19CSE456 Neural Network and Deep Learning Laboratory

List of Experiments

Week #	Experiment Title
1	Introduction to the lab and Implementation of a simple Perceptron (Hardcoding)
2	Implementation of Perceptron for Logic Gates (Hardcoding, Sklearn, TF)
3	Implementation of Multilayer Perceptron for XOR Gate and other classification problems with ML toy datasets (Hardcoding & TF)
4	Implementation of MLP for Image Classification with MNIST dataset (Hardcoding & TF)
5	Activation Functions, Loss Functions, Optimizers (Hardcoding & TF)
6	Lab Evaluation 1 (based on topics covered from w1 to w5)
7	Convolution Neural Networks for Toy Datasets (MNIST & CIFAR)
8	Convolution Neural Networks for Image Classification (Oxford Pets, Tiny ImageNet, etc.)
9	Recurrent Neural Networks for Sentiment Analysis with IMDB Movie Reviews
10	Long Short Term Memory for Stock Prices (Yahoo Finance API)

List of Experiments contd.

Week #	Experiment Title
11	Implementation of Autoencoders and Denoising Autoencoders (MNIST/CIFAR)
12	Boltzmann Machines (MNIST/CIFAR)
13	Restricted Boltzmann Machines (MNIST/CIFAR)
14	Hopfield Neural Networks (MNIST/CIFAR)
15	Lab Evaluation 2 (based on CNN, RNN, LSTM, and AEs)
16	Case Study Review (Phase 1)
17	Case Study Review (Phase 1)

Course Project

Rules for Student Teams

- Each team can have a maximum of 4 members
- Teams must submit a project proposal in a PPT file by the end of the next week i.e., before 21st Feb 2025, outlining their project idea, objectives, dataset, and a timeline
- Each team will work with one of the following Neural Network types:

Multilayer Perceptron

Boltzmann or Restricted Boltzmann Machines

Autoencoder

Convolutional Neural Network

Recurrent Neural Network

Long Short Term Memory

Course Project

Rules for Student Teams (continued)

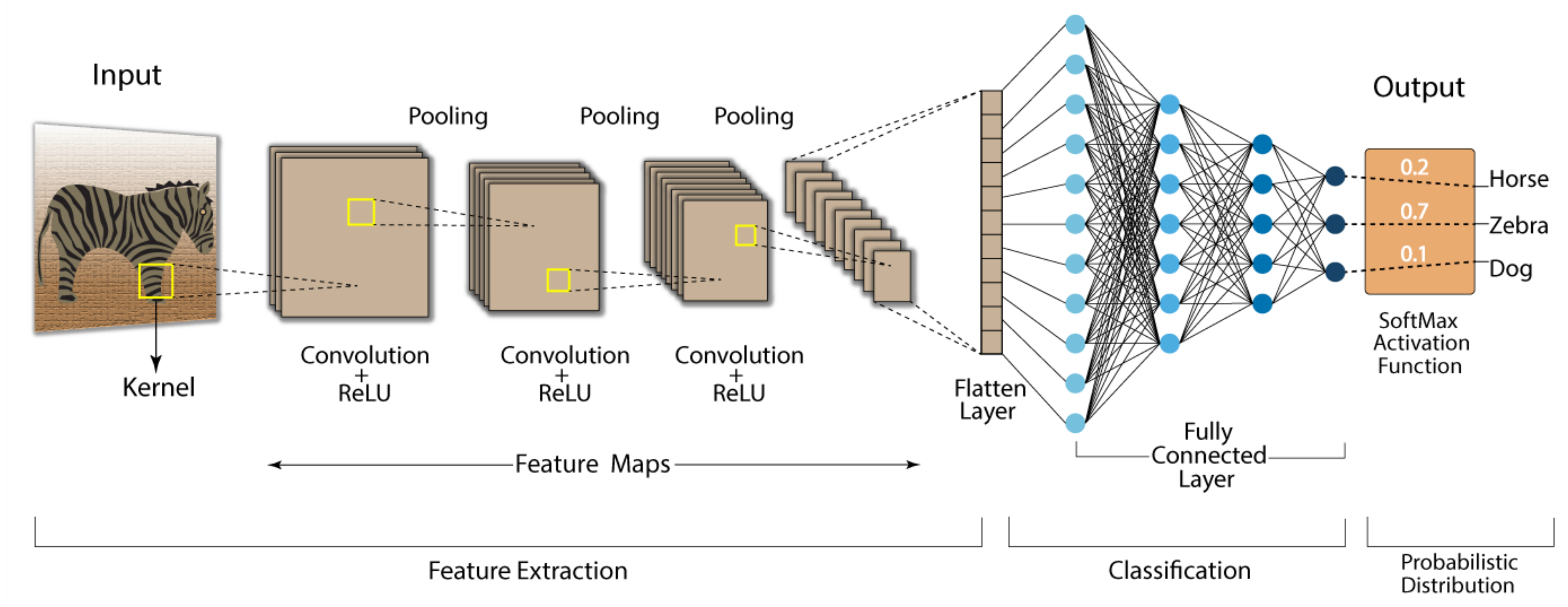
- Teams are encouraged to collaborate and share ideas, but each team must develop its own project
- Teams must present their project in the last week (1st to 4th April 2025) of the course, including a live demonstration of their application
- Teams must submit a final report documenting their project, including problem statement, methodology, results, and conclusions
- Code must be well-documented and submitted along with the report to a GitHub repository to which all teams will be added as collaborators
- Teams are encouraged to deploy their models, and build aesthetically pleasing and interactive GUIs using Streamlit framework
- A neat and clearly-defined presentation, covering every pivotal aspect of the project, will be expected from each team during the review

