TEAM 9 : FINAL

버찌 OR NOT

버찌 202000000
마히라 2023320033
인탄 2023320129
응아 2022320166

Problem Identification

Many tourists visit Korea University hoping to see the famous campus cat, Beojji.
But two main problems often happen:

1. **“Is this Beojji?”** 🐱
2. **“Is Beojji likely to be at home now?”** 🏠

Techniques: Supervised Learning

1. Binary Image Classification

CNN to extract visual features of Beojji

+

K-Nearest Neighbors (KNN) classifier for prediction.

2. Tabular Classification

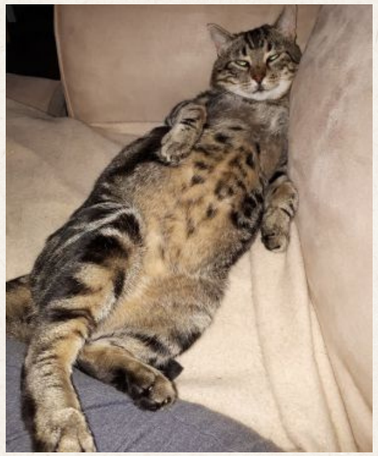
Random Forest to predict Beojji's behaviour pattern

DATASET PREPARATION

Dataset 1: Beojji or not

Tabby: 80 images

similar features with Beojji – such as body shape, stripe pattern, and overall fatness.



Not_Tabby: 80 images

any image that is not Tabby cat

Dataset 2: Beojji is home or not

manually constructed 72 rows in csv format

	A	B	C	D	E
1	season	time_of_day	is_raining	noise_level	home_or_not
2	winter	morning	0	low	1
3	winter	morning	0	medium	1
4	winter	morning	0	high	1
5	winter	morning	1	low	1
6	winter	morning	1	medium	1
7	winter	morning	1	high	1
8	winter	evening	0	low	1
9	winter	evening	0	medium	1

winter
summer
spring
autumn

morning
evening
night

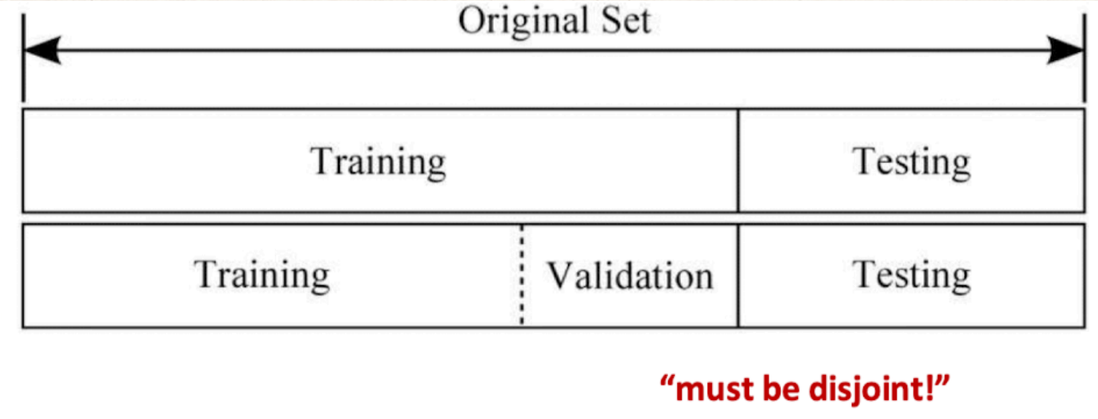
Yes -1
No -0

low
medium
high

Yes -1
No -0

- **Data Preprocessing**:
  - Resize to 224×224
  - Normalize using ***ToTensor()***

- **Split**: 80% train, 10% validation, 10% test



Our custom CNN with a pre-trained ResNet18 backbone

- **Purpose:** To utilize a pretrained Convolutional Neural Network (CNN), specifically ResNet18, as a feature extractor for input cat images.
- **Approach:**
  - Load ResNet18 with pretrained weights from ImageNet.
  - The final fully connected (classification) layer is removed.
  - The model is used only for feature extraction, with ***torch.no_grad()*** and ***eval()*** to prevent training.

