Base CNN Model. In this coding discussion, we will walk you through some basic preprocessing techniques that will help you achieve some performance boost.

This notebook has been created by Tonmoy.

(i) Importing the necessary packages

In [ ]:
```
import numpy as np
import pandas as pd
import keras
from keras.models import Sequential
from keras.layers import Dense, Activation, Flatten,Dropout
from keras.layers import Conv2D,BatchNormalization,MaxPooling2D,Reshape
from keras.utils import to_categorical
from keras_tuner import RandomSearch
import matplotlib.pyplot as plt
```

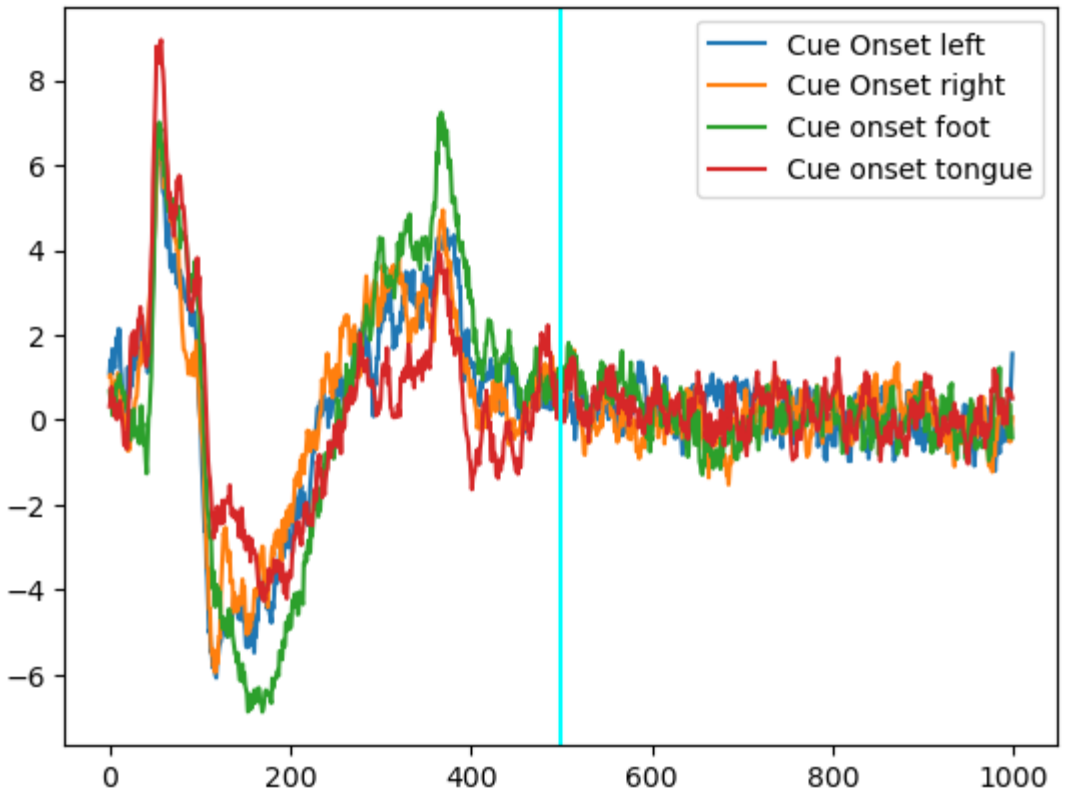
(ii)(a) Loading and visualizing the dataset

In [ ]:

