Binary classifier using Support Vector Machine models

Nikolaos Karapatsias 10661 (Department of Electrical and Computing Engineering AUTH)

This document provides an implementation of a binary classifier using Support Vector Machine (SVM) models. We used linear, Radial Basis Function (RBF) and Polynomial as type of kernels with a wide range of different parameters. This model is trained and tested with two out of 10 classes of the dataset of Cifar-10 which is consisted of 50.000 training samples and 10.000 testing samples all cassified in 10 groups. More details can be found on cifar website

I. THE PREPERATION CODE

a) Firstly we are **loading** up all of the **libraries** that are required and also we are **loading** the **Cifar10** dataset in 4 arrays each corresponding to x,y training or testing data

```
import numpy as np
import time
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import StandardScaler
from keras.datasets import cifar10

# Load CIFAR-10 into our programm
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

b) After that, we are creating a "labels" numpy array to make our project more general in case the user wants to pick 2 other classes of his choice. Also we are picking correspondingly to "labels" array the desired 2 classes. For the purposes of this document we picked the worst case scenario that cifar10 can provide which is classification of "Dog" and "Cat". Finally we are filtering out the data that contains the two classes we need and combining them into x_train,test and y_train,test arrays

```
# Class labels in CIFAR-10 dataset to make the project more general
labels = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

# Select the two desired classes
class_1, class_2 = 3, 5
mask_train = (y_train.flatten() == class_1) | (y_train.flatten() == class_2)
mask_test = (y_test.flatten() == class_1) | (y_test.flatten() == class_2)

x_train, y_train = x_train[mask_train], y_train[mask_train]
x_test, y_test = x_test[mask_test], y_test[mask_test]
```

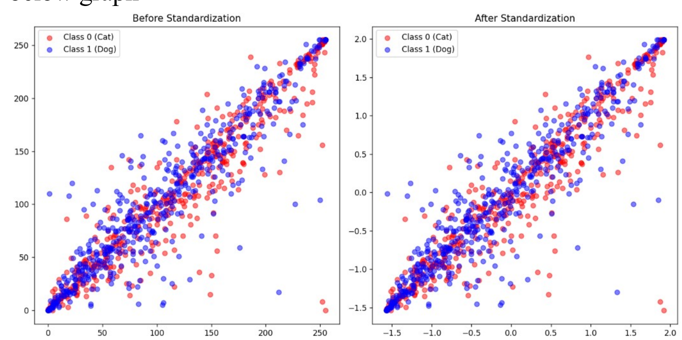
c) In this step we are **relabeling** y_train and y_test to 0 and 1 instead of 4 and 6 for simplicity reasons. Also we are **reshaping** our data to match the desired format that is set from Scikit learn library for SVM models.

```
# Relabel classes to 0 and 1
y_train = (y_train.flatten() == class_2).astype(int)
y_test = (y_test.flatten() == class_2).astype(int)

# Reshaping our data to fit sckit learn SVM requirments
x_train_flat = x_train.reshape(x_train.shape[0], -1)
x_test_flat = x_test.reshape(x_test.shape[0], -1)
```

d) **Standardizing dataset**

In this step, we are rescaling our data to have a mean of zero and a variance of . To do that, for each element we are subtracting the mean and dividing the result by the variance. In that way our data get values between -1.5 and 2 as shown in the below graph



e) **Principal Component Analysis (PCA)**

Lastly, we are applying principal component analysis (or known as PCA) where we are lowering our dataset dimensions from 3072 to 88 without loosing more than 10% of information in order to speed up training and reduce computational complexity .

```
Before PCA
X_train: (10000, 3072)
X_test: (2000, 3072)

After PCA
X_train: (10000, 88)
X_test: (2000, 88)
```

II. KERNELS

A) Linear kernel

Results

In that model, we chose to implement an SVM with a linear kernel while we used different values of C, which controls the trade-off between maximizing the margin and minimizing classification errors. As shown in the below pictures, the values of C we chose were 0.1 and 1. Even though C is getting bigger we are not having any significant increments on either training and testing accuracy as we expected, because our classes are not linearly separable. The best testing accuracy we achieved was **61.15% at 328.12 seconds**

