COMP4318/5318 Assignment 2: Image Classification

Group number: 112 , SID1: 510612241 , SID2: 500497135, SID3: N/A

This template notebook includes code to load the dataset and a skeleton for the main sections that should be included in the notebook. Please stick to this struture for your submitted notebook.

Please focus on making your code clear, with appropriate variable names and whitespace. Include comments and markdown text to aid the readability of your code where relevant. See the specification and marking criteria in the associated specification to guide you when completing your implementation.

Setup and dependencies

Please use this section to list and set up all your required libraries/dependencies and your plotting environment.

```
import numpy as np

import tensorflow as tf
from tensorflow import keras
import pandas as pd
from typing import List, Tuple
import os
import sklearn as sk
import seaborn as sns
import pandas as pd
from sklearn.model_selection import train_test_split

from sklearn.svm import SVC
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten
from sklearn.model_selection import GridSearchCV
from tensorflow.keras.optimizers import Adam
from scikeras.wrappers import KerasClassifier, KerasRegressor

import matplotlib.pyplot as plt
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score
from PIL import Image


keras.backend.clear_session()
tf.random.set_seed(5)


!pip install scikeras
```

1. Data loading, exploration, and preprocessing

Code to load the dataset is provided in the following cell. Please proceed with your data exploration and preprocessing in the remainder of this section.

```
# Load the dataset training and test sets as numpy arrays
# assuming Assignment2Data folder is present in the same directory
# as the notebook
X_train = np.load('Assignment2Data/X_train.npy')
y_train = np.load('Assignment2Data/y_train.npy')
X_test = np.load('Assignment2Data/X_test.npy')
y_test = np.load('Assignment2Data/y_test.npy')
```

```
# This script displays a sample image for each unique label in our training dataset by:

visualize_images(X_train)
```

one set of images looks way larger, maybe some balancing?

```
def plot_distribution(X_train, y_train, y_test, cols=4):
    unique_labels = np.unique(y_train)
    n_labels = len(unique_labels)
    rows = (n_labels + cols - 1) // cols  # Calculate the number of rows needed

    plt.figure(figsize=(20, 5 * rows))

    for i, label in enumerate(unique_labels):
        index = np.where(y_train == label)[0][0]
        image = X_train[index]


        plt.subplot(rows, cols, i + 1)
        plt.imshow(image)
        plt.title(f'Label: {label}', fontsize=24)
        plt.axis('off')

        train_examples = np.where(y_train == label)[0].size
        test_examples = np.where(y_test == label)[0].size

        print(f"Number of examples for class {label}: training={train_examples}, test={test_examples}")

    plt.tight_layout()
    plt.show()


plot_distribution(X_train, y_train, y_test, cols=4)
```

```
# Print the size of the training and test data
print("X_train:",X_train.shape)
print("y_train:",y_train.shape)
print("X_test:",X_test.shape)
print("y_test:",y_test.shape)
################
print(y_train[0:30])
print("number of classes",len(np.unique(y_train)))
print("number of classes",(np.unique(y_train).dtype))
print("unique classes", np.unique(y_train))
#print(X_train[0:1])
```

```
X_train: (18928, 28, 28)
y_train: (18928,)
X_test: (4732, 28, 28)
y_test: (4732,)
[10  7  6  3  2  9 10  9  8  5  6  9  0  4  3  6  9  9  1  9  2  1  6  9
  6  2  1  1  8 10]
number of classes 11
number of classes uint8
unique classes [ 0  1  2  3  4  5  6  7  8  9 10]
```

```
plot_histogram(y_train, y_test, title_suffix='for Organ dataset')
```

```
plot_histogram_fancy(X_train, y_train)
```

Examples of preprocessed data

Please print/display some examples of your preprocessed data here.

