Background on Data

The data provided for the purpose of this assignment came from the Kannada MNIST dataset, provide by Kaggle for competition purposes. It is a dataset comprised of images of number for which the goal is to be able to accurately classify them. This is a large dataset comprised of 784 features to be used, with each feature containing 6,000 instances.

Data Preparation

For this assignment, data preparation was very minimal as this assignment entailed the creation of a neural network for classification purposes. However, with neural networks, it is necessary to create a training and validation set. I used the MNIST training set to create the initial X and Y training sets. The validation sets were then comprised of the first and last 5000 instances of the aforementioned training set.

Results/Evaluation

MNIST Neural Networks: Optimizer 1

Number of Hidden Layers	Number of Hidden Nodes	Total Training Time (epochs = 10)	Optimizer	Training Accuracy	Validation Accuracy	Test Accuracy
1	100	48 s	ReLu	0.9513	0.9290	0.8644
1	25	50.9 s	ReLu	0.9462	0.9240	0.8584
3	100	50.9 s	ReLu	0.9569	0.9300	0.8770
3	25	52.1 s	ReLu	0.9293	0.9134	0.8572

MNIST Neural Networks: Optimizer 2

Number of Hidden Layers	Number of Hidden Nodes	Total Training Time (epochs = 10)	Optimizer	Training Accuracy	Validation Accuracy	Test Accuracy
3	100	1.1 min	RMSprop	0.9936	0.9486	0.9248
3	25	1 min	RMSprop	0.9969	0.9548	0.9354
5	100	1.6 min	RMSprop	0.9908	0.9512	0.9258
5	25	1.6 min	RMSprop	0.9956	0.9586	0.9324

I produced two tables, one of which used the ReLu optimizer, while the other used the RMSprop optimizer. I did this to increase the accuracy of my model while also wanting to compare the differences between the 2 optimizers. I also increased the number of hidden layers in the latter table to see if that also made a difference.

Model Performance and Recommendation

As it pertains to the performance of the model clearly the neural networks built in the second model proved to be more successful as can be seen by test accuracy. Even when comparing the performance of both models, that used 3 hidden layers, the change in optimizer made a significant difference. From a performance perspective, I would highly suggest adjusting the optimizers (including the optimizer parameter of learning rate), number of hidden layers and number nodes per layer. I believe these can all contribute to the accuracy of a model on unseen data.

Appendix

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/c
ontent/drive", force_remount=True).
```

In [ ]:
```
import sklearn
import seaborn as sns
import time
import pandas as pd
import numpy as np
import os
import tensorflow as tf
from tensorflow import keras
from keras.layers import Dense
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense
import datetime
from sklearn.manifold import LocallyLinearEmbedding
from sklearn.mixture import GaussianMixture
from scipy import stats
from sklearn.cluster import MiniBatchKMeans
from sklearn.metrics import silhouette_score
from sklearn.metrics import silhouette_samples
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
from IPython.core.interactiveshell import InteractiveShell
from sklearn.linear_model import RidgeCV, ElasticNetCV, LassoCV
from sklearn.decomposition import PCA #Principal Component Analysis
from sklearn.decomposition import IncrementalPCA
from sklearn.decomposition import KernelPCA
from sklearn.preprocessing import LabelEncoder, StandardScaler #transforms categorical into
 numbers
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import mean_squared_error, make_scorer, f1_score, confusion_matrix
import xgboost
from sklearn.cluster import KMeans


%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.ticker import FixedLocator, FixedFormatter
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
```

In [ ]:
```
MNIST_train = pd.read_csv('train.csv')
MNIST_test = pd.read_csv('test.csv')
```

In [ ]:
```
MNIST_train.head()
```

Out[ ]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel11	pixel12	pixel13	pixel14
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

5 rows × 785 columns

In [ ]:
```
print(MNIST_train.shape)
print(MNIST_test.shape)
```

```
(60000, 785)
(5000, 785)
```

In [ ]:
```
MNIST_test.head()
```

Out[ ]:

	id	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel11	pixel12	pixel13	pixel14	p
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
3	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

5 rows × 785 columns

In [ ]:
```
X_train, y_train = MNIST_train.drop('label', axis=1), MNIST_train['label']