```
SVM Classification Report with C: 0.1:
Training accuracy: 61.339999999999996%
Test accuracy: 61.0%
Execution time: 59.21244 seconds

SVM Classification Report with C: 1:
Training accuracy: 61.3%
Test accuracy: 61.150000000000006%
Execution time: 328.12970 seconds
```

Code

The code for linear kernel impmentation was pretty simple as scikit learn library provides all the required tools to declare a linear model using SVC() function, train it using the .fit() function and extracting the results using .predict() function. We also needed to run the model for various values of C, thats why we are declaring an array C in the beginning and using a for loop for each of its elements. Finallym the .perf_counter() helped us to calculate the execution time for each model

```
C = [0.1, 1]

for i in C:
    start_time = time.perf_counter()
    svm_model = SVC(kernel='linear', C=i)
    svm_model.fit(x_train_pca, y_train)
    y_pred_train = svm_model.predict(x_train_pca)
    y_pred = svm_model.predict(x_test_pca)
    end_time = time.perf_counter()
    execution_time = end_time - start_time

    print(f"\nSVM Classification Report with C: {i}:")
    print(f"Training accuracy: {accuracy_score(y_train, y_pred_train)*100}%")
    print(f"Test accuracy: {accuracy_score(y_test, y_pred)*100}%")
    print(f"Execution time: {execution_time:.5f} seconds")
```

B) Polynomial kernel

Results

In that part of our project we implemented an SVM with polynomial kernel following the formula $(\gamma * x^T y + r)^d$ where d is the degree of the polynomial and γ (gamma) is the scaling factor. The parameter C was chosen to take discrete values $\mathbf{10^{-3}, 10^{-2}, 10^{-1}}$ **and 1** while γ and d took values **0.001, 0.01, 0.1, 1, 10** and **2, 3, 4** respectively. The results for each machine are shown below:

```
SVM Classification Report with C: 0.001, gamma: 0.001, degree: 2:
Training accuracy: 53.45%
Testing accuracy: 52.45%
Execution time: 17.05925 seconds

SVM Classification Report with C: 0.001, gamma: 0.001, degree: 3:
Training accuracy: 58.65%
Testing accuracy: 57.20%
Execution time: 15.82943 seconds

SVM Classification Report with C: 0.001, gamma: 0.001, degree: 4:
Training accuracy: 57.84%
Testing accuracy: 53.65%
Execution time: 15.71487 seconds

SVM Classification Report with C: 0.001, gamma: 0.01, degree: 2:
Training accuracy: 69.53%
Testing accuracy: 61.30%
Execution time: 14.10967 seconds

SVM Classification Report with C: 0.001, gamma: 0.01, degree: 3:
Training accuracy: 97.45%
Testing accuracy: 60.75%
Execution time: 44.53193 seconds

SVM Classification Report with C: 0.001, gamma: 0.01, degree: 4:
Training accuracy: 99.96%
Testing accuracy: 60.10%
Execution time: 32.44870 seconds

SVM Classification Report with C: 0.001, gamma: 0.1, degree: 2:
Training accuracy: 84.98%
Testing accuracy: 60.70%
Execution time: 46.18574 seconds

SVM Classification Report with C: 0.001, gamma: 0.1, degree: 3:
Training accuracy: 100.00%
Testing accuracy: 58.75%
Execution time: 51.38294 seconds

SVM Classification Report with C: 0.001, gamma: 0.1, degree: 4:
Training accuracy: 100.00%
Testing accuracy: 58.25%
Execution time: 28.62357 seconds
```

C=0.001

```
SVM Classification Report with C: 0.01, gamma: 0.001, degree: 2:
Training accuracy: 59.95%
Testing accuracy: 57.50%
Execution time: 15.79905 seconds

SVM Classification Report with C: 0.01, gamma: 0.001, degree: 3:
Training accuracy: 68.31%
Testing accuracy: 60.80%
Execution time: 15.46361 seconds

SVM Classification Report with C: 0.01, gamma: 0.001, degree: 4:
Training accuracy: 73.52%
Testing accuracy: 57.75%
Execution time: 15.02829 seconds

SVM Classification Report with C: 0.01, gamma: 0.01, degree: 2:
Training accuracy: 78.18%
Testing accuracy: 63.05%
Execution time: 16.63232 seconds