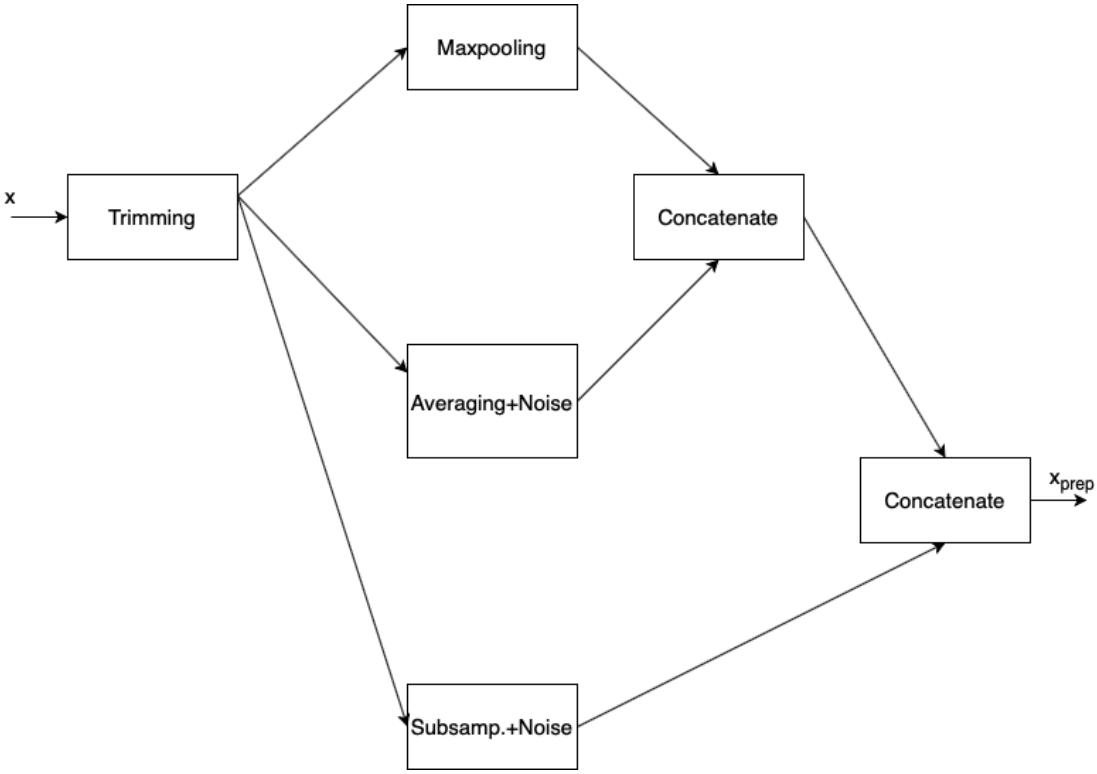
```
## Loading and visualizing the data

## Loading the dataset


X_test = np.load("X_test.npy")
y_test = np.load("y_test.npy")
person_train_valid = np.load("person_train_valid.npy")
X_train_valid = np.load("X_train_valid.npy")
y_train_valid = np.load("y_train_valid.npy")
person_test = np.load("person_test.npy")

## Adjusting the labels so that

# Cue onset left - 0
# Cue onset right - 1
# Cue onset foot - 2
# Cue onset tongue - 3

y_train_valid -= 769
y_test -= 769

## Visualizing the data

ch_data = X_train_valid[:,9,:] # extracts the 9th (out of 22) channel from t


class_0_ind = np.where(y_train_valid == 0) # finds the indices where the lab
ch_data_class_0 = ch_data[class_0_ind] # finds the data where label is 0
avg_ch_data_class_0 = np.mean(ch_data_class_0,axis=0) # finds the average re


class_1_ind = np.where(y_train_valid == 1)
ch_data_class_1 = ch_data[class_1_ind]
avg_ch_data_class_1 = np.mean(ch_data_class_1,axis=0)

class_2_ind = np.where(y_train_valid == 2)
ch_data_class_2 = ch_data[class_2_ind]
avg_ch_data_class_2 = np.mean(ch_data_class_2,axis=0)

class_3_ind = np.where(y_train_valid == 3)
ch_data_class_3 = ch_data[class_3_ind]
avg_ch_data_class_3 = np.mean(ch_data_class_3,axis=0)


plt.plot(np.arange(1000),avg_ch_data_class_0)
plt.plot(np.arange(1000),avg_ch_data_class_1)
plt.plot(np.arange(1000),avg_ch_data_class_2)
plt.plot(np.arange(1000),avg_ch_data_class_3)
plt.axvline(x=500, label='line at t=500',c='cyan')

plt.legend(["Cue Onset left", "Cue Onset right", "Cue onset foot", "Cue onse
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x7feb781e52e0>
```



