ML LAB

ASSIGNMENT 3

NAME: SWAPNIL GHOSH
ROLL: 00191100106 7
DEPT: IT
YEAR: 4TH

PART 1

HELPER FUNCTIONS 1 -

helper_functions > plot_macro_avg_roc_curve

```python
from sklearn.metrics import confusion_matrix
import numpy as np
import matplotlib.pyplot as plt
from sklearn.base import clone
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize

def index_of_max(arr, not_allowed):
    best = float('-inf')
    best_index = -1
    for i in range(len(arr)):
        if arr[i] <= best or i in not_allowed:
            continue
        best = arr[i]
        best_index = i
    return best_index

#returns the mapping of hidden state to true class
def get_mapping(classifier, X_train, y_train):

    y_pred = classifier.predict(X_train)
    matrix = confusion_matrix(y_train, y_pred)
    mapping = dict()
    used = set() #set of all the hidden states that have already been mapped

    for i in range(len(matrix)):
        best_guess = index_of_max(matrix[i], used)
        used.add(best_guess)
        mapping[best_guess] = i
    return mapping

def custom_scoring_func(y_true, y_pred):
    matrix = confusion_matrix(y_true, y_pred)
    mapping = dict()
    used = set() #set of all the hidden states that have already been mapped

    for i in range(len(matrix)):
        best_guess = index_of_max(matrix[i], used)
        used.add(best_guess)
        mapping[best_guess] = i

    y_pred = list(map(lambda hs : mapping[hs], y_pred))

    return np.count_nonzero(y_true==y_pred)/len(y_true)

def custom_learning_curve(base_classifier, X, y, train_ratios):

```

helper_functions > plot_macro_avg_roc_curve

```python
def custom_learning_curve(base_classifier, X, y, train_ratios):

    def helper(X_train, y_train, X_test, y_test):
        classifier = clone(base_classifier)
        classifier.fit(X_train)
        mapper = get_mapping(classifier, X_train, y_train)
        y_pred = list(map(lambda hs : mapper[hs], classifier.predict(X_test)))
        y_train_pred = list(map(lambda hs : mapper[hs],
            classifier.predict(X_train)))
        score_test = np.count_nonzero(y_test==y_pred)/len(y_test)
        score_train = np.count_nonzero(y_train==y_train_pred)/len(y_train)

        return score_train, score_test

    train_sizes = np.int_(train_ratios*len(X))
    train_scores = []
    test_scores = []

    for ratio in train_ratios:
        train = []
        test = []
        for random_state in range(5):
            X_train, X_test, y_train, y_test = train_test_split(X, y,
                train_size=ratio, random_state=random_state)
            score_train, score_test = helper(X_train, y_train, X_test, y_test)
            train.append(score_train)
            test.append(score_test)
        train_scores.append(train)
        test_scores.append(test)

    return train_sizes, train_scores, test_scores

def save_learning_curve(base_classifier, title, X, y, ylim=(0.4, 1.01),
    train_ratios=np.linspace(0.18, 0.8, 4), filepath="lc.png"):
    plt.figure()
    ax = plt.gca()
    ax.set_title(title)
    if ylim is not None:
        ax.set_ylim(*ylim)
    ax.set_xlabel("Training examples")
    ax.set_ylabel("Score")

    train_sizes, train_scores, test_scores = custom_learning_curve(
        base_classifier,
        X,
        y,
        train_ratios=train_ratios
    )
```

HELPER FUNCTIONS 2 -

helper_functions ❯ plot_macro_avg_roc_curve

```python
def save_learning_curve(base_classifier, title, X, y, ylim=(0.4, 1.01),
    train_ratios=np.linspace(0.18, 0.8, 4), filepath="lc.png"):
    plt.figure()
    ax = plt.gca()
    ax.set_title(title)
    if ylim is not None:
        ax.set_ylim(*ylim)
    ax.set_xlabel("Training examples")
    ax.set_ylabel("Score")

    train_sizes, train_scores, test_scores = custom_learning_curve(
        base_classifier,
        X,
        y,
        train_ratios=train_ratios
    )

    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    # Plot learning curve
    ax.grid()
    ax.fill_between(
        train_sizes,
        train_scores_mean - train_scores_std,
        train_scores_mean + train_scores_std,
        alpha=0.1,
        color="r",
    )
    ax.fill_between(
        train_sizes,
        test_scores_mean - test_scores_std,
        test_scores_mean + test_scores_std,
        alpha=0.1,
        color="g",
    )
    ax.plot(
        train_sizes, train_scores_mean, "o-", color="r", label="Training score"
    )
    ax.plot(
        train_sizes, test_scores_mean, "o-", color="g", label="Validation score"
    )
    ax.legend(loc="best")
    plt.savefig(filepath)

```

helper_functions ❯ plot_macro_avg_roc_curve

```python
def plot_macro_avg_roc_curve(classifier, n_classes, name, X_train, X_test, y_train, y_test):
    classifier.fit(X_train)
    y_pred = classifier.predict(X_test)
    mapping = get_mapping(classifier, X_train, y_train)
    y_pred_changed = list(map(lambda hs : mapping[hs], y_pred))
    y_test_binarized = label_binarize(y_test, classes=[0, 1, 2])
    y_score = classifier.predict_proba(X_test)

    #Rearranging the columns of y_score matrix
    perm_list = [0 for _ in range(n_classes)]

    for key in mapping:
        perm_list[mapping[key]] = key

    y_score= y_score[:, perm_list]

    # Compute ROC curve and ROC area for each class
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], y_score[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # First aggregate all false positive rates
    all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))

    # Then interpolate all ROC curves at this points
    mean_tpr = np.zeros_like(all_fpr)
    for i in range(n_classes):
        mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])

    # Finally average it and compute AUC
    mean_tpr /= n_classes

    fpr["macro"] = all_fpr
    tpr["macro"] = mean_tpr
    roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

    plt.plot(
        fpr["macro"],
        tpr["macro"],
        label="{0} (AUC = {1:0.2f})".format(name, roc_auc["macro"]),
        alpha=0.7,
        linewidth=1,
    )

    return plt
```

WINE - CLASSIFICATION

GAUSSIAN HMM

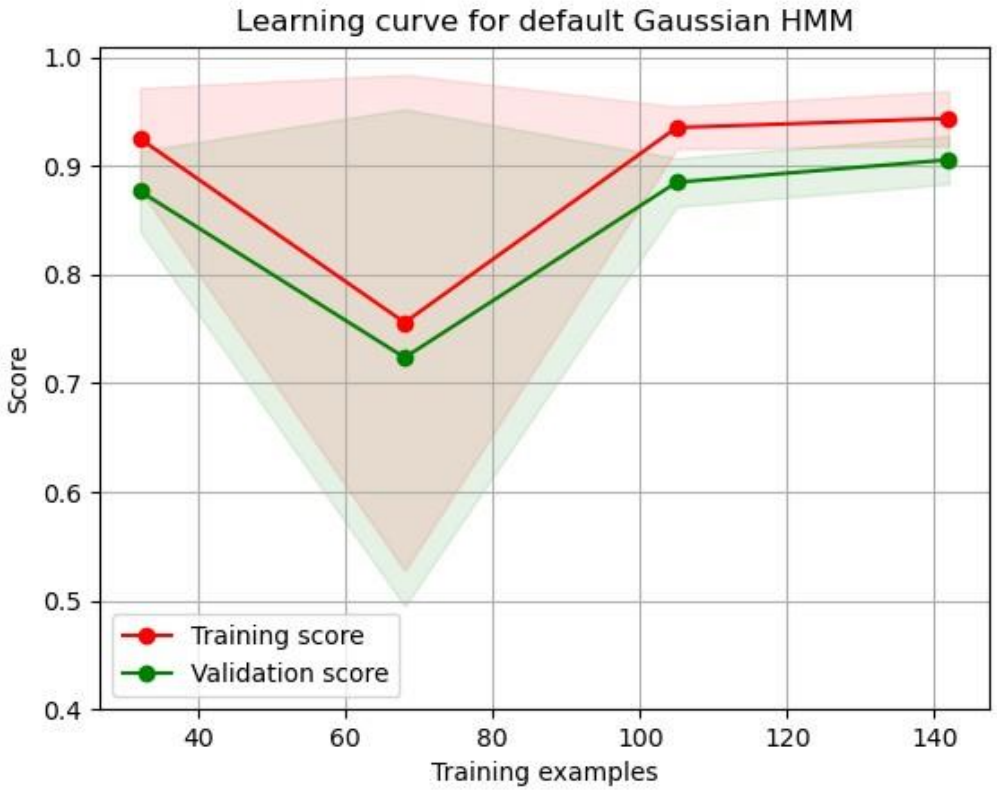
GAUSSIAN HMM (NO PARAMETER TUNING)- CODE & EVALUATION

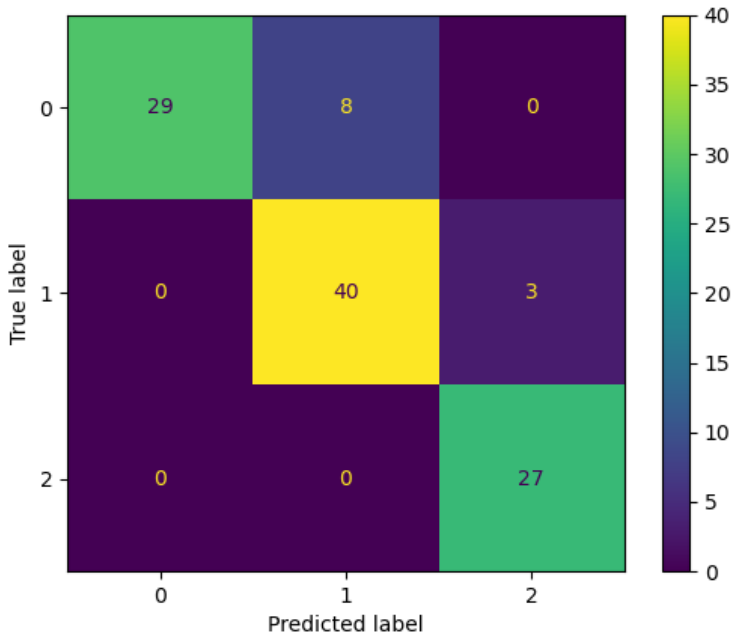
gaussian_no_param 〉No Selection

```python
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from hmmlearn import hmm
from sklearn.metrics import classification_report, confusion_matrix,
    accuracy_score, ConfusionMatrixDisplay
from helper_functions import *

NUM_CLASSES = 3

df = pd.read_csv('./wine.data', header=None)
df = df.sample(frac = 1, random_state=0)
X = df.drop([0], axis=1)
y = df[0]
y = y-1 #so that classes are 0,1,2 instead of 1,2,3

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.6,
    random_state=0)
classifier = hmm.GaussianHMM(n_components=NUM_CLASSES)
classifier.fit(X_train)

mapping = get_mapping(classifier, X_train, y_train)
y_pred = list(map(lambda hs : mapping[hs], classifier.predict(X_test)))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("-------------------------------------------------------")
print("-------------------------------------------------------")
print("Performance Evaluation:")
print(classification_report(y_test, y_pred))
print("-------------------------------------------------------")
print("-------------------------------------------------------")
print("Accuracy Score:")
print(accuracy_score(y_test, y_pred))

plt.figure()
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.savefig('./no_param.png')

plt.figure()
save_learning_curve(filepath='./no_param_lc.png',
    base_classifier=classifier, title="Learning curve for default
    Gaussian HMM", X=X, y=y)
```

```
onfusion Matrix:
[29  8  0]
[ 0 40  3]
[ 0  0 27]]
-------------------------------------------------------
-------------------------------------------------------
erformance Evaluation:
              precision    recall  f1-score   support

           0       1.00      0.78      0.88        37
           1       0.83      0.93      0.88        43
           2       0.90      1.00      0.95        27

    accuracy                           0.90       107
   macro avg       0.91      0.90      0.90       107
eighted avg       0.91      0.90      0.90       107

-------------------------------------------------------
-------------------------------------------------------
ccuracy Score:
.897196261682243
```

GAUSSIAN HMM (NO PARAMETER TUNING)- CONFUSION MATRIX AND LEARNING CURVE





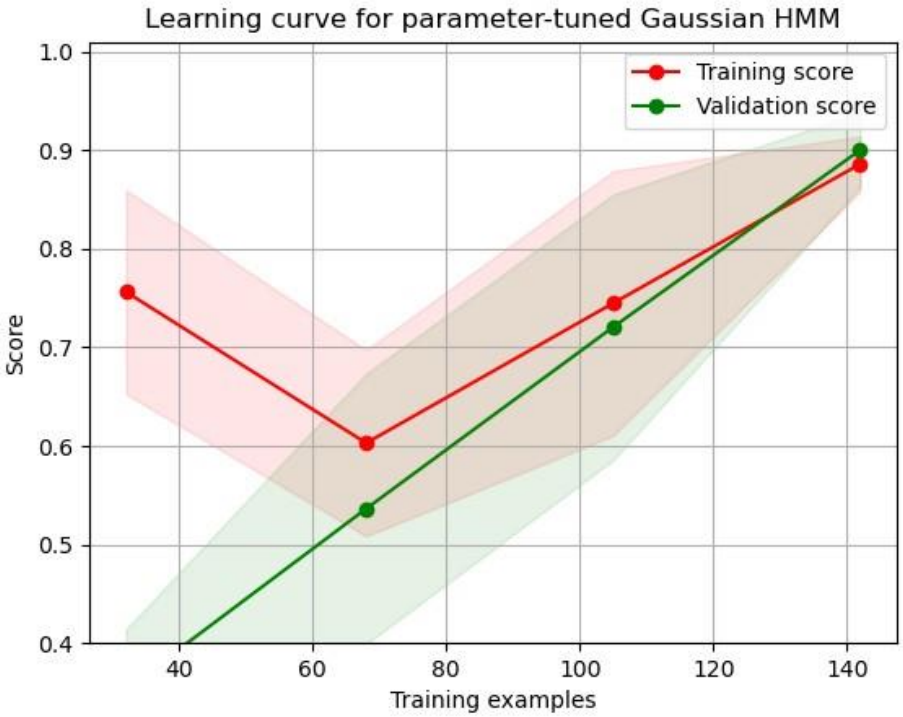
GAUSSIAN HMM (WITH PARAMETER TUNING)- CODE & EVALUATION

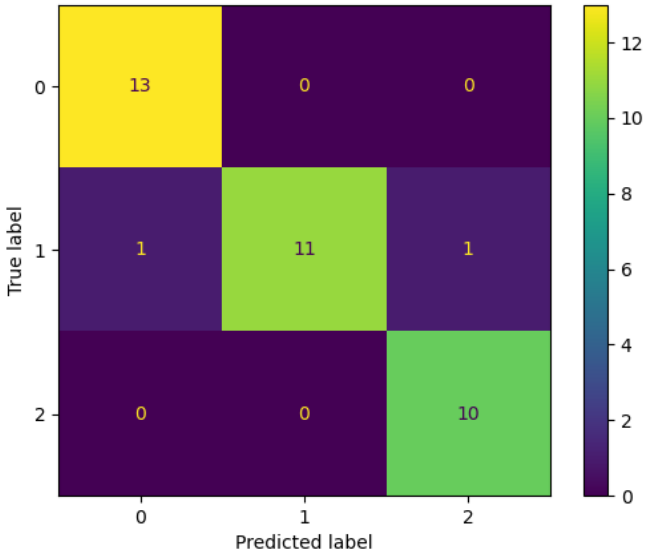
gaussian_param > No Selection

```python
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from hmmlearn import hmm
from sklearn.metrics import classification_report, confusion_matrix,
    accuracy_score, ConfusionMatrixDisplay
from helper_functions import *

NUM_CLASSES = 3

df = pd.read_csv('./wine.data', header=None)
df = df.sample(frac = 1, random_state=0)
X = df.drop([0], axis=1)
y = df[0]
y = y-1 #so that classes are 0,1,2 instead of 1,2,3

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=0)
classifier = hmm.GaussianHMM(n_components=NUM_CLASSES,
    covariance_type="full", min_covar=0.001, tol=0.01,
    implementation="log", random_state=0)
classifier.fit(X_train)

mapping = get_mapping(classifier, X_train, y_train)
y_pred = list(map(lambda hs : mapping[hs], classifier.predict(X_test)))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("--------------------------------------------------------")
print("--------------------------------------------------------")
print("Performance Evaluation:")
print(classification_report(y_test, y_pred))
print("--------------------------------------------------------")
print("--------------------------------------------------------")
print("Accuracy Score:")
print(accuracy_score(y_test, y_pred))

plt.figure()
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.savefig('./param.png')

plt.figure()
save_learning_curve(filepath='./param_lc.png',
    base_classifier=classifier, title="Learning curve for parameter-tuned
    Gaussian HMM", X=X, y=y)
```

```
onfusion Matrix:
[13  0  0]
[ 1 11  1]
[ 0  0 10]]
------------------------------------------------------------
------------------------------------------------------------
erformance Evaluation:
              precision    recall  f1-score   support

           0       0.93      1.00      0.96        13
           1       1.00      0.85      0.92        13
           2       0.91      1.00      0.95        10

    accuracy                           0.94        36
   macro avg       0.95      0.95      0.94        36
eighted avg       0.95      0.94      0.94        36

------------------------------------------------------------
------------------------------------------------------------
ccuracy Score:
.9444444444444444
```

GAUSSIAN HMM (WITH PARAMETER TUNING)- CONFUSION MATRIX AND LEARNING CURVE





HMM WITH GAUSSIAN MIXTURE MODEL EMISSIONS

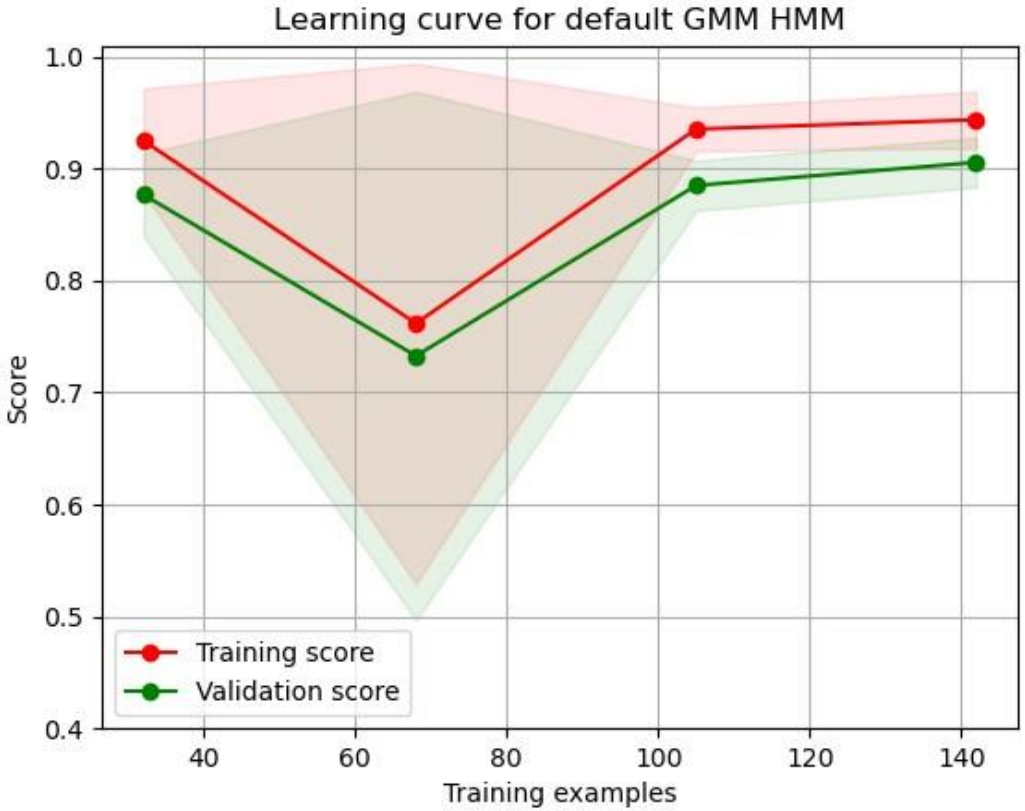
GMM HMM (NO PARAMETER TUNING)- CODE & EVALUATION

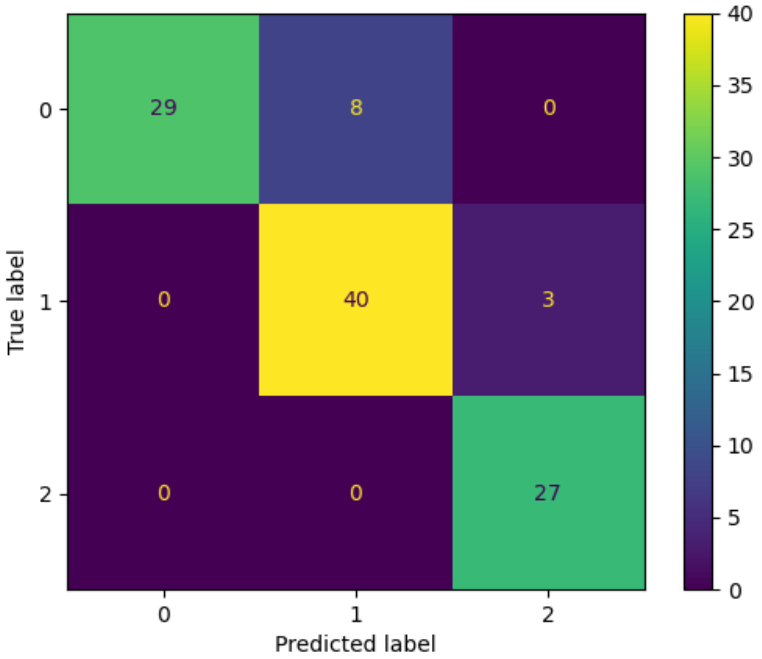
gmm_hmm_no_param ❯ No Selection

```python
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from hmmlearn import hmm
from sklearn.metrics import classification_report, confusion_matrix,
    accuracy_score, ConfusionMatrixDisplay
from helper_functions import *

NUM_CLASSES = 3

df = pd.read_csv('./wine.data', header=None)
df = df.sample(frac = 1, random_state=0)
X = df.drop([0], axis=1)
y = df[0]
y = y-1 #so that classes are 0,1,2 instead of 1,2,3

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.6,
    random_state=0)
classifier = hmm.GMMHMM(n_components=NUM_CLASSES)
classifier.fit(X_train)

mapping = get_mapping(classifier, X_train, y_train)
y_pred = list(map(lambda hs : mapping[hs], classifier.predict(X_test)))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("-------------------------------------------------------")
print("-------------------------------------------------------")
print("Performance Evaluation:")
print(classification_report(y_test, y_pred))
print("-------------------------------------------------------")
print("-------------------------------------------------------")
print("Accuracy Score:")
print(accuracy_score(y_test, y_pred))

plt.figure()
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.savefig('./no_param.png')

plt.figure()
save_learning_curve(filepath='./no_param_lc.png',
    base_classifier=classifier, title="Learning curve for default GMM
    HMM", X=X, y=y)
```

```
Confusion Matrix:
[[29  8  0]
 [ 0 40  3]
 [ 0  0 27]]
-------------------------------------------------------
-------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           0       1.00      0.78      0.88        37
           1       0.83      0.93      0.88        43
           2       0.90      1.00      0.95        27

    accuracy                           0.90       107
   macro avg       0.91      0.90      0.90       107
weighted avg       0.91      0.90      0.90       107

-------------------------------------------------------
-------------------------------------------------------
Accuracy Score:
0.897196261682243
```

GMM HMM (NO PARAMETER TUNING)- CONFUSION MATRIX AND LEARNING CURVE





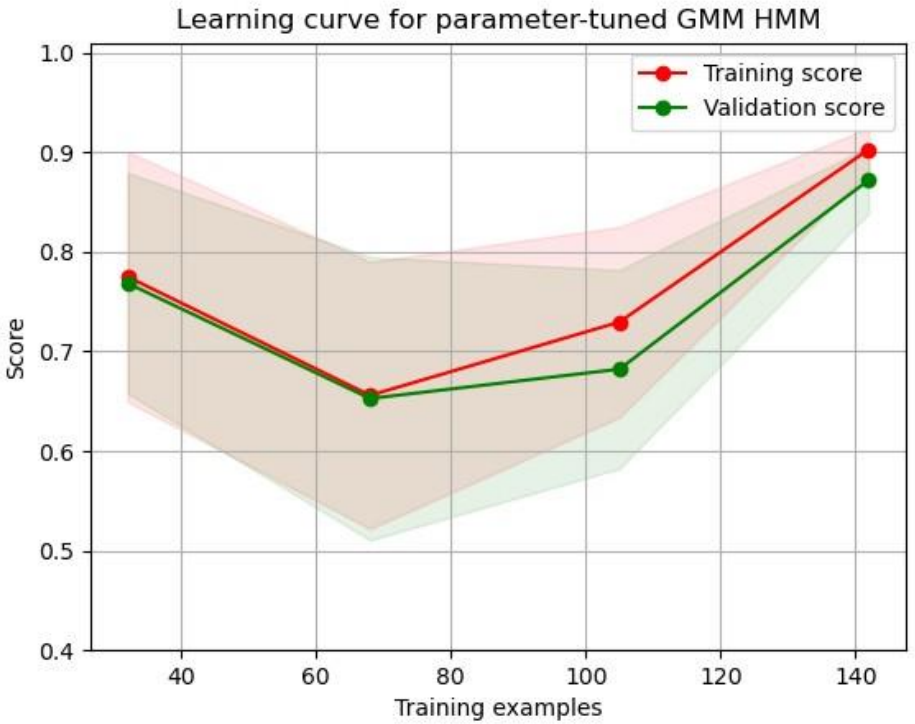
GMM HMM (WITH PARAMETER TUNING)- CODE & EVALUATION

gmm_hmm_param / No Selection

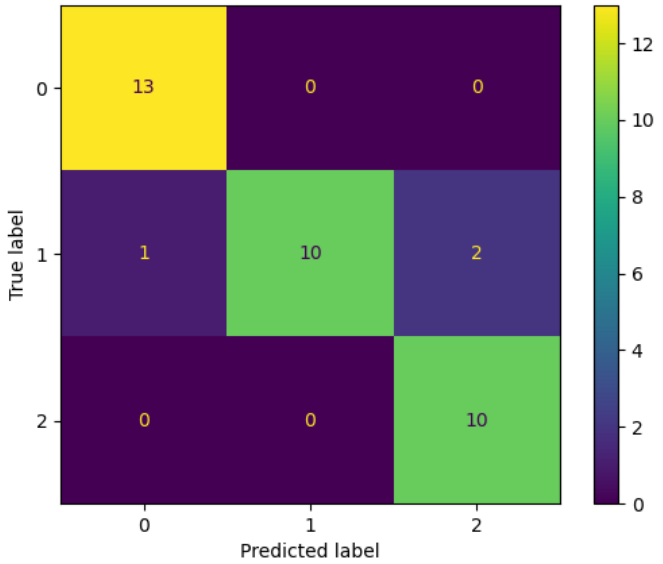
```python
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from hmmlearn import hmm
from sklearn.metrics import classification_report, confusion_matrix,
    accuracy_score, ConfusionMatrixDisplay
from helper_functions import *

NUM_CLASSES = 3

df = pd.read_csv('./wine.data', header=None)
df = df.sample(frac = 1, random_state=0)
X = df.drop([0], axis=1)
y = df[0]
y = y-1 #so that classes are 0,1,2 instead of 1,2,3

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=0)
classifier = hmm.GMMHMM(n_components=3, covariance_type="diag", n_mix=2,
    min_covar=0.01, tol=0.01, implementation="log", random_state=1)
classifier.fit(X_train)

mapping = get_mapping(classifier, X_train, y_train)
y_pred = list(map(lambda hs : mapping[hs], classifier.predict(X_test)))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("-------------------------------------------------------")
print("-------------------------------------------------------")
print("Performance Evaluation:")
print(classification_report(y_test, y_pred))
print("-------------------------------------------------------")
print("-------------------------------------------------------")
print("Accuracy Score:")
print(accuracy_score(y_test, y_pred))

plt.figure()
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.savefig('./param.png')

plt.figure()
save_learning_curve(filepath='./param_lc.png',
    base_classifier=classifier, title="Learning curve for parameter-tuned
    GMM HMM", X=X, y=y)
```

```
onfusion Matrix:
[13  0  0]
[ 1 10  2]
[ 0  0 10]]
-------------------------------------------------------
-------------------------------------------------------
erformance Evaluation:
              precision    recall  f1-score   support

           0       0.93      1.00      0.96        13
           1       1.00      0.77      0.87        13
           2       0.83      1.00      0.91        10

    accuracy                           0.92        36
   macro avg       0.92      0.92      0.91        36
eighted avg       0.93      0.92      0.91        36

-------------------------------------------------------
-------------------------------------------------------
ccuracy Score:
.9166666666666666
```

GMM HMM (WITH PARAMETER TUNING)- CONFUSION MATRIX AND LEARNING CURVE

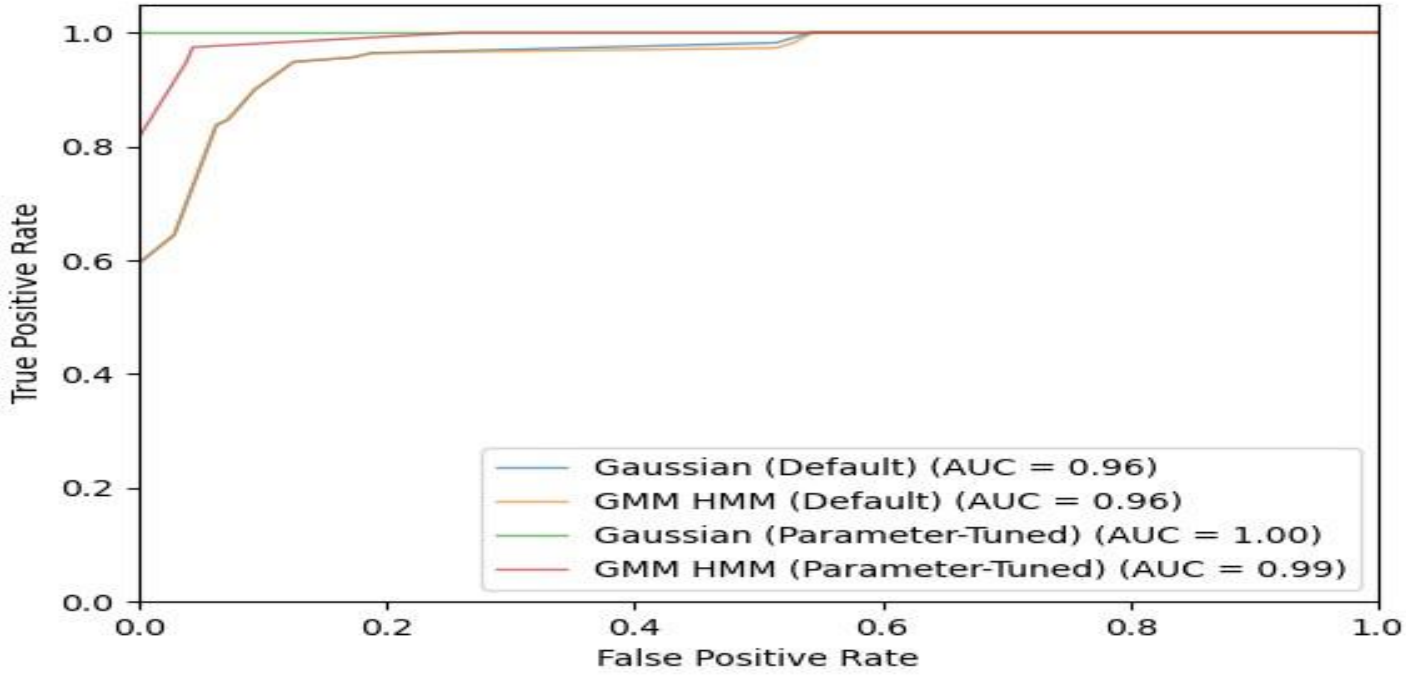


CONFUSION MATRIX
HeatMap



COMPARISON BETWEEN GAUSSIAN HMM AND GMM HMM.

ROC CURVES AND AUC COMPARISON



ROC CURVES AND AUC COMPARISON - CODE

make_roc_auc_curves

make_roc_auc_curves 〉 No Selection

```
1  import matplotlib.pyplot as plt
2  import pandas as pd
3  from sklearn.model_selection import train_test_split
4  from hmmlearn import hmm
5  from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, ConfusionMatrixDisplay
6  from helper_functions import *
7
8  NUM_CLASSES = 3
9
10 df = pd.read_csv('./wine.data', header=None)
11 df = df.sample(frac = 1, random_state=0)
12 X = df.drop([0], axis=1)
13 y = df[0]
14 y = y-1 #so that classes are 0,1,2 instead of 1,2,3
15
16
17 classifier = dict()
18 classifier['gaussian_no_param'] = hmm.GaussianHMM(n_components=NUM_CLASSES)
19 classifier['gaussian_param'] = hmm.GaussianHMM(n_components=NUM_CLASSES, covariance_type="full",
       min_covar=0.001, tol=0.01, implementation="log", random_state=0)
20
21 classifier['gmm_hmm_no_param'] = hmm.GMMHMM(n_components=NUM_CLASSES)
22 classifier['gmm_hmm_param'] = hmm.GMMHMM(n_components=3, covariance_type="diag", n_mix=2, min_covar=0.01,
       tol=0.01, implementation="log", random_state=1)
23
24 #division for default Gaussian and GMM HMM models, since they perform best in this partition
25 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.6, random_state=0)
26
27
28 plot_macro_avg_roc_curve(classifier['gaussian_no_param'], n_classes=NUM_CLASSES, name="Gaussian
       (Default)", X_train=X_train, X_test=X_test, y_train=y_train, y_test=y_test)
29 plot_macro_avg_roc_curve(classifier['gmm_hmm_no_param'], n_classes=NUM_CLASSES, name="GMM HMM (Default)",
       X_train=X_train, X_test=X_test, y_train=y_train, y_test=y_test)
30
31 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
32
33 plot_macro_avg_roc_curve(classifier['gaussian_param'], n_classes=NUM_CLASSES, name="Gaussian
       (Parameter-Tuned)", X_train=X_train, X_test=X_test, y_train=y_train, y_test=y_test)
34 plot_macro_avg_roc_curve(classifier['gmm_hmm_param'], n_classes=NUM_CLASSES, name="GMM HMM
       (Parameter-Tuned)", X_train=X_train, X_test=X_test, y_train=y_train, y_test=y_test)
35
36 plt.xlim([0.0, 1.0])
37 plt.ylim([0.0, 1.05])
38 plt.xlabel("False Positive Rate")
39 plt.ylabel("True Positive Rate")
40 plt.legend(loc="lower right")
41
42 plt.savefig('roc_auc.png')
```

Line: 1 Col: 1

COMPARISON OF PERFORMANCE

CLASSIFIER		Accuracy	P ec	Recall	F -Sco	AUC
GAUSSIAN HMM	DEFAULT	0.93	0.93	0.93	0.93	0.96
	TUNED	0.94	0.95	0.94	0.94	.00
GMM HMM	DEFAULT	0.90	0.9	0.90	0.90	0.96
	TUNED	0.92	0.93	0.92	0.9	0.99

IONOSPHERE - CLASSIFICATION

GAUSSIAN HMM

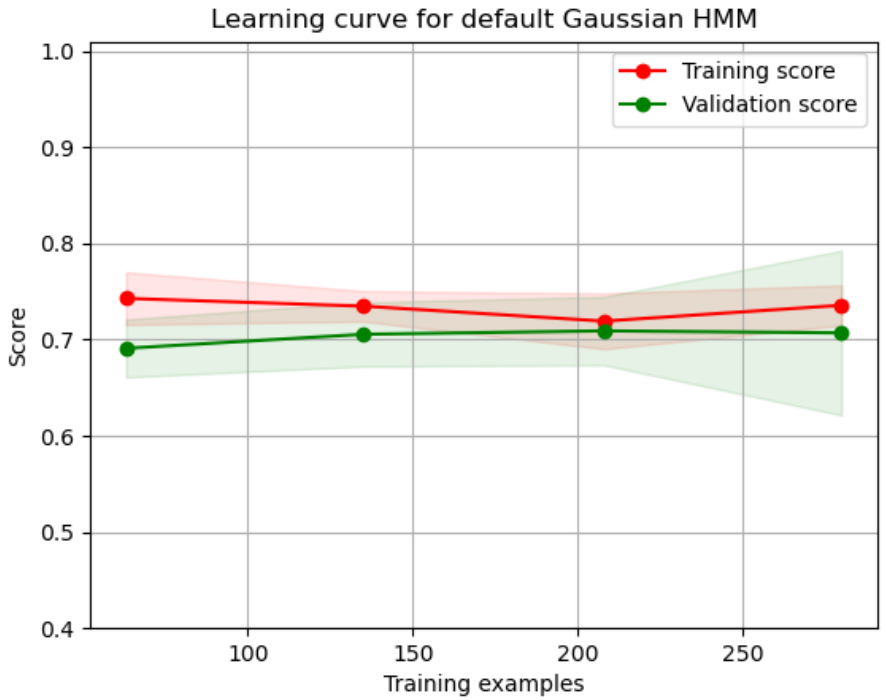
GAUSSIAN HMM (NO PARAMETER TUNING)- CODE & EVALUATION

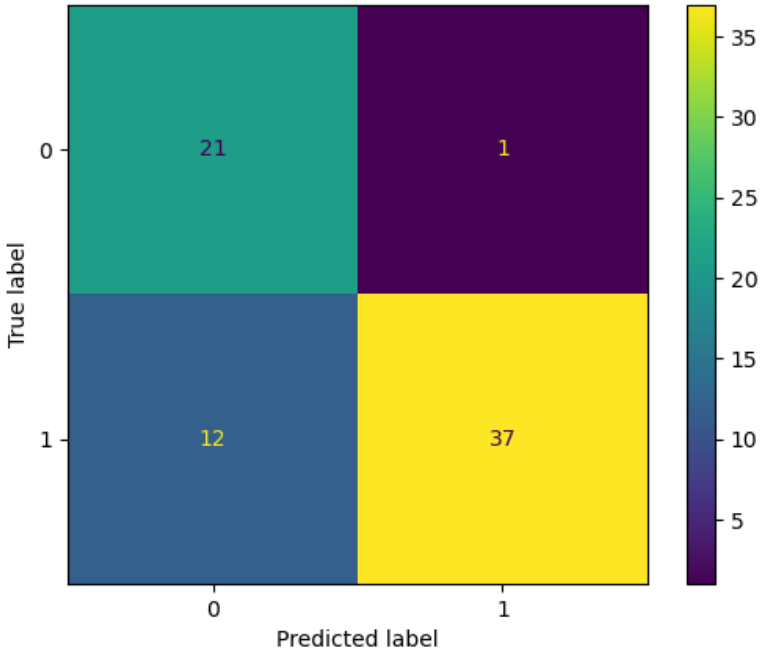
gaussian_no_param ❯ No Selection

```python
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from hmmlearn import hmm
from sklearn.metrics import classification_report, confusion_matrix,
    accuracy_score, ConfusionMatrixDisplay
from helper_functions import *

NUM_CLASSES = 2

df = pd.read_csv('./ionosphere.data', header=None)
df = df.sample(frac = 1, random_state=0)
X = df.drop([34], axis=1)
y = df[34]
y = y.map({'g':1, 'b':0})

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=3)
classifier = hmm.GaussianHMM(n_components=NUM_CLASSES)
classifier.fit(X_train)

mapping = get_mapping(classifier, X_train, y_train)
y_pred = list(map(lambda hs : mapping[hs], classifier.predict(X_test)))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("--------------------------------------------------------")
print("--------------------------------------------------------")
print("Performance Evaluation:")
print(classification_report(y_test, y_pred))
print("--------------------------------------------------------")
print("--------------------------------------------------------")
print("Accuracy Score:")
print(accuracy_score(y_test, y_pred))

plt.figure()
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.savefig('./no_param.png')

plt.figure()
save_learning_curve(filepath='./no_param_lc.png',
    base_classifier=classifier, title="Learning curve for default
    Gaussian HMM", X=X, y=y)
```

```
Confusion Matrix:
[[21  1]
 [12 37]]
--------------------------------------------------------
--------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           0       0.64      0.95      0.76        22
           1       0.97      0.76      0.85        49

    accuracy                           0.82        71
   macro avg       0.81      0.85      0.81        71
weighted avg       0.87      0.82      0.82        71

--------------------------------------------------------
--------------------------------------------------------
Accuracy Score:
0.8169014084507042
```