X_test=MNIST_test.drop('id', axis=1)
```

In [ ]:
```
X_train.shape
```

Out[ ]:
```
(60000, 784)
```

In [ ]:
```
X_train = X_train / 255.0
X_test = X_test / 255.0
X_valid, X_train = X_train[:5000], X_train[5000:]
y_valid, y_train = y_train[:5000], y_train[5000:]

tf.random.set_seed(42)
np.random.seed(42)
```

In [ ]:
```
# Define Sequential model with 3 layers

model = keras.Sequential(
  [
    layers.Dense(784, activation="relu", name="layer1"), #784 Nodes
    layers.Dense(100, activation="relu", name="layer2"), #100 hidden nodes, 1 hidden layer
    layers.Dense(10, activation= 'softmax', name="layer3"),
  ]
)

model.compile(loss="sparse_categorical_crossentropy",
             optimizer=keras.optimizers.SGD(lr=1e-3),
             metrics=["accuracy"])
```

In [ ]:
```
%time history = model.fit(X_train, y_train, epochs=10, validation_data=(X_valid, y_valid))
```

```
Epoch 1/10
1719/1719 [==============================] - 4s 2ms/step - loss: 2.0470 - accuracy: 0.4228 -
val_loss: 1.2145 - val_accuracy: 0.8434
Epoch 2/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.9866 - accuracy: 0.8732 -
val_loss: 0.6251 - val_accuracy: 0.8846
Epoch 3/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.5244 - accuracy: 0.9060 -
val_loss: 0.4538 - val_accuracy: 0.8992
Epoch 4/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.3721 - accuracy: 0.9210 -
val_loss: 0.3819 - val_accuracy: 0.9102
Epoch 5/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.2972 - accuracy: 0.9322 -
val_loss: 0.3457 - val_accuracy: 0.9168
Epoch 6/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.2573 - accuracy: 0.9387 -
val_loss: 0.3239 - val_accuracy: 0.9208
Epoch 7/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.2310 - accuracy: 0.9443 -
val_loss: 0.3079 - val_accuracy: 0.9240
Epoch 8/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.2114 - accuracy: 0.9471 -
val_loss: 0.2965 - val_accuracy: 0.9266
Epoch 9/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.1943 - accuracy: 0.9502 -
val_loss: 0.2896 - val_accuracy: 0.9280
Epoch 10/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.1860 - accuracy: 0.9513 -
val_loss: 0.2839 - val_accuracy: 0.9290
CPU times: user 42.9 s, sys: 5.13 s, total: 48 s
Wall time: 37.9 s
```

In [ ]:
```
y_pred = model.predict(X_test)
print(y_pred)
```

```
[[7.82915522e-05 5.38783206e-05 3.69442423e-04 ... 6.62148884e-03
  4.72776861e-07 5.76562743e-05]
 [9.95218575e-01 2.10866076e-03 7.97443790e-05 ... 8.56724364e-05
  1.46563724e-03 9.73540184e-04]
 [5.02179086e-04 7.53412722e-03 9.53769088e-01 ... 1.00363141e-04
  1.27425045e-03 9.04336703e-05]
 ...
 [2.69343075e-03 9.95941460e-01 2.26782562e-04 ... 1.32610703e-05
  1.13286165e-04 2.16884189e-04]
 [1.42008741e-03 2.57167779e-03 3.88672088e-05 ... 1.40739888e-01
  1.35658775e-03 4.81155608e-03]
 [3.26120062e-03 5.40218875e-03 4.05112207e-02 ... 1.48650229e-01
  1.87951387e-04 1.58611089e-02]]
```

In [ ]:
```
y_pred.shape
```

Out[ ]:
```
(5000, 10)
```

In [ ]:
```
np.argmax(y_pred, axis=1) #these are the predicted classes for each instance in test set
```

Out[ ]:
```
array([3, 0, 2, ..., 1, 6, 3])
```

In [ ]:
```
#SAVE ABOVE np.argmax AS CSV
```

In [ ]:
```
model_25 = keras.Sequential(
  [
    layers.Dense(784, activation="relu", name="layer1"), #784 Nodes
    layers.Dense(25, activation="relu", name="layer2"), #Changes from 100 to 25 nodes in mid
dle layer
    layers.Dense(10, activation= 'softmax', name="layer3"),
  ]
)

model_25.compile(loss="sparse_categorical_crossentropy",
             optimizer=keras.optimizers.SGD(lr=1e-3),
             metrics=["accuracy"])
```

In [ ]:
```
%time history = keras.Sequential(
  [
    layers.Dense(784, activation="relu", name="layer1"), #784 Nodes
    layers.Dense(25, activation="relu", name="layer2"), #Changes from 100 to 25 nodes in mid
dle layer
    layers.Dense(10, activation= 'softmax', name="layer3"),
  ]
)

model_25.compile(loss="sparse_categorical_crossentropy",
             optimizer=keras.optimizers.SGD(lr=1e-3),
             metrics=["accuracy"])
```

