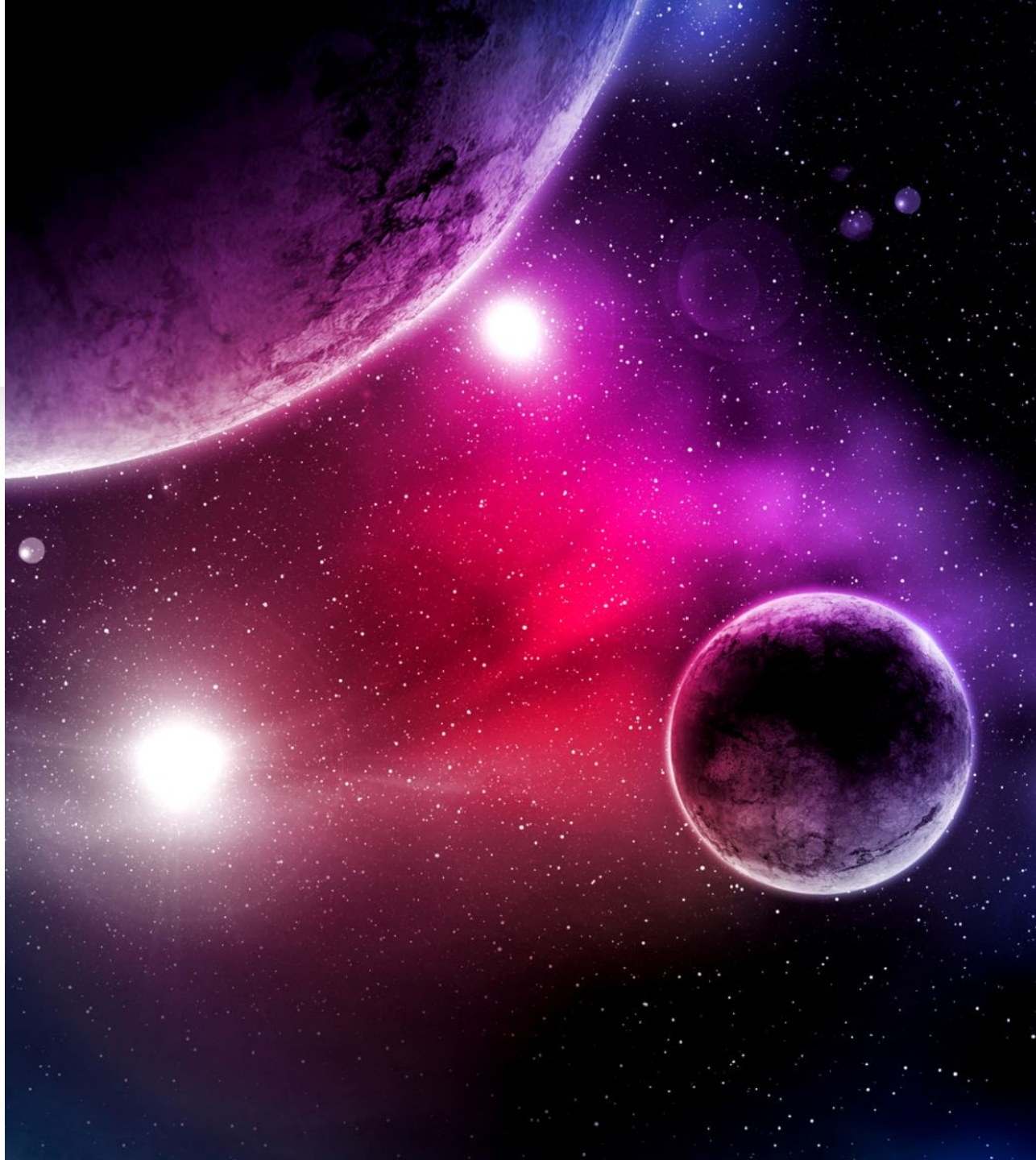
Detecting New Worlds with Machine Learning through Light Curve Analysis

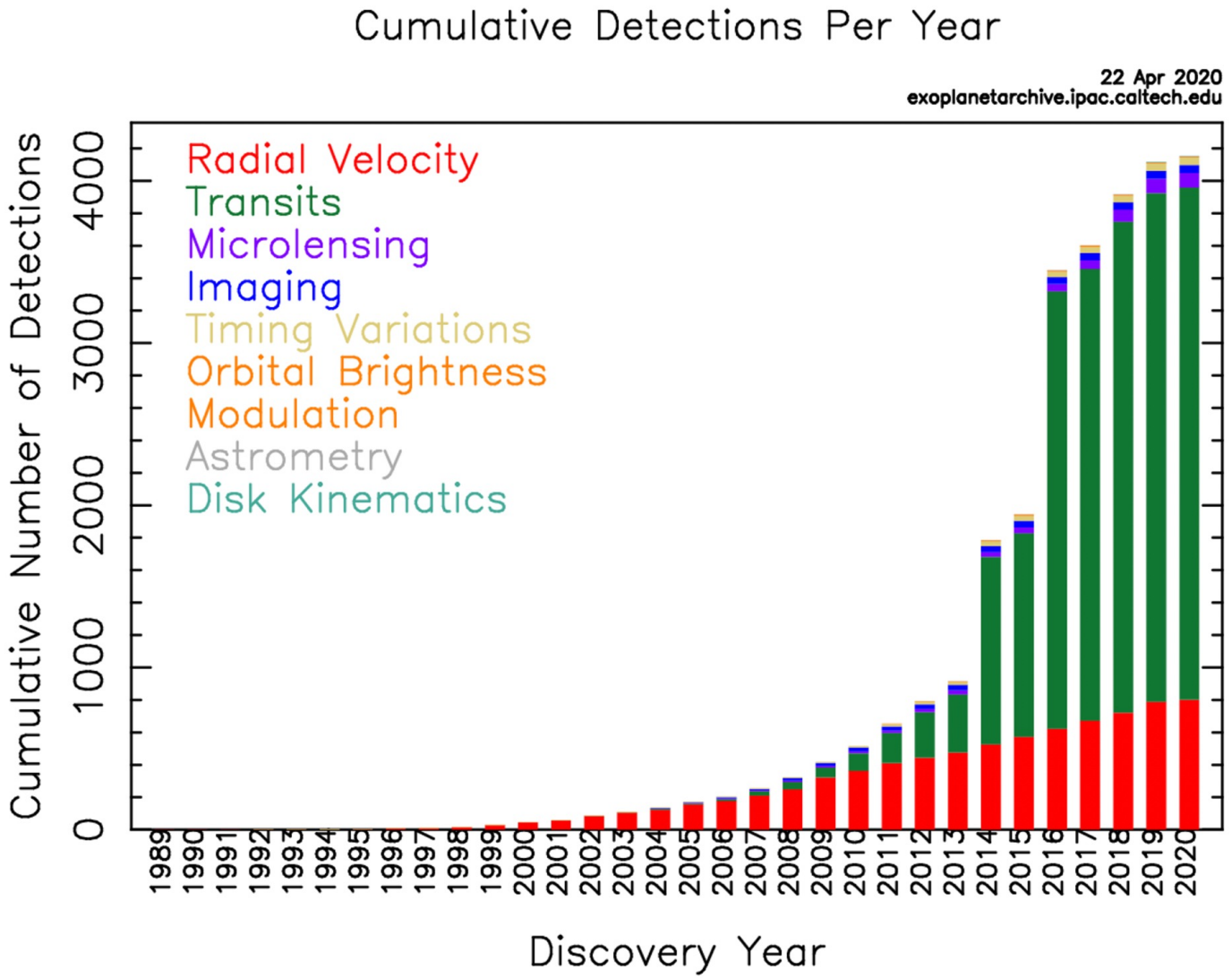
by
Vladimir Bautista
and
Dimna Prado

Background

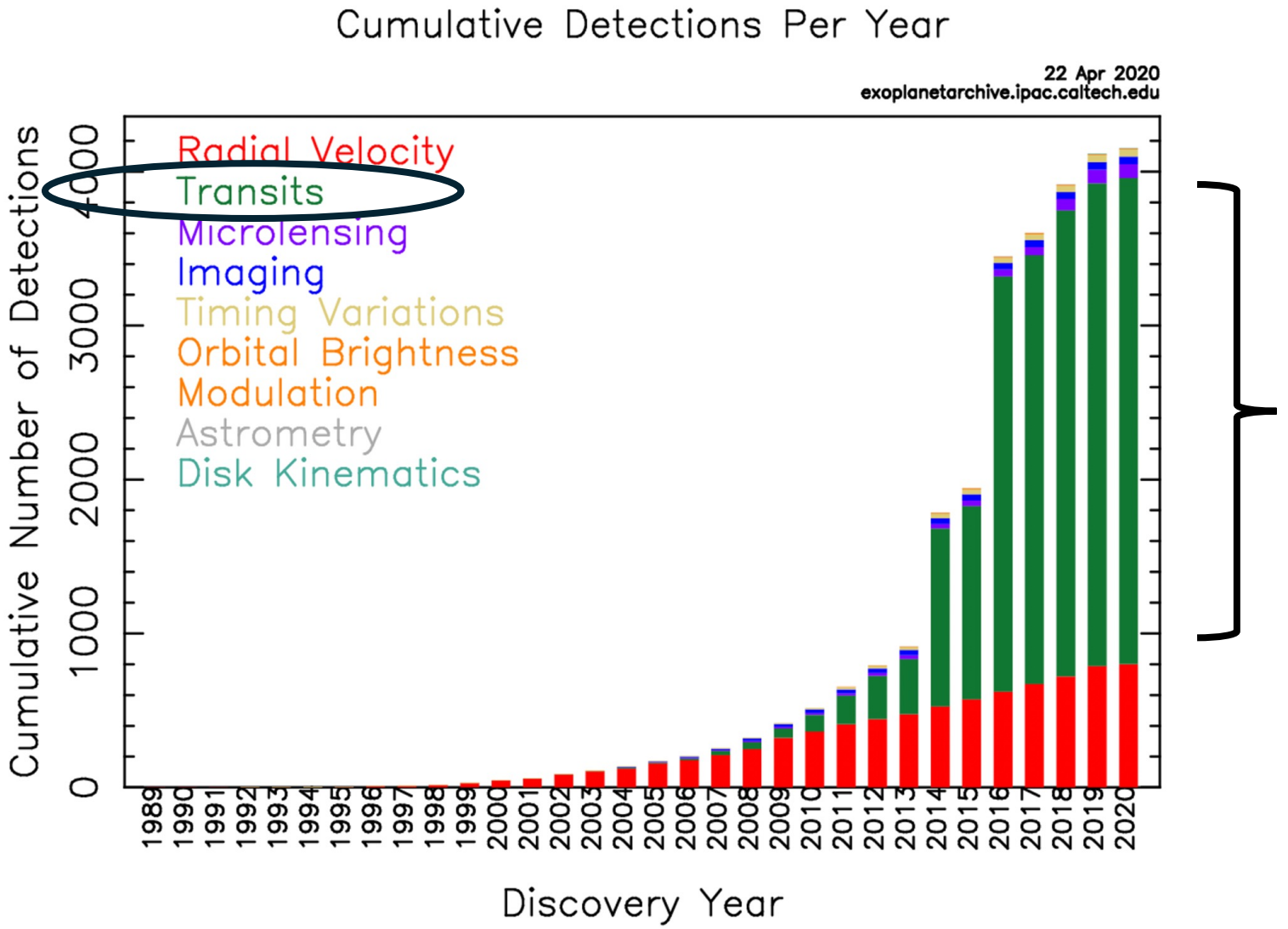
- Exoplanet - planet that orbits another star outside our solar system
- Some are possibly habitable
- 5,806 confirmed exoplanets (NASA Exoplanet Archive)