GAUSSIAN HMM (NO PARAMETER TUNING)- CONFUSION MATRIX AND



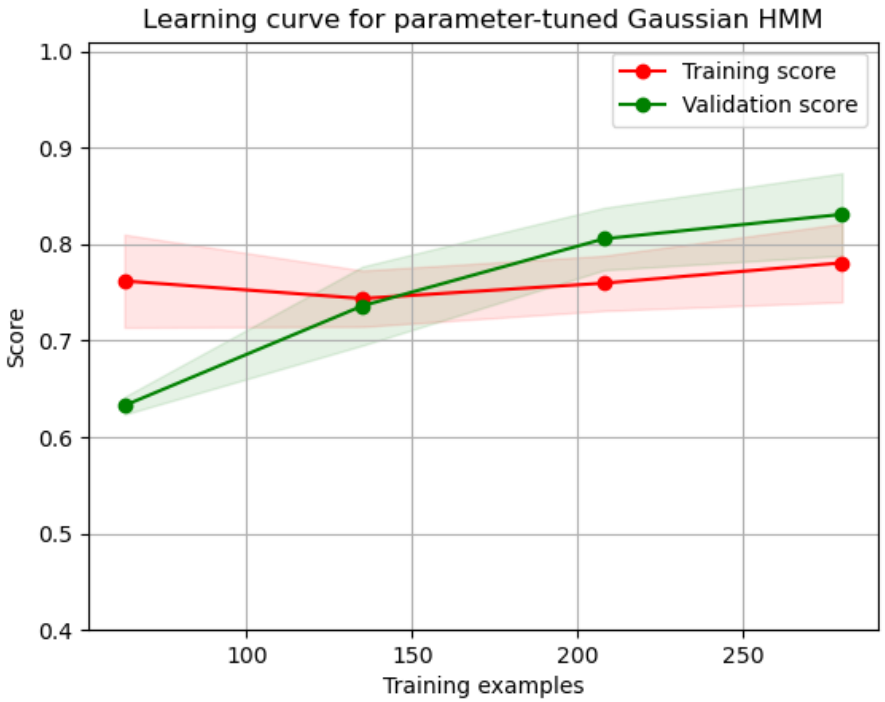


GAUSSIAN HMM (WITH PARAMETER TUNING)- CODE & EVALUATION

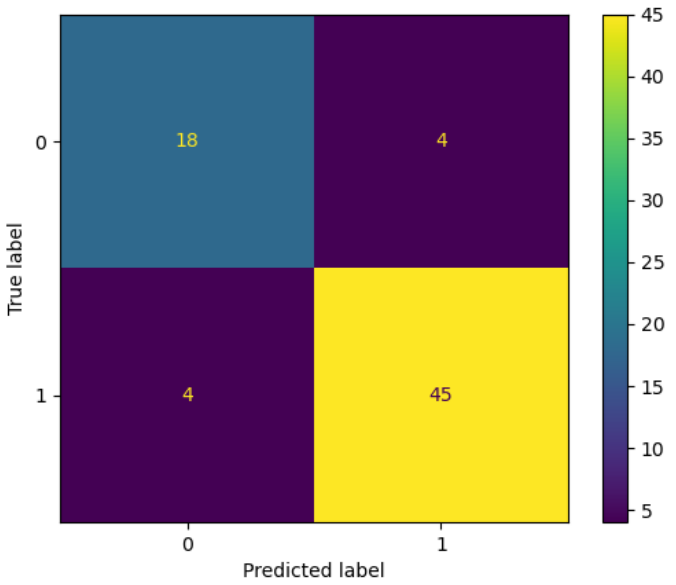
```python
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from hmmlearn import hmm
from sklearn.metrics import classification_report, confusion_matrix,
    accuracy_score, ConfusionMatrixDisplay
from helper_functions import *

NUM_CLASSES = 2

df = pd.read_csv('./ionosphere.data', header=None)
df = df.sample(frac = 1, random_state=0)
X = df.drop([34], axis=1)
y = df[34]
y = y.map({'g':1, 'b':0})

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=3)
classifier = hmm.GaussianHMM(n_components=NUM_CLASSES,
    covariance_type="full", min_covar=0.01, tol=0.01,
    implementation="log", random_state=0)
classifier.fit(X_train)

mapping = get_mapping(classifier, X_train, y_train)
y_pred = list(map(lambda hs : mapping[hs], classifier.predict(X_test)))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("-------------------------------------------------------")
print("-------------------------------------------------------")
print("Performance Evaluation:")
print(classification_report(y_test, y_pred))
print("-------------------------------------------------------")
print("-------------------------------------------------------")
print("Accuracy Score:")
print(accuracy_score(y_test, y_pred))

plt.figure()
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.savefig('./param.png')

plt.figure()
save_learning_curve(filepath='./param_lc.png',
    base_classifier=classifier, title="Learning curve for parameter-tuned
    Gaussian HMM", X=X, y=y)
```

```
Confusion Matrix:
[[18  4]
 [ 4 46]]
-------------------------------------------------------
-------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           0       0.82      0.82      0.82        22
           1       0.92      0.92      0.92        49

    accuracy                           0.89        71
   macro avg       0.87      0.87      0.87        71
weighted avg       0.89      0.89      0.89        71

-------------------------------------------------------
-------------------------------------------------------
Accuracy Score:
0.8873239436619719
```

GAUSSIAN HMM (WITH PARAMETER TUNING)- CONFUSION MATRIX AND LEARNING CURVE



CONFUSION MATRIX
HeatMap



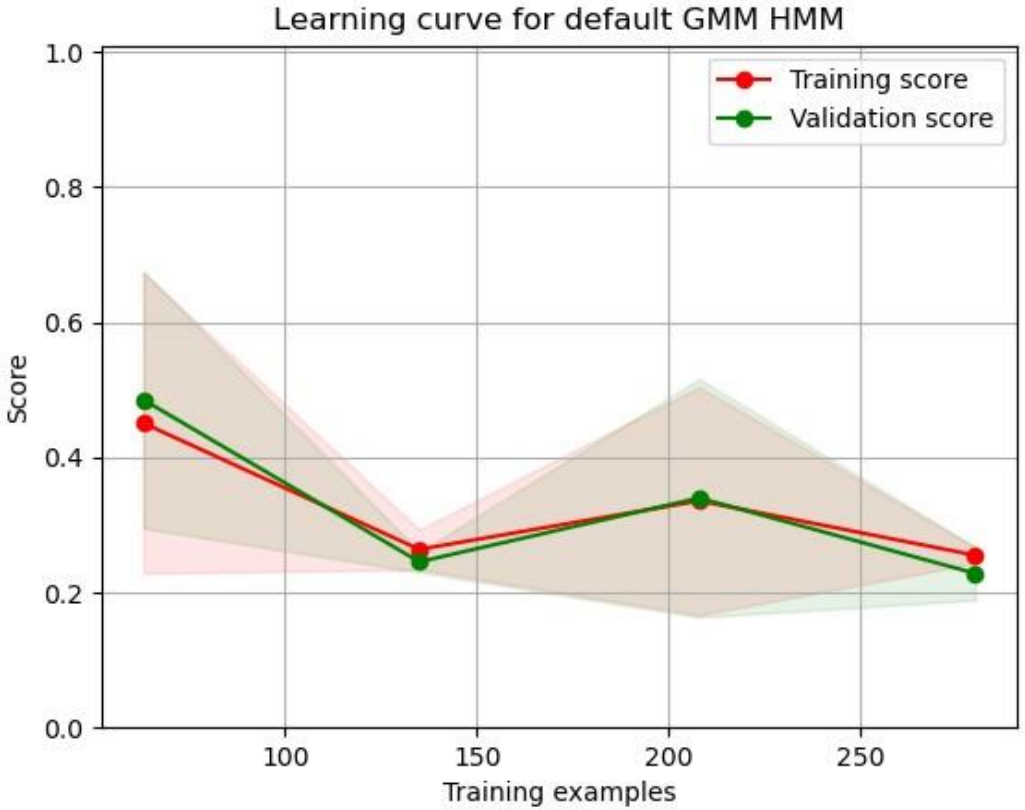
HMM WITH GAUSSIAN MIXTURE MODEL EMISSIONS

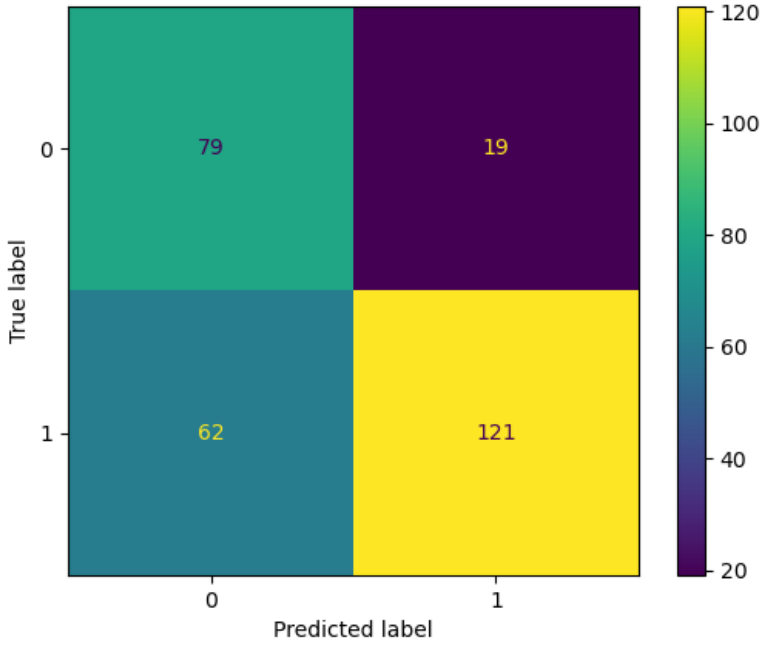
GMM HMM (NO PARAMETER TUNING)- CODE & EVALUATION

```python
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from hmmlearn import hmm
from sklearn.metrics import classification_report, confusion_matrix,
    accuracy_score, ConfusionMatrixDisplay
from helper_functions import *

NUM_CLASSES = 2

df = pd.read_csv('./ionosphere.data', header=None)
df = df.sample(frac = 1, random_state=0)
X = df.drop([34], axis=1)
y = df[34]
y = y.map({'g':1, 'b':0})

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.8,
    random_state=3)
classifier = hmm.GMMHMM(n_components=NUM_CLASSES)
classifier.fit(X_train)

mapping = get_mapping(classifier, X_train, y_train)
y_pred = list(map(lambda hs : mapping[hs], classifier.predict(X_test)))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("--------------------------------------------------------")
print("--------------------------------------------------------")
print("Performance Evaluation:")
print(classification_report(y_test, y_pred))
print("--------------------------------------------------------")
print("--------------------------------------------------------")
print("Accuracy Score:")
print(accuracy_score(y_test, y_pred))

plt.figure()
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.savefig('./no_param.png')

plt.figure()
save_learning_curve(filepath='./no_param_lc.png',
    base_classifier=classifier, title="Learning curve for default GMM
    HMM", X=X, y=y)
```

```
Confusion Matrix:
[[ 79  19]
 [ 62 121]]
--------------------------------------------------------
--------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           0       0.56      0.81      0.66        98
           1       0.86      0.66      0.75       183

    accuracy                           0.71       281
   macro avg       0.71      0.73      0.71       281
weighted avg       0.76      0.71      0.72       281

--------------------------------------------------------
--------------------------------------------------------
Accuracy Score:
0.7117437722419929
```

GMM HMM (NO PARAMETER TUNING)- CONFUSION MATRIX AND LEARNING CURVE



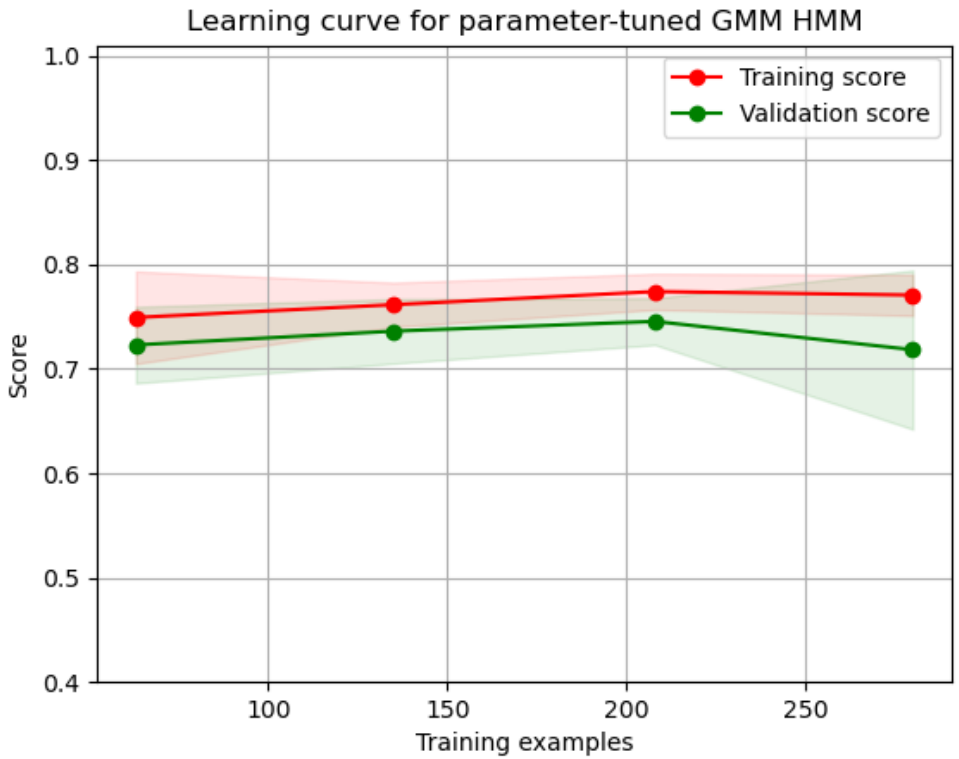


GMM HMM (WITH PARAMETER TUNING)- CODE & EVALUATION

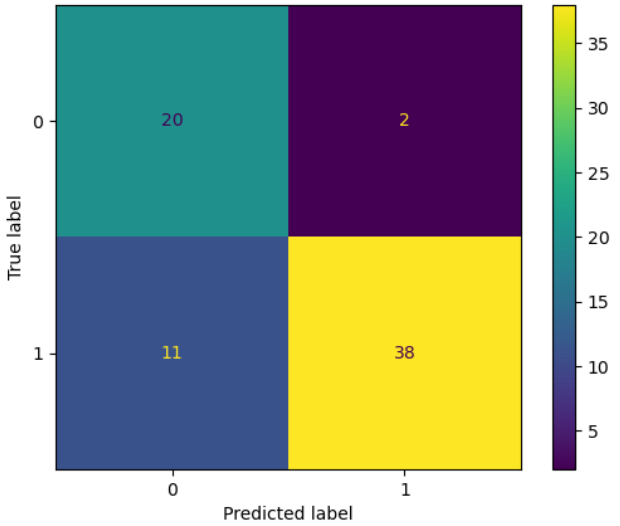
```python
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from hmmlearn import hmm
from sklearn.metrics import classification_report, confusion_matrix,
    accuracy_score, ConfusionMatrixDisplay
from helper_functions import *

NUM_CLASSES = 2

df = pd.read_csv('./ionosphere.data', header=None)
df = df.sample(frac = 1, random_state=0)
X = df.drop([34], axis=1)
y = df[34]
y = y.map({'g':1, 'b':0})

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=3)
classifier = hmm.GMMHMM(n_components=NUM_CLASSES,
    covariance_type="spherical", n_mix=3, min_covar=0.001, tol=0.01,
    implementation="log", random_state=1)
classifier.fit(X_train)

mapping = get_mapping(classifier, X_train, y_train)
y_pred = list(map(lambda hs : mapping[hs], classifier.predict(X_test)))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("-------------------------------------------------------")
print("-------------------------------------------------------")
print("Performance Evaluation:")
print(classification_report(y_test, y_pred))
print("-------------------------------------------------------")
print("-------------------------------------------------------")
print("Accuracy Score:")
print(accuracy_score(y_test, y_pred))

plt.figure()
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.savefig('./param.png')

plt.figure()
save_learning_curve(filepath='./param_lc.png',
    base_classifier=classifier, title="Learning curve for parameter-tuned
    GMM HMM", X=X, y=y)
```

```
onfusion Matrix:
[20  2]
[11 38]]
-------------------------------------------------------
-------------------------------------------------------
erformance Evaluation:
              precision    recall  f1-score   support

           0       0.65      0.91      0.75        22
           1       0.95      0.78      0.85        49

    accuracy                           0.82        71
   macro avg       0.80      0.84      0.80        71
eighted avg       0.86      0.82      0.82        71

-------------------------------------------------------
-------------------------------------------------------
ccuracy Score:
.8169014084507042
```

GMM HMM (WITH PARAMETER TUNING)- CONFUSION MATRIX AND LEARNING CURVE

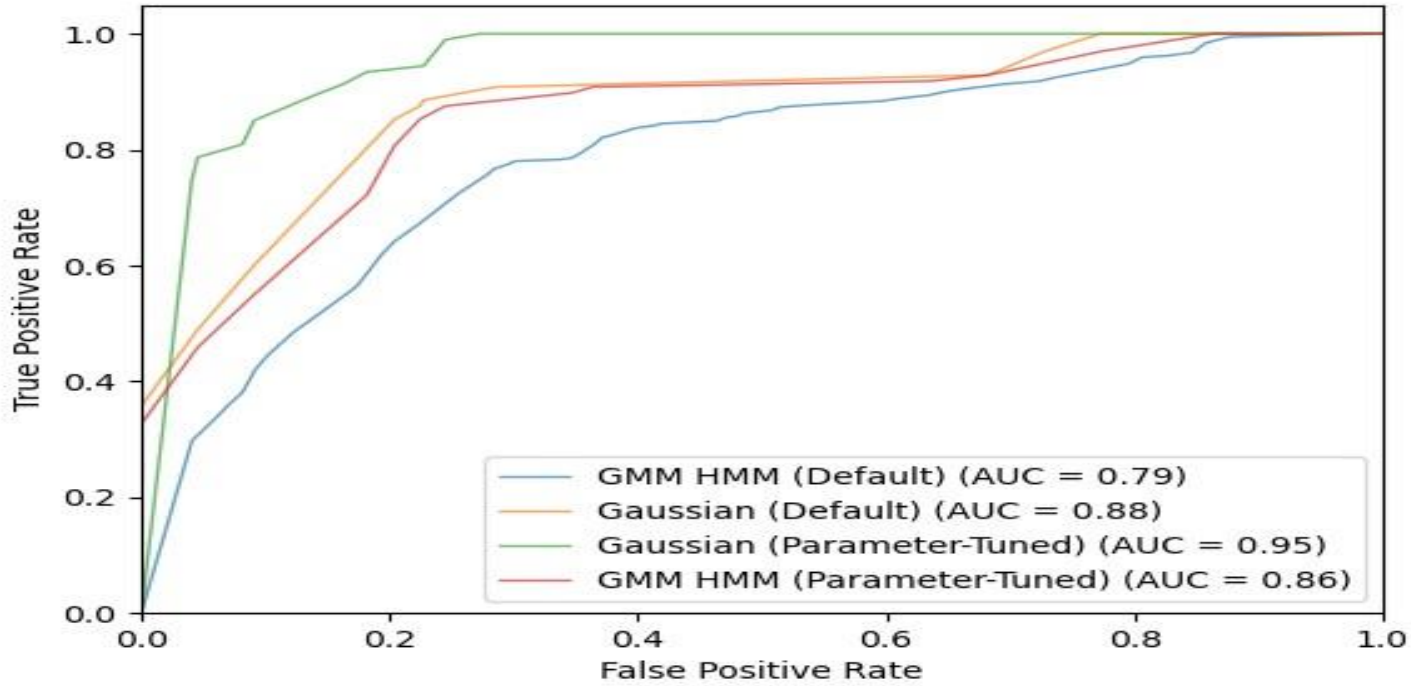


CONFUSION MATRIX
HeatMap



COMPARISON BETWEEN GAUSSIAN HMM AND GMM HMM.

ROC CURVES AND AUC COMPARISON



ROC CURVES AND AUC COMPARISON - CODE

make_roc_auc_curves

make_roc_auc_curves 〉 No Selection

```
1  import matplotlib.pyplot as plt
2  import pandas as pd
3  from sklearn.model_selection import train_test_split
4  from hmmlearn import hmm
5  from helper_functions import *
6
7  NUM_CLASSES = 2
8
9  df = pd.read_csv('./ionosphere.data', header=None)
10 df = df.sample(frac = 1, random_state=0)
11 X = df.drop([34], axis=1)
12 y = df[34]
13 y = y.map({'g':1, 'b':0})
14
15
16 classifier = dict()
17 classifier['gaussian_no_param'] = hmm.GaussianHMM(n_components=NUM_CLASSES)
18 classifier['gaussian_param'] =  hmm.GaussianHMM(n_components=NUM_CLASSES,
       covariance_type="full", min_covar=0.01, tol=0.01, implementation="log", random_state=0)
19 classifier['gmm_hmm_no_param'] = hmm.GMMHMM(n_components=NUM_CLASSES)
20 classifier['gmm_hmm_param'] = hmm.GMMHMM(n_components=NUM_CLASSES,
       covariance_type="spherical", n_mix=3, min_covar=0.001, tol=0.01, implementation="log",
       random_state=1)
21
22
23 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.8, random_state=3)
24 plot_macro_avg_roc_curve(classifier['gmm_hmm_no_param'], n_classes=NUM_CLASSES, name="GMM
       HMM (Default)", X_train=X_train, X_test=X_test, y_train=y_train, y_test=y_test)
25
26 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=3)
27
28 plot_macro_avg_roc_curve(classifier['gaussian_no_param'], n_classes=NUM_CLASSES,
       name="Gaussian (Default)", X_train=X_train, X_test=X_test, y_train=y_train,
       y_test=y_test)
29 plot_macro_avg_roc_curve(classifier['gaussian_param'], n_classes=NUM_CLASSES,
       name="Gaussian (Parameter-Tuned)", X_train=X_train, X_test=X_test, y_train=y_train,
       y_test=y_test)
30 plot_macro_avg_roc_curve(classifier['gmm_hmm_param'], n_classes=NUM_CLASSES, name="GMM HMM
       (Parameter-Tuned)", X_train=X_train, X_test=X_test, y_train=y_train, y_test=y_test)
31
32 plt.xlim([0.0, 1.0])
33 plt.ylim([0.0, 1.05])
34 plt.xlabel("False Positive Rate")
35 plt.ylabel("True Positive Rate")
36 plt.legend(loc="lower right")
37
38 plt.savefig('roc_auc.png')
39
```

COMPARISON OF PERFORMANCE

CLASSIFIER		Accuracy	P ec	Recall	F -Sco	AUC
GAUSSIAN HMM	DEFAULT	0.82	0.87	0.82	0.82	0.88
	TUNED	0.89	0.89	0.89	0.89	0.95
GMM HMM	DEFAULT	0.7	0.76	0.7	0.72	0.79
	TUNED	0.82	0.86	0.82	0.82	0.86

WISCONSIN BREAST CANCER - CLASSIFICATION

GAUSSIAN HMM

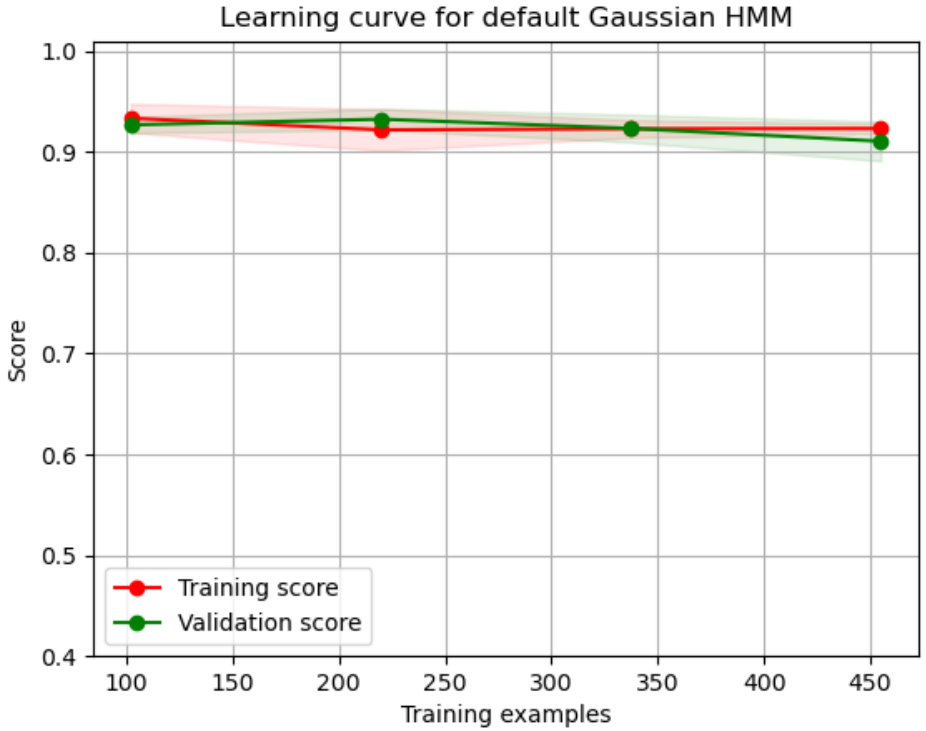
GAUSSIAN HMM (NO PARAMETER TUNING)- CODE & EVALUATION

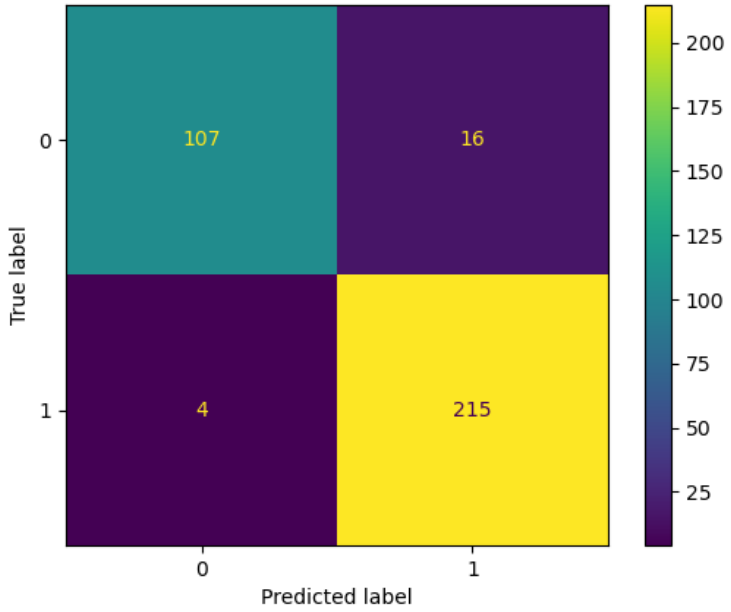
gaussian_no_param / No Selection

```python
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from hmmlearn import hmm
from sklearn.metrics import classification_report, confusion_matrix,
    accuracy_score, ConfusionMatrixDisplay
from helper_functions import *

NUM_CLASSES = 2

X, y = load_breast_cancer(return_X_y=True)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.6,
    random_state=0)
classifier = hmm.GaussianHMM(n_components=NUM_CLASSES)
classifier.fit(X_train)

mapping = get_mapping(classifier, X_train, y_train)
y_pred = list(map(lambda hs : mapping[hs], classifier.predict(X_test)))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("-------------------------------------------------------")
print("-------------------------------------------------------")
print("Performance Evaluation:")
print(classification_report(y_test, y_pred))
print("-------------------------------------------------------")
print("-------------------------------------------------------")
print("Accuracy Score:")
print(accuracy_score(y_test, y_pred))

plt.figure()
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.savefig('./no_param.png')

plt.figure()
save_learning_curve(filepath='./no_param_lc.png',
    base_classifier=classifier, title="Learning curve for default
    Gaussian HMM", X=X, y=y)

```

```
onfusion Matrix:
[107  16]
[  4 215]]
------------------------------------------------------------
------------------------------------------------------------
erformance Evaluation:
              precision    recall  f1-score   support

           0       0.96      0.87      0.91       123
           1       0.93      0.98      0.96       219

    accuracy                           0.94       342
   macro avg       0.95      0.93      0.94       342
eighted avg       0.94      0.94      0.94       342

------------------------------------------------------------
------------------------------------------------------------
ccuracy Score:
.9415204678362573
```

GAUSSIAN HMM (NO PARAMETER TUNING)- CONFUSION MATRIX AND LEARNING CURVE





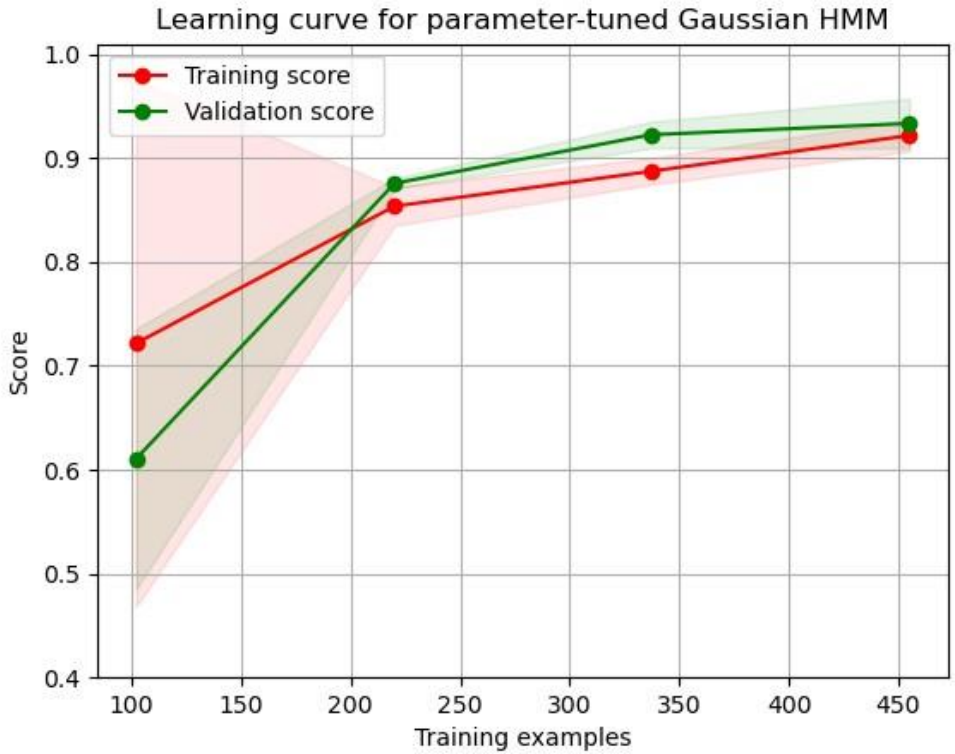
GAUSSIAN HMM (WITH PARAMETER TUNING)- CODE & EVALUATION

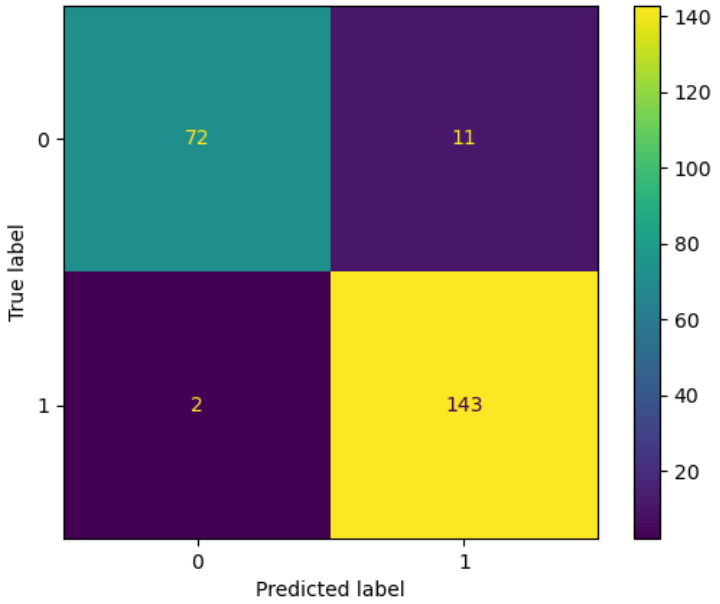
gaussian_param ⟩ No Selection

```python
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from hmmlearn import hmm
from sklearn.metrics import classification_report, confusion_matrix,
    accuracy_score, ConfusionMatrixDisplay
from helper_functions import *

NUM_CLASSES = 2

X, y = load_breast_cancer(return_X_y=True)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
    random_state=0)
classifier = hmm.GaussianHMM(n_components=NUM_CLASSES,
    covariance_type="full", min_covar=0.001, tol=0.01,
    implementation="log", random_state=0)
classifier.fit(X_train)

mapping = get_mapping(classifier, X_train, y_train)
y_pred = list(map(lambda hs : mapping[hs], classifier.predict(X_test)))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("--------------------------------------------------------")
print("--------------------------------------------------------")
print("Performance Evaluation:")
print(classification_report(y_test, y_pred))
print("--------------------------------------------------------")
print("--------------------------------------------------------")
print("Accuracy Score:")
print(accuracy_score(y_test, y_pred))

plt.figure()
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.savefig('./param.png')

plt.figure()
save_learning_curve(filepath='./param_lc.png',
    base_classifier=classifier, title="Learning curve for parameter-tuned
    Gaussian HMM", X=X, y=y)

```

```
Confusion Matrix:
[[ 72  11]
 [  2 143]]
--------------------------------------------------------
--------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           0       0.97      0.87      0.92        83
           1       0.93      0.99      0.96       145

    accuracy                           0.94       228
   macro avg       0.95      0.93      0.94       228
weighted avg       0.94      0.94      0.94       228

--------------------------------------------------------
--------------------------------------------------------
Accuracy Score:
0.9429824561403509
```

GAUSSIAN HMM (WITH PARAMETER TUNING)- CONFUSION MATRIX AND LEARNING CURVE





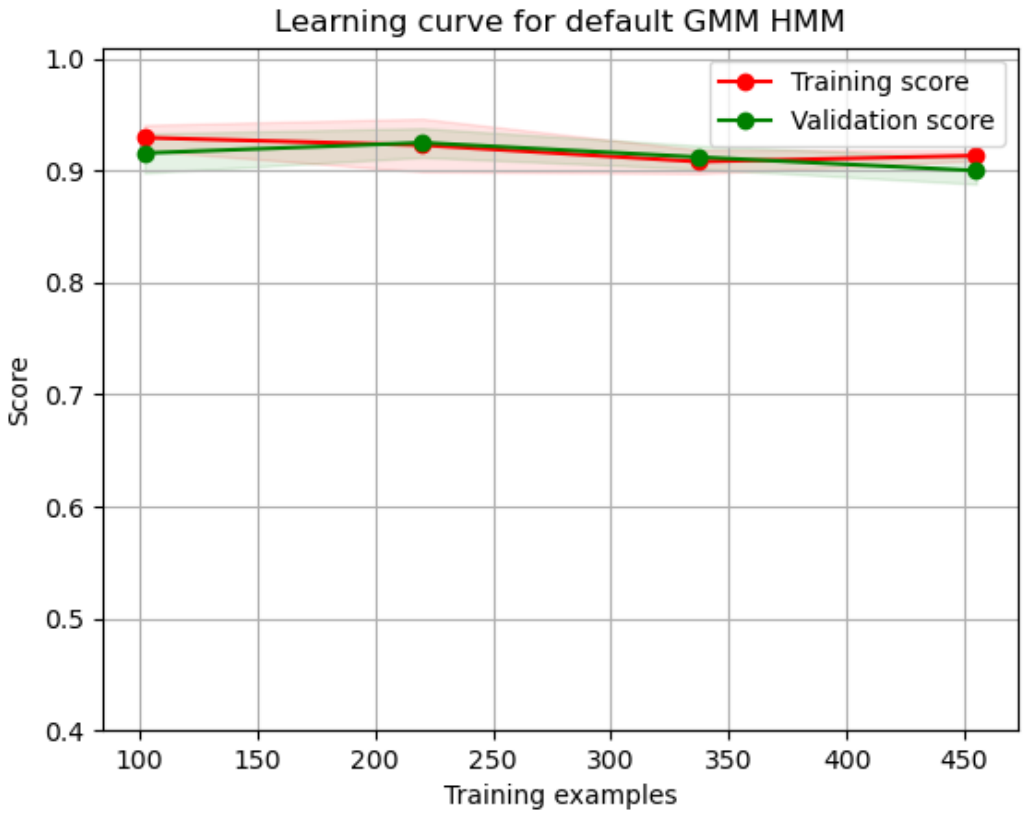
HMM WITH GAUSSIAN MIXTURE MODEL EMISSIONS

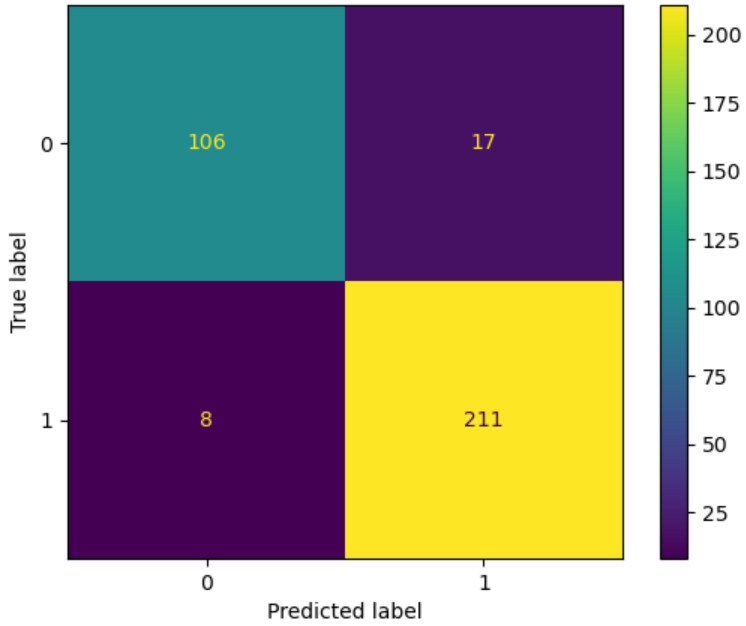
GMM HMM (NO PARAMETER TUNING)- CODE & EVALUATION

```python
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from hmmlearn import hmm
from sklearn.metrics import classification_report, confusion_matrix,
    accuracy_score, ConfusionMatrixDisplay
from helper_functions import *

NUM_CLASSES = 2

X, y = load_breast_cancer(return_X_y=True)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.6,
    random_state=0)
classifier = hmm.GMMHMM(n_components=NUM_CLASSES)
classifier.fit(X_train)

mapping = get_mapping(classifier, X_train, y_train)
y_pred = list(map(lambda hs : mapping[hs], classifier.predict(X_test)))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("-------------------------------------------------------------")
print("-------------------------------------------------------------")
print("Performance Evaluation:")
print(classification_report(y_test, y_pred))
print("-------------------------------------------------------------")
print("-------------------------------------------------------------")
print("Accuracy Score:")
print(accuracy_score(y_test, y_pred))

plt.figure()
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.savefig('./no_param.png')

plt.figure()
save_learning_curve(filepath='./no_param_lc.png',
    base_classifier=classifier, title="Learning curve for default GMM
    HMM", X=X, y=y, ylim=(0.4, 1.01))
```

```
onfusion matrix:
[106  17]
[  8 211]]
------------------------------------------------------------
------------------------------------------------------------
erformance Evaluation:
              precision    recall  f1-score   support

           0       0.93      0.86      0.89       123
           1       0.93      0.96      0.94       219

    accuracy                           0.93       342
   macro avg       0.93      0.91      0.92       342
eighted avg       0.93      0.93      0.93       342

------------------------------------------------------------
------------------------------------------------------------
ccuracy Score:
.9269005847953217
```

