Neural Networks - Deep Learning

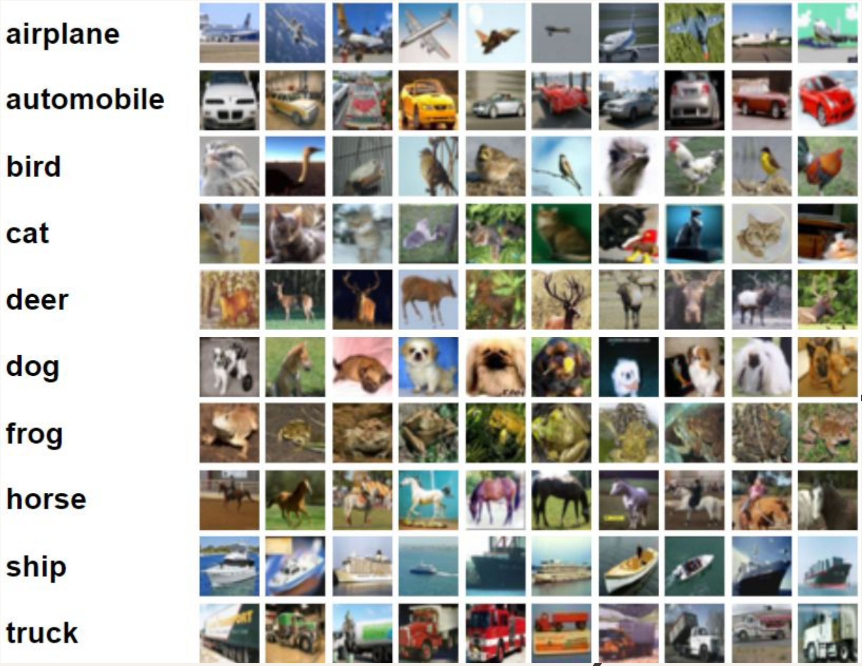
KNN - NCC - MLP - SVM - RBFNN

Alexia Ntantouri (AEM: 3871)

Table of contents

CIFAR-10 Dataset

- **60000 32x32** colour images in **10** classes
- **6000** images per class
- **50000** training images and **10000** test images
- Each image is represented by a vector of **3072** elements, each element represents a pixel
- The elements take values **0-255** and are normalized to **0-1**



01

KNN - NCC

K-nearest neighbors - Nearest Centroid Classifier

My implementation of KNN

```
def myKNN_predictions(X_train, y_train, X_test, k_neighbors):
  # compute the distances
  # each row has the euclidean distances of one sample from the test set to
  # all the samples from the training set
  distances = euclidean_distances(X_test, X_train)

  # sort the distances of each row and retrieve first k indices
  # these indices refer to the k nearest neighbors for each sample of the test set
  indices = np.argsort(distances, axis=1)[:, :k_neighbors]

  # replace indices with labels
  for i in range(len(X_test)):
    for j in range(k_neighbors):
      indices[i][j] = y_train[indices[i][j]]

  # get the label with the most votes for each sample of the test set
  predictions = np.array([np.bincount(row).argmax() for row in indices])

  # return predictions
  return predictions
```

My implementation of NCC

```
def calculate_centroids(X_train, y_train):
  count = np.zeros(10)
  centroids = np.array([ [0.0]*len(X_train[0]) for i in range(10)])

  for i in range(len(y_train)):
    centroids[y_train[i]] += X_train[i]
    count[y_train[i]] += 1

  for i in range(10):
    centroids[i] /= count[i]

  return centroids

def myNCC_predictions(X_train, y_train, X_test):
  # calculate centroids
  centroids = calculate_centroids(X_train, y_train)

  # calculate the distances
  # each row has the euclidean distances of one sample
  # from the test set to all the centroids
  distances = euclidean_distances(X_test, centroids)

  # find the index of minimum distance of each row
  # the index refers to the nearest centroid for each sample of the test set
  # predictions = np.argsort(distances, axis=1)[:, :1]
  predictions = np.argmin(distances, axis=1)

  return predictions
```