SVM Classification Report with C: 0.01, gamma: 0.01, degree: 3:
Training accuracy: 99.95%
Testing accuracy: 58.75%
Execution time: 56.00351 seconds

SVM Classification Report with C: 0.01, gamma: 0.01, degree: 4:
Training accuracy: 100.00%
Testing accuracy: 59.15%
Execution time: 30.47207 seconds
```

C=0.01

```
SVM Classification Report with C: 1, gamma: 0.001, degree:
Training accuracy: 78.18%
Testing accuracy: 63.05%
Execution time: 17.54406 seconds

SVM Classification Report with C: 1, gamma: 0.001, degree:
Training accuracy: 97.45%
Testing accuracy: 60.75%
Execution time: 44.67910 seconds

SVM Classification Report with C: 1, gamma: 0.001, degree:
Training accuracy: 99.58%
Testing accuracy: 59.60%
Execution time: 31.63244 seconds
```

C=1

```
SVM Classification Report with C: 0.1, gamma: 0.001, degree: 2:
Training accuracy: 69.53%
Testing accuracy: 61.25%
Execution time: 15.05643 seconds

SVM Classification Report with C: 0.1, gamma: 0.001, degree: 3:
Training accuracy: 85.67%
Testing accuracy: 61.75%
Execution time: 18.54613 seconds

SVM Classification Report with C: 0.1, gamma: 0.001, degree: 4:
Training accuracy: 93.74%
Testing accuracy: 61.45%
Execution time: 17.80730 seconds

SVM Classification Report with C: 0.1, gamma: 0.01, degree: 2:
Training accuracy: 84.98%
Testing accuracy: 60.70%
Execution time: 47.31086 seconds

SVM Classification Report with C: 0.1, gamma: 0.01, degree: 3:
Training accuracy: 100.00%
Testing accuracy: 58.35%
Execution time: 56.32428 seconds

SVM Classification Report with C: 0.1, gamma: 0.01, degree: 4:
Training accuracy: 100.00%
Testing accuracy: 58.45%
Execution time: 30.91814 seconds
```

C=0.1

From these results we can observe that the higher the values of C, γ and d the higher the possibility the difference between training accuracy and test accuary is big, which leads to overfitting. For example, by setting C= 10^{-3} while keeping gamma at either 10^{-3} or 10^{-2} and degree at 2 we are getting a balanced model. But if we set the value of gamma at 10^{-1} the model overfits immediately. The same applies for gamma = 10^{-2} and a degree of 3 or higher class. Also we can observe from the other 3 figures that when C gets higher our model gets more strict in the increament of the other 3 parameters before it overfits.

The execution time for each model that didn't overfit is around 14 to 17 seconds while the ones who overfit are having a wide range between 16 to 56

In conclusion, the best testing accuracy we achieved preventing our model from overfitting is **61.25%** by setting C= 10^{-1}, γ= 10^{-3} and d=2 with an execution time of around **15 seconds**

Code

Similarly to linear kernel, scikit learn library provides all the required functions to implement a polynomial kernel SVM model. We also used nested for loops to produce different models with a wide range of values for each parameter

```
C = [0.001, 0.01, 0.1, 1]

G = [0.001, 0.01, 0.1, 1, 10]
DEGREE = [2, 3, 4]  # Add degrees for the polynomial kernel

for i in C:
    for j in G:
        for d in DEGREE:
            start_time = time.perf_counter()
            svm_model = SVC(kernel='poly', gamma=j, C=i, degree=d)
            svm_model.fit(x_train_pca, y_train)

            y_train_pred = svm_model.predict(x_train_pca)
            train_accuracy = accuracy_score(y_train, y_train_pred)
            y_pred = svm_model.predict(x_test_pca)
            test_accuracy = accuracy_score(y_test, y_pred)
            end_time = time.perf_counter()
            execution_time = end_time - start_time

            print(f"\nSVM Classification Report with C: {i}, gamma: {j}, degree: {d}:")
            print(f"Training accuracy: {train_accuracy * 100:.2f}%")
            print(f"Testing accuracy: {test_accuracy * 100:.2f}%")
            print(f"Execution time: {execution_time:.5f} seconds")
```

c) Radial basis Function (RBF) kernel

Results:

In that case, we implemented an SVM model using Radial basis function as kernel with a formula the below formula: K(xi,xj)=exp(−γ‖xi−xj‖2) where **γ** determines the spread of the kernel. The parameter C and γ was chosen to take discrete values **10^{-5},10^{-4},10^{-3},10^{-2} and 10^{-1}.** The results for each machine are shown below:

