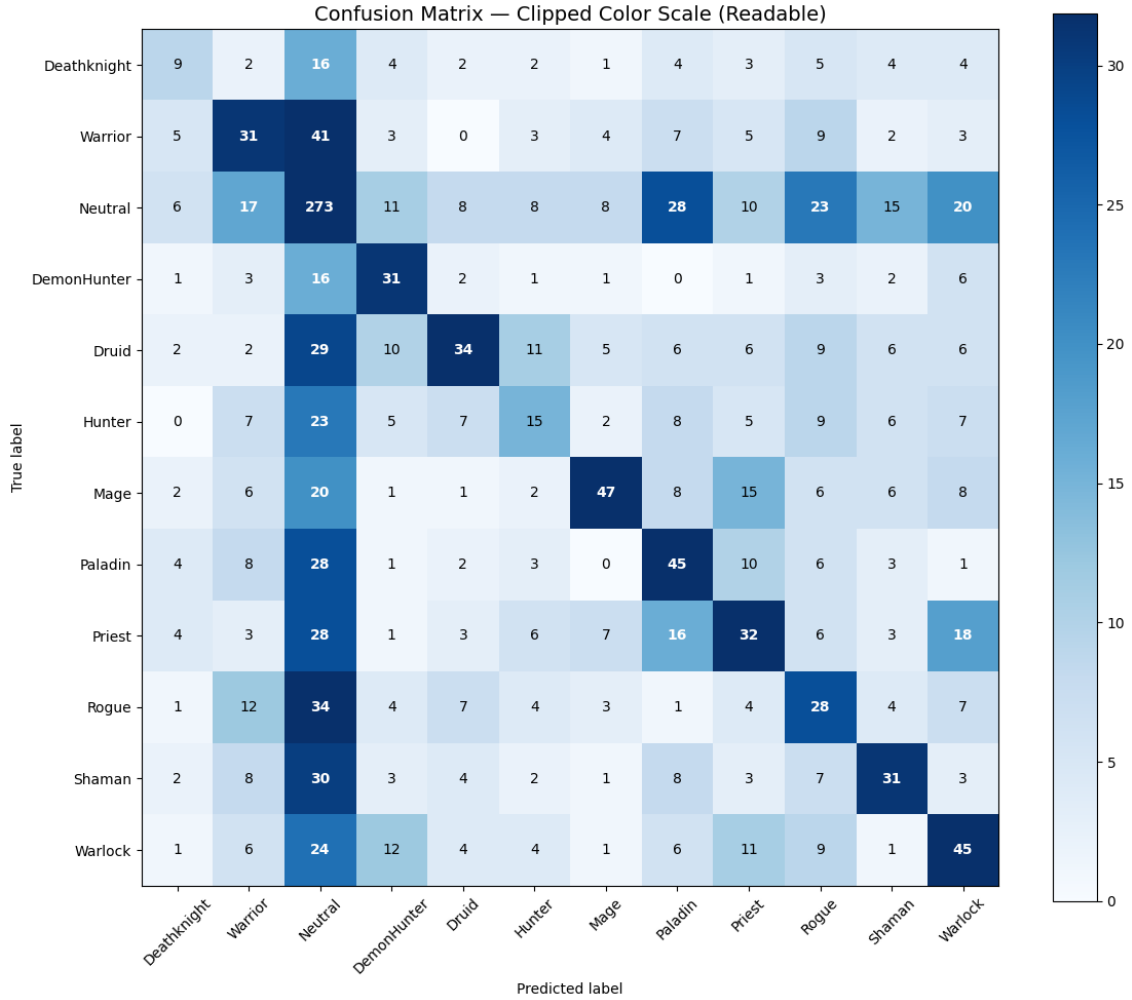


ResNet18 and DINOv2 comparison class accuracy

ResNet18

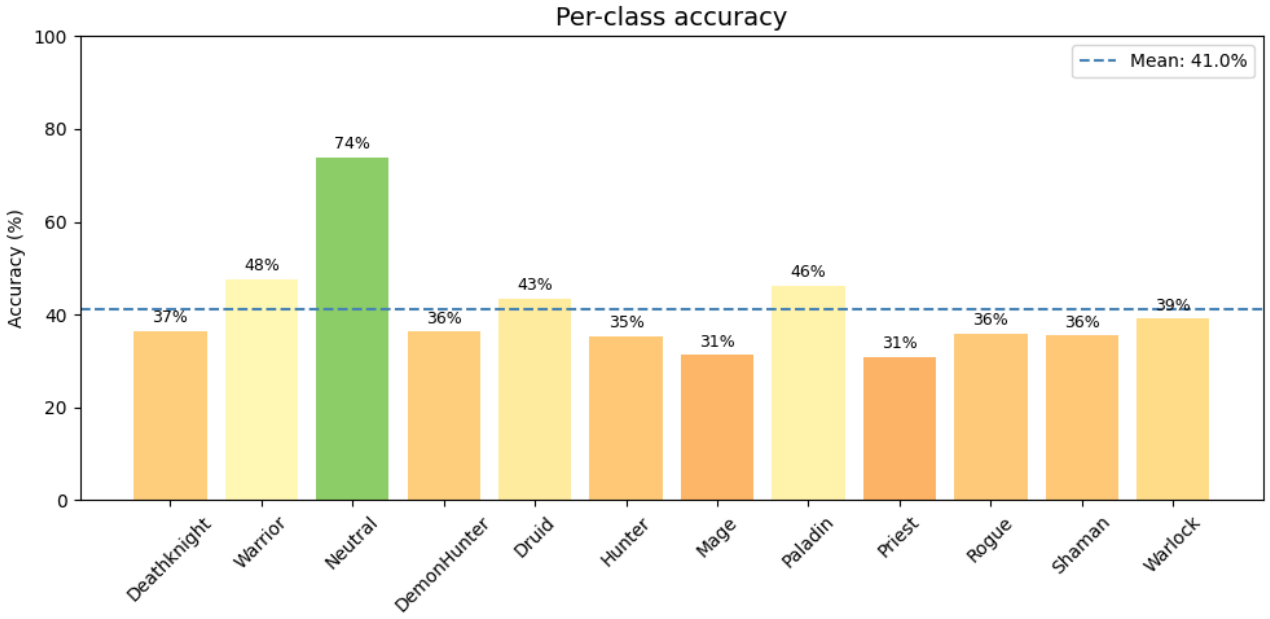


DINOv2



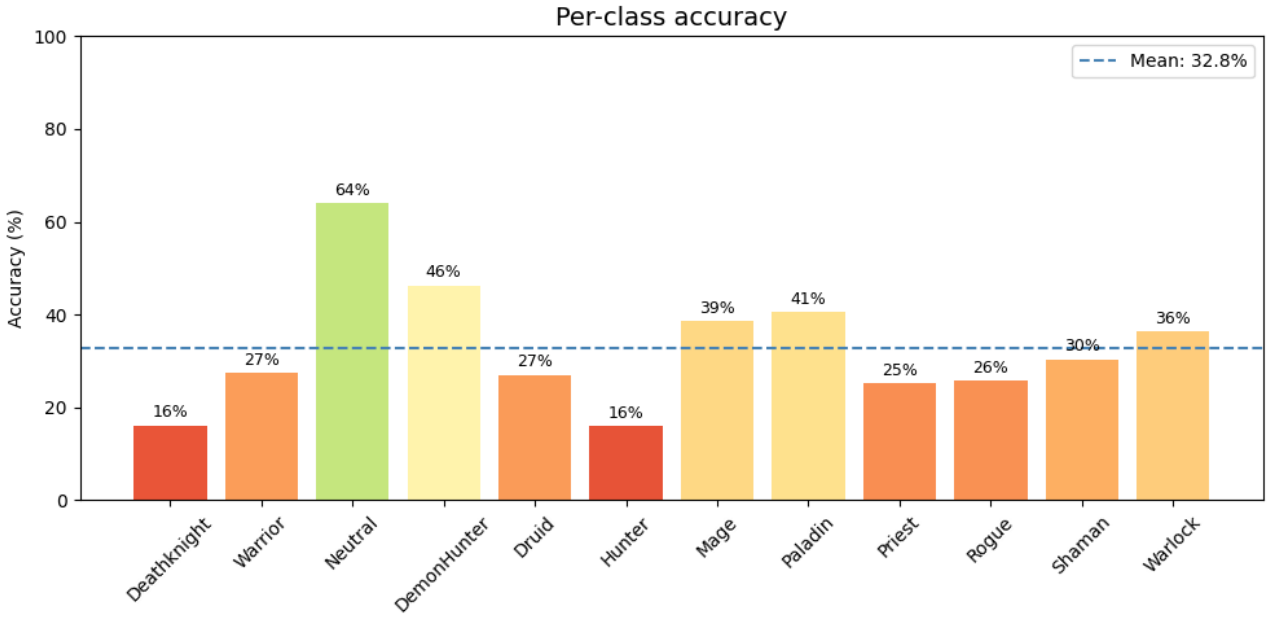
ResNet18 and DINOv2 comparison class accuracy

ResNet18



Strong neutral performance as expected

DINOv2



Worse than ResNet18

Decks!

Picture created by Gemini

- ChatGPT: “A GPT-style Transformer for Hearthstone deck generation.”
- Transformer Deep learning NN
- Pytorch
- Data - pulled from GITHUB

No data on meta available!
Get manually?
Made Claude do it.

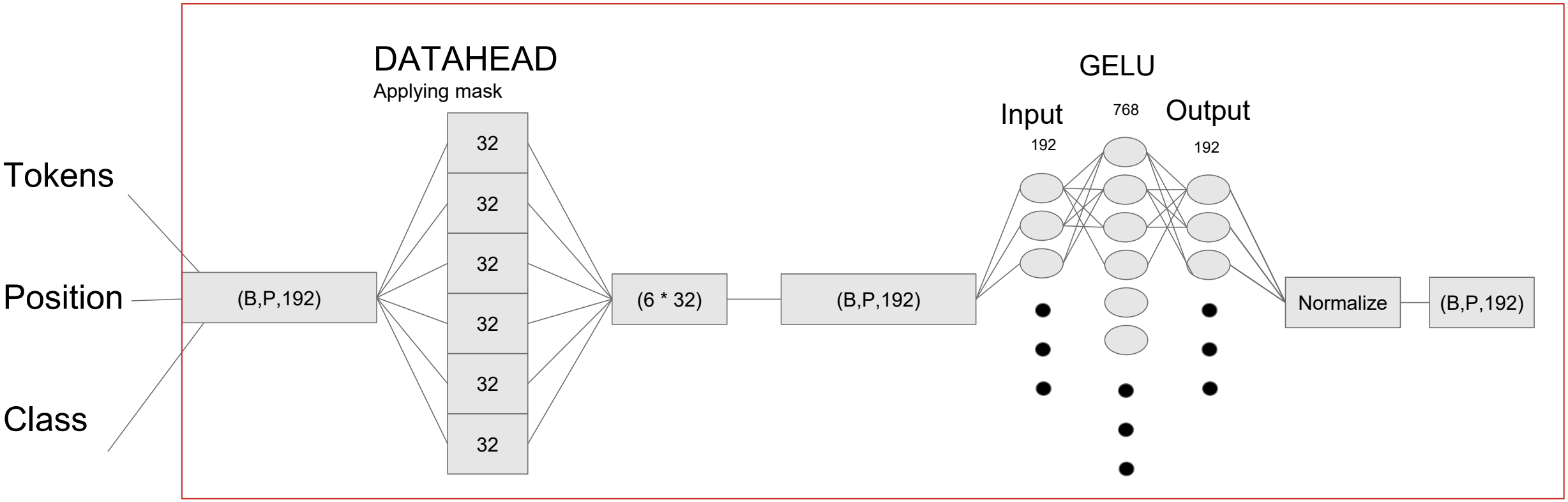
```
567 unique real decks (from 567 total)
  DEMONHUNTER  total= 84  train= 58  val=13  test=13
  PALADIN      total= 47  train= 33  val= 7  test= 7
  DEATHKNIGHT  total= 77  train= 53  val=12  test=12
  WARRIOR      total= 43  train= 31  val= 6  test= 6
  SHAMAN       total= 20  train= 14  val= 3  test= 3
  DRUID        total= 39  train= 27  val= 6  test= 6
  HUNTER       total= 46  train= 32  val= 7  test= 7
  ROGUE        total= 84  train= 58  val=13  test=13
  MAGE         total= 74  train= 52  val=11  test=11
  WARLOCK      total= 13  train=  9  val= 2  test= 2
  PRIEST       total= 40  train= 28  val= 6  test= 6

Split sizes: train=395  val=86  test=86
```

