Assignment 2

October 14, 2023

1 COMP4318/COMP5318 Assignment 2

In this template, we have provided data loading code and section headings to help structure your notebook. Please refer to the assignment specification pdf to guide the content of your notebook and report.

(Add SIDs here)

2 Setup

Here I'd like to use numpy to do some exploratory data analysis, matplotlib is used to draw some figures

sklearn is used to build some machine learning models, and pipeline, etc.

tensorflow / keras is used to build mlp and cnn models

[69]:
```
import numpy as np

import matplotlib.pyplot as plt
import tensorflow as tf
import keras

from sklearn.preprocessing import OneHotEncoder
from sklearn.base import TransformerMixin, BaseEstimator
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
```

3 Data loading, exploration, and preprocessing

3.1 Data loading

From the https://medmnist.com/, they give a recommended way to get the dataset via install their lib: medmnist

But because of the bad connection, I download the BloodMNIST.npz from Hugging Face: https://huggingface.co/datasets/albertvillanova/medmnist-v2/tree/main/data

[7]:
```python
dataset = np.load(r'./Assignment2Data/bloodmnist.npz')
dataset.files
```

[7]:
```
['train_images',
 'train_labels',
 'val_images',
 'val_labels',
 'test_images',
 'test_labels']
```

The whole dataset is divided into 6 pieces, and I'll load them respectively

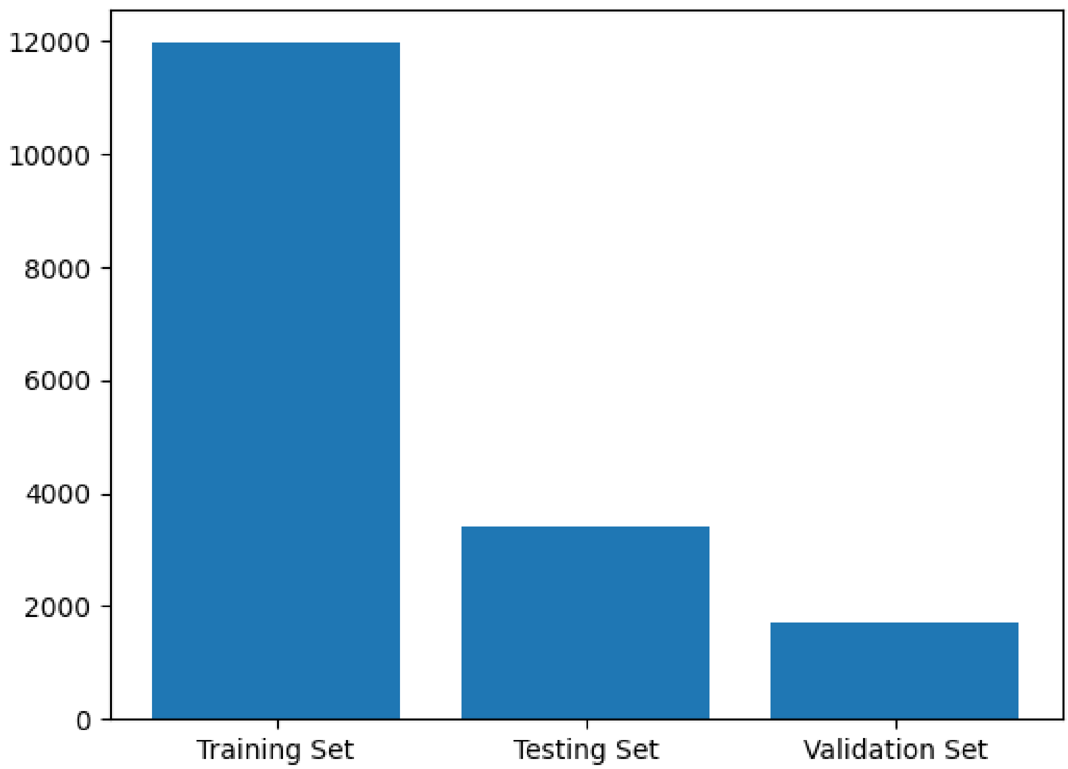
[8]:
```python
X_train, X_test, X_val = dataset['train_images'], dataset['test_images'],␣
↪dataset['val_images']
y_train, y_test, y_val = dataset['train_labels'], dataset['test_labels'],␣
↪dataset['val_labels']
```

3.2 Data exploration

First, see the shape of train set, validation set and test set

[11]:
```python
print('The shape of training set: X_train: {}, y_train: {}'.format(X_train.
↪shape, y_train.shape))
print('The shape of testing set: X_test: {}, y_test: {}'.format(X_test.shape,␣
↪y_test.shape))
print('The shape of validation set: X_val: {}, y_val: {}'.format(X_val.shape,␣
↪y_val.shape))

# x = np.array(['Training Set', 'Testing Set', 'Validation Set'])
x = np.array(['Training Set', 'Testing Set', 'Validation Set'])
y = np.array([X_train.shape[0], X_test.shape[0], X_val.shape[0]])
plt.bar(x, y)
plt.show()
```

```
The shape of training set: X_train: (11959, 28, 28, 3), y_train: (11959, 1)
The shape of testing set: X_test: (3421, 28, 28, 3), y_test: (3421, 1)
The shape of validation set: X_val: (1712, 28, 28, 3), y_val: (1712, 1)
```

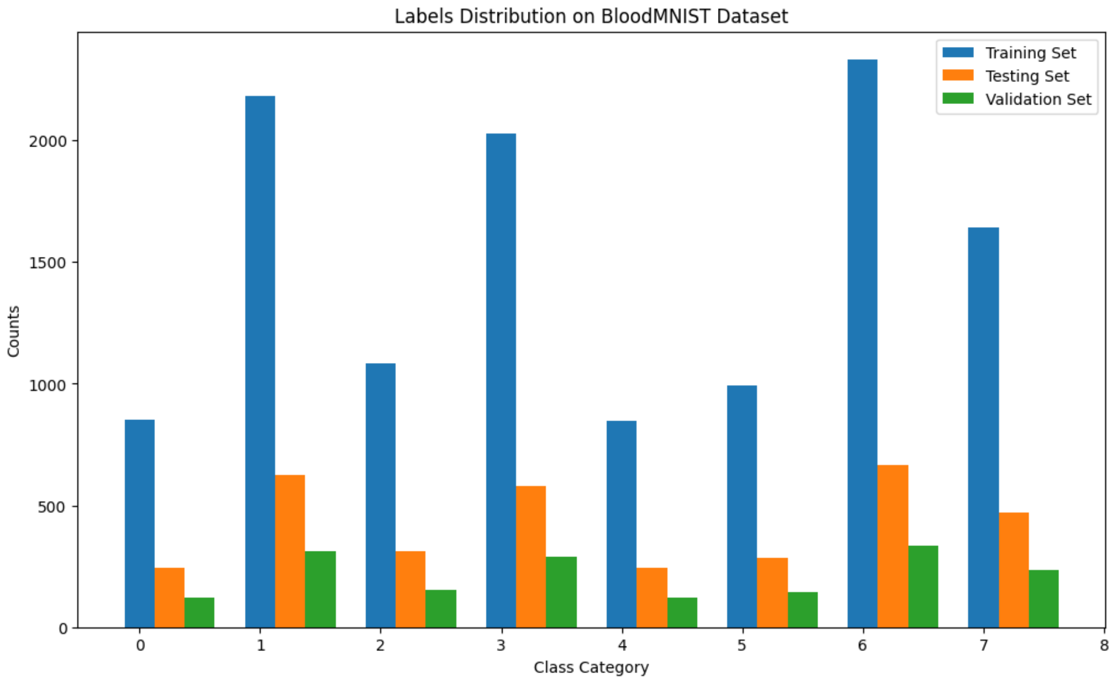


What similar to MNIST is, just talking about the training set, 1 piece of data is a 2D-figure, with the scale of 28 * 28. But, data in MNIST is gray level image, data in that is RGB image, that's the meaning of "3" in the X-set.shape[-1]

And then, it's better to check whether there's any data skew, so I'll take a grouped bar chart to visualization the distribution of 3 label sets:

[21]:
```
y_train_unique, y_train_counts = np.unique(y_train, return_counts=True)
y_test_unique, y_test_counts = np.unique(y_test, return_counts=True)
y_val_unique, y_val_counts = np.unique(y_val, return_counts=True)

width = 0.25
x1 = np.arange(len(y_train_unique))
x2 = [x + width for x in x1]
x3 = [x + width for x in x2]

plt.figure(figsize=(12, 7))
plt.bar(x1, y_train_counts, width=width, label='Training Set')
plt.bar(x2, y_test_counts, width=width, label='Testing Set')
plt.bar(x3, y_val_counts, width=width, label='Validation Set')
plt.title('Labels Distribution on BloodMNIST Dataset')
plt.xlabel('Class Category')
plt.ylabel('Counts')
```

```
plt.legend()
plt.show()
```



And I can see: the labels distributions in 3 sets are average, so I need not do some works on data skew specifically.