Preprocess of Random Forest

```
def random_forest_preprocess(X_train, Y_train, X_test, Y_test):


#don't actually need to scale
    # X_train = X_train.astype('float32') / 255.0
    # X_test = X_test.astype('float32') / 255.0

    X_train = X_train.reshape(X_train.shape[0], -1)
    X_test = X_test.reshape(X_test.shape[0], -1)

    return X_train, Y_train, X_test, Y_test
```

Preprocess of fully connected neural network

```
from sklearn.preprocessing import MinMaxScaler

def fully_conn_preprocess(X_train, Y_train, X_test, Y_test):
    X_train = X_train.reshape(X_train.shape[0], -1)
    X_test = X_test.reshape(X_test.shape[0], -1)

    # Scaling over here
    scaler = MinMaxScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # one-hot for all our 11 unique classes
    Y_train = tf.keras.utils.to_categorical(Y_train, 11)
    Y_test = tf.keras.utils.to_categorical(Y_test,11)

    return X_train, Y_train, X_test, Y_test
```

Preprocess of Conv neural network

```
def convolutional_preprocess(X_train, y_train, X_test, y_test, ):

    X_train = X_train.astype('float32') / 255.0
    X_test = X_test.astype('float32') / 255.0

    # One-hot
    y_train = tf.keras.utils.to_categorical(y_train, 11)
    y_test = tf.keras.utils.to_categorical(y_test, 11)

    # Split tr
    X_train, X_valid, y_train, y_valid = train_test_split(
        X_train, y_train, train_size=0.7)


    X_train = np.expand_dims(X_train, -1)
    X_valid = np.expand_dims(X_valid, -1)
    X_test = np.expand_dims(X_test, -1)

    return X_train, y_train, X_valid, y_valid, X_test, y_test
```

2. Algorithm design and setup

Random Forest

```
from sklearn.ensemble import RandomForestClassifier

def create_random_forest_classifier(n_estimators=100, max_depth=50, random_state=0, criterion='entropy'):
    model = RandomForestClassifier(n_estimators=n_estimators, max_depth=max_depth, random_state=random_state, criterion=cri
    return model
```

```
X_train_rf, y_train_rf,X_test_rf,y_test_rf = random_forest_preprocess(X_train,y_train,X_test,y_test)

model= create_random_forest_classifier()
RF_history = model.fit(X_train_rf, y_train_rf)


y_pred = model.predict(X_test_rf) #name got changed but previous upload should show correct result
accuracy = accuracy_score(y_test_rf, y_pred)

print(f"Test Accuracy: {accuracy:.4f}")
```

Fully connected neural network

```
#this works better than the other tuning version we will use later
import tensorflow as tf
from tensorflow.keras.optimizers import Adam

def fully_conn_network(optimizer='SGD', learning_rate=0.05):
    model = keras.models.Sequential([
    keras.layers.Input(shape=(784,)),
    keras.layers.Flatten(),
    keras.layers.Dense(300, activation="tanh"),
    keras.layers.Dense(100, activation="tanh"),
    keras.layers.Dense(11, activation="softmax")
  ])



   # Select optimizer based on input argument
    if optimizer == 'SGD':
        opt = tf.keras.optimizers.SGD(learning_rate=learning_rate)
    elif optimizer == 'Adam':
        opt = tf.keras.optimizers.Adam(learning_rate=learning_rate)
    elif optimizer == 'RMSprop':
        opt = tf.keras.optimizers.RMSprop(learning_rate=learning_rate)
    else:
        raise ValueError("Unsupported optimizer")

    model.compile(loss='categorical_crossentropy',  # Adjust loss type if needed based on your label format
                  optimizer=opt,
                  metrics=['accuracy'])

    return model
```

```
#adapted from the notebook but at the moment does not give a decent accuracy
def tunable_fully_network(n_hidden_layers=2, n_hidden_neurons=50,
    activation_function="relu",input_shape=(784,), learning_rate=0.01):


    model = keras.models.Sequential()



    model.add(keras.layers.Input(shape=input_shape)),
    model.add(keras.layers.Flatten())

    # Add the hidden layers with desired size and activation function
    for layer in range(n_hidden_layers):
        model.add(keras.layers.Dense(n_hidden_neurons, activation=activation_function))


    model.add(keras.layers.Dense(11, activation="softmax"))

    optimizer = keras.optimizers.Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

    return model


X_train_mlp, y_train_mlp, X_valid_mlp, y_valid_mlp = fully_conn_preprocess(X_train,y_train,X_test,y_test)

model = fully_conn_network(optimizer='Adam', learning_rate=0.001) #gives highest accuracy at the moment

MLP_history = model.fit(X_train_mlp, y_train_mlp, epochs=20, batch_size=24, validation_split=0.2)
# model = build_mlp(n_hidden_layers=2, n_hidden_neurons=50,activation_function="relu", input_shape=(784,), learning_rate=0.
# MLP_history = model.fit(X_train_mlp, y_train_mlp, epochs=20, batch_size=32, validation_split=0.2)
loss, accuracy = model.evaluate(X_valid_mlp, y_valid_mlp)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")
plot_training_acc(MLP_history)
```