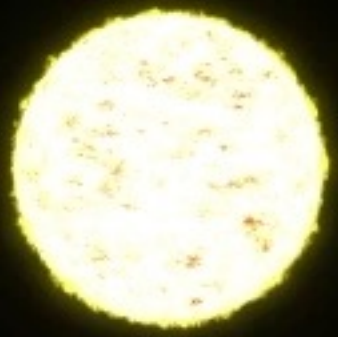


Exoplanet Detection



Exoplanet Detection






LIGHT
TIME


LIGHT
TIME

Objective

Use	Use **machine learning** to classify light curve data
Leverage	Leverage **Stochastic Gradient Descent** for efficient training
Predict	Predict whether a system hosts an **exoplanet** or not

Stochastic Gradient Descent (SGD)

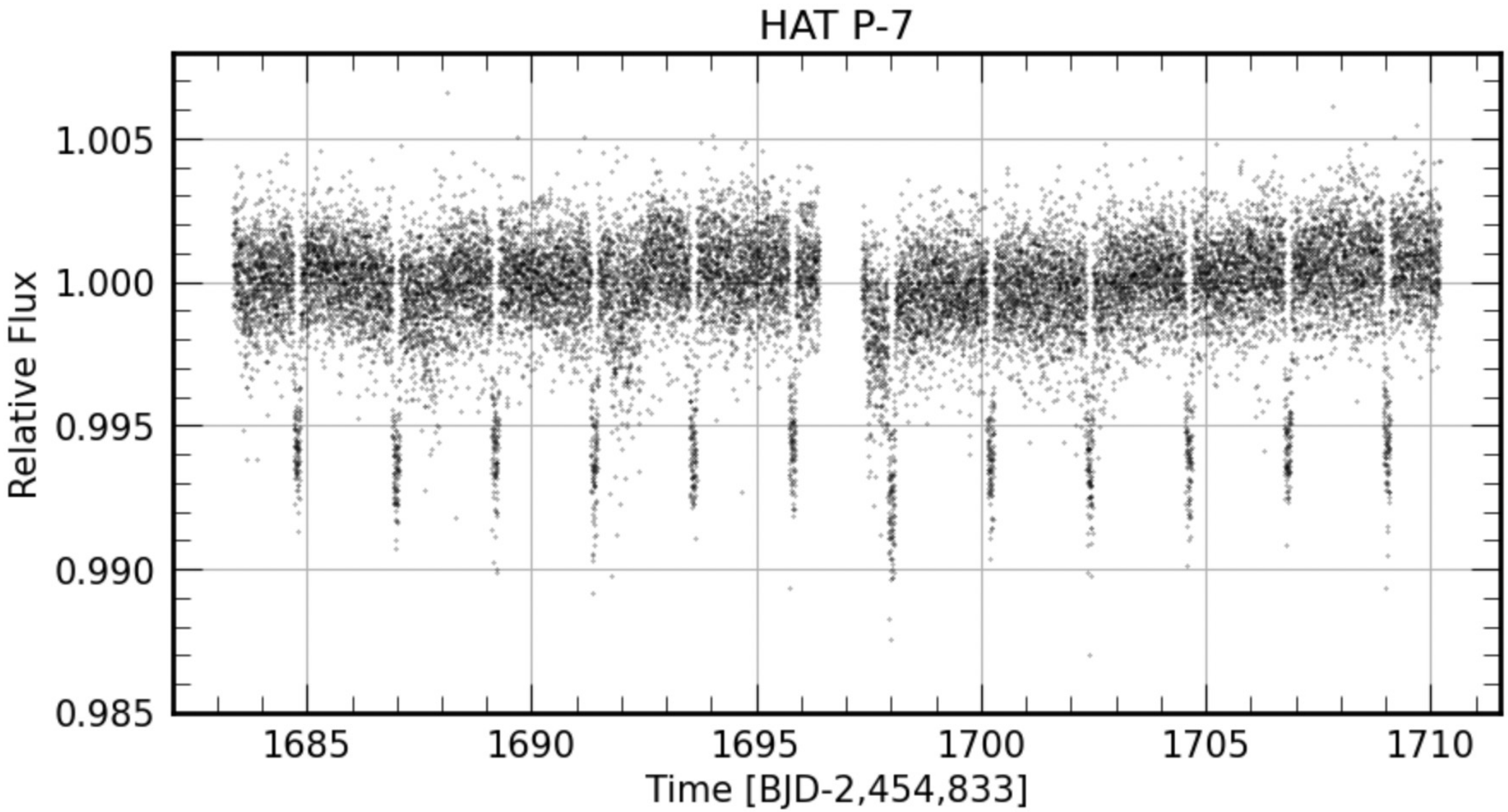
- Optimization algorithm used to train machine learning models
- **Minimizes** the **loss function** by iteratively updating model parameters
- Works by estimating gradients using **random** batches of data

Stochastic Gradient Descent (SGD)

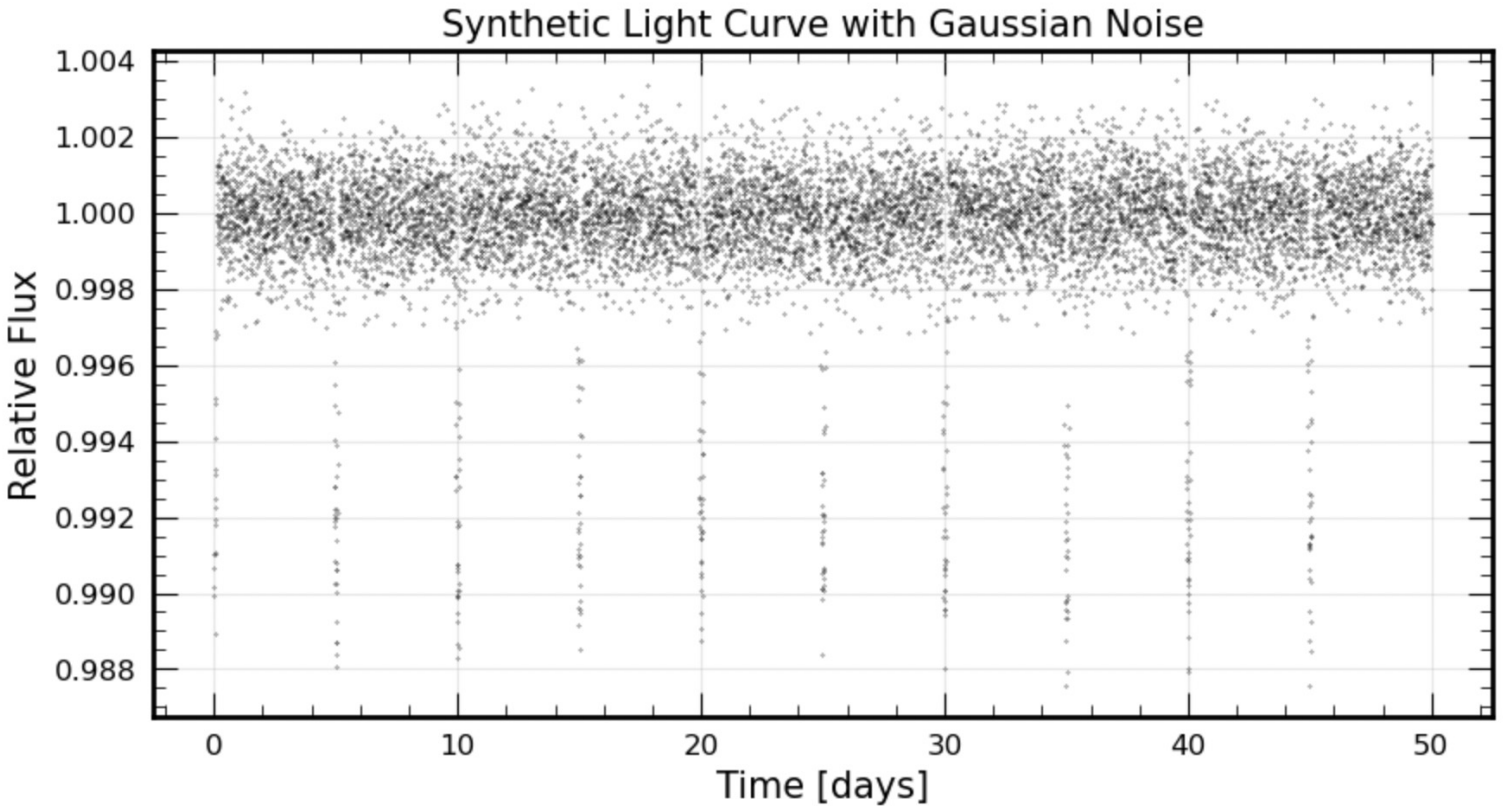
- **Step 1:** Select a random batch of data
- **Step 2:** Compute the gradient of the loss function
- **Step 3:** Update model parameters using the gradient