The 3rd step of eda is to observe the concrete images, here I displayed 1 image from class0 to class7 in training set:

[24]:
```
plt.figure(figsize=(12, 7))
selected_index = []
for idx, current_class in enumerate(np.unique(y_train)):
    current_index = np.where(y_train == current_class)[0][0]
    selected_index.append(current_index)
    plt.subplot(1, len(y_train_unique), idx + 1)
    plt.imshow(X_train[current_index])
    plt.title(f"Class: {current_class}")
    plt.axis('off')
plt.show()
print('Selected those figs from: ')
for i in range(len(y_train_unique)):
    print('class {}: index is {}'.format(i, selected_index[i]))
```


```
Selected those figs from:
class 0: index is 26
class 1: index is 10
class 2: index is 6
class 3: index is 1
class 4: index is 28
class 5: index is 22
class 6: index is 2
class 7: index is 0
```

So, I can take the next step: Data Preprocessing

3.3 Preprocessing

The 1st step, I'd like do the normalization. This step will use min-max normalization, and because we have already known that, the data value is pixel, so the min is 0, and the max is 255, I can divide 255 directly:

[27]:
```python
# Map the standardized data to [0, 1] range
def normalization(images):
    return (images - images.min()) / (images.max() - images.min())
```

[28]:
```python
X_train_norm = normalization(X_train)
X_test_norm = normalization(X_test)
X_val_norm = normalization(X_val)
```

And then, check whether it is worked

Take the training set as example:

[29]:
```python
X_train_norm.min(), X_train_norm.max()
```

[29]:
```
(0.0, 1.0)
```

And the 2nd step, I'll do One-hot Encoding, because there're so many categories

[33]:
```python
encoder = OneHotEncoder(sparse=False)

# Fit and transform the labels to one-hot encoding
y_train_onehot = encoder.fit_transform(y_train.reshape(-1, 1))
y_val_onehot = encoder.transform(y_val.reshape(-1, 1))
y_test_onehot = encoder.transform(y_test.reshape(-1, 1))
```

```
y_train_onehot.shape, y_val_onehot.shape, y_test_onehot.shape
```

```
C:\Users\10754\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\preprocessing\_encoders.py:972: FutureWarning: `sparse` was
renamed to `sparse_output` in version 1.2 and will be removed in 1.4.
`sparse_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
```

[33]: ((11959, 8), (1712, 8), (3421, 8))

Because there're 8 classes, so after One-hot, the 2nd dim becomes 8

Finally, I'll build a pipeline from sklearn, to ensure that all data can be processed by the same steps

[56]:
```
class ImageDataNormalization(BaseEstimator, TransformerMixin):
    def fit(self, X):
        self.min_value = X.min()
        self.max_value = X.max()
        return self

    def transform(self, X):
        norm_data = (X - self.min_value) / (self.max_value - self.min_value)
        return norm_data


class ImageReshaper(BaseEstimator, TransformerMixin):
    def fit(self, X):
        return self

    def transform(self, X):
        return X.reshape(X.shape[0], -1)


class LabelOneHotEncoder(BaseEstimator, TransformerMixin):
    def fit(self, y):
        self.encoder = OneHotEncoder()
        self.encoder.fit(y.reshape(-1, 1))
        return self

    def transform(self, y):
        return self.encoder.transform(y.reshape(-1, 1)).toarray()
```

Noticed that, CNNs need not reshape but for other machine learning algorithms or mlp, data must be reshaped

So I create another pipeline with reshaper

[57]:
```python
# Create the pipeline
image_pipeline_without_reshape = Pipeline([
    ("scaler", ImageDataNormalization())
])

image_pipeline_with_reshape = Pipeline([
    ("scaler", ImageDataNormalization()),
    ("reshaper", ImageReshaper())
])

label_pipeline = Pipeline([
    ("one_hot_encoder", LabelOneHotEncoder())
])
```

[58]:
```python
# Fit and transform using the pipelines
X_train_preprocessed = image_pipeline_without_reshape.fit_transform(X_train)
X_train_preprocessed_reshaped = image_pipeline_with_reshape.
 ↪fit_transform(X_train)
y_train_preprocessed = label_pipeline.fit_transform(y_train)

X_train_preprocessed.shape, X_train_preprocessed_reshaped.shape,␣
 ↪y_train_preprocessed.shape
```

[58]:
```
((11959, 28, 28, 3), (11959, 2352), (11959, 8))
```

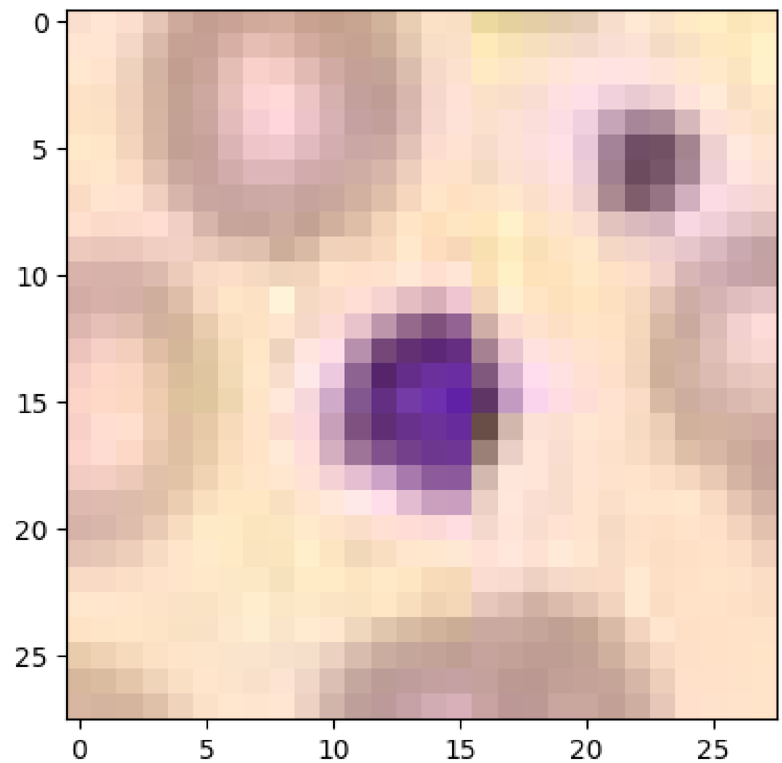
3.4 Examples of preprocessed data

Here're some preprocessed data:

[51]:
```python
# processed x data without reshape is still an image
plt.imshow(X_train_preprocessed[0])
X_train_preprocessed[0]
```

[51]:
```
array([[[0.98039216, 0.8745098 , 0.80784314],
        [1.        , 0.89803922, 0.83137255],
        [0.98431373, 0.87058824, 0.80784314],
        …,
        [1.        , 0.93333333, 0.76470588],
        [0.98431373, 0.90196078, 0.7254902 ],
        [1.        , 0.91764706, 0.74901961]],

       [[1.        , 0.89803922, 0.82352941],
        [1.        , 0.89803922, 0.82352941],
        [0.95686275, 0.84313725, 0.78039216],
        …,
        [0.99607843, 0.90980392, 0.75686275],
        [0.99607843, 0.91372549, 0.74509804],
        [1.        , 0.94901961, 0.78823529]],
```

```
[[1.        , 0.90588235, 0.81568627],
 [1.        , 0.89019608, 0.80392157],
 [0.93333333, 0.82352941, 0.74117647],
 …,
 [1.        , 0.91372549, 0.79215686],
 [0.98431373, 0.89803922, 0.75294118],
 [1.        , 0.94901961, 0.79607843]],

…,

[[0.91372549, 0.8       , 0.6745098 ],
 [0.9372549 , 0.82352941, 0.70588235],
 [0.96078431, 0.85490196, 0.7372549 ],
 …,
 [1.        , 0.88235294, 0.78039216],
 [1.        , 0.88627451, 0.78431373],
 [1.        , 0.89411765, 0.78039216]],

[[0.84313725, 0.72156863, 0.61176471],
 [0.85490196, 0.7372549 , 0.62745098],
 [0.89411765, 0.77647059, 0.66666667],
 …,
 [1.        , 0.88235294, 0.78823529],
 [0.99607843, 0.89019608, 0.78431373],
 [1.        , 0.89411765, 0.78823529]],

[[0.85098039, 0.71372549, 0.63529412],
 [0.83137255, 0.70588235, 0.62352941],
 [0.83529412, 0.70980392, 0.61960784],
 …,
 [0.99607843, 0.88627451, 0.8       ],
 [1.        , 0.89019608, 0.79607843],
 [1.        , 0.89411765, 0.78823529]]])
```



[52]:
```
# but after reshaping, it becomes a 28*28*3, 1 dim array
X_train_preprocessed_reshaped[0]
```

[52]:
```
array([0.98039216, 0.8745098 , 0.80784314, …, 1.        , 0.89411765,
       0.78823529])
```

Actually, y data after one-hot will be a len * num_of_category matrix Here's also a sparse format: the parameters in a row are: (index, original category) values the difference between those 2 is: whether call .toarray() when return the y data

[60]:
```
print(y_train_preprocessed[:5])
```

```
[[0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1.]]
```

4 Algorithm design and setup

In this part, I just give the format of models configuration And the detailed fitting, hyperparameters searching are in the next part

4.1 Model 1 - Fully Connected Neural Network

[97]:
```
def create_mlp_model(lr=1e-4, activation='relu', optimizer=tf.keras.optimizers.
↪Adam):
    mlp_model = tf.keras.models.Sequential([
        tf.keras.layers.InputLayer(input_shape=(2352,)),
        tf.keras.layers.Dense(1024, activation=activation, ),
        tf.keras.layers.Dense(256, activation=activation, ),
        tf.keras.layers.Dense(64, activation=activation),
        tf.keras.layers.Dense(8, activation='softmax')
    ])
    mlp_model.compile(
        optimizer=optimizer(learning_rate=lr),
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )
    return mlp_model
```

[79]:
```
mlp_model = create_mlp_model()
mlp_model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_16 (Dense)            (None, 1024)              2409472

 dense_17 (Dense)            (None, 256)               262400

 dense_18 (Dense)            (None, 64)                16448

 dense_19 (Dense)            (None, 8)                 520

=================================================================
Total params: 2688840 (10.26 MB)
Trainable params: 2688840 (10.26 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

4.2 Model 2 - Convolutional Neural Network

[99]:
```
def create_cnn_model(lr=1e-4, activation='relu', optimizer=tf.keras.optimizers.
↪Adam):
    cnn =  tf.keras.models.Sequential([

        tf.keras.layers.Conv2D(32, (3, 3), activation=activation,␣
↪input_shape=(28, 28, 3)),
        tf.keras.layers.MaxPooling2D((2, 2)),
```

```
        tf.keras.layers.Conv2D(64, (3, 3), activation=activation),
        tf.keras.layers.MaxPooling2D((2, 2)),

        tf.keras.layers.Flatten(),

        tf.keras.layers.Dense(64, activation=activation),

        tf.keras.layers.Dense(8, activation='softmax')
    ])
    cnn.compile(
        optimizer=optimizer(learning_rate=lr),
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )
    return cnn
```

[100]:
```
cnn_model = create_cnn_model()
cnn_model.summary()
```

```
Model: "sequential_16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 26, 26, 32)        896

 max_pooling2d_4 (MaxPoolin  (None, 13, 13, 32)        0
 g2D)

 conv2d_5 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_5 (MaxPoolin  (None, 5, 5, 64)          0
 g2D)

 flatten_2 (Flatten)         (None, 1600)              0

 dense_60 (Dense)            (None, 64)                102464

 dense_61 (Dense)            (None, 8)                 520

=================================================================
Total params: 122376 (478.03 KB)
Trainable params: 122376 (478.03 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

4.3 Model 3 - Algorithm Choice 1: SVM

[67]:
```
svc_model = SVC(kernel='linear', random_state=42)
svc_model
```

[67]:
```
SVC(kernel='linear', random_state=42)
```

4.4 Model 4 - Algorithm Choice 2

[68]:
```
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model
```

[68]:
```
RandomForestClassifier(random_state=42)
```

5 Hyperparameter tuning

5.1 Model 1 - Fully Connected Neural Network

For MLP models built via tensorflow, KerasClassifier() could not be used in the newest version So I do the searching by setting lots of models with different parameters