(ii) (b) Preprocessing the dataset



In [ ]:
```
def data_prep(X,y,sub_sample,average,noise):

    total_X = None
    total_y = None

    # Trimming the data (sample,22,1000) -> (sample,22,500)
    X = X[:,:,0:500]
    print('Shape of X after trimming:',X.shape)

    # Maxpooling the data (sample,22,1000) -> (sample,22,500/sub_sample)
    X_max = np.max(X.reshape(X.shape[0], X.shape[1], -1, sub_sample), axis=3


    total_X = X_max
    total_y = y
    print('Shape of X after maxpooling:',total_X.shape)


    #data augmentation

    # Averaging + noise
    X_average = np.mean(X.reshape(X.shape[0], X.shape[1], -1, average),axis=
    X_average = X_average + np.random.normal(0.0, 0.5, X_average.shape)

    total_X = np.vstack((total_X, X_average))
    total_y = np.hstack((total_y, y))
    print('Shape of X after averaging+noise and concatenating:',total_X.shap

    # Subsampling

    for i in range(sub_sample):

        X_subsample = X[:, :, i::sub_sample] + \
                            (np.random.normal(0.0, 0.5, X[:, :,i::sub_sample

        total_X = np.vstack((total_X, X_subsample))
        total_y = np.hstack((total_y, y))


    print('Shape of X after subsampling and concatenating:',total_X.shape)
    return total_X,total_y


X_train_valid_prep,y_train_valid_prep = data_prep(X_train_valid,y_train_vali
```

```
Shape of X after trimming: (2115, 22, 500)
Shape of X after maxpooling: (2115, 22, 250)
Shape of X after averaging+noise and concatenating: (4230, 22, 250)
Shape of X after subsampling and concatenating: (8460, 22, 250)
```

(ii)(c) Preparing the training, validation, and test datasets

In [ ]:

```
## Random splitting and reshaping the data
# First generating the training and validation indices using random splittir

ind_valid = np.random.choice(2115, 375, replace=False)
ind_train = np.array(list(set(range(2115)).difference(set(ind_valid))))

# Creating the training and validation sets using the generated indices
(X_train, X_valid) = X_train_valid[ind_train], X_train_valid[ind_valid]
(y_train, y_valid) = y_train_valid[ind_train], y_train_valid[ind_valid]


## Preprocessing the dataset
print("training data")
x_train,y_train = data_prep(X_train,y_train,2,2,True)
print("\nvalidation data")
x_valid,y_valid = data_prep(X_valid,y_valid,2,2,True)
print("\ntest data")
X_test_prep,y_test_prep = data_prep(X_test,y_test,2,2,True)


print('Shape of testing set:',X_test_prep.shape)
print('Shape of testing labels:',y_test_prep.shape)

print('Shape of training set:',x_train.shape)
print('Shape of validation set:',x_valid.shape)
print('Shape of training labels:',y_train.shape)
print('Shape of validation labels:',y_valid.shape)


# Converting the labels to categorical variables for multiclass classificati
y_train = to_categorical(y_train, 4)
y_valid = to_categorical(y_valid, 4)
y_test = to_categorical(y_test_prep, 4)
print('Shape of training labels after categorical conversion:',y_train.shape
print('Shape of validation labels after categorical conversion:',y_valid.sha
print('Shape of test labels after categorical conversion:',y_test.shape)

# Adding width of the segment to be 1
x_train = x_train.reshape(x_train.shape[0], x_train.shape[1], x_train.shape[
x_valid = x_valid.reshape(x_valid.shape[0], x_valid.shape[1], x_train.shape[
x_test = X_test_prep.reshape(X_test_prep.shape[0], X_test_prep.shape[1], X_t
print('Shape of training set after adding width info:',x_train.shape)
print('Shape of validation set after adding width info:',x_valid.shape)
print('Shape of test set after adding width info:',x_test.shape)


# Reshaping the training and validation dataset
x_train = np.swapaxes(x_train, 1,3)
x_train = np.swapaxes(x_train, 1,2)
x_valid = np.swapaxes(x_valid, 1,3)
x_valid = np.swapaxes(x_valid, 1,2)
x_test = np.swapaxes(x_test, 1,3)
x_test = np.swapaxes(x_test, 1,2)
print('Shape of training set after dimension reshaping:',x_train.shape)
print('Shape of validation set after dimension reshaping:',x_valid.shape)
print('Shape of test set after dimension reshaping:',x_test.shape)
```

```
training data
Shape of X after trimming: (1740, 22, 500)
Shape of X after maxpooling: (1740, 22, 250)
Shape of X after averaging+noise and concatenating: (3480, 22, 250)
Shape of X after subsampling and concatenating: (6960, 22, 250)

validation data
Shape of X after trimming: (375, 22, 500)
Shape of X after maxpooling: (375, 22, 250)
Shape of X after averaging+noise and concatenating: (750, 22, 250)
Shape of X after subsampling and concatenating: (1500, 22, 250)

test data
Shape of X after trimming: (443, 22, 500)
Shape of X after maxpooling: (443, 22, 250)
Shape of X after averaging+noise and concatenating: (886, 22, 250)
Shape of X after subsampling and concatenating: (1772, 22, 250)
Shape of testing set: (1772, 22, 250)
Shape of testing labels: (1772,)
Shape of training set: (6960, 22, 250)
Shape of validation set: (1500, 22, 250)
Shape of training labels: (6960,)
Shape of validation labels: (1500,)
Shape of training labels after categorical conversion: (6960, 4)
Shape of validation labels after categorical conversion: (1500, 4)
Shape of test labels after categorical conversion: (1772, 4)
Shape of training set after adding width info: (6960, 22, 250, 1)
Shape of validation set after adding width info: (1500, 22, 250, 1)
Shape of test set after adding width info: (1772, 22, 250, 1)
Shape of training set after dimension reshaping: (6960, 250, 1, 22)
Shape of validation set after dimension reshaping: (1500, 250, 1, 22)
Shape of test set after dimension reshaping: (1772, 250, 1, 22)
```