Convolutional neural network

```
#normal CNN or something fancier to use?
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization, Activation
from tensorflow.keras.optimizers import Adam, SGD, RMSprop
def convolutional_network(input_shape, num_classes=11, optimizer='Adam', learning_rate=0.001, loss='categorical_crossentrop


    model = Sequential()

    num_filters = [32, 64, 128]
    kernel_size = (3,3)
    pool_size = (2,2)
    dropout_rate = 0.25

    for i, filters in enumerate(num_filters):

        if i == 0:
            model.add(Conv2D(filters, kernel_size, padding='same', input_shape=input_shape))
        else:
            model.add(Conv2D(filters, kernel_size, padding='same'))

        model.add(BatchNormalization())
        model.add(Activation('relu'))
        model.add(MaxPooling2D(pool_size))
        model.add(Dropout(dropout_rate))

    # Final convolution layer before flattening
    model.add(Conv2D(256, (3,3), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2,2)))
    model.add(Dropout(dropout_rate))

    model.add(Flatten())
    model.add(Dense(num_classes, activation='softmax'))  # Output layer for classification

    # Compile the model with the specified optimizer, learning rate, and loss function
    if optimizer.lower() == 'adam':
        opt = Adam(learning_rate=learning_rate)
    elif optimizer.lower() == 'sgd':
        opt = SGD(learning_rate=learning_rate)
    elif optimizer.lower() == 'rmsprop':
        opt = RMSprop(learning_rate=learning_rate)
    else:
         raise ValueError("Unsupported optimizer. Supported: 'adam', 'sgd', 'rmsprop'.")

    model.compile(optimizer=opt, loss=loss, metrics=['accuracy'])

    return model


model = convolutional_network(input_shape=[28,28,1], num_classes=11, optimizer='Adam', learning_rate=0.001, loss='categoric
X_train_CNN, y_train_CNN,X_valid_CNN,y_valid_CNN, X_test_CNN,y_test_CNN= convolutional_preprocess(X_train,y_train,X_test,y_

cnn_history = model.fit(X_train_CNN, y_train_CNN, epochs=20, batch_size=32, validation_split=0.2)
loss, accuracy = model.evaluate(X_valid_CNN, y_valid_CNN)
print(f"Validation Loss: {loss:.4f}")
print(f"Validation Accuracy: {accuracy:.4f}")

# plot_accuracy(cnn_history)
# plot_loss(cnn_history)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `in
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/20
332/332 ━━━━━━━━━━━━━━━━━━━━ 43s 116ms/step - accuracy: 0.6174 - loss: 1.2180 - val_accuracy: 0.1521 - val_loss: 3.8652
Epoch 2/20
332/332 ━━━━━━━━━━━━━━━━━━━━ 42s 118ms/step - accuracy: 0.8504 - loss: 0.4404 - val_accuracy: 0.8381 - val_loss: 0.4591
Epoch 3/20
332/332 ━━━━━━━━━━━━━━━━━━━━ 36s 105ms/step - accuracy: 0.8852 - loss: 0.3280 - val_accuracy: 0.9034 - val_loss: 0.2845