[81]:
```
X_val_preprocessed_reshaped = image_pipeline_with_reshape.fit_transform(X_val)
y_val_preprocessed = label_pipeline.fit_transform(y_val)
X_test_preprocessed_reshaped = image_pipeline_with_reshape.fit_transform(X_test)
y_test_preprocessed = label_pipeline.fit_transform(y_test)
```

[86]:
```
mlp_history = mlp_model.fit(X_train_preprocessed_reshaped, y_train_preprocessed,
              epochs=10, batch_size=32,
              validation_data=(X_test_preprocessed_reshaped,␣
 ↪y_test_preprocessed)
)
```

```
Epoch 1/10
374/374 [==============================] - 8s 22ms/step - loss: 0.5501 -
accuracy: 0.8014 - val_loss: 0.6066 - val_accuracy: 0.7846
Epoch 2/10
374/374 [==============================] - 8s 21ms/step - loss: 0.5315 -
accuracy: 0.8063 - val_loss: 0.6326 - val_accuracy: 0.7702
Epoch 3/10
374/374 [==============================] - 8s 22ms/step - loss: 0.5149 -
accuracy: 0.8117 - val_loss: 0.5507 - val_accuracy: 0.8030
Epoch 4/10
374/374 [==============================] - 8s 22ms/step - loss: 0.4925 -
accuracy: 0.8209 - val_loss: 0.5164 - val_accuracy: 0.8103
Epoch 5/10
374/374 [==============================] - 8s 22ms/step - loss: 0.4923 -
accuracy: 0.8215 - val_loss: 0.5836 - val_accuracy: 0.7720
Epoch 6/10
```

```
374/374 [==============================] - 8s 22ms/step - loss: 0.4755 -
accuracy: 0.8268 - val_loss: 0.6185 - val_accuracy: 0.7621
Epoch 7/10
374/374 [==============================] - 8s 21ms/step - loss: 0.4958 -
accuracy: 0.8163 - val_loss: 0.5148 - val_accuracy: 0.8080
Epoch 8/10
374/374 [==============================] - 8s 22ms/step - loss: 0.4665 -
accuracy: 0.8275 - val_loss: 0.6374 - val_accuracy: 0.7632
Epoch 9/10
374/374 [==============================] - 8s 21ms/step - loss: 0.4636 -
accuracy: 0.8262 - val_loss: 0.4856 - val_accuracy: 0.8176
Epoch 10/10
374/374 [==============================] - 8s 22ms/step - loss: 0.4561 -
accuracy: 0.8354 - val_loss: 0.5138 - val_accuracy: 0.8088
```

Change the activation from relu to tanh

[87]:
```
mlp_model_tanh = create_mlp_model(lr=1e-4, activation='tanh')
```

[88]:
```
mlp_tanh_history = mlp_model_tanh.fit(X_train_preprocessed_reshaped,␣
 ↪y_train_preprocessed,
                epochs=10, batch_size=32,
                validation_data=(X_test_preprocessed_reshaped,␣
 ↪y_test_preprocessed)
)
```

```
Epoch 1/10
374/374 [==============================] - 9s 22ms/step - loss: 1.1739 -
accuracy: 0.5785 - val_loss: 0.8744 - val_accuracy: 0.7010
Epoch 2/10
374/374 [==============================] - 8s 23ms/step - loss: 0.8065 -
accuracy: 0.7129 - val_loss: 0.7351 - val_accuracy: 0.7299
Epoch 3/10
374/374 [==============================] - 9s 23ms/step - loss: 0.7133 -
accuracy: 0.7430 - val_loss: 0.7022 - val_accuracy: 0.7390
Epoch 4/10
374/374 [==============================] - 9s 23ms/step - loss: 0.6483 -
accuracy: 0.7656 - val_loss: 0.6420 - val_accuracy: 0.7583
Epoch 5/10
374/374 [==============================] - 8s 22ms/step - loss: 0.6152 -
accuracy: 0.7797 - val_loss: 0.6144 - val_accuracy: 0.7770
Epoch 6/10
374/374 [==============================] - 8s 22ms/step - loss: 0.6064 -
accuracy: 0.7792 - val_loss: 0.5628 - val_accuracy: 0.7983
Epoch 7/10
374/374 [==============================] - 8s 22ms/step - loss: 0.5617 -
accuracy: 0.7988 - val_loss: 0.5629 - val_accuracy: 0.7930
Epoch 8/10
```

```
374/374 [==============================] - 8s 22ms/step - loss: 0.5473 -
accuracy: 0.8006 - val_loss: 0.6280 - val_accuracy: 0.7550
Epoch 9/10
374/374 [==============================] - 8s 22ms/step - loss: 0.5380 -
accuracy: 0.8034 - val_loss: 0.5456 - val_accuracy: 0.7936
Epoch 10/10
374/374 [==============================] - 8s 22ms/step - loss: 0.5168 -
accuracy: 0.8155 - val_loss: 0.5537 - val_accuracy: 0.7878
```

change the learning rate

[89]:
```
mlp_model_2lr = create_mlp_model(lr=2e-4)
mlp_2r_history = mlp_model_2lr.fit(X_train_preprocessed_reshaped,␣
 ↪y_train_preprocessed,
              epochs=10, batch_size=32,
              validation_data=(X_test_preprocessed_reshaped,␣
 ↪y_test_preprocessed)
)
```

```
Epoch 1/10
374/374 [==============================] - 8s 21ms/step - loss: 1.2279 -
accuracy: 0.5542 - val_loss: 0.9858 - val_accuracy: 0.6367
Epoch 2/10
374/374 [==============================] - 8s 22ms/step - loss: 0.8747 -
accuracy: 0.6859 - val_loss: 0.8511 - val_accuracy: 0.6986
Epoch 3/10
374/374 [==============================] - 8s 22ms/step - loss: 0.7849 -
accuracy: 0.7154 - val_loss: 0.7940 - val_accuracy: 0.7007
Epoch 4/10
374/374 [==============================] - 8s 22ms/step - loss: 0.7081 -
accuracy: 0.7410 - val_loss: 0.6939 - val_accuracy: 0.7448
Epoch 5/10
374/374 [==============================] - 8s 22ms/step - loss: 0.6642 -
accuracy: 0.7560 - val_loss: 0.6613 - val_accuracy: 0.7457
Epoch 6/10
374/374 [==============================] - 8s 22ms/step - loss: 0.6284 -
accuracy: 0.7747 - val_loss: 0.6247 - val_accuracy: 0.7682
Epoch 7/10
374/374 [==============================] - 8s 22ms/step - loss: 0.6117 -
accuracy: 0.7751 - val_loss: 0.5979 - val_accuracy: 0.7834
Epoch 8/10
374/374 [==============================] - 9s 23ms/step - loss: 0.5914 -
accuracy: 0.7825 - val_loss: 0.6034 - val_accuracy: 0.7702
Epoch 9/10
374/374 [==============================] - 9s 23ms/step - loss: 0.5515 -
accuracy: 0.7965 - val_loss: 0.5329 - val_accuracy: 0.8018
Epoch 10/10
374/374 [==============================] - 9s 23ms/step - loss: 0.5474 -
```

```
accuracy: 0.8001 - val_loss: 0.5390 - val_accuracy: 0.7963
```

[90]:
```
mlp_model_02lr = create_mlp_model(lr=2e-5)
mlp_02r_history = mlp_model_02lr.fit(X_train_preprocessed_reshaped,␣
↪y_train_preprocessed,
                epochs=10, batch_size=32,
                validation_data=(X_test_preprocessed_reshaped,␣
↪y_test_preprocessed)
)
```

```
Epoch 1/10
374/374 [==============================] - 8s 21ms/step - loss: 1.4596 -
accuracy: 0.4872 - val_loss: 1.1759 - val_accuracy: 0.6296
Epoch 2/10
374/374 [==============================] - 8s 22ms/step - loss: 1.0645 -
accuracy: 0.6438 - val_loss: 0.9660 - val_accuracy: 0.6852
Epoch 3/10
374/374 [==============================] - 8s 22ms/step - loss: 0.9240 -
accuracy: 0.6889 - val_loss: 0.9106 - val_accuracy: 0.7027
Epoch 4/10
374/374 [==============================] - 8s 22ms/step - loss: 0.8566 -
accuracy: 0.7121 - val_loss: 0.8230 - val_accuracy: 0.7124
Epoch 5/10
374/374 [==============================] - 8s 22ms/step - loss: 0.8082 -
accuracy: 0.7246 - val_loss: 0.8408 - val_accuracy: 0.6922
Epoch 6/10
374/374 [==============================] - 8s 22ms/step - loss: 0.7741 -
accuracy: 0.7351 - val_loss: 0.7777 - val_accuracy: 0.7264
Epoch 7/10
374/374 [==============================] - 8s 22ms/step - loss: 0.7483 -
accuracy: 0.7467 - val_loss: 0.7250 - val_accuracy: 0.7489
Epoch 8/10
374/374 [==============================] - 8s 22ms/step - loss: 0.7146 -
accuracy: 0.7601 - val_loss: 0.7091 - val_accuracy: 0.7451
Epoch 9/10
374/374 [==============================] - 8s 22ms/step - loss: 0.6933 -
accuracy: 0.7682 - val_loss: 0.6844 - val_accuracy: 0.7624
Epoch 10/10
374/374 [==============================] - 8s 22ms/step - loss: 0.6741 -
accuracy: 0.7718 - val_loss: 0.6595 - val_accuracy: 0.7726
```

change both of learning rate and activation

[91]:
```
mlp_model_tanh_2r = create_mlp_model(lr=2e-4, activation='tanh')
mlp_tanh_2r_history = mlp_model_tanh_2r.fit(X_train_preprocessed_reshaped,␣
↪y_train_preprocessed,
                epochs=10, batch_size=32,
```

```
                validation_data=(X_test_preprocessed_reshaped,␣
 ↪y_test_preprocessed)
)
```

```
Epoch 1/10
374/374 [==============================] - 9s 22ms/step - loss: 1.2104 -
accuracy: 0.5658 - val_loss: 0.8680 - val_accuracy: 0.6887
Epoch 2/10
374/374 [==============================] - 8s 22ms/step - loss: 0.8179 -
accuracy: 0.7084 - val_loss: 0.7743 - val_accuracy: 0.7220
Epoch 3/10
374/374 [==============================] - 8s 22ms/step - loss: 0.7616 -
accuracy: 0.7216 - val_loss: 0.6857 - val_accuracy: 0.7577
Epoch 4/10
374/374 [==============================] - 8s 22ms/step - loss: 0.7112 -
accuracy: 0.7416 - val_loss: 0.6822 - val_accuracy: 0.7600
Epoch 5/10
374/374 [==============================] - 8s 22ms/step - loss: 0.6586 -
accuracy: 0.7616 - val_loss: 0.7637 - val_accuracy: 0.7167
Epoch 6/10
374/374 [==============================] - 8s 22ms/step - loss: 0.6364 -
accuracy: 0.7689 - val_loss: 0.6503 - val_accuracy: 0.7612
Epoch 7/10
374/374 [==============================] - 8s 22ms/step - loss: 0.6258 -
accuracy: 0.7714 - val_loss: 0.6742 - val_accuracy: 0.7498
Epoch 8/10
374/374 [==============================] - 8s 22ms/step - loss: 0.5756 -
accuracy: 0.7910 - val_loss: 0.6509 - val_accuracy: 0.7545
Epoch 9/10
374/374 [==============================] - 8s 22ms/step - loss: 0.5487 -
accuracy: 0.8028 - val_loss: 0.6086 - val_accuracy: 0.7805
Epoch 10/10
374/374 [==============================] - 8s 22ms/step - loss: 0.5564 -
accuracy: 0.7956 - val_loss: 0.5524 - val_accuracy: 0.7895
```