Advantages

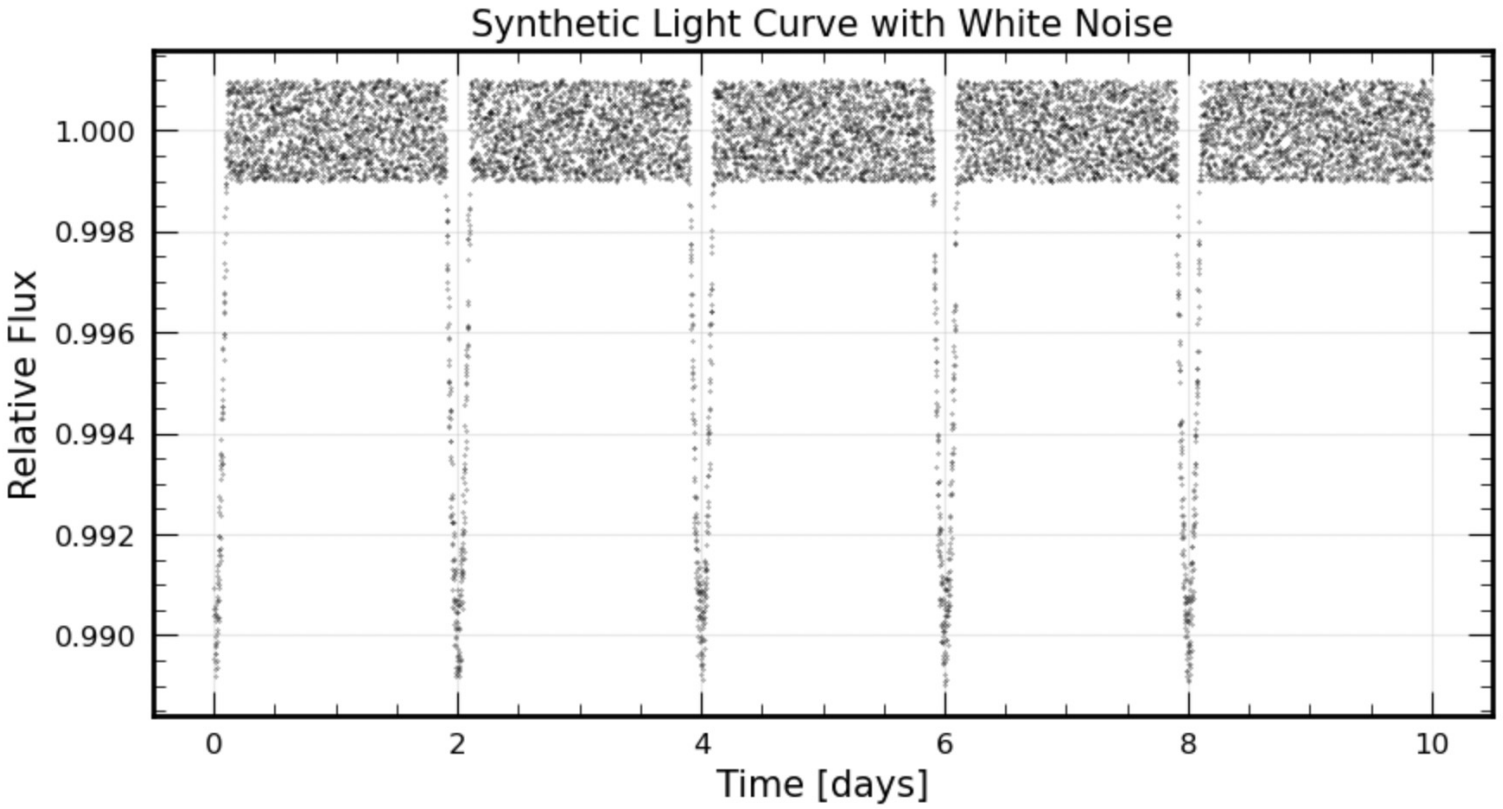
- **Efficient** for large datasets
- Introduces **randomness** to help escape local minima
- Converges **faster** than Gradient Descent


HAT P-7
Relative Flux
1.005
1.000
0.995
0.990
0.985
1685
1690
1695
1700
1705
1710
Time [BJD-2,454,833]