Course Project Rubrics

Sl. No.	Criterion	Marks
1	Project Proposal	2
2	Data Handling	2
3	Model Development	5
4	Model Performance	5
5	Deployment & GUI	3
6	Presentation	3
Total Marks		20

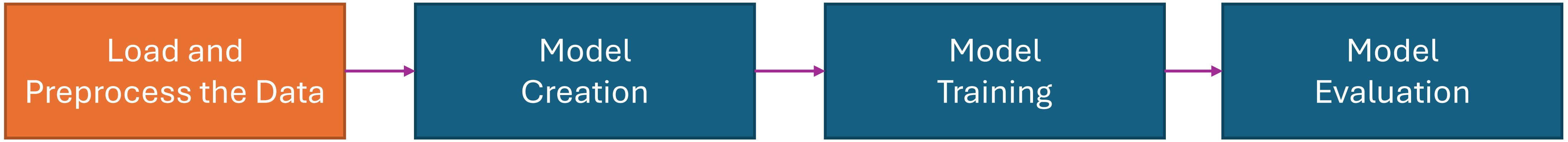
CNN using TensorFlow

Implementing Convolutional Neural Networks (CNNs) using TensorFlow has several advantages, making it a popular choice among developers and researchers



Load and Preprocess the Data → Model Creation → Model Training → Model Evaluation

CNN for Image Classification



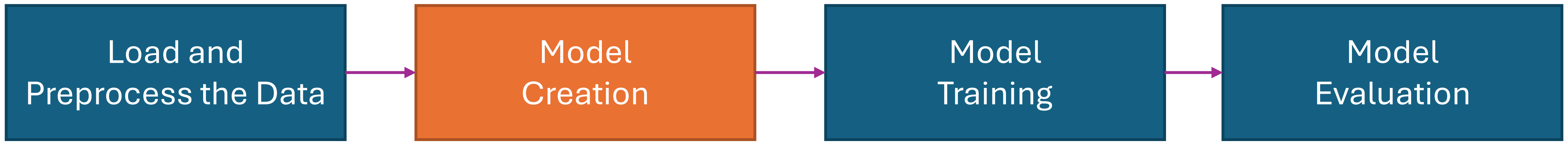
```
def load_and_preprocess_data():
    # Load CIFAR-10 dataset
    (X_train, y_train), (X_test, y_test) = tf.keras.datasets.cifar10.load_data()

    # Normalize pixel values to be between 0 and 1
    X_train = X_train.astype('float32') / 255.0
    X_test = X_test.astype('float32') / 255.0

    # Convert labels to categorical
    y_train = tf.keras.utils.to_categorical(y_train, 10)
    y_test = tf.keras.utils.to_categorical(y_test, 10)

    return X_train, y_train, X_test, y_test
```

Loads the CIFAR-10 dataset, which consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class

Normalize the pixel values of the images

Convert the class labels to a one-hot encoded format

CNN for Image Classification



```
model = tf.keras.Sequential([
```

First Convolutional Block

```
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Dropout(0.25),
```

These layers are responsible for detecting features like edges, textures, and patterns in the image

These layers are responsible for detecting features like edges, textures, and patterns in the image

This normalizes the output of the previous layer, improving the stability and performance of the network

This randomly drops a fraction of the units during training to prevent overfitting and improve generalization

Convolution Layer

First Convolutional Block

```
tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)),
tf.keras.layers.BatchNormalization(),
tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
tf.keras.layers.BatchNormalization(),
tf.keras.layers.MaxPooling2D((2, 2)),
tf.keras.layers.Dropout(0.25),
```

Each filter has a size of 3x3

This layer applies 32 convolutional filters to the input image

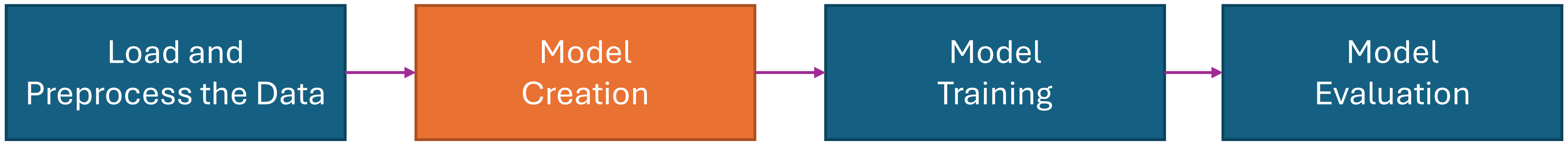
Ensures the output size remains the same as the input size

The shape of the input image is 32x32 pixels with 3 color channels

This layer reduces the spatial dimensions (height and width) of the input by a factor of 2 using max pooling.