Epoch 4/20
332/332 ━━━━━━━━━━━━━━━━━━━━ 42s 108ms/step - accuracy: 0.9030 - loss: 0.2681 - val_accuracy: 0.9132 - val_loss: 0.2652
Epoch 5/20
332/332 ━━━━━━━━━━━━━━━━━━━━ 35s 105ms/step - accuracy: 0.9226 - loss: 0.2302 - val_accuracy: 0.9351 - val_loss: 0.2016
Epoch 6/20
332/332 ━━━━━━━━━━━━━━━━━━━━ 35s 104ms/step - accuracy: 0.9261 - loss: 0.2093 - val_accuracy: 0.9200 - val_loss: 0.2415
Epoch 7/20
332/332 ━━━━━━━━━━━━━━━━━━━━ 34s 103ms/step - accuracy: 0.9375 - loss: 0.1798 - val_accuracy: 0.9370 - val_loss: 0.1921
Epoch 8/20
332/332 ━━━━━━━━━━━━━━━━━━━━ 35s 104ms/step - accuracy: 0.9382 - loss: 0.1752 - val_accuracy: 0.9200 - val_loss: 0.2493
Epoch 9/20
332/332 ━━━━━━━━━━━━━━━━━━━━ 41s 103ms/step - accuracy: 0.9451 - loss: 0.1583 - val_accuracy: 0.9389 - val_loss: 0.1881
Epoch 10/20
332/332 ━━━━━━━━━━━━━━━━━━━━ 34s 101ms/step - accuracy: 0.9479 - loss: 0.1478 - val_accuracy: 0.8792 - val_loss: 0.3493
Epoch 11/20
332/332 ━━━━━━━━━━━━━━━━━━━━ 41s 100ms/step - accuracy: 0.9525 - loss: 0.1374 - val_accuracy: 0.9351 - val_loss: 0.1886
```

```
Epoch 12/20
332/332 ━━━━━━━━━━━━━━━━━━━━ 42s 104ms/step - accuracy: 0.9597 - loss: 0.1233 - val_accuracy: 0.9230 - val_loss: 0.2421
Epoch 13/20
332/332 ━━━━━━━━━━━━━━━━━━━━ 35s 105ms/step - accuracy: 0.9587 - loss: 0.1158 - val_accuracy: 0.9377 - val_loss: 0.1955
Epoch 14/20
332/332 ━━━━━━━━━━━━━━━━━━━━ 41s 104ms/step - accuracy: 0.9664 - loss: 0.1010 - val_accuracy: 0.9125 - val_loss: 0.2987
Epoch 15/20
332/332 ━━━━━━━━━━━━━━━━━━━━ 33s 100ms/step - accuracy: 0.9632 - loss: 0.1056 - val_accuracy: 0.9426 - val_loss: 0.1623
Epoch 16/20
332/332 ━━━━━━━━━━━━━━━━━━━━ 41s 100ms/step - accuracy: 0.9638 - loss: 0.1031 - val_accuracy: 0.9589 - val_loss: 0.1213
Epoch 17/20
332/332 ━━━━━━━━━━━━━━━━━━━━ 41s 100ms/step - accuracy: 0.9692 - loss: 0.0988 - val_accuracy: 0.9562 - val_loss: 0.1282
Epoch 18/20
332/332 ━━━━━━━━━━━━━━━━━━━━ 43s 105ms/step - accuracy: 0.9712 - loss: 0.0861 - val_accuracy: 0.9509 - val_loss: 0.1404
Epoch 19/20
332/332 ━━━━━━━━━━━━━━━━━━━━ 39s 99ms/step - accuracy: 0.9733 - loss: 0.0794 - val_accuracy: 0.8845 - val_loss: 0.3875
Epoch 20/20
332/332 ━━━━━━━━━━━━━━━━━━━━ 42s 102ms/step - accuracy: 0.9685 - loss: 0.0859 - val_accuracy: 0.8989 - val_loss: 0.3366
178/178 ━━━━━━━━━━━━━━━━━━━━ 3s 19ms/step - accuracy: 0.9038 - loss: 0.3212
Validation Loss: 0.3461
Validation Accuracy: 0.9000
```

```
loss, accuracy = model.evaluate(X_test_CNN, y_test_CNN)
print(f"TestLoss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")
```

```
148/148 ━━━━━━━━━━━━━━━━━━━━ 5s 31ms/step - accuracy: 0.8992 - loss: 0.3506
TestLoss: 0.3464
Test Accuracy: 0.8984
```