KNN vs NCC

	my KNN	
	time (sec)	accuracy
1 neighbor	171.832	0.3539
3 neighbors	144.948	0.3303

	sklearn KNN	
	time (sec)	accuracy
1 neighbor	128.844	0.3539
3 neighbors	109.204	0.3303

my NCC	
time (sec)	accuracy
2.114	0.2774

sklearn NCC	
time (sec)	accuracy
1.023	0.2774

- **KNN 1** a **little bit slower** but **more accurate** than **KNN 3**
- **KNN** significantly **slower** but **more accurate** than **NCC**

02

Multilayer Perceptron

a) Fully Connected MLP

Data Preprocessing

	No of features	No of neurons	No of epochs	Time in seconds	Train accuracy	Test accuracy
Grayscale images without PCA	1024	(100,50)	60	206	48%	41%
Grayscale images with PCA	76	(400,100)	23	22	61%	48%
Original images without PCA	3072	(100,50)	40	241	53%	49%
Original images with PCA	99	(400,100)	18	79	**67%**	**54%**

- **Coloured images** are better than **grayscale images** for training the MLP
- **PCA** is great for **reducing the dimensionality** of the data making the training **faster** and increasing the **accuracy** of the model

a) Fully Connected MLP

Hyperparameter Tuning

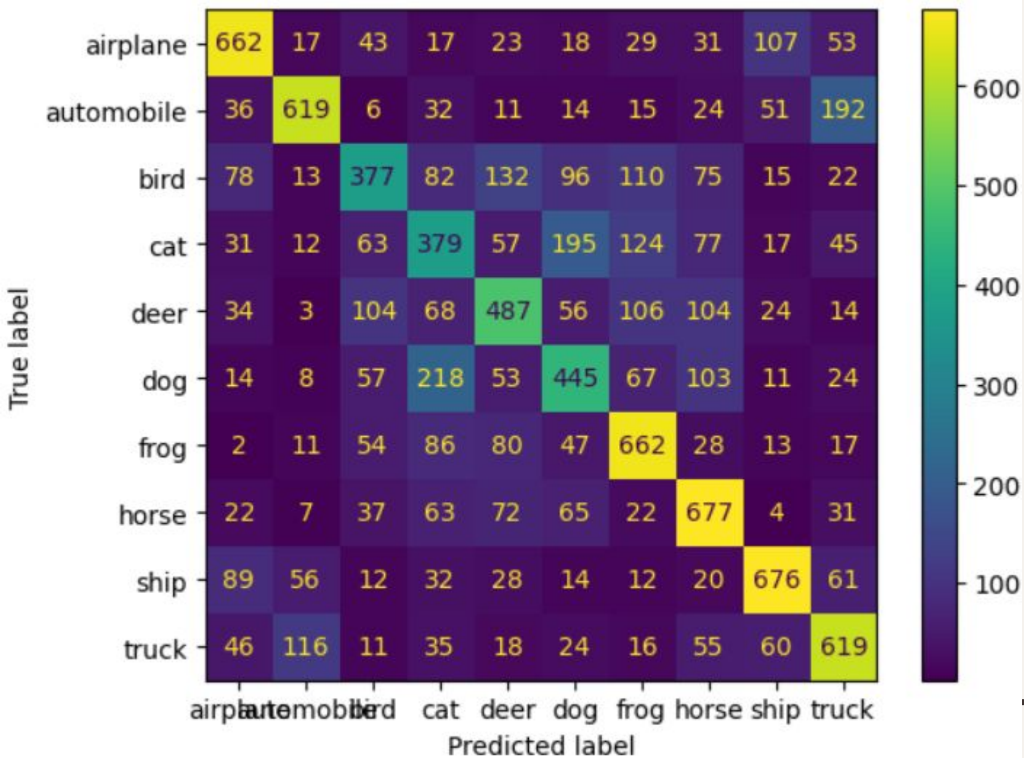
```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

mlp = MLPClassifier(max_iter=300, early_stopping=True)

parameter_grid = {
    'hidden_layer_sizes': [(400,100), (500), (600,200)],
    'activation': ['logistic', 'tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.001],
    'learning_rate': ['constant','adaptive'],
}

# grid_search = GridSearchCV(mlp, parameter_grid, cv=5)
grid_search = RandomizedSearchCV(mlp, parameter_grid, random_state=2, n_iter=5, cv=5, n_jobs=-1)

start = timer()

grid_search.fit(train_pca, y_train.ravel())

end = timer()
print("Time: ", end - start) # time in seconds

print("Best params:")
print(grid_search.best_params_)
```

```
Time:  2740.676056788001
Best params:
{'solver': 'adam', 'learning_rate': 'adaptive', 'hidden_layer_sizes': (600, 200), 'alpha': 0.001, 'activation': 'relu'}
```

