Advance CNN

[GAN, Transfer Learning]

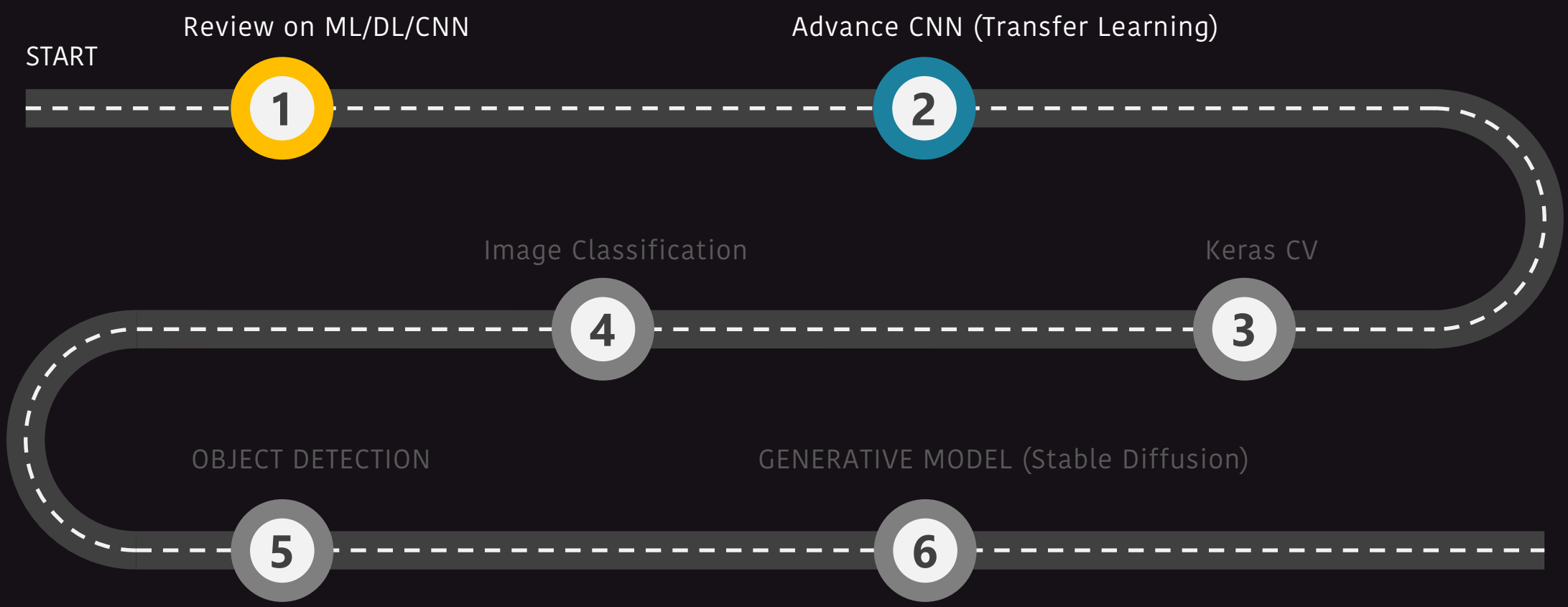
Powered by

Keras CV

Related Resources

https://arif.works/unikl

LEARNING JOURNEY



2.0 Advance CNN

- Generative Adversarial Network
- Transfer Learning
- Fine Tuning
- Hands-on

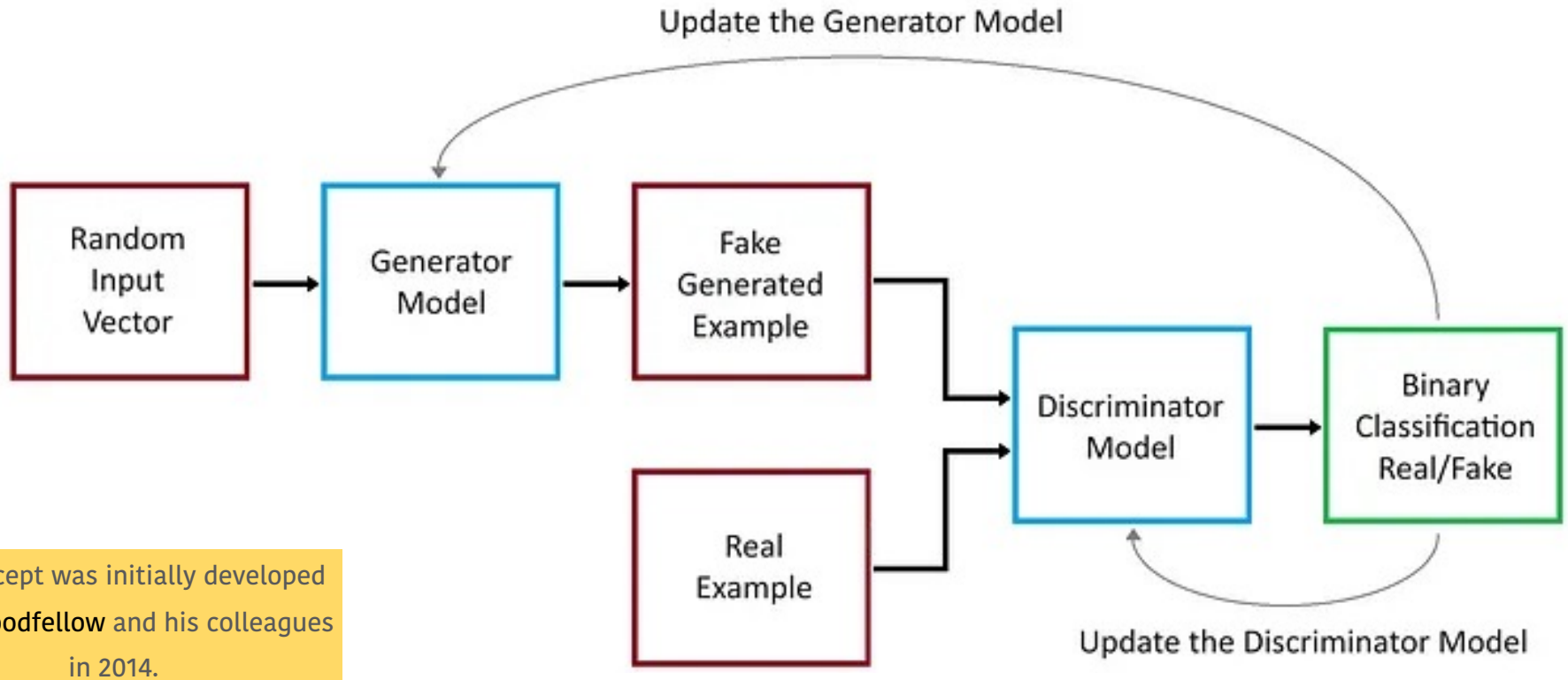
Generative Model

Figure 1: Images generated by a GAN created by NVIDIA.