3. Hyperparameter tuning

Algorithm of choice from first six weeks of course (Random Forest)

```
def tuning_random_forest_classifier(X_train, y_train):
    from sklearn.model_selection import GridSearchCV
    from sklearn.ensemble import RandomForestClassifier

    param_grid = {
    "n_estimators": [50, 100, 200],
    "max_depth": [10, 30, 50],
    "criterion": ['gini', 'entropy']
    }
    model = create_random_forest_classifier(random_state=0)
    grid_search_cv = GridSearchCV(model, param_grid, cv=3, verbose=2)
    grid_search_cv.fit(X_train, y_train)
    best_params = grid_search_cv.best_params_
    best_model = grid_search_cv.best_estimator_

    return best_model, best_params
```

```
X_train_RF, y_train_RF,X_test_RF,y_test_RF = random_forest_preprocess(X_train,y_train,X_test,y_test)

best_RF_model, best_RF_params = tuning_random_forest_classifier(X_train_RF, y_train_RF)

print("Finding the best pararms for Random forest:", best_RF_params)
```

```
Fitting 3 folds for each of 18 candidates, totalling 54 fits
[CV] END ......criterion=gini, max_depth=10, n_estimators=50; total time=   9.6s
[CV] END ......criterion=gini, max_depth=10, n_estimators=50; total time=  14.8s
[CV] END ......criterion=gini, max_depth=10, n_estimators=50; total time=   9.6s
[CV] END .....criterion=gini, max_depth=10, n_estimators=100; total time=  18.1s
[CV] END .....criterion=gini, max_depth=10, n_estimators=100; total time=  18.2s
[CV] END .....criterion=gini, max_depth=10, n_estimators=100; total time=  19.2s
[CV] END .....criterion=gini, max_depth=10, n_estimators=200; total time=  36.9s
[CV] END .....criterion=gini, max_depth=10, n_estimators=200; total time=  38.5s
[CV] END .....criterion=gini, max_depth=10, n_estimators=200; total time=  37.6s
[CV] END ......criterion=gini, max_depth=30, n_estimators=50; total time=  14.3s
[CV] END ......criterion=gini, max_depth=30, n_estimators=50; total time=  14.3s
[CV] END ......criterion=gini, max_depth=30, n_estimators=50; total time=  14.2s
[CV] END .....criterion=gini, max_depth=30, n_estimators=100; total time=  28.5s
[CV] END .....criterion=gini, max_depth=30, n_estimators=100; total time=  44.4s
[CV] END .....criterion=gini, max_depth=30, n_estimators=100; total time=  30.9s
[CV] END .....criterion=gini, max_depth=30, n_estimators=200; total time=  57.8s
[CV] END .....criterion=gini, max_depth=30, n_estimators=200; total time=  57.1s
[CV] END .....criterion=gini, max_depth=30, n_estimators=200; total time=  56.8s
[CV] END ......criterion=gini, max_depth=50, n_estimators=50; total time=  14.5s
[CV] END ......criterion=gini, max_depth=50, n_estimators=50; total time=  14.3s
[CV] END ......criterion=gini, max_depth=50, n_estimators=50; total time=  14.4s