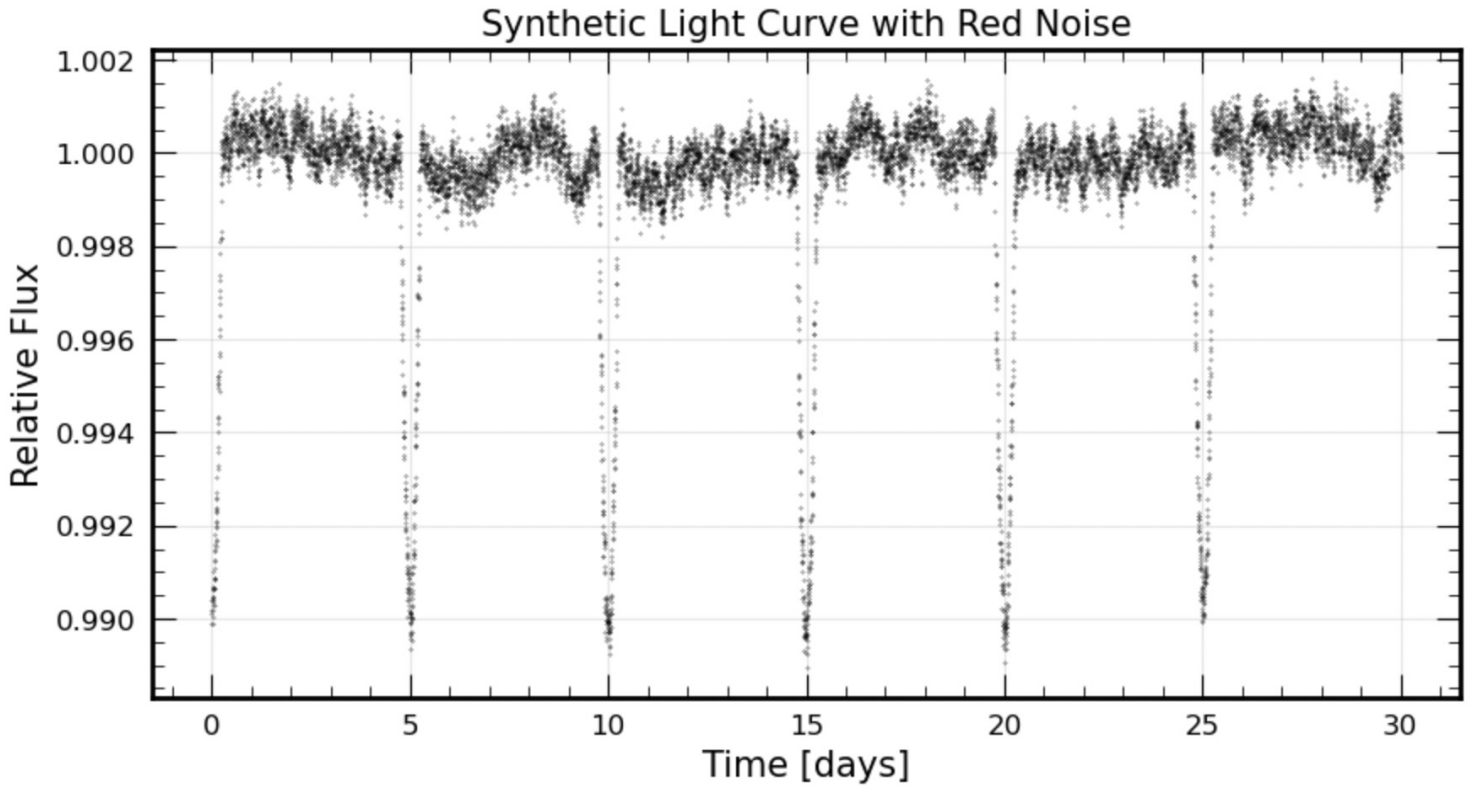
Training Data



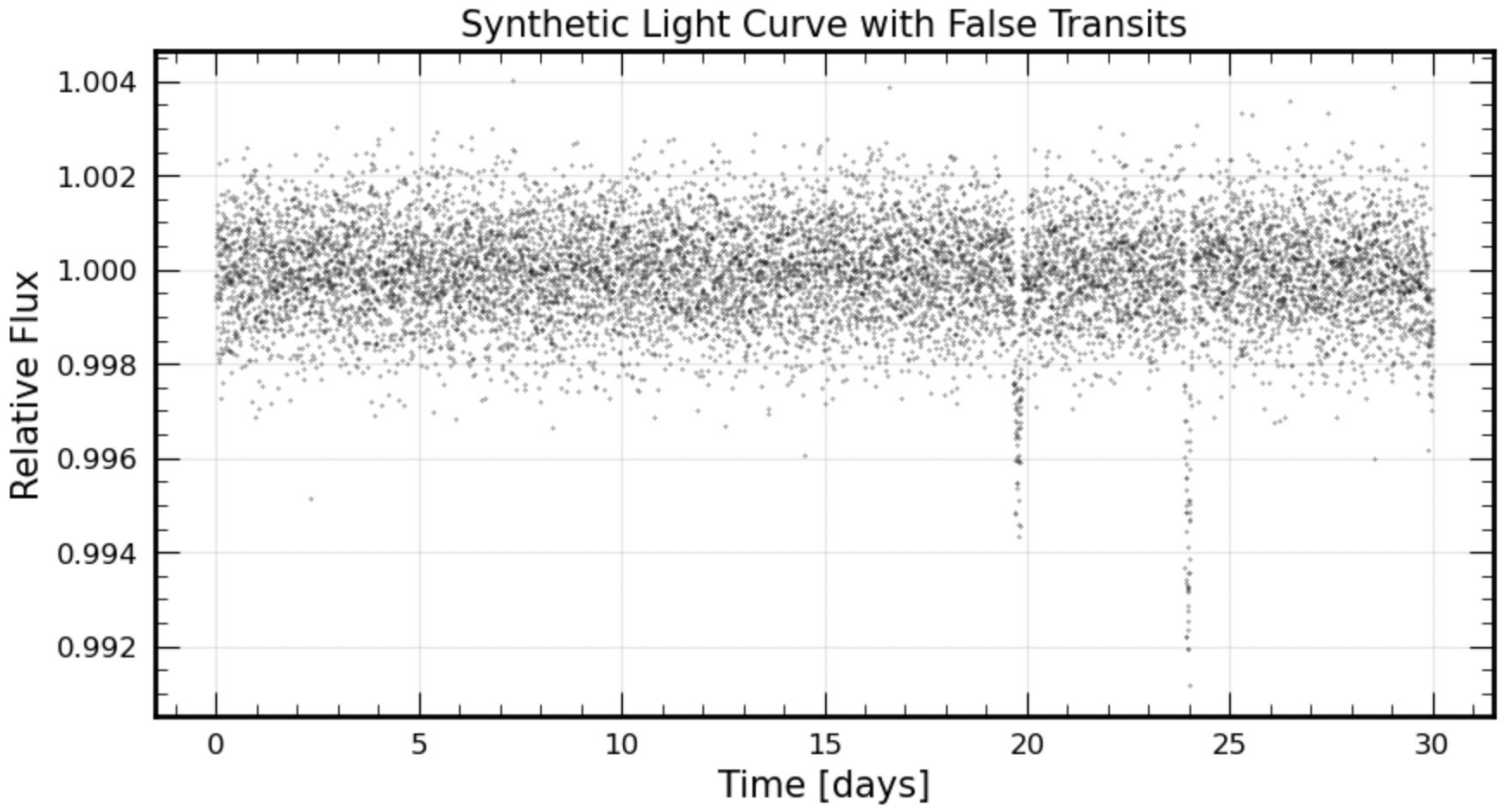
Training Data

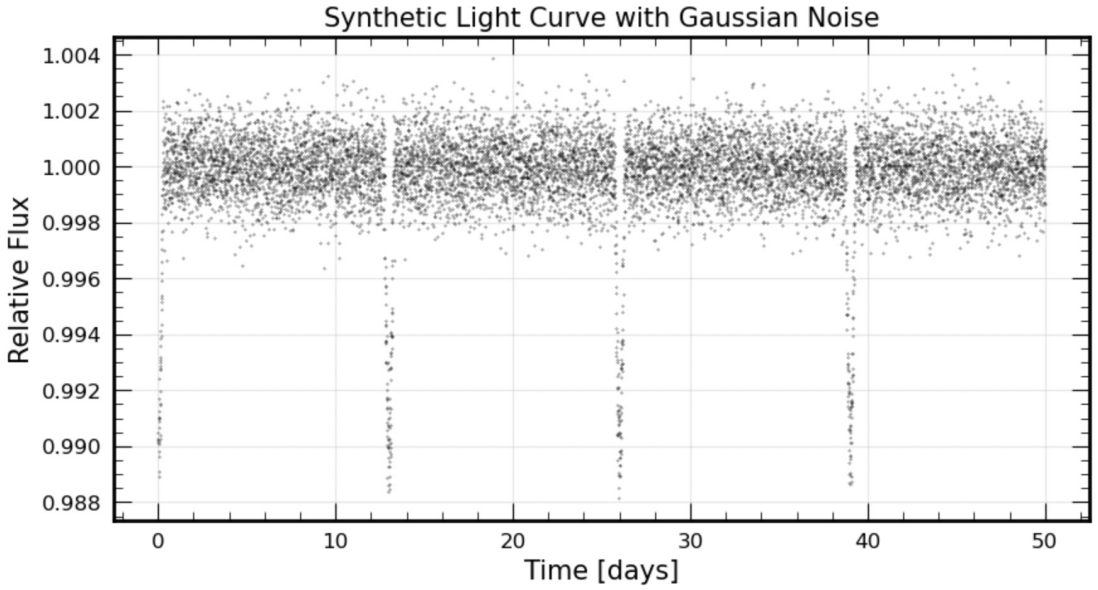


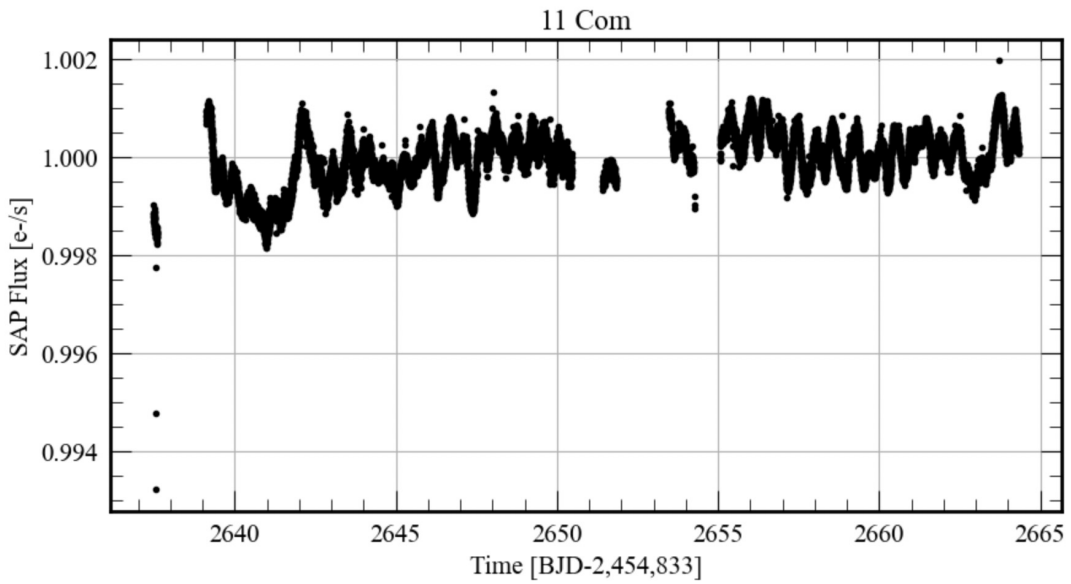
Training Data



Training Data

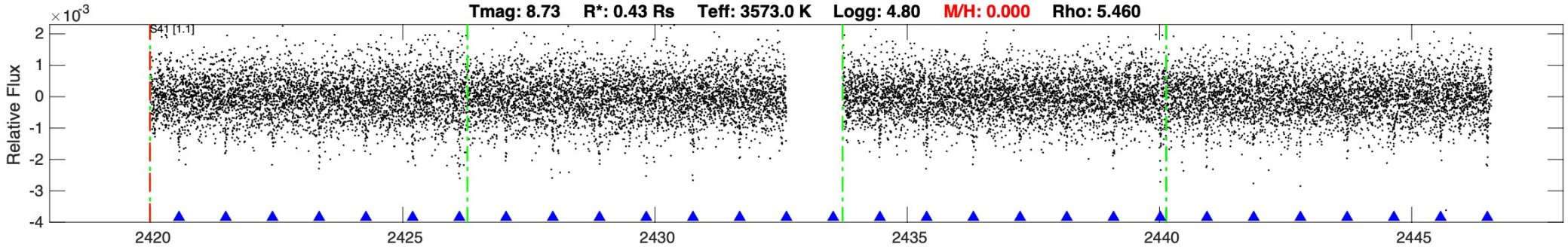






TIC: 239332587 Candidate: 1 of 1 Period: 0.926 d

Tmag: 8.73 R*: 0.43 Rs Teff: 3573.0 K Logg: 4.80 M/H: 0.000 Rho: 5.460



Software Revision: spoc-5.0.42-20210903 -- Cadence Type: TARGET (2-min) -- Date Generated: 09-Sep-2021 16:55:13 Z

This Data Validation Report Summary was produced in the TESS Science Processing Operations Center Pipeline at NASA Ames Research Center

Test Data

Data Loading and Preprocessing

Custom Dataset Class:

- Loads images of varying light-curves types and their labels with PyTorch.
- Images: Grayscale → Resized (128x128) → Tensor format.

Data Splitting

```
from torch.utils.data import random_split
```

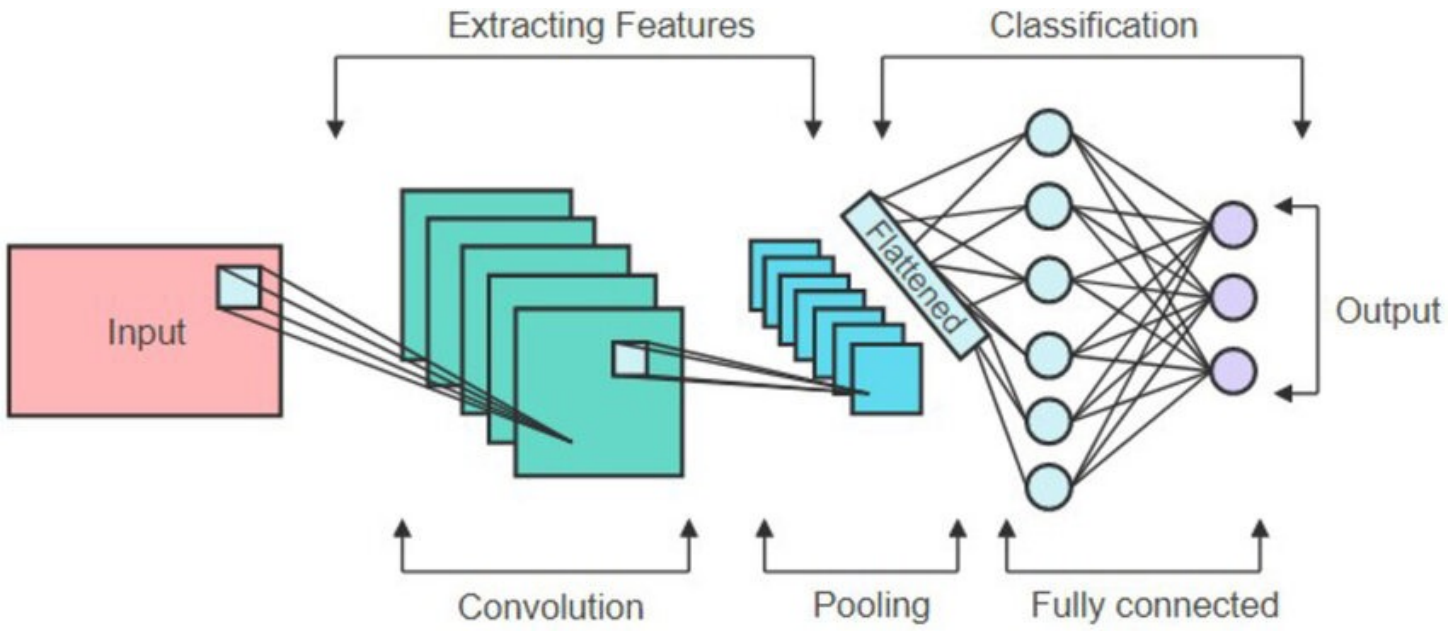
```
# split dataset: 70% train, 15% validation, 15% test
train_size = int(0.7 * len(LC_dataset))
valid_size = int(0.15 * len(LC_dataset))
test_size = len(LC_dataset) - train_size - valid_size

LC_train, LC_valid, LC_test = random_split(LC_dataset, [train_size, valid_size, test_size])

print(f"Training set size: {len(LC_train)}")
print(f"Validation set size: {len(LC_valid)}")
print(f"Test set size: {len(LC_test)}")
```

```
Training set size: 3114
Validation set size: 667
Test set size: 668
```

Data Splitting:

- 70% Training | 15% Validation | 15% Testing

Neural Network Architecture

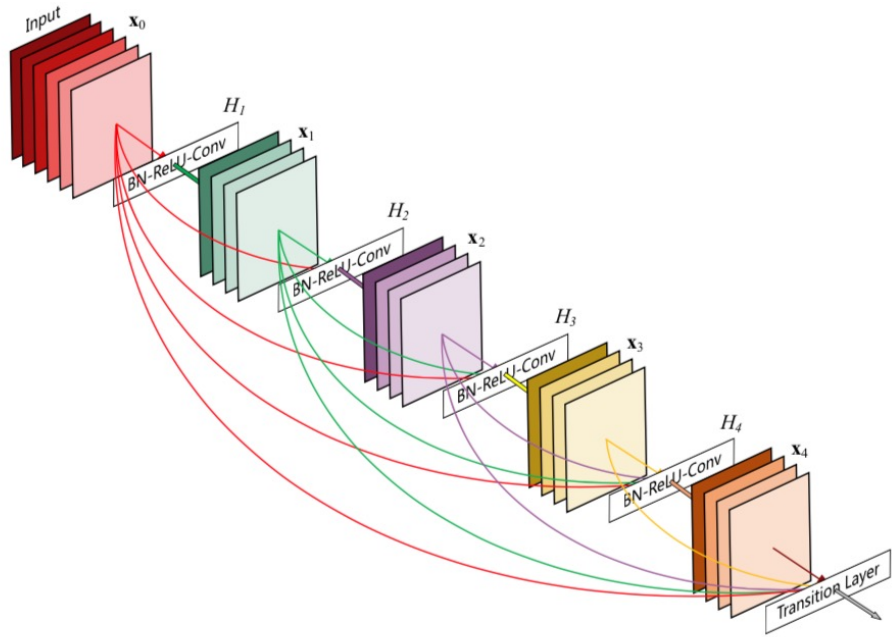


- Our model is a **4-block Convolutional Neural Network (CNN)** designed for binary classification on grayscale images (**1 channel** input).
- SGD is the core optimization algorithm driving the training process.

