Project 3 - Report

Project Title: Deep Learning and Support Vector Machine Project

Objective:

To build two separate models:

1. **Classification Model**: A Convolutional Neural Network trained to classify mushroom images into one of 9 classes using a softmax output layer.

2. **Feature Extraction + SVM Model**: A CNN-based feature extractor that feeds its output into an SVM classifier for classification.

Dataset:

- The dataset consists of labeled mushroom images with 9 classes:
  - Agaricus
  - Amanita
  - Boletus
  - Cortinarius
  - Entoloma
  - Hygrocybe
  - Lactarius
  - Russula
  - Suillus
- The training data is in the folder structure as below.

```
Mushrooms
├── Agaricus
├── Amanita
├── Boletus
├── Cortinarius
├── Entoloma
├── Hygrocybe
├── Lactarius
├── Russula
└── Suillus
```

- The test data consists of images and a CSV file with image paths and labels as below.

image_path,label → mushrooms_test/test1.jpg,Amanita
Saved Model:
- Classification Model → classifier.keras
- Feature Extraction + SVM Model → svm_classifier.pkl

<u>Model Explanation:</u>

1) **Model 1: CNN Classification Model (with Softmax)**

   Purpose:

   This model is used for end-to-end classification of mushroom images into one of 9 species.

   Architecture Overview:

   - Input: Images resized to 128×128×3
   - Layers:
     - Data Augmentation (flip, rotate, zoom, brightness, etc.)
     - Convolutional Layers: 4 layers with increasing filters (32 → 64 → 128 → 128), using ReLU activation.
     - Pooling Layers: MaxPooling2D after each conv layer to reduce spatial dimensions.
     - GlobalAveragePooling2D to flatten before dense layers.
     - Dense Layer with 256 units + Dropout (0.5) for regularization.
     - Output Layer: Softmax activation with 9 units (for 9 classes).

   Output: A probability distribution across 9 classes — the class with the highest probability is the predicted label.

   Loss Function: categorical_crossentropy

   Metric: accuracy

2) **Model 2: Feature Extractor + SVM Classifier**

Purpose:

To extract high-level features from images using the CNN model and classify them using an SVM (Support Vector Machine) instead of softmax.

Architecture Overview:

- Uses the same CNN layers as Model 1 up to the last dropout layer, but removes the softmax classification layer.

- Output of the model is a 256-dimensional feature vector (from the last dense layer).

Pipeline:

1. Pass images through the CNN feature extractor.

2. Collect the feature vectors (X_train) and corresponding labels (y_train).

3. Train a Support Vector Machine (SVM) classifier (SVC(kernel='linear')) on these features.

4. Evaluate SVM on extracted validation features

Approach:

1. **Load and preprocess image data**

   - Used tf.keras.utils.image_dataset_from_directory to load images from the dataset folder.

   - Applied resizing to **128x128** resolution and used categorical labels for multi-class classification.

   - Split the dataset into 80% training and 20% validation using the validation_split argument.

2. **Apply data augmentation**

    - Created a custom Sequential pipeline using tf.keras.layers for transformations such as:

        - Horizontal flipping
        - Rotation
        - Zoom
        - Brightness and contrast adjustment
        - Translation

    - This helps the model generalize better and prevent overfitting.