- Used sklearn's **RandomizedSearchCV** to find the best hyperparameter values for the model
- Also experimented with different **activation functions**, different **hyperparameter values** and different **architectures** for the MLP

a) Final Fully Connected MLP

Using sklearn's **PCA** and **MLPClassifier**




```
mlp = MLPClassifier(hidden_layer_sizes=(600,200),
                    activation='relu',
                    solver='adam',
                    alpha=0.001,
                    learning_rate='adaptive',
                    early_stopping=True,
                    )
```

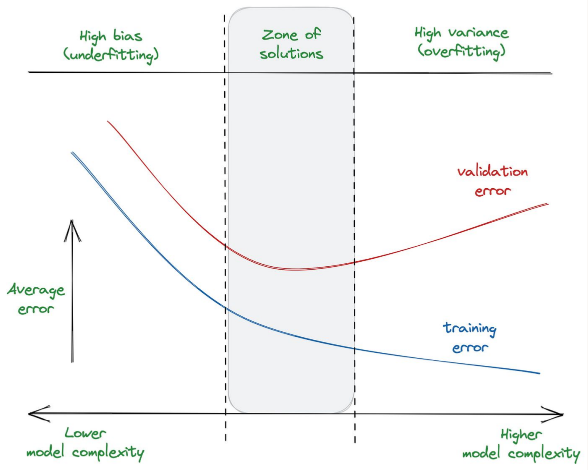
```
Time:  52.91149928399997
Number of iterations with early stopping: 20
Train accuracy: 0.77094
Test accuracy: 0.5603
```

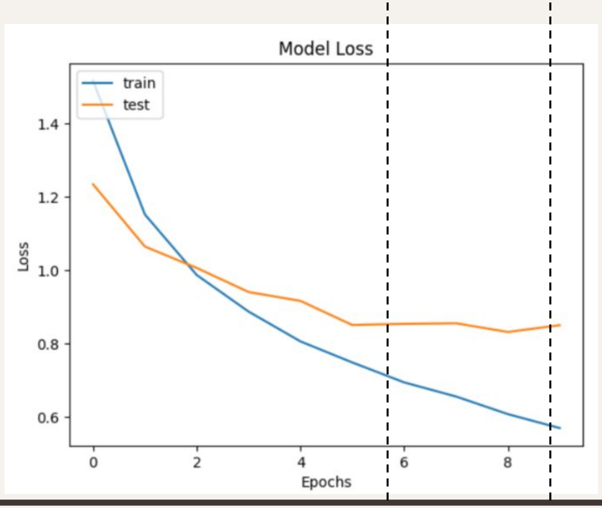
b) Convolutional Neural Network

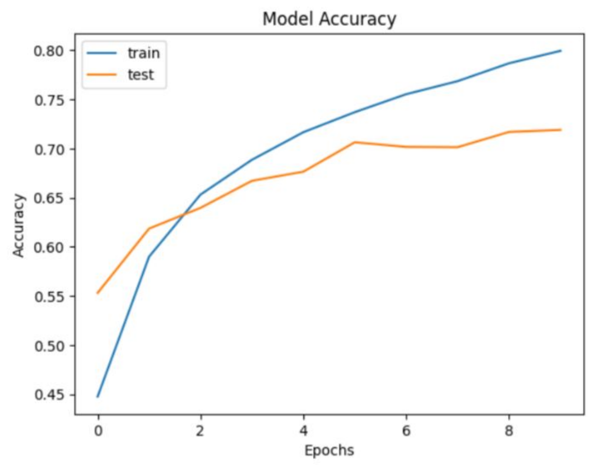
Using **Keras** Library

```
Train accuracy : 0.8401600122451782
```

```
Test accuracy : 0.718999981880188
```







Results Comparison

	NCC	KNN 1	KNN 3	FC MLP	CNN
Training time	0.7	574 (train set) 129 (test set)	556 (train set) 109 (test set)	102	745
Train accuracy	**27%**	**100%**	**58%**	**76%**	**84%**
Test accuracy	**28%**	**35%**	**33%**	**55%**	**72%**