[92]:

```
mlp_model_tanh_02r = create_mlp_model(lr=2e-5, activation='tanh')
mlp_tanh_02r_history = mlp_model_tanh_2r.fit(X_train_preprocessed_reshaped,␣
 ↪y_train_preprocessed,
                epochs=10, batch_size=32,
                validation_data=(X_test_preprocessed_reshaped,␣
 ↪y_test_preprocessed)
)
```

```
Epoch 1/10
374/374 [==============================] - 8s 22ms/step - loss: 0.5552 -
accuracy: 0.7982 - val_loss: 0.6980 - val_accuracy: 0.7729
Epoch 2/10
374/374 [==============================] - 8s 22ms/step - loss: 0.5172 -
```

```
accuracy: 0.8094 - val_loss: 0.6697 - val_accuracy: 0.7521
Epoch 3/10
374/374 [==============================] - 8s 22ms/step - loss: 0.5192 -
accuracy: 0.8105 - val_loss: 0.5941 - val_accuracy: 0.7913
Epoch 4/10
374/374 [==============================] - 8s 22ms/step - loss: 0.5009 -
accuracy: 0.8200 - val_loss: 0.4996 - val_accuracy: 0.8126
Epoch 5/10
374/374 [==============================] - 8s 22ms/step - loss: 0.4871 -
accuracy: 0.8226 - val_loss: 0.5174 - val_accuracy: 0.8100
Epoch 6/10
374/374 [==============================] - 8s 22ms/step - loss: 0.4865 -
accuracy: 0.8157 - val_loss: 0.4626 - val_accuracy: 0.8278
Epoch 7/10
374/374 [==============================] - 8s 22ms/step - loss: 0.4654 -
accuracy: 0.8306 - val_loss: 0.5005 - val_accuracy: 0.8100
Epoch 8/10
374/374 [==============================] - 8s 22ms/step - loss: 0.4785 -
accuracy: 0.8271 - val_loss: 0.7968 - val_accuracy: 0.7121
Epoch 9/10
374/374 [==============================] - 8s 22ms/step - loss: 0.4918 -
accuracy: 0.8206 - val_loss: 0.5440 - val_accuracy: 0.8004
Epoch 10/10
374/374 [==============================] - 8s 22ms/step - loss: 0.4522 -
accuracy: 0.8343 - val_loss: 0.6043 - val_accuracy: 0.7650
```

change the batch size

[93]:
```
mlp_model_batch64 = create_mlp_model()
mlp_batch64_history = mlp_model_batch64.fit(X_train_preprocessed_reshaped,␣
 ↪y_train_preprocessed,
               epochs=10, batch_size=64,
               validation_data=(X_test_preprocessed_reshaped,␣
 ↪y_test_preprocessed)
)
```

```
Epoch 1/10
187/187 [==============================] - 5s 23ms/step - loss: 1.3123 -
accuracy: 0.5366 - val_loss: 1.0537 - val_accuracy: 0.5837
Epoch 2/10
187/187 [==============================] - 4s 23ms/step - loss: 0.9303 -
accuracy: 0.6760 - val_loss: 0.8979 - val_accuracy: 0.6767
Epoch 3/10
187/187 [==============================] - 4s 24ms/step - loss: 0.8218 -
accuracy: 0.7127 - val_loss: 0.7923 - val_accuracy: 0.7243
Epoch 4/10
187/187 [==============================] - 4s 24ms/step - loss: 0.7523 -
accuracy: 0.7388 - val_loss: 0.7216 - val_accuracy: 0.7454
```

```
Epoch 5/10
187/187 [==============================] - 4s 24ms/step - loss: 0.7078 -
accuracy: 0.7534 - val_loss: 0.6742 - val_accuracy: 0.7679
Epoch 6/10
187/187 [==============================] - 4s 24ms/step - loss: 0.6762 -
accuracy: 0.7654 - val_loss: 0.6849 - val_accuracy: 0.7428
Epoch 7/10
187/187 [==============================] - 4s 24ms/step - loss: 0.6577 -
accuracy: 0.7663 - val_loss: 0.6197 - val_accuracy: 0.7799
Epoch 8/10
187/187 [==============================] - 4s 24ms/step - loss: 0.6243 -
accuracy: 0.7769 - val_loss: 0.5975 - val_accuracy: 0.7852
Epoch 9/10
187/187 [==============================] - 5s 24ms/step - loss: 0.6033 -
accuracy: 0.7879 - val_loss: 0.6361 - val_accuracy: 0.7694
Epoch 10/10
187/187 [==============================] - 5s 24ms/step - loss: 0.5768 -
accuracy: 0.7951 - val_loss: 0.5634 - val_accuracy: 0.8015
```

[94]:

```
mlp_model_batch128 = create_mlp_model()
mlp_batch128_history = mlp_model_batch128.fit(X_train_preprocessed_reshaped,␣
 ↪y_train_preprocessed,
               epochs=10, batch_size=128,
               validation_data=(X_test_preprocessed_reshaped,␣
 ↪y_test_preprocessed)
)
```

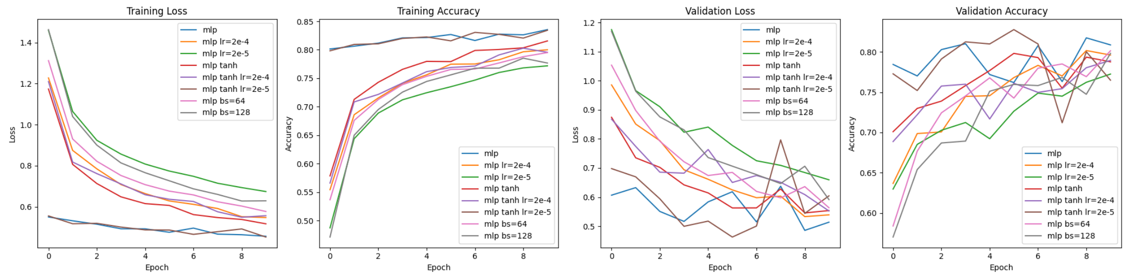
```
Epoch 1/10
94/94 [==============================] - 3s 30ms/step - loss: 1.4634 - accuracy:
0.4710 - val_loss: 1.1698 - val_accuracy: 0.5700
Epoch 2/10
94/94 [==============================] - 3s 29ms/step - loss: 1.0398 - accuracy:
0.6499 - val_loss: 0.9640 - val_accuracy: 0.6539
Epoch 3/10
94/94 [==============================] - 3s 29ms/step - loss: 0.9006 - accuracy:
0.6951 - val_loss: 0.8757 - val_accuracy: 0.6869
Epoch 4/10
94/94 [==============================] - 3s 29ms/step - loss: 0.8137 - accuracy:
0.7256 - val_loss: 0.8311 - val_accuracy: 0.6893
Epoch 5/10
94/94 [==============================] - 3s 29ms/step - loss: 0.7659 - accuracy:
0.7444 - val_loss: 0.7360 - val_accuracy: 0.7512
Epoch 6/10
94/94 [==============================] - 3s 30ms/step - loss: 0.7266 - accuracy:
0.7559 - val_loss: 0.7064 - val_accuracy: 0.7597
Epoch 7/10
94/94 [==============================] - 3s 29ms/step - loss: 0.6873 - accuracy:
0.7676 - val_loss: 0.6773 - val_accuracy: 0.7583
```

```
Epoch 8/10
94/94 [==============================] - 3s 29ms/step - loss: 0.6601 - accuracy:
0.7678 - val_loss: 0.6465 - val_accuracy: 0.7685
Epoch 9/10
94/94 [==============================] - 3s 29ms/step - loss: 0.6279 - accuracy:
0.7853 - val_loss: 0.7062 - val_accuracy: 0.7474
Epoch 10/10
94/94 [==============================] - 3s 29ms/step - loss: 0.6289 - accuracy:
0.7767 - val_loss: 0.5914 - val_accuracy: 0.7983
```

Visualization the result of comparision

[96]:
```
mlps_history_list = [
    mlp_history, mlp_2r_history, mlp_02r_history,
    mlp_tanh_history, mlp_tanh_2r_history, mlp_tanh_02r_history,
    mlp_batch64_history, mlp_batch128_history
]
mlps_legend_list = [
   'mlp', 'mlp lr=2e-4', 'mlp lr=2e-5',
    'mlp tanh', 'mlp tanh lr=2e-4', 'mlp tanh lr=2e-5',
    'mlp bs=64', 'mlp bs=128'
]

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

# 1. Training Loss
plt.subplot(1, 4, 1)
for h, l in zip(mlps_history_list, mlps_legend_list):
    plt.plot(h.history['loss'], label=l)
    plt.title('Training Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

# 2. Training Accuracy
plt.subplot(1, 4, 2)
for h, l in zip(mlps_history_list, mlps_legend_list):
    plt.plot(h.history['accuracy'], label=l)
    plt.title('Training Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

# 3. Validation Loss
plt.subplot(1, 4, 3)
for h, l in zip(mlps_history_list, mlps_legend_list):
    plt.plot(h.history['val_loss'], label=l)
    plt.title('Validation Loss')
```

```
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

# 4. Validation Accuracy
plt.subplot(1, 4, 4)
for h, l in zip(mlps_history_list, mlps_legend_list):
    plt.plot(h.history['val_accuracy'], label=l)
    plt.title('Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

plt.tight_layout()
plt.show()
```