GMM HMM (NO PARAMETER TUNING)- CONFUSION MATRIX AND LEARNING CURVE



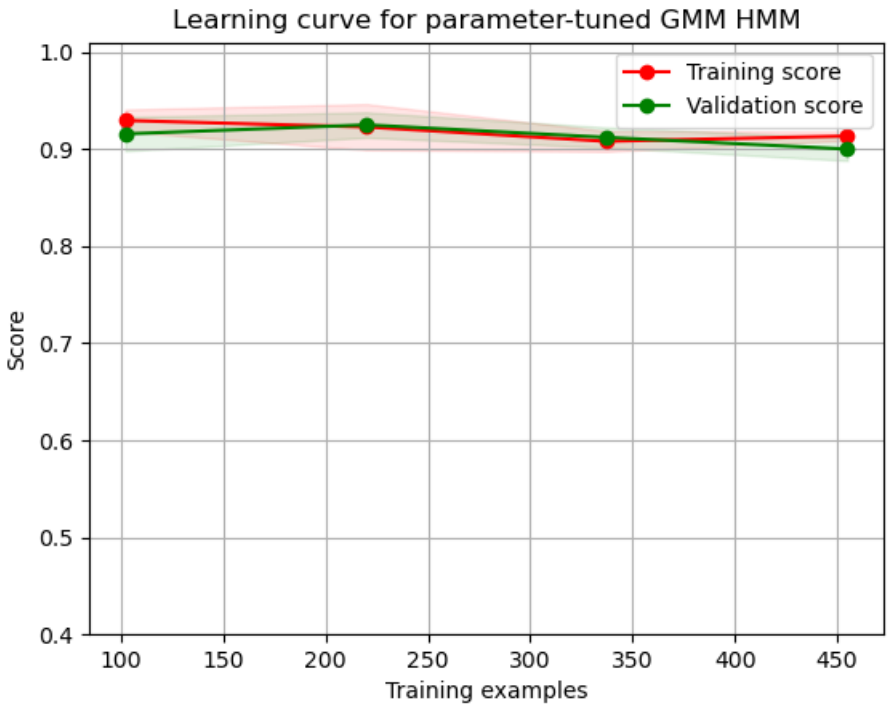


GMM HMM (WITH PARAMETER TUNING)- CODE & EVALUATION

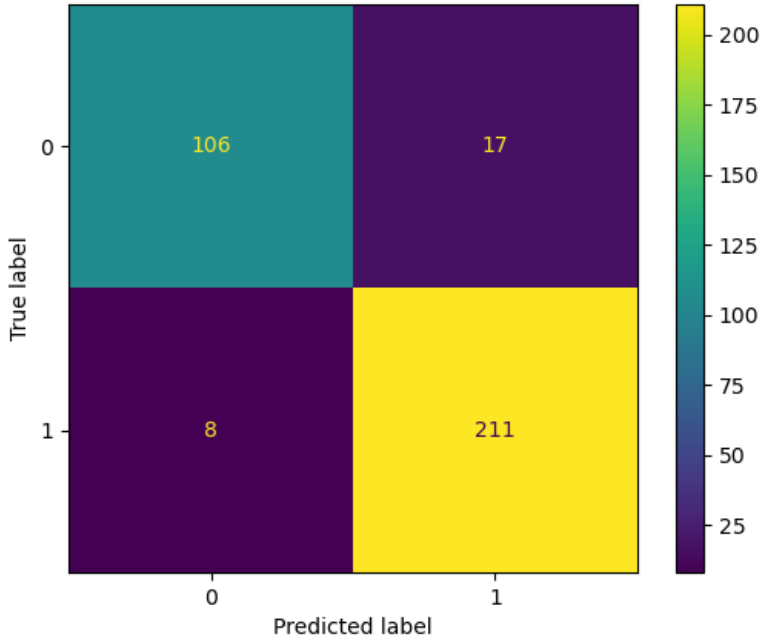
```python
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from hmmlearn import hmm
from sklearn.metrics import classification_report, confusion_matrix,
    accuracy_score, ConfusionMatrixDisplay
from helper_functions import *

NUM_CLASSES = 2

X, y = load_breast_cancer(return_X_y=True)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.6,
    random_state=0)
classifier = hmm.GMMHMM(n_components=NUM_CLASSES, n_mix=1,
    covariance_type="diag", min_covar=0.001, tol=0.01,
    implementation="log")
classifier.fit(X_train)

mapping = get_mapping(classifier, X_train, y_train)
y_pred = list(map(lambda hs : mapping[hs], classifier.predict(X_test)))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("--------------------------------------------------------")
print("--------------------------------------------------------")
print("Performance Evaluation:")
print(classification_report(y_test, y_pred))
print("--------------------------------------------------------")
print("--------------------------------------------------------")
print("Accuracy Score:")
print(accuracy_score(y_test, y_pred))

plt.figure()
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.savefig('./param.png')

plt.figure()
save_learning_curve(filepath='./param_lc.png',
    base_classifier=classifier, title="Learning curve for parameter-tuned
    GMM HMM", X=X, y=y)

```

```
onfusion Matrix:
[106  17]
[  8 211]]
--------------------------------------------------------
--------------------------------------------------------
erformance Evaluation:
              precision    recall  f1-score   support

           0       0.93      0.86      0.89       123
           1       0.93      0.96      0.94       219

    accuracy                           0.93       342
   macro avg       0.93      0.91      0.92       342
eighted avg       0.93      0.93      0.93       342

--------------------------------------------------------
--------------------------------------------------------
ccuracy Score:
.9269005847953217
```

GMM HMM (WITH PARAMETER TUNING)- CONFUSION MATRIX AND LEARNING CURVE

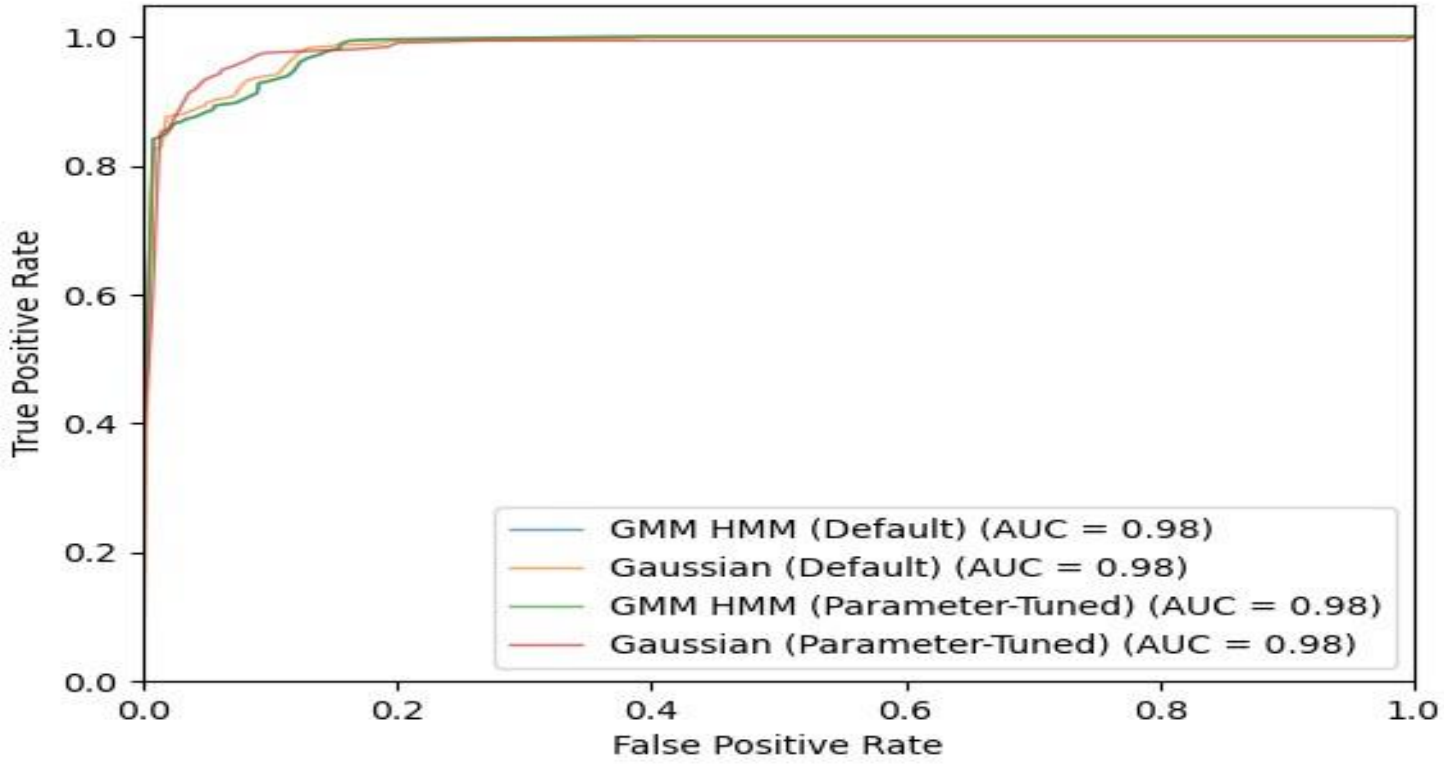


CONFUSION MATRIX



COMPARISON BETWEEN GAUSSIAN HMM AND GMM HMM.

ROC CURVES AND AUC COMPARISON



ROC CURVES AND AUC COMPARISON - CODE

make_roc_auc_curves

make_roc_auc_curves 〉 No Selection

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
from hmmlearn import hmm
from helper_functions import *

NUM_CLASSES = 2

X, y = load_breast_cancer(return_X_y=True)

classifier = dict()
classifier['gaussian_no_param'] =
    hmm.GaussianHMM(n_components=NUM_CLASSES)
classifier['gaussian_param'] =  hmm.GaussianHMM(n_components=NUM_CLASSES,
    covariance_type="full", min_covar=0.001, tol=0.01,
    implementation="log", random_state=0)
classifier['gmm_hmm_no_param'] = hmm.GMMHMM(n_components=NUM_CLASSES)
classifier['gmm_hmm_param'] = hmm.GMMHMM(n_components=NUM_CLASSES,
    n_mix=1, covariance_type="diag", min_covar=0.001, tol=0.01,
    implementation="log")


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.6,
    random_state=0)
plot_macro_avg_roc_curve(classifier['gmm_hmm_no_param'],
    n_classes=NUM_CLASSES, name="GMM HMM (Default)", X_train=X_train,
    X_test=X_test, y_train=y_train, y_test=y_test)
plot_macro_avg_roc_curve(classifier['gaussian_no_param'],
    n_classes=NUM_CLASSES, name="Gaussian (Default)", X_train=X_train,
    X_test=X_test, y_train=y_train, y_test=y_test)
plot_macro_avg_roc_curve(classifier['gmm_hmm_param'],
    n_classes=NUM_CLASSES, name="GMM HMM (Parameter-Tuned)",
    X_train=X_train, X_test=X_test, y_train=y_train, y_test=y_test)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
    random_state=0)

plot_macro_avg_roc_curve(classifier['gaussian_param'],
    n_classes=NUM_CLASSES, name="Gaussian (Parameter-Tuned)",
    X_train=X_train, X_test=X_test, y_train=y_train, y_test=y_test)


plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")

plt.savefig('roc_auc.png')
```

COMPARISON OF PERFORMANCE

CLASSIFIER		Accuracy	P ec	Recall	F -Sco	AUC
GAUSSIAN HMM	DEFAULT	0.94	0.94	0.94	0.94	0.98
	TUNED	0.94	0.94	0.94	0.94	0.98
GMM HMM	DEFAULT	0.93	0.93	0.93	0.93	0.98
	TUNED	0.93	0.93	0.93	0.93	0.98

PART 2 , 3

CIFAR - CLASSIFICATION

CNN

IMPORTS AND DATA PREPROCESSING

Convolutional Neural Network

Importing the libraries

In [4]:
```
import numpy as np
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers, datasets
from tensorflow.keras.models import Sequential
import matplotlib.pyplot as plt
```

Part 1 - Data Preprocessing

Normalization and augmentation will be done in the CNN itself

In [5]:
```
img_height = 32
img_width = 32
num_classes = 10
```

In [6]:
```
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()
```

MODEL ARCHITECTURE

Building the CNN

In [38]:
```
cnn = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(filters=32, kernel_size=3, padding='same', activation='relu'),
  layers.Conv2D(filters=32, kernel_size=3, padding='same', activation='relu'),
  layers.MaxPooling2D(pool_size=2, strides=2),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(pool_size=2, strides=2),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes, activation="softmax")
])
```

COMPILATION AND SUMMARY

Compiling the CNN

In [39]:
```
cnn.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0005, epsilon=0.01),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
```

In [40]:
```
cnn.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 32, 32, 3)         0

 conv2d_10 (Conv2D)          (None, 32, 32, 32)        896

 conv2d_11 (Conv2D)          (None, 32, 32, 32)        9248

 max_pooling2d_8 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 conv2d_12 (Conv2D)          (None, 16, 16, 32)        9248

 max_pooling2d_9 (MaxPooling  (None, 8, 8, 32)         0
 2D)

 dropout_7 (Dropout)         (None, 8, 8, 32)          0

 flatten_2 (Flatten)         (None, 2048)              0

 dense_8 (Dense)             (None, 128)               262272

 dense_9 (Dense)             (None, 10)                1290

=================================================================
Total params: 282,954
Trainable params: 282,954
Non-trainable params: 0
_________________________________________________________________
```

TRAINING AND SAVING

Training the CNN on the Training set and evaluating it on the Test set

In [41]:
```
history = cnn.fit(train_images, train_labels, epochs=20, validation_data=(test_images, test_labels))
```

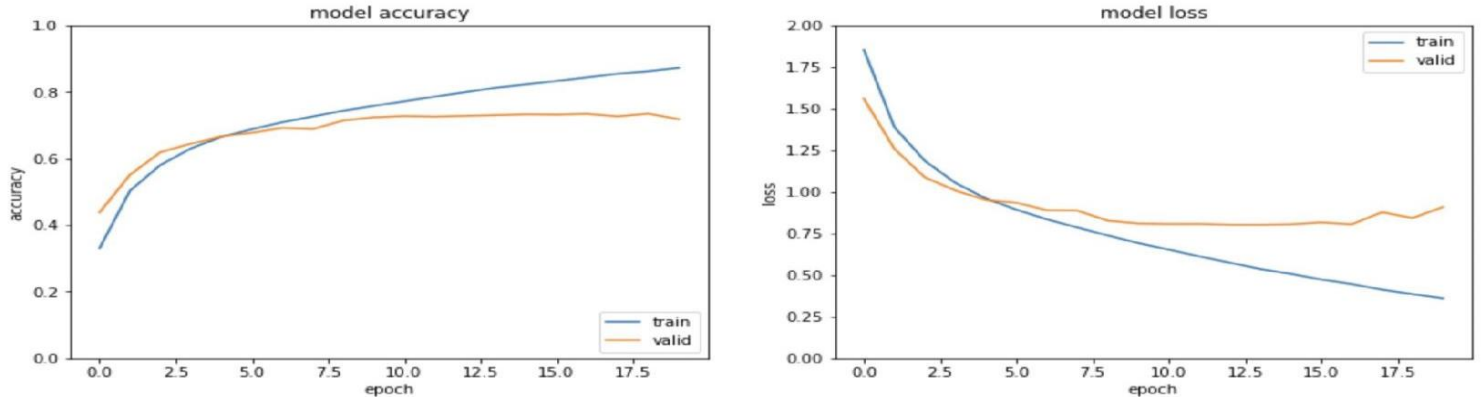
```
al_accuracy: 0.7304
Epoch 15/20
1563/1563 [==============================] - 52s 33ms/step - loss: 0.5073 - accuracy: 0.8229 - val_loss: 0.8052 - v
al_accuracy: 0.7329
Epoch 16/20
1563/1563 [==============================] - 50s 32ms/step - loss: 0.4732 - accuracy: 0.8331 - val_loss: 0.8167 - v
al_accuracy: 0.7322
Epoch 17/20
1563/1563 [==============================] - 54s 34ms/step - loss: 0.4459 - accuracy: 0.8441 - val_loss: 0.8052 - v
al_accuracy: 0.7342
Epoch 18/20
1563/1563 [==============================] - 53s 34ms/step - loss: 0.4118 - accuracy: 0.8551 - val_loss: 0.8775 - v
al_accuracy: 0.7267
Epoch 19/20
1563/1563 [==============================] - 54s 35ms/step - loss: 0.3850 - accuracy: 0.8624 - val_loss: 0.8425 - v
al_accuracy: 0.7348
Epoch 20/20
1563/1563 [==============================] - 54s 34ms/step - loss: 0.3596 - accuracy: 0.8725 - val_loss: 0.9090 - v
al_accuracy: 0.7184
```

In [42]:
```
cnn.save('cnn.h5')
```

TRAINING AND LOSS CURVES

In [47]:
```
def save_train_loss_curves(history, filename):
    plt.figure(figsize=(15,5))
    plt.subplot(1,2,1)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.ylim(0, 1)
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'valid'], loc='lower right')
    plt.subplot(1,2,2)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'valid'], loc='upper right')
    plt.ylim([0,2])
    plt.savefig(filename)
```

In [48]:
```
save_train_loss_curves(history, 'cnn_loss_acc.png')
```



PERFORMANCE - CODE

Performance

In [44]:
```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, ConfusionMatrixDisplay
```

In [45]:
```
def get_predictions(model, inputs):

    def index_max_arr_element(arr):
        best_index = -1
        best = float('-inf')
        for i, num in enumerate(arr):
            if num > best:
                best = num
                best_index = i
        return best_index

    pred_arr = model.predict(inputs)
    pred_labels = [index_max_arr_element(row) for row in pred_arr]
    return pred_labels
```

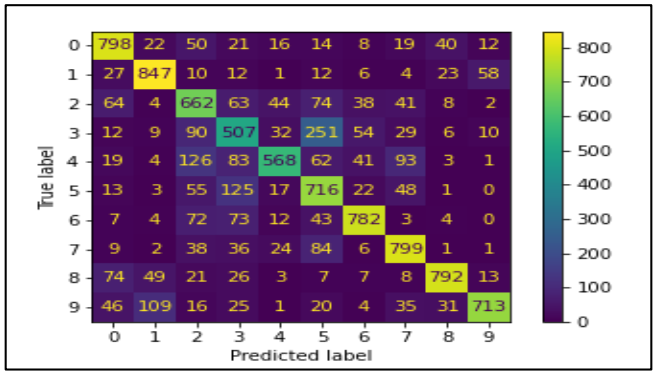
In [46]:
```
pred_labels = get_predictions(cnn, test_images)

print("Confusion Matrix:")
print(confusion_matrix(test_labels, pred_labels))
print("------------------------------------------------------------")
print("------------------------------------------------------------")
print("Performance Evaluation:")
print(classification_report(test_labels, pred_labels))
print("------------------------------------------------------------")
print("------------------------------------------------------------")
print("Accuracy Score:")
print(accuracy_score(test_labels, pred_labels))

plt.figure()
ConfusionMatrixDisplay.from_predictions(test_labels, pred_labels)
plt.savefig('./cnn_matrix.png')
```

```
  9/313 [..............................] - ETA: 4s
```

```
2022-09-16 18:28:02.208514: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113] Plugin op
timizer for device type GPU is enabled.
```

PERFORMANCE

```
2022-09-16 18:28:02.208514: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113] Plugin op
timizer for device_type GPU is enabled.

313/313 [==============================] - 3s 8ms/step
Confusion Matrix:
[[798  22  50  21  16  14   8  19  40  12]
 [ 27 847  10  12   1  12   6   4  23  58]
 [ 64   4 662  63  44  74  38  41   8   2]
 [ 12   9  90 507  32 251  54  29   6  10]
 [ 19   4 126  83 568  62  41  93   3   1]
 [ 13   3  55 125  17 716  22  48   1   0]
 [  7   4  72  73  12  43 782   3   4   0]
 [  9   2  38  36  24  84   6 799   1   1]
 [ 74  49  21  26   3   7   7   8 792  13]
 [ 46 109  16  25   1  20   4  35  31 713]]
------------------------------------------------------------
------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           0       0.75      0.80      0.77      1000
           1       0.80      0.85      0.83      1000
           2       0.58      0.66      0.62      1000
           3       0.52      0.51      0.51      1000
           4       0.79      0.57      0.66      1000
           5       0.56      0.72      0.63      1000
           6       0.81      0.78      0.79      1000
           7       0.74      0.80      0.77      1000
           8       0.87      0.79      0.83      1000
           9       0.88      0.71      0.79      1000

    accuracy                           0.72     10000
   macro avg       0.73      0.72      0.72     10000
weighted avg       0.73      0.72      0.72     10000

------------------------------------------------------------
------------------------------------------------------------
Accuracy Score:
0.7184
```



VGG-16

IMPORTS, DATA PREPROCESSING & ARCHITECTURE

VGG-16

In [49]:
```
from tensorflow.keras.applications import vgg16
```

In [50]:
```
train_images_p, test_images_p = vgg16.preprocess_input(train_images), vgg16.preprocess_input(test_images)
```

In [51]:
```
vgg_based_model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  vgg16.VGG16(include_top=False, weights="imagenet", input_shape=(img_height, img_width, 3), pooling="max"),
  layers.Flatten(),
  layers.Dense(512, activation=('relu')),
  layers.Dropout(0.2),
  layers.Dense(128, activation=('relu')),
  layers.Dropout(0.2),
  layers.Dense(num_classes, activation="softmax")
])
```

COMPILATION AND SUMMARY

In [52]:
```
vgg_based_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0002, epsilon=0.01),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
```

In [53]:
```
vgg_based_model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_2 (Rescaling)     (None, 32, 32, 3)         0

 vgg16 (Functional)          (None, 512)               14714688

 flatten_3 (Flatten)         (None, 512)               0

 dense_10 (Dense)            (None, 512)               262656

 dropout_8 (Dropout)         (None, 512)               0

 dense_11 (Dense)            (None, 128)               65664

 dropout_9 (Dropout)         (None, 128)               0

 dense_12 (Dense)            (None, 10)                1290

=================================================================
Total params: 15,044,298
Trainable params: 15,044,298
Non-trainable params: 0
_________________________________________________________________
```

TRAINING AND SAVING

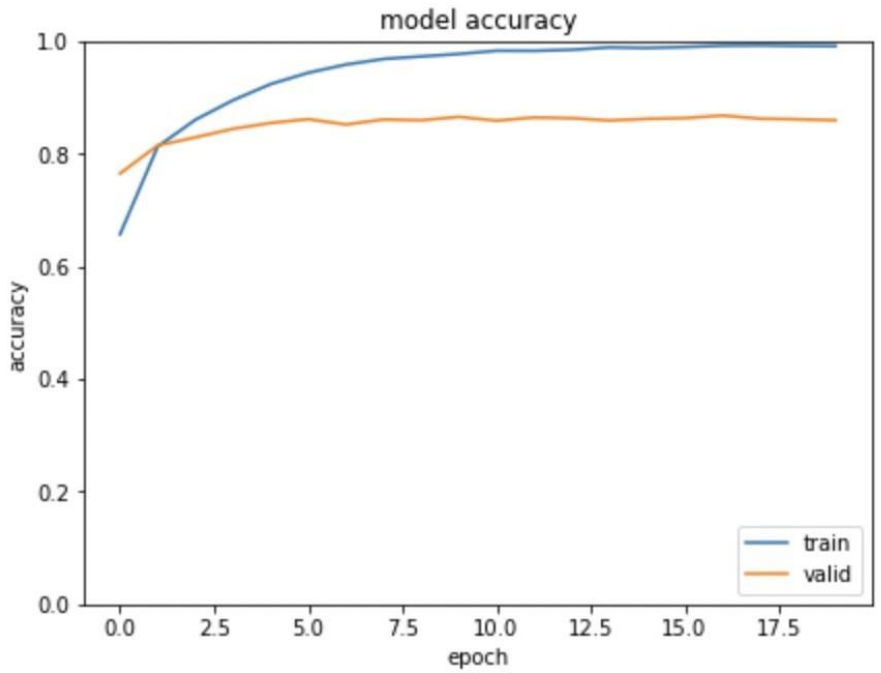
In [54]:
```
history = vgg_based_model.fit(train_images_p, train_labels, epochs=20, validation_data=(test_images_p, test_labels))
```

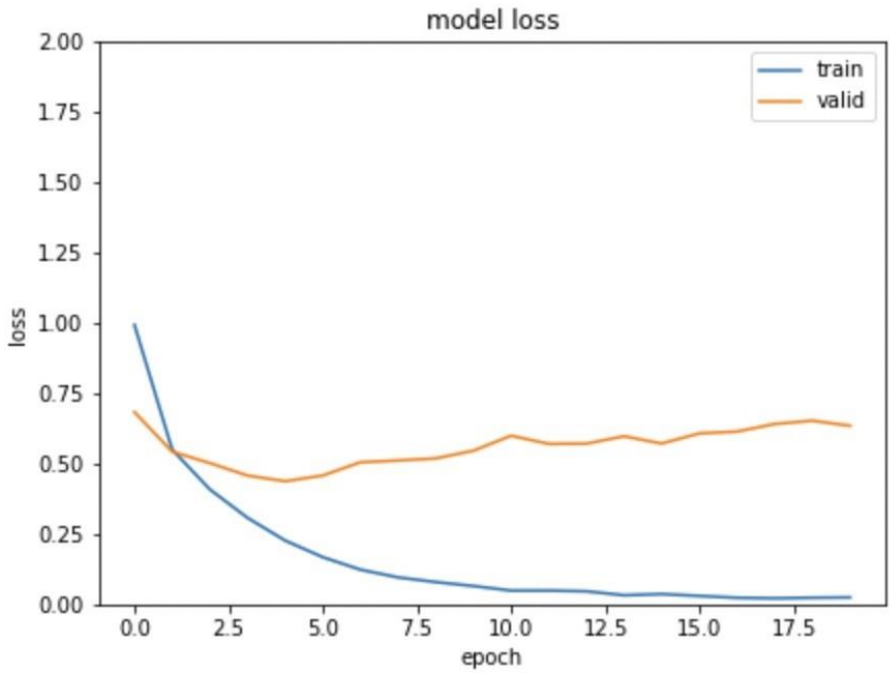
```
val_accuracy: 0.8596
Epoch 15/20
1563/1563 [==============================] - 288s 184ms/step - loss: 0.0361 - accuracy: 0.9885 - val_loss: 0.5713 -
val_accuracy: 0.8625
Epoch 16/20
1563/1563 [==============================] - 282s 181ms/step - loss: 0.0296 - accuracy: 0.9902 - val_loss: 0.6073 -
val_accuracy: 0.8638
Epoch 17/20
1563/1563 [==============================] - 297s 190ms/step - loss: 0.0230 - accuracy: 0.9925 - val_loss: 0.6136 -
val_accuracy: 0.8684
Epoch 18/20
1563/1563 [==============================] - 292s 187ms/step - loss: 0.0211 - accuracy: 0.9929 - val_loss: 0.6407 -
val_accuracy: 0.8628
Epoch 19/20
1563/1563 [==============================] - 295s 188ms/step - loss: 0.0232 - accuracy: 0.9922 - val_loss: 0.6523 -
val_accuracy: 0.8617
Epoch 20/20
1563/1563 [==============================] - 293s 187ms/step - loss: 0.0244 - accuracy: 0.9918 - val_loss: 0.6342 -
val_accuracy: 0.8599
```

In [56]:
```
vgg_based_model.save('vgg16.h5')
```

TRAINING AND LOSS CURVES

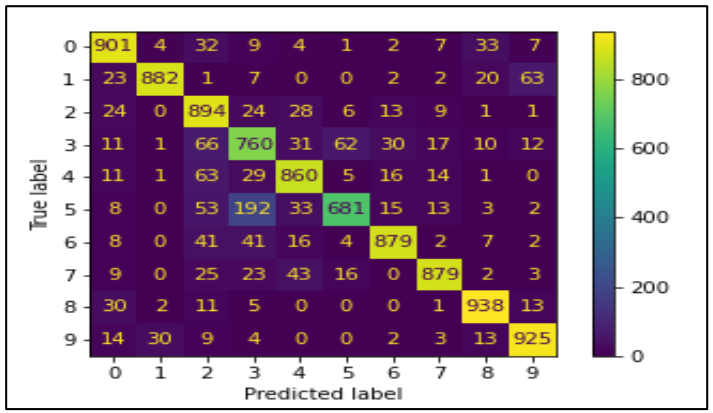
```
In [55]: save_train_loss_curves(history, 'vgg_loss_acc.png')
```





PERFORMANCE

```
2022-09-16 23:40:57.867625: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.

313/313 [==============================] - 5s 15ms/step
Confusion Matrix:
[[901   4  32   9   4   1   2   7  33   7]
 [ 23 882   1   7   0   0   2   2  20  63]
 [ 24   0 894  24  28   6  13   9   1   1]
 [ 11   1  66 760  31  62  30  17  10  12]
 [ 11   1  63  29 860   5  16  14   1   0]
 [  8   0  53 192  33 681  15  13   3   2]
 [  8   0  41  41  16   4 879   2   7   2]
 [  9   0  25  23  43  16   0 879   2   3]
 [ 30   2  11   5   0   0   0   1 938  13]
 [ 14  30   9   4   0   0   2   3  13 925]]
------------------------------------------------------------
------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           0       0.87      0.90      0.88      1000
           1       0.96      0.88      0.92      1000
           2       0.75      0.89      0.81      1000
           3       0.69      0.76      0.73      1000
           4       0.85      0.86      0.85      1000
           5       0.88      0.68      0.77      1000
           6       0.92      0.88      0.90      1000
           7       0.93      0.88      0.90      1000
           8       0.91      0.94      0.93      1000
           9       0.90      0.93      0.91      1000

    accuracy                           0.86     10000
   macro avg       0.87      0.86      0.86     10000
weighted avg       0.87      0.86      0.86     10000

------------------------------------------------------------
------------------------------------------------------------
Accuracy Score:
0.8599
```



RESNET-50

IMPORTS, DATA PREPROCESSING & ARCHITECTURE

RESNET-50

In [57]:
```
from tensorflow.keras.applications import resnet50
```

In [58]:
```
train_images_pr, test_images_pr = resnet50.preprocess_input(train_images), resnet50.preprocess_input(test_images)
```

In [59]:
```
resnet_based_model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  resnet50.ResNet50(include_top=False, weights="imagenet", input_shape=(img_height, img_width, 3), pooling="max"),
  layers.Flatten(),
  layers.Dense(512, activation=('relu')),
  layers.Dropout(0.2),
  layers.Dense(128, activation=('relu')),
  layers.Dropout(0.2),
  layers.Dense(num_classes, activation="softmax")
])
```

COMPILATION AND SUMMARY

In [60]:
```
resnet_based_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0002, epsilon=0.01),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
```

In [61]:
```
resnet_based_model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_3 (Rescaling)     (None, 32, 32, 3)         0

 resnet50 (Functional)       (None, 2048)              23587712

 flatten_4 (Flatten)         (None, 2048)              0

 dense_13 (Dense)            (None, 512)               1049088

 dropout_10 (Dropout)        (None, 512)               0

 dense_14 (Dense)            (None, 128)               65664

 dropout_11 (Dropout)        (None, 128)               0

 dense_15 (Dense)            (None, 10)                1290

=================================================================
Total params: 24,703,754
Trainable params: 24,650,634
Non-trainable params: 53,120
_________________________________________________________________
```

TRAINING AND SAVING

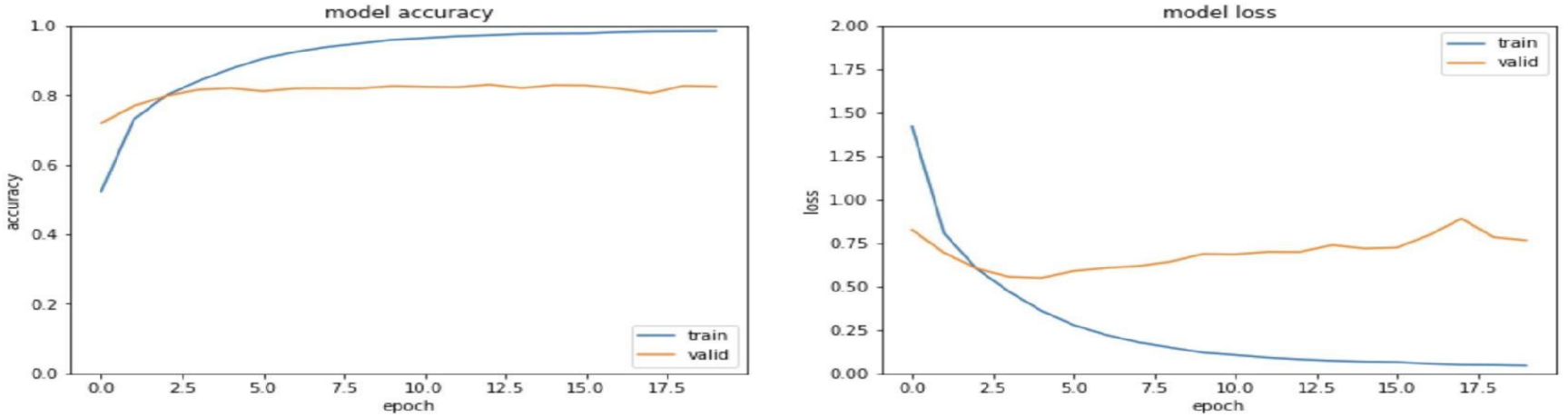
```
In [62]: history = resnet_based_model.fit(train_images_pr, train_labels, epochs=20, validation_data=(test_images_pr, test_lab
```

```
1563/1563 [==============================] - 212s 136ms/step - loss: 0.0728 - accuracy: 0.9768 - val_loss: 0.7386 -
val_accuracy: 0.8207
Epoch 15/20
1563/1563 [==============================] - 211s 135ms/step - loss: 0.0672 - accuracy: 0.9777 - val_loss: 0.7188 -
val_accuracy: 0.8294
Epoch 16/20
1563/1563 [==============================] - 213s 136ms/step - loss: 0.0651 - accuracy: 0.9784 - val_loss: 0.7238 -
val_accuracy: 0.8285
Epoch 17/20
1563/1563 [==============================] - 215s 137ms/step - loss: 0.0551 - accuracy: 0.9820 - val_loss: 0.7959 -
val_accuracy: 0.8196
Epoch 18/20
1563/1563 [==============================] - 215s 138ms/step - loss: 0.0494 - accuracy: 0.9839 - val_loss: 0.8906 -
val_accuracy: 0.8058
Epoch 19/20
1563/1563 [==============================] - 231s 148ms/step - loss: 0.0482 - accuracy: 0.9843 - val_loss: 0.7835 -
val_accuracy: 0.8271
Epoch 20/20
1563/1563 [==============================] - 279s 179ms/step - loss: 0.0450 - accuracy: 0.9849 - val_loss: 0.7657 -
val_accuracy: 0.8252
```

```
In [64]: resnet_based_model.save('resnet50.h5')
```

TRAINING AND LOSS CURVES

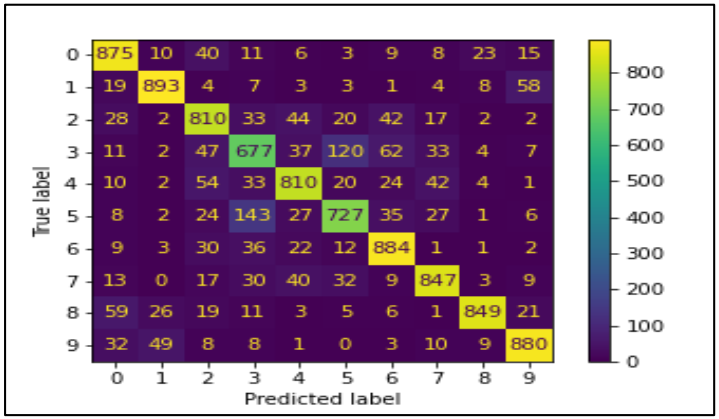
In [63]:
```
save_train_loss_curves(history, 'resnet_loss_acc.png')
```



In [64]:
```
resnet_based_model.save('resnet50.h5')
```

PERFORMANCE

```
2022-09-16 23:47:09.495012: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113] Plugin op
timizer for device_type GPU is enabled.

313/313 [==============================] - 9s 25ms/step
Confusion Matrix:
[[875  10  40  11   6   3   9   8  23  15]
 [ 19 893   4   7   3   3   1   4   8  58]
 [ 28   2 810  33  44  20  42  17   2   2]
 [ 11   2  47 677  37 120  62  33   4   7]
 [ 10   2  54  33 810  20  24  42   4   1]
 [  8   2  24 143  27 727  35  27   1   6]
 [  9   3  30  36  22  12 884   1   1   2]
 [ 13   0  17  30  40  32   9 847   3   9]
 [ 59  26  19  11   3   5   6   1 849  21]
 [ 32  49   8   8   1   0   3  10   9 880]]
--------------------------------------------------------------
--------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           0       0.82      0.88      0.85      1000
           1       0.90      0.89      0.90      1000
           2       0.77      0.81      0.79      1000
           3       0.68      0.68      0.68      1000
           4       0.82      0.81      0.81      1000
           5       0.77      0.73      0.75      1000
           6       0.82      0.88      0.85      1000
           7       0.86      0.85      0.85      1000
           8       0.94      0.85      0.89      1000
           9       0.88      0.88      0.88      1000

    accuracy                           0.83     10000
   macro avg       0.83      0.83      0.83     10000
weighted avg       0.83      0.83      0.83     10000

--------------------------------------------------------------
--------------------------------------------------------------
Accuracy Score:
0.8252
```



RNN (LSTM)

IMAGE PROCESSING AND MODEL ARCHITECTURE

RNN (LSTM based)

In [10]:
```
train_images_res, test_images_res = train_images.reshape(train_images.shape[0], 1024, 3), test_images.reshape(test_i
```

In [11]:
```
train_images_res.shape
```

Out[11]:
```
(50000, 1024, 3)
```

In [12]:
```
lstm_based_model = Sequential([
  layers.Rescaling(1./255, input_shape=(1024, 3)),
  layers.LSTM(units=50, return_sequences=True),
  layers.Dropout(0.2),
  layers.LSTM(units=50),
  layers.Dropout(0.2),
  layers.Dense(128, activation=('relu')),
  layers.Dropout(0.2),
  layers.Dense(num_classes, activation="softmax")
])
```

```
Metal device set to: Apple M1
```

```
2022-09-16 11:44:55.396775: I tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:305] Coul
d not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel may not have been built with NUMA suppo
rt.
2022-09-16 11:44:55.397008: I tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:271] Crea
ted TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0 MB memory) -> physical PluggableDevice (
device: 0, name: METAL, pci bus id: <undefined>)
```

COMPILATION AND SUMMARY

In [13]:
```
lstm_based_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001),
                         loss=tf.keras.losses.SparseCategoricalCrossentropy(),
                         metrics=['accuracy'])
```

In [15]:
```
lstm_based_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 1024, 3)           0

 lstm (LSTM)                 (None, 1024, 50)          10800

 dropout (Dropout)           (None, 1024, 50)          0

 lstm_1 (LSTM)               (None, 50)                20200

 dropout_1 (Dropout)         (None, 50)                0

 dense (Dense)               (None, 128)               6528

 dropout_2 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 38,818
Trainable params: 38,818
Non-trainable params: 0
_________________________________________________________________
```

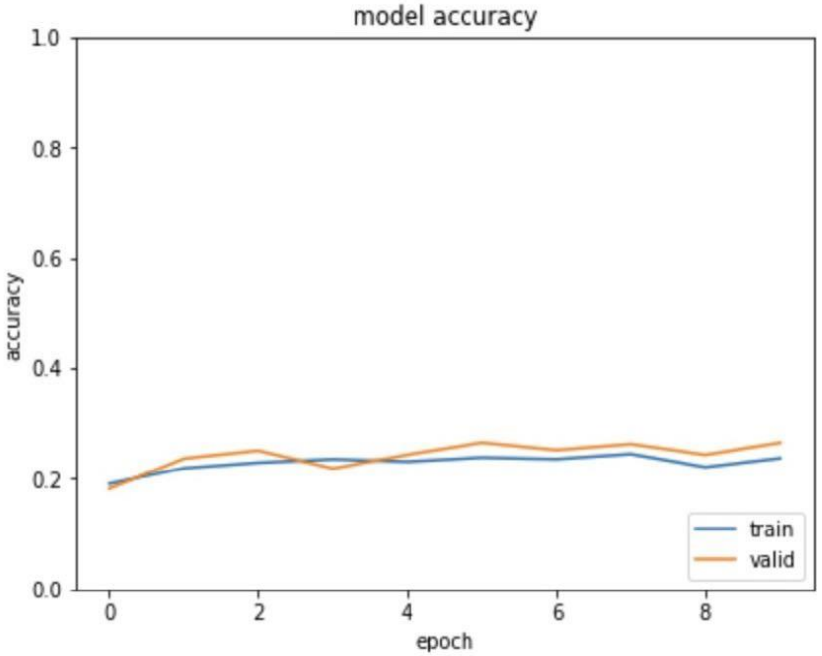
TRAINING AND SAVING

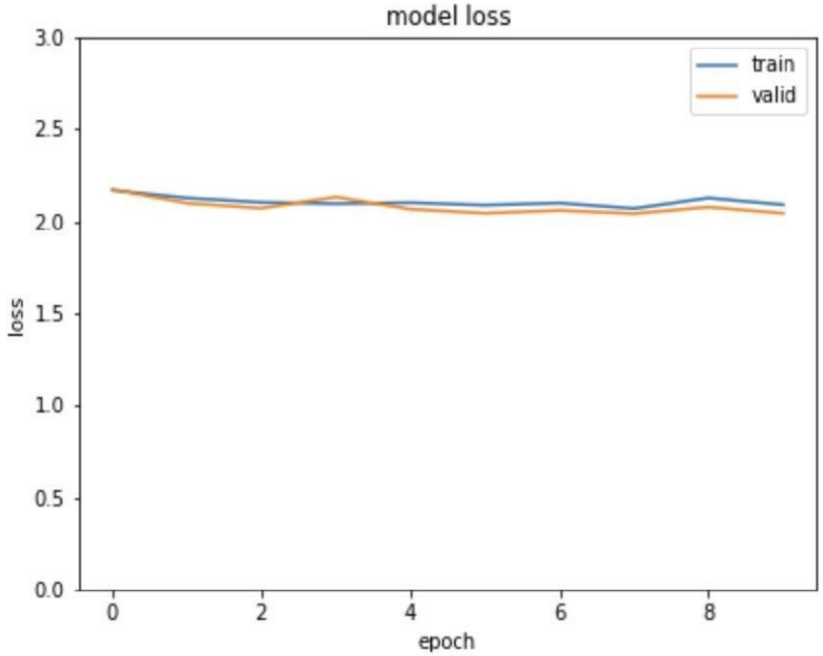
```
In [14]: history = lstm_based_model.fit(train_images_res, train_labels, epochs=10, validation_data=(test_images_res, test_lab
         val_accuracy: 0.2188
         Epoch 5/10
         1563/1563 [==============================] - 795s 509ms/step - loss: 2.1007 - accuracy: 0.2312 - val_loss: 2.0660 -
         val_accuracy: 0.2436
         Epoch 6/10
         1563/1563 [==============================] - 807s 516ms/step - loss: 2.0879 - accuracy: 0.2386 - val_loss: 2.0442 -
         val_accuracy: 0.2653
         Epoch 7/10
         1563/1563 [==============================] - 877s 561ms/step - loss: 2.0990 - accuracy: 0.2357 - val_loss: 2.0592 -
         val_accuracy: 0.2520
         Epoch 8/10
         1563/1563 [==============================] - 927s 593ms/step - loss: 2.0696 - accuracy: 0.2450 - val_loss: 2.0422 -
         val_accuracy: 0.2628
         Epoch 9/10
         1563/1563 [==============================] - 917s 587ms/step - loss: 2.1258 - accuracy: 0.2214 - val_loss: 2.0771 -
         val_accuracy: 0.2435
         Epoch 10/10
         1563/1563 [==============================] - 815s 522ms/step - loss: 2.0896 - accuracy: 0.2373 - val_loss: 2.0438 -
         val_accuracy: 0.2651
```

```
In [19]: lstm_based_model.save('lstm.h5')
```

TRAINING AND LOSS CURVES

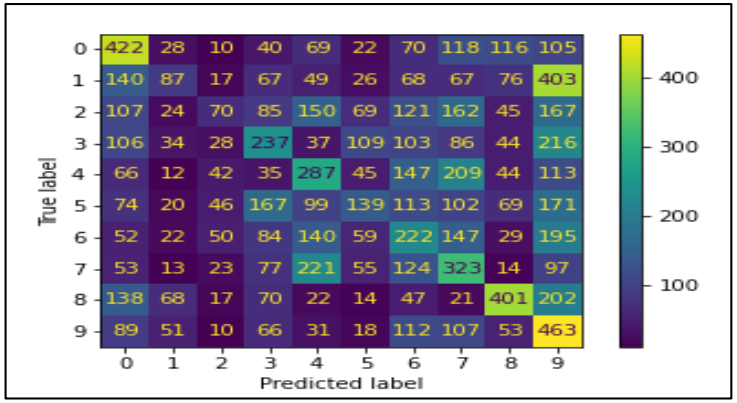
```
In [18]: save_train_loss_curves(history, 'lstm_loss_acc.png')
```





PERFORMANCE

```
2022-09-16 23:56:19.986646: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113] Plugin op
timizer for device_type GPU is enabled.
2022-09-16 23:56:23.320163: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113] Plugin op
timizer for device_type GPU is enabled.