HyperParameters

```
        self,
        vocab_size: int,
        n_classes: int,
        d_model: int = 192,
        nhead: int = 6,
        n_layers: int = 4,
        max_len: int = DECK_SIZE + 4,
        dropout: float = 0.1,
```

```
EPOCHS = 25
LR = 3e-4
```

Architecture - Transformer Deep Learning NN



This corresponds to 1 layer, The model has 4 layers

Evaluation

Held-out TEST perplexity: 20.602

Mean Kullback-Leibler (KL) Curve: 0.226

This means that the Model is choosing between ~20 cards for its next card out of all usable cards.

KL(generated curve || test curve) measures the statistical difference, or "information lost". Closer to zero = better

However, the model just associates cards, it doesn't understand the game

Creating “meta trained” Deck datasets

Use the model to generate decks.

10, 100, 1000, 10000, 100000 decks per class

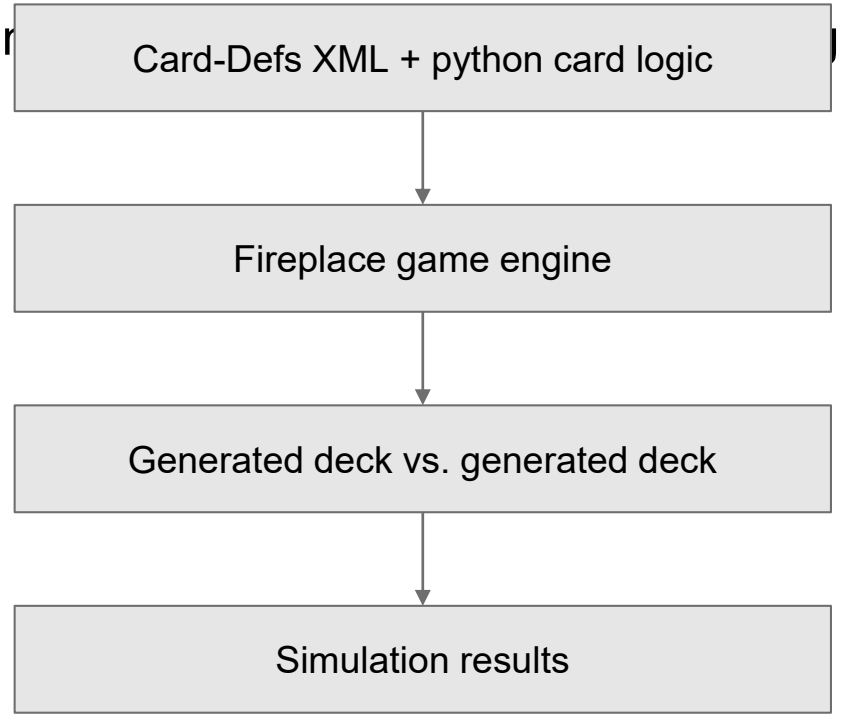
Optimized by parallelizing the code → Large dataset can be created

The deck datasets can then be sent for the next step

Which is…

Fireplace Battle Simulator

- Fireplace is an open-source Hearthstone simulator written in python
- Made by the HearthSim community originally
- Ir[illegible]generated decks automatically



Picture created by Gemini

The Before and After:

Original Fireplace card pool

- Updated to Patch 17.6.053261
- CardDefs.xml: 9,344 cards
- Collectible cards: 2,540

Modified card pool

- Filtered CardDefs_modern_filtered.xml: 12,872 CardIDs
- Deck-eligible cards after filter 7,690
- Added cards: 5,211

Filters

- Only collectable cards
- Minions, weapons, spells, hero cards
- Neutral or match class
- 3 special non-deck cards excluded

Card type	Before	Added	Total in simulation
Hero cards	63	689	752
Minions	1,693	2,916	4,609
Spells	676	1,544	2,220
Weapons	47	62	109
Total	2,479	5,211	7,690


```
class AT_001:
    """Flame Lance\nDeal $25 damage to a minion.\n"""
    requirements = {
            PlayReq.REQ_TARGET_TO_PLAY: 0,
            PlayReq.REQ_MINION_TARGET: 0,
        }
    play = Hit(TARGET, 25)
```

The Main Simulation

Simulation type:	All vs. All tournament
Crash mode:	Not strict
Maximum turns per game:	40 turns
Maximum actions per turn:	20 actions
Generated decks tested:	110 decks
Decks per class:	10 decks
Classes:	11 classes
Parallel workers/ cores:	8 cores
Games per matchup:	20 games
Total matchups:	5,995
Total simulated games:	119,900 games

Simulations Results

Rank	Class	**Win rate**	Wins	Losses	Draws
1	Shaman	**74.7%**	16,274	5,509	17
2	Warlock	**74.0%**	16,122	5,653	25
3	Mage	**72.2%**	15,748	6,034	18
4	Hunter	**66.1%**	14,418	7,359	23
5	Death Knight	**55.6%**	12,129	9,637	34
6	Warrior	**43.0%**	9,382	12,387	31
7	Paladin	**41.8%**	9,119	12,644	37
8	Demon Hunter	**40.9%**	8,920	12,825	55
9	Druid	**39.8%**	8,675	13,093	32
10	Priest	**32.3%**	7,044	14,724	32
11	Rogue	**8.8%**	1,909	19,875	16

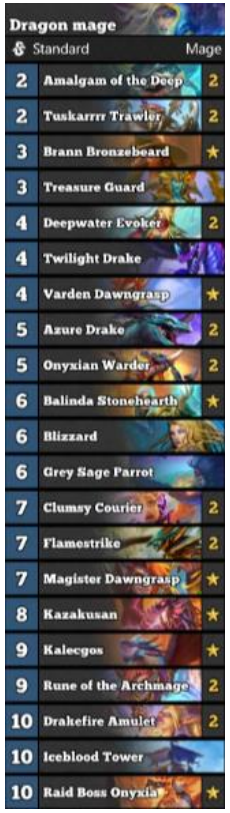
Machine Learning: Predict Simulated Deck Strength

Model	Main parameters	Mean Absolute Error (MAE)	R^2
Baseline	Predicts mean win rate	0.181	-0.045
Neural Network (MLP)	MLP regressor, 64/32 hidden layers	0.523	-9.473
Ridge Regression	Regularized linear model, α=10	0.097	0.517
LightGBM	300 trees, depth 3, lr 0.03	0.091	0.622
Random Forest	500 trees, min leaf 2	0.081	0.673
XGBoost	300 trees, depth 3, lr 0.03	0.065	0.800

MAE = average prediction error in win-rate points.
R^2 = how much variation in simulated win rate the model explains.
110 decks, 311 features. (train/test split = 75/25%)

Conclusion

Class = …



VS.