CNNs are highly **effective** for image classification tasks

FC MLP does **better** than **KNN** and **NCC**

03
Support Vector Machine

SVM

Tested

- Different **kernels**
- Different **degrees**
- Different **C** values
- Different **gamma** values
- Different **coef0** values

Also

- **Randomized Search**
- **Voting Ensemble Classifier**
- Passed the **output of the last hidden layer** of the **CNN** to the **SVM**

	Training Time	Train Accuracy	Test Accuracy
Linear	1694.97	0.42032	0.4075
Polynomial	570.09	0.63514	0.4647
Sigmoid	466.35	0.22366	0.2235
RBF	405.14	**0.65472**	**0.5401**

	Training Time	Train Accuracy	Test Accuracy
C = 0.01	645.64	0.35946	0.3644
C = 0.1	412.11	0.47696	0.4605
C = 1 (default)	366.03	0.65472	0.5401
C = 2	361.56	0.73614	0.5536
C = 10	458.86	**0.9216**	**0.5623**
C = 20	567.77	0.96436	0.5555
C = 50	668.88	0.98974	0.5498

	Training Time	Train Accuracy	Test Accuracy
nma = scale 00583) (default)	445.73	0.65472	0.5401
mma = auto .010101)	459.74	0.75644	**0.5523**
amma = 0.02	754.62	**0.89596**	0.5472
amma = 0.1	1774.66	0.99842	0.2694

	Training Time	Train Accuracy	Test Acc
deg = 1	462.56	0.4181	0.4111
deg = 2	438.24	0.54444	**0.4731**
deg = 3 (default)	568.23	**0.63514**	0.4647
deg = 4	702.01	0.60616	0.3919

	Training Time	Train Accuracy	Test Accuracy
coef0 = -1.0	341.85	0.16496	0.1626
coef0 = 0.0 (default)	614.83	0.63514	0.4647
coef0 = 1.0	417.96	0.77496	0.549
oef0 = 2.0	515.31	**0.80234**	**0.5496**
oef0 = 5.0	1005.01	0.84216	0.5414

	Training Time	Train Accuracy	Test Accu
Hard Voting	983.17	0.89246	0.5602
Soft Voting	5053.36	0.90184	0.5695

Results Comparison

	NCC	KNN 1	KNN 3	FC MLP	CNN	SVM	SVM VOTING	CNN passed to SVM
Training time	0.7	574 (train set) 129 (test set)	556 (train set) 109 (test set)	53	745	337	5053	915
Train accuracy	26.968%	100%	57.904%	77.094%	84.02%	86.304%	90.184%	88.128%
Test accuracy	27.74%	35.39%	33.03%	56.03%	71.9%	56.36%	56.95%	72.4%

Combining the **CNN** with the **SVM** gives us the **best** results

SVM Voting is a little bit **better** than the **SVM** but much **slower**

04

RBF Neural Network

RBF Neural Network

Using custom RBF layer for **Keras** Library

	Train Accuracy	Test Accuracy
Neurons = 120	0.4121	0.4067
Neurons = 150	0.4300	0.4269
Neurons = 200	0.4273	0.4261

```
optimizer='adam'
loss='categorical_crossentropy'
betas = 0.1
nodes = 150

model = Sequential()
model.add(Input(shape=train_pca[0].shape))
model.add(RBFLayer(nodes, initializer=InitCentersRandom(train_pca), betas=betas))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
```

```
Train accuracy: 0.4321399927139282
Test accuracy: 0.4241999837875366
```

Results Comparison

	Train Accuracy	Test Accuracy	Training time	
1. CNN + SVM	88.128%	72.4%	915	
2. CNN	84.02%	71.9%	745	
3. SVM Voting	90.184%	56.95%	5053	
4. SVM	86.304%	56.36%	337	
5. FC MLP	77.094%	56.03%	53	
6. RBF NN	44.36%	43.97%	1583	
7. KNN 1	100%	35.39%	574 (train set)	129 (test set)
8. KNN 3	57.904%	33.03%	556 (train set)	109 (test set)
9. NCC	26.968%	27.74%	0.7	

Thank you for your attention!

Alexia Ntantouri (AEM: 3871)

Aristotle University of Thessaloniki

Fall Semester 2023-24

Professor: Anastasios Tefas