Our custom CNN with a pre-trained ResNet18 backbone

1 - Training Configuration

- Loss Function: ***nn.CrossEntropyLoss***
  => measure the error between the model's prediction and the actual label.
- Optimizer: ***optim.Adam***
  (Learning Rate = 0.001) => efficiently updates model weights to minimize the loss.
- LR Scheduler: ***StepLR***
  => reduce learning rate by 90% (gamma=0.1) every 5 epochs to help the model fine-tune more precisely over time.

```
#training CNN
model=CNN(num_classes=2).to(device)

criterion=nn.CrossEntropyLoss()
optimizer=optim.Adam(model.parameters(),lr=0.001)
scheduler=StepLR(optimizer,step_size=5,gamma=0.1)
```

Our custom CNN with a pre-trained ResNet18 backbone

2 - Training Process (10 Epochs)

- Training Phase (***model.train()***)
  - The model learns patterns from the ***train_loader***.
  - Weights are updated via backpropagation after calculating loss.
- Validation Phase (***model.eval()***)
  - The model's performance is tested on the ***valid_loader*** (data it has not seen before).
  - We calculate Validation Accuracy (***val_acc***) to check for generalization.

```
for images,labels in train_loader:
  #train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True)
    images,labels =images.to(device), labels.to(device)

    # backprop compute output-> compute loss-> backprop-> update weight
    optimizer.zero_grad()
    outputs =model(images)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()

    total_loss += loss.item() #accumulate loss

    #to get predicted class index for each sample in bacth
    # find mac value and index
    _, predicted = torch.max(outputs, 1)
    total+=labels.size(0)

    correct+=(predicted==labels).nonzero().size(0)
# calculate avg loss and training acc
avg_loss = total_loss/len(train_loader)

train_acc = correct/total
train_losses.append(avg_loss)
scheduler.step()

#track how lr reduced
for param_group in optimizer.param_groups:
    current_lr = param_group['lr']
print(f"Learning Rate: {current_lr}")

model.eval() #use eval
correct=0
total= 0
total_valloss=0.0
```

Our custom CNN with a pre-trained ResNet18 backbone

```python
# save the best model based on validation accuracy
if val_acc>best_val_acc:
    best_val_acc=val_acc
    torch.save(model.state_dict(), "best_model.pth")
    print(f"Saved new best model with Val Acc: {val_acc:.2f}")

print(f"Epoch {epoch+1}/{num_epochs} | Loss: {avg_loss:.4f} | Train Acc: {train_acc:.2f} | Val Acc: {val_acc:.2f}")
```

3. Saving the Best Model

- Goal: To prevent overfitting and select the model that performs best on new, unseen data.
- Method:
  - We monitor ***val_acc*** after every epoch.
  - If the current ***val_acc*** is higher than any previous one, we save the model's weights to ***best_model.pth***.
  - This ensures our final model is the one that generalizes the best.

LOSS FUNCTION

Cross Entropy Loss()

- for classification (binary or multi class)
- combine: LogSoftmax and Negative Log Likelihood Loss (NLLLoss)
- to predict right class with high confidence

$$\text{Loss} = -\sum_{i=1}^{C} y_i \cdot \log(\hat{y}_i)$$

loss near 0-> confident and right
loss is high-> confident and wrong

1.
```
self.classifier=nn.Linear(512, num_classes)
```

- Model Structure: final fully connected layer
- converts the 512-dim CNN feature vector into raw class scores (**logits**)