(iii)(CNN) Defining the architecture of a basic CNN model

In [ ]:

```
def build_model(params):
    # Building the CNN model using sequential class
    basic_cnn_model = Sequential()

    #tune number of filters
    filter1 = params.Int('filter1', min_value =15, max_value = 30, step = 5)
    filter2 = params.Int('filter2', min_value =40, max_value = 60, step = 5)
    filter3 = params.Int('filter3', min_value =80, max_value = 120, step = 1
    filter4 = params.Int('filter4', min_value =180, max_value = 220, step =



    #tune kernel size
    kernel = params.Choice('kernel_size', values=[3,5,10])
    #tune pool size
    pool = params.Choice('pool_size', values=[2,3,5])
    # tune dropout
    dropout = 0.5
    # 0.5 is optimal for dropout

    # Conv. block 1
    basic_cnn_model.add(Conv2D(filters=filter1, kernel_size=(kernel,1), padd
    basic_cnn_model.add(MaxPooling2D(pool_size=(pool,1), padding='same')) #
    basic_cnn_model.add(BatchNormalization())
    basic_cnn_model.add(Dropout(dropout))

    # Conv. block 2
    basic_cnn_model.add(Conv2D(filters=filter2, kernel_size=(kernel,1), padd
    basic_cnn_model.add(MaxPooling2D(pool_size=(pool,1), padding='same'))
    basic_cnn_model.add(BatchNormalization())
    basic_cnn_model.add(Dropout(dropout))

    # Conv. block 3
    basic_cnn_model.add(Conv2D(filters=filter3, kernel_size=(kernel,1), padd
    basic_cnn_model.add(MaxPooling2D(pool_size=(pool,1), padding='same'))
    basic_cnn_model.add(BatchNormalization())
    basic_cnn_model.add(Dropout(dropout))

    # Conv. block 4
    basic_cnn_model.add(Conv2D(filters=filter4, kernel_size=(kernel,1), padd
    basic_cnn_model.add(MaxPooling2D(pool_size=(pool,1), padding='same'))
    basic_cnn_model.add(BatchNormalization())
    basic_cnn_model.add(Dropout(dropout))

    # Output layer with Softmax activation
    basic_cnn_model.add(Flatten()) # Flattens the input
    basic_cnn_model.add(Dense(4, activation='softmax')) # Output FC layer wi


    # Printing the model summary
    # basic_cnn_model.summary()

    learning_rate = 1e-3
    # epochs = 50
    cnn_optimizer = keras.optimizers.Adam(lr=learning_rate)

    # Compiling the model
    basic_cnn_model.compile(loss='categorical_crossentropy',
                            optimizer=cnn_optimizer
```

```
                    optimizer=cnn_optimizer,
                    metrics=['accuracy'])

    return basic_cnn_model
```

(iv) (CNN) Defining the hyperparameters of the basic CNN model

In [ ]:
```
# Model parameters
# learning_rate = 1e-3
# epochs = 50
# cnn_optimizer = keras.optimizers.Adam(lr=learning_rate)

#early stopping
callback = keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
tuner = RandomSearch(build_model,
                     objective='val_accuracy',
                     max_trials = 20,
                     overwrite=True)
tuner.search(x_train,y_train,epochs=80,validation_data=(x_valid,y_valid), ca
```

```
Trial 20 Complete [00h 02m 23s]
val_accuracy: 0.7133333086967468

Best val_accuracy So Far: 0.7480000257492065
Total elapsed time: 00h 34m 31s
INFO:tensorflow:Oracle triggered exit
```