Changing the hyperparameters above can impact the acc and loss in training

5.2 Model 2 - Convolutional Neural Network

[101]:
```
X_val_preprocessed = image_pipeline_without_reshape.fit_transform(X_val)
X_test_preprocessed = image_pipeline_without_reshape.fit_transform(X_test)
```

[102]:
```
cnn = create_cnn_model()
cnn_history = cnn.fit(X_train_preprocessed, y_train_preprocessed,
              epochs=10, batch_size=32,
              validation_data=(X_test_preprocessed, y_test_preprocessed)
)
```

```
Epoch 1/10
374/374 [==============================] - 6s 16ms/step - loss: 1.6178 -
accuracy: 0.4630 - val_loss: 1.1893 - val_accuracy: 0.6501
Epoch 2/10
374/374 [==============================] - 6s 16ms/step - loss: 0.9881 -
accuracy: 0.6793 - val_loss: 0.8914 - val_accuracy: 0.6884
Epoch 3/10
374/374 [==============================] - 6s 16ms/step - loss: 0.8119 -
```

```
accuracy: 0.7215 - val_loss: 0.7652 - val_accuracy: 0.7346
Epoch 4/10
374/374 [==============================] - 6s 17ms/step - loss: 0.7438 -
accuracy: 0.7445 - val_loss: 0.7218 - val_accuracy: 0.7480
Epoch 5/10
374/374 [==============================] - 6s 16ms/step - loss: 0.7011 -
accuracy: 0.7567 - val_loss: 0.6983 - val_accuracy: 0.7539
Epoch 6/10
374/374 [==============================] - 6s 16ms/step - loss: 0.6699 -
accuracy: 0.7652 - val_loss: 0.6800 - val_accuracy: 0.7574
Epoch 7/10
374/374 [==============================] - 6s 16ms/step - loss: 0.6431 -
accuracy: 0.7736 - val_loss: 0.6291 - val_accuracy: 0.7802
Epoch 8/10
374/374 [==============================] - 6s 17ms/step - loss: 0.6165 -
accuracy: 0.7854 - val_loss: 0.6011 - val_accuracy: 0.7892
Epoch 9/10
374/374 [==============================] - 6s 16ms/step - loss: 0.5891 -
accuracy: 0.7993 - val_loss: 0.5791 - val_accuracy: 0.8053
Epoch 10/10
374/374 [==============================] - 6s 16ms/step - loss: 0.5686 -
accuracy: 0.8038 - val_loss: 0.5637 - val_accuracy: 0.8080
```

change the activation from relu to tanh

[103]:
```
cnn_tanh = create_cnn_model(activation='tanh')
cnn_tanh_history = cnn_tanh.fit(X_train_preprocessed, y_train_preprocessed,
                epochs=10, batch_size=32,
                validation_data=(X_test_preprocessed, y_test_preprocessed)
)
```

```
Epoch 1/10
374/374 [==============================] - 7s 17ms/step - loss: 1.3507 -
accuracy: 0.5561 - val_loss: 0.9363 - val_accuracy: 0.7200
Epoch 2/10
374/374 [==============================] - 6s 16ms/step - loss: 0.8432 -
accuracy: 0.7230 - val_loss: 0.7582 - val_accuracy: 0.7521
Epoch 3/10
374/374 [==============================] - 6s 16ms/step - loss: 0.7218 -
accuracy: 0.7590 - val_loss: 0.6770 - val_accuracy: 0.7735
Epoch 4/10
374/374 [==============================] - 6s 16ms/step - loss: 0.6559 -
accuracy: 0.7777 - val_loss: 0.6279 - val_accuracy: 0.7878
Epoch 5/10
374/374 [==============================] - 6s 16ms/step - loss: 0.5992 -
accuracy: 0.8007 - val_loss: 0.5712 - val_accuracy: 0.8053
Epoch 6/10
374/374 [==============================] - 6s 16ms/step - loss: 0.5589 -
accuracy: 0.8122 - val_loss: 0.5485 - val_accuracy: 0.8042
```

```
Epoch 7/10
374/374 [==============================] - 6s 16ms/step - loss: 0.5250 -
accuracy: 0.8242 - val_loss: 0.5148 - val_accuracy: 0.8229
Epoch 8/10
374/374 [==============================] - 6s 16ms/step - loss: 0.4985 -
accuracy: 0.8302 - val_loss: 0.5140 - val_accuracy: 0.8158
Epoch 9/10
374/374 [==============================] - 6s 16ms/step - loss: 0.4748 -
accuracy: 0.8411 - val_loss: 0.4767 - val_accuracy: 0.8319
Epoch 10/10
374/374 [==============================] - 6s 16ms/step - loss: 0.4574 -
accuracy: 0.8419 - val_loss: 0.4707 - val_accuracy: 0.8302
```

change the learning rate

[104]:
```
cnn_2r = create_cnn_model(lr=2e-4)
cnn_2r_history = cnn_2r.fit(X_train_preprocessed, y_train_preprocessed,
               epochs=10, batch_size=32,
               validation_data=(X_test_preprocessed, y_test_preprocessed)
)
```

```
Epoch 1/10
374/374 [==============================] - 7s 16ms/step - loss: 1.2822 -
accuracy: 0.5587 - val_loss: 0.8592 - val_accuracy: 0.6980
Epoch 2/10
374/374 [==============================] - 6s 16ms/step - loss: 0.7719 -
accuracy: 0.7364 - val_loss: 0.7017 - val_accuracy: 0.7588
Epoch 3/10
374/374 [==============================] - 6s 15ms/step - loss: 0.6469 -
accuracy: 0.7818 - val_loss: 0.5894 - val_accuracy: 0.7945
Epoch 4/10
374/374 [==============================] - 6s 16ms/step - loss: 0.5745 -
accuracy: 0.8030 - val_loss: 0.6077 - val_accuracy: 0.7761
Epoch 5/10
374/374 [==============================] - 6s 15ms/step - loss: 0.5359 -
accuracy: 0.8146 - val_loss: 0.5342 - val_accuracy: 0.8068
Epoch 6/10
374/374 [==============================] - 6s 16ms/step - loss: 0.5021 -
accuracy: 0.8246 - val_loss: 0.4791 - val_accuracy: 0.8284
Epoch 7/10
374/374 [==============================] - 6s 16ms/step - loss: 0.4881 -
accuracy: 0.8290 - val_loss: 0.4659 - val_accuracy: 0.8354
Epoch 8/10
374/374 [==============================] - 6s 15ms/step - loss: 0.4582 -
accuracy: 0.8409 - val_loss: 0.4755 - val_accuracy: 0.8308
Epoch 9/10
374/374 [==============================] - 6s 16ms/step - loss: 0.4466 -
accuracy: 0.8466 - val_loss: 0.4442 - val_accuracy: 0.8366
Epoch 10/10
```

```
374/374 [==============================] - 6s 15ms/step - loss: 0.4262 -
accuracy: 0.8499 - val_loss: 0.4340 - val_accuracy: 0.8448
```

[105]:

```
cnn_02r = create_cnn_model(lr=2e-5)
cnn_02r_history = cnn_02r.fit(X_train_preprocessed, y_train_preprocessed,
              epochs=10, batch_size=32,
              validation_data=(X_test_preprocessed, y_test_preprocessed)
)
```

```
Epoch 1/10
374/374 [==============================] - 6s 16ms/step - loss: 1.9684 -
accuracy: 0.2596 - val_loss: 1.8823 - val_accuracy: 0.3560
Epoch 2/10
374/374 [==============================] - 6s 16ms/step - loss: 1.7715 -
accuracy: 0.4185 - val_loss: 1.6519 - val_accuracy: 0.4993
Epoch 3/10
374/374 [==============================] - 6s 16ms/step - loss: 1.5278 -
accuracy: 0.5164 - val_loss: 1.4108 - val_accuracy: 0.5805
Epoch 4/10
374/374 [==============================] - 6s 16ms/step - loss: 1.3032 -
accuracy: 0.5971 - val_loss: 1.2099 - val_accuracy: 0.6185
Epoch 5/10
374/374 [==============================] - 6s 16ms/step - loss: 1.1379 -
accuracy: 0.6307 - val_loss: 1.0857 - val_accuracy: 0.6445
Epoch 6/10
374/374 [==============================] - 6s 16ms/step - loss: 1.0317 -
accuracy: 0.6615 - val_loss: 0.9958 - val_accuracy: 0.6755
Epoch 7/10
374/374 [==============================] - 6s 16ms/step - loss: 0.9604 -
accuracy: 0.6831 - val_loss: 0.9368 - val_accuracy: 0.6881
Epoch 8/10
374/374 [==============================] - 6s 16ms/step - loss: 0.9100 -
accuracy: 0.7003 - val_loss: 0.8957 - val_accuracy: 0.7065
Epoch 9/10
374/374 [==============================] - 6s 16ms/step - loss: 0.8736 -
accuracy: 0.7083 - val_loss: 0.8601 - val_accuracy: 0.7153
Epoch 10/10
374/374 [==============================] - 6s 16ms/step - loss: 0.8469 -
accuracy: 0.7164 - val_loss: 0.8429 - val_accuracy: 0.7241
```

change both learning rate and activation

[106]:

```
cnn_tanh_2r = create_cnn_model(lr=2e-4, activation='tanh')
cnn_tanh_2r_history = cnn_tanh_2r.fit(X_train_preprocessed,␣
 ↪y_train_preprocessed,
              epochs=10, batch_size=32,
              validation_data=(X_test_preprocessed, y_test_preprocessed)
)
```

```
Epoch 1/10
374/374 [==============================] - 6s 16ms/step - loss: 1.1339 -
accuracy: 0.6155 - val_loss: 0.7437 - val_accuracy: 0.7615
Epoch 2/10
374/374 [==============================] - 6s 16ms/step - loss: 0.6807 -
accuracy: 0.7715 - val_loss: 0.6095 - val_accuracy: 0.7819
Epoch 3/10
374/374 [==============================] - 6s 16ms/step - loss: 0.5789 -
accuracy: 0.7999 - val_loss: 0.5508 - val_accuracy: 0.8153
Epoch 4/10
374/374 [==============================] - 6s 16ms/step - loss: 0.5172 -
accuracy: 0.8257 - val_loss: 0.4878 - val_accuracy: 0.8313
Epoch 5/10
374/374 [==============================] - 6s 16ms/step - loss: 0.4731 -
accuracy: 0.8396 - val_loss: 0.4456 - val_accuracy: 0.8442
Epoch 6/10
374/374 [==============================] - 6s 16ms/step - loss: 0.4423 -
accuracy: 0.8535 - val_loss: 0.4323 - val_accuracy: 0.8451
Epoch 7/10
374/374 [==============================] - 6s 16ms/step - loss: 0.4197 -
accuracy: 0.8563 - val_loss: 0.4810 - val_accuracy: 0.8308
Epoch 8/10
374/374 [==============================] - 6s 16ms/step - loss: 0.3952 -
accuracy: 0.8651 - val_loss: 0.3914 - val_accuracy: 0.8576
Epoch 9/10
374/374 [==============================] - 6s 16ms/step - loss: 0.3775 -
accuracy: 0.8705 - val_loss: 0.3873 - val_accuracy: 0.8647
Epoch 10/10
374/374 [==============================] - 6s 16ms/step - loss: 0.3573 -
accuracy: 0.8781 - val_loss: 0.3688 - val_accuracy: 0.8705
```