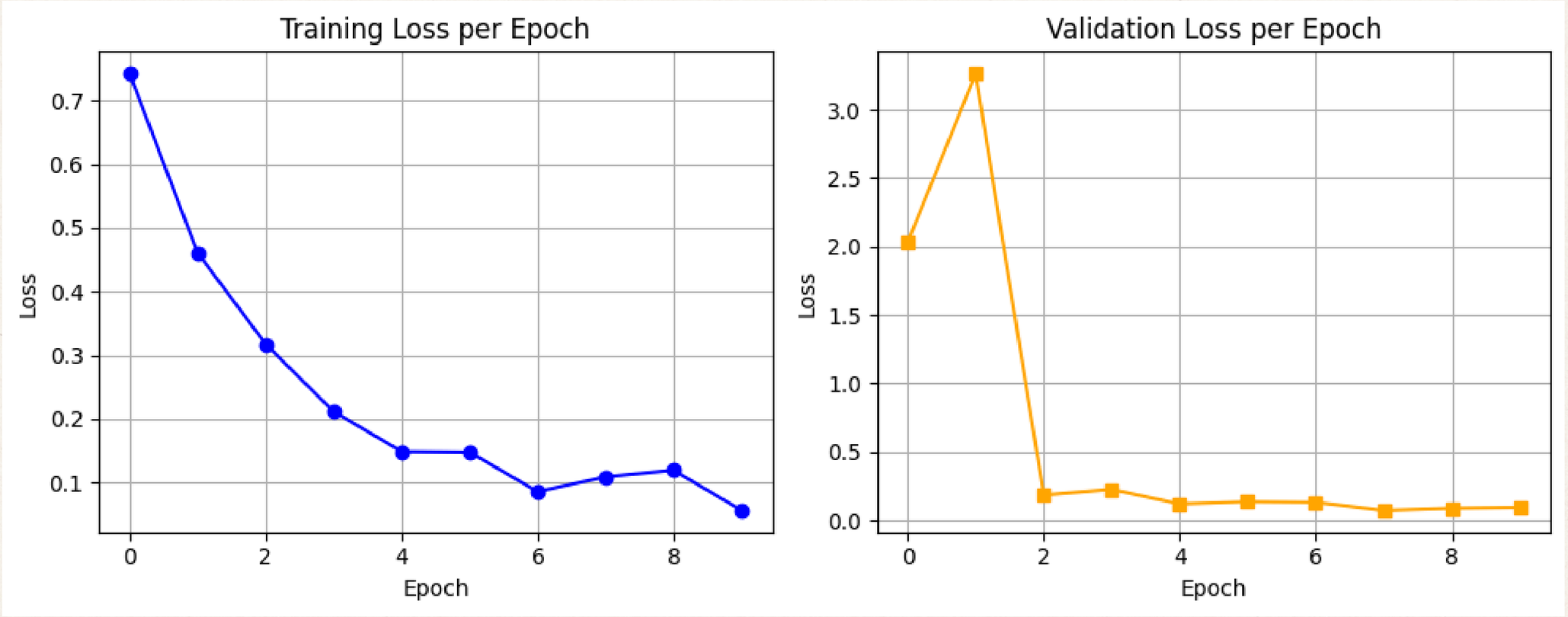
2.
```
criterion=nn.CrossEntropyLoss()
loss = criterion(outputs, labels)
loss.backward()
```

- takes raw scores from self.classifier()
- applies softmax: **convert logits to probabilities**
- **compares** with true labels
- **backpropagates** error: adjust model weights to increase confidence in the correct class

```
Saved new best model with Val Acc: 1.00
Epoch 5/10 | Loss: 0.1488 | Train Acc: 0.95 | Val Acc: 1.00
```

```
Saved new best model with Train Loss: 0.0564
Learning Rate: 1e-05
Epoch 10/10 | Loss: 0.0564 | Train Acc: 0.99 | Val Acc: 1.00
```

Training vs Validation Loss



model is learning well and fitting the training data

model generalize well on (new,unseen) validation data

KNN: IMAGE CLASSIFICATION

- **Purpose:** To train a K-Nearest Neighbors (KNN) classifier to distinguish between Tabby and other cats based on the extracted features.
- **Implementation:**
  - Train KNN models with k values of 3, 5, and 7.
  - Accuracy is measured on the validation set to determine the best k.
  - The best-performing KNN model is retrained on the full training set.