(v) (CNN) Compiling, training and validating the model

In [ ]:
```
#train with best hyperparameters
best_hps=tuner.get_best_hyperparameters(num_trials=1)[0]
model = tuner.hypermodel.build(best_hps)
history = model.fit(x_train, y_train, epochs=50,batch_size=64, validation_da

val_acc_per_epoch = history.history['val_accuracy']
best_epoch = val_acc_per_epoch.index(max(val_acc_per_epoch)) + 1
print('Best epoch: %d' % (best_epoch,))
```

```
Epoch 1/50
109/109 [==============================] - 4s 34ms/step - loss: 1.9290 - ac
curacy: 0.3203 - val_loss: 1.3925 - val_accuracy: 0.3973
Epoch 2/50
109/109 [==============================] - 4s 33ms/step - loss: 1.5287 - ac
curacy: 0.3807 - val_loss: 1.2395 - val_accuracy: 0.4427
Epoch 3/50
109/109 [==============================] - 4s 32ms/step - loss: 1.3329 - ac
curacy: 0.4353 - val_loss: 1.1689 - val_accuracy: 0.4893
Epoch 4/50
109/109 [==============================] - 4s 32ms/step - loss: 1.2191 - ac
curacy: 0.4892 - val_loss: 1.1516 - val_accuracy: 0.4940
Epoch 5/50
109/109 [==============================] - 4s 33ms/step - loss: 1.1397 - ac
curacy: 0.5165 - val_loss: 1.1252 - val_accuracy: 0.5133
Epoch 6/50
109/109 [==============================] - 4s 33ms/step - loss: 1.0886 - ac
curacy: 0.5398 - val_loss: 1.0996 - val_accuracy: 0.5047
Epoch 7/50
109/109 [==============================] - 4s 33ms/step - loss: 1.0479 - ac
curacy: 0.5737 - val_loss: 1.0523 - val_accuracy: 0.5540
Epoch 8/50
109/109 [==============================] - 4s 34ms/step - loss: 0.9987 - ac
curacy: 0.5822 - val_loss: 1.0802 - val_accuracy: 0.5287
Epoch 9/50
109/109 [==============================] - 4s 32ms/step - loss: 0.9690 - ac
curacy: 0.6037 - val_loss: 0.9931 - val_accuracy: 0.5807
Epoch 10/50
109/109 [==============================] - 4s 32ms/step - loss: 0.9158 - ac
curacy: 0.6313 - val_loss: 0.9705 - val_accuracy: 0.5873
Epoch 11/50
109/109 [==============================] - 4s 32ms/step - loss: 0.8929 - ac
curacy: 0.6408 - val_loss: 0.9327 - val_accuracy: 0.6253
Epoch 12/50
109/109 [==============================] - 4s 32ms/step - loss: 0.8539 - ac
curacy: 0.6566 - val_loss: 0.8794 - val_accuracy: 0.6593
Epoch 13/50
109/109 [==============================] - 4s 33ms/step - loss: 0.8165 - ac
curacy: 0.6733 - val_loss: 0.8591 - val_accuracy: 0.6420
Epoch 14/50
109/109 [==============================] - 4s 33ms/step - loss: 0.7896 - ac
curacy: 0.6846 - val_loss: 0.8255 - val_accuracy: 0.6600
Epoch 15/50
109/109 [==============================] - 4s 34ms/step - loss: 0.7511 - ac
curacy: 0.7010 - val_loss: 0.8646 - val_accuracy: 0.6360
Epoch 16/50
109/109 [==============================] - 4s 35ms/step - loss: 0.7357 - ac
curacy: 0.7050 - val_loss: 0.8328 - val_accuracy: 0.6740
Epoch 17/50
109/109 [==============================] - 4s 33ms/step - loss: 0.7049 - ac
curacy: 0.7231 - val_loss: 0.7896 - val_accuracy: 0.6827
Epoch 18/50
109/109 [==============================] - 4s 32ms/step - loss: 0.6826 - ac
curacy: 0.7320 - val_loss: 0.8439 - val_accuracy: 0.6893
Epoch 19/50
109/109 [==============================] - 4s 33ms/step - loss: 0.6572 - ac
curacy: 0.7420 - val_loss: 0.8132 - val_accuracy: 0.6780
Epoch 20/50
109/109 [==============================] - 4s 33ms/step - loss: 0.6337 - ac
```