[CV] END .....criterion=gini, max_depth=50, n_estimators=100; total time=  28.7s
```

```
[CV] END .....criterion=gini, max_depth=50, n_estimators=100; total time=  28.5s
[CV] END .....criterion=gini, max_depth=50, n_estimators=100; total time=  28.5s
[CV] END .....criterion=gini, max_depth=50, n_estimators=200; total time=  57.4s
[CV] END .....criterion=gini, max_depth=50, n_estimators=200; total time=  57.8s
[CV] END .....criterion=gini, max_depth=50, n_estimators=200; total time=  57.0s
[CV] END ...criterion=entropy, max_depth=10, n_estimators=50; total time=  13.5s
[CV] END ...criterion=entropy, max_depth=10, n_estimators=50; total time=  13.4s
[CV] END ...criterion=entropy, max_depth=10, n_estimators=50; total time=  13.5s
[CV] END ..criterion=entropy, max_depth=10, n_estimators=100; total time=  26.7s
[CV] END ..criterion=entropy, max_depth=10, n_estimators=100; total time=  37.6s
[CV] END ..criterion=entropy, max_depth=10, n_estimators=100; total time=  37.1s
[CV] END ..criterion=entropy, max_depth=10, n_estimators=200; total time= 1.2min
[CV] END ..criterion=entropy, max_depth=10, n_estimators=200; total time=  57.7s
[CV] END ..criterion=entropy, max_depth=10, n_estimators=200; total time= 1.3min
[CV] END ...criterion=entropy, max_depth=30, n_estimators=50; total time=  19.6s
[CV] END ...criterion=entropy, max_depth=30, n_estimators=50; total time=  22.4s
[CV] END ...criterion=entropy, max_depth=30, n_estimators=50; total time=  28.7s
[CV] END ..criterion=entropy, max_depth=30, n_estimators=100; total time= 1.2min
[CV] END ..criterion=entropy, max_depth=30, n_estimators=100; total time=  39.5s
[CV] END ..criterion=entropy, max_depth=30, n_estimators=100; total time=  49.5s
[CV] END ..criterion=entropy, max_depth=30, n_estimators=200; total time= 1.6min
[CV] END ..criterion=entropy, max_depth=30, n_estimators=200; total time= 1.5min
[CV] END ..criterion=entropy, max_depth=30, n_estimators=200; total time= 1.3min
[CV] END ...criterion=entropy, max_depth=50, n_estimators=50; total time=  20.3s
[CV] END ...criterion=entropy, max_depth=50, n_estimators=50; total time=  19.3s
[CV] END ...criterion=entropy, max_depth=50, n_estimators=50; total time=  20.4s
[CV] END ..criterion=entropy, max_depth=50, n_estimators=100; total time=  38.4s
[CV] END ..criterion=entropy, max_depth=50, n_estimators=100; total time=  42.5s
[CV] END ..criterion=entropy, max_depth=50, n_estimators=100; total time=  57.6s
[CV] END ..criterion=entropy, max_depth=50, n_estimators=200; total time= 1.6min
[CV] END ..criterion=entropy, max_depth=50, n_estimators=200; total time= 1.6min
[CV] END ..criterion=entropy, max_depth=50, n_estimators=200; total time= 1.3min
Best parameters for RF classifier: {'criterion': 'entropy', 'max_depth': 30, 'n_estimators': 200}
```