Generative Model

Generative Adversarial Network (GAN)

- ✓ It is a **deep learning architecture**.
- ✓ It trains **two neural networks** to compete against each other to generate more authentic new data from a given training dataset.



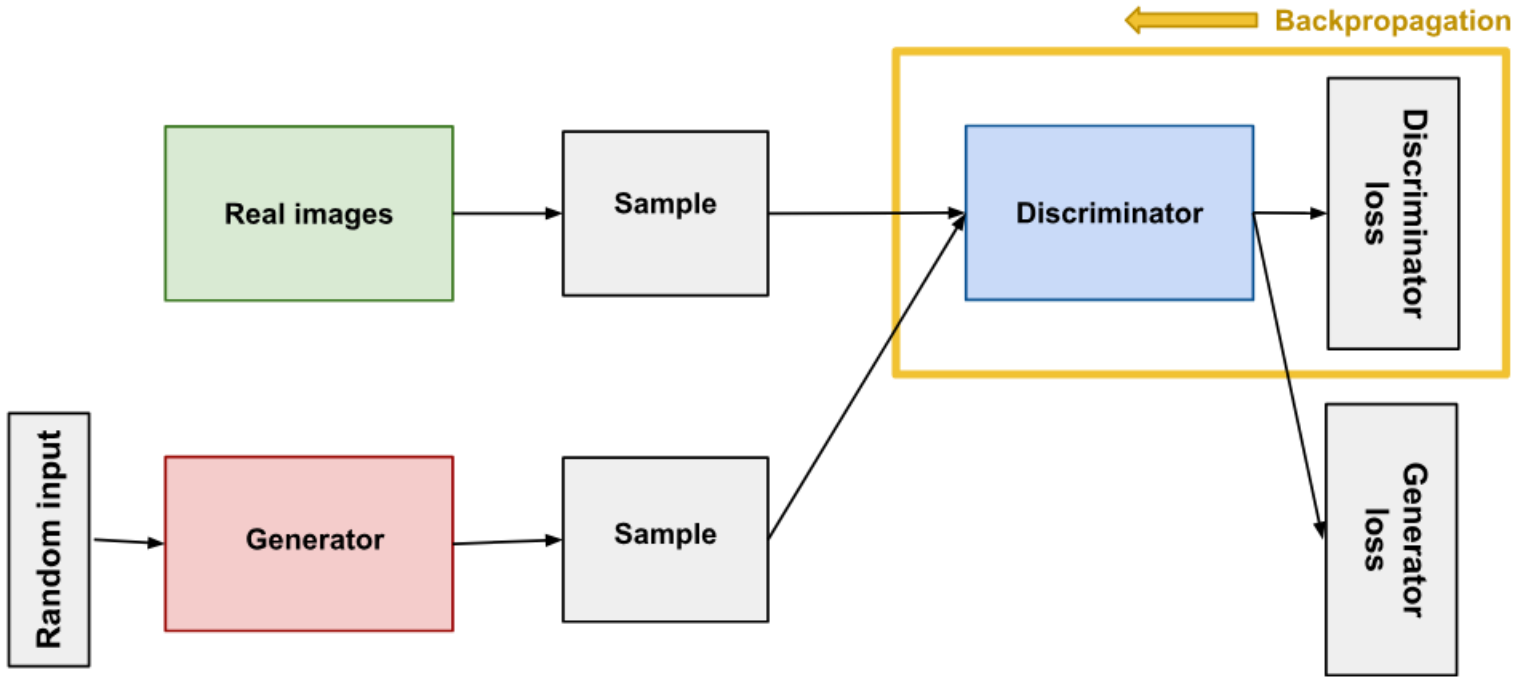
The concept was initially developed by **Ian Goodfellow** and his colleagues in 2014.