```
Epoch 1/10
1719/1719 [==============================] - 5s 2ms/step - loss: 2.0146 - accuracy: 0.3666 -
val_loss: 1.2358 - val_accuracy: 0.7644
Epoch 2/10
1719/1719 [==============================] - 4s 2ms/step - loss: 1.0092 - accuracy: 0.8475 -
val_loss: 0.7062 - val_accuracy: 0.8720
Epoch 3/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.5799 - accuracy: 0.8946 -
val_loss: 0.5244 - val_accuracy: 0.8922
Epoch 4/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.4199 - accuracy: 0.9104 -
val_loss: 0.4409 - val_accuracy: 0.9042
Epoch 5/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.3359 - accuracy: 0.9238 -
val_loss: 0.3965 - val_accuracy: 0.9110
Epoch 6/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.2891 - accuracy: 0.9313 -
val_loss: 0.3676 - val_accuracy: 0.9152
Epoch 7/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.2554 - accuracy: 0.9377 -
val_loss: 0.3460 - val_accuracy: 0.9184
Epoch 8/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.2332 - accuracy: 0.9406 -
val_loss: 0.3298 - val_accuracy: 0.9204
Epoch 9/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.2128 - accuracy: 0.9450 -
val_loss: 0.3196 - val_accuracy: 0.9226
Epoch 10/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.2029 - accuracy: 0.9462 -
val_loss: 0.3112 - val_accuracy: 0.9240
CPU times: user 45.6 s, sys: 5.3 s, total: 50.9 s
Wall time: 40.3 s
```

In [ ]:
```
y_pred_25 = model_25.predict(X_test)
print(y_pred_25)
print(y_pred_25.shape)
np.argmax(y_pred_25, axis=1) #these are the predicted classes for each instance in test set
```

```
[[6.26811307e-05 3.17284779e-04 6.31646137e-04 ... 1.03583783e-02
  7.79210040e-07 1.04003375e-04]
 [9.86058176e-01 6.73264870e-03 2.12393697e-05 ... 1.83389027e-04
  5.53293107e-03 1.07472145e-03]
 [9.67629821e-05 1.35268236e-03 9.43647563e-01 ... 1.95494550e-03
  4.51850425e-03 1.33290282e-03]
 ...
 [8.01860634e-03 9.70373154e-01 5.25576936e-04 ... 1.44227024e-03
  1.11914182e-03 1.01954332e-02]
 [2.71674036e-03 1.28639312e-02 5.55170584e-04 ... 1.50052458e-01
  7.46092468e-04 2.06199801e-03]
 [1.31201115e-03 6.06477493e-03 4.71229516e-02 ... 2.18631133e-01
  1.73219742e-04 1.27421478e-02]]
(5000, 10)
```

Out[ ]:
```
array([3, 0, 2, ..., 1, 6, 3])
```

In [ ]:
```
#SAVE ABOVE np.argmax AS CSV
```

In [ ]:
```
model_2_25 = keras.Sequential(
  [
    layers.Dense(784, activation="relu", name="layer1"), #784 Nodes
    layers.Dense(25, activation="relu", name="layer2"),
    layers.Dense(25, activation="relu", name="layer2_2"),
    layers.Dense(25, activation="relu", name="layer2_3"),
    layers.Dense(10, activation= 'softmax', name="layer3"),
  ]
)

model_2_25.compile(loss="sparse_categorical_crossentropy",
             optimizer=keras.optimizers.SGD(lr=1e-3),
             metrics=["accuracy"])
```

In [ ]:
```
%time history = model_2_25.fit(X_train, y_train, epochs=10, validation_data=(X_valid, y_vali
d))
```