This layer randomly drops 25% of the units during training to prevent overfitting and improve generalization

CNN for Image Classification



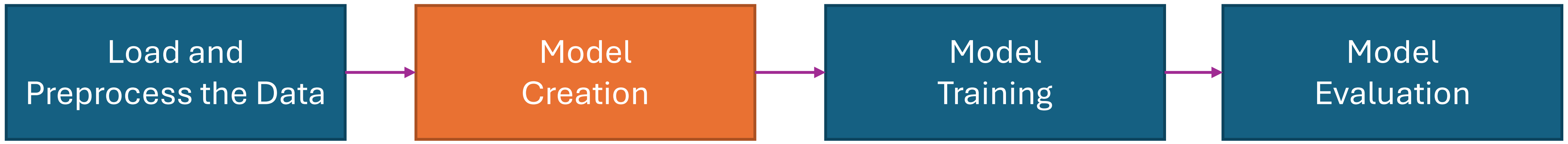
Second Convolutional Block

```
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Dropout(0.25),
```

Third Convolutional Block

```
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Dropout(0.25),
```

CNN for Image Classification

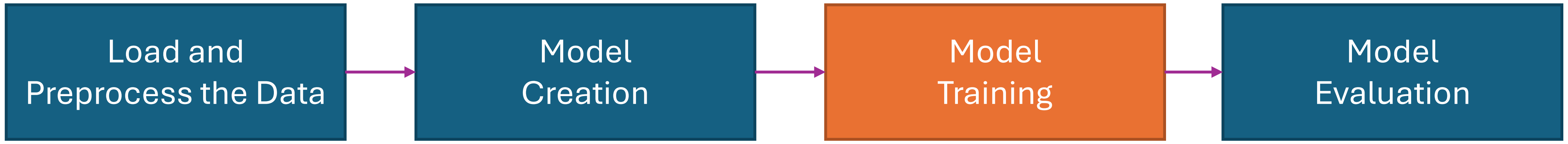


Dense Layers

```
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(512, activation='relu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(10, activation='softmax')
```

```
])
return model
```

CNN for Image Classification



Compiling the Model

```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
```

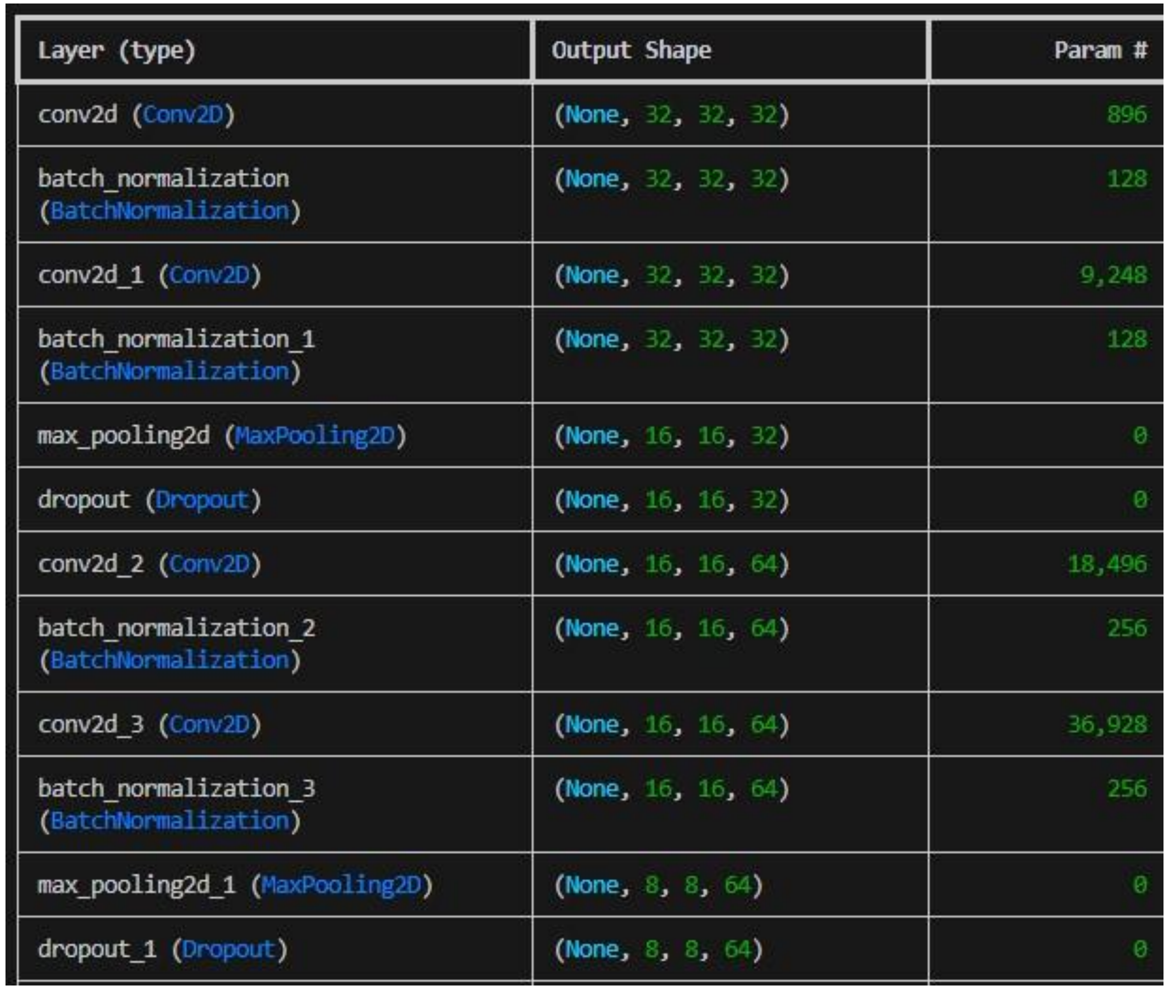
Compiles a CNN model with the Adam optimizer, categorical crossentropy loss, and accuracy metric, then prints a summary of the model architecture