Generative Model

GAN: Discriminator

The discriminator's training data comes from two sources:

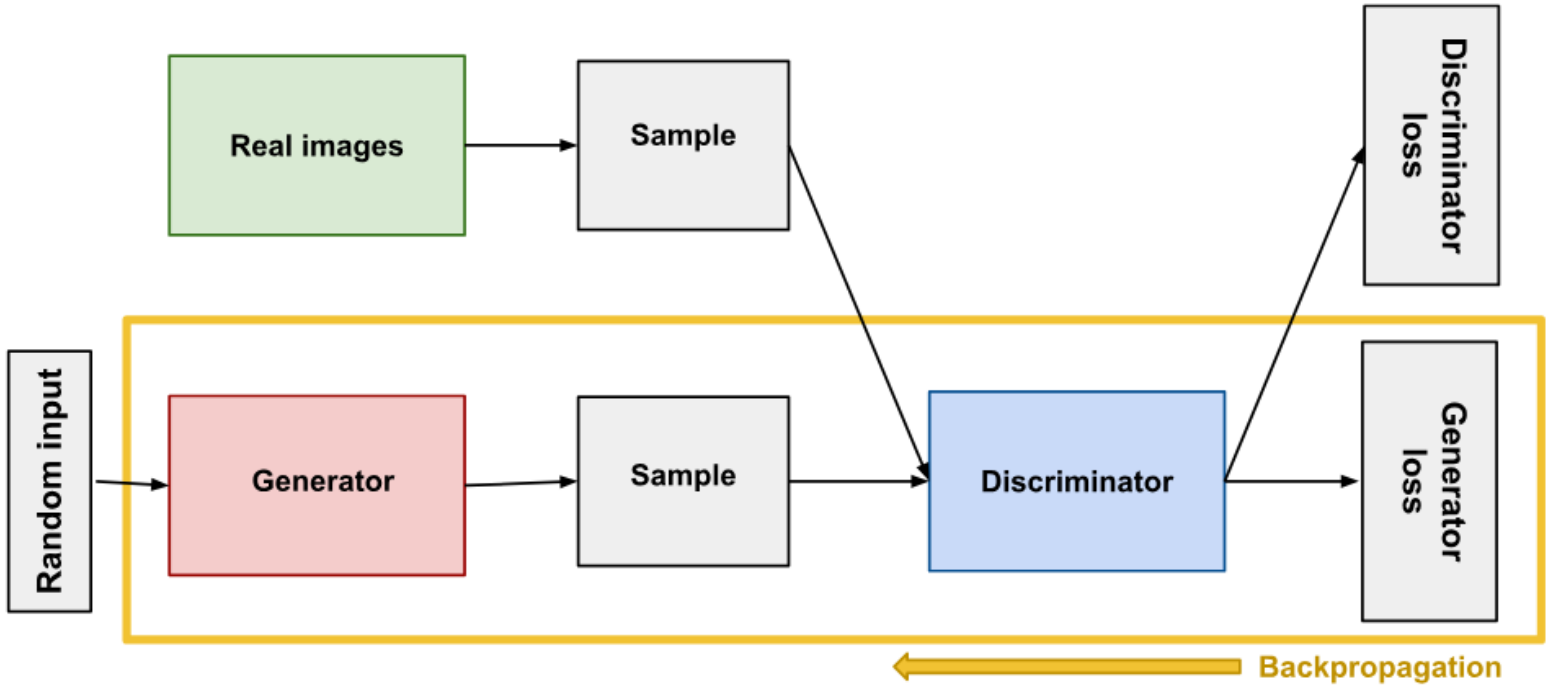
- ✓ **Real data** instances, such as real pictures of people. The discriminator uses these instances as positive examples during training.
- ✓ **Fake data** instances created by the generator. The discriminator uses these instances as negative examples during training.



Generative Model

GAN: Generator

The generator part of a GAN learns to create fake data by incorporating feedback from the discriminator. It learns to make the discriminator classify its output as real.



Generative Model

GAN: Training

GAN training proceeds in alternating periods:

1.The discriminator trains for one or more epochs.