```
SVM Classification Report with C: 1e-05, gamma: 1e-05:
Training Accuracy: 58.14%
Testing Accuracy: 58.20%
Execution time: 23.07653 seconds

SVM Classification Report with C: 1e-05, gamma: 0.0001:
Training Accuracy: 59.15%
Testing Accuracy: 59.55%
Execution time: 22.94590 seconds

SVM Classification Report with C: 1e-05, gamma: 0.001:
Training Accuracy: 58.74%
Testing Accuracy: 57.40%
Execution time: 22.63770 seconds

SVM Classification Report with C: 1e-05, gamma: 0.01:
Training Accuracy: 100.00%
Testing Accuracy: 50.45%
Execution time: 21.93188 seconds

SVM Classification Report with C: 1e-05, gamma: 0.1:
Training Accuracy: 100.00%
Testing Accuracy: 50.00%
Execution time: 22.76138 seconds
```

C=0.00001

```
SVM Classification Report with C: 0.0001, gamma: 1e-05:
Training Accuracy: 58.14%
Testing Accuracy: 58.20%
Execution time: 22.55261 seconds

SVM Classification Report with C: 0.0001, gamma: 0.0001:
Training Accuracy: 59.15%
Testing Accuracy: 59.55%
Execution time: 22.92224 seconds

SVM Classification Report with C: 0.0001, gamma: 0.001:
Training Accuracy: 58.74%
Testing Accuracy: 57.40%
Execution time: 22.62719 seconds

SVM Classification Report with C: 0.0001, gamma: 0.01:
Training Accuracy: 100.00%
Testing Accuracy: 50.45%
Execution time: 21.82504 seconds

SVM Classification Report with C: 0.0001, gamma: 0.1:
Training Accuracy: 100.00%
Testing Accuracy: 50.00%
Execution time: 22.72426 seconds
```

C=0.0001

```
SVM Classification Report with C: 0.001, gamma: 1e-05:
Training Accuracy: 58.14%
Testing Accuracy: 58.20%
Execution time: 22.60319 seconds

SVM Classification Report with C: 0.001, gamma: 0.0001:
Training Accuracy: 59.15%
Testing Accuracy: 59.55%
Execution time: 22.99392 seconds

SVM Classification Report with C: 0.001, gamma: 0.001:
Training Accuracy: 58.74%
Testing Accuracy: 57.40%
Execution time: 22.55776 seconds

SVM Classification Report with C: 0.001, gamma: 0.01:
Training Accuracy: 100.00%
Testing Accuracy: 50.45%
Execution time: 21.97277 seconds

SVM Classification Report with C: 0.001, gamma: 0.1:
Training Accuracy: 100.00%
Testing Accuracy: 50.00%
Execution time: 22.79116 seconds
```

C=0.001

```
SVM Classification Report with C: 0.01, gamma: 1e-05:
Training Accuracy: 58.14%
Testing Accuracy: 58.20%
Execution time: 22.75687 seconds

SVM Classification Report with C: 0.01, gamma: 0.0001:
Training Accuracy: 59.20%
Testing Accuracy: 59.65%
Execution time: 22.47699 seconds

SVM Classification Report with C: 0.01, gamma: 0.001:
Training Accuracy: 58.74%
Testing Accuracy: 57.40%
Execution time: 22.48425 seconds

SVM Classification Report with C: 0.01, gamma: 0.01:
Training Accuracy: 100.00%
Testing Accuracy: 50.45%
Execution time: 22.51577 seconds

SVM Classification Report with C: 0.01, gamma: 0.1:
Training Accuracy: 100.00%
Testing Accuracy: 50.00%
Execution time: 23.25796 seconds
```

C=0.01

```
SVM Classification Report with C: 0.1, gamma: 1e-05:
Training Accuracy: 59.66%
Testing Accuracy: 59.05%
Execution time: 21.89699 seconds