Convolutional Block Breakdown

Model Structure

[23]:
```
model = nn.Sequential()

# 1st convolutional block                  # 1 for grayscale
model.add_module('conv1', nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3, padding=1))
model.add_module('relu1', nn.ReLU())
model.add_module('pool1', nn.MaxPool2d(kernel_size=2))
model.add_module('dropout1', nn.Dropout(p=0.5)) # Retained dropout with p=0.5 to reduce overfitting

# 2nd convolutional block
model.add_module('conv2', nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1))
model.add_module('relu2', nn.ReLU())
model.add_module('pool2', nn.MaxPool2d(kernel_size=2))
model.add_module('dropout2', nn.Dropout(p=0.5)) # Retained dropout with p=0.5 to reduce overfitting

# 3rd convolutional block
model.add_module('conv3', nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1))
model.add_module('relu3', nn.ReLU())
model.add_module('pool3', nn.MaxPool2d(kernel_size=2))

# 4th convolutional block
model.add_module('conv4', nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3, padding=1))
model.add_module('relu4', nn.ReLU())
model.add_module('pool4', nn.MaxPool2d(kernel_size=8))
model.add_module('flatten', nn.Flatten())

# fully connected layer
# calculate the dimensions dynamically
x = torch.ones((4, 1, 128, 128))  # dummy input to infer dimensions
dims = model(x).shape
model.add_module('fc', nn.Linear(dims[1], 1))
model.add_module('sigmoid', nn.Sigmoid())
```



Conv2D → Activation: ReLU. → MaxPooling → Dropout
Conv2D → Activation: ReLU. → MaxPooling → Dropout
Conv2D → Activation: ReLU. → MaxPooling
Conv2D → Activation: ReLU. → MaxPooling → Flatten

Flatten and Fully Connected Layer

Flatten	Converts the multi-dimensional feature map into a 1D vector for input into the fully connected layer.
Fully Connected Layer	Input Features (calculated dynamically) → 1 Neuron Outputs a single value (logit) for binary classification.
Sigmoid Activation	Ensures the output is between 0 and 1 $\hat{y}_i = \sigma(z) = \frac{1}{1+e^{-z}}$

Loss Function and Optimization

BCE Loss

$$\mathrm{L}_{\mathrm{BCE}} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right]$$

- y_i: True label (0 or 1)
- $\hat{y}_i$: Predicted probability $(\hat{y}_i = \sigma(z))$, where $\sigma(z)$ is the sigmoid function.
- N: Number of samples in a batch.

Stochastic Gradient Descent

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta \mathrm{L}_{\text{mini-batch}}$$

- η: Learning rate.
- Updates parameters using gradients from a **mini-batch** instead of the whole dataset.

Adam Optimizer

$$\theta_t \leftarrow \theta_{t-1} - \eta \cdot \frac{m_t}{\sqrt{v_t} + \epsilon}$$

m_t: Exponentially weighted moving average of the gradient.

v_t: Exponentially weighted moving average of the squared gradient.

η: Learning rate (default: 0.001).

ϵ: Small value to avoid division by zero.

Validation Phase (15%)

- **Validation Loss** (BCELoss)
- **Validation Accuracy** (probability threshold = 0.5).

Testing Phase (15%)

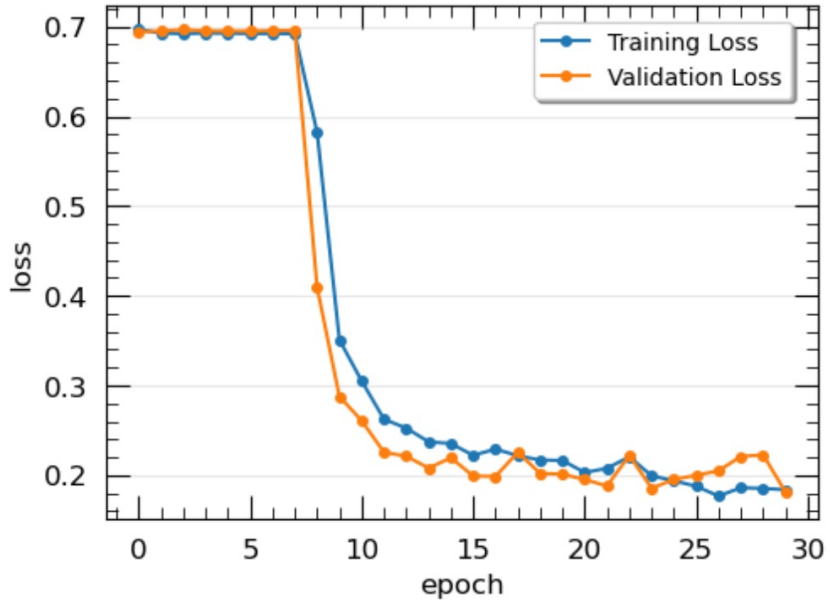
- Final check on new data after training.
- Goal: **90%+ accuracy** for strong real-world results.

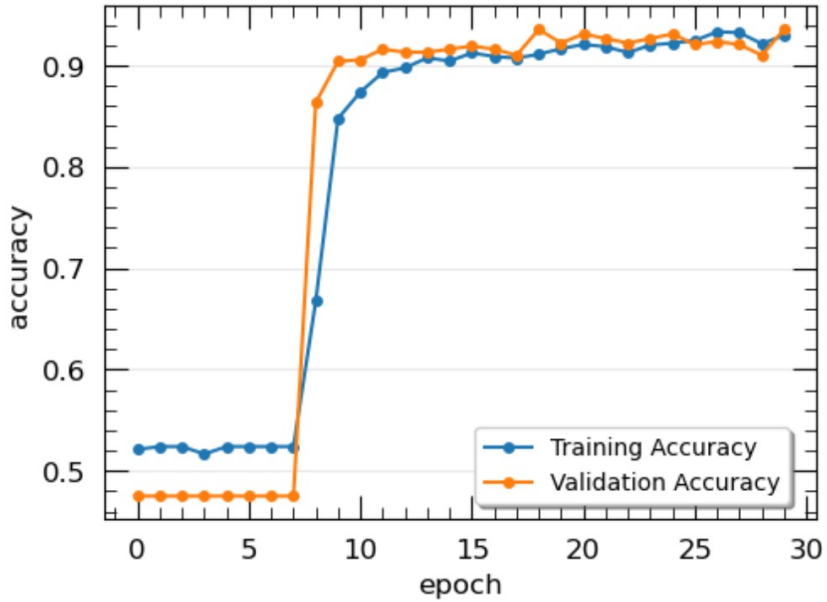
$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Predictions}}$$

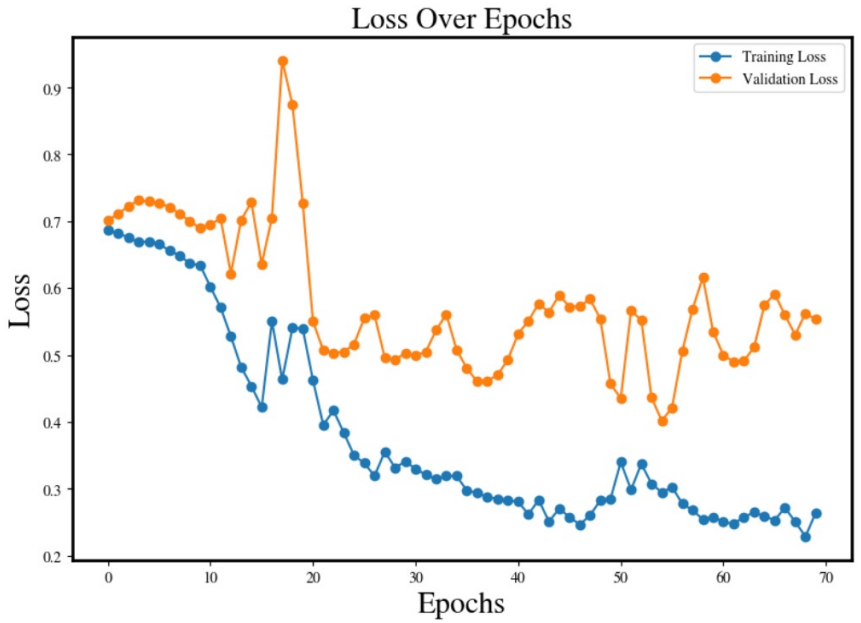
SGD

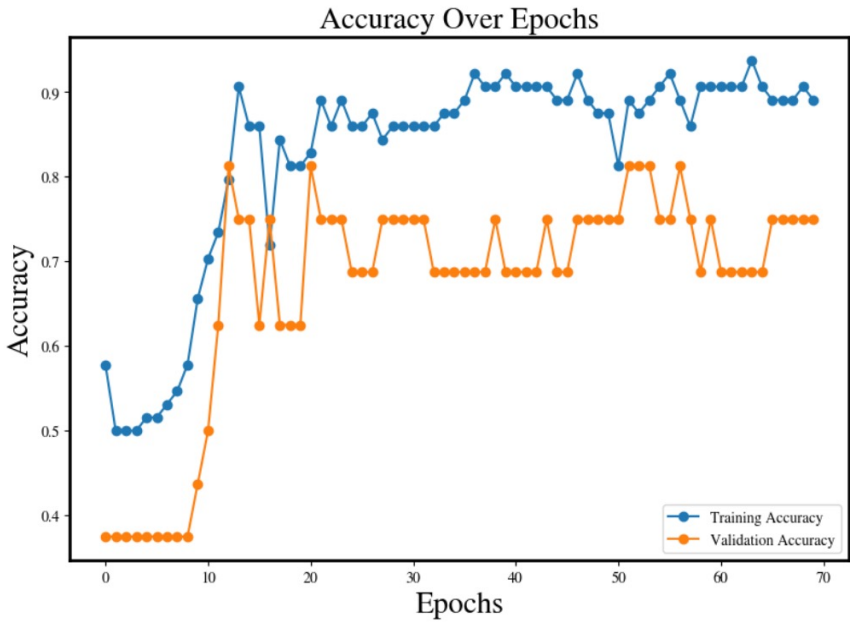
vs.