3. **Build a custom CNN model with softmax output**

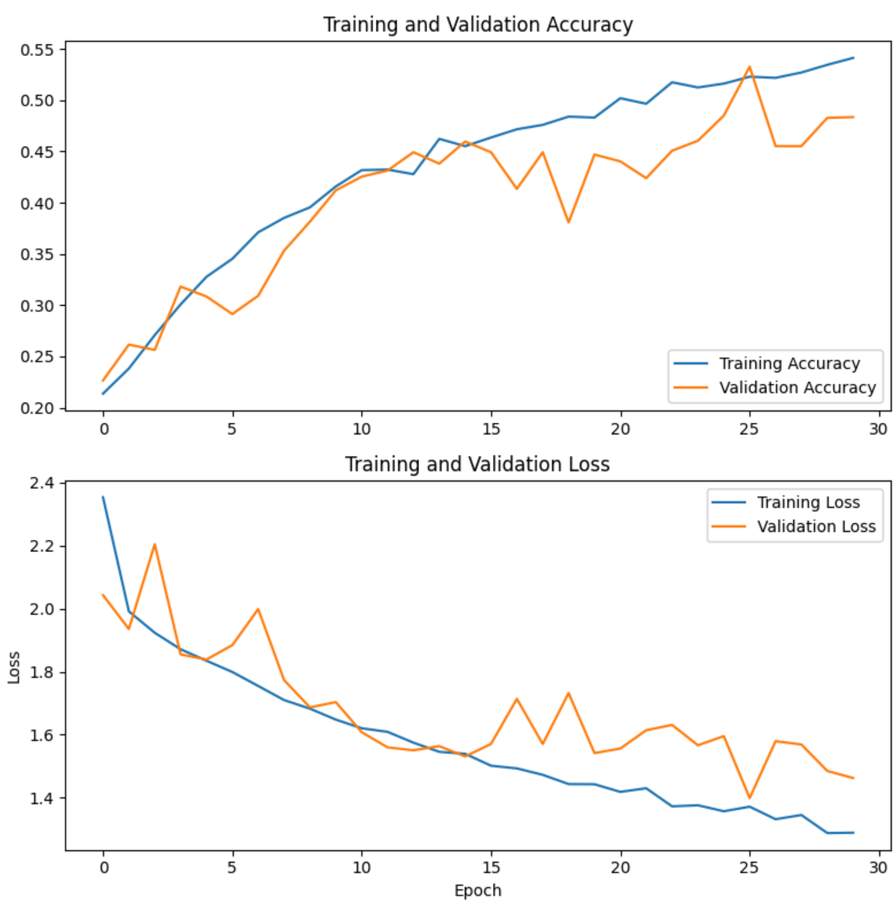
    - Constructed a CNN from scratch using Conv2D, MaxPooling2D, GlobalAveragePooling2D, and Dense layers.
    - Final layer is a Dense layer with softmax activation for 9-class classification.
    - Included Dropout (0.5) before the output layer to reduce overfitting.

Layer (type)	Output Shape	Param #
input_layer_16 (InputLayer)	(None, 128, 128, 3)	0
sequential_7 (Sequential)	(None, 128, 128, 3)	0
conv2d_32 (Conv2D)	(None, 128, 128, 32)	896
max_pooling2d_24 (MaxPooling2D)	(None, 64, 64, 32)	0
conv2d_33 (Conv2D)	(None, 64, 64, 64)	18,496
max_pooling2d_25 (MaxPooling2D)	(None, 32, 32, 64)	0
conv2d_34 (Conv2D)	(None, 32, 32, 128)	73,856
max_pooling2d_26 (MaxPooling2D)	(None, 16, 16, 128)	0
conv2d_35 (Conv2D)	(None, 16, 16, 128)	147,584
global_average_pooling2d_8 (GlobalAveragePooling2D)	(None, 128)	0
dense_16 (Dense)	(None, 256)	33,024
dropout_8 (Dropout)	(None, 256)	0
dense_17 (Dense)	(None, 9)	2,313

4. **Train the CNN model**

    - Compiled with **ADAM** optimizer and **categorical crossentropy** loss function.
    - Trained the model for 30 epochs using model.fit() with augmented training and validation datasets.
    - Saved the model in .keras format for reuse: model.save("trial_model.keras").

```
Epoch 1/30
167/167 ━━━━━━━━━━━━━━━━━━━━ 116s 622ms/step - accuracy: 0.1980 - loss: 3.9597 - val_accuracy: 0.2303 - val_loss: 2.0340
Epoch 2/30
167/167 ━━━━━━━━━━━━━━━━━━━━ 118s 701ms/step - accuracy: 0.2315 - loss: 2.0061 - val_accuracy: 0.2474 - val_loss: 1.9620
Epoch 3/30
167/167 ━━━━━━━━━━━━━━━━━━━━ 108s 641ms/step - accuracy: 0.2640 - loss: 1.9454 - val_accuracy: 0.2861 - val_loss: 1.9385
Epoch 4/30
167/167 ━━━━━━━━━━━━━━━━━━━━ 98s 585ms/step - accuracy: 0.3039 - loss: 1.8854 - val_accuracy: 0.3212 - val_loss: 1.8805
Epoch 5/30
167/167 ━━━━━━━━━━━━━━━━━━━━ 109s 653ms/step - accuracy: 0.3148 - loss: 1.8664 - val_accuracy: 0.2988 - val_loss: 1.8839
Epoch 6/30
167/167 ━━━━━━━━━━━━━━━━━━━━ 110s 654ms/step - accuracy: 0.3137 - loss: 1.8471 - val_accuracy: 0.2705 - val_loss: 1.8973
Epoch 7/30
167/167 ━━━━━━━━━━━━━━━━━━━━ 100s 598ms/step - accuracy: 0.3434 - loss: 1.8014 - val_accuracy: 0.3465 - val_loss: 1.7638
Epoch 8/30
167/167 ━━━━━━━━━━━━━━━━━━━━ 95s 570ms/step - accuracy: 0.3758 - loss: 1.7350 - val_accuracy: 0.3025 - val_loss: 1.8914
Epoch 9/30
167/167 ━━━━━━━━━━━━━━━━━━━━ 101s 602ms/step - accuracy: 0.3766 - loss: 1.7111 - val_accuracy: 0.3696 - val_loss: 1.7240
Epoch 10/30
167/167 ━━━━━━━━━━━━━━━━━━━━ 83s 498ms/step - accuracy: 0.3975 - loss: 1.6747 - val_accuracy: 0.3793 - val_loss: 1.7704
Epoch 11/30
167/167 ━━━━━━━━━━━━━━━━━━━━ 76s 452ms/step - accuracy: 0.4014 - loss: 1.6516 - val_accuracy: 0.4367 - val_loss: 1.5947
Epoch 12/30
167/167 ━━━━━━━━━━━━━━━━━━━━ 76s 456ms/step - accuracy: 0.4205 - loss: 1.6216 - val_accuracy: 0.4143 - val_loss: 1.6578
Epoch 13/30
...
Epoch 29/30
167/167 ━━━━━━━━━━━━━━━━━━━━ 65s 385ms/step - accuracy: 0.5486 - loss: 1.2779 - val_accuracy: 0.5171 - val_loss: 1.4795
Epoch 30/30
167/167 ━━━━━━━━━━━━━━━━━━━━ 67s 400ms/step - accuracy: 0.5495 - loss: 1.2578 - val_accuracy: 0.4747 - val_loss: 1.5388
```