[107]:

```
cnn_tanh_02r = create_cnn_model(lr=2e-5, activation='tanh')
cnn_tanh_02r_history = cnn_tanh_02r.fit(X_train_preprocessed,␣
 ↪y_train_preprocessed,
                epochs=10, batch_size=32,
                validation_data=(X_test_preprocessed, y_test_preprocessed)
)
```

```
Epoch 1/10
374/374 [==============================] - 7s 17ms/step - loss: 1.8528 -
accuracy: 0.3494 - val_loss: 1.6673 - val_accuracy: 0.4993
Epoch 2/10
374/374 [==============================] - 6s 16ms/step - loss: 1.5049 -
accuracy: 0.5370 - val_loss: 1.3603 - val_accuracy: 0.5785
Epoch 3/10
374/374 [==============================] - 6s 16ms/step - loss: 1.2563 -
accuracy: 0.6204 - val_loss: 1.1639 - val_accuracy: 0.6399
Epoch 4/10
```

```
374/374 [==============================] - 6s 16ms/step - loss: 1.0976 -
accuracy: 0.6685 - val_loss: 1.0365 - val_accuracy: 0.6782
Epoch 5/10
374/374 [==============================] - 6s 16ms/step - loss: 0.9928 -
accuracy: 0.6930 - val_loss: 0.9507 - val_accuracy: 0.7018
Epoch 6/10
374/374 [==============================] - 6s 16ms/step - loss: 0.9213 -
accuracy: 0.7058 - val_loss: 0.8962 - val_accuracy: 0.7018
Epoch 7/10
374/374 [==============================] - 6s 16ms/step - loss: 0.8699 -
accuracy: 0.7200 - val_loss: 0.8511 - val_accuracy: 0.7197
Epoch 8/10
374/374 [==============================] - 6s 16ms/step - loss: 0.8296 -
accuracy: 0.7306 - val_loss: 0.8102 - val_accuracy: 0.7331
Epoch 9/10
374/374 [==============================] - 6s 16ms/step - loss: 0.7978 -
accuracy: 0.7395 - val_loss: 0.7803 - val_accuracy: 0.7480
Epoch 10/10
374/374 [==============================] - 6s 17ms/step - loss: 0.7683 -
accuracy: 0.7491 - val_loss: 0.7560 - val_accuracy: 0.7530
```

change the batch size

[108]:
```
cnn_batch64 = create_cnn_model()
cnn_batch64_history = cnn_batch64.fit(X_train_preprocessed,␣
 ↪y_train_preprocessed,
                epochs=10, batch_size=64,
                validation_data=(X_test_preprocessed, y_test_preprocessed)
)
```

```
Epoch 1/10
187/187 [==============================] - 5s 27ms/step - loss: 1.7099 -
accuracy: 0.4178 - val_loss: 1.3681 - val_accuracy: 0.5539
Epoch 2/10
187/187 [==============================] - 5s 26ms/step - loss: 1.1426 -
accuracy: 0.6267 - val_loss: 0.9883 - val_accuracy: 0.6814
Epoch 3/10
187/187 [==============================] - 5s 26ms/step - loss: 0.9083 -
accuracy: 0.6934 - val_loss: 0.8547 - val_accuracy: 0.7036
Epoch 4/10
187/187 [==============================] - 5s 27ms/step - loss: 0.8271 -
accuracy: 0.7175 - val_loss: 0.7858 - val_accuracy: 0.7241
Epoch 5/10
187/187 [==============================] - 5s 27ms/step - loss: 0.7739 -
accuracy: 0.7340 - val_loss: 0.7691 - val_accuracy: 0.7317
Epoch 6/10
187/187 [==============================] - 5s 27ms/step - loss: 0.7422 -
accuracy: 0.7433 - val_loss: 0.7284 - val_accuracy: 0.7533
```

```
Epoch 7/10
187/187 [==============================] - 5s 27ms/step - loss: 0.7146 -
accuracy: 0.7547 - val_loss: 0.6976 - val_accuracy: 0.7586
Epoch 8/10
187/187 [==============================] - 5s 27ms/step - loss: 0.6897 -
accuracy: 0.7586 - val_loss: 0.6794 - val_accuracy: 0.7624
Epoch 9/10
187/187 [==============================] - 5s 27ms/step - loss: 0.6691 -
accuracy: 0.7645 - val_loss: 0.6669 - val_accuracy: 0.7688
Epoch 10/10
187/187 [==============================] - 5s 27ms/step - loss: 0.6514 -
accuracy: 0.7714 - val_loss: 0.6492 - val_accuracy: 0.7755
```

[109]:

```python
cnn_batch128 = create_cnn_model()
cnn_batch128_history = cnn_batch128.fit(X_train_preprocessed,
 ↪y_train_preprocessed,
               epochs=10, batch_size=128,
               validation_data=(X_test_preprocessed, y_test_preprocessed)
)
```

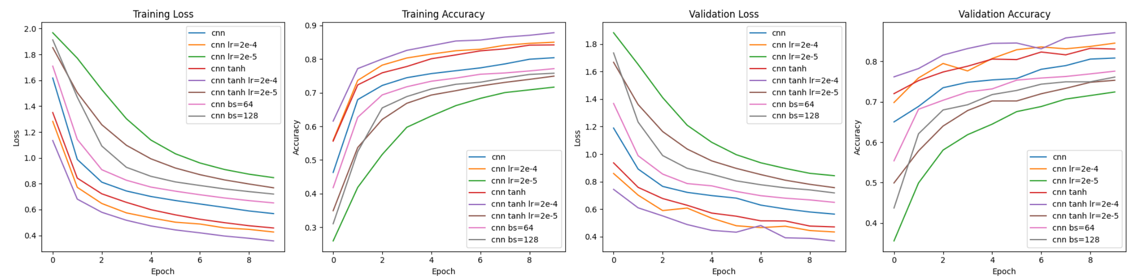
```
Epoch 1/10
94/94 [==============================] - 4s 41ms/step - loss: 1.9133 - accuracy:
0.3106 - val_loss: 1.7354 - val_accuracy: 0.4373
Epoch 2/10
94/94 [==============================] - 4s 43ms/step - loss: 1.4696 - accuracy:
0.5247 - val_loss: 1.2326 - val_accuracy: 0.6209
Epoch 3/10
94/94 [==============================] - 4s 43ms/step - loss: 1.0916 - accuracy:
0.6548 - val_loss: 0.9894 - val_accuracy: 0.6796
Epoch 4/10
94/94 [==============================] - 4s 43ms/step - loss: 0.9272 - accuracy:
0.6884 - val_loss: 0.8967 - val_accuracy: 0.6925
Epoch 5/10
94/94 [==============================] - 4s 43ms/step - loss: 0.8574 - accuracy:
0.7109 - val_loss: 0.8534 - val_accuracy: 0.7176
Epoch 6/10
94/94 [==============================] - 4s 43ms/step - loss: 0.8138 - accuracy:
0.7249 - val_loss: 0.8054 - val_accuracy: 0.7279
Epoch 7/10
94/94 [==============================] - 4s 42ms/step - loss: 0.7862 - accuracy:
0.7311 - val_loss: 0.7771 - val_accuracy: 0.7433
Epoch 8/10
94/94 [==============================] - 4s 43ms/step - loss: 0.7597 - accuracy:
0.7435 - val_loss: 0.7546 - val_accuracy: 0.7492
Epoch 9/10
94/94 [==============================] - 4s 43ms/step - loss: 0.7390 - accuracy:
0.7542 - val_loss: 0.7397 - val_accuracy: 0.7492
Epoch 10/10
```

```
94/94 [==============================] - 4s 43ms/step - loss: 0.7198 - accuracy:
0.7578 - val_loss: 0.7178 - val_accuracy: 0.7609
```

Visualization the result of comparision

[110]:
```python
cnns_history_list = [
    cnn_history, cnn_2r_history, cnn_02r_history,
    cnn_tanh_history, cnn_tanh_2r_history, cnn_tanh_02r_history,
    cnn_batch64_history, cnn_batch128_history
]
cnns_legend_list = [
   'cnn', 'cnn lr=2e-4', 'cnn lr=2e-5',
    'cnn tanh', 'cnn tanh lr=2e-4', 'cnn tanh lr=2e-5',
    'cnn bs=64', 'cnn bs=128'
]

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

# 1. Training Loss
plt.subplot(1, 4, 1)
for h, l in zip(cnns_history_list, cnns_legend_list):
    plt.plot(h.history['loss'], label=l)
    plt.title('Training Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

# 2. Training Accuracy
plt.subplot(1, 4, 2)
for h, l in zip(cnns_history_list, cnns_legend_list):
    plt.plot(h.history['accuracy'], label=l)
    plt.title('Training Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

# 3. Validation Loss
plt.subplot(1, 4, 3)
for h, l in zip(cnns_history_list, cnns_legend_list):
    plt.plot(h.history['val_loss'], label=l)
    plt.title('Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

# 4. Validation Accuracy
plt.subplot(1, 4, 4)
for h, l in zip(cnns_history_list, cnns_legend_list):
```

```
    plt.plot(h.history['val_accuracy'], label=l)
    plt.title('Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

plt.tight_layout()
plt.show()
```



There're some difference between mlp model and cnn model cnn could get a better acc and lower loss when changing the activation from relu to tanh and changing learning rate in a appropriate range will make the efficiency better

5.3 Model 3 - Algorithm Choice 1: SVM

SVC classifier could be built via sklearn, so it can use gridsearchcv to search the best hyperparameters