```
Epoch 1/10
1719/1719 [==============================] - 5s 3ms/step - loss: 2.2654 - accuracy: 0.1083 -
val_loss: 2.1603 - val_accuracy: 0.1160
Epoch 2/10
1719/1719 [==============================] - 4s 2ms/step - loss: 2.1167 - accuracy: 0.1661 -
val_loss: 1.9634 - val_accuracy: 0.2690
Epoch 3/10
1719/1719 [==============================] - 4s 2ms/step - loss: 1.8766 - accuracy: 0.4002 -
val_loss: 1.5300 - val_accuracy: 0.5836
Epoch 4/10
1719/1719 [==============================] - 4s 2ms/step - loss: 1.3652 - accuracy: 0.6684 -
val_loss: 0.9423 - val_accuracy: 0.7290
Epoch 5/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.7934 - accuracy: 0.8019 -
val_loss: 0.5804 - val_accuracy: 0.8572
Epoch 6/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.4814 - accuracy: 0.8836 -
val_loss: 0.4699 - val_accuracy: 0.8826
Epoch 7/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.3678 - accuracy: 0.9022 -
val_loss: 0.4102 - val_accuracy: 0.8962
Epoch 8/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.3116 - accuracy: 0.9148 -
val_loss: 0.3867 - val_accuracy: 0.9018
Epoch 9/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.2707 - accuracy: 0.9249 -
val_loss: 0.3616 - val_accuracy: 0.9118
Epoch 10/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.2517 - accuracy: 0.9293 -
val_loss: 0.3536 - val_accuracy: 0.9134
CPU times: user 46.5 s, sys: 5.69 s, total: 52.1 s
Wall time: 41.9 s
```

In [ ]:
```
y_pred_2_25 = model_2_25.predict(X_test)
print(y_pred_2_25)
print(y_pred_2_25.shape)
np.argmax(y_pred_2_25, axis=1) #these are the predicted classes for each instance in test se
t
```

```
[[1.1075933e-04 4.4466610e-06 6.2838639e-03 ... 1.8720099e-03
  4.6364907e-08 2.4146860e-04]
 [9.9339044e-01 1.5360003e-03 4.9452142e-06 ... 1.0978460e-07
  4.4316701e-03 5.6911039e-04]
 [3.1756343e-05 1.1568510e-05 9.9156618e-01 ... 8.5373489e-05
  3.8063864e-04 1.2837730e-04]
 ...
 [1.1989583e-03 9.7436804e-01 7.5226570e-05 ... 1.4024148e-05
  2.4100456e-03 1.7909793e-02]
 [4.1493342e-05 1.2306364e-04 3.8276507e-06 ... 2.0905353e-01
  1.6229799e-04 6.8200422e-03]
 [2.4331945e-03 4.1812556e-04 1.9626804e-02 ... 8.7630369e-02
  2.1020674e-05 1.4627339e-02]]
(5000, 10)
```

Out[ ]:
```
array([3, 0, 2, ..., 1, 6, 3])
```

In [ ]:
```
#SAVE ABOVE AS CSV
```

In [ ]:
```
model_2_100 = keras.Sequential(
  [
    layers.Dense(784, activation="relu", name="layer1"), #784 Nodes
    layers.Dense(100, activation="relu", name="layer2"),
    layers.Dense(100, activation="relu", name="layer2_2"),
    layers.Dense(100, activation="relu", name="layer2_3"),
    layers.Dense(10, activation= 'softmax', name="layer3"),
  ]
)

model_2_100.compile(loss="sparse_categorical_crossentropy",
             optimizer=keras.optimizers.SGD(lr=1e-3),
             metrics=["accuracy"])
```

In [ ]:
```
%time history = model_2_100.fit(X_train, y_train, epochs=10, validation_data=(X_valid, y_val
id))
```

```
Epoch 1/10
1719/1719 [==============================] - 5s 2ms/step - loss: 2.2579 - accuracy: 0.1696 -
val_loss: 1.9732 - val_accuracy: 0.4606
Epoch 2/10
1719/1719 [==============================] - 4s 2ms/step - loss: 1.7065 - accuracy: 0.6426 -
val_loss: 0.8514 - val_accuracy: 0.8638
Epoch 3/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.6585 - accuracy: 0.8851 -
val_loss: 0.4764 - val_accuracy: 0.8890
Epoch 4/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.3612 - accuracy: 0.9177 -
val_loss: 0.3796 - val_accuracy: 0.9062
Epoch 5/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.2623 - accuracy: 0.9342 -
val_loss: 0.3444 - val_accuracy: 0.9146
Epoch 6/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.2208 - accuracy: 0.9414 -
val_loss: 0.3250 - val_accuracy: 0.9196
Epoch 7/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.1939 - accuracy: 0.9478 -
val_loss: 0.3071 - val_accuracy: 0.9238
Epoch 8/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.1750 - accuracy: 0.9504 -
val_loss: 0.2967 - val_accuracy: 0.9266
Epoch 9/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.1583 - accuracy: 0.9553 -
val_loss: 0.2893 - val_accuracy: 0.9292
Epoch 10/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.1510 - accuracy: 0.9569 -
val_loss: 0.2860 - val_accuracy: 0.9300
CPU times: user 45.6 s, sys: 5.23 s, total: 50.9 s
Wall time: 40.5 s
```

In [ ]:
```
y_pred_2_100 = model_2_100.predict(X_test)
print(y_pred_2_100)
print(y_pred_2_100.shape)
np.argmax(y_pred_2_100, axis=1) #these are the predicted classes for each instance in test s
et
```