Importance of a Reliable Dataset

1. Initial dataset2 (csv)

- randomly constructed dataset with no clear logic
- Result: Model struggled to learn patterns
- Accuracy: **low → ~0.60**

2. Improved dataset2(csv)

- constructed with clear conditions:
  - Winter → at home
  - Morning → at home
  - Spring/Autumn Evening → not at home
  - Summer + Rain + Medium/High → not at home
- Accuracy: **higher → up to 0.92 !**

why?

- Random Forest now can **better** detect **decision thresholds** when data is structured
- Even simple models perform well with reliable dataset

EVALUATION METRICS

Confusion Matrix

Class 0 (e.g., Not Tabby):

- True Negatives (TN): 7
- False Positives (FP): 0

Class 1 (e.g., Tabby):

- True Positives (TP) : 11
- False Negatives (FN): 0

✅ Perfect Classification:

- No misclassifications.
- Accuracy = 100%.
- Precision, recall, and F1-score = 1.00 for both classes.

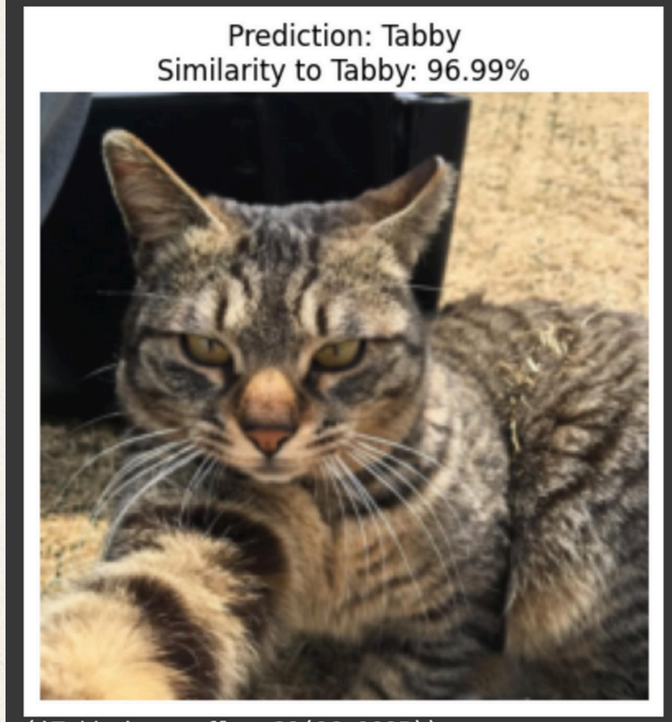
```
Accuracy:  1.0
Confusion Matrix:
 [[ 5  0]
 [ 0 11]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         5
           1       1.00      1.00      1.00        11

    accuracy                           1.00        16
   macro avg       1.00      1.00      1.00        16
weighted avg       1.00      1.00      1.00        16
```

RESULT ANALYSIS

```
predict_image("IMG_1.jpg", feature_extractor, knn, device, dataset.classes, tabby_mean)
```



TABBY !!!

```
('Tabby', np.float32(96.9885))
```

```
predict_image("IMG_2.jpg", feature_extractor, knn, device, dataset.classes, tabby_mean)
```

NOPE

```
('Not_Tabby', np.float32(29.6101))
```

- **Model Limitation:** Analysis of a Misclassification
- **Input:** An image of a rabbit
- **Prediction:** Tabby (Similarity: 96.94%)

Why did this happen?

- The model learns **visual patterns**, not concepts. Our CNN does not understand "cat" or "rabbit." It learns to associate specific textures, colors, and patterns with a label.
- **Dominant Feature:** The model identified the rabbit's brownish, mottled fur as the most critical feature. This pattern is visually very similar to the striped pattern of our Tabby cat.