Win Rate = ...

Future work

- Use more parameters
- Which card has highest win rate
- Deck win rate based on our models
- Strategy and reinforced learning
- Make the simulator even better

Acknowledgements

Thanks to Troels for great teaching throughout the course, and to the TAs for their helpful guidance and support. I would also like to acknowledge the use of generative AI as a sparring partner for debugging, proofreading, and identifying possible mistakes.

Questions & comments?

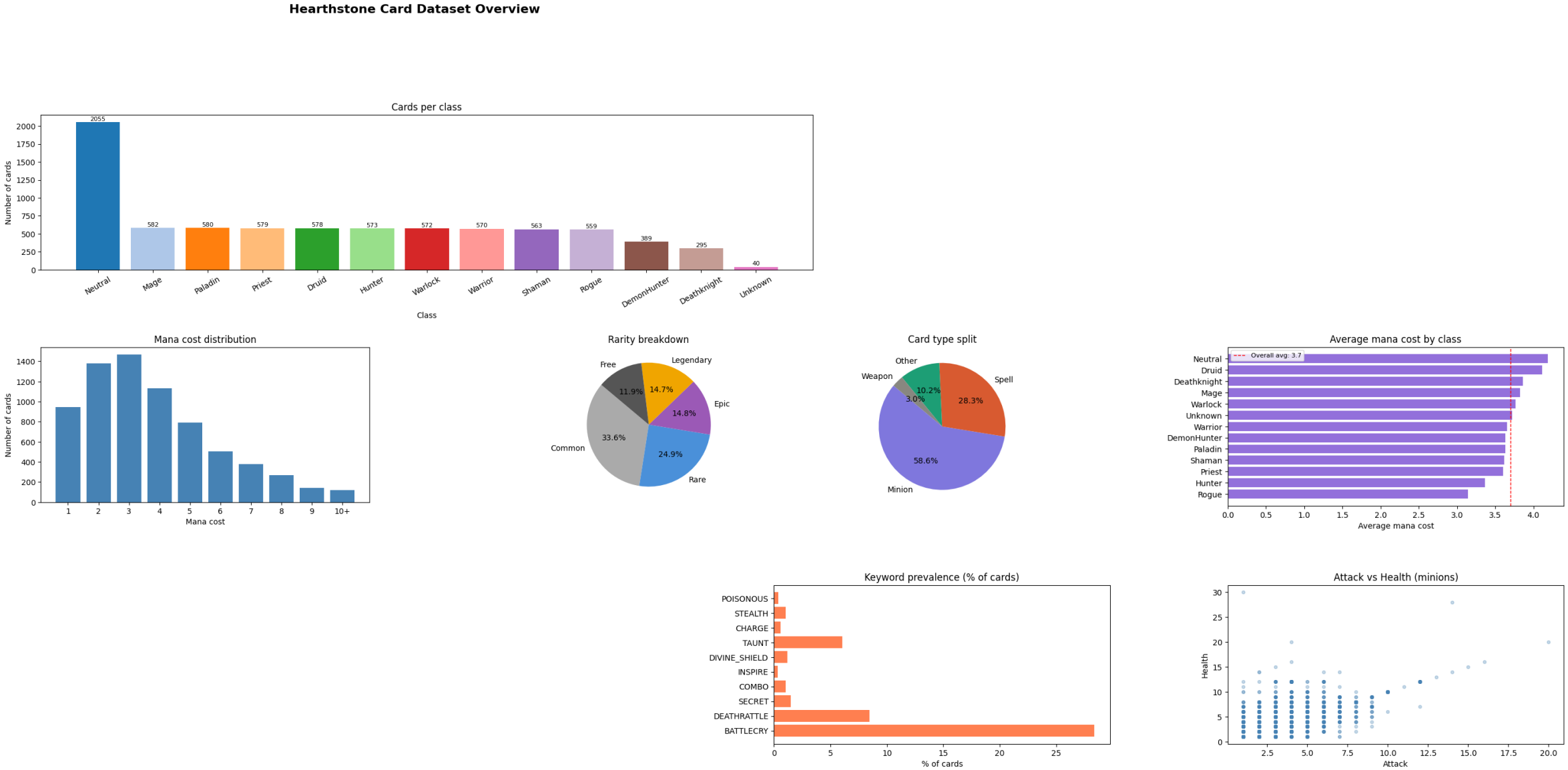
Github repository

UNIVERSITY OF COPENHAGEN

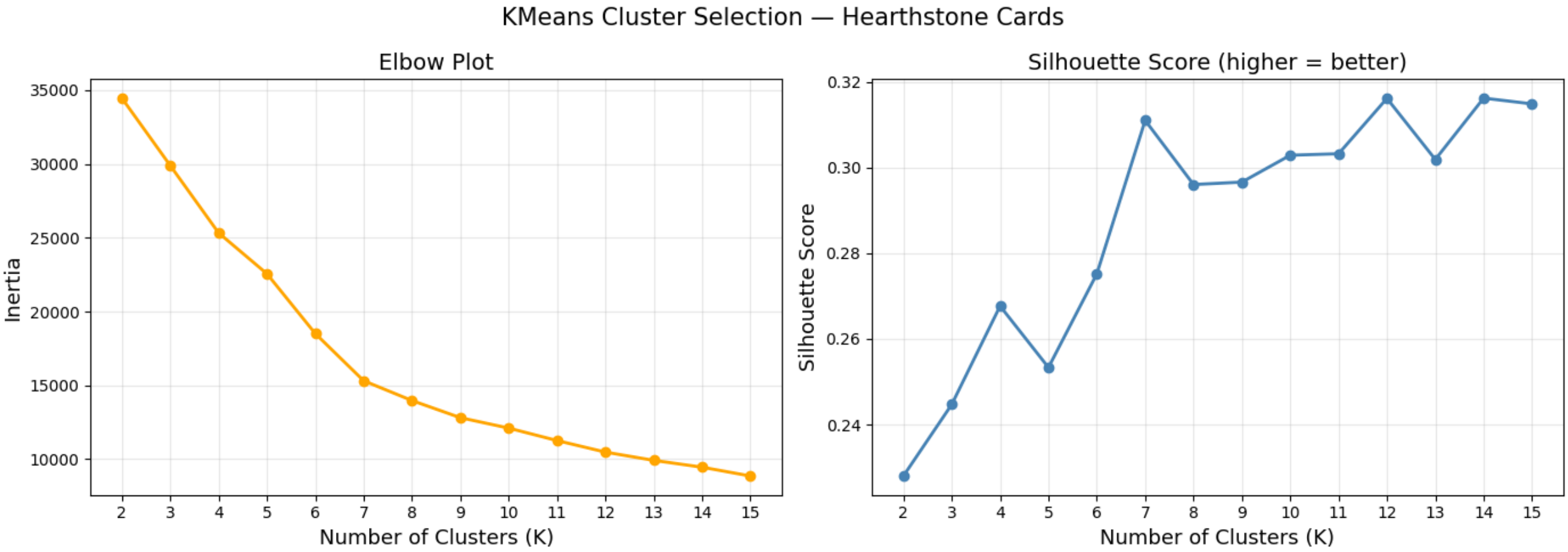
Picture created by Gemini

Appendix

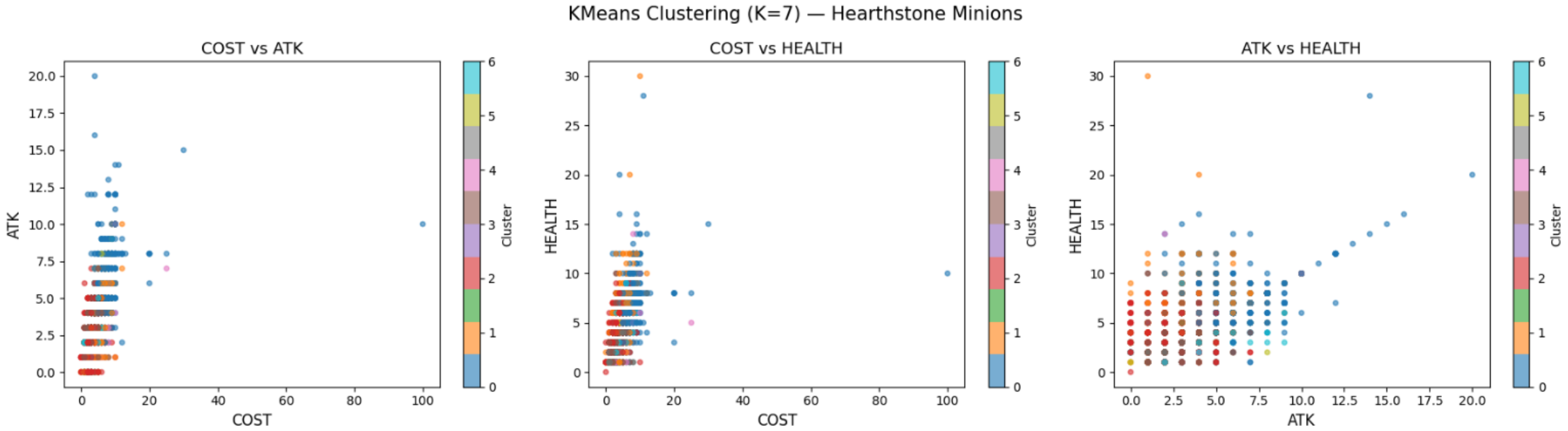
Data visualisation