Fully connected neural network

```
from scikeras.wrappers import KerasClassifier


keras_classifier = KerasClassifier(tunable_fully_network,
                                   n_hidden_layers=2,
                                   n_hidden_neurons=50,
                                   activation_function="relu",
                                   loss="categorical_crossentropy",
                                   optimizer="sgd",
                                   optimizer__learning_rate=0.01,
                                   metrics=["accuracy"]
                                  )
print(keras_classifier.get_params().keys())
param_grid = {
    "n_hidden_neurons": [100, 200],
    "optimizer__learning_rate": [0.1, 0.01, 0.001],
    "activation_function": ["relu", "sigmoid", "tanh"]
}

# Use standard scoring strings for simplicity where possible
scoring = {
    'accuracy': 'accuracy',  # Directly use string identifier
    'precision': make_scorer(precision_score, average='macro', zero_division=0),
    'recall': make_scorer(recall_score, average='macro', zero_division=0)
}

grid_search_cv = GridSearchCV(keras_classifier, param_grid, cv=3, verbose=2)
grid_search_cv.fit(X_train_MLP, y_train_MLP, epochs=20)

best_params = grid_search_cv.best_params_
best_model = grid_search_cv.best_estimator_.model
results = grid_search_cv.cv_results_
```

Double-click (or enter) to edit

```
#hyperparam tuning for fully connected model (MLP)
#fully_conn_preprocess(X_train, Y_train, X_test, Y_test)

# X_train_MLP, y_train_MLP,X_valid_MLP,y_valid_MLP = fully_conn_preprocess(X_train,y_train,X_test,y_test)

# MLP_best_model, MLP_best_params, CV_results = fully_conn_tuning(X_train_MLP, y_train_MLP)
print(best_params) #just lost them
print(results)

# print("Mean test accuracy for each parameter combination:", results['mean_test_accuracy'])
# print("Mean test precision for each parameter combination:", results['mean_test_precision'])
# print("Mean test recall for each parameter combination:", results['mean_test_recall'])
# print("Fully connected best parameters:", MLP_best_params)
```

Convolutional neural network

```
from scipy.stats import uniform, randint
from sklearn.model_selection import RandomizedSearchCV

def conv_tuning(X_train, y_train):

    #keras_classifier = KerasClassifier(build_fn=create_resnet50, input_shape=[28,28,3], num_classes=8)
    keras_classifier = KerasClassifier(build_fn=convolutional_network, input_shape=[28,28,1], num_classes=11)

    param_dist = {
        "optimizer__learning_rate": uniform(0.0001, 0.1),
        "optimizer": ['SGD', 'RMSprop', 'Adam', 'Adagrad'],
        "loss": ['categorical_crossentropy'],
        "batch_size": randint(16, 129)  # this will sample integers between 16 (inclusive) and 129 (exclusive)
    }

    random_search = RandomizedSearchCV(keras_classifier, param_distributions=param_dist, n_iter=50, cv=3, verbose=2)
    random_search.fit(X_train, y_train)
    best_params = random_search.best_params_
    best_model = random_search.best_estimator_.model



    return best_model, best_params


X_train_CNN, y_train_CNN,X_valid_CNN,y_valid_CNN, X_test,y_test = convolutional_preprocess(X_train,y_train,X_test,y_test)

best_model, best_params = conv_tuning(X_train, y_train)

# Output the best parameters and optionally evaluate the model
print("Best hyperparameters:", best_params)
#Best output hyperparameters: {'batch_size': 24, 'loss': 'categorical_crossentropy', 'optimizer': 'RMSprop', 'optimizer__le

# If you also have validation or test data prepared, evaluate the best model
loss, accuracy = best_model.evaluate(X_valid, y_valid)  # or use X_test, y_test
print(f"Validation Loss: {loss:.4f}")
print(f"Validation Accuracy: {accuracy:.4f}")
```

```
# If you also have validation or test data prepared, evaluate the best model
loss, accuracy = best_model.evaluate(X_valid_CNN, y_valid_CNN)  # or use X_test, y_test
print(f"Validation Loss: {loss:.4f}")
print(f"Validation Accuracy: {accuracy:.4f}")
```

4. Final models

In this section, please ensure to include cells to train each model with its best hyperparmater combination independently of the hyperparameter tuning cells, i.e. don't rely on the hyperparameter tuning cells having been run.