5. **Build a CNN-based feature extractor model**

    ○ Removed the final softmax layer and used the CNN base up to the dense representation layer (256 units).

    ○ This model outputs feature vectors instead of class predictions.

Layer (type)	Output Shape	Param #
input_layer_18 (InputLayer)	(None, 128, 128, 3)	0
sequential_7 (Sequential)	(None, 128, 128, 3)	0
conv2d_40 (Conv2D)	(None, 128, 128, 32)	896
max_pooling2d_30 (MaxPooling2D)	(None, 64, 64, 32)	0
conv2d_41 (Conv2D)	(None, 64, 64, 64)	18,496
max_pooling2d_31 (MaxPooling2D)	(None, 32, 32, 64)	0
conv2d_42 (Conv2D)	(None, 32, 32, 128)	73,856
max_pooling2d_32 (MaxPooling2D)	(None, 16, 16, 128)	0
conv2d_43 (Conv2D)	(None, 16, 16, 128)	147,584
global_average_pooling2d_10 (GlobalAveragePooling2D)	(None, 128)	0
dense_19 (Dense)	(None, 256)	33,024
dropout_10 (Dropout)	(None, 256)	0

6. **Extract features and train an SVM**

    ○ Used the CNN feature extractor to generate fixed-size feature vectors for both training and validation datasets.

    ○ Flattened labels from one-hot to class indices using np.argmax().

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 127ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 68ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 72ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 78ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 78ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 68ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 99ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 72ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step
...
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 92ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 101ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 100ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 104ms/step
```

7. **Train and evaluate an SVM classifier**

    - Used scikit-learn's SVC with a linear kernel.

- Trained it on the CNN-generated features.
- Evaluated the SVM model on validation features using accuracy_score.
- Saved the trained model as svm_classifier.pkl using joblib.

8. **Visualize training performance**
   - Plotted training and validation accuracy and loss over epochs using matplotlib.

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits
SVM classifier saved as svm_classifier.pkl
Best parameters: {'C': 1, 'max_iter': 1000}
```

```
SVM Classifier Accuracy: 0.4396
```

<u>Testing:</u>

The`proj3_classification_test.py` and proj3_extractSVM_test.py script loads the saved model and processes the test CSV and images.

<u>Requirements:</u>

python==3.11.0
pip==22.3
tensorflow==2.19.0
numpy==1.24.3
pandas==2.2.3
matplotlib==3.10.1
scikit-learn==1.6.1
joblib==1.5.0

<u>Instructions to Run the Test Script:</u>

1. Set up the Python environment. See requirements.txt for dependencies.

2. Run the test script for Model 1 using:
   python proj3_classification_test.py --model <model_path> --test_csv <test_data_csv_path>

eg.
python proj3_classification_test.py --model classification.keras --test_csv mushrooms_test.csv

3. Run the test script for Model 2 using:
python proj3_extractSVM_test.py --model <model_path> --test_csv <test_data_csv_path> --feature_model <feature_model_path>

eg.
python proj3_extractSVM_test.py --model svm_classifier.pkl --feature_model feature_extractor.keras

3. Output - Model accuracy

Note: Ensure the model_path and test_data_csv_path are correct before running the script.