Training the Model

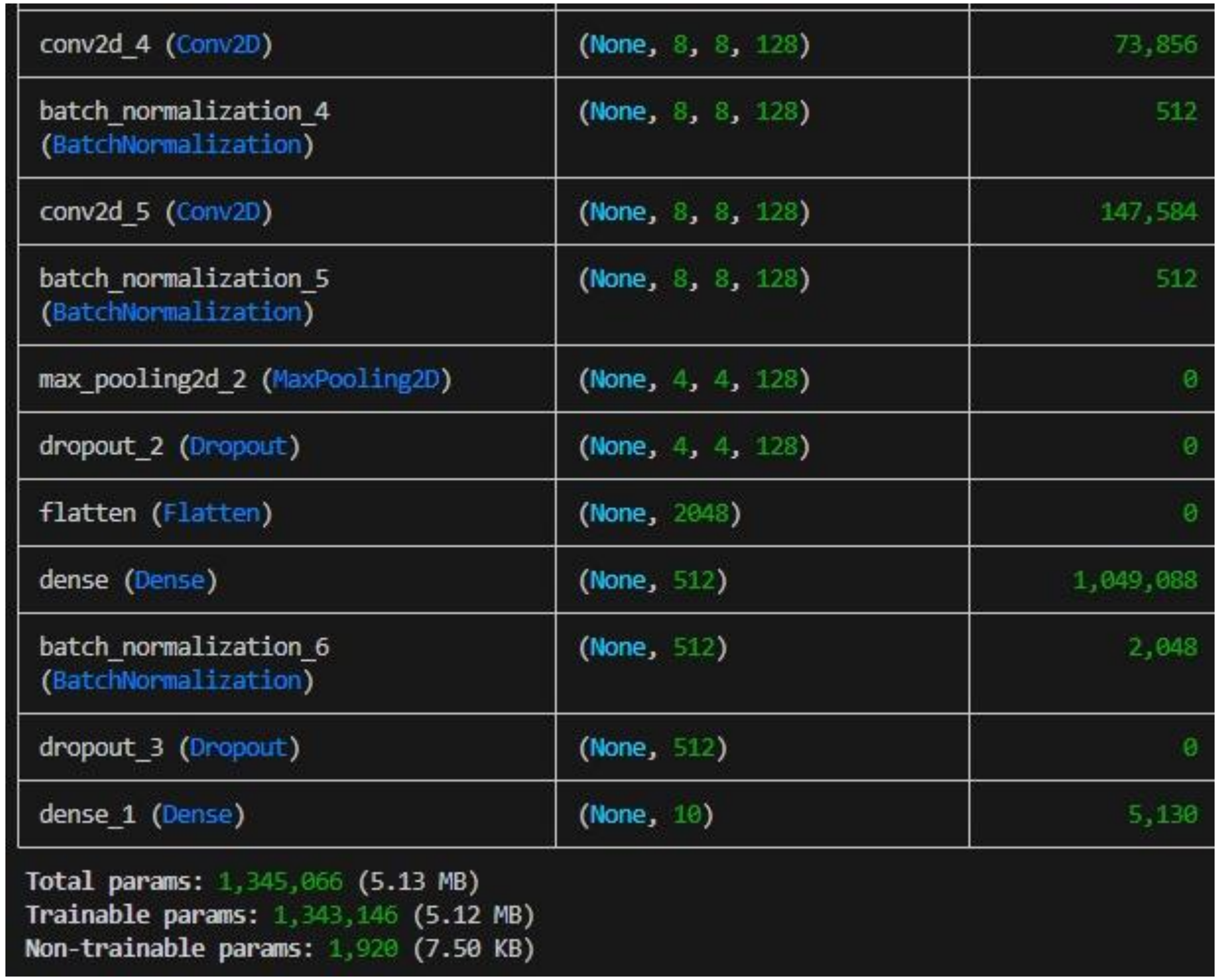
```
history = model.fit(X_train, y_train,
                    batch_size=128,
                    epochs=50,
                    validation_split=0.2,
                    verbose=1)
```