Data - Clustering



Data - Clustering part 2



ResNet18 - Class Prediction settings

Input

- Card artwork image
- Resolution: **224 × 224 × 3**
- Classes: **12 Hearthstone classes**

Architecture

- Pretrained **ResNet18**
- Final layer replaced with:
  - Dropout (p = 0.4)
  - Fully Connected Layer
  - Softmax output

CONV → POOL → RESBLOCK ×4 → GAP → DROPOUT → DENSE → SOFTMAX

Training

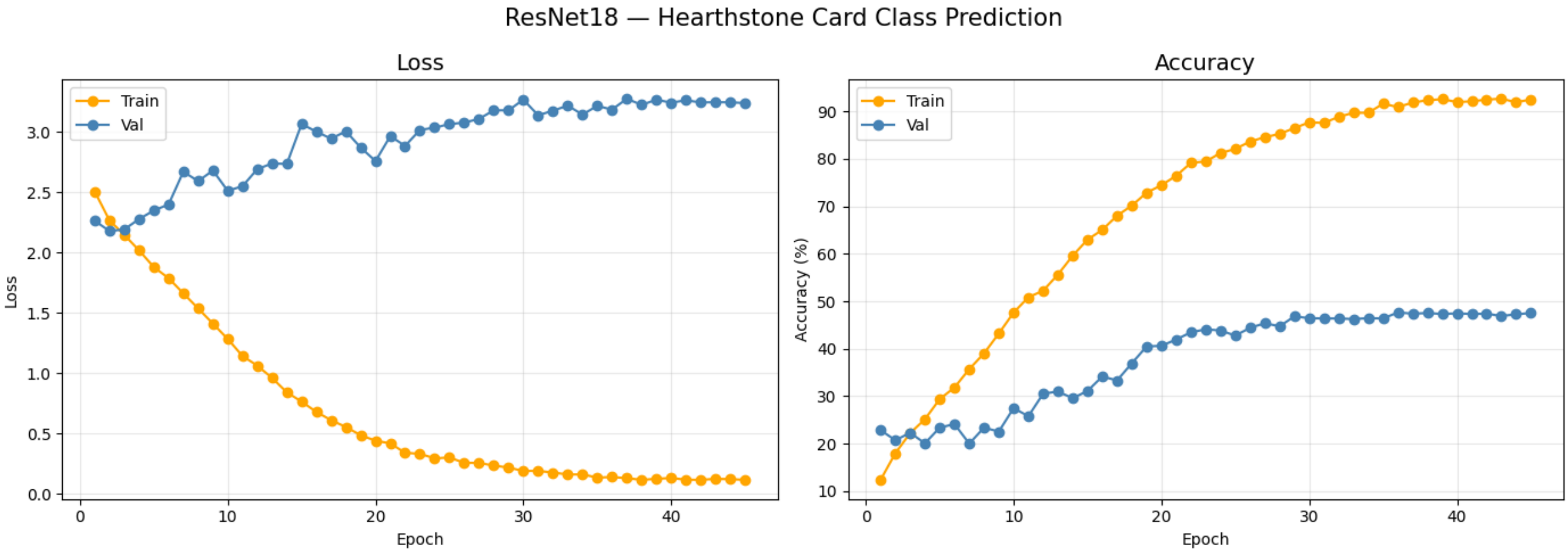
- Optimizer: **Adam**
- Learning Rate: **3×10^{-4}**
- Weight Decay: **1×10^{-4}**
- Scheduler: **Cosine Annealing**
- Epochs: **45**
- Mixed Precision Training (AMP)
- Loss:
  - **Weighted Cross Entropy**
  - Class weights inversely proportional to class frequency