313/313 [==============================] - 107s 318ms/step
Confusion Matrix:
[[422  28  10  40  69  22  70 118 116 105]
 [140  87  17  67  49  26  68  67  76 403]
 [107  24  70  85 150  69 121 162  45 167]
 [106  34  28 237  37 109 103  86  44 216]
 [ 66  12  42  35 287  45 147 209  44 113]
 [ 74  20  46 167  99 139 113 102  69 171]
 [ 52  22  50  84 140  59 222 147  29 195]
 [ 53  13  23  77 221  55 124 323  14  97]
 [138  68  17  70  22  14  47  21 401 202]
 [ 89  51  10  66  31  18 112 107  53 463]]
--------------------------------------------------------------
--------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           0       0.34      0.42      0.38      1000
           1       0.24      0.09      0.13      1000
           2       0.22      0.07      0.11      1000
           3       0.26      0.24      0.25      1000
           4       0.26      0.29      0.27      1000
           5       0.25      0.14      0.18      1000
           6       0.20      0.22      0.21      1000
           7       0.24      0.32      0.28      1000
           8       0.45      0.40      0.42      1000
           9       0.22      0.46      0.30      1000

    accuracy                           0.27     10000
   macro avg       0.27      0.27      0.25     10000
weighted avg       0.27      0.27      0.25     10000

--------------------------------------------------------------
--------------------------------------------------------------
Accuracy Score:
0.2651
```



ALEXNET

MODEL ARCHITECTURE - I

[30]:
```
def alexnet_model(img_shape=(32, 32, 3), n_classes=10, l2_reg=0.,
    weights=None):

    # Initialize model
    alexnet = Sequential()

    # Layer 1
    alexnet.add(Conv2D(96, (11, 11), input_shape=img_shape, padding='same', kernel_regularizer=l2(l2_reg)))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(MaxPooling2D(pool_size=(2, 2)))

    # Layer 2
    alexnet.add(Conv2D(256, (5, 5), padding='same'))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(MaxPooling2D(pool_size=(2, 2)))

    # Layer 3
    alexnet.add(ZeroPadding2D((1, 1)))
    alexnet.add(Conv2D(512, (3, 3), padding='same'))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(MaxPooling2D(pool_size=(2, 2)))

    # Layer 4
    alexnet.add(ZeroPadding2D((1, 1)))
    alexnet.add(Conv2D(1024, (3, 3), padding='same'))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))

    # Layer 5
    alexnet.add(ZeroPadding2D((1, 1)))
    alexnet.add(Conv2D(1024, (3, 3), padding='same'))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(MaxPooling2D(pool_size=(2, 2)))
```

MODEL ARCHITECTURE - II

```
    # Layer 6
    alexnet.add(Flatten())
    alexnet.add(Dense(3072))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(Dropout(0.5))

    # Layer 7
    alexnet.add(Dense(4096))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(Dropout(0.5))

    # Layer 8
    alexnet.add(Dense(n_classes))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('softmax'))

    if weights is not None:
        alexnet.load_weights(weights)

    return alexnet
```

[31]:
```
alexnet_model = alexnet_model(img_shape=(32, 32, 3), n_classes=10)
```

[32]:
```
alexnet_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0002, epsilon=0.05),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
```

TRAINING AND SAVING

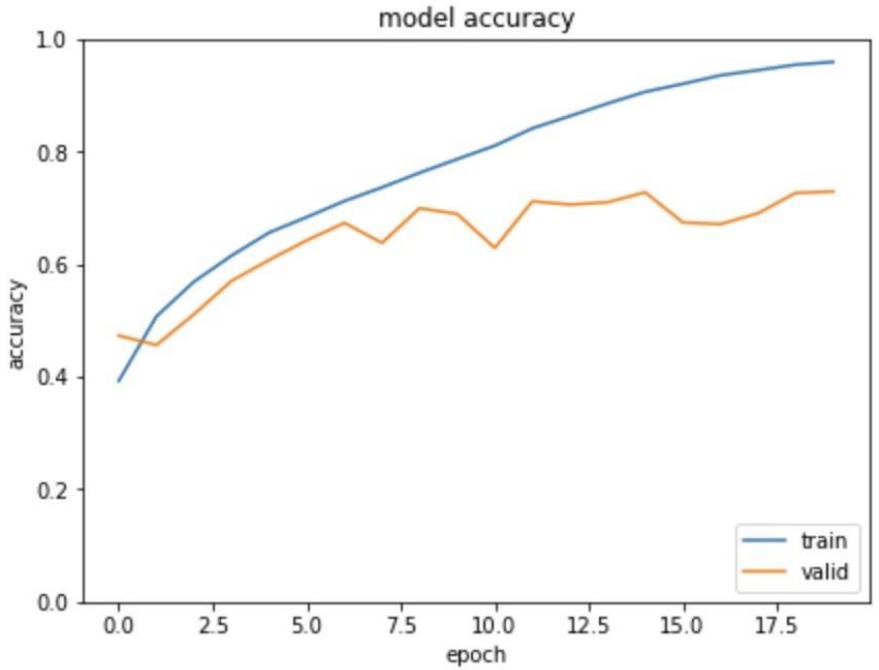
```
In [34]: history = alexnet_model.fit(train_images, train_labels, epochs=20, validation_data=(test_images, test_labels))
```

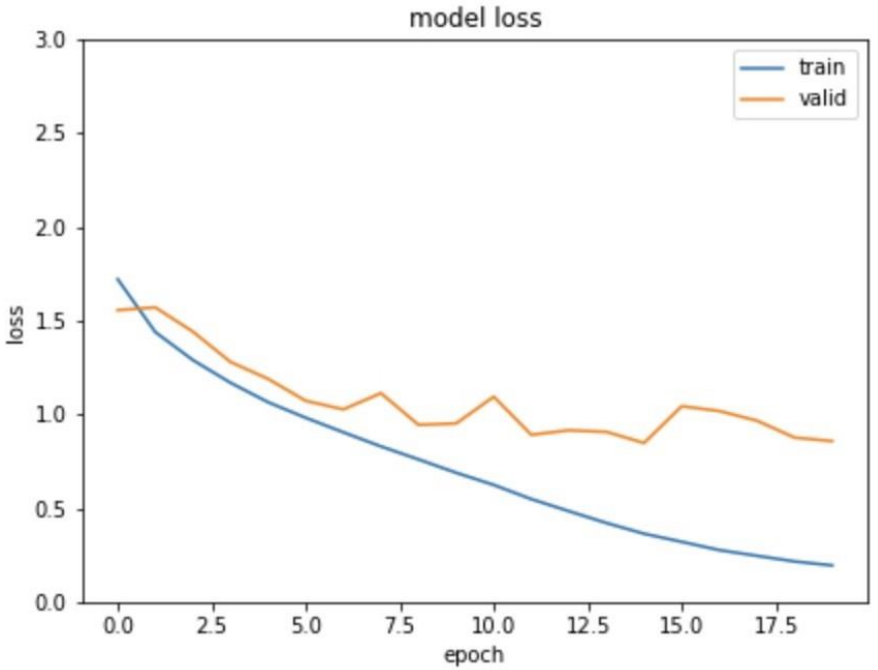
```
val_accuracy: 0.7104
Epoch 15/20
1563/1563 [==============================] - 639s 409ms/step - loss: 0.3634 - accuracy: 0.9068 - val_loss: 0.8472 -
val_accuracy: 0.7281
Epoch 16/20
1563/1563 [==============================] - 576s 369ms/step - loss: 0.3203 - accuracy: 0.9209 - val_loss: 1.0437 -
val_accuracy: 0.6746
Epoch 17/20
1563/1563 [==============================] - 569s 364ms/step - loss: 0.2760 - accuracy: 0.9361 - val_loss: 1.0177 -
val_accuracy: 0.6715
Epoch 18/20
1563/1563 [==============================] - 597s 382ms/step - loss: 0.2458 - accuracy: 0.9453 - val_loss: 0.9665 -
val_accuracy: 0.6907
Epoch 19/20
1563/1563 [==============================] - 656s 419ms/step - loss: 0.2156 - accuracy: 0.9549 - val_loss: 0.8762 -
val_accuracy: 0.7269
Epoch 20/20
1563/1563 [==============================] - 634s 406ms/step - loss: 0.1944 - accuracy: 0.9598 - val_loss: 0.8573 -
val_accuracy: 0.7295
```

```
In [37]: alexnet_model.save('alexnet.h5')
```

TRAINING AND LOSS CURVES

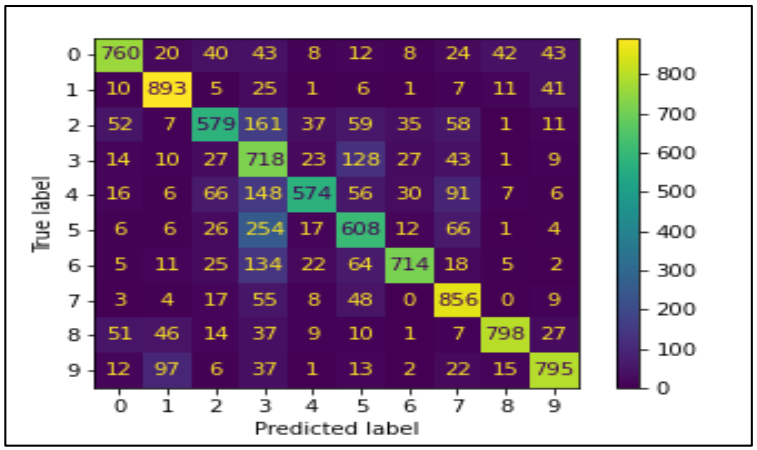
```
In [35]: save_train_loss_curves(history, 'alexnet_loss_acc.png')
```





PERFORMANCE

```
2022-09-17 00:04:14.825904: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.

313/313 [==============================] - 23s 72ms/step
Confusion Matrix:
[[760  20  40  43   8  12   8  24  42  43]
 [ 10 893   5  25   1   6   1   7  11  41]
 [ 52   7 579 161  37  59  35  58   1  11]
 [ 14  10  27 718  23 128  27  43   1   9]
 [ 16   6  66 148 574  56  30  91   7   6]
 [  6   6  26 254  17 608  12  66   1   4]
 [  5  11  25 134  22  64 714  18   5   2]
 [  3   4  17  55   8  48   0 856   0   9]
 [ 51  46  14  37   9  10   1   7 798  27]
 [ 12  97   6  37   1  13   2  22  15 795]]
--------------------------------------------------------------
--------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           0       0.82      0.76      0.79      1000
           1       0.81      0.89      0.85      1000
           2       0.72      0.58      0.64      1000
           3       0.45      0.72      0.55      1000
           4       0.82      0.57      0.68      1000
           5       0.61      0.61      0.61      1000
           6       0.86      0.71      0.78      1000
           7       0.72      0.86      0.78      1000
           8       0.91      0.80      0.85      1000
           9       0.84      0.80      0.82      1000

    accuracy                           0.73     10000
   macro avg       0.75      0.73      0.73     10000
weighted avg       0.75      0.73      0.73     10000

--------------------------------------------------------------
--------------------------------------------------------------
Accuracy Score:
0.7295
```



GOOGLENET

IMPORTS, DATA PREPROCESSING AND INCEPTION MODULE

GOOGLENET

In [9]:
```
from tensorflow.keras.layers import Flatten, Input, Conv2D, Dense, Dropout, MaxPooling2D, AveragePooling2D, GlobalAv
from tensorflow.keras.layers import concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.regularizers import l2
```

In [11]:
```
train_images_normalized = train_images/255.0
test_images_normalized = test_images/255.0
```

In [12]:
```
def inception(x,
              filters_1x1,
              filters_3x3_reduce,
              filters_3x3,
              filters_5x5_reduce,
              filters_5x5,
              filters_pool):
    path1 = layers.Conv2D(filters_1x1, (1, 1), padding='same',    activation='relu')(x)
    path2 = layers.Conv2D(filters_3x3_reduce, (1, 1), padding='same', activation='relu')(x)
    path2 = layers.Conv2D(filters_3x3, (1, 1), padding='same', activation='relu')(path2)
    path3 = layers.Conv2D(filters_5x5_reduce, (1, 1), padding='same', activation='relu')(x)
    path3 = layers.Conv2D(filters_5x5, (1, 1), padding='same', activation='relu')(path3)
    path4 = layers.MaxPool2D((3, 3), strides=(1, 1), padding='same')(x)
    path4 = layers.Conv2D(filters_pool, (1, 1), padding='same', activation='relu')(path4)
    return tf.concat([path1, path2, path3, path4], axis=3)
```

MODEL ARCHITECTURE

In [14]:
```
inp = layers.Input(shape=(32, 32, 3))
input_tensor = layers.experimental.preprocessing.Resizing(224, 224, interpolation="bilinear", input_shape=train_imag
x = layers.Conv2D(64, 7, strides=2, padding='same', activation='relu')(input_tensor)
x = layers.MaxPooling2D(3, strides=2)(x)
x = layers.Conv2D(64, 1, strides=1, padding='same', activation='relu')(x)
x = layers.Conv2D(192, 3, strides=1, padding='same', activation='relu')(x)
x = layers.MaxPooling2D(3, strides=2)(x)
x = inception(x, filters_1x1=64, filters_3x3_reduce=96, filters_3x3=128, filters_5x5_reduce=16, filters_5x5=32, filt
x = inception(x, filters_1x1=128, filters_3x3_reduce=128, filters_3x3=192, filters_5x5_reduce=32, filters_5x5=96, fi
x = layers.MaxPooling2D(3, strides=2)(x)
x = inception(x, filters_1x1=192, filters_3x3_reduce=96, filters_3x3=208, filters_5x5_reduce=16, filters_5x5=48, fil
aux1 = layers.AveragePooling2D((5, 5), strides=3)(x)
aux1 =layers.Conv2D(128, 1, padding='same', activation='relu')(aux1)
aux1 = layers.Flatten()(aux1)
aux1 = layers.Dense(1024, activation='relu')(aux1)
aux1 = layers.Dropout(0.7)(aux1)
aux1 = layers.Dense(10, activation='softmax')(aux1)
x = inception(x, filters_1x1=160, filters_3x3_reduce=112, filters_3x3=224, filters_5x5_reduce=24, filters_5x5=64, fi
x = inception(x, filters_1x1=128, filters_3x3_reduce=128, filters_3x3=256, filters_5x5_reduce=24, filters_5x5=64, fi
x = inception(x, filters_1x1=112, filters_3x3_reduce=144, filters_3x3=288, filters_5x5_reduce=32, filters_5x5=64, fi
aux2 = layers.AveragePooling2D((5, 5), strides=3)(x)
aux2 =layers.Conv2D(128, 1, padding='same', activation='relu')(aux2)
aux2 = layers.Flatten()(aux2)
aux2 = layers.Dense(1024, activation='relu')(aux2)
aux2 = layers.Dropout(0.7)(aux2)
aux2 = layers.Dense(10, activation='softmax')(aux2)
x = inception(x, filters_1x1=256, filters_3x3_reduce=160, filters_3x3=320, filters_5x5_reduce=32, filters_5x5=128, f
x = layers.MaxPooling2D(3, strides=2)(x)
x = inception(x, filters_1x1=256, filters_3x3_reduce=160, filters_3x3=320, filters_5x5_reduce=32, filters_5x5=128, f
x = inception(x, filters_1x1=384, filters_3x3_reduce=192, filters_3x3=384, filters_5x5_reduce=48, filters_5x5=128, f
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dropout(0.4)(x)
out = layers.Dense(10, activation='softmax')(x)
```

```
Metal device set to: Apple M1

2022-09-21 21:16:54.156905: I tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:305] Coul
d not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel may not have been built with NUMA suppo
```

COMPILATION

In [15]:
```
googlenet_model = Model(inputs = inp, outputs = [out, aux1, aux2])
```

In [16]:
```
googlenet_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.01, epsilon=0.05),
              loss=[keras.losses.sparse_categorical_crossentropy,
                    keras.losses.sparse_categorical_crossentropy,
                    keras.losses.sparse_categorical_crossentropy],
              loss_weights=[1, 0.3, 0.3],
              metrics=['accuracy'])
```

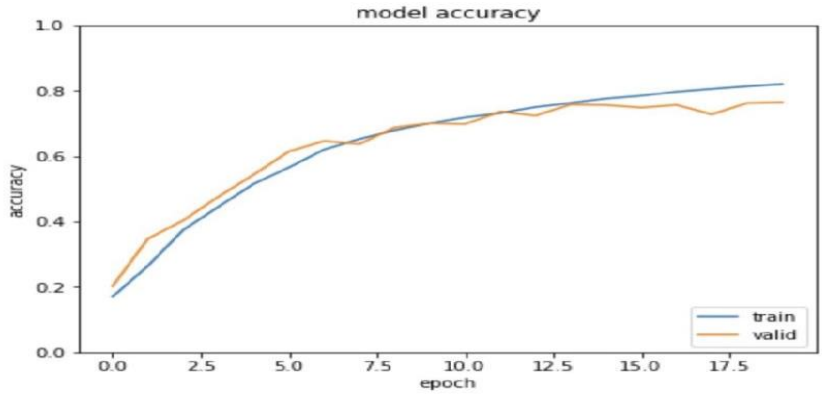
TRAINING

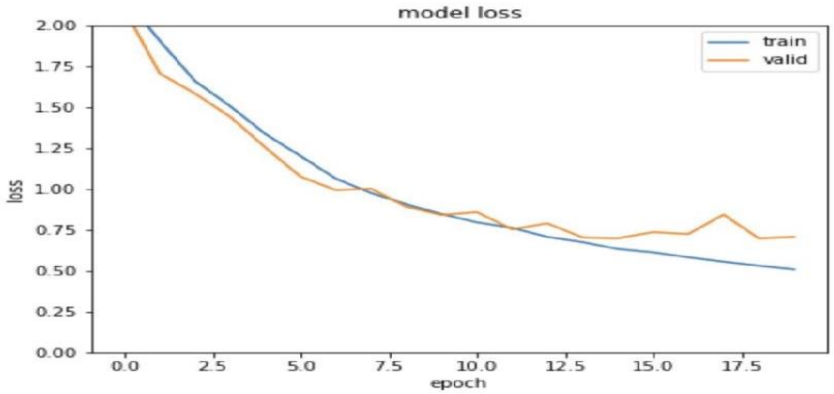
```
In [18]: history = googlenet_model.fit(train_images_normalized, [train_labels, train_labels, train_labels], \
                     validation_data=(test_images_normalized, [test_labels, test_labels, test_labels]), \
                     batch_size=64, epochs=20)
```

```
782/782 [==============================] - 1585s 2s/step - loss: 0.9642 - dense_4_loss: 0.5818 - dense_1_loss: 0.65
83 - dense_3_loss: 0.6166 - dense_4_accuracy: 0.7971 - dense_1_accuracy: 0.7675 - dense_3_accuracy: 0.7840 - val_lo
ss: 1.1571 - val_dense_4_loss: 0.7238 - val_dense_1_loss: 0.7220 - val_dense_3_loss: 0.7225 - val_dense_4_accuracy:
0.7569 - val_dense_1_accuracy: 0.7496 - val_dense_3_accuracy: 0.7564
Epoch 18/20
782/782 [==============================] - 957s 1s/step - loss: 0.9217 - dense_4_loss: 0.5546 - dense_1_loss: 0.632
4 - dense_3_loss: 0.5915 - dense_4_accuracy: 0.8060 - dense_1_accuracy: 0.7782 - dense_3_accuracy: 0.7909 - val_los
s: 1.3123 - val_dense_4_loss: 0.8436 - val_dense_1_loss: 0.7811 - val_dense_3_loss: 0.7814 - val_dense_4_accuracy:
0.7280 - val_dense_1_accuracy: 0.7348 - val_dense_3_accuracy: 0.7318
Epoch 19/20
782/782 [==============================] - 960s 1s/step - loss: 0.8858 - dense_4_loss: 0.5302 - dense_1_loss: 0.614
2 - dense_3_loss: 0.5714 - dense_4_accuracy: 0.8140 - dense_1_accuracy: 0.7841 - dense_3_accuracy: 0.8000 - val_los
s: 1.1291 - val_dense_4_loss: 0.6975 - val_dense_1_loss: 0.7393 - val_dense_3_loss: 0.6996 - val_dense_4_accuracy:
0.7623 - val_dense_1_accuracy: 0.7431 - val_dense_3_accuracy: 0.7600
Epoch 20/20

782/782 [==============================] - 930s 1s/step - loss: 0.8481 - dense_4_loss: 0.5082 - dense_1_loss: 0.586
0 - dense_3_loss: 0.5471 - dense_4_accuracy: 0.8207 - dense_1_accuracy: 0.7938 - dense_3_accuracy: 0.8087 - val_los
s: 1.1265 - val_dense_4_loss: 0.7073 - val_dense_1_loss: 0.6846 - val_dense_3_loss: 0.7131 - val_dense_4_accuracy:
0.7648 - val_dense_1_accuracy: 0.7649 - val_dense_3_accuracy: 0.7601
```

TRAINING AND LOSS CURVES

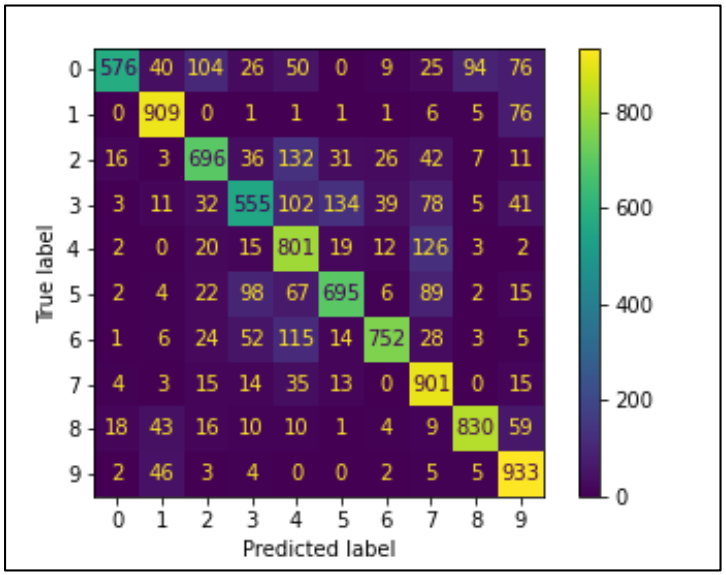
In [26]:
```
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.plot(history.history['dense_4_accuracy'])
plt.plot(history.history['val_dense_4_accuracy'])
plt.ylim(0, 1)
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='lower right')
plt.subplot(1,2,2)
plt.plot(history.history['dense_4_loss'])
plt.plot(history.history['val_dense_4_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='upper right')
plt.ylim([0,2])
plt.savefig("googlenet_loss_acc.png")
```





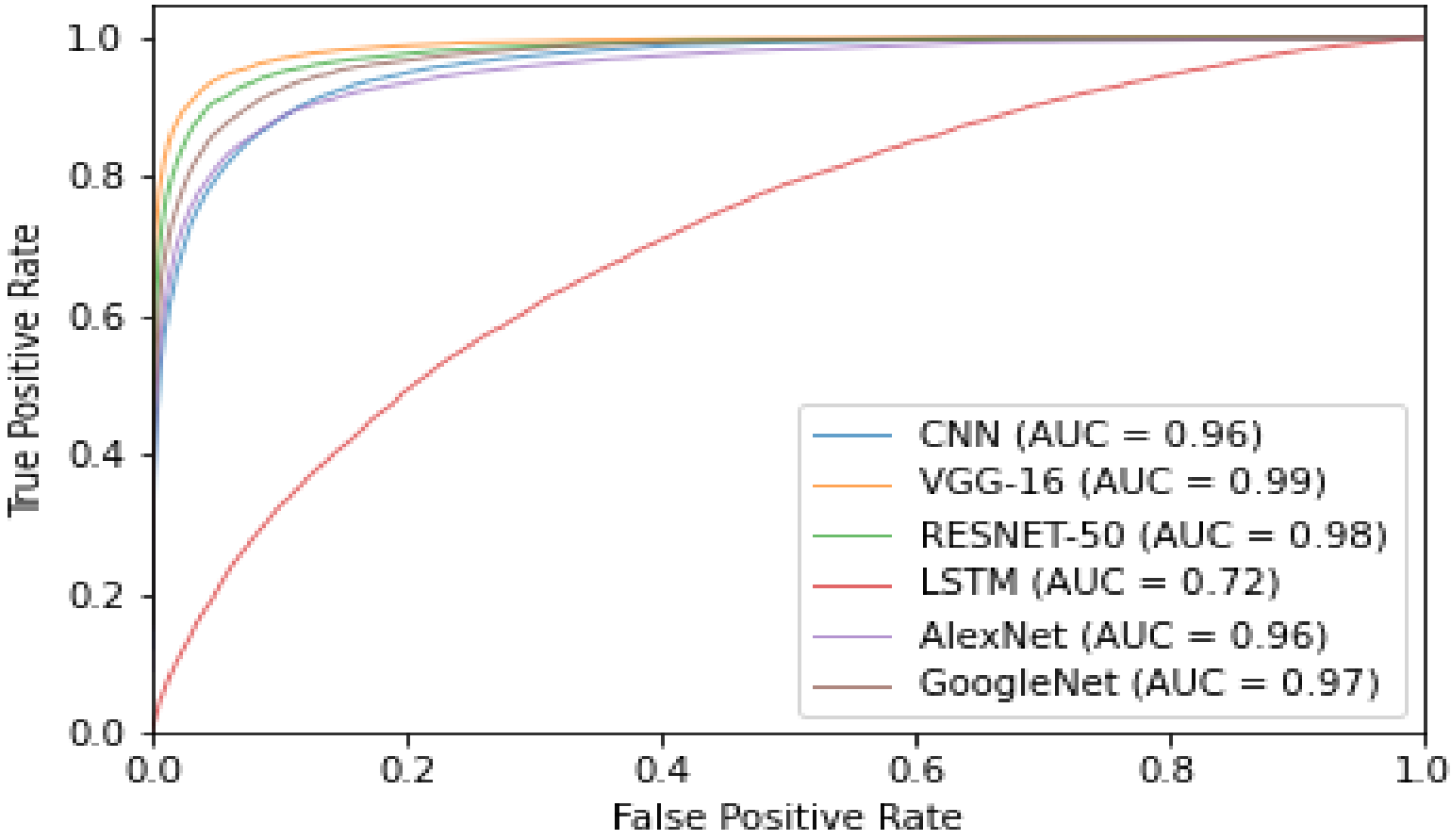
PERFORMANCE

```
313/313 [==============================] - 32s 104ms/step
Confusion Matrix:
[[576  40 104  26  50   0   9  25  94  76]
 [  0 909   0   1   1   1   1   6   5  76]
 [ 16   3 696  36 132  31  26  42   7  11]
 [  3  11  32 555 102 134  39  78   5  41]
 [  2   0  20  15 801  19  12 126   3   2]
 [  2   4  22  98  67 695   6  89   2  15]
 [  1   6  24  52 115  14 752  28   3   5]
 [  4   3  15  14  35  13   0 901   0  15]
 [ 18  43  16  10  10   1   4   9 830  59]
 [  2  46   3   4   0   0   2   5   5 933]]
--------------------------------------------------------------
--------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           0       0.92      0.58      0.71      1000
           1       0.85      0.91      0.88      1000
           2       0.75      0.70      0.72      1000
           3       0.68      0.56      0.61      1000
           4       0.61      0.80      0.69      1000
           5       0.77      0.69      0.73      1000
           6       0.88      0.75      0.81      1000
           7       0.69      0.90      0.78      1000
           8       0.87      0.83      0.85      1000
           9       0.76      0.93      0.84      1000

    accuracy                           0.76     10000
   macro avg       0.78      0.76      0.76     10000
weighted avg       0.78      0.76      0.76     10000

--------------------------------------------------------------
--------------------------------------------------------------
Accuracy Score:
0.7648
```



COMPARISON BETWEEN CNN, VGG-16, RESNET-50, LSTM(RNN), ALEXNET AND GOOGLENET.

ROC CURVES AND AUC COMPARISON



ROC CURVES AND AUC COMPARISON - CODE (I)

In [51]:

```
def plot_macro_avg_roc_curve(classifier, n_classes, name, X_test, y_test, single_output=True):

    y_test_binarized = label_binarize(y_test, classes=list(range(0,n_classes)))
    y_score = classifier.predict(X_test)

    #for cases where the model (like googlenet) has extra (auxillary) outputs
    if not single_output:
        y_score = y_score[0]

    # Compute ROC curve and ROC area for each class
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], y_score[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # First aggregate all false positive rates
    all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))

    # Then interpolate all ROC curves at this points
    mean_tpr = np.zeros_like(all_fpr)
    for i in range(n_classes):
        mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])

    # Finally average it and compute AUC
    mean_tpr /= n_classes

    fpr["macro"] = all_fpr
    tpr["macro"] = mean_tpr
    roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

    plt.plot(
        fpr["macro"],
        tpr["macro"],
        label="{0} (AUC = {1:0.2f})".format(name, roc_auc["macro"]),
        alpha=0.7,
        linewidth=1,
    )

    return plt
```

ROC CURVES AND AUC COMPARISON - CODE (II)

In [43]:
```
cnn = load_model('cnn.h5')
vgg_based_model = load_model('vgg16.h5')
resnet_based_model = load_model('resnet50.h5')
lstm_based_model = load_model('lstm.h5')
alexnet_model = load_model('alexnet.h5')
googlenet_model = load_model('googlenet.h5')
```

In [58]:
```
plot_macro_avg_roc_curve(cnn, 10, "CNN", test_images, test_labels)
plot_macro_avg_roc_curve(vgg_based_model, 10, "VGG-16", test_images_p, test_labels)
plot_macro_avg_roc_curve(resnet_based_model, 10, "RESNET-50", test_images_pr, test_labels)
plot_macro_avg_roc_curve(lstm_based_model, 10, "LSTM", test_images_res, test_labels)
plot_macro_avg_roc_curve(alexnet_model, 10, "AlexNet", test_images, test_labels)
plot_macro_avg_roc_curve(googlenet_model, 10, "GoogleNet", test_images_normalized, test_labels, single_output=False)

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")

plt.savefig('roc_auc.png')
```

```
313/313 [==============================] - 2s 6ms/step
313/313 [==============================] - 5s 17ms/step
313/313 [==============================] - 8s 25ms/step
313/313 [==============================] - 92s 294ms/step
313/313 [==============================] - 22s 72ms/step
2022-09-22 16:22:55.443709: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
313/313 [==============================] - 35s 108ms/step
```

COMPARISON OF PERFORMANCE

CLASSIFIER	Accuracy	Precision	Recall	F1-Score	AUC
CNN	0.72	0.73	0.72	0.72	0.96
VGG-16	0.86	0.87	0.86	0.86	0.99
RESNET-50	0.83	0.83	0.83	0.83	0.98
LSTM	0.27	0.27	0.27	0.25	0.72
ALEXNET	0.73	0.75	0.73	0.73	0.96
GOOGLENET	0.76	0.78	0.76	0.76	0.97

CONCLUSION:

GoogleNet performed better than AlexNet which, in turn, performed better than a simpler CNN. This is expected. However, VGG-16 and RESNET-50 performed much better than all the other models. This is because they were directly imported from the Keras library, retaining the weights that they learned whilst being trained on the 'ImageNet'. This probably enhanced their feature detection ability.

Also, LSTM performed poorly but this was expected. RNNs learn patterns within sequences and the CIFAR-10 dataset simply does not allow this. For example, two pictures of the same dog from different angles would have scarcely any common sequences of pixel intensities.

MNIST - CLASSIFICATION

CNN

IMPORTS AND DATA PREPROCESSING

Convolutional Neural Network

Importing the libraries

In [1]:
```
import numpy as np
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers, datasets
from tensorflow.keras.models import Sequential
import matplotlib.pyplot as plt
```

Part 1 - Data Preprocessing

Normalization and augmentation will be done in the CNN itself

In [2]:
```
img_height = 28
img_width = 28
num_classes = 10
```

In [34]:
```
(train_images, train_labels), (test_images, test_labels) = datasets.mnist.load_data()
```

MODEL ARCHITECTURE

Part 2 - Building the CNN

Building the CNN ¶

In [5]:
```
cnn = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 1)),
  layers.Conv2D(filters=32, kernel_size=3, padding='same', activation='relu'),
  layers.Conv2D(filters=32, kernel_size=3, padding='same', activation='relu'),
  layers.MaxPooling2D(pool_size=2, strides=2),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(pool_size=2, strides=2),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes, activation="softmax")
])
```

COMPILATION AND SUMMARY

Compiling the CNN

In [6]:
```
cnn.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0005, epsilon=0.01),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
```

In [7]:
```
cnn.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 28, 28, 1)         0

 conv2d_3 (Conv2D)           (None, 28, 28, 32)        320

 conv2d_4 (Conv2D)           (None, 28, 28, 32)        9248

 max_pooling2d_2 (MaxPooling  (None, 14, 14, 32)       0
 2D)

 conv2d_5 (Conv2D)           (None, 14, 14, 32)        9248

 max_pooling2d_3 (MaxPooling  (None, 7, 7, 32)         0
 2D)

 dropout_1 (Dropout)         (None, 7, 7, 32)          0

 flatten_1 (Flatten)         (None, 1568)              0

 dense_2 (Dense)             (None, 128)               200832

 dense_3 (Dense)             (None, 10)                1290

=================================================================
Total params: 220,938
Trainable params: 220,938
Non-trainable params: 0
_________________________________________________________________
```

TRAINING AND SAVING

Training the CNN on the Training set and evaluating it on the Test set

In [8]:
```
history = cnn.fit(train_images, train_labels, epochs=20, validation_data=(test_images, test_labels))
```

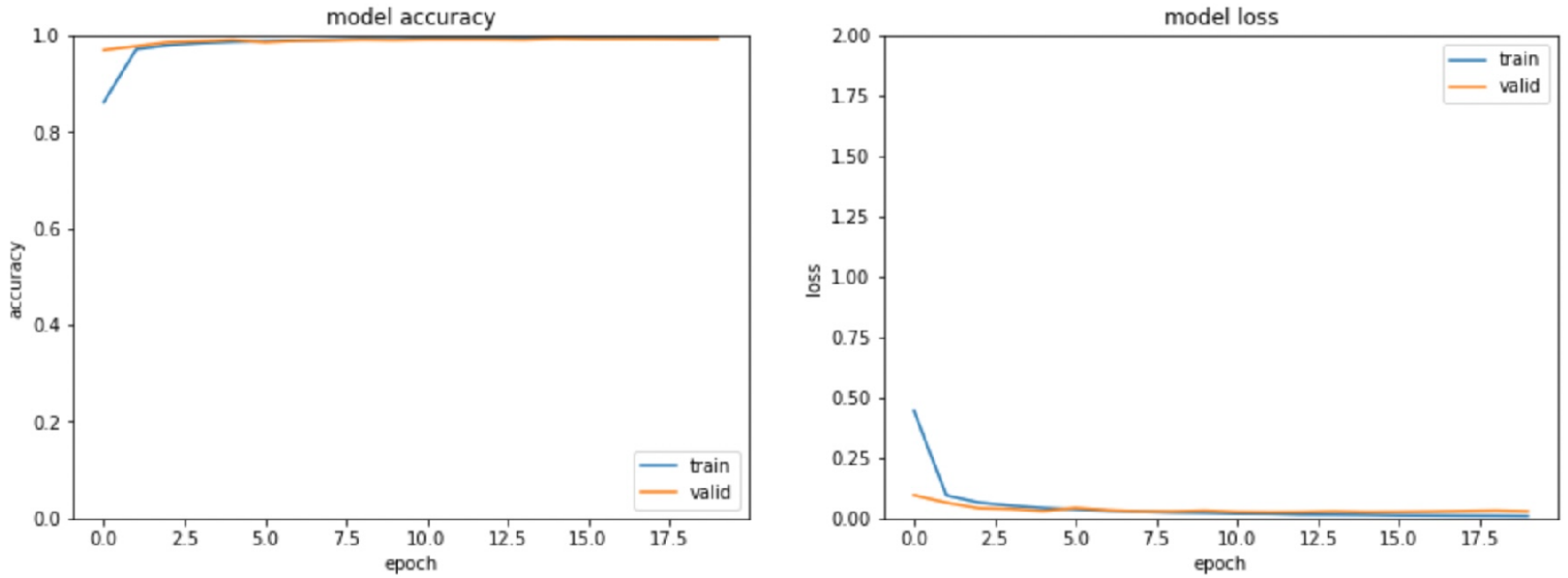
```
al_accuracy: 0.9908
Epoch 15/20
1875/1875 [==============================] - 20s 11ms/step - loss: 0.0138 - accuracy: 0.9956 - val_loss: 0.0252 - v
al_accuracy: 0.9927
Epoch 16/20
1875/1875 [==============================] - 20s 11ms/step - loss: 0.0118 - accuracy: 0.9965 - val_loss: 0.0258 - v
al_accuracy: 0.9918
Epoch 17/20
1875/1875 [==============================] - 20s 11ms/step - loss: 0.0111 - accuracy: 0.9965 - val_loss: 0.0273 - v
al_accuracy: 0.9919
Epoch 18/20
1875/1875 [==============================] - 20s 11ms/step - loss: 0.0104 - accuracy: 0.9965 - val_loss: 0.0295 - v
al_accuracy: 0.9920
Epoch 19/20
1875/1875 [==============================] - 20s 11ms/step - loss: 0.0100 - accuracy: 0.9965 - val_loss: 0.0314 - v
al_accuracy: 0.9916
Epoch 20/20
1875/1875 [==============================] - 20s 11ms/step - loss: 0.0090 - accuracy: 0.9973 - val_loss: 0.0283 - v
al_accuracy: 0.9916
```

In [9]:
```
cnn.save('cnn.h5')
```

TRAINING AND LOSS CURVES

In [14]:
```
def save_train_loss_curves(history, filename):
    plt.figure(figsize=(15,5))
    plt.subplot(1,2,1)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.ylim(0, 1)
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'valid'], loc='lower right')
    plt.subplot(1,2,2)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'valid'], loc='upper right')
    plt.ylim([0,2])
    plt.savefig(filename)
```

In [15]:
```
save_train_loss_curves(history, 'cnn_loss_acc.png')
```



PERFORMANCE - CODE

Performance

In [11]:
```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, ConfusionMatrixDisplay
```

In [12]:
```
def get_predictions(model, inputs):

    def index_max_arr_element(arr):
        best_index = -1
        best = float('-inf')
        for i, num in enumerate(arr):
            if num > best:
                best = num
                best_index = i
        return best_index

    pred_arr = model.predict(inputs)
    pred_labels = [index_max_arr_element(row) for row in pred_arr]
    return pred_labels
```

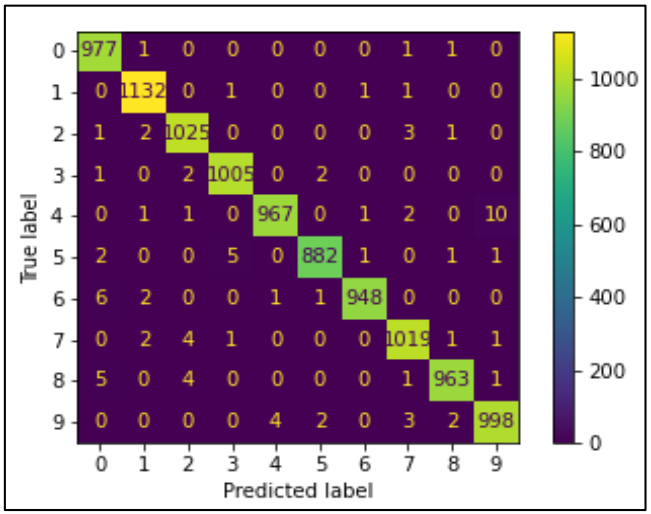
In [13]:
```
pred_labels = get_predictions(cnn, test_images)

print("Confusion Matrix:")
print(confusion_matrix(test_labels, pred_labels))
print("------------------------------------------------------------")
print("------------------------------------------------------------")
print("Performance Evaluation:")
print(classification_report(test_labels, pred_labels))
print("------------------------------------------------------------")
print("------------------------------------------------------------")
print("Accuracy Score:")
print(accuracy_score(test_labels, pred_labels))

plt.figure()
ConfusionMatrixDisplay.from_predictions(test_labels, pred_labels)
plt.savefig('./cnn_matrix.png')
```

```
 37/313 [==>...........................] - ETA: 1s
```

```
2022-09-15 19:38:44.820728: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113] Plugin op
```

PERFORMANCE

```
 37/313 [==>...........................] - ETA: 1s

2022-09-15 19:38:44.820728: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113] Plugin op
timizer for device_type GPU is enabled.