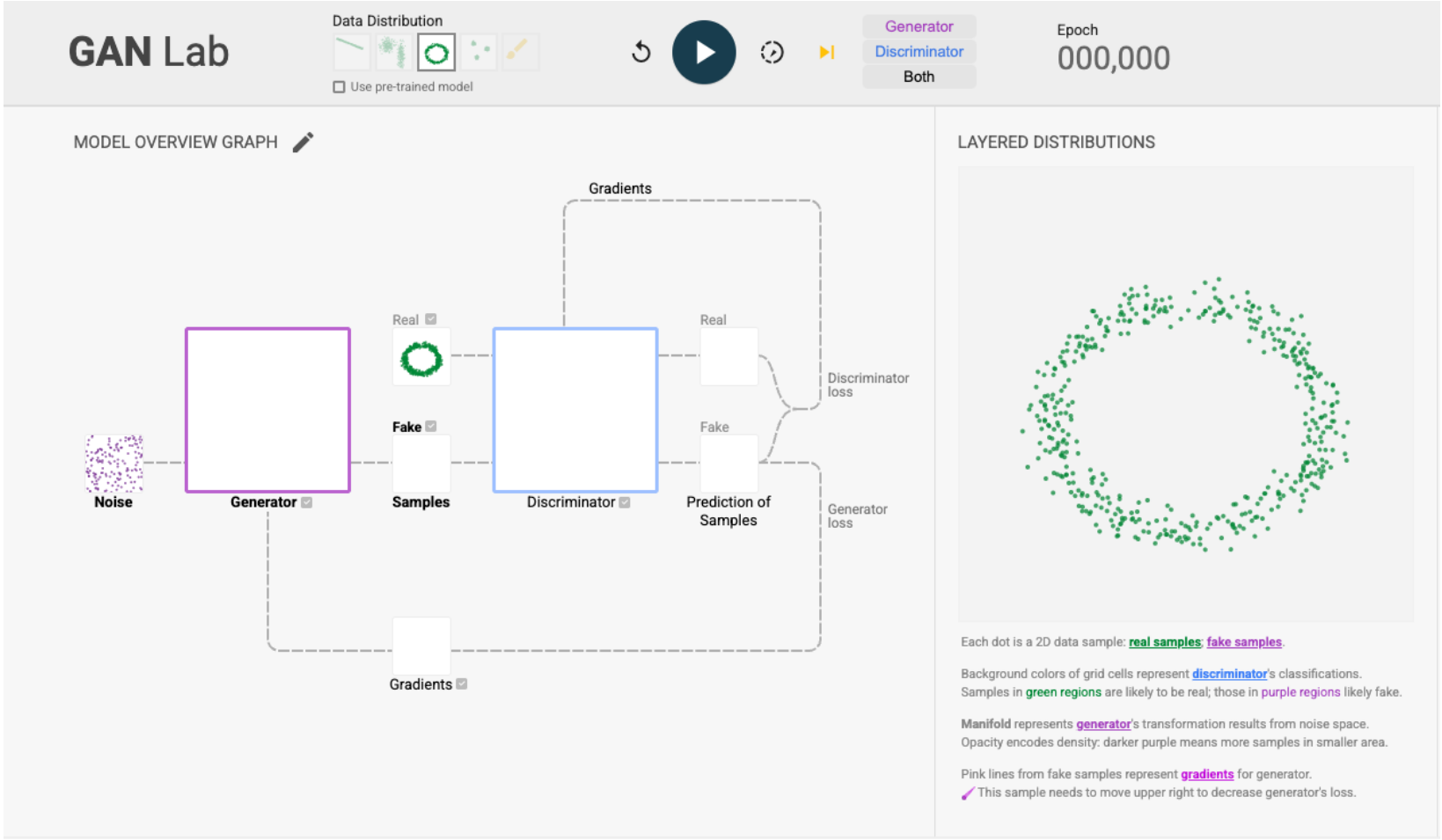
2.The generator trains for one or more epochs.