Data Augmentation

Applied only to the training set:

- Random Horizontal Flip
- Random Rotation (±15°)
- Color Jitter
  - Brightness ±30%
  - Contrast ±30%
  - Saturation ±30%
  - Hue ±10%
- Random Grayscale (10%)
- Random Perspective Distortion (30%)
- Random Resized Crop
  - Scale range: 70–100%
- Random Erasing (30%)
- ImageNet Normalization

ResNet18 - Class Prediction plots



DINOv2 - Class Prediction settings

Input

- Input: Card artwork (224×224×3)
- Backbone: Pretrained DINOv2 Vision Transformer

Patch Embedding → Vision Transformer Blocks → Linear Head → Softmax

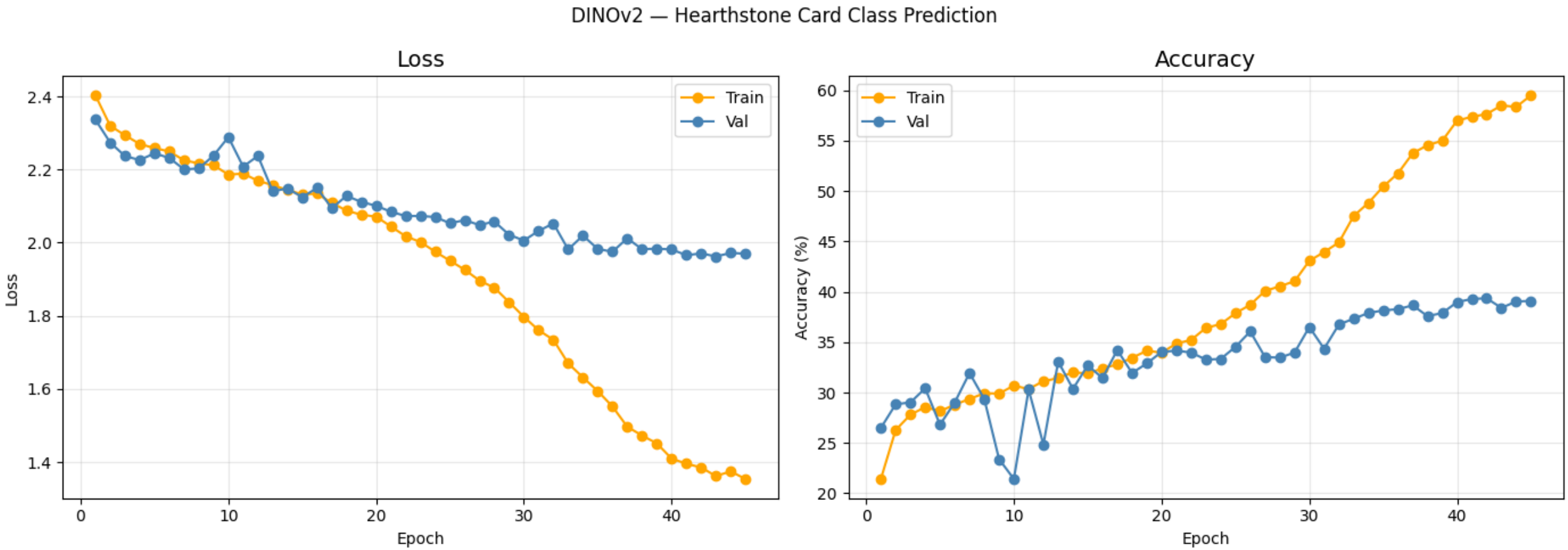
- Fine-tuning: Entire backbone unfrozen
- Classification Head: Linear layer + Softmax
- Optimizer: AdamW
- LR: 5×10^{-5}
- Scheduler: Cosine Annealing
- Loss: Weighted Cross Entropy
- Train/Validation Split: 80/20

Data Augmentation

Applied only to the training set:

- Random Horizontal Flip
- Random Rotation (±15°)
- Color Jitter
  - Brightness ±30%
  - Contrast ±30%
  - Saturation ±30%
  - Hue ±10%
- Random Grayscale (10%)
- Random Perspective Distortion (30%)
- Random Resized Crop
  - Scale range: 70–100%
- Random Erasing (30%)
- ImageNet Normalization

DINOv2 - Class Prediction plots



Deck Generator logbook

- Started out by finding the proper card Data on Github.
- Then used Claude to make the framework of the model since I couldn’t find a prerequisite one.
- Set the Pytorch model up and got it working, Data wasn’t split, it was training on itself with no real deck building direction, certain classes wasn’t working because of outdated data.
- Used Claude to manually pull deck codes to create a meta deck dataset to train and test against.
- Fixed the outdated Data issues by reloading a updated version that I added to the dataset, this fixed the class issues as well
- Optimized the model more and more, it goes through epochs fairly fast when using the GPU
- The Model Works! It is a viable model doing what its supposed to!
- Can now take a (any) Card dataset and make a deck that is based on the current meta in HS. YAY!
- Can also create a dataset of multiple decks (100, 1000, 10000), However the code was slow even with the GPU because it created decks sequentially, (1000 for WARRIOR, 1000 for WARLOCK and so on…). I then changed the code to make the decks in parallel making the the code ~ 9x faster .
- Dataset can now be easily be variables for each class 10000, even 100000 (takes ~15 min).