313/313 [==============================] - 1s 4ms/step
Confusion Matrix:
[[ 977    1    0    0    0    0    0    1    1    0]
 [   0 1132    0    1    0    0    1    1    0    0]
 [   1    2 1025    0    0    0    0    3    1    0]
 [   1    0    2 1005    0    2    0    0    0    0]
 [   0    1    1    0  967    0    1    2    0   10]
 [   2    0    0    5    0  882    1    0    1    1]
 [   6    2    0    0    1    1  948    0    0    0]
 [   0    2    4    1    0    0    0 1019    1    1]
 [   5    0    4    0    0    0    0    1  963    1]
 [   0    0    0    0    4    2    0    3    2  998]]
------------------------------------------------------------
------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       980
           1       0.99      1.00      1.00      1135
           2       0.99      0.99      0.99      1032
           3       0.99      1.00      0.99      1010
           4       0.99      0.98      0.99       982
           5       0.99      0.99      0.99       892
           6       1.00      0.99      0.99       958
           7       0.99      0.99      0.99      1028
           8       0.99      0.99      0.99       974
           9       0.99      0.99      0.99      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000

------------------------------------------------------------
------------------------------------------------------------
Accuracy Score:
0.9916
```



VGG-16

IMPORTS, DATA PREPROCESSING & ARCHITECTURE

VGG-16

In [16]:
```
from tensorflow.keras.applications import vgg16
```

Preprocessing data to make it compatible to VGG and RESNET models

In [41]:
```
train_images_stacked = np.dstack([train_images] * 3)
test_images_stacked = np.dstack([test_images]*3)

test_images_stacked = test_images_stacked.reshape(-1,28,28,3)
train_images_stacked = train_images_stacked.reshape(-1,28,28,3)

train_images_resized= tf.image.resize(train_images_stacked, tf.constant([32, 32]))
test_images_resized= tf.image.resize(test_images_stacked, tf.constant([32, 32]))
```

In [42]:
```
train_images_p, test_images_p = vgg16.preprocess_input(train_images_resized), vgg16.preprocess_input(test_images_res
```

In [43]:
```
vgg_based_model = Sequential([
  layers.Rescaling(1./255, input_shape=(32, 32, 3)),
  vgg16.VGG16(include_top=False, weights="imagenet", pooling="max", input_shape=(32, 32, 3)),
  layers.Flatten(),
  layers.Dense(512, activation=('relu')),
  layers.Dropout(0.2),
  layers.Dense(128, activation=('relu')),
  layers.Dropout(0.2),
  layers.Dense(num_classes, activation="softmax")
])
```

COMPILATION AND SUMMARY

In [44]:
```
vgg_based_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0002, epsilon=0.01),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
```

In [45]:
```
vgg_based_model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_13 (Rescaling)    (None, 32, 32, 3)         0

 vgg16 (Functional)          (None, 512)               14714688

 flatten_5 (Flatten)         (None, 512)               0

 dense_17 (Dense)            (None, 512)               262656

 dropout_14 (Dropout)        (None, 512)               0

 dense_18 (Dense)            (None, 128)               65664

 dropout_15 (Dropout)        (None, 128)               0

 dense_19 (Dense)            (None, 10)                1290

=================================================================
Total params: 15,044,298
Trainable params: 15,044,298
Non-trainable params: 0
_________________________________________________________________
```

TRAINING AND SAVING

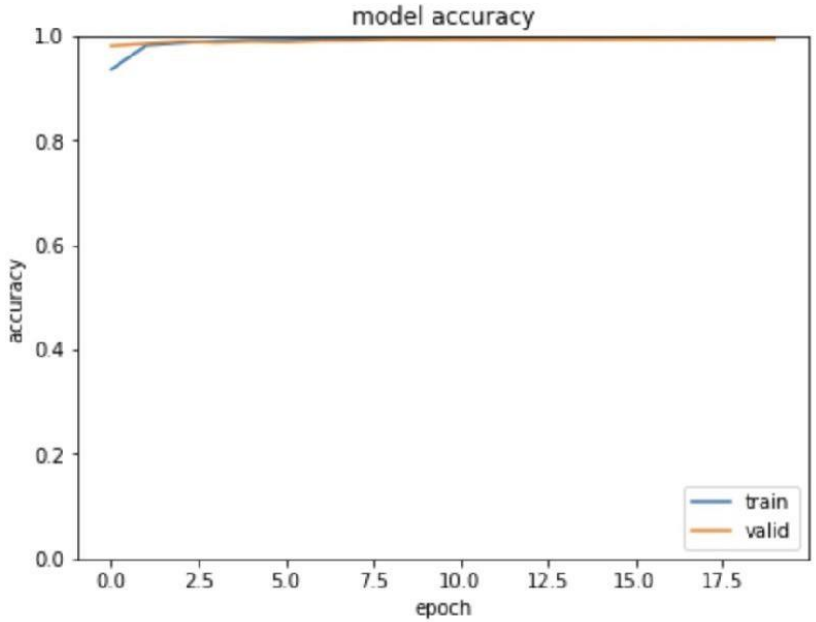
```
In [46]: history = vgg_based_model.fit(train_images_p, train_labels, epochs=20, validation_data=(test_images_p, test_labels))
```

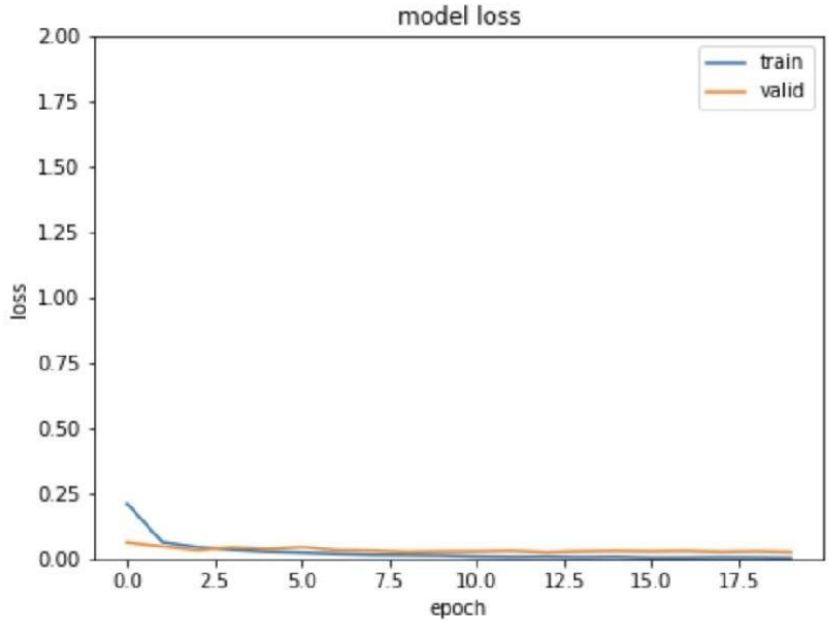
```
val_accuracy: 0.9927
Epoch 15/20
1875/1875 [==============================] - 188s 100ms/step - loss: 0.0077 - accuracy: 0.9976 - val_loss: 0.0290 -
val_accuracy: 0.9928
Epoch 16/20
1875/1875 [==============================] - 180s 96ms/step - loss: 0.0040 - accuracy: 0.9989 - val_loss: 0.0276 -
val_accuracy: 0.9933
Epoch 17/20
1875/1875 [==============================] - 177s 95ms/step - loss: 0.0043 - accuracy: 0.9987 - val_loss: 0.0291 -
val_accuracy: 0.9927
Epoch 18/20
1875/1875 [==============================] - 204s 109ms/step - loss: 0.0057 - accuracy: 0.9985 - val_loss: 0.0254 -
val_accuracy: 0.9934
Epoch 19/20
1875/1875 [==============================] - 247s 132ms/step - loss: 0.0051 - accuracy: 0.9984 - val_loss: 0.0275 -
val_accuracy: 0.9935
Epoch 20/20
1875/1875 [==============================] - 277s 148ms/step - loss: 0.0033 - accuracy: 0.9990 - val_loss: 0.0247 -
val_accuracy: 0.9942
```

```
In [48]: vgg_based_model.save('vgg16.h5')
```

TRAINING AND LOSS CURVES

```
In [47]: save_train_loss_curves(history, 'vgg_loss_acc.png')
```





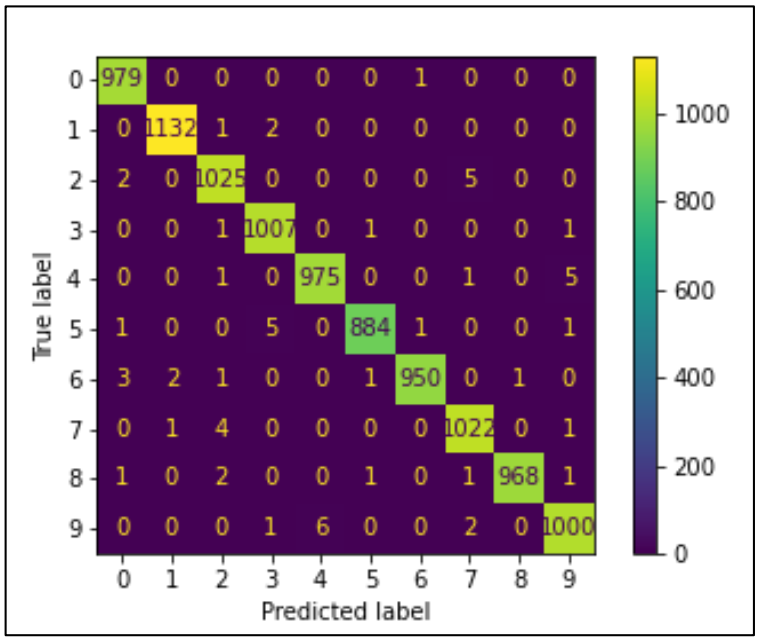
PERFORMANCE - CODE

In [63]:

```
pred_labels = get_predictions(vgg_based_model, test_images_p)

print("Confusion Matrix:")
print(confusion_matrix(test_labels, pred_labels))
print("-----------------------------------------------------------")
print("-----------------------------------------------------------")
print("Performance Evaluation:")
print(classification_report(test_labels, pred_labels))
print("-----------------------------------------------------------")
print("-----------------------------------------------------------")
print("Accuracy Score:")
print(accuracy_score(test_labels, pred_labels))

plt.figure()
ConfusionMatrixDisplay.from_predictions(test_labels, pred_labels)
plt.savefig('./vgg_matrix.png')
```

PERFORMANCE

```
  5/313 [..............................] - ETA: 4s
2022-09-16 11:40:18.080302: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113] Plugin op
timizer for device_type GPU is enabled.

313/313 [==============================] - 5s 14ms/step
Confusion Matrix:
[[ 979    0    0    0    0    0    1    0    0    0]
 [   0 1132    1    2    0    0    0    0    0    0]
 [   2    0 1025    0    0    0    0    5    0    0]
 [   0    0    1 1007    0    1    0    0    0    1]
 [   0    0    1    0  975    0    0    1    0    5]
 [   1    0    0    5    0  884    1    0    0    1]
 [   3    2    1    0    0    1  950    0    1    0]
 [   0    1    4    0    0    0    0 1022    0    1]
 [   1    0    2    0    0    1    0    1  968    1]
 [   0    0    0    1    6    0    0    2    0 1000]]
------------------------------------------------------------
------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00       980
           1       1.00      1.00      1.00      1135
           2       0.99      0.99      0.99      1032
           3       0.99      1.00      0.99      1010
           4       0.99      0.99      0.99       982
           5       1.00      0.99      0.99       892
           6       1.00      0.99      0.99       958
           7       0.99      0.99      0.99      1028
           8       1.00      0.99      1.00       974
           9       0.99      0.99      0.99      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000

------------------------------------------------------------
------------------------------------------------------------
Accuracy Score:
0.9942
```



RESNET-50

IMPORTS, DATA PREPROCESSING & ARCHITECTURE

RESNET-50

In [49]:
```
from tensorflow.keras.applications import resnet50
```

In [50]:
```
train_images_pr, test_images_pr = resnet50.preprocess_input(train_images_resized), resnet50.preprocess_input(test_im
```

In [51]:
```
resnet_based_model = Sequential([
  layers.Rescaling(1./255, input_shape=(32, 32, 3)),
  resnet50.ResNet50(include_top=False, weights="imagenet", pooling="max", input_shape=(32, 32, 3)),
  layers.Flatten(),
  layers.Dense(512, activation=('relu')),
  layers.Dropout(0.2),
  layers.Dense(128, activation=('relu')),
  layers.Dropout(0.2),
  layers.Dense(num_classes, activation="softmax")
])
```

COMPILATION AND SUMMARY

In [52]:
```
resnet_based_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0002, epsilon=0.01),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
```

In [53]:
```
resnet_based_model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_14 (Rescaling)    (None, 32, 32, 3)         0

 resnet50 (Functional)       (None, 2048)              23587712

 flatten_6 (Flatten)         (None, 2048)              0

 dense_20 (Dense)            (None, 512)               1049088

 dropout_16 (Dropout)        (None, 512)               0

 dense_21 (Dense)            (None, 128)               65664

 dropout_17 (Dropout)        (None, 128)               0

 dense_22 (Dense)            (None, 10)                1290

=================================================================
Total params: 24,703,754
Trainable params: 24,650,634
Non-trainable params: 53,120
_________________________________________________________________
```

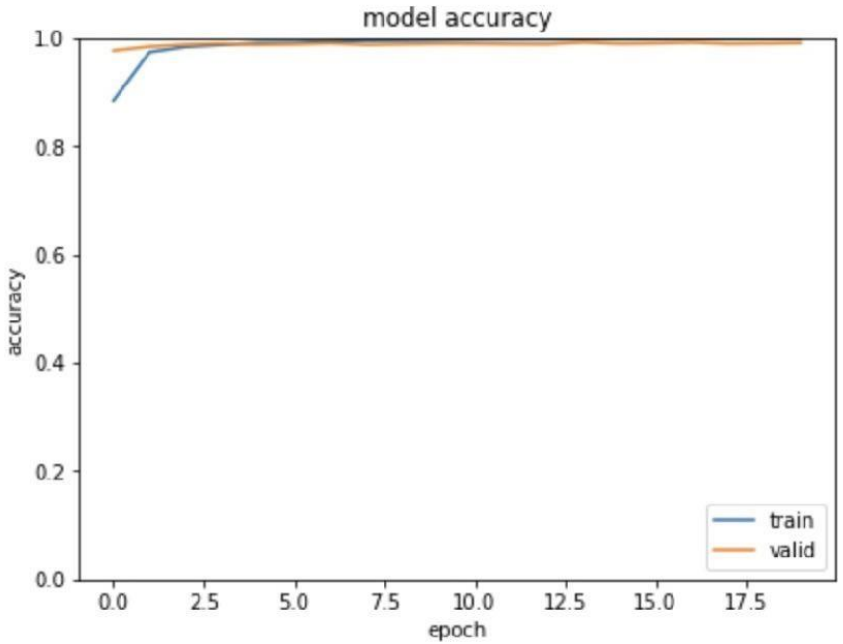
TRAINING AND SAVING

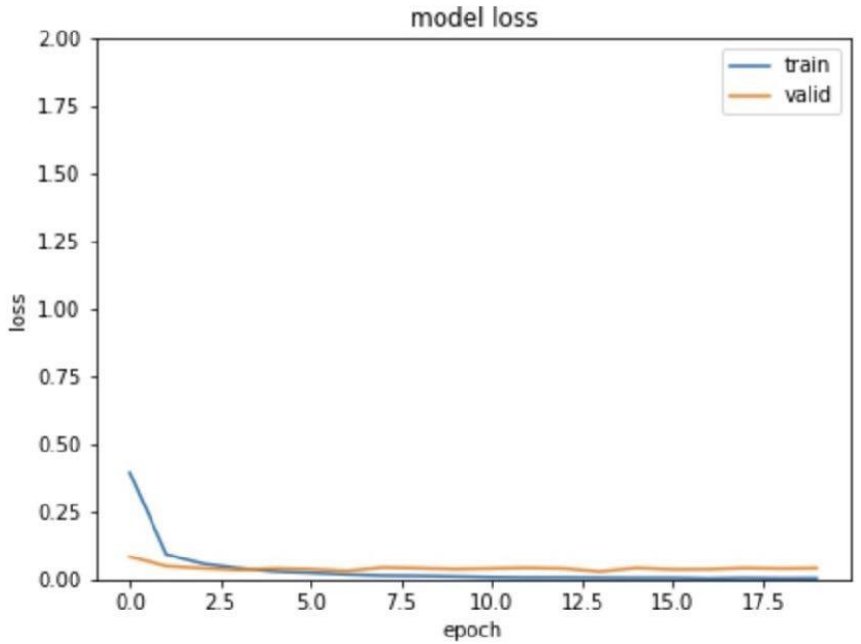
```
In [54]: history = resnet_based_model.fit(train_images_pr, train_labels, epochs=20, validation_data=(test_images_pr, test_lab
- val_accuracy: 0.9928
Epoch 15/20
1875/1875 [==============================] - 196s 105ms/step - loss: 0.0053 - accuracy: 0.9985 - val_loss: 0.0410 -
val_accuracy: 0.9903
Epoch 16/20
1875/1875 [==============================] - 211s 112ms/step - loss: 0.0054 - accuracy: 0.9984 - val_loss: 0.0357 -
val_accuracy: 0.9910
Epoch 17/20
1875/1875 [==============================] - 235s 125ms/step - loss: 0.0030 - accuracy: 0.9991 - val_loss: 0.0363 -
val_accuracy: 0.9926
Epoch 18/20
1875/1875 [==============================] - 256s 136ms/step - loss: 0.0051 - accuracy: 0.9986 - val_loss: 0.0408 -
val_accuracy: 0.9902
Epoch 19/20
1875/1875 [==============================] - 314s 168ms/step - loss: 0.0035 - accuracy: 0.9988 - val_loss: 0.0389 -
val_accuracy: 0.9908
Epoch 20/20
1875/1875 [==============================] - 344s 184ms/step - loss: 0.0040 - accuracy: 0.9988 - val_loss: 0.0405 -
val_accuracy: 0.9915
```

```
In [56]: resnet_based_model.save('resnet50.h5')
```

TRAINING AND LOSS CURVES

```
In [55]: save_train_loss_curves(history, 'resnet_loss_acc.png')
```





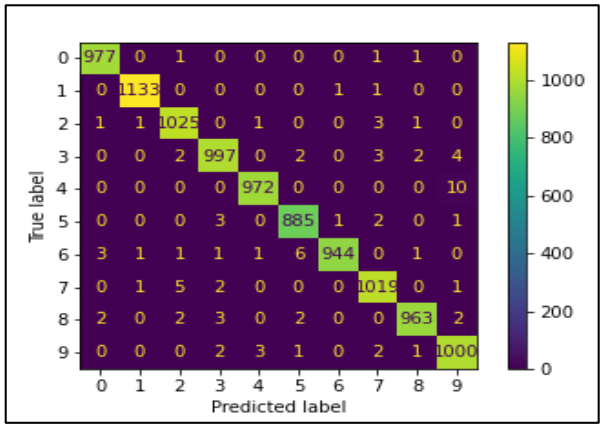
PERFORMANCE - CODE

In [64]:
```
pred_labels = get_predictions(resnet_based_model, test_images_pr)

print("Confusion Matrix:")
print(confusion_matrix(test_labels, pred_labels))
print("-------------------------------------------------------------")
print("-------------------------------------------------------------")
print("Performance Evaluation:")
print(classification_report(test_labels, pred_labels))
print("-------------------------------------------------------------")
print("-------------------------------------------------------------")
print("Accuracy Score:")
print(accuracy_score(test_labels, pred_labels))

plt.figure()
ConfusionMatrixDisplay.from_predictions(test_labels, pred_labels)
plt.savefig('./resnet_matrix.png')
```

```
2022-09-16 11:41:07.726305: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113] Plugin op
timizer for device_type GPU is enabled.
```

PERFORMANCE

```
2022-09-16 11:41:07.726305: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.

313/313 [==============================] - 8s 25ms/step
Confusion Matrix:
[[ 977    0    1    0    0    0    0    1    1    0]
 [   0 1133    0    0    0    0    1    1    0    0]
 [   1    1 1025    0    1    0    0    3    1    0]
 [   0    0    2  997    0    2    0    3    2    4]
 [   0    0    0    0  972    0    0    0    0   10]
 [   0    0    0    3    0  885    1    2    0    1]
 [   3    1    1    1    1    6  944    0    1    0]
 [   0    1    5    2    0    0    0 1019    0    1]
 [   2    0    2    3    0    2    0    0  963    2]
 [   0    0    0    2    3    1    0    2    1 1000]]
------------------------------------------------------------
------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00       980
           1       1.00      1.00      1.00      1135
           2       0.99      0.99      0.99      1032
           3       0.99      0.99      0.99      1010
           4       0.99      0.99      0.99       982
           5       0.99      0.99      0.99       892
           6       1.00      0.99      0.99       958
           7       0.99      0.99      0.99      1028
           8       0.99      0.99      0.99       974
           9       0.98      0.99      0.99      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000

------------------------------------------------------------
------------------------------------------------------------
Accuracy Score:
0.9915
```



RNN (LSTM)

ARCHITECTURE AND COMPILATION

In [57]:
```
lstm_based_model = Sequential([
  layers.Rescaling(1./255, input_shape=(28, 28)),
  layers.LSTM(units=50, return_sequences=True),
  layers.Dropout(0.2),
  layers.LSTM(units=50),
  layers.Dropout(0.2),
  layers.Dense(128, activation=('relu')),
  layers.Dropout(0.2),
  layers.Dense(num_classes, activation="softmax")
])
```

In [58]:
```
lstm_based_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
```

In [60]:
```
lstm_based_model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_15 (Rescaling)    (None, 28, 28)            0

 lstm_4 (LSTM)               (None, 28, 50)            15800

 dropout_18 (Dropout)        (None, 28, 50)            0

 lstm_5 (LSTM)               (None, 50)                20200

 dropout_19 (Dropout)        (None, 50)                0

 dense_23 (Dense)            (None, 128)               6528

 dropout_20 (Dropout)        (None, 128)               0

 dense_24 (Dense)            (None, 10)                1290

=================================================================
Total params: 43,818
Trainable params: 43,818
```

TRAINING AND SAVING

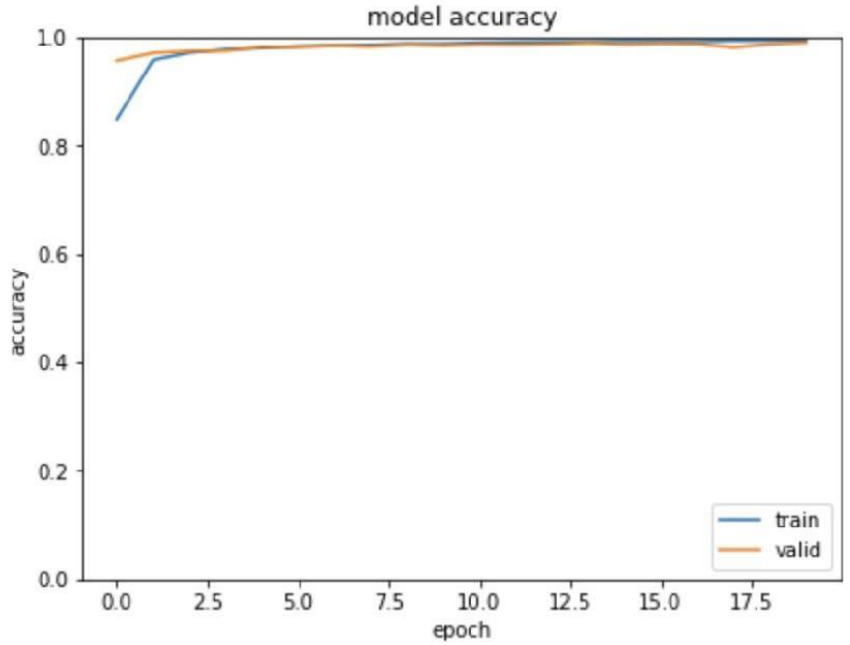
```
In [59]: history = lstm_based_model.fit(train_images, train_labels, epochs=20, validation_data=(test_images, test_labels))
```

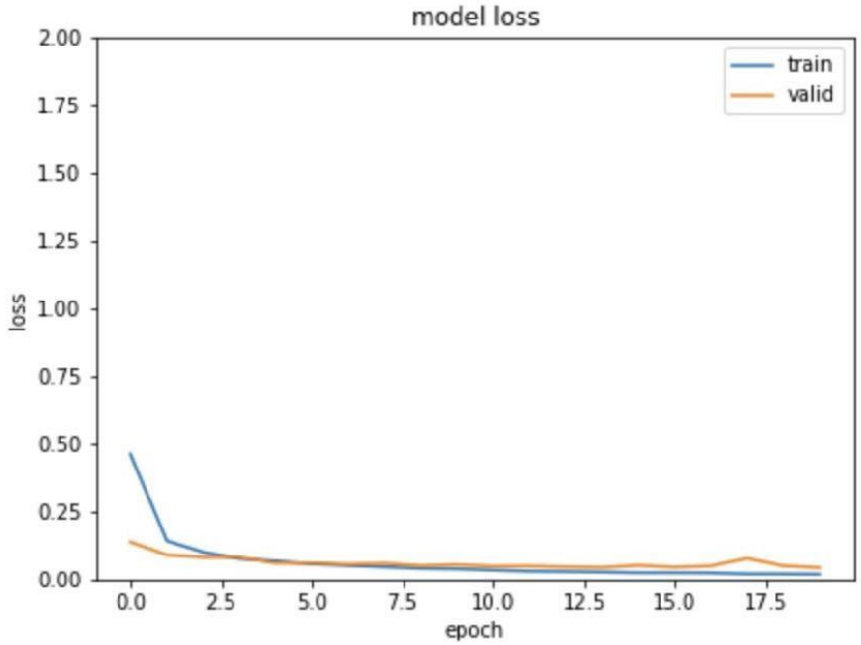
```
al_accuracy: 0.9895
Epoch 15/20
1875/1875 [==============================] - 46s 24ms/step - loss: 0.0236 - accuracy: 0.9929 - val_loss: 0.0506 - v
al_accuracy: 0.9870
Epoch 16/20
1875/1875 [==============================] - 45s 24ms/step - loss: 0.0236 - accuracy: 0.9929 - val_loss: 0.0444 - v
al_accuracy: 0.9884
Epoch 17/20
1875/1875 [==============================] - 45s 24ms/step - loss: 0.0229 - accuracy: 0.9927 - val_loss: 0.0485 - v
al_accuracy: 0.9879
Epoch 18/20
1875/1875 [==============================] - 46s 24ms/step - loss: 0.0192 - accuracy: 0.9941 - val_loss: 0.0766 - v
al_accuracy: 0.9816
Epoch 19/20
1875/1875 [==============================] - 45s 24ms/step - loss: 0.0190 - accuracy: 0.9940 - val_loss: 0.0491 - v
al_accuracy: 0.9875
Epoch 20/20
1875/1875 [==============================] - 45s 24ms/step - loss: 0.0180 - accuracy: 0.9945 - val_loss: 0.0426 - v
al_accuracy: 0.9905
```

```
In [76]: lstm_based_model.save('lstm.h5')
```

TRAINING AND LOSS CURVES

```
In [75]: save_train_loss_curves(history, 'lstm_loss_acc.png')
```



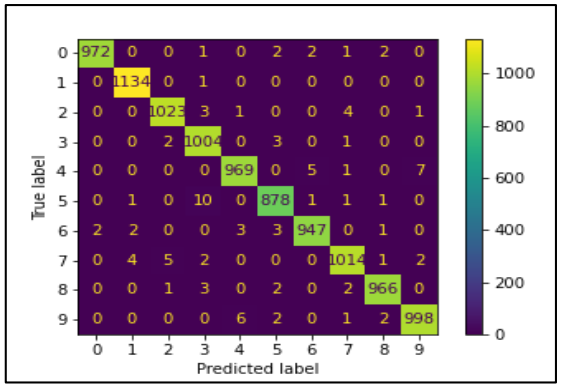


PERFORMANCE - CODE

In [65]:
```
pred_labels = get_predictions(lstm_based_model, test_images)

print("Confusion Matrix:")
print(confusion_matrix(test_labels, pred_labels))
print("------------------------------------------------------------")
print("------------------------------------------------------------")
print("Performance Evaluation:")
print(classification_report(test_labels, pred_labels))
print("------------------------------------------------------------")
print("------------------------------------------------------------")
print("Accuracy Score:")
print(accuracy_score(test_labels, pred_labels))

plt.figure()
ConfusionMatrixDisplay.from_predictions(test_labels, pred_labels)
plt.savefig('./lstm_matrix.png')
```

PERFORMANCE

```
2022-09-16 11:41:58.096347: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113] Plugin op
timizer for device_type GPU is enabled.
2022-09-16 11:41:58.157487: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113] Plugin op
timizer for device_type GPU is enabled.

313/313 [==============================] - 3s 9ms/step
Confusion Matrix:
[[ 972    0    0    1    0    2    2    1    2    0]
 [   0 1134    0    1    0    0    0    0    0    0]
 [   0    0 1023    3    1    0    0    4    0    1]
 [   0    0    2 1004    0    3    0    1    0    0]
 [   0    0    0    0  969    0    5    1    0    7]
 [   0    1    0   10    0  878    1    1    1    0]
 [   2    2    0    0    3    3  947    0    1    0]
 [   0    4    5    2    0    0    0 1014    1    2]
 [   0    0    1    3    0    2    0    2  966    0]
 [   0    0    0    0    6    2    0    1    2  998]]
--------------------------------------------------------------
--------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       980
           1       0.99      1.00      1.00      1135
           2       0.99      0.99      0.99      1032
           3       0.98      0.99      0.99      1010
           4       0.99      0.99      0.99       982
           5       0.99      0.98      0.99       892
           6       0.99      0.99      0.99       958
           7       0.99      0.99      0.99      1028
           8       0.99      0.99      0.99       974
           9       0.99      0.99      0.99      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000

--------------------------------------------------------------
--------------------------------------------------------------
Accuracy Score:
0.9905
```



ALEXNET

IMPORTS & DATA PREPROCESSING

ALEXNET

In [66]:
```
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D, ZeroPadding2D, BatchN
from tensorflow.keras.regularizers import l2
```

In [68]:
```
train_images_stacked = np.dstack([train_images] * 3)
test_images_stacked = np.dstack([test_images]*3)

test_images_stacked = test_images_stacked.reshape(-1,28,28,3)
train_images_stacked = train_images_stacked.reshape(-1,28,28,3)
```

MODEL ARCHITECTURE - I

In [69]:
```
def alexnet_model(img_shape=(28, 28, 3), n_classes=10, l2_reg=0.,
    weights=None):

    # Initialize model
    alexnet = Sequential()

    # Layer 1
    alexnet.add(Conv2D(96, (11, 11), input_shape=img_shape, padding='same', kernel_regularizer=l2(l2_reg)))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(MaxPooling2D(pool_size=(2, 2)))

    # Layer 2
    alexnet.add(Conv2D(256, (5, 5), padding='same'))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(MaxPooling2D(pool_size=(2, 2)))

    # Layer 3
    alexnet.add(ZeroPadding2D((1, 1)))
    alexnet.add(Conv2D(512, (3, 3), padding='same'))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(MaxPooling2D(pool_size=(2, 2)))

    # Layer 4
    alexnet.add(ZeroPadding2D((1, 1)))
    alexnet.add(Conv2D(1024, (3, 3), padding='same'))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))

    # Layer 5
    alexnet.add(ZeroPadding2D((1, 1)))
    alexnet.add(Conv2D(1024, (3, 3), padding='same'))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(MaxPooling2D(pool_size=(2, 2)))

    # Layer 6
    alexnet.add(Flatten())
    alexnet.add(Dense(3072))
```

MODEL ARCHITECTURE - II

```
    # Layer 6
    alexnet.add(Flatten())
    alexnet.add(Dense(3072))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(Dropout(0.5))

    # Layer 7
    alexnet.add(Dense(4096))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(Dropout(0.5))

    # Layer 8
    alexnet.add(Dense(n_classes))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('softmax'))

    if weights is not None:
        alexnet.load_weights(weights)

    return alexnet
```

In [70]:
```
alexnet_model = alexnet_model(img_shape=(28, 28, 3), n_classes=10)
```

In [71]:
```
alexnet_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0002, epsilon=0.05),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
```

TRAINING AND SAVING

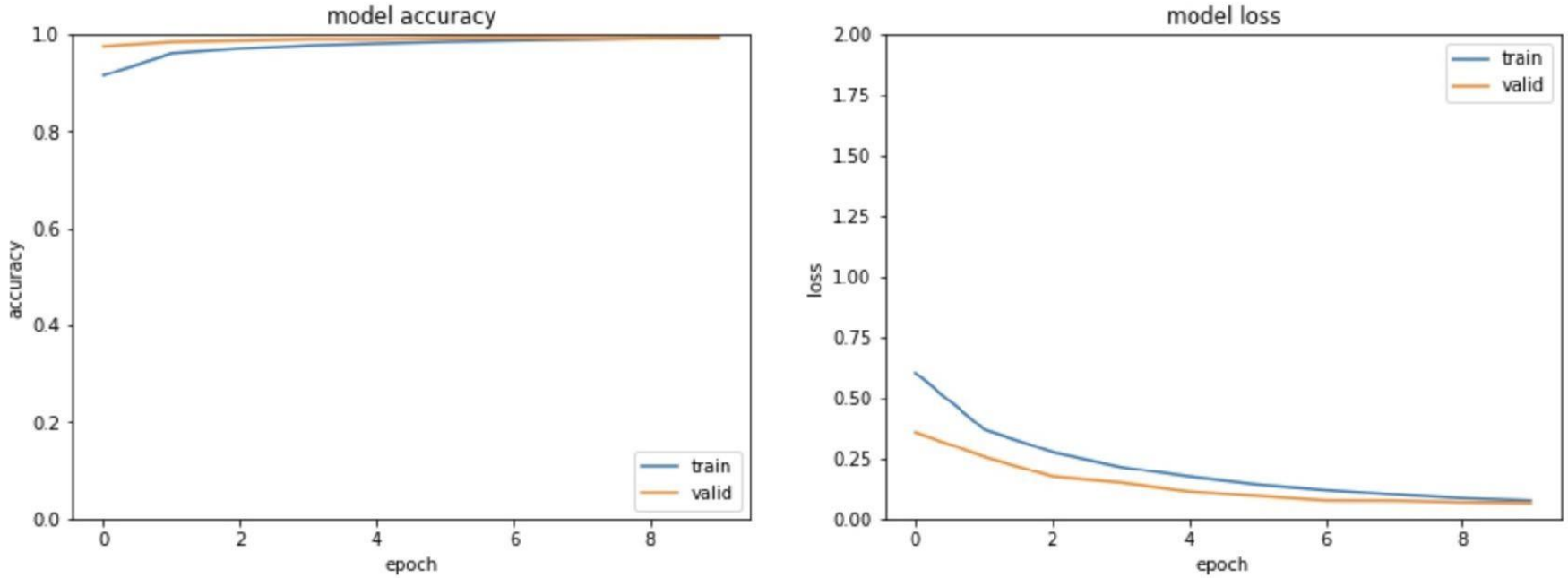
```
In [77]: history = alexnet_model.fit(train_images_stacked, train_labels, epochs=10, validation_data=(test_images_stacked, tes
```

```
Epoch 1/10
1875/1875 [==============================] - ETA: 0s - loss: 0.6029 - accuracy: 0.9153
2022-09-16 18:14:56.536461: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113] Plugin op
timizer for device_type GPU is enabled.
1875/1875 [==============================] - 771s 411ms/step - loss: 0.6029 - accuracy: 0.9153 - val_loss: 0.3567 -
val_accuracy: 0.9757
Epoch 2/10
1875/1875 [==============================] - 747s 398ms/step - loss: 0.3695 - accuracy: 0.9604 - val_loss: 0.2572 -
val_accuracy: 0.9848
Epoch 3/10
1875/1875 [==============================] - 1021s 545ms/step - loss: 0.2755 - accuracy: 0.9704 - val_loss: 0.1749
- val_accuracy: 0.9873
Epoch 4/10
1875/1875 [==============================] - 1054s 562ms/step - loss: 0.2144 - accuracy: 0.9772 - val_loss: 0.1506
- val_accuracy: 0.9902
Epoch 5/10
1875/1875 [==============================] - 1223s 652ms/step - loss: 0.1748 - accuracy: 0.9814 - val_loss: 0.1136
- val_accuracy: 0.9902
Epoch 6/10
1875/1875 [==============================] - 948s 505ms/step - loss: 0.1422 - accuracy: 0.9850 - val_loss: 0.0946 -
val_accuracy: 0.9920
Epoch 7/10
1875/1875 [==============================] - 923s 492ms/step - loss: 0.1194 - accuracy: 0.9876 - val_loss: 0.0755 -
val_accuracy: 0.9923
Epoch 8/10
1875/1875 [==============================] - 942s 503ms/step - loss: 0.1012 - accuracy: 0.9897 - val_loss: 0.0747 -
val_accuracy: 0.9920
Epoch 9/10
1875/1875 [==============================] - 618s 330ms/step - loss: 0.0848 - accuracy: 0.9918 - val_loss: 0.0679 -
val_accuracy: 0.9917
Epoch 10/10
1875/1875 [==============================] - 548s 292ms/step - loss: 0.0747 - accuracy: 0.9925 - val_loss: 0.0642 -
val_accuracy: 0.9920
```

```
In [78]: alexnet_model.save('alexnet.h5')
```

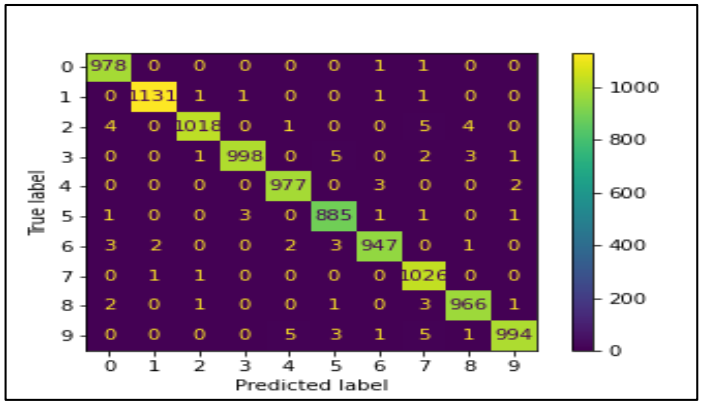
TRAINING AND LOSS CURVES

```
In [79]: save_train_loss_curves(history, 'alexnet_loss_acc.png')
```



PERFORMANCE

```
  1/313 [..............................] - ETA: 1:06
2022-09-16 20:44:23.437447: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
313/313 [==============================] - 17s 54ms/step
Confusion Matrix:
[[ 978    0    0    0    0    0    1    1    0    0]
 [   0 1131    1    1    0    0    1    1    0    0]
 [   4    0 1018    0    1    0    0    5    4    0]
 [   0    0    1  998    0    5    0    2    3    1]
 [   0    0    0    0  977    0    3    0    0    2]
 [   1    0    0    3    0  885    1    1    0    1]
 [   3    2    0    0    2    3  947    0    1    0]
 [   0    1    1    0    0    0    0 1026    0    0]
 [   2    0    1    0    0    1    0    3  966    1]
 [   0    0    0    0    5    3    1    5    1  994]]
--------------------------------------------------------------
--------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       980
           1       1.00      1.00      1.00      1135
           2       1.00      0.99      0.99      1032
           3       1.00      0.99      0.99      1010
           4       0.99      0.99      0.99       982
           5       0.99      0.99      0.99       892
           6       0.99      0.99      0.99       958
           7       0.98      1.00      0.99      1028
           8       0.99      0.99      0.99       974
           9       0.99      0.99      0.99      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000