Trains the CNN model on the training data for 50 epochs with a batch size of 128, using 20% of the data for validation

Model Summary

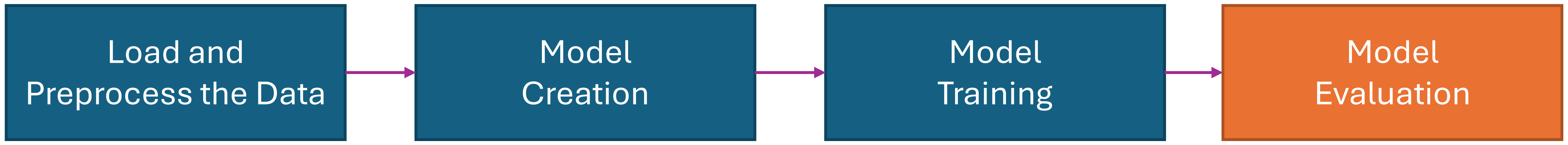


```
Layer (type)                     Output Shape               Param #
conv2d (Conv2D)                  (None, 32, 32, 32)             896
batch_normalization              (None, 32, 32, 32)             128
(BatchNormalization)
conv2d_1 (Conv2D)                (None, 32, 32, 32)           9,248
batch_normalization_1            (None, 32, 32, 32)             128
(BatchNormalization)
max_pooling2d (MaxPooling2D)     (None, 16, 16, 32)               0
dropout (Dropout)                (None, 16, 16, 32)               0
conv2d_2 (Conv2D)                (None, 16, 16, 64)          18,496
batch_normalization_2            (None, 16, 16, 64)             256
(BatchNormalization)
conv2d_3 (Conv2D)                (None, 16, 16, 64)          36,928
batch_normalization_3            (None, 16, 16, 64)             256
(BatchNormalization)
max_pooling2d_1 (MaxPooling2D)   (None, 8, 8, 64)                 0
dropout_1 (Dropout)              (None, 8, 8, 64)                 0
```



```
conv2d_4 (Conv2D)                (None, 8, 8, 128)           73,856
batch_normalization_4            (None, 8, 8, 128)              512
(BatchNormalization)
conv2d_5 (Conv2D)                (None, 8, 8, 128)          147,584
batch_normalization_5            (None, 8, 8, 128)              512
(BatchNormalization)
max_pooling2d_2 (MaxPooling2D)   (None, 4, 4, 128)                0
dropout_2 (Dropout)              (None, 4, 4, 128)                0
flatten (Flatten)                (None, 2048)                     0
dense (Dense)                    (None, 512)              1,049,088
batch_normalization_6            (None, 512)                  2,048
(BatchNormalization)
dropout_3 (Dropout)              (None, 512)                      0
dense_1 (Dense)                  (None, 10)                   5,130

Total params: 1,345,066 (5.13 MB)
Trainable params: 1,343,146 (5.12 MB)
Non-trainable params: 1,920 (7.50 KB)
```

CNN for Image Classification



Evaluate the Model

```
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"\nTest accuracy: {test_accuracy:.4f}")
print(f"Test loss: {test_loss:.4f}")
```

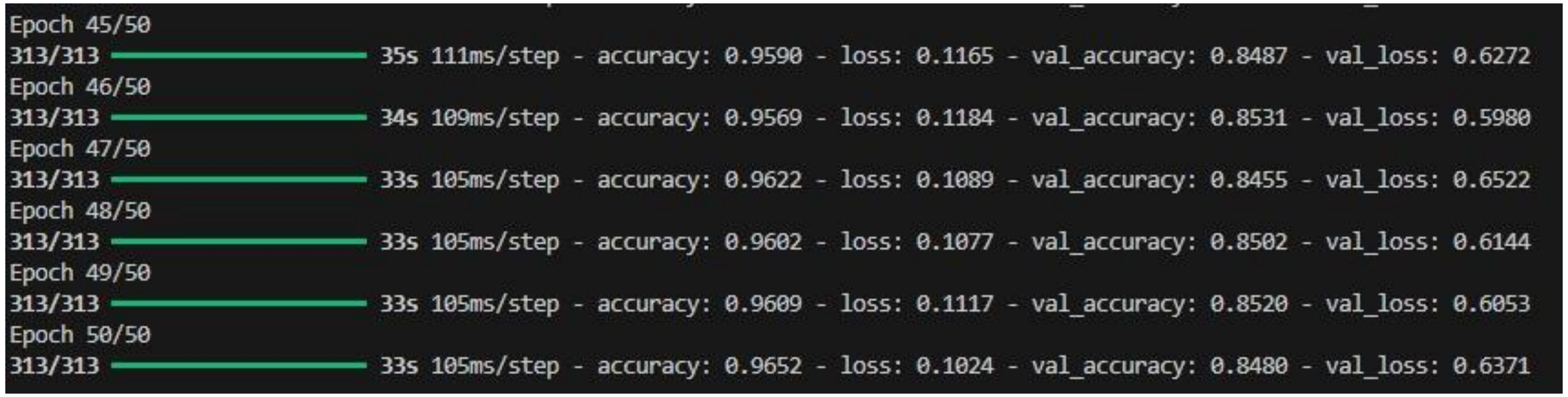
Evaluates the trained CNN model on the test data (X_test, y_test) to determine its performance by calculating the loss and accuracy