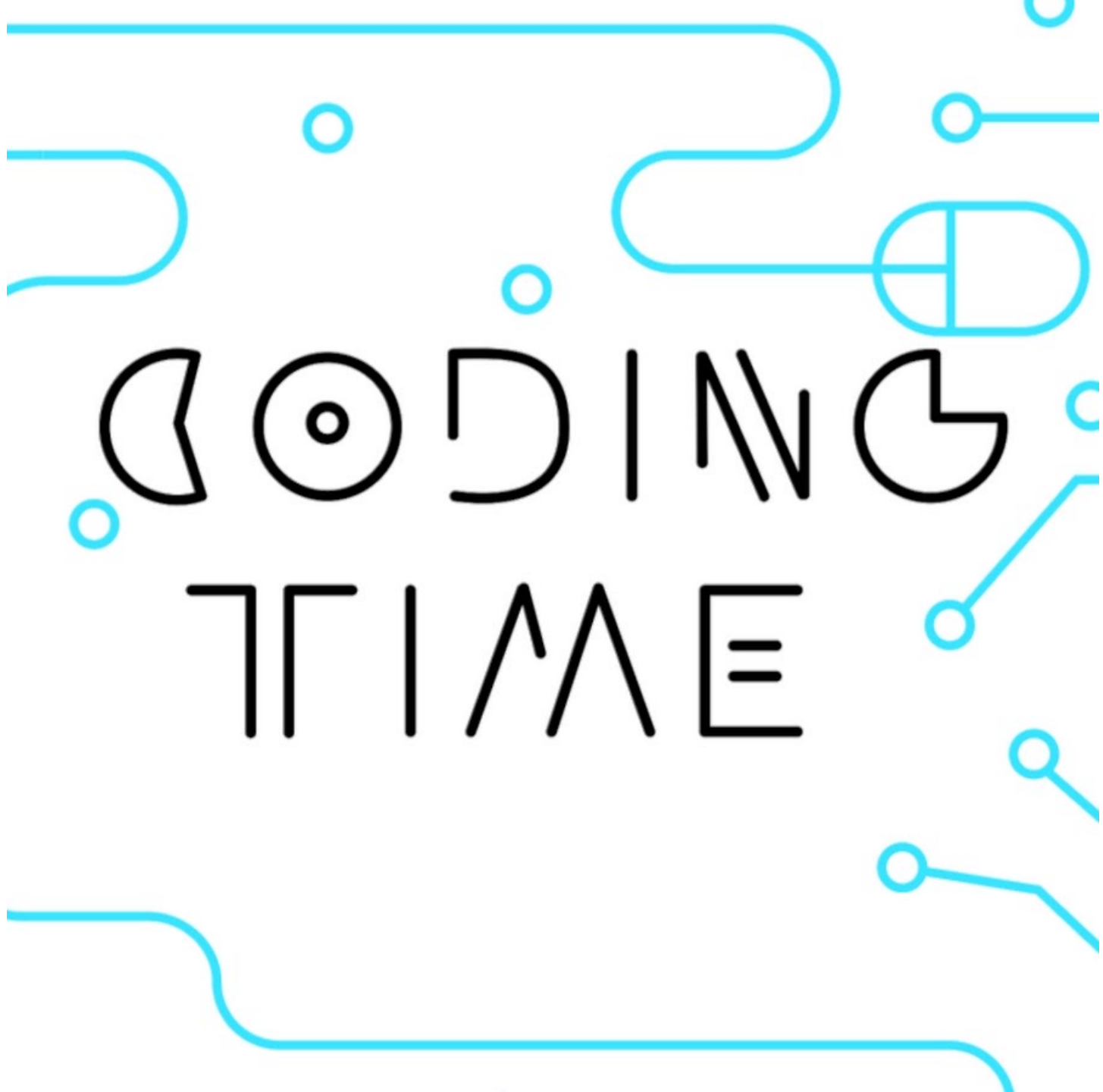
3.Repeat steps 1 and 2 to continue to train the generator and discriminator networks.

Generative Model

Generative Adversarial Network (GAN)

https://poloclub.github.io/ganlab/




CODING
TIME

Transfer Learning

Transfer Learning is a advance CNN technique where a model developed for one task is reused as the starting point for a model on a second task.

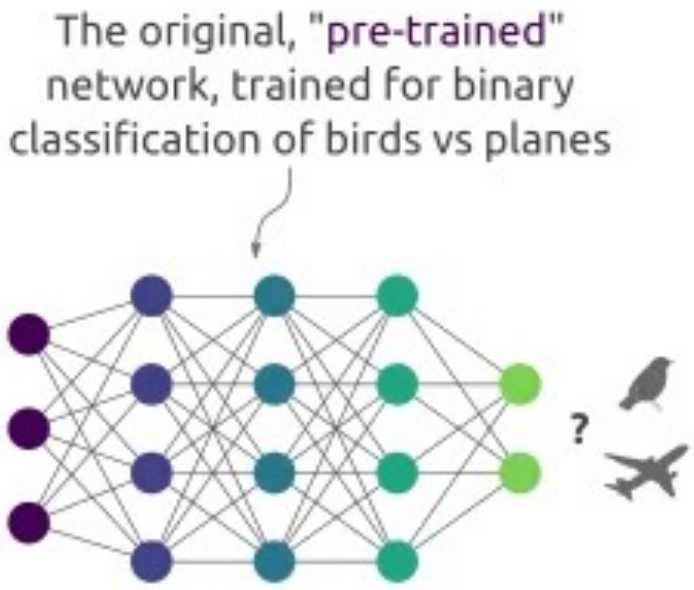
In computer vision, this usually means using a pre-trained model (like VGG16, ResNet, or InceptionV3) that has been trained on a large dataset (like ImageNet) and adapting it for our specific use case.