--------------------------------------------------------------
--------------------------------------------------------------
Accuracy Score:
0.992
```



GOOGLENET

IMPORTS, DATA PREPROCESSING AND INCEPTION MODULE

GOOGLENET

In [7]:
```
from tensorflow.keras.layers import Flatten, Input, Conv2D, Dense, Dropout, MaxPooling2D, AveragePooling2D, GlobalAv
from tensorflow.keras.layers import concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.regularizers import l2
```

In [8]:
```
train_images_normalized = train_images/255
test_images_normalized = test_images/255
```

In [9]:
```
train_images_stacked = np.dstack([train_images] * 3)
test_images_stacked = np.dstack([test_images]*3)

test_images_stacked = test_images_stacked.reshape(-1,28,28,3)
train_images_stacked = train_images_stacked.reshape(-1,28,28,3)
```

In [10]:
```
def inception(x,
              filters_1x1,
              filters_3x3_reduce,
              filters_3x3,
              filters_5x5_reduce,
              filters_5x5,
              filters_pool):
    path1 = layers.Conv2D(filters_1x1, (1, 1), padding='same',    activation='relu')(x)
    path2 = layers.Conv2D(filters_3x3_reduce, (1, 1), padding='same', activation='relu')(x)
    path2 = layers.Conv2D(filters_3x3, (1, 1), padding='same', activation='relu')(path2)
    path3 = layers.Conv2D(filters_5x5_reduce, (1, 1), padding='same', activation='relu')(x)
    path3 = layers.Conv2D(filters_5x5, (1, 1), padding='same', activation='relu')(path3)
    path4 = layers.MaxPool2D((3, 3), strides=(1, 1), padding='same')(x)
    path4 = layers.Conv2D(filters_pool, (1, 1), padding='same', activation='relu')(path4)
    return tf.concat([path1, path2, path3, path4], axis=3)
```

MODEL ARCHITECTURE

In [11]:
```
inp = layers.Input(shape=(28, 28, 3))
input_tensor = layers.experimental.preprocessing.Resizing(224, 224, interpolation="bilinear", input_shape=train_imag
x = layers.Conv2D(64, 7, strides=2, padding='same', activation='relu')(input_tensor)
x = layers.MaxPooling2D(3, strides=2)(x)
x = layers.Conv2D(64, 1, strides=1, padding='same', activation='relu')(x)
x = layers.Conv2D(192, 3, strides=1, padding='same', activation='relu')(x)
x = layers.MaxPooling2D(3, strides=2)(x)
x = inception(x, filters_1x1=64, filters_3x3_reduce=96, filters_3x3=128, filters_5x5_reduce=16, filters_5x5=32, filt
x = inception(x, filters_1x1=128, filters_3x3_reduce=128, filters_3x3=192, filters_5x5_reduce=32, filters_5x5=96, fi
x = layers.MaxPooling2D(3, strides=2)(x)
x = inception(x, filters_1x1=192, filters_3x3_reduce=96, filters_3x3=208, filters_5x5_reduce=16, filters_5x5=48, fil
aux1 = layers.AveragePooling2D((5, 5), strides=3)(x)
aux1 =layers.Conv2D(128, 1, padding='same', activation='relu')(aux1)
aux1 = layers.Flatten()(aux1)
aux1 = layers.Dense(1024, activation='relu')(aux1)
aux1 = layers.Dropout(0.7)(aux1)
aux1 = layers.Dense(10, activation='softmax')(aux1)
x = inception(x, filters_1x1=160, filters_3x3_reduce=112, filters_3x3=224, filters_5x5_reduce=24, filters_5x5=64, fi
x = inception(x, filters_1x1=128, filters_3x3_reduce=128, filters_3x3=256, filters_5x5_reduce=24, filters_5x5=64, fi
x = inception(x, filters_1x1=112, filters_3x3_reduce=144, filters_3x3=288, filters_5x5_reduce=32, filters_5x5=64, fi
aux2 = layers.AveragePooling2D((5, 5), strides=3)(x)
aux2 =layers.Conv2D(128, 1, padding='same', activation='relu')(aux2)
aux2 = layers.Flatten()(aux2)
aux2 = layers.Dense(1024, activation='relu')(aux2)
aux2 = layers.Dropout(0.7)(aux2)
aux2 = layers.Dense(10, activation='softmax')(aux2)
x = inception(x, filters_1x1=256, filters_3x3_reduce=160, filters_3x3=320, filters_5x5_reduce=32, filters_5x5=128, f
x = layers.MaxPooling2D(3, strides=2)(x)
x = inception(x, filters_1x1=256, filters_3x3_reduce=160, filters_3x3=320, filters_5x5_reduce=32, filters_5x5=128, f
x = inception(x, filters_1x1=384, filters_3x3_reduce=192, filters_3x3=384, filters_5x5_reduce=48, filters_5x5=128, f
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dropout(0.4)(x)
out = layers.Dense(10, activation='softmax')(x)
```

```
Metal device set to: Apple M1

2022-09-21 22:36:22.919386: I tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:305] Coul
d not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel may not have been built with NUMA suppo
rt.
2022-09-21 22:36:22.919552: I tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:271] Crea
ted TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0 MB memory) -> physical PluggableDevice (
```

COMPILATION

In [16]:
```
googlenet_model = Model(inputs = inp, outputs = [out, aux1, aux2])
```

In [17]:
```
googlenet_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.01, epsilon=0.05),
              loss=[keras.losses.sparse_categorical_crossentropy,
                    keras.losses.sparse_categorical_crossentropy,
                    keras.losses.sparse_categorical_crossentropy],
              loss_weights=[1, 0.3, 0.3],
              metrics=['accuracy'])
```

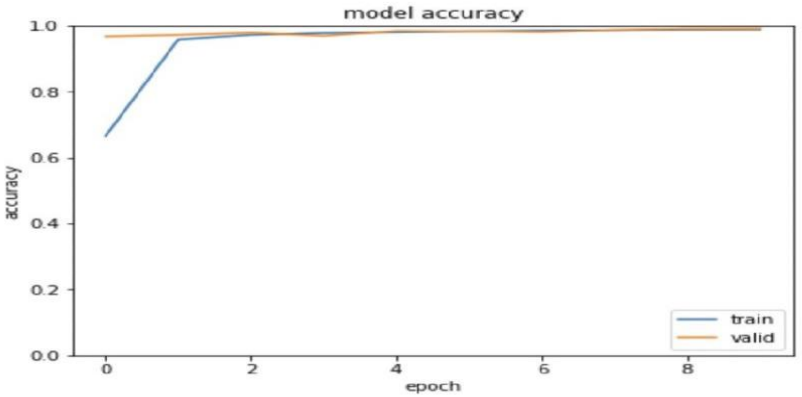
TRAINING

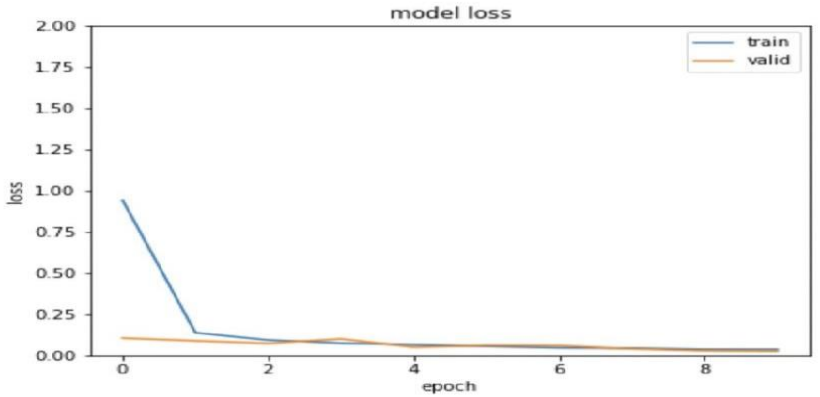
```
In [19]: history = googlenet_model.fit(train_images_stacked, [train_labels, train_labels, train_labels], \
                     validation_data=(test_images_stacked, [test_labels, test_labels, test_labels]), \
                     batch_size=64, epochs=10)
```

```
938/938 [==============================] - 2443s 3s/step - loss: 0.0771 - dense_4_loss: 0.0465 - dense_1_loss: 0.05
34 - dense_3_loss: 0.0486 - dense_4_accuracy: 0.9857 - dense_1_accuracy: 0.9841 - dense_3_accuracy: 0.9853 - val_lo
ss: 0.0955 - val_dense_4_loss: 0.0598 - val_dense_1_loss: 0.0592 - val_dense_3_loss: 0.0600 - val_dense_4_accuracy:
0.9810 - val_dense_1_accuracy: 0.9801 - val_dense_3_accuracy: 0.9804
Epoch 8/10
938/938 [==============================] - 2489s 3s/step - loss: 0.0738 - dense_4_loss: 0.0441 - dense_1_loss: 0.05
23 - dense_3_loss: 0.0467 - dense_4_accuracy: 0.9867 - dense_1_accuracy: 0.9841 - dense_3_accuracy: 0.9855 - val_lo
ss: 0.0597 - val_dense_4_loss: 0.0401 - val_dense_1_loss: 0.0329 - val_dense_3_loss: 0.0324 - val_dense_4_accuracy:
0.9872 - val_dense_1_accuracy: 0.9882 - val_dense_3_accuracy: 0.9891
Epoch 9/10
938/938 [==============================] - 2519s 3s/step - loss: 0.0626 - dense_4_loss: 0.0367 - dense_1_loss: 0.04
57 - dense_3_loss: 0.0406 - dense_4_accuracy: 0.9886 - dense_1_accuracy: 0.9856 - dense_3_accuracy: 0.9875 - val_lo
ss: 0.0508 - val_dense_4_loss: 0.0289 - val_dense_1_loss: 0.0427 - val_dense_3_loss: 0.0304 - val_dense_4_accuracy:
0.9909 - val_dense_1_accuracy: 0.9864 - val_dense_3_accuracy: 0.9904
Epoch 10/10
938/938 [==============================] - 2530s 3s/step - loss: 0.0621 - dense_4_loss: 0.0366 - dense_1_loss: 0.04
54 - dense_3_loss: 0.0394 - dense_4_accuracy: 0.9892 - dense_1_accuracy: 0.9865 - dense_3_accuracy: 0.9880 - val_lo
ss: 0.0423 - val_dense_4_loss: 0.0263 - val_dense_1_loss: 0.0280 - val_dense_3_loss: 0.0252 - val_dense_4_accuracy:
0.9914 - val_dense_1_accuracy: 0.9899 - val_dense_3_accuracy: 0.9924
```

TRAINING AND LOSS CURVES

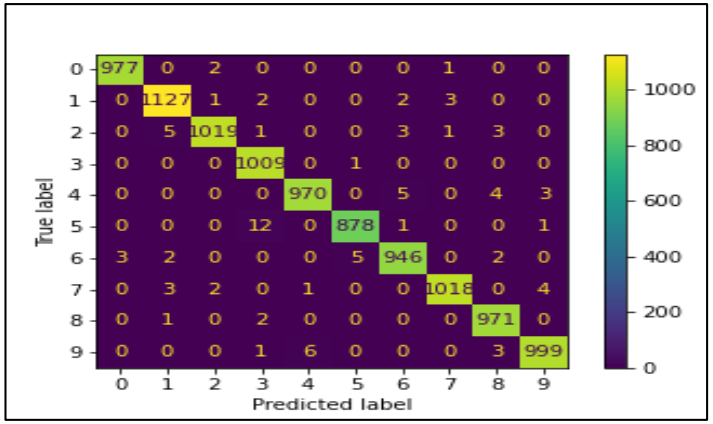
In [25]:
```
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.plot(history.history['dense_4_accuracy'])
plt.plot(history.history['val_dense_4_accuracy'])
plt.ylim(0, 1)
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='lower right')
plt.subplot(1,2,2)
plt.plot(history.history['dense_4_loss'])
plt.plot(history.history['val_dense_4_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='upper right')
plt.ylim([0,2])
plt.savefig("googlenet_loss_acc.png")
```





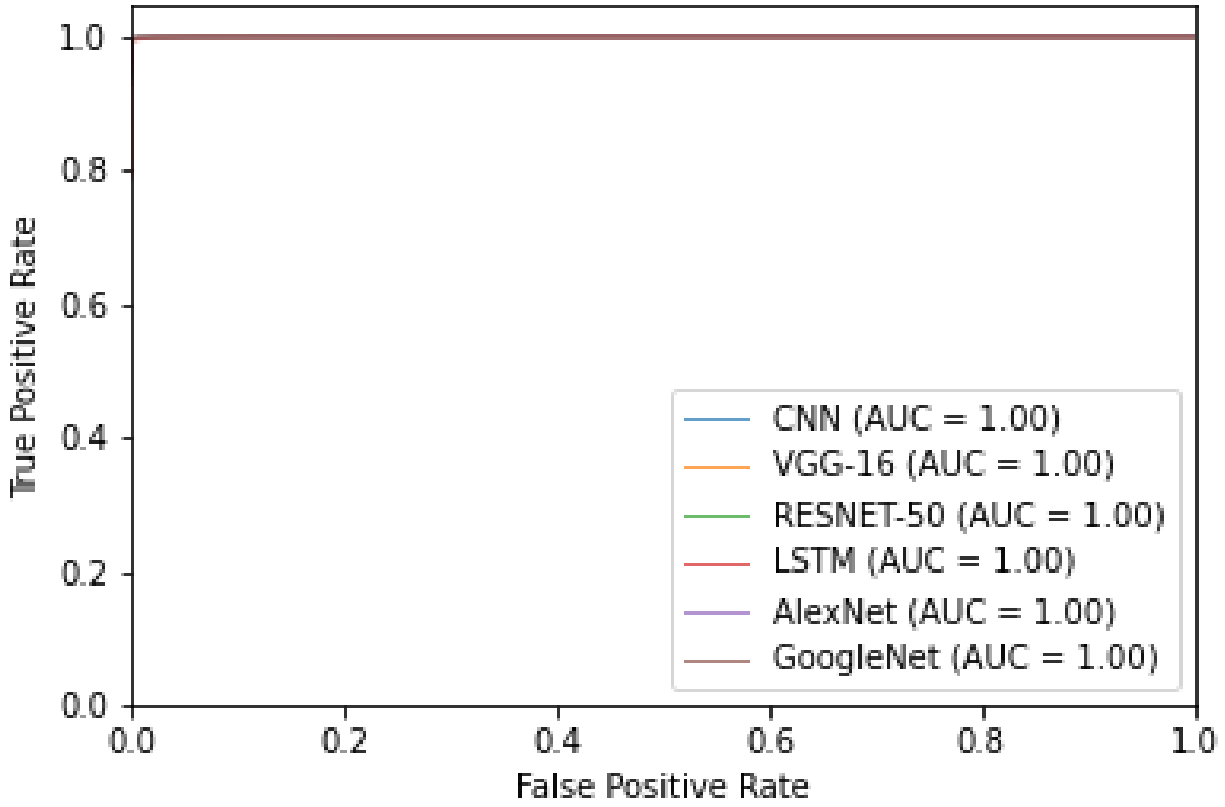
PERFORMANCE

```
313/313 [==============================] - 33s 104ms/step
Confusion Matrix:
[[ 977    0    2    0    0    0    0    1    0    0]
 [   0 1127    1    2    0    0    2    3    0    0]
 [   0    5 1019    1    0    0    3    1    3    0]
 [   0    0    0 1009    0    1    0    0    0    0]
 [   0    0    0    0  970    0    5    0    4    3]
 [   0    0    0   12    0  878    1    0    0    1]
 [   3    2    0    0    0    5  946    0    2    0]
 [   0    3    2    0    1    0    0 1018    0    4]
 [   0    1    0    2    0    0    0    0  971    0]
 [   0    0    0    1    6    0    0    0    3  999]]
--------------------------------------------------------------
--------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       980
           1       0.99      0.99      0.99      1135
           2       1.00      0.99      0.99      1032
           3       0.98      1.00      0.99      1010
           4       0.99      0.99      0.99       982
           5       0.99      0.98      0.99       892
           6       0.99      0.99      0.99       958
           7       1.00      0.99      0.99      1028
           8       0.99      1.00      0.99       974
           9       0.99      0.99      0.99      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000

--------------------------------------------------------------
--------------------------------------------------------------
Accuracy Score:
0.9914
```



COMPARISON BETWEEN CNN, VGG-16, RESNET-50, LSTM(RNN), ALEXNET AND GOOGLENET.

ROC CURVES AND AUC COMPARISON



ROC CURVES AND AUC COMPARISON - CODE (I)

In [40]:

```
def plot_macro_avg_roc_curve(classifier, n_classes, name, X_test, y_test, single_output=True):

    y_test_binarized = label_binarize(y_test, classes=list(range(0,n_classes)))
    y_score = classifier.predict(X_test)

    #for cases where the model (like googlenet) has extra (auxillary) outputs
    if not single_output:
        y_score = y_score[0]

    # Compute ROC curve and ROC area for each class
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], y_score[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # First aggregate all false positive rates
    all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))

    # Then interpolate all ROC curves at this points
    mean_tpr = np.zeros_like(all_fpr)
    for i in range(n_classes):
        mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])

    # Finally average it and compute AUC
    mean_tpr /= n_classes

    fpr["macro"] = all_fpr
    tpr["macro"] = mean_tpr
    roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

    plt.plot(
        fpr["macro"],
        tpr["macro"],
        label="{0} (AUC = {1:0.2f})".format(name, roc_auc["macro"]),
        alpha=0.7,
        linewidth=1,
    )

    return plt
```

ROC CURVES AND AUC COMPARISON - CODE (II)

In [41]:
```
cnn = load_model('cnn.h5')
vgg_based_model = load_model('vgg16.h5')
resnet_based_model = load_model('resnet50.h5')
lstm_based_model = load_model('lstm.h5')
alexnet_model = load_model('alexnet.h5')
googlenet_model = load_model('googlenet.h5')
```

In [44]:
```
plot_macro_avg_roc_curve(cnn, 10, "CNN", test_images, test_labels)
plot_macro_avg_roc_curve(vgg_based_model, 10, "VGG-16", test_images_p, test_labels)
plot_macro_avg_roc_curve(resnet_based_model, 10, "RESNET-50", test_images_pr, test_labels)
plot_macro_avg_roc_curve(lstm_based_model, 10, "LSTM", test_images, test_labels)
plot_macro_avg_roc_curve(alexnet_model, 10, "AlexNet", test_images_stacked, test_labels)
plot_macro_avg_roc_curve(googlenet_model, 10, "GoogleNet", test_images_n_stacked, test_labels, single_output=False)

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")

plt.savefig('roc_auc.png')
```

COMPARISON OF PERFORMANCE

CLASSIFIER	Accuracy	Precision	Recall	F1-Score	AUC
CNN	0.99	0.99	0.99	0.99	1.00
VGG-16	0.99	0.99	0.99	0.99	1.00
RESNET-50	0.99	0.99	0.99	0.99	1.00
LSTM	0.99	0.99	0.99	0.99	1.00
ALEXNET	0.99	0.99	0.99	0.99	1.00
GOOGLENET	0.99	0.99	0.99	0.99	1.00

CONCLUSION:

All the networks performed very well on the MNIST dataset. The images present patterns in all dimensions for different models to exploit. Even the LSTM gave a good performance.

For such simple classifications, (less) time spent in training and predicting is more important as a distinguishing factor. Hence, it can be said that a simple CNN is ideal for this problem.

SAVEE - CLASSIFICATION

CONVERSION TO WAVEFORM (IMAGE)

IMPORTS AND CLASS NAMES

In [1]:
```
import matplotlib.pyplot as plt
import tensorflow as tf
from matplotlib.backend_bases import RendererBase
from scipy import signal
from scipy.io import wavfile
#import soundfile as sf
import os
import numpy as np
from PIL import Image
from scipy.fftpack import fft

%matplotlib inline
```

Some mappings

In [2]:
```
class_names = ['anger', 'disgust', 'fear', 'happiness', 'neutral', 'sadness', 'surprise']
lsymbols = ['a', 'd', 'f', 'h', 'n', 'sa', 'su']
full = {
    'a': 'anger',
    'd': 'disgust',
    'f': 'fear',
    'h': 'happiness',
    'n': 'neutral',
    'sa': 'sadness',
    'su': 'surprise'
}
```

AUDIO PREPROCESSING

Audio Preprocessing (Conversion to images of waveform)

In [3]:
```
audio_path = './AudioData/'
pict_Path = './ImageData/'
samples = []
```

In [4]:
```
if not os.path.exists(pict_Path):
    os.makedirs(pict_Path)

subFolderList = []
for x in os.listdir(audio_path):
    if os.path.isdir(audio_path + '/' + x):
        subFolderList.append(x)
```

In [5]:
```
for x in class_names:
    os.makedirs(pict_Path +'/'+ x)
```

In [6]:
```
def wav2img_waveform(wav_path, targetdir='', figsize=(4,4), num=0):
    samplerate,test_sound  = wavfile.read(wav_path)
    fig = plt.figure(figsize=figsize)
    plt.plot(test_sound)
    plt.axis('off')
    output_file = wav_path.split('/')[-1].split('.wav')[0]
    output_file = targetdir +'/'+ output_file
    plt.savefig(f'{output_file}_{num}.png')
    plt.close()
```

In [7]:
```
num = 0
for i, x in enumerate(subFolderList):
    # get all the wave files
    all_files = [y for y in os.listdir(audio_path + x) if '.wav' in y]
    for file in all_files:
        if file[1].isdigit():
            directory = full[file[0]]
        else:
            directory = full[file[0:2]]
        wav2img_waveform(audio_path + x + '/' + file, pict_Path + '/' + directory, num=num)
        num += 1
```

LOADING THE IMAGES INTO TRAIN AND TEST DATASETS

In [4]:
```
class_names = ['anger', 'disgust', 'fear', 'happiness', 'neutral', 'sadness', 'surprise']
img_height = 224
img_width = 224
num_train_images = 384
num_test_images = 96
num_classes = len(class_names)
```

In [5]:
```
train_ds = tf.keras.utils.image_dataset_from_directory(
  pict_Path,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=num_train_images,
  label_mode='int')
```

```
Found 480 files belonging to 7 classes.
Using 384 files for training.
Metal device set to: Apple M1
```

```
2022-09-23 17:51:24.731527: I tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:305] Coul
d not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel may not have been built with NUMA suppo
rt.
2022-09-23 17:51:24.731713: I tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:271] Crea
ted TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0 MB memory) -> physical PluggableDevice (
device: 0, name: METAL, pci bus id: <undefined>)
```

In [6]:
```
test_ds = tf.keras.utils.image_dataset_from_directory(
  pict_Path,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=num_test_images,
  label_mode='int')
```

```
Found 480 files belonging to 7 classes.
Using 96 files for validation.
```

BREAKING THE DATASETS INTO IMAGES AND LABELS

```
In [7]: train_images = train_labels = test_images = test_labels = None

In [60]: train_images, train_labels = next(train_ds.as_numpy_iterator())

In [62]: test_images, test_labels = next(test_ds.as_numpy_iterator())
```

CNN

IMPORTS AND MODEL ARCHITECTURE

Convolutional Neural Network

Importing the libraries

In [10]:
```
from tensorflow import keras
from tensorflow.keras import layers, datasets
from tensorflow.keras.models import Sequential
import matplotlib.pyplot as plt
```

Building the CNN

In [37]:
```
cnn = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(filters=32, kernel_size=3, padding='same', activation='relu'),
  layers.Conv2D(filters=32, kernel_size=3, padding='same', activation='relu'),
  layers.MaxPooling2D(pool_size=2, strides=2),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(pool_size=2, strides=2),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes, activation="softmax")
])
```

COMPILATION AND SUMMARY

Compiling the CNN

In [38]:
```
cnn.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0005, epsilon=0.01),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
```

In [39]:
```
cnn.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_3 (Rescaling)     (None, 224, 224, 3)       0

 conv2d_9 (Conv2D)           (None, 224, 224, 32)      896

 conv2d_10 (Conv2D)          (None, 224, 224, 32)      9248

 max_pooling2d_6 (MaxPooling  (None, 112, 112, 32)     0
 2D)

 conv2d_11 (Conv2D)          (None, 112, 112, 32)      9248

 max_pooling2d_7 (MaxPooling  (None, 56, 56, 32)       0
 2D)

 dropout_3 (Dropout)         (None, 56, 56, 32)        0

 flatten_3 (Flatten)         (None, 100352)            0

 dense_6 (Dense)             (None, 128)               12845184

 dense_7 (Dense)             (None, 7)                 903

=================================================================
Total params: 12,865,479
Trainable params: 12,865,479
Non-trainable params: 0
_________________________________________________________________
```

TRAINING AND SAVING

Training the CNN on the Training set and evaluating it on the Test set

In [40]:
```
history = cnn.fit(train_images, train_labels, epochs=20, validation_data=(test_images, test_labels))
```

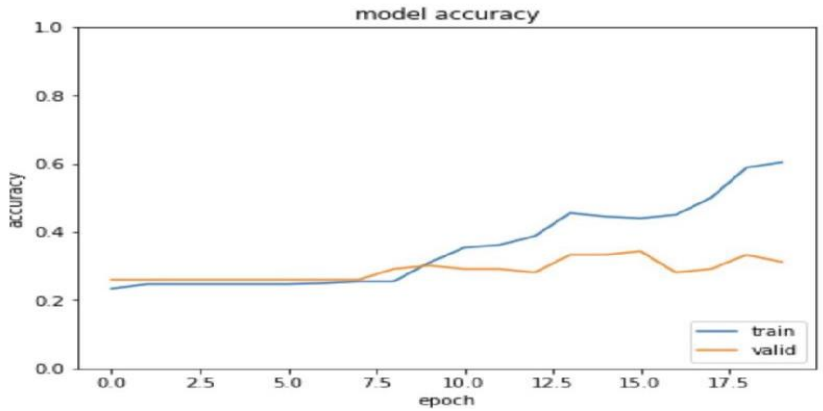
```
ccuracy: 0.3333
Epoch 15/20
12/12 [==============================] - 3s 254ms/step - loss: 1.5198 - accuracy: 0.4453 - val_loss: 1.8071 - val_a
ccuracy: 0.3333
Epoch 16/20
12/12 [==============================] - 3s 254ms/step - loss: 1.4734 - accuracy: 0.4401 - val_loss: 1.8280 - val_a
ccuracy: 0.3438
Epoch 17/20
12/12 [==============================] - 3s 255ms/step - loss: 1.4640 - accuracy: 0.4505 - val_loss: 1.7854 - val_a
ccuracy: 0.2812
Epoch 18/20
12/12 [==============================] - 3s 258ms/step - loss: 1.3295 - accuracy: 0.5000 - val_loss: 1.8504 - val_a
ccuracy: 0.2917
Epoch 19/20
12/12 [==============================] - 3s 255ms/step - loss: 1.2158 - accuracy: 0.5885 - val_loss: 1.8208 - val_a
ccuracy: 0.3333
Epoch 20/20
12/12 [==============================] - 3s 258ms/step - loss: 1.1194 - accuracy: 0.6042 - val_loss: 1.8733 - val_a
ccuracy: 0.3125
```

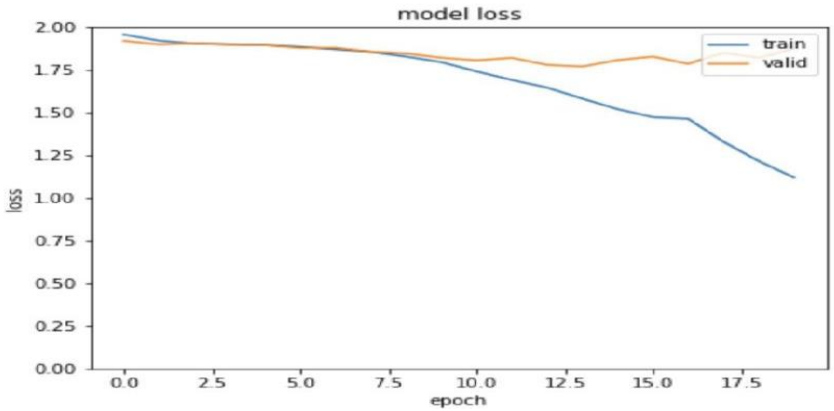
In [41]:
```
cnn.save('cnn.h5')
```

TRAINING AND LOSS CURVES

In [30]:
```
def save_train_loss_curves(history, filename):
    plt.figure(figsize=(15,5))
    plt.subplot(1,2,1)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.ylim(0, 1)
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'valid'], loc='lower right')
    plt.subplot(1,2,2)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'valid'], loc='upper right')
    plt.ylim([0,3])
    plt.savefig(filename)
```

In [47]:
```
save_train_loss_curves(history, 'cnn_loss_acc.png')
```





PERFORMANCE - CODE

Performance

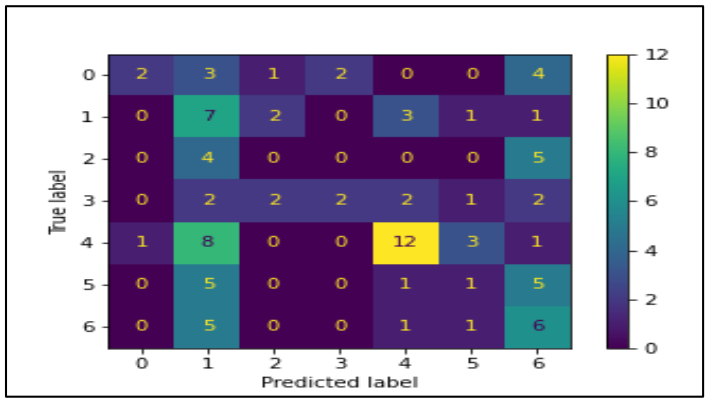
In [11]:
```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, ConfusionMatrixDisplay
```

In [12]:
```
def get_predictions(model, inputs):

    def index_max_arr_element(arr):
        best_index = -1
        best = float('-inf')
        for i, num in enumerate(arr):
            if num > best:
                best = num
                best_index = i
        return best_index

    pred_arr = model.predict(inputs)
    pred_labels = [index_max_arr_element(row) for row in pred_arr]
    return pred_labels
```

In [45]:
```
pred_labels = get_predictions(cnn, test_images)

print("Confusion Matrix:")
print(confusion_matrix(test_labels, pred_labels))
print("------------------------------------------------------------")
print("------------------------------------------------------------")
print("Performance Evaluation:")
print(classification_report(test_labels, pred_labels))
print("------------------------------------------------------------")
print("------------------------------------------------------------")
print("Accuracy Score:")
print(accuracy_score(test_labels, pred_labels))

plt.figure()
ConfusionMatrixDisplay.from_predictions(test_labels, pred_labels)
plt.savefig('./cnn_matrix.png')
```

PERFORMANCE

```
Confusion Matrix:
[[ 2  3  1  2  0  0  4]
 [ 0  7  2  0  3  1  1]
 [ 0  4  0  0  0  0  5]
 [ 0  2  2  2  2  1  2]
 [ 1  8  0  0 12  3  1]
 [ 0  5  0  0  1  1  5]
 [ 0  5  0  0  1  1  6]]
--------------------------------------------------------
--------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           0       0.67      0.17      0.27        12
           1       0.21      0.50      0.29        14
           2       0.00      0.00      0.00         9
           3       0.50      0.18      0.27        11
           4       0.63      0.48      0.55        25
           5       0.14      0.08      0.11        12
           6       0.25      0.46      0.32        13

    accuracy                           0.31        96
   macro avg       0.34      0.27      0.26        96
weighted avg       0.39      0.31      0.31        96

--------------------------------------------------------
--------------------------------------------------------
Accuracy Score:
0.3125
```



VGG-16

IMPORTS, DATA PREPROCESSING & ARCHITECTURE

In [55]:
```
from tensorflow.keras.applications import vgg16
```

In [56]:
```
train_images_p, test_images_p = vgg16.preprocess_input(np.copy(train_images)), vgg16.preprocess_input(np.copy(test_i
```

In [50]:
```
vgg_based_model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  vgg16.VGG16(include_top=False, weights="imagenet", input_shape=(img_height, img_width, 3), pooling="max"),
  layers.Flatten(),
  layers.Dense(512, activation=('relu')),
  layers.Dropout(0.2),
  layers.Dense(128, activation=('relu')),
  layers.Dropout(0.2),
  layers.Dense(num_classes, activation="softmax")
])
```

COMPILATION AND SUMMARY

In [51]:
```
vgg_based_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0005, epsilon=0.01),
                        loss=tf.keras.losses.SparseCategoricalCrossentropy(),
                        metrics=['accuracy'])
```

In [52]:
```
vgg_based_model.summary()
```

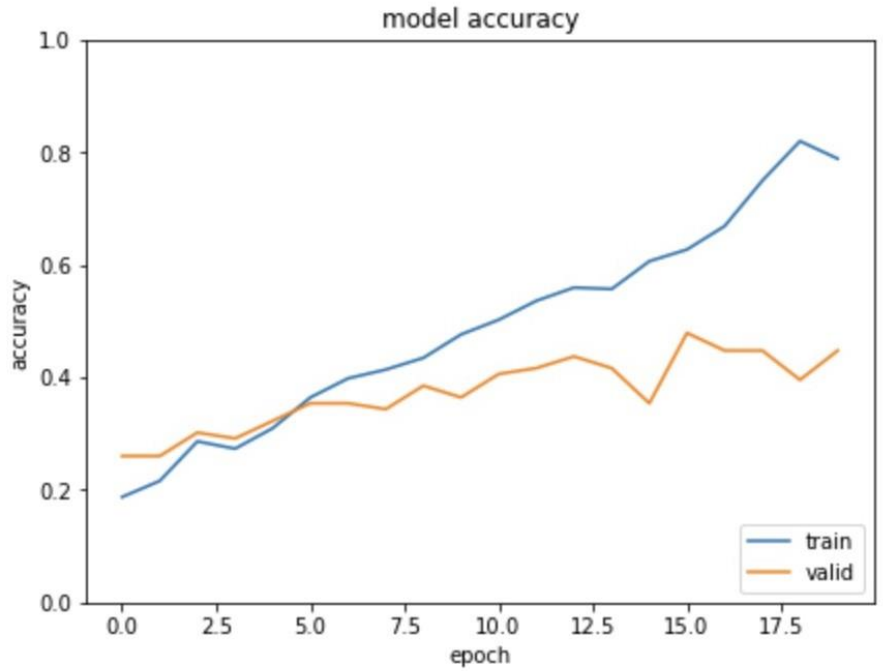
```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_4 (Rescaling)     (None, 224, 224, 3)       0

 vgg16 (Functional)          (None, 512)               14714688

 flatten_4 (Flatten)         (None, 512)               0

 dense_8 (Dense)             (None, 512)               262656

 dropout_4 (Dropout)         (None, 512)               0

 dense_9 (Dense)             (None, 128)               65664

 dropout_5 (Dropout)         (None, 128)               0

 dense_10 (Dense)            (None, 7)                 903

=================================================================
Total params: 15,043,911
Trainable params: 15,043,911
Non-trainable params: 0
_________________________________________________________________
```

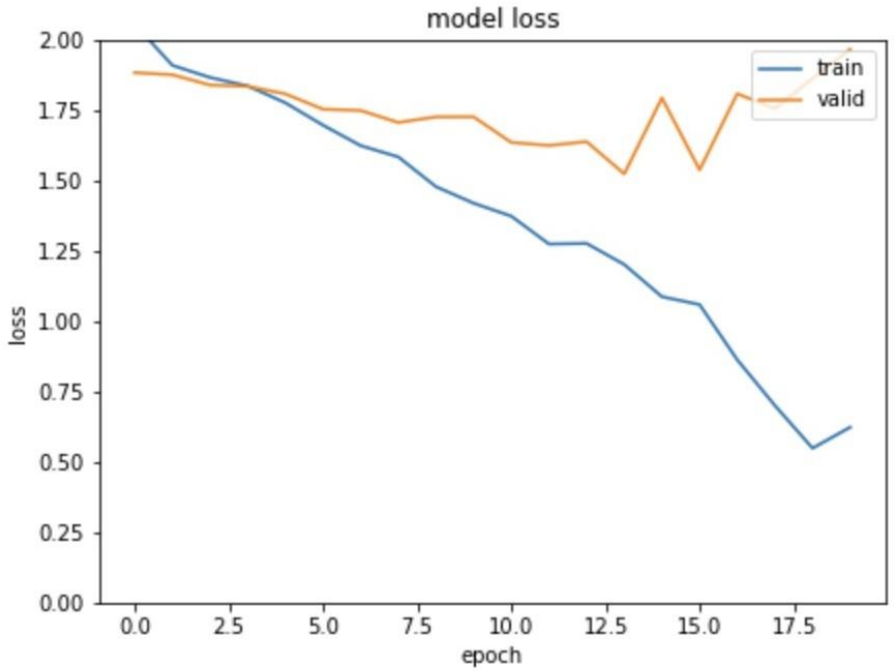
TRAINING AND SAVING

```
In [53]: history = vgg_based_model.fit(train_images_p, train_labels, epochs=20, validation_data=(test_images_p, test_labels))
uracy: 0.4167
Epoch 15/20
12/12 [==============================] - 40s 3s/step - loss: 1.0874 - accuracy: 0.6068 - val_loss: 1.7943 - val_acc
uracy: 0.3542
Epoch 16/20
12/12 [==============================] - 40s 3s/step - loss: 1.0594 - accuracy: 0.6276 - val_loss: 1.5384 - val_acc
uracy: 0.4792
Epoch 17/20
12/12 [==============================] - 40s 3s/step - loss: 0.8631 - accuracy: 0.6693 - val_loss: 1.8091 - val_acc
uracy: 0.4479
Epoch 18/20
12/12 [==============================] - 40s 3s/step - loss: 0.7004 - accuracy: 0.7500 - val_loss: 1.7563 - val_acc
uracy: 0.4479
Epoch 19/20
12/12 [==============================] - 40s 3s/step - loss: 0.5487 - accuracy: 0.8203 - val_loss: 1.8636 - val_acc
uracy: 0.3958
Epoch 20/20
12/12 [==============================] - 40s 3s/step - loss: 0.6232 - accuracy: 0.7891 - val_loss: 1.9675 - val_acc
uracy: 0.4479

In [55]: vgg_based_model.save('vgg16.h5')
```

TRAINING AND LOSS CURVES

```
n [54]: save_train_loss_curves(history, 'vgg_loss_acc.png')
```



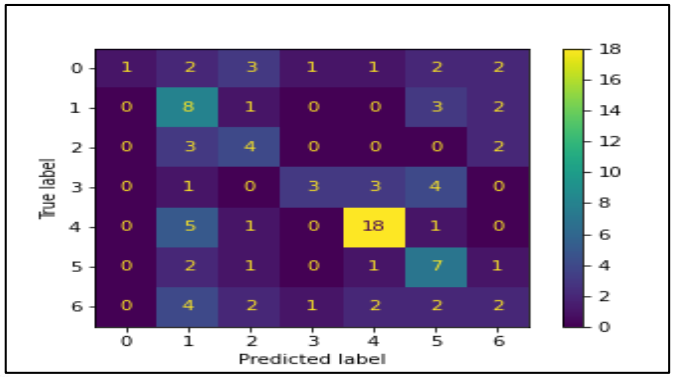


PERFORMANCE

```
Confusion Matrix:
[[ 1  2  3  1  1  2  2]
 [ 0  8  1  0  0  3  2]
 [ 0  3  4  0  0  0  2]
 [ 0  1  0  3  3  4  0]
 [ 0  5  1  0 18  1  0]
 [ 0  2  1  0  1  7  1]
 [ 0  4  2  1  2  2  2]]
```

```
Performance Evaluation:
              precision    recall  f1-score   support

           0       1.00      0.08      0.15        12
           1       0.32      0.57      0.41        14
           2       0.33      0.44      0.38         9
           3       0.60      0.27      0.37        11
           4       0.72      0.72      0.72        25
           5       0.37      0.58      0.45        12
           6       0.22      0.15      0.18        13

    accuracy                           0.45        96
   macro avg       0.51      0.40      0.38        96
weighted avg       0.54      0.45      0.43        96
```

```
Accuracy Score:
0.4479166666666667
```



RESNET-50

IMPORTS, DATA PREPROCESSING & ARCHITECTURE

RESNET-50

In [57]:
```
from tensorflow.keras.applications import resnet50
```

In [58]:
```
train_images_pr, test_images_pr = resnet50.preprocess_input(np.copy(train_images)), resnet50.preprocess_input(np.cop
```

In [17]:
```
resnet_based_model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  resnet50.ResNet50(include_top=False, weights="imagenet", input_shape=(img_height, img_width, 3), pooling="max"),
  layers.Flatten(),
  layers.Dense(512, activation=('relu')),
  layers.Dropout(0.2),
  layers.Dense(128, activation=('relu')),
  layers.Dropout(0.2),
  layers.Dense(num_classes, activation="softmax")
])
```

COMPILATION AND SUMMARY

In [18]:
```
resnet_based_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0005, epsilon=0.01),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
```

In [19]:
```
resnet_based_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 224, 224, 3)       0

 resnet50 (Functional)       (None, 2048)              23587712

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 512)               1049088

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 128)               65664

 dropout_1 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 7)                 903