Evaluate the Model

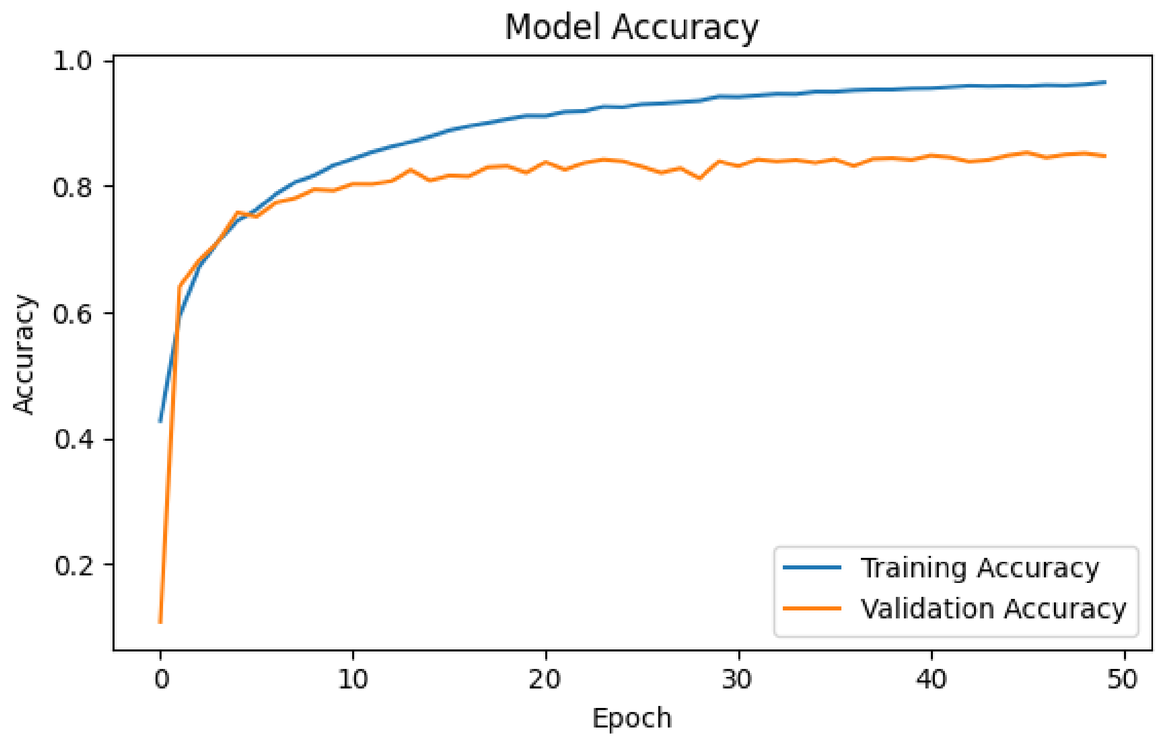
```
y_pred = model.predict(X_test)
print("\nClassification Report:")
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']
print(classification_report(np.argmax(y_test, axis=1),
                            np.argmax(y_pred, axis=1),
                            target_names=class_names))
```

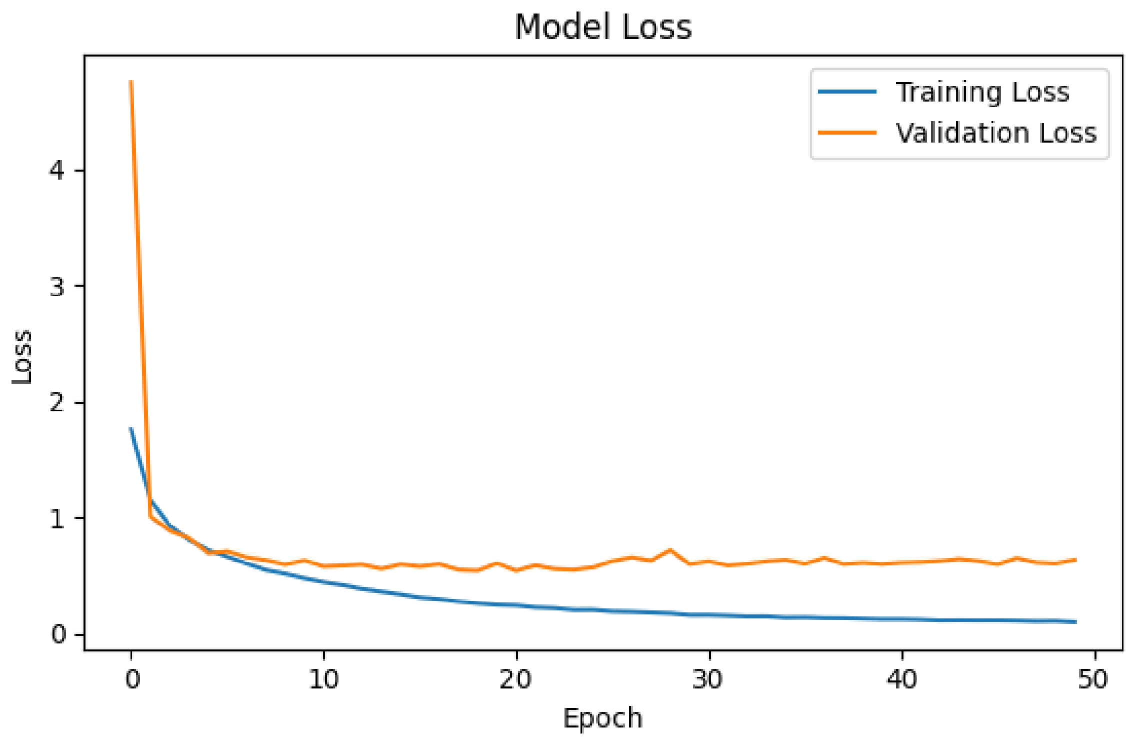
Makes predictions on the test data, then prints a detailed classification report that evaluates the model's performance across all classes