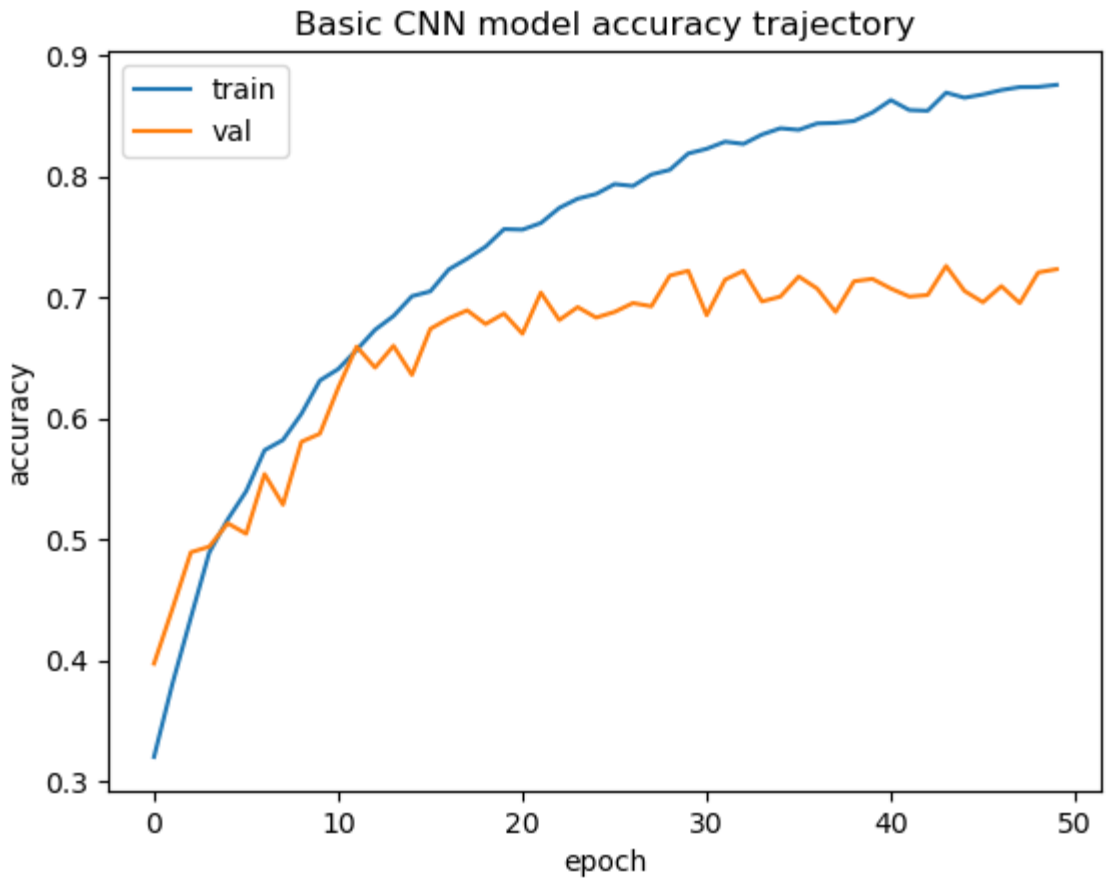
```
curacy: 0.7565 - val_loss: 0.8039 - val_accuracy: 0.6867
Epoch 21/50
109/109 [==============================] - 4s 34ms/step - loss: 0.6274 - ac
curacy: 0.7560 - val_loss: 0.8501 - val_accuracy: 0.6700
Epoch 22/50
109/109 [==============================] - 4s 33ms/step - loss: 0.5989 - ac
curacy: 0.7615 - val_loss: 0.7739 - val_accuracy: 0.7040
Epoch 23/50
109/109 [==============================] - 4s 32ms/step - loss: 0.5868 - ac
curacy: 0.7740 - val_loss: 0.8355 - val_accuracy: 0.6813
Epoch 24/50
109/109 [==============================] - 3s 31ms/step - loss: 0.5583 - ac
curacy: 0.7816 - val_loss: 0.7987 - val_accuracy: 0.6920
Epoch 25/50
109/109 [==============================] - 4s 33ms/step - loss: 0.5590 - ac
curacy: 0.7855 - val_loss: 0.8597 - val_accuracy: 0.6833
Epoch 26/50
109/109 [==============================] - 4s 37ms/step - loss: 0.5476 - ac
curacy: 0.7937 - val_loss: 0.8374 - val_accuracy: 0.6880
Epoch 27/50
109/109 [==============================] - 4s 34ms/step - loss: 0.5341 - ac
curacy: 0.7921 - val_loss: 0.8188 - val_accuracy: 0.6953
Epoch 28/50
109/109 [==============================] - 3s 31ms/step - loss: 0.5117 - ac
curacy: 0.8016 - val_loss: 0.8186 - val_accuracy: 0.6927
Epoch 29/50
109/109 [==============================] - 4s 33ms/step - loss: 0.5068 - ac
curacy: 0.8055 - val_loss: 0.7627 - val_accuracy: 0.7180
Epoch 30/50