=================================================================
Total params: 24,703,367
Trainable params: 24,650,247
Non-trainable params: 53,120
_________________________________________________________________
```

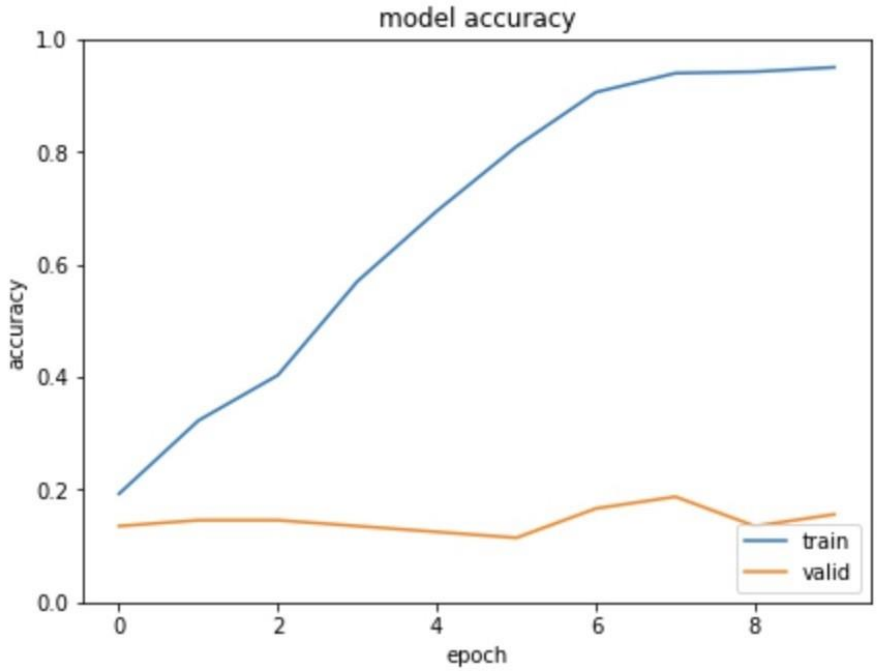
TRAINING AND SAVING

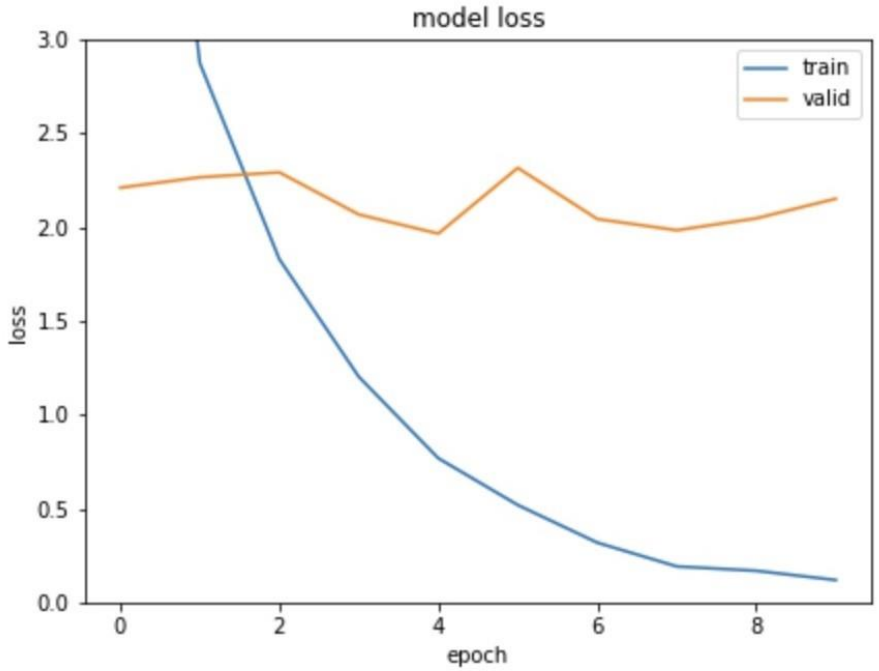
```
In [20]: history = resnet_based_model.fit(train_images_pr, train_labels, epochs=10, validation_data=(test_images_pr, test_lab
uracy: 0.1354
Epoch 5/10
12/12 [==============================] - 15s 1s/step - loss: 0.7679 - accuracy: 0.6953 - val_loss: 1.9651 - val_acc
uracy: 0.1250
Epoch 6/10
12/12 [==============================] - 15s 1s/step - loss: 0.5185 - accuracy: 0.8099 - val_loss: 2.3155 - val_acc
uracy: 0.1146
Epoch 7/10
12/12 [==============================] - 16s 1s/step - loss: 0.3179 - accuracy: 0.9062 - val_loss: 2.0430 - val_acc
uracy: 0.1667
Epoch 8/10
12/12 [==============================] - 18s 1s/step - loss: 0.1912 - accuracy: 0.9401 - val_loss: 1.9831 - val_acc
uracy: 0.1875
Epoch 9/10
12/12 [==============================] - 20s 2s/step - loss: 0.1681 - accuracy: 0.9427 - val_loss: 2.0465 - val_acc
uracy: 0.1354
Epoch 10/10
12/12 [==============================] - 21s 2s/step - loss: 0.1185 - accuracy: 0.9505 - val_loss: 2.1512 - val_acc
uracy: 0.1562
```

```
In [27]: resnet_based_model.save('resnet50.h5')
```

TRAINING AND LOSS CURVES

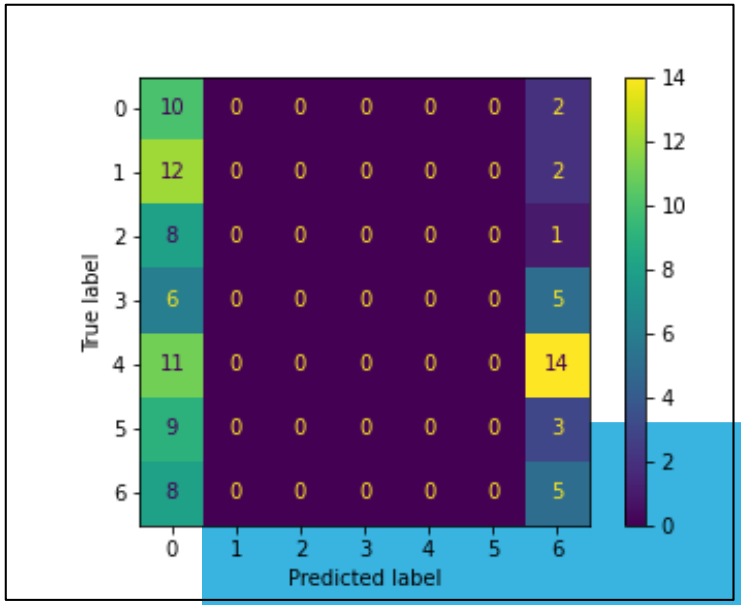
```
In [26]: save_train_loss_curves(history, 'resnet_loss_acc.png')
```





PERFORMANCE

```
Confusion Matrix:
[[10  0  0  0  0  0  2]
 [12  0  0  0  0  0  2]
 [ 8  0  0  0  0  0  1]
 [ 6  0  0  0  0  0  5]
 [11  0  0  0  0  0 14]
 [ 9  0  0  0  0  0  3]
 [ 8  0  0  0  0  0  5]]
--------------------------------------------------------
--------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           0       0.16      0.83      0.26        12
           1       0.00      0.00      0.00        14
           2       0.00      0.00      0.00         9
           3       0.00      0.00      0.00        11
           4       0.00      0.00      0.00        25
           5       0.00      0.00      0.00        12
           6       0.16      0.38      0.22        13

    accuracy                           0.16        96
   macro avg       0.04      0.17      0.07        96
weighted avg       0.04      0.16      0.06        96

--------------------------------------------------------
--------------------------------------------------------
Accuracy Score:
0.15625
```



RNN (LSTM)

MODEL ARCHITECTURE AND COMPILATION

RNN (LSTM based)

In [24]:
```
lstm_based_model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Resizing(64, 64),
  layers.Reshape((4096, 3)),
  layers.LSTM(units=50, return_sequences=True),
  layers.Dropout(0.2),
  layers.LSTM(units=50),
  layers.Dropout(0.2),
  layers.Dense(128, activation=('relu')),
  layers.Dropout(0.2),
  layers.Dense(num_classes, activation="softmax")
])
```

In [25]:
```
lstm_based_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
```

TRAINING AND SAVING

```
In [27]: history = lstm_based_model.fit(train_images, train_labels, epochs=20, validation_data=(test_images, test_labels))
```

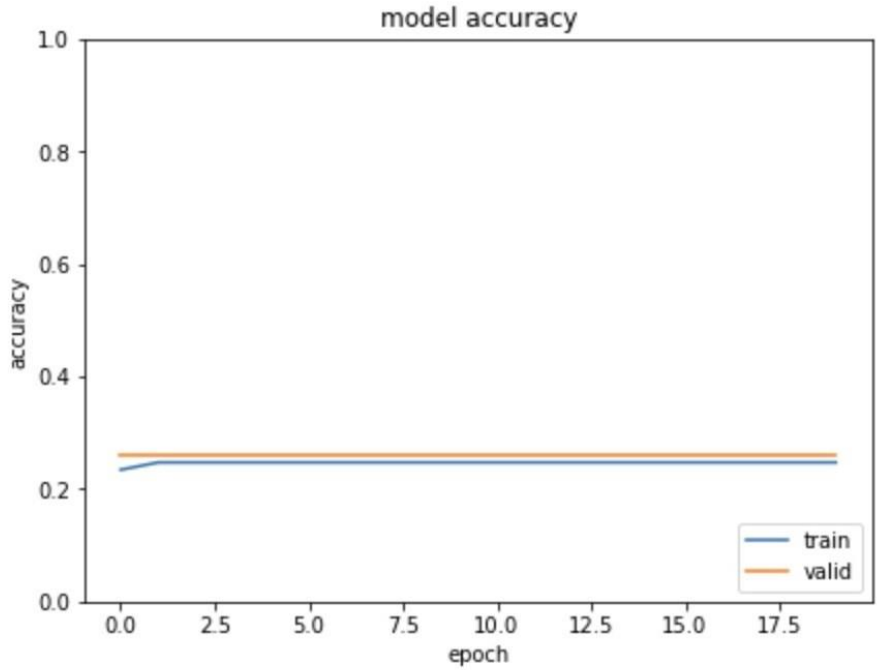
```
uracy: 0.2604
Epoch 15/20
12/12 [==============================] - 43s 4s/step - loss: 1.9130 - accuracy: 0.2474 - val_loss: 1.9036 - val_acc
uracy: 0.2604
Epoch 16/20
12/12 [==============================] - 42s 4s/step - loss: 1.9091 - accuracy: 0.2474 - val_loss: 1.9030 - val_acc
uracy: 0.2604
Epoch 17/20
12/12 [==============================] - 43s 4s/step - loss: 1.9039 - accuracy: 0.2474 - val_loss: 1.9030 - val_acc
uracy: 0.2604
Epoch 18/20
12/12 [==============================] - 43s 4s/step - loss: 1.9095 - accuracy: 0.2474 - val_loss: 1.9016 - val_acc
uracy: 0.2604
Epoch 19/20
12/12 [==============================] - 43s 4s/step - loss: 1.9072 - accuracy: 0.2474 - val_loss: 1.9030 - val_acc
uracy: 0.2604
Epoch 20/20
12/12 [==============================] - 43s 4s/step - loss: 1.9059 - accuracy: 0.2474 - val_loss: 1.9033 - val_acc
uracy: 0.2604
```

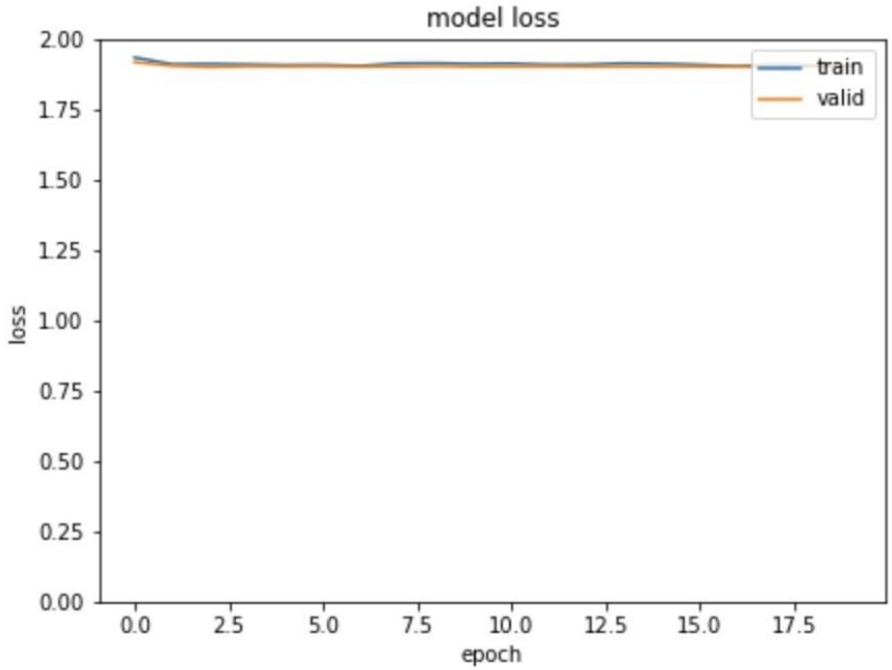
```
In [28]: lstm_based_model.save('lstm.h5')
```

TRAINING AND LOSS CURVES

```
In [28]: lstm_based_model.save('lstm.h5')
```

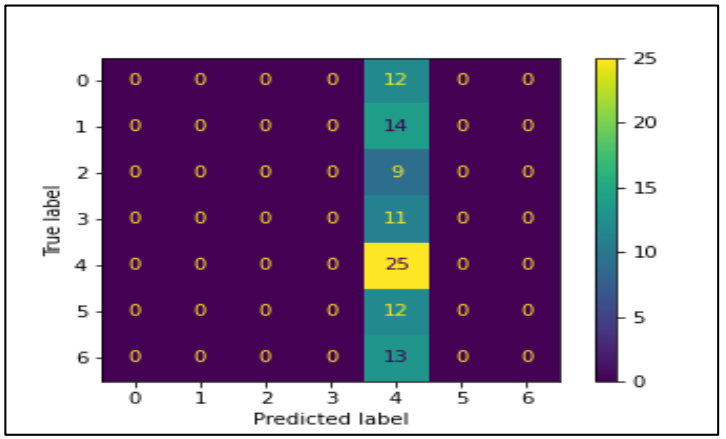
```
In [29]: save_train_loss_curves(history, 'lstm_loss_acc.png')
```





PERFORMANCE

```
Confusion Matrix:
[[ 0  0  0  0 12  0  0]
 [ 0  0  0  0 14  0  0]
 [ 0  0  0  0  9  0  0]
 [ 0  0  0  0 11  0  0]
 [ 0  0  0  0 25  0  0]
 [ 0  0  0  0 12  0  0]
 [ 0  0  0  0 13  0  0]]
------------------------------------------------------------
------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        12
           1       0.00      0.00      0.00        14
           2       0.00      0.00      0.00         9
           3       0.00      0.00      0.00        11
           4       0.26      1.00      0.41        25
           5       0.00      0.00      0.00        12
           6       0.00      0.00      0.00        13

    accuracy                           0.26        96
   macro avg       0.04      0.14      0.06        96
weighted avg       0.07      0.26      0.11        96

------------------------------------------------------------
------------------------------------------------------------
Accuracy Score:
0.2604166666666667
```



ALEXNET

MODEL ARCHITECTURE - I

In [14]:
```
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D, ZeroPadding2D, BatchNo
from tensorflow.keras.regularizers import l2
```

In [23]:
```
def alexnet_model(img_shape=(32, 32, 3), n_classes=10, l2_reg=0.,
    weights=None):

    # Initialize model
    alexnet = Sequential()
    alexnet.add(layers.Input(shape=img_shape))
    alexnet.add(layers.Resizing(64, 64))

    # Layer 1
    alexnet.add(Conv2D(96, (11, 11), padding='same', kernel_regularizer=l2(l2_reg)))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(MaxPooling2D(pool_size=(2, 2)))

    # Layer 2
    alexnet.add(Conv2D(256, (5, 5), padding='same'))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(MaxPooling2D(pool_size=(2, 2)))

    # Layer 3
    alexnet.add(ZeroPadding2D((1, 1)))
    alexnet.add(Conv2D(512, (3, 3), padding='same'))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(MaxPooling2D(pool_size=(2, 2)))

    # Layer 4
    alexnet.add(ZeroPadding2D((1, 1)))
    alexnet.add(Conv2D(1024, (3, 3), padding='same'))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
```

MODEL ARCHITECTURE - II

```
    # Layer 5
    alexnet.add(ZeroPadding2D((1, 1)))
    alexnet.add(Conv2D(1024, (3, 3), padding='same'))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(MaxPooling2D(pool_size=(2, 2)))

    # Layer 6
    alexnet.add(Flatten())
    alexnet.add(Dense(3072))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(Dropout(0.5))

    # Layer 7
    alexnet.add(Dense(4096))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(Dropout(0.5))

    # Layer 8
    alexnet.add(Dense(n_classes))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('softmax'))

    if weights is not None:
        alexnet.load_weights(weights)

    return alexnet
```

In [24]:
```
alexnet_model = alexnet_model(img_shape=(img_height, img_width, 3), n_classes=num_classes)
```

In [25]:
```
alexnet_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0002, epsilon=0.05),
                      loss=tf.keras.losses.SparseCategoricalCrossentropy(),
                      metrics=['accuracy'])
```

TRAINING AND SAVING

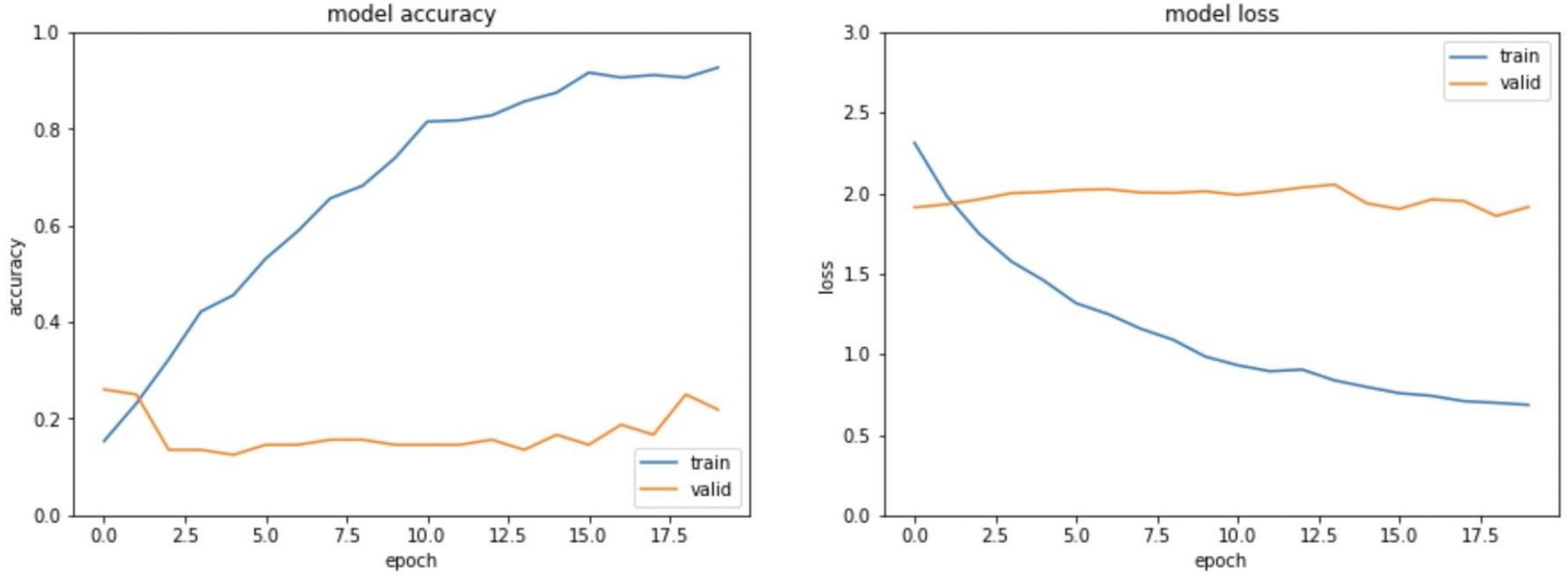
```
In [27]: history = alexnet_model.fit(train_images, train_labels, epochs=20, validation_data=(test_images, test_labels))
```

```
ccuracy: 0.1354
Epoch 15/20
12/12 [==============================] - 9s 746ms/step - loss: 0.7966 - accuracy: 0.8750 - val_loss: 1.9376 - val_a
ccuracy: 0.1667
Epoch 16/20
12/12 [==============================] - 9s 762ms/step - loss: 0.7583 - accuracy: 0.9167 - val_loss: 1.9018 - val_a
ccuracy: 0.1458
Epoch 17/20
12/12 [==============================] - 9s 779ms/step - loss: 0.7418 - accuracy: 0.9062 - val_loss: 1.9614 - val_a
ccuracy: 0.1875
Epoch 18/20
12/12 [==============================] - 9s 781ms/step - loss: 0.7084 - accuracy: 0.9115 - val_loss: 1.9521 - val_a
ccuracy: 0.1667
Epoch 19/20
12/12 [==============================] - 9s 786ms/step - loss: 0.6981 - accuracy: 0.9062 - val_loss: 1.8590 - val_a
ccuracy: 0.2500
Epoch 20/20
12/12 [==============================] - 9s 789ms/step - loss: 0.6857 - accuracy: 0.9271 - val_loss: 1.9149 - val_a
ccuracy: 0.2188
```

```
In [28]: alexnet_model.save('alexnet.h5')
```

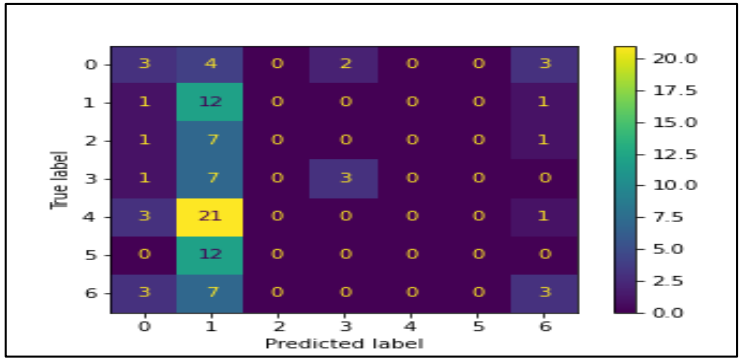
TRAINING AND LOSS CURVES

```
In [31]: save_train_loss_curves(history, 'alexnet_loss_acc.png')
```



PERFORMANCE

```
Confusion Matrix:
[[ 3  4  0  2  0  0  3]
 [ 1 12  0  0  0  0  1]
 [ 1  7  0  0  0  0  1]
 [ 1  7  0  3  0  0  0]
 [ 3 21  0  0  0  0  1]
 [ 0 12  0  0  0  0  0]
 [ 3  7  0  0  0  0  3]]
------------------------------------------------------------
------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           0       0.25      0.25      0.25        12
           1       0.17      0.86      0.29        14
           2       0.00      0.00      0.00         9
           3       0.60      0.27      0.37        11
           4       0.00      0.00      0.00        25
           5       0.00      0.00      0.00        12
           6       0.33      0.23      0.27        13

    accuracy                           0.22        96
   macro avg       0.19      0.23      0.17        96
weighted avg       0.17      0.22      0.15        96

------------------------------------------------------------
------------------------------------------------------------
Accuracy Score:
0.21875
```



GOOGLENET

IMPORTS, DATA PREPROCESSING AND INCEPTION MODULE

GOOGLENET

In [34]:
```
from tensorflow.keras.layers import Flatten, Input, Conv2D, Dense, Dropout, MaxPooling2D, AveragePooling2D, GlobalAv
from tensorflow.keras.layers import concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.regularizers import l2
```

In [35]:
```
train_images_normalized = train_images/255.0
test_images_normalized = test_images/255.0
```

In [36]:
```
def inception(x,
              filters_1x1,
              filters_3x3_reduce,
              filters_3x3,
              filters_5x5_reduce,
              filters_5x5,
              filters_pool):
    path1 = layers.Conv2D(filters_1x1, (1, 1), padding='same',    activation='relu')(x)
    path2 = layers.Conv2D(filters_3x3_reduce, (1, 1), padding='same', activation='relu')(x)
    path2 = layers.Conv2D(filters_3x3, (1, 1), padding='same', activation='relu')(path2)
    path3 = layers.Conv2D(filters_5x5_reduce, (1, 1), padding='same', activation='relu')(x)
    path3 = layers.Conv2D(filters_5x5, (1, 1), padding='same', activation='relu')(path3)
    path4 = layers.MaxPool2D((3, 3), strides=(1, 1), padding='same')(x)
    path4 = layers.Conv2D(filters_pool, (1, 1), padding='same', activation='relu')(path4)
    return tf.concat([path1, path2, path3, path4], axis=3)
```

MODEL ARCHITECTURE

In [40]:
```
inp = layers.Input(shape=(224, 224, 3))
x = layers.Conv2D(64, 7, strides=2, padding='same', activation='relu')(inp)
x = layers.MaxPooling2D(3, strides=2)(x)
x = layers.Conv2D(64, 1, strides=1, padding='same', activation='relu')(x)
x = layers.Conv2D(192, 3, strides=1, padding='same', activation='relu')(x)
x = layers.MaxPooling2D(3, strides=2)(x)
x = inception(x, filters_1x1=64, filters_3x3_reduce=96, filters_3x3=128, filters_5x5_reduce=16, filters_5x5=32, filt
x = inception(x, filters_1x1=128, filters_3x3_reduce=128, filters_3x3=192, filters_5x5_reduce=32, filters_5x5=96, fi
x = layers.MaxPooling2D(3, strides=2)(x)
x = inception(x, filters_1x1=192, filters_3x3_reduce=96, filters_3x3=208, filters_5x5_reduce=16, filters_5x5=48, fil
aux1 = layers.AveragePooling2D((5, 5), strides=3)(x)
aux1 =layers.Conv2D(128, 1, padding='same', activation='relu')(aux1)
aux1 = layers.Flatten()(aux1)
aux1 = layers.Dense(1024, activation='relu')(aux1)
aux1 = layers.Dropout(0.7)(aux1)
aux1 = layers.Dense(10, activation='softmax')(aux1)
x = inception(x, filters_1x1=160, filters_3x3_reduce=112, filters_3x3=224, filters_5x5_reduce=24, filters_5x5=64, fi
x = inception(x, filters_1x1=128, filters_3x3_reduce=128, filters_3x3=256, filters_5x5_reduce=24, filters_5x5=64, fi
x = inception(x, filters_1x1=112, filters_3x3_reduce=144, filters_3x3=288, filters_5x5_reduce=32, filters_5x5=64, fi
aux2 = layers.AveragePooling2D((5, 5), strides=3)(x)
aux2 =layers.Conv2D(128, 1, padding='same', activation='relu')(aux2)
aux2 = layers.Flatten()(aux2)
aux2 = layers.Dense(1024, activation='relu')(aux2)
aux2 = layers.Dropout(0.7)(aux2)
aux2 = layers.Dense(10, activation='softmax')(aux2)
x = inception(x, filters_1x1=256, filters_3x3_reduce=160, filters_3x3=320, filters_5x5_reduce=32, filters_5x5=128, f
x = layers.MaxPooling2D(3, strides=2)(x)
x = inception(x, filters_1x1=256, filters_3x3_reduce=160, filters_3x3=320, filters_5x5_reduce=32, filters_5x5=128, f
x = inception(x, filters_1x1=384, filters_3x3_reduce=192, filters_3x3=384, filters_5x5_reduce=48, filters_5x5=128, f
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dropout(0.4)(x)
out = layers.Dense(10, activation='softmax')(x)
```

In [41]:
```
googlenet_model = Model(inputs = inp, outputs = [out, aux1, aux2])
```

COMPILATION

In [42]:
```
googlenet_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.01, epsilon=0.05),
                loss=[keras.losses.sparse_categorical_crossentropy,
                      keras.losses.sparse_categorical_crossentropy,
                      keras.losses.sparse_categorical_crossentropy],
                loss_weights=[1, 0.3, 0.3],
                metrics=['accuracy'])
```

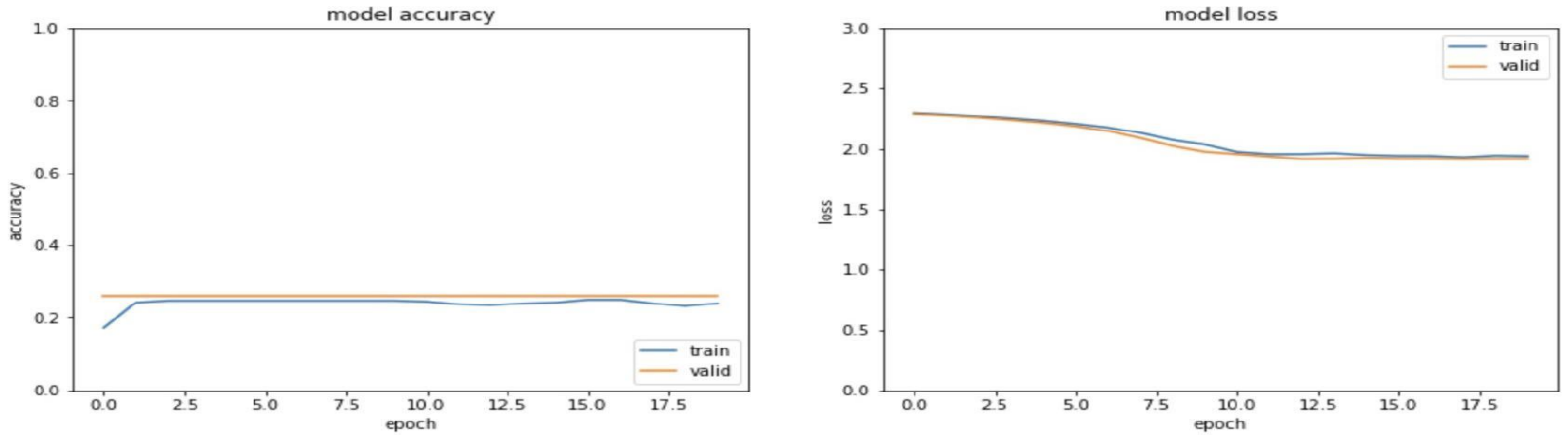
TRAINING

```
In [43]: history = googlenet_model.fit(train_images_normalized, [train_labels, train_labels, train_labels], \
                     validation_data=(test_images_normalized, [test_labels, test_labels, test_labels]), \
                     batch_size=64, epochs=20)
```

```
6/6 [==============================] - 7s 1s/step - loss: 3.1279 - dense_20_loss: 1.9389 - dense_17_loss: 1.9769 -
dense_19_loss: 1.9865 - dense_20_accuracy: 0.2500 - dense_17_accuracy: 0.2266 - dense_19_accuracy: 0.2109 - val_los
s: 3.0757 - val_dense_20_loss: 1.9162 - val_dense_17_loss: 1.9300 - val_dense_19_loss: 1.9349 - val_dense_20_accura
cy: 0.2604 - val_dense_17_accuracy: 0.2604 - val_dense_19_accuracy: 0.2604
Epoch 18/20
6/6 [==============================] - 7s 1s/step - loss: 3.0956 - dense_20_loss: 1.9276 - dense_17_loss: 1.9489 -
dense_19_loss: 1.9446 - dense_20_accuracy: 0.2396 - dense_17_accuracy: 0.2344 - dense_19_accuracy: 0.2448 - val_los
s: 3.0679 - val_dense_20_loss: 1.9132 - val_dense_17_loss: 1.9202 - val_dense_19_loss: 1.9288 - val_dense_20_accura
cy: 0.2604 - val_dense_17_accuracy: 0.2604 - val_dense_19_accuracy: 0.2604
Epoch 19/20
6/6 [==============================] - 7s 1s/step - loss: 3.1197 - dense_20_loss: 1.9413 - dense_17_loss: 1.9704 -
dense_19_loss: 1.9575 - dense_20_accuracy: 0.2318 - dense_17_accuracy: 0.2188 - dense_19_accuracy: 0.2266 - val_los
s: 3.0688 - val_dense_20_loss: 1.9148 - val_dense_17_loss: 1.9197 - val_dense_19_loss: 1.9271 - val_dense_20_accura
cy: 0.2604 - val_dense_17_accuracy: 0.2604 - val_dense_19_accuracy: 0.2604
Epoch 20/20

6/6 [==============================] - 7s 1s/step - loss: 3.1085 - dense_20_loss: 1.9374 - dense_17_loss: 1.9522 -
dense_19_loss: 1.9514 - dense_20_accuracy: 0.2396 - dense_17_accuracy: 0.2448 - dense_19_accuracy: 0.2422 - val_los
s: 3.0718 - val_dense_20_loss: 1.9160 - val_dense_17_loss: 1.9246 - val_dense_19_loss: 1.9281 - val_dense_20_accura
cy: 0.2604 - val_dense_17_accuracy: 0.2604 - val_dense_19_accuracy: 0.2604
```

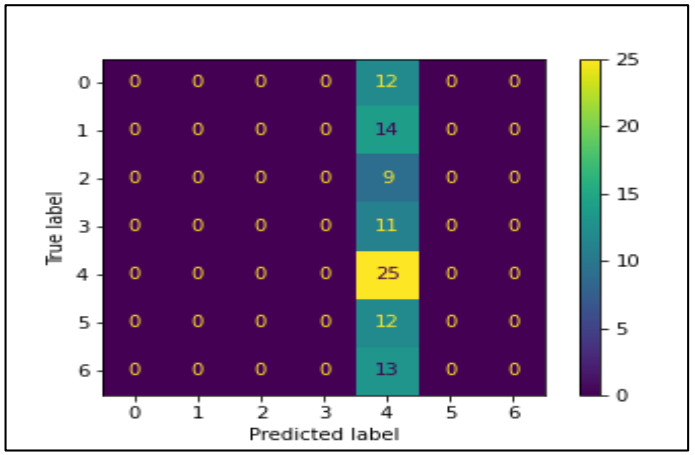
TRAINING AND LOSS CURVES

In [46]:
```
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.plot(history.history['dense_20_accuracy'])
plt.plot(history.history['val_dense_20_accuracy'])
plt.ylim(0, 1)
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='lower right')
plt.subplot(1,2,2)
plt.plot(history.history['dense_20_loss'])
plt.plot(history.history['val_dense_20_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='upper right')
plt.ylim([0,3])
plt.savefig("googlenet_loss_acc.png")
```



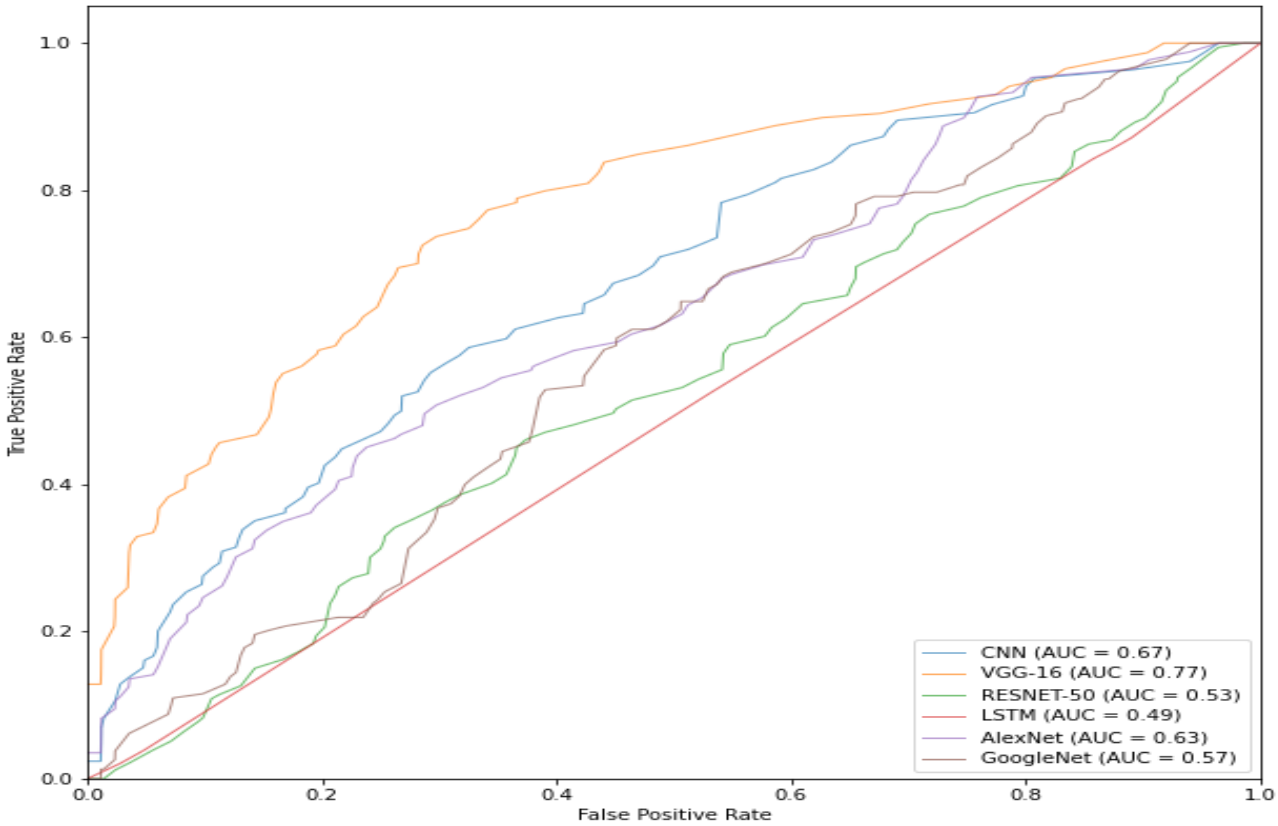
PERFORMANCE

```
Confusion Matrix:
[[ 0  0  0  0 12  0  0]
 [ 0  0  0  0 14  0  0]
 [ 0  0  0  0  9  0  0]
 [ 0  0  0  0 11  0  0]
 [ 0  0  0  0 25  0  0]
 [ 0  0  0  0 12  0  0]
 [ 0  0  0  0 13  0  0]]
------------------------------------------------------------
------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        12
           1       0.00      0.00      0.00        14
           2       0.00      0.00      0.00         9
           3       0.00      0.00      0.00        11
           4       0.26      1.00      0.41        25
           5       0.00      0.00      0.00        12
           6       0.00      0.00      0.00        13

    accuracy                           0.26        96
   macro avg       0.04      0.14      0.06        96
weighted avg       0.07      0.26      0.11        96

------------------------------------------------------------
------------------------------------------------------------
Accuracy Score:
0.2604166666666667
```



COMPARISON BETWEEN CNN, VGG-16, RESNET-50, LSTM(RNN), ALEXNET AND GOOGLENET.

ROC CURVES AND AUC COMPARISON



ROC CURVES AND AUC COMPARISON - CODE (I)

In [51]:

```
def plot_macro_avg_roc_curve(classifier, n_classes, name, X_test, y_test, single_output=True):

    y_test_binarized = label_binarize(y_test, classes=list(range(0,n_classes)))
    y_score = classifier.predict(X_test)

    #for cases where the model (like googlenet) has extra (auxillary) outputs
    if not single_output:
        y_score = y_score[0]

    # Compute ROC curve and ROC area for each class
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], y_score[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # First aggregate all false positive rates
    all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))

    # Then interpolate all ROC curves at this points
    mean_tpr = np.zeros_like(all_fpr)
    for i in range(n_classes):
        mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])

    # Finally average it and compute AUC
    mean_tpr /= n_classes

    fpr["macro"] = all_fpr
    tpr["macro"] = mean_tpr
    roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

    plt.plot(
        fpr["macro"],
        tpr["macro"],
        label="{0} (AUC = {1:0.2f})".format(name, roc_auc["macro"]),
        alpha=0.7,
        linewidth=1,
    )

    return plt
```

ROC CURVES AND AUC COMPARISON - CODE (II)

In [52]:
```
cnn = load_model('cnn.h5')
vgg_based_model = load_model('vgg16.h5')
resnet_based_model = load_model('resnet50.h5')
lstm_based_model = load_model('lstm.h5')
alexnet_model = load_model('alexnet.h5')
googlenet_model = load_model('googlenet.h5')
```

In [59]:
```
plt.figure(figsize=(10, 10))
plot_macro_avg_roc_curve(cnn, num_classes, "CNN", test_images, test_labels)
plot_macro_avg_roc_curve(vgg_based_model, num_classes, "VGG-16", test_images_p, test_labels)
plot_macro_avg_roc_curve(resnet_based_model, num_classes, "RESNET-50", test_images_pr, test_labels)
plot_macro_avg_roc_curve(lstm_based_model, num_classes, "LSTM", test_images, test_labels)
plot_macro_avg_roc_curve(alexnet_model, num_classes, "AlexNet", test_images, test_labels)
plot_macro_avg_roc_curve(googlenet_model, num_classes, "GoogleNet", test_images_normalized, test_labels, single_outp

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")

plt.savefig('roc_auc.png')
```

COMPARISON OF PERFORMANCE

CLASSIFIER	Accuracy	Precision	Recall	F1-Score	AUC
CNN	0.31	0.39	0.31	0.31	0.67
VGG-16	0.45	0.54	0.45	0.43	0.77
RESNET-50	0.16	0.04	0.16	0.06	0.53
LSTM	0.26	0.07	0.26	0.11	0.49
ALEXNET	0.22	0.17	0.22	0.15	0.63
GOOGLENET	0.26	0.07	0.26	0.11	0.57

CONCLUSION:

VGG-16 performed better than the other models. However, all the models performed poorly. It appears that it is hard to generalise patterns as all humans convey emotions differently.

EMODB-CLASSIFICATION

CONVERSION TO WAVEFORM (IMAGE)

IMPORTS AND CLASS NAMES

In [1]:
```
import matplotlib.pyplot as plt
import tensorflow as tf
from matplotlib.backend_bases import RendererBase
from scipy import signal
from scipy.io import wavfile
#import soundfile as sf
import os
import numpy as np
from PIL import Image
from scipy.fftpack import fft