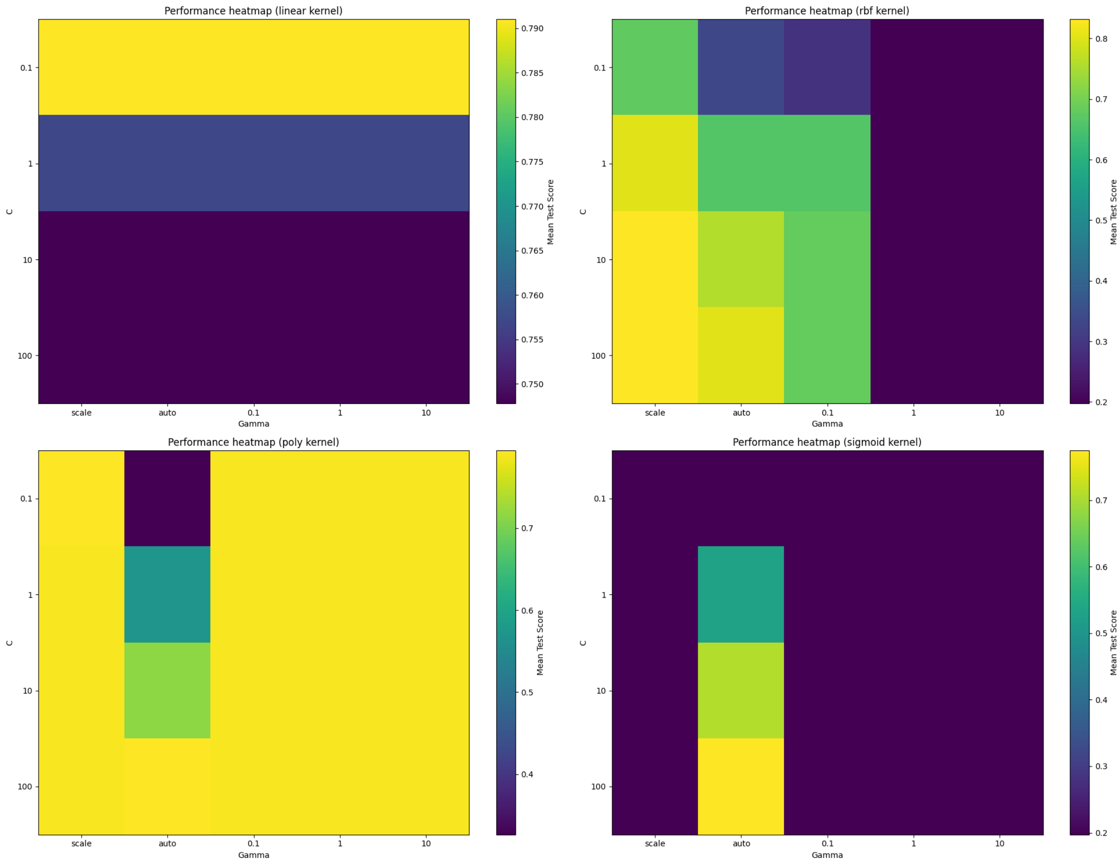
[113]:
```
svc_param_grid = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['linear', 'rbf', 'poly', 'sigmoid'],
    'gamma': ['scale', 'auto', 0.1, 1, 10],
    # Uncomment the line below if you want to also search over different degrees for 'poly' kernel
    # 'degree': [2, 3, 4, 5]
}
# Create a SVC classifier
svc = SVC()

# Create GridSearchCV
grid = GridSearchCV(estimator=svc, param_grid=svc_param_grid, n_jobs=-1, cv=3, verbose=2)
grid_result = grid.fit(X_train_preprocessed_reshaped[:5000], y_train[:5000])
```

```
Fitting 3 folds for each of 80 candidates, totalling 240 fits
```

```
C:\Users\10754\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\utils\validation.py:1184: DataConversionWarning: A column-
```

```
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

[115]:
```python
plt.figure(figsize=(20, 15))
kernels = svc_param_grid['kernel']

for idx, kernel in enumerate(kernels):
    # Extract results for the current kernel
    mask = (np.array(grid_result.cv_results_['param_kernel']) == kernel)
    scores = np.array(grid_result.cv_results_['mean_test_score'])[mask].
 ↪reshape(len(svc_param_grid['C']), len(svc_param_grid['gamma']))

    # Create a heatmap for the current kernel using plt.imshow
    plt.subplot(2, 2, idx+1)
    plt.imshow(scores, cmap="viridis", aspect="auto")
    plt.colorbar(label="Mean Test Score")
    plt.xticks(range(len(svc_param_grid['gamma'])), svc_param_grid['gamma'])
    plt.yticks(range(len(svc_param_grid['C'])), svc_param_grid['C'])
    plt.title(f"Performance heatmap ({kernel} kernel)")
    plt.xlabel("Gamma")
    plt.ylabel("C")

plt.tight_layout()
plt.show()
```



Here I have specified that the kernal is constant for comparison, because of the control variable method, there is always a parameter that needs to be constant

it could be summraized that, when the kernal is linear, whether what the gamma is, when c=0.1, the model is the best And the same is true for the analysis of other heatmaps, but the final best parameter combination can only have one set:

[114]:
```
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Best: 0.831799 using {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
```

5.4 Model 4 - Algorithm Choice 2: Random Forest

Just like SVC, random forest can use gridsearchcv to search the best hyperparameters, too

[116]:
```
# Define the hyperparameters grid
rf_param_grid = {
    'n_estimators': [50, 100, 200],
    'max_features': ['auto', 'sqrt'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
```

```
}

# Create a Random Forest classifier
rf = RandomForestClassifier()

# Create GridSearchCV
grid_rf = GridSearchCV(estimator=rf, param_grid=rf_param_grid, n_jobs=-1, cv=3,␣
↪verbose=2)
grid_result_rf = grid_rf.fit(X_train_preprocessed_reshaped[:5000], y_train[:
↪5000])
```

```
Fitting 3 folds for each of 216 candidates, totalling 648 fits
```

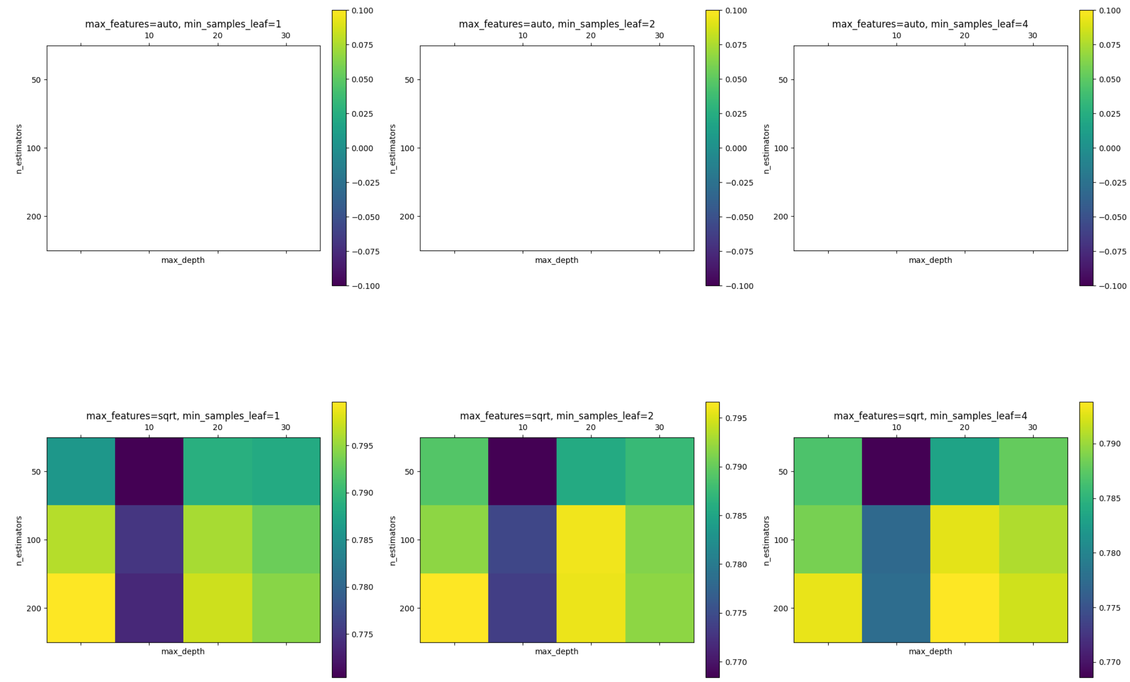
```
C:\Users\10754\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\model_selection\_validation.py:425: FitFailedWarning:
324 fits failed out of a total of 648.
The score on these train-test partitions for these parameters will be set to
nan.
If these failures are not expected, you can try to debug them by setting
error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
61 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\10754\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\model_selection\_validation.py", line 732, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\10754\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\base.py", line 1144, in wrapper
    estimator._validate_params()
  File "C:\Users\10754\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\base.py", line 637, in _validate_params
    validate_parameter_constraints(
  File "C:\Users\10754\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\utils\_param_validation.py", line 95, in
validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features'
parameter of RandomForestClassifier must be an int in the range [1, inf), a
float in the range (0.0, 1.0], a str among {'log2', 'sqrt'} or None. Got 'auto'
instead.