109/109 [==============================] - 4s 35ms/step - loss: 0.4664 - ac
curacy: 0.8190 - val_loss: 0.7886 - val_accuracy: 0.7220
Epoch 31/50
109/109 [==============================] - 4s 33ms/step - loss: 0.4611 - ac
curacy: 0.8228 - val_loss: 0.8247 - val_accuracy: 0.6853
Epoch 32/50
109/109 [==============================] - 4s 33ms/step - loss: 0.4540 - ac
curacy: 0.8287 - val_loss: 0.8074 - val_accuracy: 0.7147
Epoch 33/50
109/109 [==============================] - 4s 33ms/step - loss: 0.4533 - ac
curacy: 0.8269 - val_loss: 0.8097 - val_accuracy: 0.7220
Epoch 34/50
109/109 [==============================] - 4s 33ms/step - loss: 0.4337 - ac
curacy: 0.8346 - val_loss: 0.8402 - val_accuracy: 0.6967
Epoch 35/50
109/109 [==============================] - 4s 32ms/step - loss: 0.4223 - ac
curacy: 0.8398 - val_loss: 0.8190 - val_accuracy: 0.7007
Epoch 36/50
109/109 [==============================] - 4s 33ms/step - loss: 0.4064 - ac
curacy: 0.8385 - val_loss: 0.8193 - val_accuracy: 0.7173
Epoch 37/50
109/109 [==============================] - 4s 33ms/step - loss: 0.4194 - ac
curacy: 0.8438 - val_loss: 0.8512 - val_accuracy: 0.7073
Epoch 38/50
109/109 [==============================] - 4s 32ms/step - loss: 0.4064 - ac
curacy: 0.8443 - val_loss: 0.8155 - val_accuracy: 0.6880
Epoch 39/50
109/109 [==============================] - 4s 32ms/step - loss: 0.3954 - ac
curacy: 0.8458 - val_loss: 0.8125 - val_accuracy: 0.7133
Epoch 40/50
```