Plots

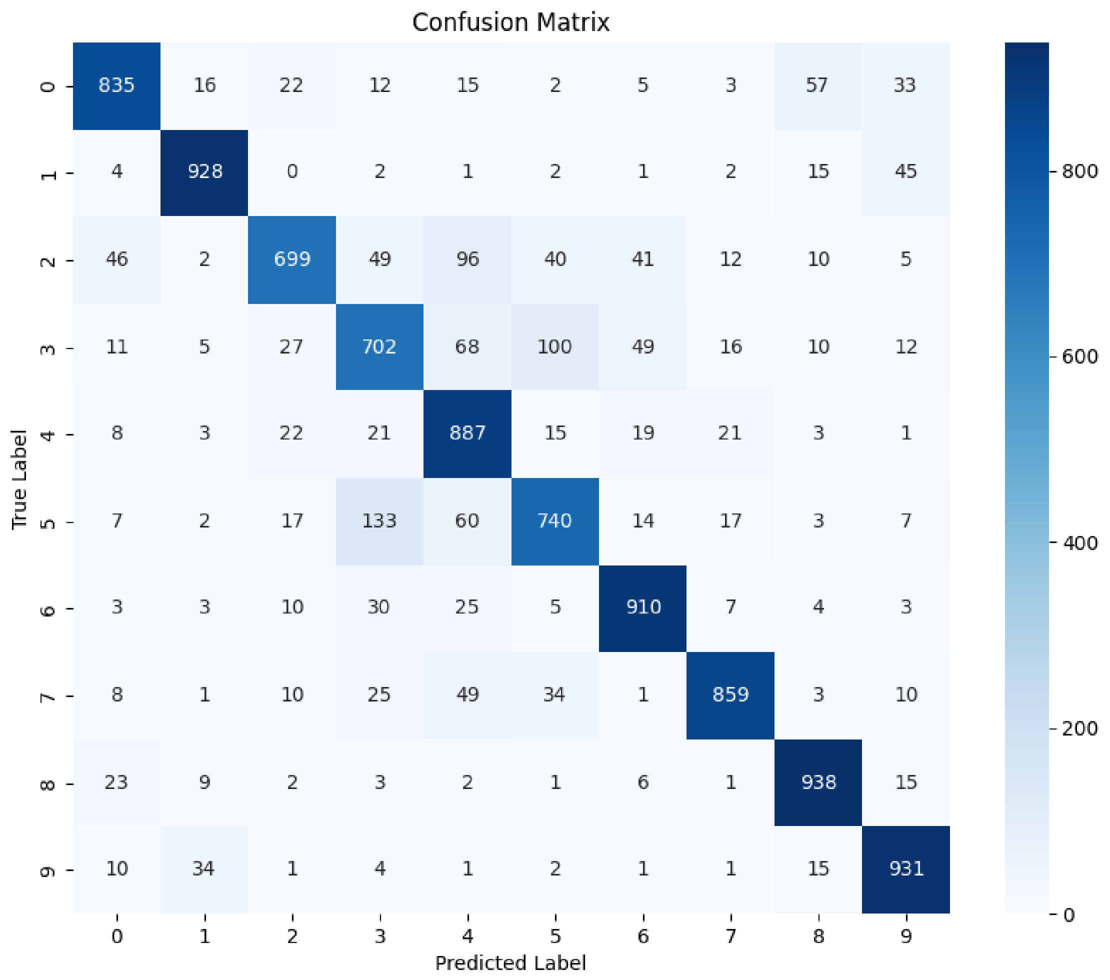


```
Epoch 45/50
313/313 ━━━━━━━━━━━━━━━━━━━━ 35s 111ms/step - accuracy: 0.9590 - loss: 0.1165 - val_accuracy: 0.8487 - val_loss: 0.6272
Epoch 46/50
313/313 ━━━━━━━━━━━━━━━━━━━━ 34s 109ms/step - accuracy: 0.9569 - loss: 0.1184 - val_accuracy: 0.8531 - val_loss: 0.5980
Epoch 47/50
313/313 ━━━━━━━━━━━━━━━━━━━━ 33s 105ms/step - accuracy: 0.9622 - loss: 0.1089 - val_accuracy: 0.8455 - val_loss: 0.6522
Epoch 48/50
313/313 ━━━━━━━━━━━━━━━━━━━━ 33s 105ms/step - accuracy: 0.9602 - loss: 0.1077 - val_accuracy: 0.8502 - val_loss: 0.6144
Epoch 49/50
313/313 ━━━━━━━━━━━━━━━━━━━━ 33s 105ms/step - accuracy: 0.9609 - loss: 0.1117 - val_accuracy: 0.8520 - val_loss: 0.6053
Epoch 50/50
313/313 ━━━━━━━━━━━━━━━━━━━━ 33s 105ms/step - accuracy: 0.9652 - loss: 0.1024 - val_accuracy: 0.8480 - val_loss: 0.6371
```