SVM Classification Report with C: 0.1, gamma: 0.0001:
Training Accuracy: 61.59%
Testing Accuracy: 61.95%
Execution time: 20.65076 seconds

SVM Classification Report with C: 0.1, gamma: 0.001:
Training Accuracy: 65.91%
Testing Accuracy: 60.70%
Execution time: 21.80137 seconds

SVM Classification Report with C: 0.1, gamma: 0.01:
Training Accuracy: 100.00%
Testing Accuracy: 50.45%
Execution time: 21.51533 seconds

SVM Classification Report with C: 0.1, gamma: 0.1:
Training Accuracy: 100.00%
```

In conclusion, from these results we can clearly see that no matter the value of C, our model overfits for gamma equal or higher than 10^{-2} as the training accuracy reaches 100%. The best test accuracy we achieved in a balanced model (without overfitting or underfitting) is **61.95%** with C=0.1 and gamma=10^{-4} at **21.8 seconds**

Code

Similarly to the two previous kernel, scikit learn library provides all the required functions to implement a polynomial kernel SVM model. We also used nested for loops to produce different models for different values of C and γ

```
C = [0.00001, 0.0001, 0.001, 0.01, 0.1]
GAMMA = [0.00001, 0.0001, 0.001, 0.01, 0.1]


for i in C:
    for j in GAMMA:
        start_time = time.perf_counter()
        svm_model = SVC(kernel='rbf', gamma=j, C=i)
        svm_model.fit(x_train_pca, y_train)
        y_train_pred = svm_model.predict(x_train_pca)
        train_accuracy = accuracy_score(y_train, y_train_pred)
        y_pred = svm_model.predict(x_test_pca)
        test_accuracy = accuracy_score(y_test, y_pred)
        end_time = time.perf_counter()
        execution_time = end_time - start_time

        print(f"\nSVM Classification Report with C: {i}, gamma: {j}:")
        print(f"Training Accuracy: {train_accuracy * 100:.2f}%")
        print(f"Testing Accuracy: {test_accuracy * 100:.2f}%")
        print(f"Execution time: {execution_time:.5f} seconds")
```

III. EXAMPLES OF RIGHT AND WRONG CLASSIFICATION

*Fig 1: Right class using **linear** kernel $C=10^{-1}$*

*Fig2: Wrong class using **linear** kernel $C=10^{-1}$*

IV. VS MLP NETWORK

In that section we are comparing our SVM models with a multilayer perceptron neural network (MLP) with one hidden layer which uses hinge loss to calculating the training loss. We produced an initial model using **ChatGPT** and used it as a starting point to produce our model.

A) *Code explanation*

First of all, we are loading up the set of libraries we are willing to use and creating our model. The first layer is an input layer with an input shape equal to the shape of x_train_pca extracted by the first cell, a hidden layer of 256 neurons (each using a leaky relu as an activation function) and an output layer of one neuron using softmax as an activation function

```
from keras.models import Sequential
from keras.layers import Dense, Input, LeakyReLU
from keras.optimizers import Adam
from keras.losses import hinge
from keras.callbacks import LearningRateScheduler
import numpy as np


model = Sequential([
    Input(shape=(x_train_pca.shape[1],)),  # Input layer with shape eequal to x_train_pca array
    Dense(256),                            # Hidden layer with 128 neurons
    LeakyReLU(alpha=0.01),                 # Leaky ReLU activation with a negative slope of 0.01
    Dense(1, activation='sigmoid')          # Output layer with linear activation for hinge loss
])
```

Then, we are setting up our training parameters choosing "Adam" as optimizer with a learning rate of 10^4 and a "hinge loss" to calculate the loss.

```
# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001), loss=hinge, metrics=['accuracy'])
```

From there, we are ready to train our model using a variable learning rate and running it for 25 epochs with a batch size equal to 32. Finally, we are printing out the required info for each epoch and our final model accuracy.