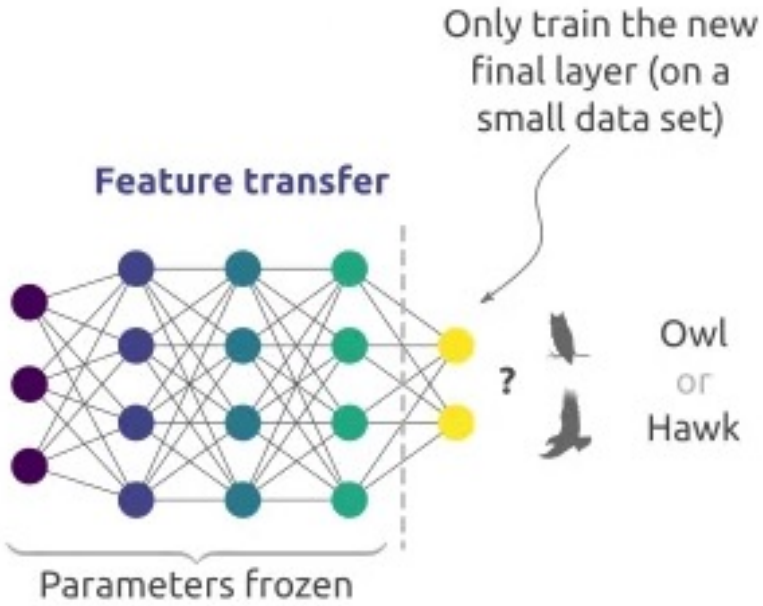
https://www.learndatasci.com/tutorials/hands-on-transfer-learning-keras/

Transfer Learning

Types of Transfer Learning

Feature transfer uses the network as is, with new final layer

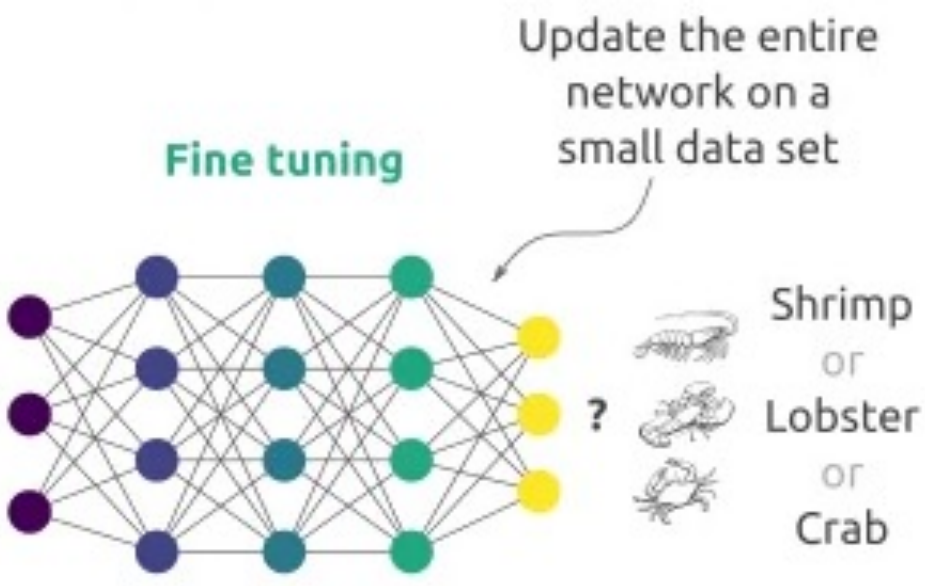




Transfer Learning

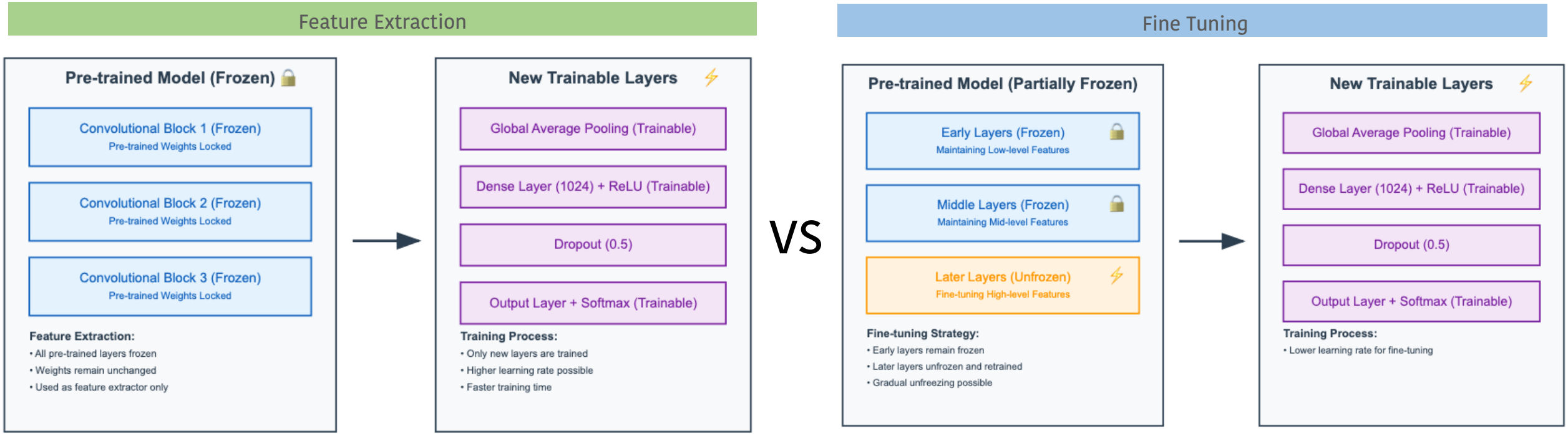
Types of Transfer Learning

Fine Tuning updates the entire network



Transfer Learning

Types of Transfer Learning



Transfer Learning

Steps for Implementation [Feature Extraction]