LSTM

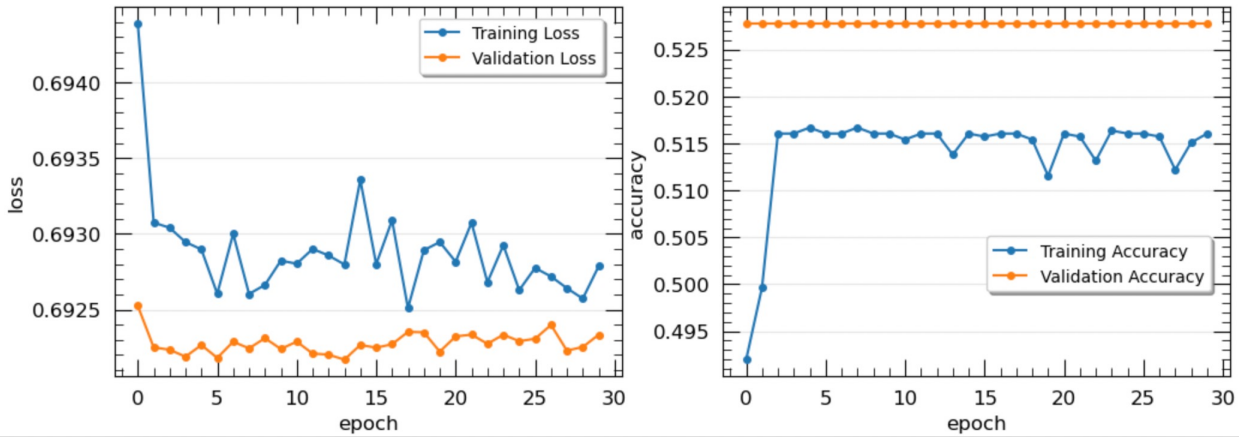

Training Loss
Validation Loss
loss
epoch


Training Accuracy
Validation Accuracy
accuracy
epoch

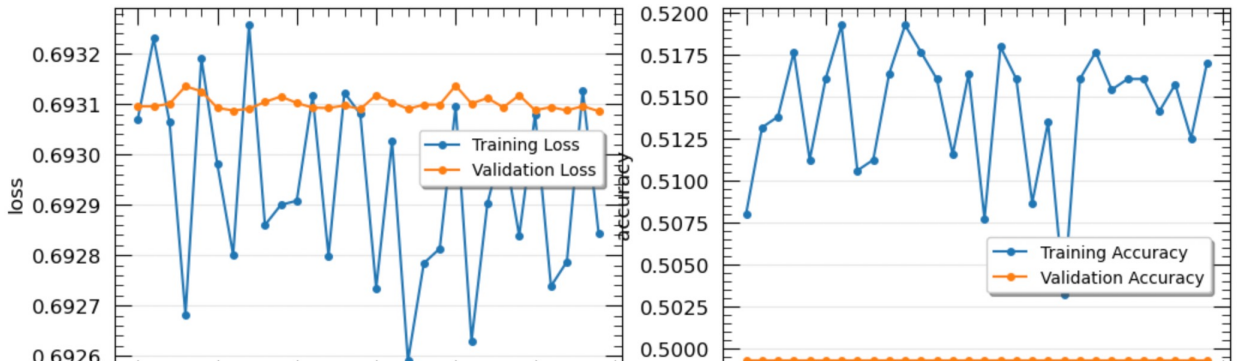

Loss Over Epochs
Training Loss
Validation Loss
Loss
Epochs


Accuracy Over Epochs
Training Accuracy
Validation Accuracy
Accuracy
Epochs

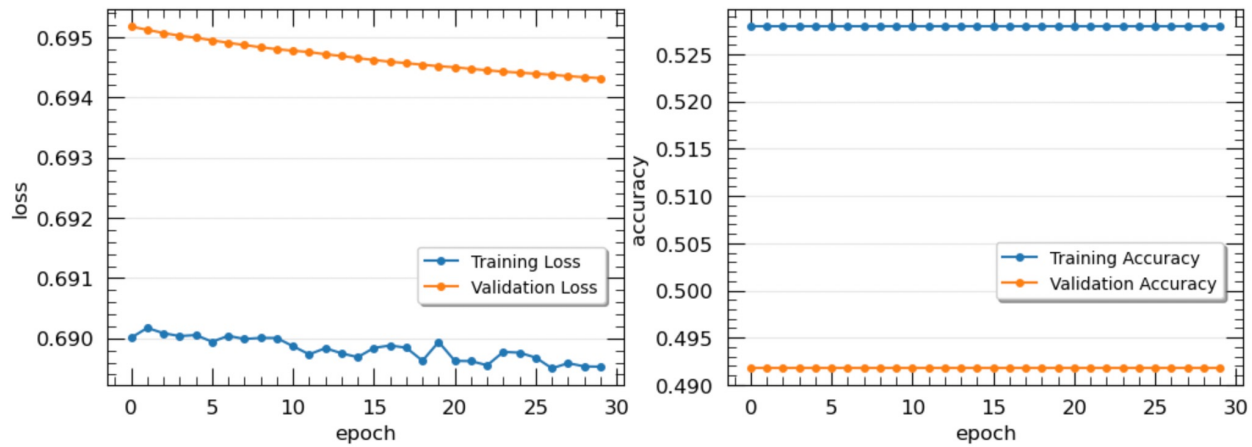
learning rate = 0.001 momentum = 0.1



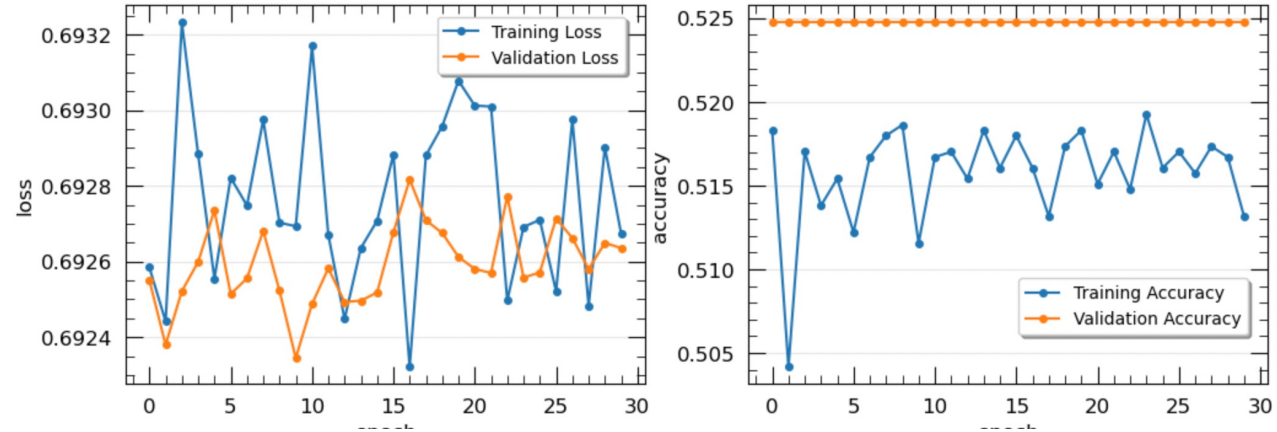
learning rate = 0.001 momentum = 0.2



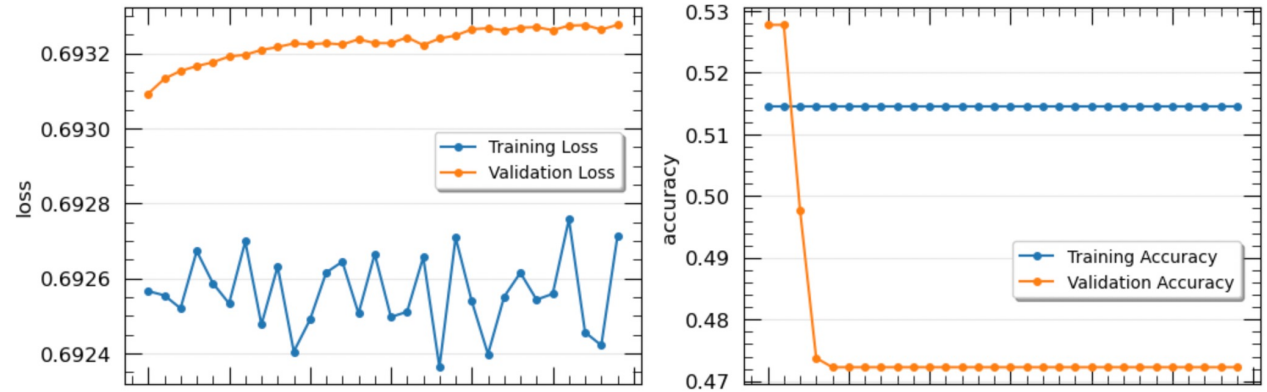
learning rate = 0.001 momentum = 0.3



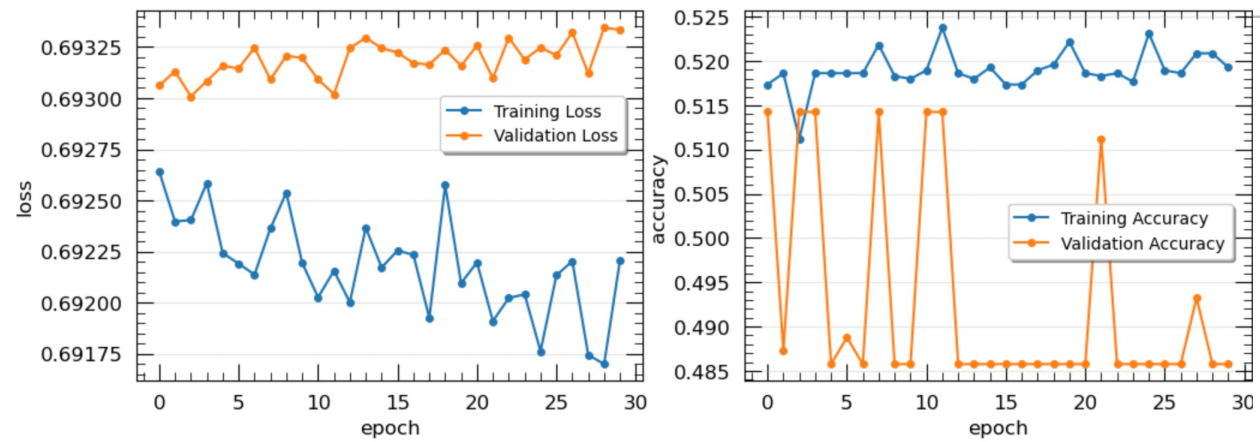
learning rate = 0.001 momentum = 0.4



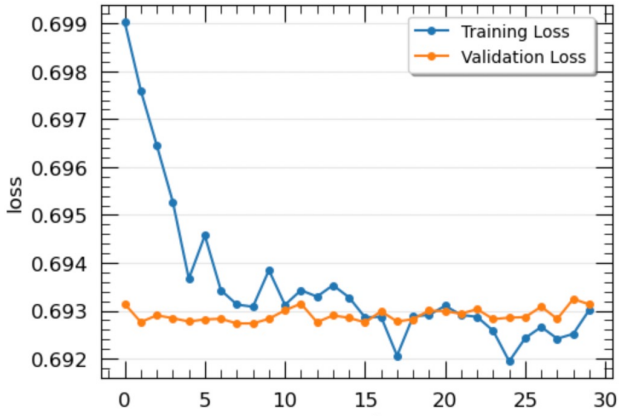
learning rate = 0.001 momentum = 0.5

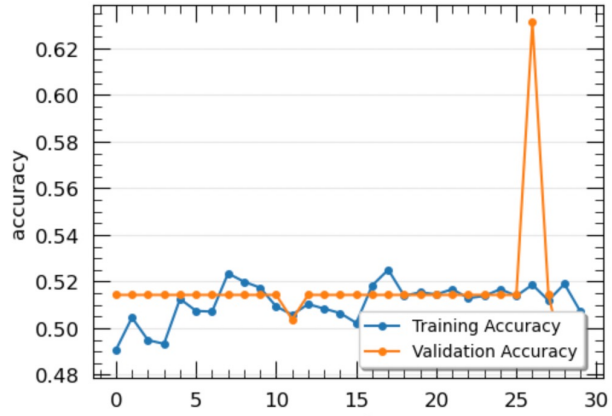


learning rate = 0.001 momentum = 0.9

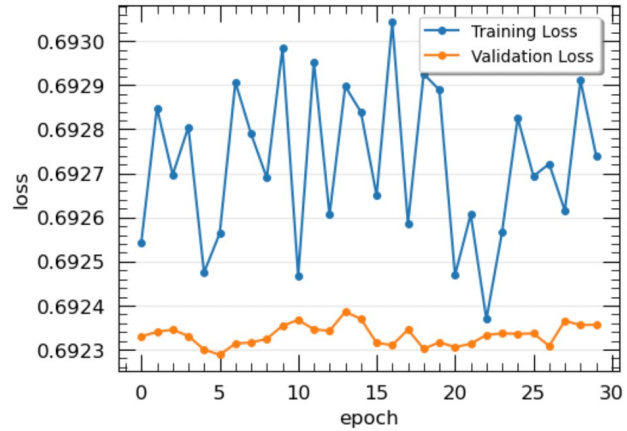