```
[[2.9006733e-05 1.2193653e-05 2.4667667e-04 ... 4.3842616e-03
  1.4061388e-07 2.5092022e-06]
 [9.9682581e-01 1.6541600e-03 1.8947580e-05 ... 4.4810040e-05
  8.8215841e-04 5.6080299e-04]
 [6.7755078e-05 7.2706089e-04 9.7068936e-01 ... 4.6954956e-05
  5.6141318e-04 4.0946248e-05]
 ...
 [2.9262768e-03 9.9100566e-01 9.9146680e-05 ... 1.3576997e-04
  6.7136023e-04 1.9035401e-03]
 [3.9726199e-04 2.2452166e-03 3.4476557e-07 ... 8.5847765e-02
  7.4017141e-04 8.4708980e-04]
 [4.1686068e-03 6.6827778e-03 3.2401767e-02 ... 2.3307298e-01
  1.9379286e-04 7.2839856e-03]]
(5000, 10)
```

Out[ ]:
```
array([3, 0, 2, ..., 1, 6, 3])
```

DIFFERENT OPTIMIZER

In [ ]:
```
model_2_100_opt2 = keras.Sequential(
  [
    layers.Dense(784, activation="relu", name="layer1"), #784 Nodes
    layers.Dense(100, activation="relu", name="layer2"),
    layers.Dense(100, activation="relu", name="layer2_2"),
    layers.Dense(100, activation="relu", name="layer2_3"),
    layers.Dense(10, activation= 'softmax', name="layer3"),
  ]
)

model_2_100_opt2.compile(loss="sparse_categorical_crossentropy",
             optimizer = keras.optimizers.RMSprop(lr=0.001, rho=0.9),
             metrics=["accuracy"])
```

In [ ]:
```
%time history = model_2_100_opt2.fit(X_train, y_train, epochs=10, validation_data=(X_valid,
 y_valid))
```

```
Epoch 1/10
1719/1719 [==============================] - 6s 3ms/step - loss: 0.2420 - accuracy: 0.9247 -
val_loss: 0.2492 - val_accuracy: 0.9478
Epoch 2/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0557 - accuracy: 0.9863 -
val_loss: 0.3961 - val_accuracy: 0.9404
Epoch 3/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0449 - accuracy: 0.9912 -
val_loss: 0.2823 - val_accuracy: 0.9578
Epoch 4/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0415 - accuracy: 0.9919 -
val_loss: 0.3025 - val_accuracy: 0.9524
Epoch 5/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0454 - accuracy: 0.9926 -
val_loss: 0.6099 - val_accuracy: 0.9512
Epoch 6/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0418 - accuracy: 0.9929 -
val_loss: 0.7108 - val_accuracy: 0.9470
Epoch 7/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0488 - accuracy: 0.9922 -
val_loss: 0.5294 - val_accuracy: 0.9566
Epoch 8/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0417 - accuracy: 0.9936 -
val_loss: 0.9088 - val_accuracy: 0.9590
Epoch 9/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0401 - accuracy: 0.9947 -
val_loss: 1.0024 - val_accuracy: 0.9516
Epoch 10/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0408 - accuracy: 0.9936 -
val_loss: 0.8081 - val_accuracy: 0.9486
CPU times: user 55.8 s, sys: 5.51 s, total: 1min 1s
Wall time: 49 s
```

In [ ]:
```
y_pred_2_100_opt2= model_2_100_opt2.predict(X_test)
print(y_pred_2_100_opt2)
print(y_pred_2_100_opt2.shape)
np.argmax(y_pred_2_100_opt2, axis=1) #these are the predicted classes for each instance in t
est set
```