1. Dataset Preparation
- Download flowers dataset
- Create directory structure
- Split into train/validation (80/20)
- Organize into class folders
- Check data distribution

2. Data Augmentation
- ImageDataGenerator setup
- Rescale pixels (1/255)
- Rotation (30 degrees)
- Width/Height shift (0.2)
- Horizontal flip

3. Create Data Generators
- Set batch size (32)
- Configure image size (224x224)
- Set class mode (categorical)
- Enable shuffling
- Create validation generator

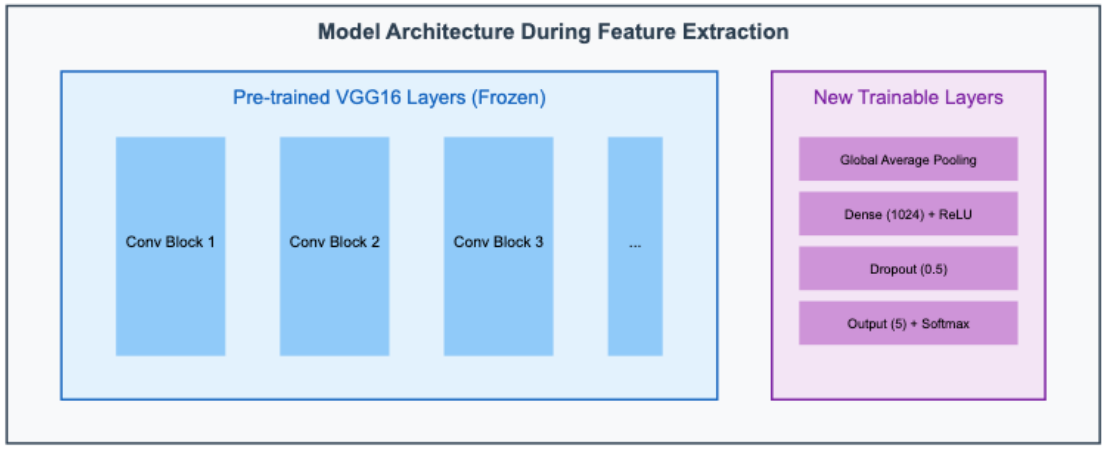
4. Base Model Setup
- Load VGG16 (no top)
- ImageNet weights
- Freeze all layers
- Set input shape (224,224,3)
- trainable = False

5. Add New Layers
- Global Average Pooling
- Dense (1024) + ReLU
- Dropout (0.5)
- Dense (512) + ReLU
- Output (5) + Softmax

6. Compile Model
- Adam optimizer
- Learning rate (1e-4)
- Categorical crossentropy
- Accuracy metric
- Initialize weights

7. Training Setup
- Set epochs (15)
- Configure callbacks
- Early stopping (val_loss)
- Patience = 3
- Save best weights

8. Train and Evaluate
- Fit model
- Monitor training process
- Plot training history
- Evaluate performance
- Save final model



Transfer Learning

Steps for Implementation [Fine Tuning]

1. Load Feature Extraction Model
- Load trained model
- Check layer configurations
- Review model architecture
- Verify weights loaded
- Initialize new model state

2. Data Preparation
- Same augmentation as before
- Verify data generators
- Check batch size settings
- Confirm data splits
- Validate preprocessing

3. Layer Configuration
- Unfreeze last 4 Conv layers
- Keep early layers frozen
- Set layer.trainable = True
- Verify trainable params
- Check layer status

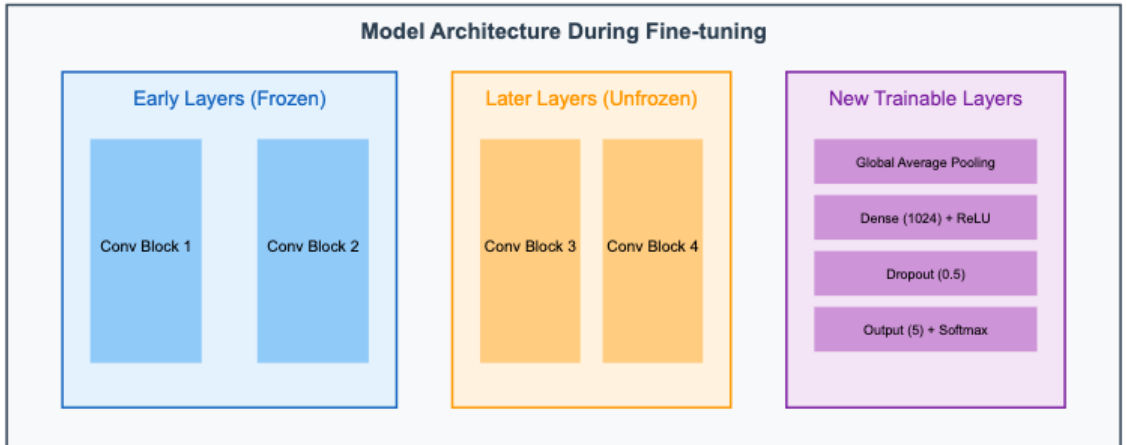
4. Model Compilation
- Lower learning rate (1e-5)
- Adam optimizer
- Categorical crossentropy
- Accuracy metrics
- Verify compilation