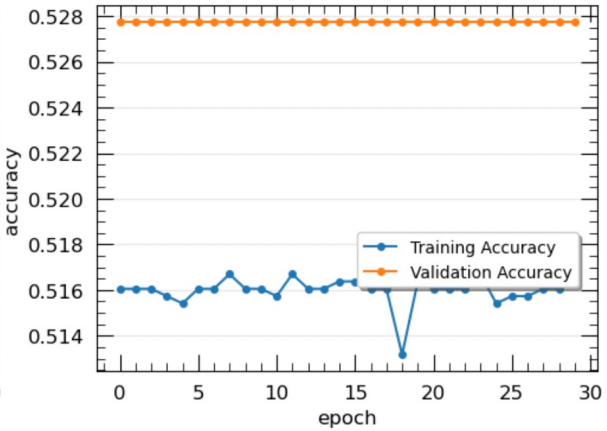
learning rate = 0.01 momentum = 0.3



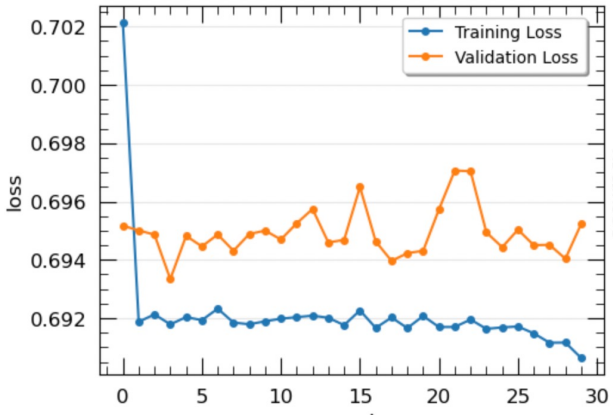


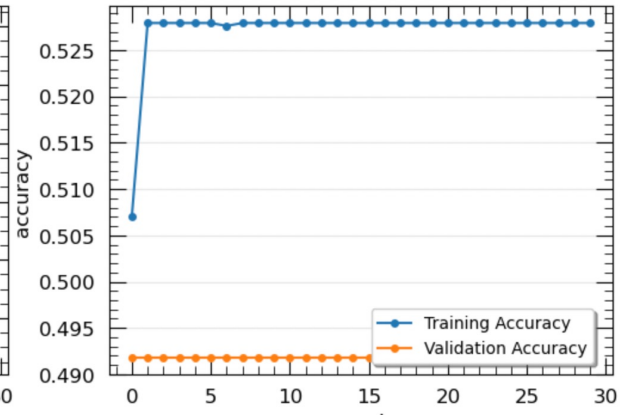
learning rate = 0.0005 momentum = 0.1



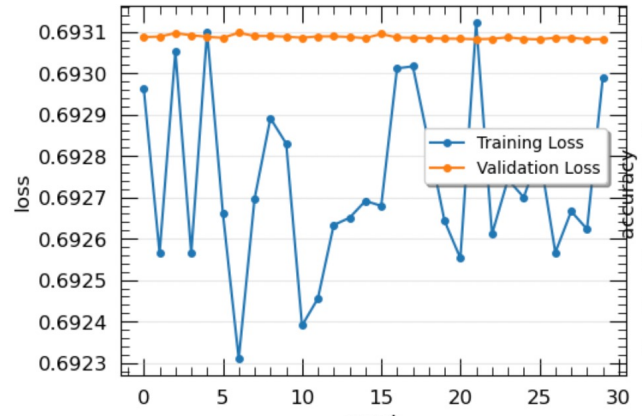


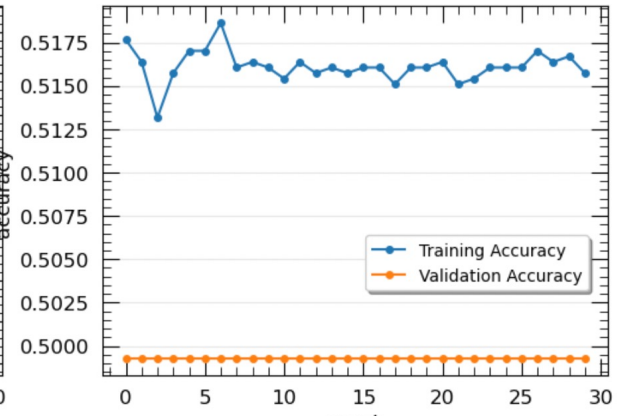
learning rate = 0.01 momentum = 0.9



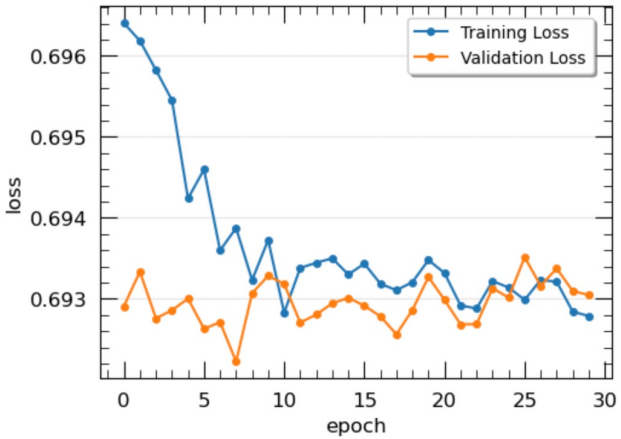


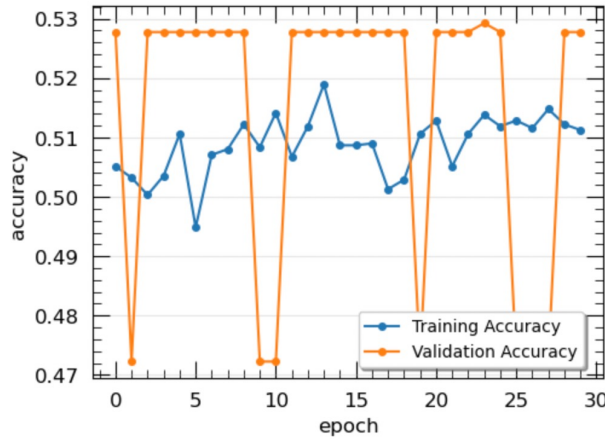
learning rate = 0.0005 momentum = 0.2



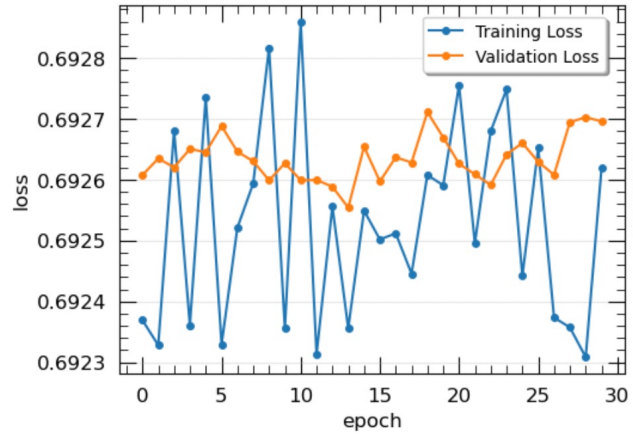


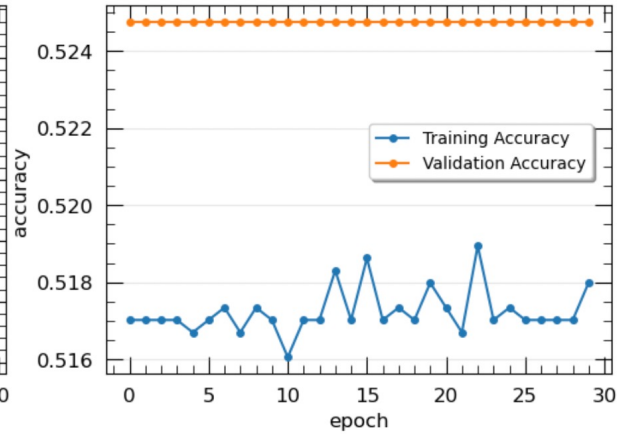
learning rate = 0.01 momentum = 0.5





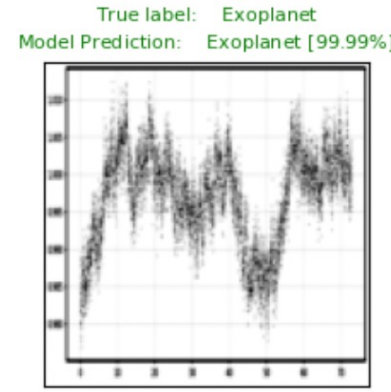
learning rate = 0.0005 momentum = 0.3

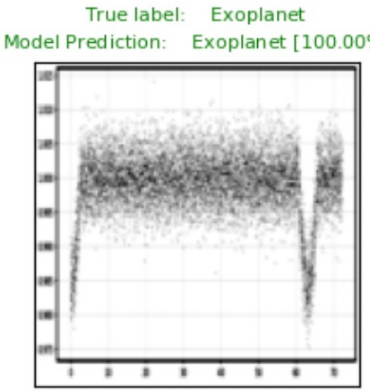


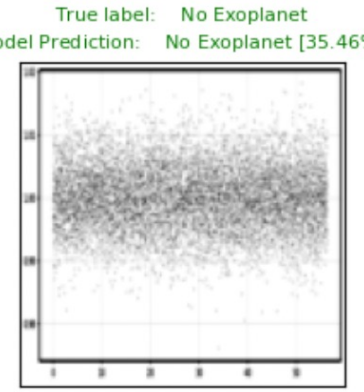


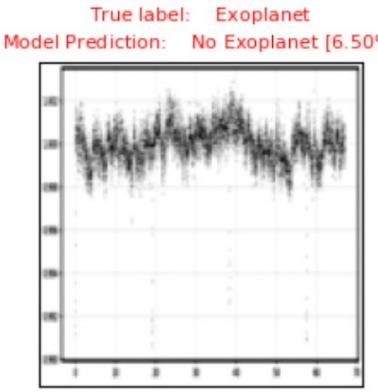
Test Results: 1st batch

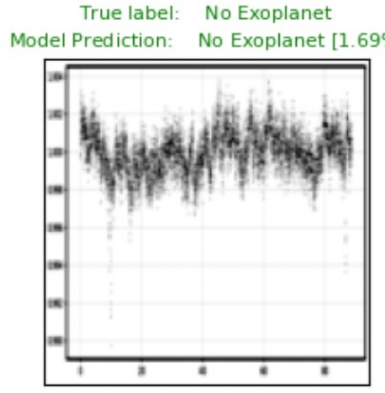
Accuracy: 93.4%