Supplementary Settings for the DECK GENERATOR Model

Optimizer: AdamW

Loss: Crossentropy

Learning Rate scheduler: Cosine Annealing LR

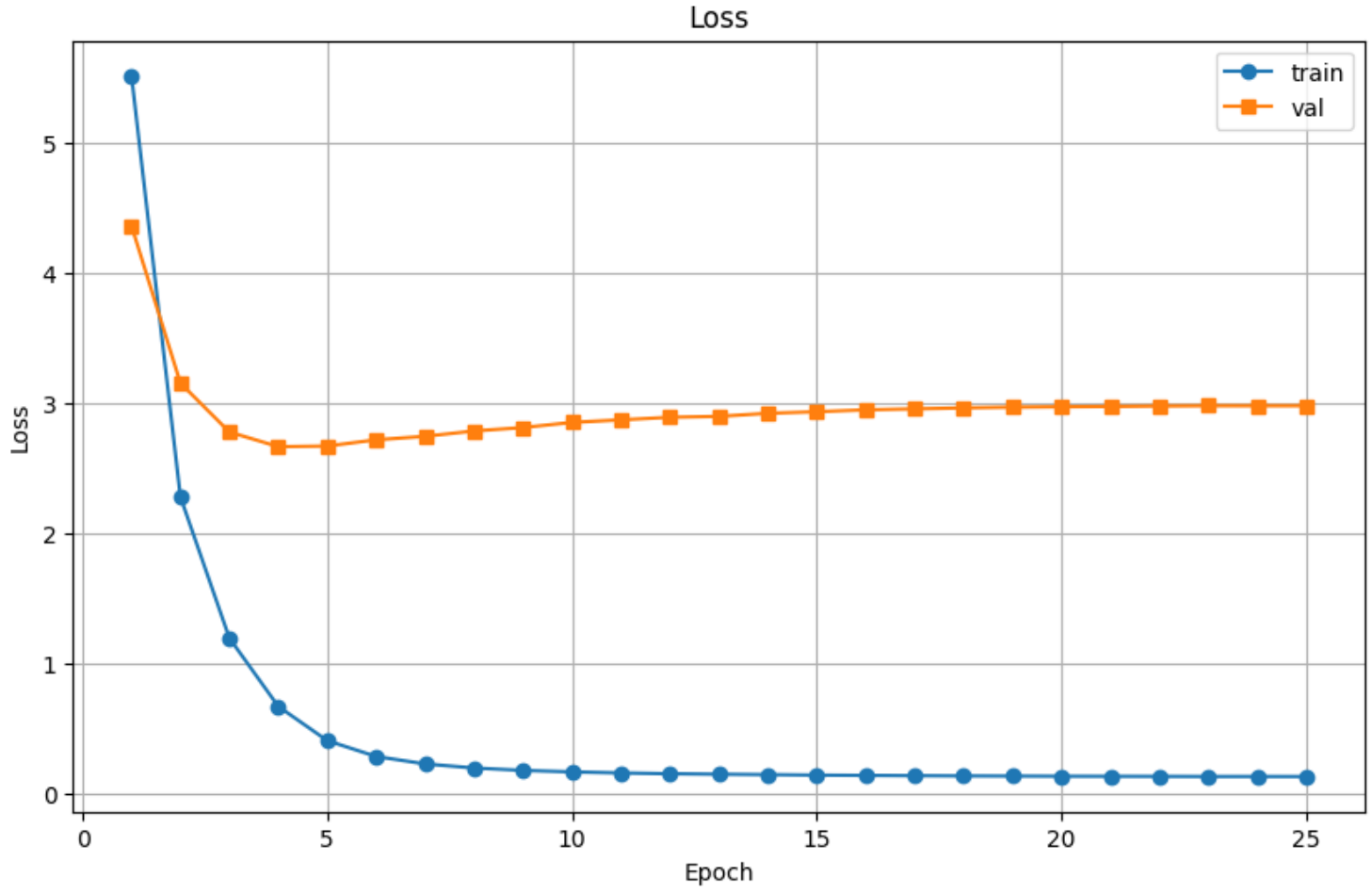
Earlystopping: model.load_state_dict(best_state)

Batch: 64 (GPU cores)

Data from Deck Generator

Best Validation Loss selection (early stopping without stopping) is on.

```
epoch 1/25  train=5.5131  val=4.3615  best_val=4.3615
epoch 2/25  train=2.2799  val=3.1536  best_val=3.1536
epoch 3/25  train=1.1963  val=2.7783  best_val=2.7783
epoch 4/25  train=0.6718  val=2.6658  best_val=2.6658
epoch 5/25  train=0.4094  val=2.6707  best_val=2.6658
epoch 6/25  train=0.2876  val=2.7186  best_val=2.6658
epoch 7/25  train=0.2307  val=2.7452  best_val=2.6658
epoch 8/25  train=0.1994  val=2.7864  best_val=2.6658
epoch 9/25  train=0.1809  val=2.8126  best_val=2.6658
epoch 10/25  train=0.1697  val=2.8521  best_val=2.6658
epoch 11/25  train=0.1614  val=2.8716  best_val=2.6658
epoch 12/25  train=0.1553  val=2.8914  best_val=2.6658
epoch 13/25  train=0.1516  val=2.8982  best_val=2.6658
epoch 14/25  train=0.1484  val=2.9220  best_val=2.6658
epoch 15/25  train=0.1447  val=2.9340  best_val=2.6658
epoch 16/25  train=0.1427  val=2.9486  best_val=2.6658
epoch 17/25  train=0.1405  val=2.9560  best_val=2.6658
epoch 18/25  train=0.1387  val=2.9636  best_val=2.6658
epoch 19/25  train=0.1376  val=2.9691  best_val=2.6658
epoch 20/25  train=0.1364  val=2.9723  best_val=2.6658
epoch 21/25  train=0.1357  val=2.9745  best_val=2.6658
epoch 22/25  train=0.1352  val=2.9781  best_val=2.6658
epoch 23/25  train=0.1341  val=2.9801  best_val=2.6658
epoch 24/25  train=0.1342  val=2.9799  best_val=2.6658
epoch 25/25  train=0.1339  val=2.9800  best_val=2.6658
```