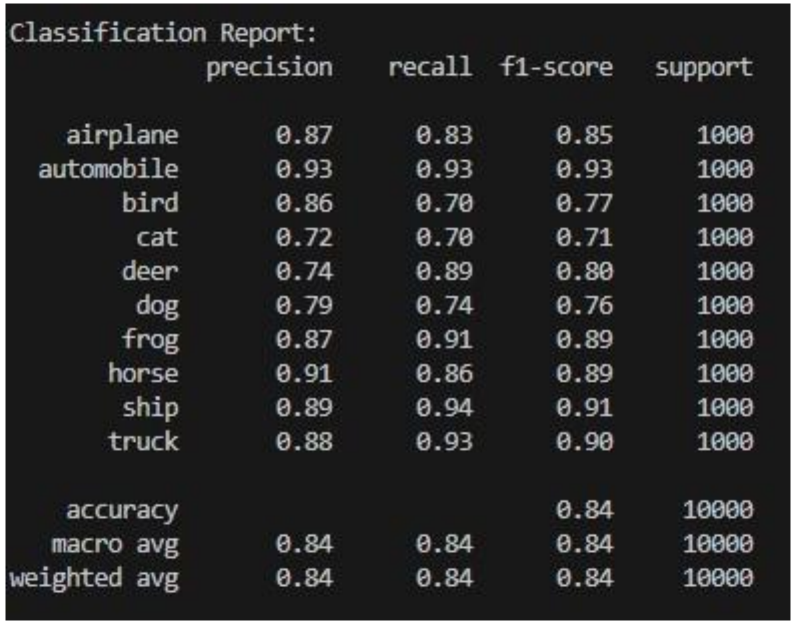




Confusion Matrix and Classification Report



	0	1	2	3	4	5	6	7	8	9
0	835	16	22	12	15	2	5	3	57	33
1	4	928	0	2	1	2	1	2	15	45
2	46	2	699	49	96	40	41	12	10	5
3	11	5	27	702	68	100	49	16	10	12
4	8	3	22	21	887	15	19	21	3	1
5	7	2	17	133	60	740	14	17	3	7
6	3	3	10	30	25	5	910	7	4	3
7	8	1	10	25	49	34	1	859	3	10
8	23	9	2	3	2	1	6	1	938	15
9	10	34	1	4	1	2	1	1	15	931



```
Classification Report:
              precision    recall  f1-score   support

    airplane       0.87      0.83      0.85      1000
  automobile       0.93      0.93      0.93      1000
        bird       0.86      0.70      0.77      1000
         cat       0.72      0.70      0.71      1000
        deer       0.74      0.89      0.80      1000
         dog       0.79      0.74      0.76      1000
        frog       0.87      0.91      0.89      1000
       horse       0.91      0.86      0.89      1000
        ship       0.89      0.94      0.91      1000
       truck       0.88      0.93      0.90      1000

    accuracy                           0.84     10000
   macro avg       0.84      0.84      0.84     10000
weighted avg       0.84      0.84      0.84     10000
```

Week 7 Exercises

1. Butterfly Species Classification Exercise

Objective: Using the Butterfly Image Classification dataset (https://www.kaggle.com/datasets/phucthaiv02/butterfly-image-classification) from Kaggle (which contains 75 different species of butterflies), your task is to build a CNN classifier with the following requirements:

Data Processing:

- Load and organize the butterfly images
- Resize images to 128x128 pixels
- Implement dataset splitting (70% training, 30% validation)
- Normalize pixel values to [0,1] range

Model Architecture

- Design a CNN from scratch with:
  - At least 3 convolutional blocks
  - Each block should contain Conv2D layer(s), appropriate activation, and pooling
  - Include regularization techniques (dropout and/or batch normalization)
  - Final dense layers for classification
- The total number of parameters should not exceed 5 million

Training and Evaluation:

- Train for 25-30 epochs
- Plot training/validation accuracy and loss curves
- Achieve minimum validation accuracy of 65%
- Generate a classification report for the validation set
- Create and visualize a confusion matrix for 10 selected species

Inferences

- Identify 5 butterfly species your model classifies most accurately
- Identify 5 species your model struggles with most
- Display 3 examples of misclassified butterflies and analyze possible reasons for misclassification