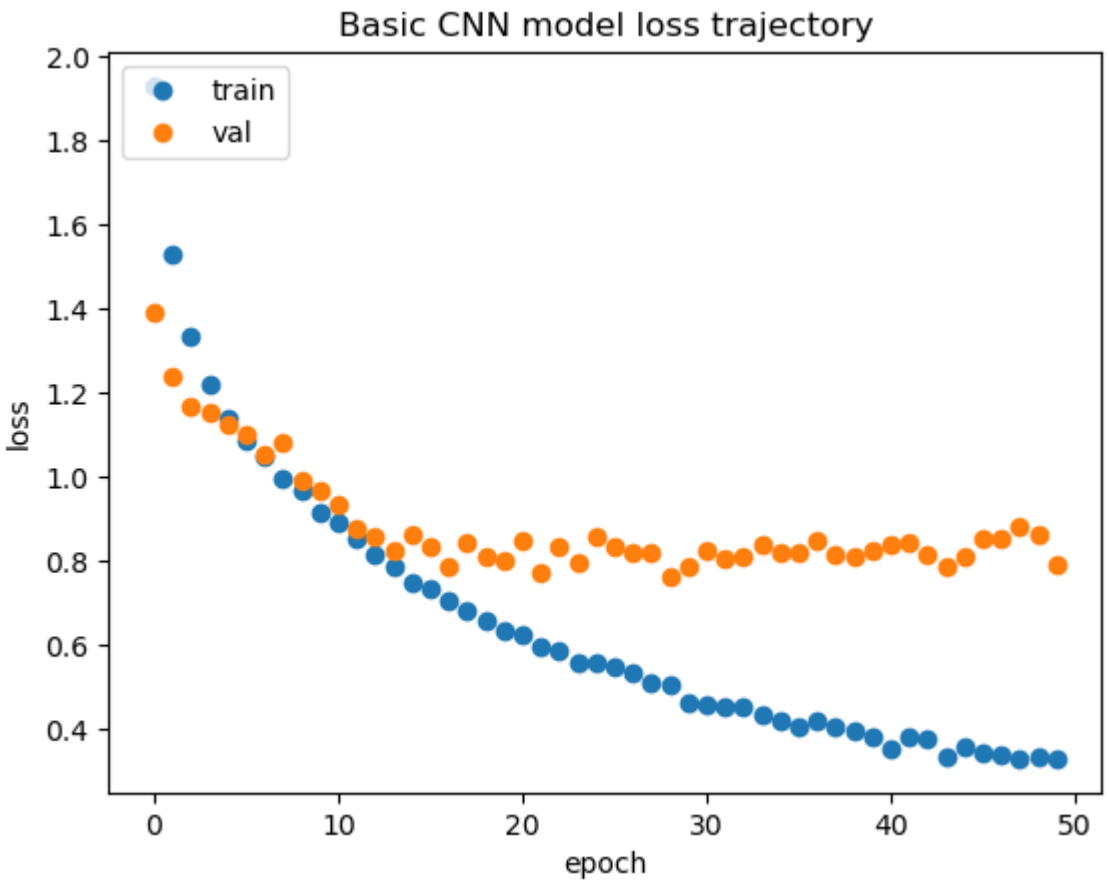
```
109/109 [==============================] - 4s 32ms/step - loss: 0.3815 - ac
curacy: 0.8529 - val_loss: 0.8274 - val_accuracy: 0.7153
Epoch 41/50
109/109 [==============================] - 4s 33ms/step - loss: 0.3567 - ac
curacy: 0.8631 - val_loss: 0.8399 - val_accuracy: 0.7073
Epoch 42/50
109/109 [==============================] - 4s 33ms/step - loss: 0.3832 - ac
curacy: 0.8549 - val_loss: 0.8423 - val_accuracy: 0.7007
Epoch 43/50
109/109 [==============================] - 4s 33ms/step - loss: 0.3788 - ac
curacy: 0.8543 - val_loss: 0.8171 - val_accuracy: 0.7020
Epoch 44/50
109/109 [==============================] - 4s 33ms/step - loss: 0.3375 - ac
curacy: 0.8693 - val_loss: 0.7868 - val_accuracy: 0.7260
Epoch 45/50
109/109 [==============================] - 4s 32ms/step - loss: 0.3601 - ac
curacy: 0.8651 - val_loss: 0.8134 - val_accuracy: 0.7053
Epoch 46/50
109/109 [==============================] - 4s 32ms/step - loss: 0.3461 - ac
curacy: 0.8678 - val_loss: 0.8552 - val_accuracy: 0.6960
Epoch 47/50
109/109 [==============================] - 4s 33ms/step - loss: 0.3419 - ac
curacy: 0.8714 - val_loss: 0.8548 - val_accuracy: 0.7093
Epoch 48/50
109/109 [==============================] - 3s 31ms/step - loss: 0.3327 - ac
curacy: 0.8739 - val_loss: 0.8808 - val_accuracy: 0.6953
Epoch 49/50
109/109 [==============================] - 4s 33ms/step - loss: 0.3358 - ac
curacy: 0.8740 - val_loss: 0.8625 - val_accuracy: 0.7207
Epoch 50/50
109/109 [==============================] - 3s 32ms/step - loss: 0.3314 - ac
curacy: 0.8757 - val_loss: 0.7923 - val_accuracy: 0.7233
Best epoch: 44
```

(vi) (CNN) Visualizing the accuracy and loss trajectory

In [ ]:

```python
import matplotlib.pyplot as plt

# Plotting accuracy trajectory
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Basic CNN model accuracy trajectory')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

# Plotting loss trajectory
plt.plot(history.history['loss'],'o')
plt.plot(history.history['val_loss'],'o')
plt.title('Basic CNN model loss trajectory')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```


Basic CNN model accuracy trajectory
train
val
accuracy
epoch
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
10
20
30
40
50


Basic CNN model loss trajectory
train
val
loss
epoch
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0
10
20
30
40
50

(vii)(CNN) Testing the performance of the basic CNN model on the held out test set

In [ ]:
```
## Testing the basic CNN model

cnn_score = model.evaluate(x_test, y_test, verbose=0)
print('Test accuracy of the basic CNN model:',cnn_score[1])
print(best_hps.values)
```

```
Test accuracy of the basic CNN model: 0.7009029388427734
{'filter1': 30, 'filter2': 55, 'filter3': 80, 'filter4': 180, 'kernel_size
': 10, 'pool_size': 3}
```