Random Forest

- Our goal is to predict wether Tabby is likely to be at home or not based on season, time of day, last seen time, rainy day and noise level
- Transformed using one-hot encoding and passed to the model
- Each tree gives a vote: home(1) or not home(0)
- Forest picks the majority vote as final prediction
- Accuracy : ~93%
- General behavioral trends : spring & autumn evening Tabby is not at home, rainy summer with medium to high noise level, Tabby is at home

```
Season (spring/summer/autumn/winter): summer
Time of day (morning/evening/night): night
Is it raining? (yes=1, no=0): 1
Noise level (low/medium/high)): high
Tabby is probably out and about.
```

Visualization: PCA

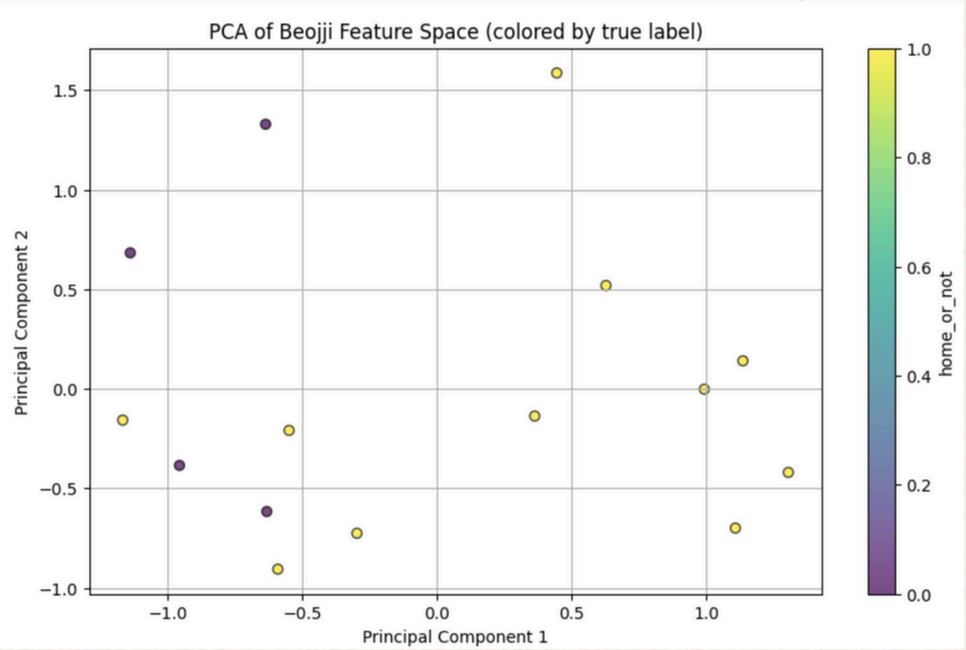
- Each point represents a single test sample (day/time/weather condition)
- Color:-

  Yellow : home_or_not = 1 (Tabby was at home)

  Dark purple : home_or_not = 0 (Tabby was outside)

Observation:

- There is some overlap between the two labels, especially in the center.
- Despite that, the model achieved 93% accuracy, meaning it can still separate the classes well.
- This suggests the original features have useful patterns that the model can learn, even if they're not obvious in 2D.



THANK YOU

CHECKPOINT PRESENTATION

Problem statement

Detecting 버찌 or not using Image classification

버찌 = Tabby (due to limited dataset of real 버찌)

Learning Algorithm

CNN as feature extractor

- Use models like MobileNetV2, ResNet50
- Image input → output feature vector

KNN as image classification using the feature extracted by CNN

- A set of features from labeled images: “Tabby” or “not Tabby”
- For a new input image: extract features → compare with the known feature set → use KNN (or cosine similarity) for classification

Next Step

- **Training Loss:** Using Gradient Descent technique e.g. ADAM optimizer + Categorical Crossentropy loss function
- **Evaluation Metrics:** Confusion Matrix → Accuracy + Similarity Threshold Analysis
- **Visualization:** PCA of feature space to show clusters (버찌 vs. non-버찌)