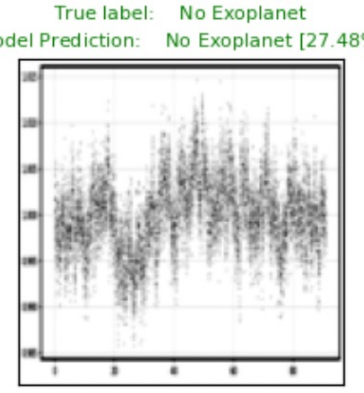


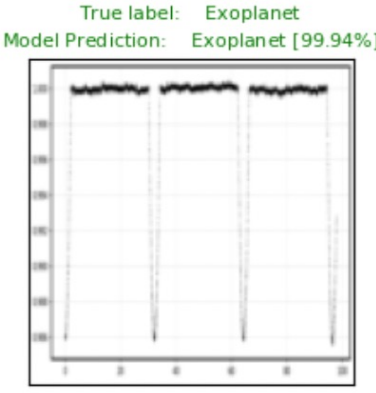


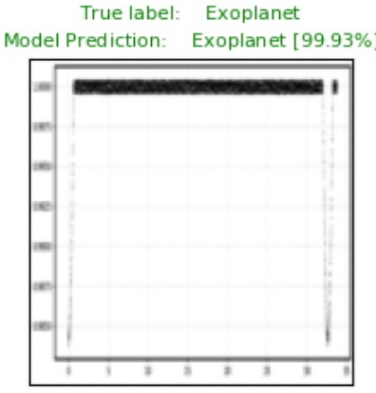


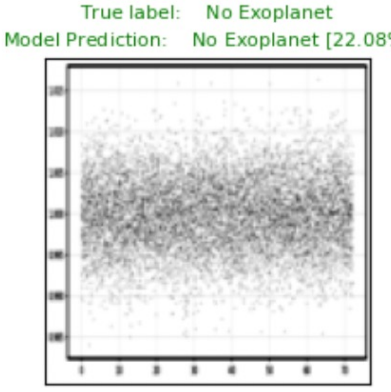


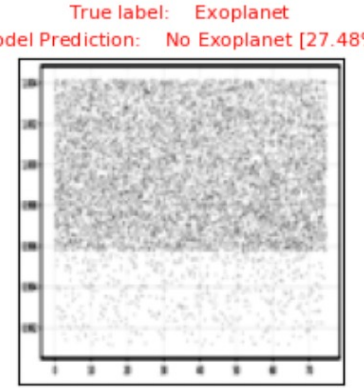


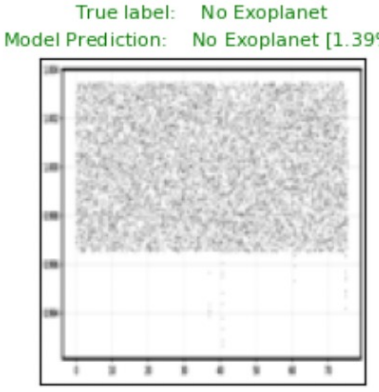


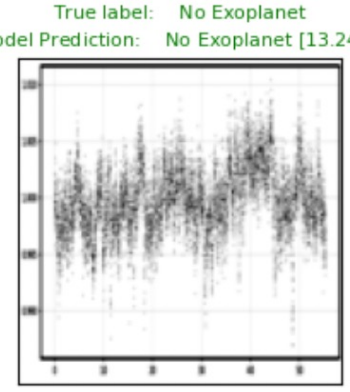






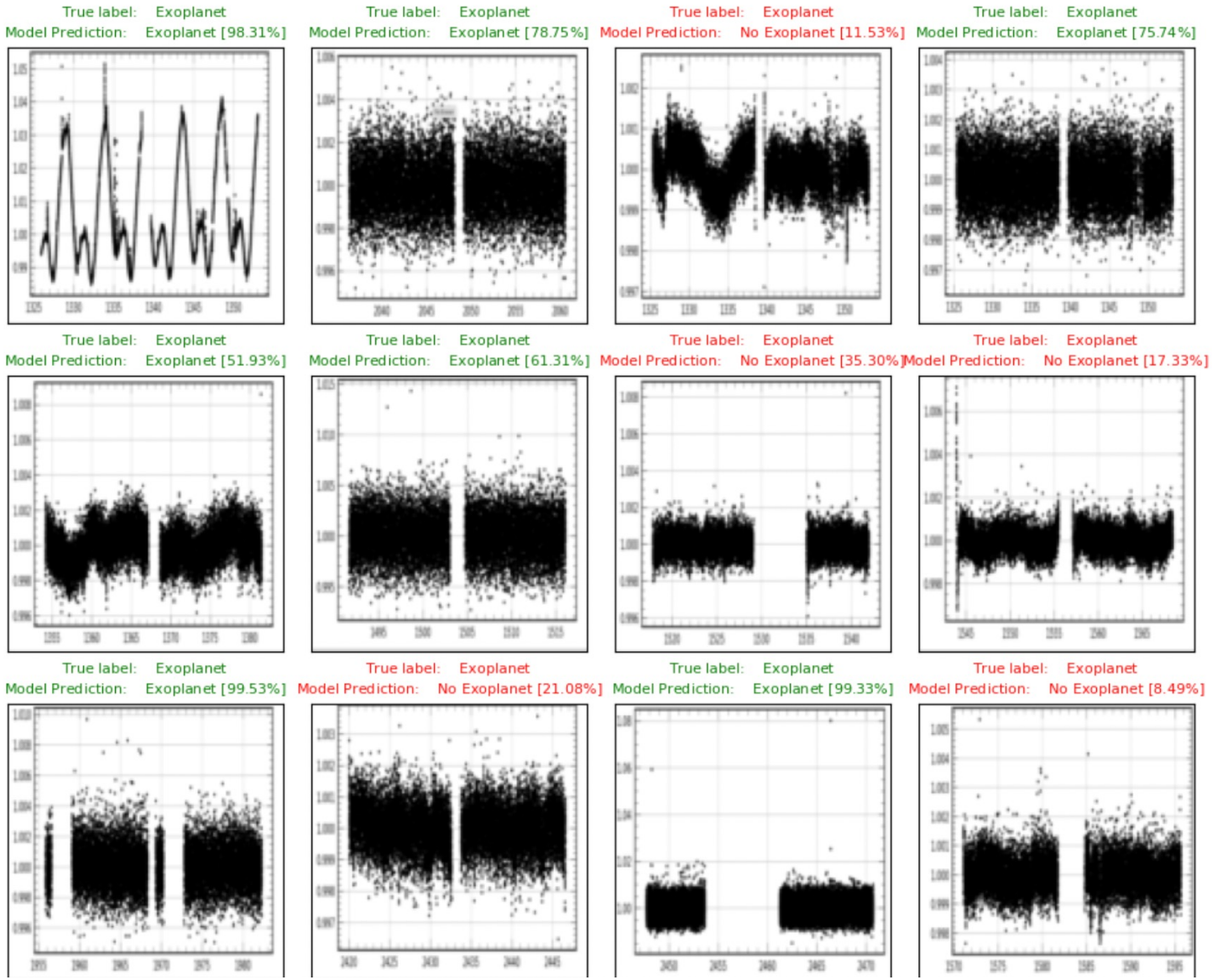






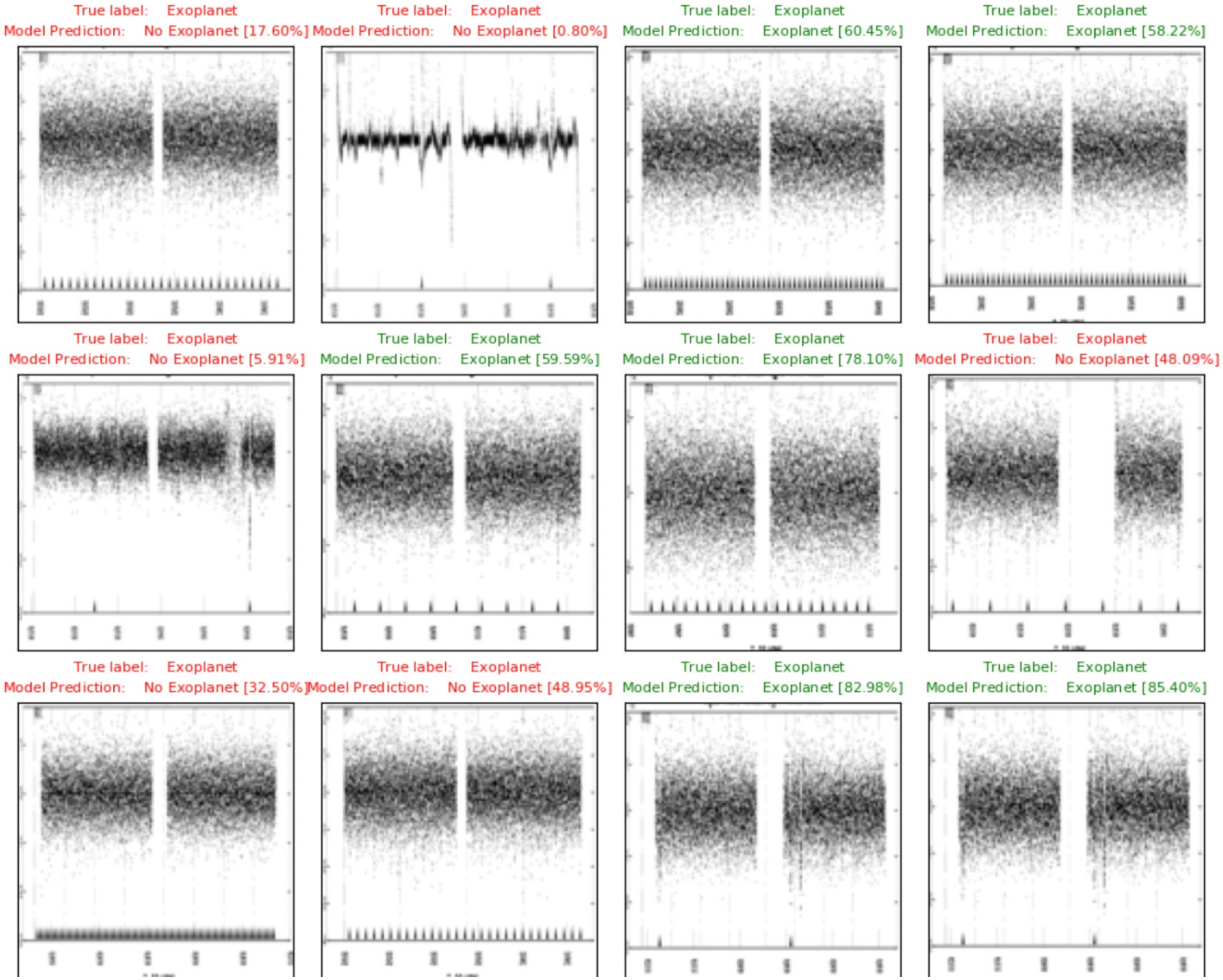
Test Results: 2nd batch

Accuracy: 58.3%



Test Results: 3rd batch

Accuracy: 50.0%



Limitations

- model learns synthetic data patterns too well, struggles with real observation data
- approach assumes exoplanets are detectable via transits
- real data are not pretty
- data extraction is time-consuming

Benefits

- can be used as a first-pass for exoplanet detection
- sophisticated models save time and resources
- revolutionize exoplanet detection