--------------------------------------------------------------------------------
263 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\10754\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\model_selection\_validation.py", line 732, in _fit_and_score
```

```
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\10754\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\base.py", line 1144, in wrapper
    estimator._validate_params()
  File "C:\Users\10754\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\base.py", line 637, in _validate_params
    validate_parameter_constraints(
  File "C:\Users\10754\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\utils\_param_validation.py", line 95, in
validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features'
parameter of RandomForestClassifier must be an int in the range [1, inf), a
float in the range (0.0, 1.0], a str among {'sqrt', 'log2'} or None. Got 'auto'
instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
C:\Users\10754\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\model_selection\_search.py:976: UserWarning: One or more of the
test scores are non-finite: [       nan        nan        nan        nan
nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan 0.7886005  0.8042005  0.79880038
 0.79280002 0.7980003  0.79880014 0.78580106 0.79639858 0.79960166
 0.78999938 0.80040054 0.79640026 0.79219954 0.79440162 0.7976005
 0.78899982 0.7918013  0.79660034 0.78639974 0.78880094 0.7936001
 0.78540006 0.79440078 0.79720082 0.78680134 0.78860122 0.79299962
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan 0.7704003  0.7695995  0.7800003
 0.76999942 0.77679998 0.7782003  0.77039838 0.7750019  0.77379998
 0.76699966 0.77259998 0.77879922 0.77039922 0.7716009  0.77819994
 0.76839998 0.77439986 0.77360038 0.76860078 0.77239966 0.7742011
 0.76939978 0.77499986 0.77600014 0.76859922 0.77720146 0.77760054
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan 0.79060274 0.79260162 0.80420014
 0.78620182 0.79680174 0.79920126 0.78880118 0.79580038 0.79740102
 0.7919997  0.79819906 0.7960001  0.7939993  0.79619922 0.79900094
 0.78560122 0.79600094 0.79580002 0.78240042 0.7886005  0.79360022
 0.78740014 0.79239902 0.7936001  0.78320074 0.7928005  0.79380042
        nan        nan        nan        nan        nan        nan
```

```
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan 0.79280074 0.80199962 0.80039958
 0.79260066 0.79220146 0.79980114 0.78820166 0.79300082 0.79439994
 0.79519954 0.79500078 0.79560126 0.78919966 0.79780094 0.79840106
 0.78740098 0.79140066 0.79180022 0.79159858 0.78759962 0.79279978
 0.78460046 0.78919942 0.79139958 0.78780078 0.79080006 0.79200042]
  warnings.warn(
C:\Users\10754\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\base.py:1151: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  return fit_method(estimator, *args, **kwargs)
```

[121]:
```python
# Create a dict to store scores for each combination of 'max_features' and␣
↪'min_samples_leaf'
heatmap_data_corrected = {}

for i, score in enumerate(grid_result_rf.cv_results_['mean_test_score']):
    params = grid_result_rf.cv_results_['params'][i]
    key = (params['max_features'], params['min_samples_leaf'])
    if key not in heatmap_data_corrected:
        heatmap_data_corrected[key] = np.
↪zeros((len(rf_param_grid['n_estimators']), len(rf_param_grid['max_depth'])))
    heatmap_data_corrected[key][rf_param_grid['n_estimators'].
↪index(params['n_estimators']),
                                rf_param_grid['max_depth'].
↪index(params['max_depth'])] = score

# Plotting
fig, axes = plt.subplots(len(rf_param_grid['max_features']),␣
↪len(rf_param_grid['min_samples_leaf']), figsize=(20, 15))

for i, max_feature in enumerate(rf_param_grid['max_features']):
    for j, min_sample_leaf in enumerate(rf_param_grid['min_samples_leaf']):
        ax = axes[i, j]
        cax = ax.matshow(heatmap_data_corrected[(max_feature,␣
↪min_sample_leaf)], cmap="viridis")
        plt.colorbar(cax, ax=ax, orientation="vertical", fraction=0.046, pad=0.
↪04)
        ax.set_xticks(range(len(rf_param_grid['max_depth'])))
        ax.set_yticks(range(len(rf_param_grid['n_estimators'])))
        ax.set_xticklabels(rf_param_grid['max_depth'])
        ax.set_yticklabels(rf_param_grid['n_estimators'])
        ax.set_title(f"max_features={max_feature},␣
↪min_samples_leaf={min_sample_leaf}")
```

```
        ax.set_xlabel('max_depth')
        ax.set_ylabel('n_estimators')

plt.tight_layout()
plt.show()
```



There are so mant blanks in the 1st row, it means there's no need to set the hyperparameters So just analyse the 2nd row's Here I also fixed max_features and min_samples_leaf it could be ovserved, when n_estimators is 100 or 200, and max_depth is None or 20, the score of rf is high

[117]:
```
print("Best: %f using %s" % (grid_result_rf.best_score_, grid_result_rf.
  ↪best_params_))
```

```
Best: 0.804201 using {'max_depth': None, 'max_features': 'sqrt',
'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
```

6 Final Models

After doing the search, pick up the best parameters and fit models

6.1 Model 1 - Fully Connected Neural Network

[138]:
```
mlp_best = create_mlp_model()
mlp_history = mlp_best.fit(X_train_preprocessed_reshaped, y_train_preprocessed,
                epochs=10, batch_size=32,
```

```
                validation_data=(X_val_preprocessed_reshaped, y_val_preprocessed)
)
```

```
Epoch 1/10
374/374 [==============================] - 8s 21ms/step - loss: 1.1890 -
accuracy: 0.5788 - val_loss: 0.8863 - val_accuracy: 0.6887
Epoch 2/10
374/374 [==============================] - 8s 21ms/step - loss: 0.8782 -
accuracy: 0.6919 - val_loss: 0.8217 - val_accuracy: 0.6945
Epoch 3/10
374/374 [==============================] - 8s 21ms/step - loss: 0.7964 -
accuracy: 0.7195 - val_loss: 0.7855 - val_accuracy: 0.6904
Epoch 4/10
374/374 [==============================] - 8s 21ms/step - loss: 0.7334 -
accuracy: 0.7426 - val_loss: 0.6526 - val_accuracy: 0.7769
Epoch 5/10
374/374 [==============================] - 8s 21ms/step - loss: 0.6871 -
accuracy: 0.7559 - val_loss: 0.6242 - val_accuracy: 0.7798
Epoch 6/10
374/374 [==============================] - 8s 20ms/step - loss: 0.6546 -
accuracy: 0.7680 - val_loss: 0.6988 - val_accuracy: 0.7225
Epoch 7/10
374/374 [==============================] - 8s 21ms/step - loss: 0.6272 -
accuracy: 0.7751 - val_loss: 0.5998 - val_accuracy: 0.7891
Epoch 8/10
374/374 [==============================] - 8s 21ms/step - loss: 0.5806 -
accuracy: 0.7956 - val_loss: 0.5351 - val_accuracy: 0.8096
Epoch 9/10
374/374 [==============================] - 8s 21ms/step - loss: 0.5712 -
accuracy: 0.7950 - val_loss: 0.5318 - val_accuracy: 0.8189
Epoch 10/10
374/374 [==============================] - 8s 21ms/step - loss: 0.5582 -
accuracy: 0.7986 - val_loss: 0.5723 - val_accuracy: 0.7961
```

6.2 Model 2 - Convolutional Neural Network

[134]:
```
cnn_best = create_cnn_model(lr=2e-4, activation='tanh')
cnn_history = cnn_best.fit(X_train_preprocessed, y_train_preprocessed,
                epochs=10, batch_size=32,
                validation_data=(X_val_preprocessed, y_val_preprocessed)
)
```

```
Epoch 1/10
374/374 [==============================] - 6s 16ms/step - loss: 1.1244 -
accuracy: 0.6169 - val_loss: 0.7456 - val_accuracy: 0.7342
Epoch 2/10
374/374 [==============================] - 6s 15ms/step - loss: 0.6842 -
accuracy: 0.7670 - val_loss: 0.6030 - val_accuracy: 0.7734
```

```
Epoch 3/10
374/374 [==============================] - 6s 15ms/step - loss: 0.5825 -
accuracy: 0.7999 - val_loss: 0.5082 - val_accuracy: 0.8277
Epoch 4/10
374/374 [==============================] - 6s 15ms/step - loss: 0.5164 -
accuracy: 0.8236 - val_loss: 0.4658 - val_accuracy: 0.8335
Epoch 5/10
374/374 [==============================] - 6s 15ms/step - loss: 0.4759 -
accuracy: 0.8375 - val_loss: 0.4414 - val_accuracy: 0.8528
Epoch 6/10
374/374 [==============================] - 6s 15ms/step - loss: 0.4420 -
accuracy: 0.8514 - val_loss: 0.4049 - val_accuracy: 0.8727
Epoch 7/10
374/374 [==============================] - 6s 15ms/step - loss: 0.4154 -
accuracy: 0.8594 - val_loss: 0.3974 - val_accuracy: 0.8639
Epoch 8/10
374/374 [==============================] - 6s 15ms/step - loss: 0.3926 -
accuracy: 0.8692 - val_loss: 0.3627 - val_accuracy: 0.8744
Epoch 9/10
374/374 [==============================] - 6s 15ms/step - loss: 0.3686 -
accuracy: 0.8767 - val_loss: 0.3556 - val_accuracy: 0.8797
Epoch 10/10
374/374 [==============================] - 6s 15ms/step - loss: 0.3507 -
accuracy: 0.8815 - val_loss: 0.3517 - val_accuracy: 0.8732
```

6.3 Model 3 - Algorithm Choice 1: SVM

[127]:
```python
svc_best = SVC(C=10, kernel='rbf', gamma='scale')
svc_best.fit(X_train_preprocessed_reshaped, y_train)
```

```
C:\Users\10754\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\utils\validation.py:1184: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

[127]:
```
SVC(C=10)
```

6.4 Model 4 - Algorithm Choice 2: Random Forest

[128]:
```python
rf_best = RandomForestClassifier(n_estimators=100, max_features='sqrt',␣
 ↪min_samples_leaf=1, min_samples_split=2)
rf_best.fit(X_train_preprocessed_reshaped, y_train)
```

```
C:\Users\10754\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\base.py:1151: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
```

```
  return fit_method(estimator, *args, **kwargs)
```

[128]: RandomForestClassifier()

[140]:
```
mlp_predictions = np.argmax(mlp_best.predict(X_test_preprocessed_reshaped),␣
 ↪axis=1)
cnn_predictions = np.argmax(cnn_best.predict(X_test_preprocessed), axis=1)
svc_predictions = svc_best.predict(X_test_preprocessed_reshaped.reshape(-1,␣
 ↪28*28*3))
rf_predictions = rf_best.predict(X_test_preprocessed_reshaped.reshape(-1,␣
 ↪28*28*3))
```

```
107/107 [==============================] - 0s 4ms/step
107/107 [==============================] - 1s 5ms/step
```

[141]:
```
# Calculate accuracy for each model
mlp_accuracy = accuracy_score(mlp_predictions, y_test)
cnn_accuracy = accuracy_score(cnn_predictions,y_test)
svc_accuracy = accuracy_score(svc_predictions,y_test)
rf_accuracy = accuracy_score(rf_predictions,y_test)

print(f"MLP Accuracy: {mlp_accuracy}")
print(f"CNN Accuracy: {cnn_accuracy}")
print(f"SVC Accuracy: {svc_accuracy}")
print(f"Random Forest Accuracy: {rf_accuracy}")
```

```
MLP Accuracy: 0.77491961414791
CNN Accuracy: 0.8693364513300205
SVC Accuracy: 0.8857059339374452
Random Forest Accuracy: 0.8319204910844782
```

As we can see that, SVC is the best one, and then will be CNN and RF, and finally, mlp

[ ]:
```

```