```
[[1.0705646e-33 0.0000000e+00 0.0000000e+00 ... 7.5464782e-28
  0.0000000e+00 0.0000000e+00]
 [1.0000000e+00 5.1885558e-27 5.6998861e-26 ... 2.2738344e-24
  2.0118445e-13 2.6991816e-26]
 [0.0000000e+00 0.0000000e+00 1.0000000e+00 ... 0.0000000e+00
  0.0000000e+00 0.0000000e+00]
 ...
 [5.7767937e-23 1.0000000e+00 0.0000000e+00 ... 5.3364084e-30
  2.5481179e-32 0.0000000e+00]
 [2.5297875e-16 5.5446489e-17 3.0502633e-22 ... 6.7544702e-06
  8.6908446e-23 7.1782871e-11]
 [4.3879239e-13 5.7042450e-23 8.7623615e-19 ... 7.0257321e-11
  1.2877200e-21 2.0377031e-21]]
(5000, 10)
```

Out[ ]:
```
array([3, 0, 2, ..., 1, 6, 3])
```

In [ ]:
```
model_2_25_opt2 = keras.Sequential(
  [
    layers.Dense(784, activation="relu", name="layer1"), #784 Nodes
    layers.Dense(25, activation="relu", name="layer2"),
    layers.Dense(25, activation="relu", name="layer2_2"),
    layers.Dense(25, activation="relu", name="layer2_3"),
    layers.Dense(10, activation= 'softmax', name="layer3"),
  ]
)

model_2_25_opt2.compile(loss="sparse_categorical_crossentropy",
             optimizer = keras.optimizers.RMSprop(lr=0.001, rho=0.9),
             metrics=["accuracy"])
```

In [ ]:
```
%time history = model_2_25_opt2.fit(X_train, y_train, epochs=10, validation_data=(X_valid, y
_valid))
```

```
Epoch 1/10
1719/1719 [==============================] - 6s 3ms/step - loss: 0.3225 - accuracy: 0.9009 -
val_loss: 0.2650 - val_accuracy: 0.9478
Epoch 2/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0541 - accuracy: 0.9860 -
val_loss: 0.2861 - val_accuracy: 0.9410
Epoch 3/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0373 - accuracy: 0.9901 -
val_loss: 0.2626 - val_accuracy: 0.9550
Epoch 4/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0299 - accuracy: 0.9926 -
val_loss: 0.2932 - val_accuracy: 0.9544
Epoch 5/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0244 - accuracy: 0.9942 -
val_loss: 0.3643 - val_accuracy: 0.9548
Epoch 6/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0226 - accuracy: 0.9954 -
val_loss: 0.4322 - val_accuracy: 0.9536
Epoch 7/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0175 - accuracy: 0.9961 -
val_loss: 0.4481 - val_accuracy: 0.9594
Epoch 8/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0172 - accuracy: 0.9961 -
val_loss: 0.5016 - val_accuracy: 0.9592
Epoch 9/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0139 - accuracy: 0.9973 -
val_loss: 0.5783 - val_accuracy: 0.9602
Epoch 10/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0143 - accuracy: 0.9969 -
val_loss: 0.6497 - val_accuracy: 0.9548
CPU times: user 55.1 s, sys: 5.5 s, total: 1min
Wall time: 48.2 s
```

In [ ]:
```
y_pred_2_25_opt2 = model_2_25_opt2.predict(X_test)
print(y_pred_2_25_opt2)
print(y_pred_2_25_opt2.shape)
np.argmax(y_pred_2_25_opt2, axis=1) #these are the predicted classes for each instance in te
st set
```

```
[[3.0619941e-11 1.3670253e-23 6.8690784e-20 ... 3.2319890e-11
  2.5094984e-21 8.1621527e-27]
 [1.0000000e+00 6.3239810e-15 4.4274383e-27 ... 4.1068786e-21
  2.6502174e-19 6.2929998e-18]
 [4.5895952e-12 4.1906369e-17 1.0000000e+00 ... 6.3751045e-25
  4.3211016e-19 4.7203823e-33]
 ...
 [1.8820134e-06 9.9999809e-01 1.7635190e-17 ... 3.4972837e-28
  2.9465089e-24 1.6217434e-18]
 [1.8023973e-14 1.6340161e-12 2.5050305e-22 ... 1.1479281e-11
  1.3312681e-13 4.0508447e-12]
 [1.1785231e-06 5.4550640e-09 6.9305102e-06 ... 2.1926016e-06
  1.3518715e-10 1.6033536e-12]]
(5000, 10)
```

Out[ ]:
```
array([3, 0, 2, ..., 1, 6, 3])
```

In [ ]:
```
model_2_25_opt2_layer5 = keras.Sequential(
  [
    layers.Dense(784, activation="relu", name="layer1"), #784 Nodes
    layers.Dense(25, activation="relu", name="layer2"),
    layers.Dense(25, activation="relu", name="layer2_2"),
    layers.Dense(25, activation="relu", name="layer2_3"),
    layers.Dense(25, activation="relu", name="layer2_4"),
    layers.Dense(25, activation="relu", name="layer2_5"),
    layers.Dense(10, activation= 'softmax', name="layer3"),
  ]
)

model_2_25_opt2_layer5.compile(loss="sparse_categorical_crossentropy",
             optimizer = keras.optimizers.RMSprop(lr=0.001, rho=0.9),
             metrics=["accuracy"])
```

In [ ]:
```
%time history = model_2_25_opt2_layer5.fit(X_train, y_train, epochs=10, validation_data=(X_v
alid, y_valid))
```