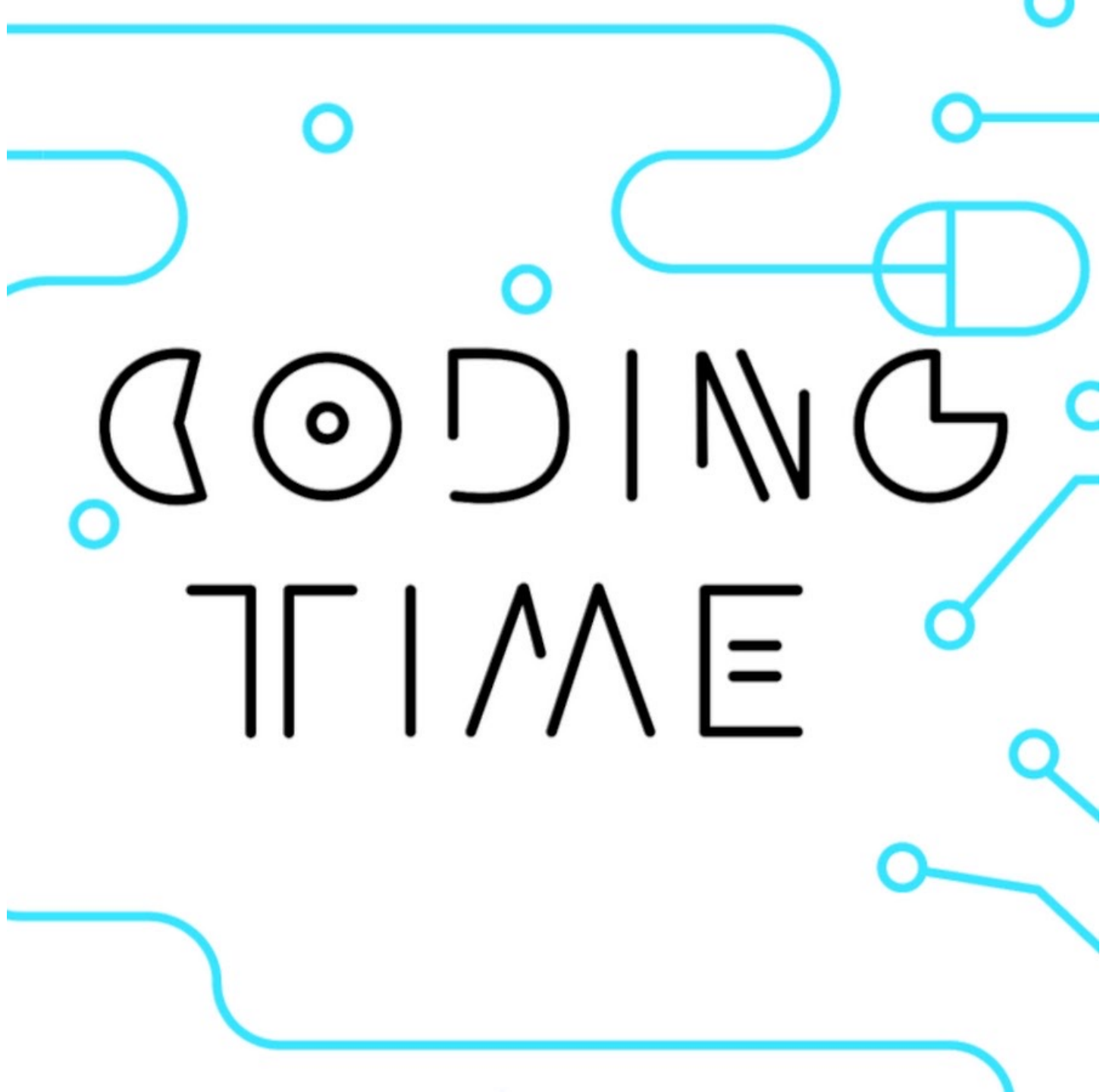
5. Training Configuration
- Set epochs (10-15)
- Configure callbacks
- Early stopping setup
- Model checkpointing
- Learning rate schedule

6. Training Process
- Execute model.fit()
- Monitor loss/accuracy
- Track validation metrics
- Watch for overfitting
- Save checkpoints

7. Model Evaluation
- Plot training history
- Compare with feature extraction
- Analyze performance gains
- Check confusion matrix
- Validate on test set

8. Save and Deploy
- Save final model
- Export model weights
- Document performance
- Prepare for inference
- Test deployment




CODING
TIME