Data from Deck Generator

OLD

Current

Class	N meta	Model	Synth	Δ	CurveKL	Staple%
WARRIOR	43	26.3	0.8	+25.5	0.126	89%
SHAMAN	20	27.9	0.8	+27.1	0.231	8%
ROGUE	84	28.1	1.2	+26.9	0.165	100%
PALADIN	47	24.5	1.0	+23.5	0.211	82%
HUNTER	46	25.9	0.9	+25.0	0.199	1%
DRUID	39	27.9	0.7	+27.1	0.238	48%
WARLOCK	13	29.2	0.3	+29.0	0.094	95%
MAGE	74	28.5	1.3	+27.2	0.239	0%
PRIEST	40	20.9	1.1	+19.7	0.158	85%
DEMONHUNTER	84	27.3	1.0	+26.3	0.255	39%
OVERALL		26.6	0.9	+25.7		

Training-set perplexity on real decks: 132.452 (lower = better fit)

Reading the table:
- Model/Synth = mean card slots (of 30) shared with nearest meta deck.
- Δ = model lift over the synthetic baseline (higher = more meta-aware).
- CurveKL = KL(generated curve || meta curve). Lower = better.
- Staple% = fraction of class staples present in generated decks.

Class	N test	Model	Synth	Δ	CurveKL	Staple%	Self-sim
DEATHKNIGHT	12	25.7	1.0	+24.8	0.131	81%	17.7
DEMONHUNTER	13	23.0	0.8	+22.2	0.215	62%	12.9
DRUID	6	25.5	0.4	+25.1	0.191	88%	22.2
HUNTER	7	13.5	0.7	+12.8	0.162	10%	9.2
MAGE	11	24.3	1.1	+23.2	0.200	79%	10.6
PALADIN	7	21.8	0.8	+21.0	0.208	61%	17.4
PRIEST	6	14.2	0.7	+13.5	0.226	38%	10.0
ROGUE	13	24.3	0.8	+23.5	0.133	100%	19.1
SHAMAN	3	19.7	0.4	+19.3	0.442	8%	23.0
WARLOCK	2	20.2	0.3	+19.8	0.354	95%	24.7
WARRIOR	6	22.0	0.4	+21.6	0.220	66%	15.1
OVERALL		21.3	0.7	+20.6			

Mean CurveKL: 0.226
Mean Staple%: 63%

Held-out TEST perplexity: 20.602 (lower = better generalization)

Reading the table:
- N test = held-out decks for that class.
- Model = mean overlap (of 30) between generated decks and nearest TEST meta deck.
- Synth = same metric for random synthetic decks — the baseline.
- Δ = model lift over baseline. This is the real signal now.
- CurveKL = KL(generated curve || test curve). Lower = better.
- Staple% = fraction of TRAIN-defined class staples present in generations.
- Self-sim = mean pairwise overlap among the 30 generated decks.
  30 = mode collapse (always same deck), low = diverse.

Classes and their added cards

Class	Before	Added	Total
Death Knight	0	274	274
Demon Hunter	44	313	357
Druid	174	391	565
Hunter	169	387	556
Mage	176	395	571
Neutral	875	1,151	2,026
Paladin	169	385	554
Priest	184	387	571
Rogue	173	367	540
Shaman	170	368	538
Warlock	177	383	560
Unspecified	0	35	35
Total	2,479	5,211	7,690

Features for the Machine Learning Models

Feature group	Example features	Purpose
Deck statistics	`avg_cost`, `avg_attack`, `avg_health`	Describes overall deck strength/curve
Card type counts	`num_minions`, `num_spells`, `num_weapons`, `num_heroes`	Describes deck composition
Rarity/count features	`num_legendary`, `missing_cards`	Captures rarity and missing card data
Class encoding	`class_name_MAGE`, `class_name_WARLOCK`, etc.	Tells the model which class the deck belongs to
Card identity features	`card_CORE_EX1_145`, `card_TLC_603`, etc.	Shows which specific cards are in the deck

Champions vs. Rest of Their Class

Class	Champion deck	Opponents	Games	Win-Lose-Draw	Champion win rate	Win rate range
Warlock	WARLOCK_3	9	900	753-146-1	83.7%	61–94%
Mage	MAGE_0	9	900	738-161-1	82.0%	63–95%
Demon Hunter	DEMONHUNTER_5	9	900	710-187-3	78.9%	58–94%
Hunter	HUNTER_0	9	900	702-197-1	78.0%	52–95%
Death Knight	DEATHKNIGHT_5	9	900	701-195-4	77.9%	49–94%
Shaman	SHAMAN_8	9	900	678-221-1	75.3%	56–96%
Paladin	PALADIN_1	9	900	629-271-0	69.9%	42–94%
Druid	DRUID_4	9	900	616-282-2	68.4%	38–98%
Warrior	WARRIOR_0	9	900	611-287-2	67.9%	45–89%
Rogue	ROGUE_0	9	900	485-414-1	53.9%	40–66%
Priest	PRIEST_5	9	900	370-524-6	41.1%	14–73%

XGBoost Feature Importance

Feature	Importance
card_CORE_EX1_145	0.0859
avg_attack	0.0771
card_TLC_603	0.0441
num_minions	0.0413
card_TLC_252	0.0396
card_SC_015	0.0307
card_CORE_TSC_827	0.0287
num_spells	0.0273
class_name_PALADIN	0.0273
card_TLC_249	0.0268

Simulation Scale

Experiment	Setup	Games	Crashes
Full all-vs-all tournament	110 decks, 5,995 matchups, 20 games each	119,900	0
Champion tests	11 class champions, 9 opponents each, 100 games each	9,900	0
Total evaluated games	Full tournament + champion tests	**129,800**	**0**