```
Epoch 1/10
1719/1719 [==============================] - 6s 3ms/step - loss: 0.3688 - accuracy: 0.8781 -
val_loss: 0.2805 - val_accuracy: 0.9394
Epoch 2/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0611 - accuracy: 0.9830 -
val_loss: 0.3307 - val_accuracy: 0.9384
Epoch 3/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0430 - accuracy: 0.9895 -
val_loss: 0.2480 - val_accuracy: 0.9574
Epoch 4/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0370 - accuracy: 0.9911 -
val_loss: 0.3962 - val_accuracy: 0.9368
Epoch 5/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0327 - accuracy: 0.9930 -
val_loss: 0.3527 - val_accuracy: 0.9564
Epoch 6/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0328 - accuracy: 0.9935 -
val_loss: 0.3221 - val_accuracy: 0.9530
Epoch 7/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0259 - accuracy: 0.9945 -
val_loss: 0.3429 - val_accuracy: 0.9574
Epoch 8/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0230 - accuracy: 0.9954 -
val_loss: 0.4059 - val_accuracy: 0.9560
Epoch 9/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0251 - accuracy: 0.9953 -
val_loss: 0.5171 - val_accuracy: 0.9538
Epoch 10/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0251 - accuracy: 0.9956 -
val_loss: 0.4688 - val_accuracy: 0.9586
CPU times: user 1min, sys: 5.97 s, total: 1min 6s
Wall time: 53.9 s
```

In [ ]:
```
y_pred_2_25_opt2_layer5 = model_2_25_opt2_layer5.predict(X_test)
print(y_pred_2_25_opt2_layer5)
print(y_pred_2_25_opt2_layer5.shape)
np.argmax(y_pred_2_25_opt2_layer5, axis=1) #these are the predicted classes for each instanc
e in test set
```

```
[[7.1867874e-15 2.2956566e-18 3.8496229e-18 ... 5.9298694e-12
  1.7024296e-27 1.5916893e-34]
 [1.0000000e+00 3.9153833e-24 1.3754389e-27 ... 6.3647435e-28
  9.5146088e-19 4.8698204e-38]
 [8.3773380e-15 3.4186449e-15 1.0000000e+00 ... 1.6892491e-23
  0.0000000e+00 0.0000000e+00]
 ...
 [4.6279893e-05 9.9995375e-01 2.3779719e-16 ... 1.1872189e-18
  7.0044025e-18 4.8840335e-15]
 [3.1742947e-17 1.0741989e-15 4.6924886e-23 ... 5.8311109e-13
  2.5128467e-17 5.9571014e-15]
 [4.8277737e-08 5.0337507e-09 4.5693574e-10 ... 7.7491552e-07
  1.9918240e-13 1.8626774e-17]]
(5000, 10)
```

Out[ ]:
```
array([3, 0, 2, ..., 1, 6, 3])
```

In [ ]:
```
model_2_100_opt2_layer5 = keras.Sequential(
  [
    layers.Dense(784, activation="relu", name="layer1"), #784 Nodes
    layers.Dense(100, activation="relu", name="layer2"),
    layers.Dense(100, activation="relu", name="layer2_2"),
    layers.Dense(100, activation="relu", name="layer2_3"),
    layers.Dense(100, activation="relu", name="layer2_4"),
    layers.Dense(100, activation="relu", name="layer2_5"),
    layers.Dense(10, activation= 'softmax', name="layer3"),
  ]
)

model_2_100_opt2_layer5.compile(loss="sparse_categorical_crossentropy",
             optimizer = keras.optimizers.RMSprop(lr=0.001, rho=0.9),
             metrics=["accuracy"])
```

In [ ]:
```
%time history = model_2_100_opt2_layer5.fit(X_train, y_train, epochs=10, validation_data=(X_
valid, y_valid))
```