```
# Learning rate scheduler function
def lr_schedule(epoch):
    if epoch <= 8:
        return 0.0001
    else:
        return 0.0001 * 0.75

# Learning rate scheduler callback
lr_scheduler = LearningRateScheduler(lr_schedule)

# Train our model
start_time = time.time()
history = model.fit(x_train_pca, y_train, epochs=25, batch_size=32, validation_data=(x_test_pca, y_test), callbacks=[lr_scheduler])
end_time = time.time()

print(f"Training time: {end_time - start_time:.2f} seconds")

# Print our models test accuracy
_, accuracy = model.evaluate(x_test_pca, y_test, verbose=0)
print(f"Model Accuracy: {accuracy*100:.2f}%")
```

B)Writers adjusments

The initial ChatGPT's model was lead to overfitting really fast and with a bad accuracy around 50%. In order to improve that we chose to tweak or add a lot of parameters which are shown below:

- Replaced **SGD** optimizer with **Adam**
- Added 256 neurons instead of initial 64 for more precise calculation and used **Leaky relu** instead of **relu** to prevent some neurons to remain 0
- In the output layer we chose to use **Softmax** activation function instead of **linear** which rised significantly our models accuracy
- Replaced the **initial learning rate** of 0.01 with a **variable one** that starts from 10^{-4} and reduced by 25% after 8th epoch while we also **reduced by half the epochs** from 50 to 25. In that way we avoided overfiting on the training process.

By making the above tweaks we eliminated overfiting and achieved a **test accuracy around 62.85%** in **19.9 seconds** of training time as shown below. This MPL model is better than all of the previous SVM models

```
Epoch 1/25
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.5116 - loss: 0.9881 - val_accuracy: 0.5735 - val_loss: 0.9364 - learning_rate: 1.0000e-04
Epoch 2/25
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.5798 - loss: 0.9221 - val_accuracy: 0.6005 - val_loss: 0.9098 - learning_rate: 1.0000e-04
Epoch 3/25
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.6132 - loss: 0.8982 - val_accuracy: 0.6060 - val_loss: 0.9016 - learning_rate: 1.0000e-04
Epoch 4/25
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.6316 - loss: 0.8811 - val_accuracy: 0.6100 - val_loss: 0.8974 - learning_rate: 1.0000e-04
Epoch 5/25
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.6303 - loss: 0.8955 - val_accuracy: 0.6115 - val_loss: 0.8935 - learning_rate: 1.0000e-04
Epoch 6/25
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.6460 - loss: 0.8752 - val_accuracy: 0.6145 - val_loss: 0.8909 - learning_rate: 1.0000e-04
Epoch 7/25
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.6570 - loss: 0.8626 - val_accuracy: 0.6175 - val_loss: 0.8886 - learning_rate: 1.0000e-04
Epoch 8/25
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.6682 - loss: 0.8571 - val_accuracy: 0.6175 - val_loss: 0.8877 - learning_rate: 1.0000e-04
Epoch 9/25
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.6633 - loss: 0.8443 - val_accuracy: 0.6190 - val_loss: 0.8870 - learning_rate: 1.0000e-04
Epoch 10/25
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.6718 - loss: 0.8556 - val_accuracy: 0.6195 - val_loss: 0.8858 - learning_rate: 7.5000e-05
Epoch 11/25
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.6757 - loss: 0.8409 - val_accuracy: 0.6180 - val_loss: 0.8858 - learning_rate: 7.5000e-05
Epoch 12/25
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.6699 - loss: 0.8484 - val_accuracy: 0.6175 - val_loss: 0.8846 - learning_rate: 7.5000e-05
Epoch 13/25
...
Epoch 25/25
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7284 - loss: 0.8017 - val_accuracy: 0.6295 - val_loss: 0.8752 - learning_rate: 7.5000e-05
Training time: 19.93 seconds
Model Accuracy: 62.95%
```

V. VS CNN AND KNN

In that section of the project we are comparing our SVM model outputs with the ones we executed in a previous project for Nearest Centroid Classifier (NCC) and K-Nearest Neighbors (KNN). We tweaked a bit the code to apply only for the two desired classes of our dataset (Cad and Dog) and not for all the classes. The results are shown below:

```
KNN (k=1) Accuracy: 0.5895
KNN (k=1) Execution Time: 0.0871 seconds
KNN (k=3) Accuracy: 59.6000%
KNN (k=3) Execution Time: 0.0543 seconds
NCC Accuracy: 57.9000%
NCC Execution Time: 0.0274 seconds
```

In conclusion as the accuracy for each model is 57.9% and 59.6% our SVM model remains better with an exchange of execution time