Algorithm of choice from first six weeks of course

```
import time
from sklearn import metrics
final_RF_hyperparams = {
    'criterion': 'entropy',
    'max_depth': 30,
    'n_estimators': 200}

# test hyperparamter tuning fo RF classifier
X_train_rf, y_train_rf,X_test_rf,y_test_rf = random_forest_preprocess(X_train,y_train,X_test,y_test)

# Start measuring the time
start_time = time.time()
best_RF_model = create_random_forest_classifier(**final_RF_hyperparams)
best_RF_history = best_RF_model.fit(X_train_rf, y_train_rf)

# End measuring the time
end_time = time.time()

# Calculate the runtime
rf_runtime = end_time - start_time
print(f"Training for RF model took {rf_runtime} seconds.")



y_pred = best_RF_model.predict(X_test)
accuracy = accuracy_score(y_test_rf, y_pred) #not sure why it is not working now at the end
f1 = metrics.f1_score(y_test_rf,y_pred,average='micro')
y_prob = best_RF_model.predict_proba(X_test_rf)
loss = metrics.log_loss(y_test_rf, y_prob)

print(f"Test Accuracy: {accuracy:.4f}")
print(f"Test F1 score: {f1:.4f}")
print(f"Test Loss: {loss:.4f}")
```

```
y_pred = best_RF_model.predict(X_test_rf)
accuracy = accuracy_score(y_test_rf, y_pred)
```

```
metrics_plot(best_RF_model, X_test_rf, y_test_rf)
```

```
features_plot(best_RF_model, X_train_rf) #not really usefull so, we should delete this one
```

Fully connected neural network

```
import time
final_MLP_hyperparams = {
    'criterion': 'sigmoid',
    'max_depth': 30,
    'n_estimators': 200,
    }


X_train_mlp, y_train_mlp, X_test_mlp, y_test_mlp  = fully_conn_preprocess(X_train, y_train, X_test, y_test)

start_time = time.time()

best_mlp_model = tunable_fully_network(n_hidden_layers=2, n_hidden_neurons=200,activation_function="relu", input_shape=(784
MLP_history = best_mlp_model.fit(X_train_mlp, y_train_mlp, epochs=20, batch_size=32, validation_split=0.2)
# # best_MLP_history = best_mlp_model.fit(X_train_mlp, y_train_mlp, epochs=20, batch_size=32, validation_split=0.2) #needs
# final_MLP_model = tunable_fully_network(**final_MLP_hyperparams)
# End measuring the time
end_time = time.time()

# Calculate the runtime
mlp_runtime = end_time - start_time
print(f"Training for MLP model took {mlp_runtime} seconds.")

# Evaluate the model
loss, accuracy = model.evaluate(X_test_mlp, y_test_mlp)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")
plot_training_acc(MLP_history)
```

Convolutional neural network

```
best_CNN_hyperparameters = {'batch_size': 24,
                            'loss': 'categorical_crossentropy',
                            'optimizer': 'RMSprop',
                            'optimizer__learning_rate': 0.06146926284871222,
                            }
```

```
# best CNN model
X_train_CNN, y_train_CNN,X_valid_CNN,y_valid_CNN,X_test_CNN,y_test_CNN = convolutional_preprocess(X_train,y_train,X_test,y_
```

```
# Start measuring the time
start_time = time.time()
#adam actually appears to be better than selected RMSprop as hyperparm
best_cnn_model = convolutional_network(input_shape=[28,28,1], num_classes=11, optimizer='RMSprop', learning_rate=0.06146926

cnn_history = best_cnn_model .fit(X_train_CNN, y_train_CNN, epochs=20, batch_size=24, validation_split=0.2)
loss, accuracy = best_cnn_model .evaluate(X_valid_CNN, y_valid_CNN)
```

```
# End measuring the time
end_time = time.time()

# Calculate the runtime
cnn_runtime = end_time - start_time
print(f"Training for CNN model took {cnn_runtime} seconds.")
```

```
# Evaluate the model
loss, accuracy = best_cnn_model.evaluate(X_test_CNN, y_test_CNN)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")

plot_accuracy(cnn_history)
plot_loss(cnn_history)
```

```
# Evaluate the model
loss, accuracy = model.evaluate(X_test_CNN, y_test_CNN)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")

plot_accuracy(cnn_history)
plot_loss(cnn_history)
```