```
Epoch 1/10
1719/1719 [==============================] - 6s 3ms/step - loss: 0.2985 - accuracy: 0.9074 -
val_loss: 0.2988 - val_accuracy: 0.9344
Epoch 2/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0658 - accuracy: 0.9846 -
val_loss: 0.3183 - val_accuracy: 0.9478
Epoch 3/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0623 - accuracy: 0.9880 -
val_loss: 0.2698 - val_accuracy: 0.9542
Epoch 4/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0647 - accuracy: 0.9895 -
val_loss: 0.5470 - val_accuracy: 0.9354
Epoch 5/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0796 - accuracy: 0.9897 -
val_loss: 1.0880 - val_accuracy: 0.9508
Epoch 6/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.1180 - accuracy: 0.9894 -
val_loss: 0.4701 - val_accuracy: 0.9420
Epoch 7/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0757 - accuracy: 0.9907 -
val_loss: 0.5719 - val_accuracy: 0.9568
Epoch 8/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0871 - accuracy: 0.9907 -
val_loss: 0.7901 - val_accuracy: 0.9596
Epoch 9/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0910 - accuracy: 0.9919 -
val_loss: 1.8643 - val_accuracy: 0.9428
Epoch 10/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0806 - accuracy: 0.9908 -
val_loss: 1.1824 - val_accuracy: 0.9512
CPU times: user 1min, sys: 5.87 s, total: 1min 6s
Wall time: 53.2 s
```

In [ ]:
```
y_pred_2_100_opt2_layer5= model_2_100_opt2_layer5.predict(X_test)
print(y_pred_2_100_opt2_layer5)
print(y_pred_2_100_opt2_layer5.shape)
np.argmax(y_pred_2_100_opt2_layer5, axis=1) #these are the predicted classes for each instan
ce in test set
```

```
[[2.3567978e-22 0.0000000e+00 2.3646956e-37 ... 2.0915286e-24
  3.7517699e-37 0.0000000e+00]
 [1.0000000e+00 0.0000000e+00 0.0000000e+00 ... 0.0000000e+00
  3.6752400e-34 0.0000000e+00]
 [1.5629971e-31 0.0000000e+00 1.0000000e+00 ... 0.0000000e+00
  0.0000000e+00 0.0000000e+00]
 ...
 [2.5822962e-06 9.9999738e-01 1.9686020e-15 ... 4.7535632e-11
  1.4682948e-10 1.2726593e-16]
 [3.8173408e-22 2.8534778e-22 3.9804455e-37 ... 6.8834018e-12
  6.5883891e-27 3.2822986e-10]
 [1.0207655e-09 6.2484895e-16 4.1444919e-14 ... 1.1519238e-08
  1.4829884e-13 2.4366496e-17]]
(5000, 10)
```

Out[ ]:
```
array([3, 0, 2, ..., 1, 6, 3])
```

Submission and Description	Status	Private Score	Public Score	Use for Final Score
Kannada MNIST ANN_1 Neural Net 4 (version 2/2) 2 minutes ago by Michael Venit Hidden 3, 100 Nodes	Succeeded	0.88620	0.87700	☐
[Deleted Notebook] 20 minutes ago by Michael Venit Hidden 3, 25 Nodes	Succeeded	0.86320	0.85720	☐
[Deleted Notebook] 25 minutes ago by Michael Venit Hidden 1, 25 Nodes	Succeeded	0.87060	0.85840	☐
[Deleted Notebook] an hour ago by Michael Venit Hidden 1, 100 Nodes	Succeeded	0.87500	0.86440	☐

All Successful Selected

Submission and Description	Status	Private Score	Public Score	Use for Final Score
Kannada MNIST ANN_1 Neural Net Opt 2_4 (version 6/6) 5 minutes ago by Michael Venit Hidden 5, 100 Nodes, Optimizer 2	Succeeded	0.91880	0.92580	☐
Kannada MNIST ANN_1 Neural Net Opt 2_3 (version 5/6) 11 minutes ago by Michael Venit Hidden 5, 25 Nodes, Optimizer 2	Succeeded	0.92800	0.93240	☐
Kannada MNIST ANN_1 Neural Net Opt_2_2 (version 4/6) 18 minutes ago by Michael Venit Hidden 3, 25 Nodes, Optimizer 2	Succeeded	0.93600	0.93540	☐
Kannada MNIST ANN_1 Neural Net Opt 2_1 (version 3/6) 25 minutes ago by Michael Venit Hidden 3, 100 Nodes, Optimizer 2	Succeeded	0.92380	0.92480	☐