%matplotlib inline
```

Some mappings

In [2]:
```
class_names = ['anger', 'disgust', 'fear', 'happiness', 'neutral', 'sadness', 'boredom']
lsymbols = ['a', 'd', 'f', 'h', 'n', 'sa', 'su']
full = {
    'W': 'anger',
    'E': 'disgust',
    'A': 'fear',
    'L': 'boredom',
    'F': 'happiness',
    'N': 'neutral',
    'T': 'sadness',
}
```

AUDIO PREPROCESSING

Audio Preprocessing (Conversion to images of waveform)

In [3]:
```
audio_path = './AudioData/'
pict_Path = './ImageData/'
samples = []
```

In [4]:
```
if not os.path.exists(pict_Path):
    os.makedirs(pict_Path)
```

In [5]:
```
for x in class_names:
    os.makedirs(pict_Path +'/'+ x)
```

In [6]:
```
def wav2img_waveform(wav_path, targetdir='', figsize=(4,4)):
    samplerate,test_sound  = wavfile.read(wav_path)
    fig = plt.figure(figsize=figsize)
    plt.plot(test_sound)
    plt.axis('off')
    output_file = wav_path.split('/')[-1].split('.wav')[0]
    output_file = targetdir +'/'+ output_file
    plt.savefig(f'{output_file}.png')
    plt.close()
```

In [7]:
```
all_files = [y for y in os.listdir(audio_path) if '.wav' in y]
for file in all_files:
    directory = full[file[5]]
    wav2img_waveform(audio_path + file, pict_Path + '/' + directory)
```

LOADING THE IMAGES INTO TRAIN AND TEST DATASETS

In [10]:
```
class_names = ['anger', 'disgust', 'fear', 'happiness', 'neutral', 'sadness', 'boredom']
img_height = 224
img_width = 224
num_train_images = 428
num_test_images = 107
num_classes = len(class_names)
```

In [11]:
```
train_ds = tf.keras.utils.image_dataset_from_directory(
  pict_Path,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=num_train_images,
  label_mode='int')
```

```
Found 535 files belonging to 7 classes.
Using 428 files for training.
```

In [12]:
```
test_ds = tf.keras.utils.image_dataset_from_directory(
  pict_Path,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=num_test_images,
  label_mode='int')
```

```
Found 535 files belonging to 7 classes.
Using 107 files for validation.
```

In [13]:
```
train_images = train_labels = test_images = test_labels = None
```

In [49]:
```
train_images, train_labels = next(train_ds.as_numpy_iterator())
```

In [15]:
```
test_images, test_labels = next(test_ds.as_numpy_iterator())
```

CNN

IMPORTS AND MODEL ARCHITECTURE

Convolutional Neural Network

Importing the libraries

In [16]:
```
from tensorflow import keras
from tensorflow.keras import layers, datasets
from tensorflow.keras.models import Sequential
import matplotlib.pyplot as plt
```

Building the CNN

In [17]:
```
cnn = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(filters=32, kernel_size=3, padding='same', activation='relu'),
  layers.Conv2D(filters=32, kernel_size=3, padding='same', activation='relu'),
  layers.MaxPooling2D(pool_size=2, strides=2),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(pool_size=2, strides=2),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes, activation="softmax")
])
```

COMPILATION AND SUMMARY

Compiling the CNN

In [18]:
```
cnn.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0005, epsilon=0.01),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
```

In [19]:
```
cnn.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 224, 224, 3)       0

 conv2d (Conv2D)             (None, 224, 224, 32)      896

 conv2d_1 (Conv2D)           (None, 224, 224, 32)      9248

 max_pooling2d (MaxPooling2D  (None, 112, 112, 32)     0
 )

 conv2d_2 (Conv2D)           (None, 112, 112, 32)      9248

 max_pooling2d_1 (MaxPooling  (None, 56, 56, 32)       0
 2D)

 dropout (Dropout)           (None, 56, 56, 32)        0

 flatten (Flatten)           (None, 100352)            0

 dense (Dense)               (None, 128)               12845184

 dense_1 (Dense)             (None, 7)                 903

=================================================================
Total params: 12,865,479
Trainable params: 12,865,479
Non-trainable params: 0
_________________________________________________________________
```

TRAINING AND SAVING

Training the CNN on the Training set and evaluating it on the Test set

In [20]:
```
history = cnn.fit(train_images, train_labels, epochs=20, validation_data=(test_images, test_labels))
```

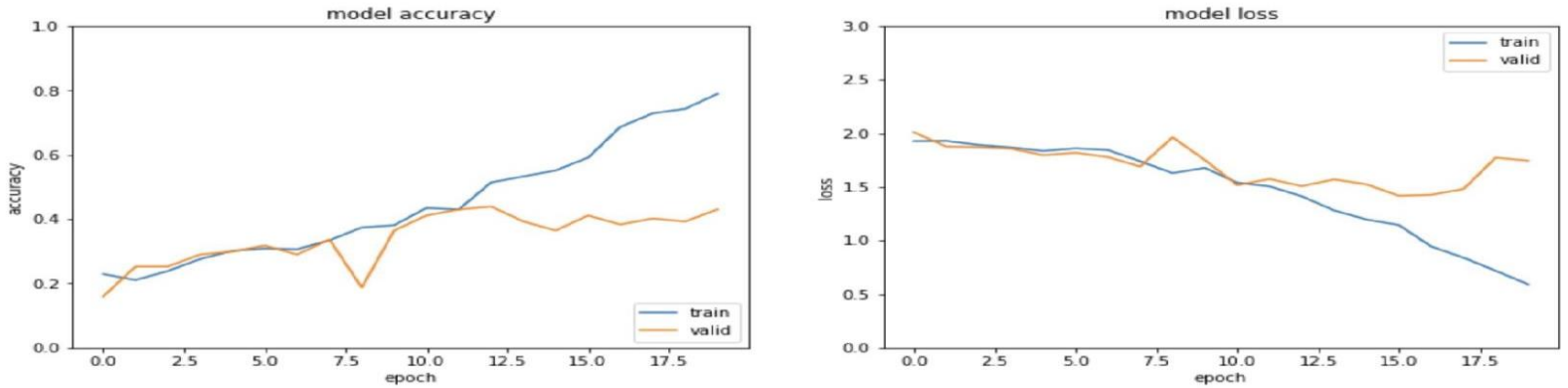
```
ccuracy: 0.3925
Epoch 15/20
14/14 [==============================] - 4s 254ms/step - loss: 1.1960 - accuracy: 0.5514 - val_loss: 1.5242 - val_a
ccuracy: 0.3645
Epoch 16/20
14/14 [==============================] - 4s 253ms/step - loss: 1.1445 - accuracy: 0.5911 - val_loss: 1.4174 - val_a
ccuracy: 0.4112
Epoch 17/20
14/14 [==============================] - 4s 253ms/step - loss: 0.9447 - accuracy: 0.6869 - val_loss: 1.4263 - val_a
ccuracy: 0.3832
Epoch 18/20
14/14 [==============================] - 4s 253ms/step - loss: 0.8399 - accuracy: 0.7290 - val_loss: 1.4798 - val_a
ccuracy: 0.4019
Epoch 19/20
14/14 [==============================] - 4s 254ms/step - loss: 0.7169 - accuracy: 0.7430 - val_loss: 1.7733 - val_a
ccuracy: 0.3925
Epoch 20/20
14/14 [==============================] - 4s 252ms/step - loss: 0.5905 - accuracy: 0.7897 - val_loss: 1.7445 - val_a
ccuracy: 0.4299
```

In [21]:
```
cnn.save('cnn.h5')
```

TRAINING AND LOSS CURVES

In [25]:
```
def save_train_loss_curves(history, filename):
    plt.figure(figsize=(15,5))
    plt.subplot(1,2,1)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.ylim(0, 1)
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'valid'], loc='lower right')
    plt.subplot(1,2,2)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'valid'], loc='upper right')
    plt.ylim([0,3])
    plt.savefig(filename)
```

In [26]:
```
save_train_loss_curves(history, 'cnn_loss_acc.png')
```



PERFORMANCE - CODE

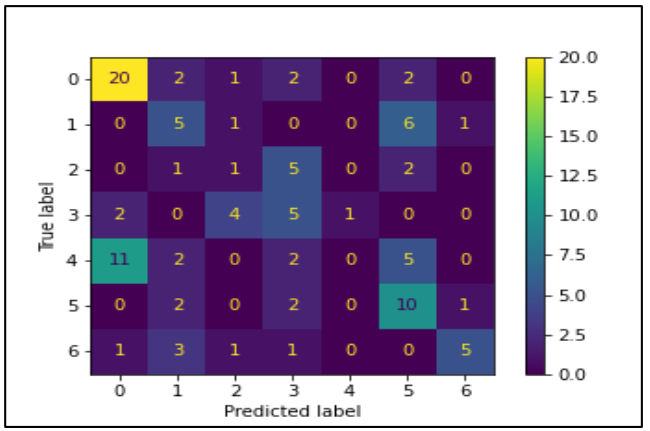
In [22]:
```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, ConfusionMatrixDisplay
```

In [23]:
```
def get_predictions(model, inputs):

    def index_max_arr_element(arr):
        best_index = -1
        best = float('-inf')
        for i, num in enumerate(arr):
            if num > best:
                best = num
                best_index = i
        return best_index

    pred_arr = model.predict(inputs)
    pred_labels = [index_max_arr_element(row) for row in pred_arr]
    return pred_labels
```

In [24]:
```
pred_labels = get_predictions(cnn, test_images)

print("Confusion Matrix:")
print(confusion_matrix(test_labels, pred_labels))
print("------------------------------------------------------------")
print("------------------------------------------------------------")
print("Performance Evaluation:")
print(classification_report(test_labels, pred_labels))
print("------------------------------------------------------------")
print("------------------------------------------------------------")
print("Accuracy Score:")
print(accuracy_score(test_labels, pred_labels))

plt.figure()
ConfusionMatrixDisplay.from_predictions(test_labels, pred_labels)
plt.savefig('./cnn_matrix.png')
```

PERFORMANCE

```
Confusion Matrix:
[[20  2  1  2  0  2  0]
 [ 0  5  1  0  0  6  1]
 [ 0  1  1  5  0  2  0]
 [ 2  0  4  5  1  0  0]
 [11  2  0  2  0  5  0]
 [ 0  2  0  2  0 10  1]
 [ 1  3  1  1  0  0  5]]
------------------------------------------------------------
------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           0       0.59      0.74      0.66        27
           1       0.33      0.38      0.36        13
           2       0.12      0.11      0.12         9
           3       0.29      0.42      0.34        12
           4       0.00      0.00      0.00        20
           5       0.40      0.67      0.50        15
           6       0.71      0.45      0.56        11

    accuracy                           0.43       107
   macro avg       0.35      0.40      0.36       107
weighted avg       0.36      0.43      0.38       107

------------------------------------------------------------
------------------------------------------------------------
Accuracy Score:
0.42990654205607476
```



VGG-16

IMPORTS, DATA PREPROCESSING & ARCHITECTURE

VGG-16

In [27]:
```
from tensorflow.keras.applications import vgg16
```

In [28]:
```
train_images_p, test_images_p = vgg16.preprocess_input(np.copy(train_images)), vgg16.preprocess_input(np.copy(test_i
```

In [29]:
```
vgg_based_model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  vgg16.VGG16(include_top=False, weights="imagenet", input_shape=(img_height, img_width, 3), pooling="max"),
  layers.Flatten(),
  layers.Dense(512, activation=('relu')),
  layers.Dropout(0.2),
  layers.Dense(128, activation=('relu')),
  layers.Dropout(0.2),
  layers.Dense(num_classes, activation="softmax")
])
```

COMPILATION AND SUMMARY

In [30]:
```
vgg_based_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0005, epsilon=0.01),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
```

In [31]:
```
vgg_based_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 224, 224, 3)       0

 vgg16 (Functional)          (None, 512)               14714688

 flatten_1 (Flatten)         (None, 512)               0

 dense_2 (Dense)             (None, 512)               262656

 dropout_1 (Dropout)         (None, 512)               0

 dense_3 (Dense)             (None, 128)               65664

 dropout_2 (Dropout)         (None, 128)               0

 dense_4 (Dense)             (None, 7)                 903

=================================================================
Total params: 15,043,911
Trainable params: 15,043,911
Non-trainable params: 0
_________________________________________________________________
```

TRAINING AND SAVING

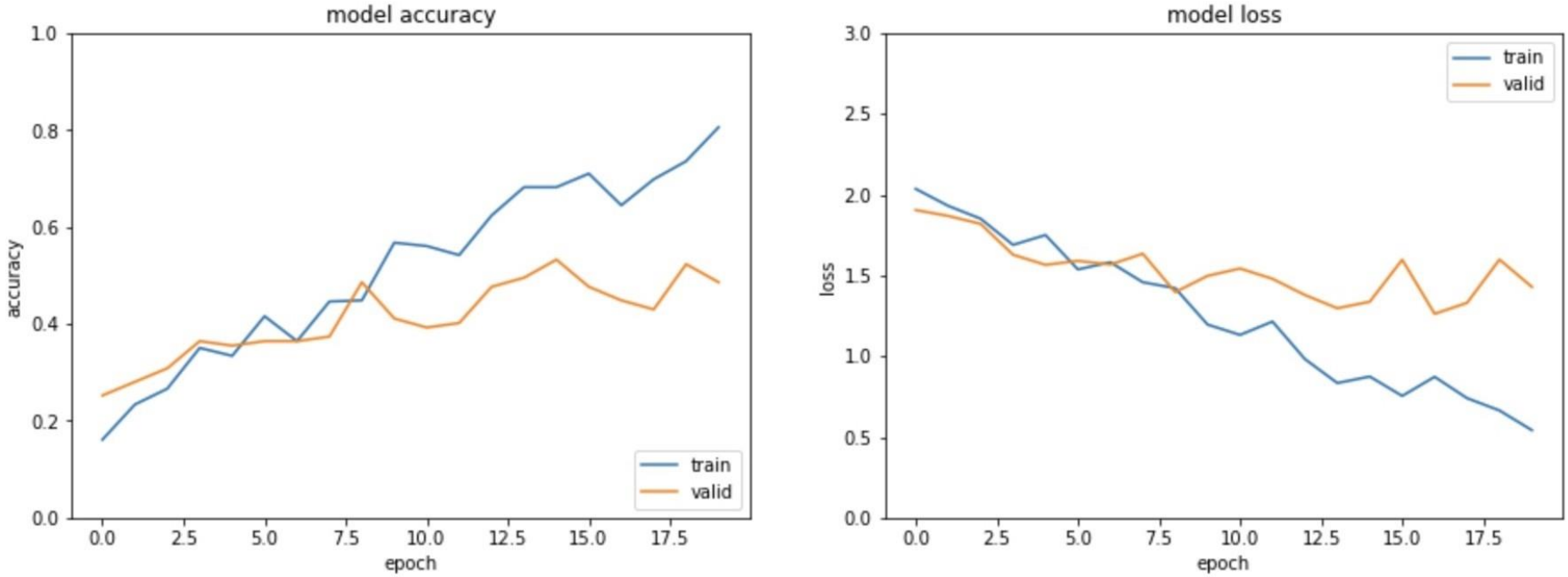
```
In [32]: history = vgg_based_model.fit(train_images_p, train_labels, epochs=20, validation_data=(test_images_p, test_labels))
```

```
uracy: 0.4953
Epoch 15/20
14/14 [==============================] - 39s 3s/step - loss: 0.8741 - accuracy: 0.6822 - val_loss: 1.3374 - val_acc
uracy: 0.5327
Epoch 16/20
14/14 [==============================] - 40s 3s/step - loss: 0.7542 - accuracy: 0.7103 - val_loss: 1.5978 - val_acc
uracy: 0.4766
Epoch 17/20
14/14 [==============================] - 41s 3s/step - loss: 0.8725 - accuracy: 0.6449 - val_loss: 1.2624 - val_acc
uracy: 0.4486
Epoch 18/20
14/14 [==============================] - 39s 3s/step - loss: 0.7401 - accuracy: 0.6986 - val_loss: 1.3304 - val_acc
uracy: 0.4299
Epoch 19/20
14/14 [==============================] - 39s 3s/step - loss: 0.6640 - accuracy: 0.7360 - val_loss: 1.5984 - val_acc
uracy: 0.5234
Epoch 20/20
14/14 [==============================] - 38s 3s/step - loss: 0.5418 - accuracy: 0.8061 - val_loss: 1.4294 - val_acc
uracy: 0.4860
```

```
In [33]: vgg_based_model.save('vgg16.h5')
```

TRAINING AND LOSS CURVES

```
In [34]: save_train_loss_curves(history, 'vgg_loss_acc.png')
```



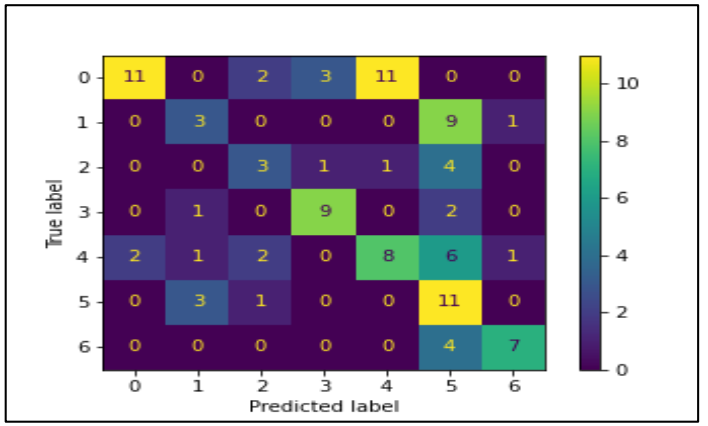
PERFORMANCE

```
Confusion Matrix:
[[11  0  2  3 11  0  0]
 [ 0  3  0  0  0  9  1]
 [ 0  0  3  1  1  4  0]
 [ 0  1  0  9  0  2  0]
 [ 2  1  2  0  8  6  1]
 [ 0  3  1  0  0 11  0]
 [ 0  0  0  0  0  4  7]]
```

Performance Evaluation:

	precision	recall	f1-score	support
0	0.85	0.41	0.55	27
1	0.38	0.23	0.29	13
2	0.38	0.33	0.35	9
3	0.69	0.75	0.72	12
4	0.40	0.40	0.40	20
5	0.31	0.73	0.43	15
6	0.78	0.64	0.70	11
accuracy			0.49	107
macro avg	0.54	0.50	0.49	107
weighted avg	0.57	0.49	0.49	107

Accuracy Score:
0.4859813084112149



RESNET-50

IMPORTS, DATA PREPROCESSING & ARCHITECTURE

In [36]:
```
from tensorflow.keras.applications import resnet50
```

In [37]:
```
train_images_pr, test_images_pr = resnet50.preprocess_input(np.copy(train_images)), resnet50.preprocess_input(np.cop
```

In [38]:
```
resnet_based_model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  resnet50.ResNet50(include_top=False, weights="imagenet", input_shape=(img_height, img_width, 3), pooling="max"),
  layers.Flatten(),
  layers.Dense(512, activation=('relu')),
  layers.Dropout(0.2),
  layers.Dense(128, activation=('relu')),
  layers.Dropout(0.2),
  layers.Dense(num_classes, activation="softmax")
])
```

COMPILATION AND SUMMARY

In [39]:
```
resnet_based_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0005, epsilon=0.01),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
```

In [40]:
```
resnet_based_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_2 (Rescaling)     (None, 224, 224, 3)       0

 resnet50 (Functional)       (None, 2048)              23587712

 flatten_2 (Flatten)         (None, 2048)              0

 dense_5 (Dense)             (None, 512)               1049088

 dropout_3 (Dropout)         (None, 512)               0

 dense_6 (Dense)             (None, 128)               65664

 dropout_4 (Dropout)         (None, 128)               0

 dense_7 (Dense)             (None, 7)                 903

=================================================================
Total params: 24,703,367
Trainable params: 24,650,247
Non-trainable params: 53,120
```

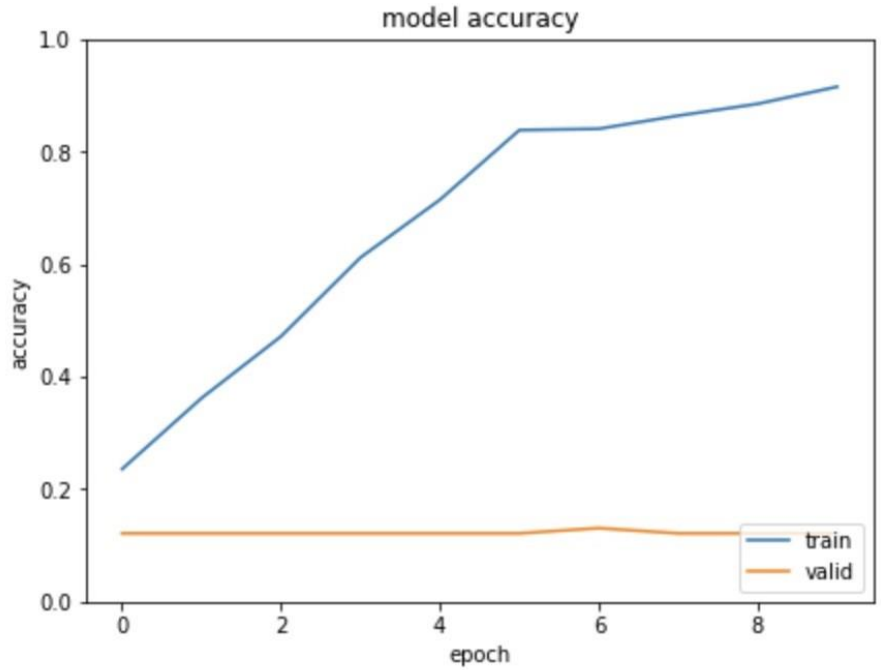
TRAINING AND SAVING

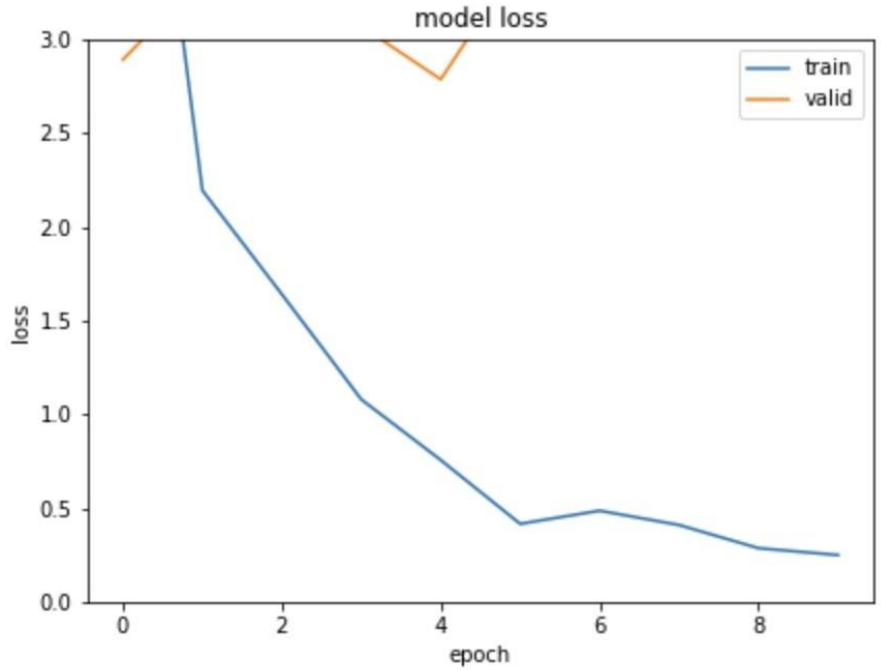
```
In [41]: history = resnet_based_model.fit(train_images_pr, train_labels, epochs=10, validation_data=(test_images_pr, test_lab
uracy: 0.1215
Epoch 5/10
14/14 [==============================] - 17s 1s/step - loss: 0.7559 - accuracy: 0.7150 - val_loss: 2.7873 - val_acc
uracy: 0.1215
Epoch 6/10
14/14 [==============================] - 17s 1s/step - loss: 0.4153 - accuracy: 0.8388 - val_loss: 3.4255 - val_acc
uracy: 0.1215
Epoch 7/10
14/14 [==============================] - 17s 1s/step - loss: 0.4854 - accuracy: 0.8411 - val_loss: 3.2850 - val_acc
uracy: 0.1308
Epoch 8/10
14/14 [==============================] - 17s 1s/step - loss: 0.4092 - accuracy: 0.8645 - val_loss: 4.2360 - val_acc
uracy: 0.1215
Epoch 9/10
14/14 [==============================] - 17s 1s/step - loss: 0.2854 - accuracy: 0.8855 - val_loss: 3.5137 - val_acc
uracy: 0.1215
Epoch 10/10
14/14 [==============================] - 17s 1s/step - loss: 0.2489 - accuracy: 0.9159 - val_loss: 4.0176 - val_acc
uracy: 0.1215
```

```
In [42]: resnet_based_model.save('resnet50.h5')
```

TRAINING AND LOSS CURVES

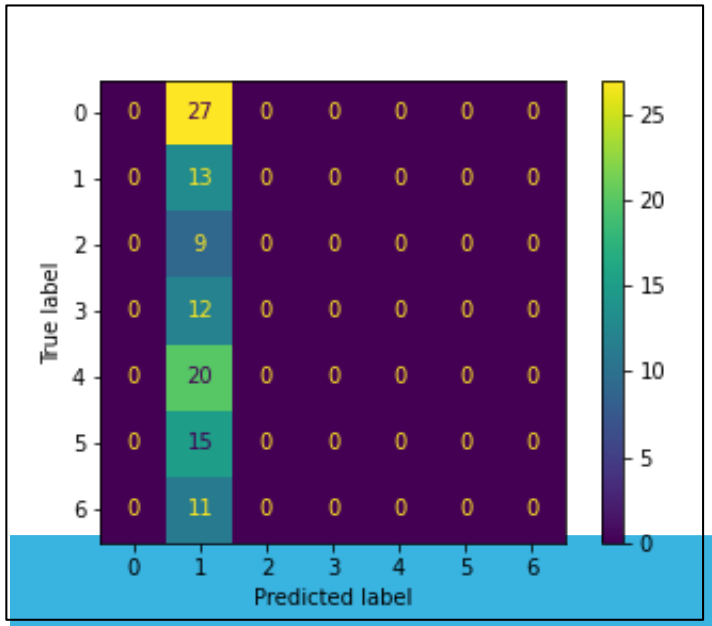
```
[n [43]: save_train_loss_curves(history, 'resnet_loss_acc.png')
```





PERFORMANCE

```
Confusion Matrix:
[[ 0 27  0  0  0  0  0]
 [ 0 13  0  0  0  0  0]
 [ 0  9  0  0  0  0  0]
 [ 0 12  0  0  0  0  0]
 [ 0 20  0  0  0  0  0]
 [ 0 15  0  0  0  0  0]
 [ 0 11  0  0  0  0  0]]
------------------------------------------------------------
------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        27
           1       0.12      1.00      0.22        13
           2       0.00      0.00      0.00         9
           3       0.00      0.00      0.00        12
           4       0.00      0.00      0.00        20
           5       0.00      0.00      0.00        15
           6       0.00      0.00      0.00        11

    accuracy                           0.12       107
   macro avg       0.02      0.14      0.03       107
weighted avg       0.01      0.12      0.03       107

------------------------------------------------------------
------------------------------------------------------------
Accuracy Score:
0.12149532710280374
```



RNN (LSTM)

MODEL ARCHITECTURE AND COMPILATION

RNN (LSTM based)

In [45]:
```
lstm_based_model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Resizing(64, 64),
  layers.Reshape((4096, 3)),
  layers.LSTM(units=50, return_sequences=True),
  layers.Dropout(0.2),
  layers.LSTM(units=50),
  layers.Dropout(0.2),
  layers.Dense(128, activation=('relu')),
  layers.Dropout(0.2),
  layers.Dense(num_classes, activation="softmax")
])
```

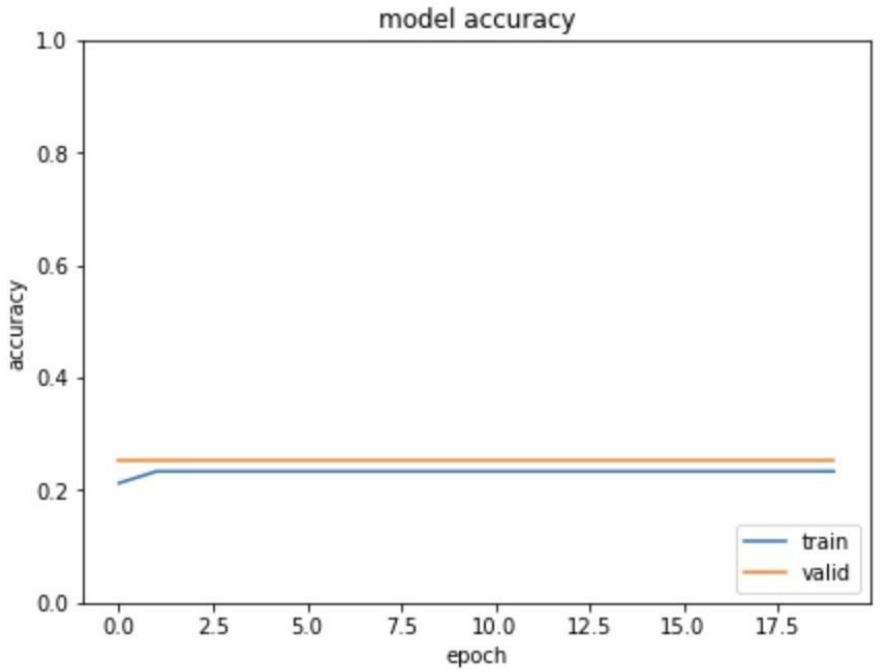
In [46]:
```
lstm_based_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
```

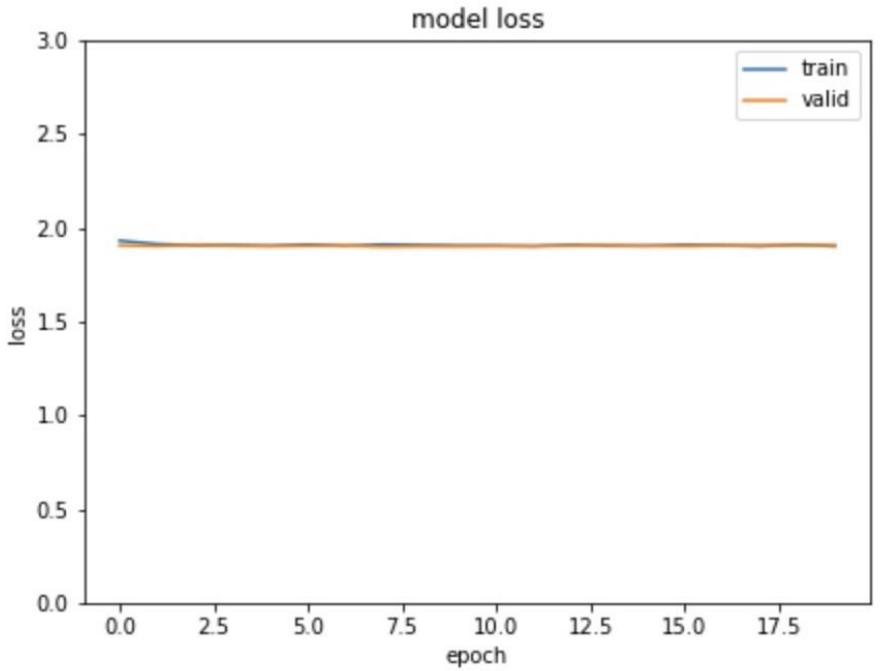
TRAINING AND SAVING

```
In [48]: history = lstm_based_model.fit(train_images, train_labels, epochs=20, validation_data=(test_images, test_labels))
uracy: 0.2523
Epoch 15/20
14/14 [==============================] - 52s 4s/step - loss: 1.9035 - accuracy: 0.2336 - val_loss: 1.9032 - val_acc
uracy: 0.2523
Epoch 16/20
14/14 [==============================] - 53s 4s/step - loss: 1.9087 - accuracy: 0.2336 - val_loss: 1.9022 - val_acc
uracy: 0.2523
Epoch 17/20
14/14 [==============================] - 52s 4s/step - loss: 1.9065 - accuracy: 0.2336 - val_loss: 1.9051 - val_acc
uracy: 0.2523
Epoch 18/20
14/14 [==============================] - 52s 4s/step - loss: 1.9017 - accuracy: 0.2336 - val_loss: 1.9059 - val_acc
uracy: 0.2523
Epoch 19/20
14/14 [==============================] - 52s 4s/step - loss: 1.9107 - accuracy: 0.2336 - val_loss: 1.9067 - val_acc
uracy: 0.2523
Epoch 20/20
14/14 [==============================] - 53s 4s/step - loss: 1.9017 - accuracy: 0.2336 - val_loss: 1.9070 - val_acc
uracy: 0.2523

In [50]: lstm_based_model.save('lstm.h5')
```

TRAINING AND LOSS CURVES

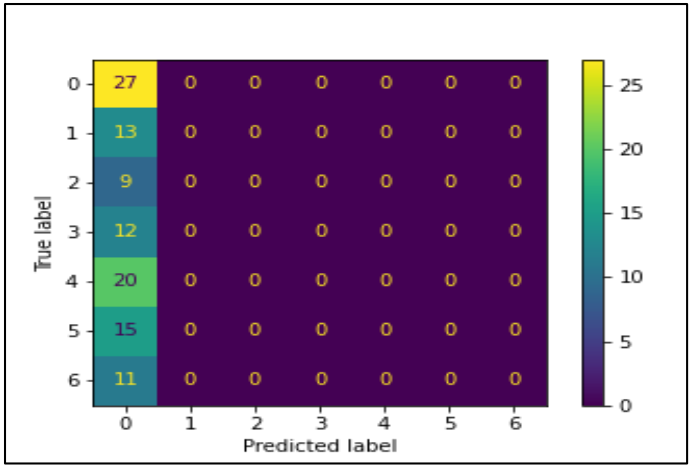
```
[51]: save_train_loss_curves(history, 'lstm_loss_acc.png')
```





PERFORMANCE

```
Confusion Matrix:
[[27  0  0  0  0  0  0]
 [13  0  0  0  0  0  0]
 [ 9  0  0  0  0  0  0]
 [12  0  0  0  0  0  0]
 [20  0  0  0  0  0  0]
 [15  0  0  0  0  0  0]
 [11  0  0  0  0  0  0]]
------------------------------------------------------------
------------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           0       0.25      1.00      0.40        27
           1       0.00      0.00      0.00        13
           2       0.00      0.00      0.00         9
           3       0.00      0.00      0.00        12
           4       0.00      0.00      0.00        20
           5       0.00      0.00      0.00        15
           6       0.00      0.00      0.00        11

    accuracy                           0.25       107
   macro avg       0.04      0.14      0.06       107
weighted avg       0.06      0.25      0.10       107

------------------------------------------------------------
------------------------------------------------------------
Accuracy Score:
0.2523364485981308
```



ALEXNET

MODEL ARCHITECTURE - I

ALEXNET

In [52]:
```
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D, ZeroPadding2D, BatchN
from tensorflow.keras.regularizers import l2
```

In [53]:
```
def alexnet_model(img_shape=(32, 32, 3), n_classes=10, l2_reg=0.,
    weights=None):

    # Initialize model
    alexnet = Sequential()
    alexnet.add(layers.Input(shape=img_shape))
    alexnet.add(layers.Resizing(64, 64))

    # Layer 1
    alexnet.add(Conv2D(96, (11, 11), padding='same', kernel_regularizer=l2(l2_reg)))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(MaxPooling2D(pool_size=(2, 2)))

    # Layer 2
    alexnet.add(Conv2D(256, (5, 5), padding='same'))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(MaxPooling2D(pool_size=(2, 2)))

    # Layer 3
    alexnet.add(ZeroPadding2D((1, 1)))
    alexnet.add(Conv2D(512, (3, 3), padding='same'))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(MaxPooling2D(pool_size=(2, 2)))

    # Layer 4
    alexnet.add(ZeroPadding2D((1, 1)))
    alexnet.add(Conv2D(1024, (3, 3), padding='same'))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
```

MODEL ARCHITECTURE - II

```
    # Layer 5
    alexnet.add(ZeroPadding2D((1, 1)))
    alexnet.add(Conv2D(1024, (3, 3), padding='same'))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(MaxPooling2D(pool_size=(2, 2)))

    # Layer 6
    alexnet.add(Flatten())
    alexnet.add(Dense(3072))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(Dropout(0.5))

    # Layer 7
    alexnet.add(Dense(4096))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('relu'))
    alexnet.add(Dropout(0.5))

    # Layer 8
    alexnet.add(Dense(n_classes))
    alexnet.add(BatchNormalization())
    alexnet.add(Activation('softmax'))

    if weights is not None:
        alexnet.load_weights(weights)

    return alexnet
```

[54]:
```
alexnet_model = alexnet_model(img_shape=(img_height, img_width, 3), n_classes=num_classes)
```

[55]:
```
alexnet_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0002, epsilon=0.05),
                      loss=tf.keras.losses.SparseCategoricalCrossentropy(),
                      metrics=['accuracy'])
```

TRAINING AND SAVING

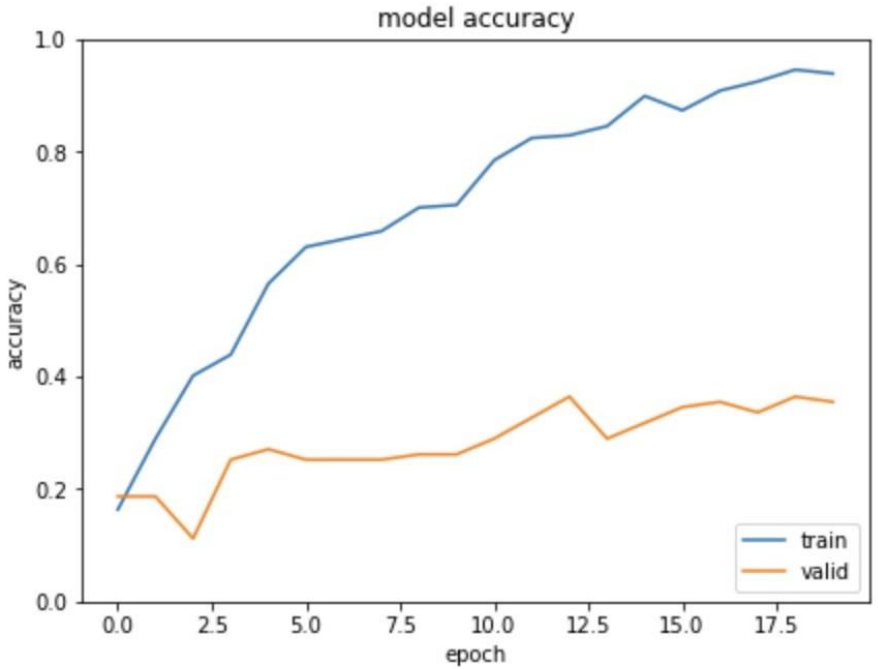
```
In [57]: history = alexnet_model.fit(train_images, train_labels, epochs=20, validation_data=(test_images, test_labels))
```

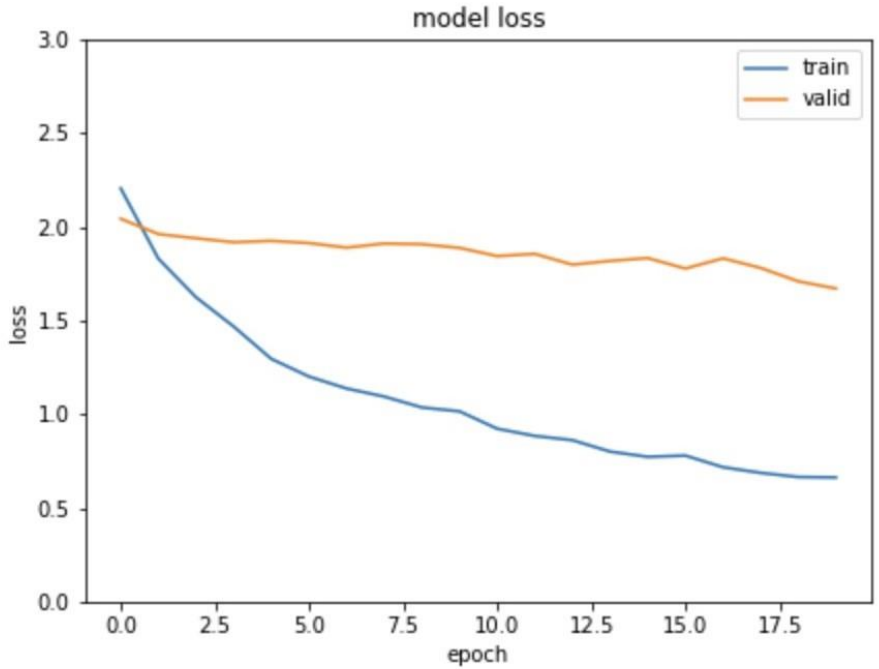
```
accuracy: 0.2897
Epoch 15/20
14/14 [==============================] - 12s 840ms/step - loss: 0.7722 - accuracy: 0.8995 - val_loss: 1.8331 - val_
accuracy: 0.3178
Epoch 16/20
14/14 [==============================] - 12s 871ms/step - loss: 0.7789 - accuracy: 0.8738 - val_loss: 1.7774 - val_
accuracy: 0.3458
Epoch 17/20
14/14 [==============================] - 12s 865ms/step - loss: 0.7169 - accuracy: 0.9089 - val_loss: 1.8318 - val_
accuracy: 0.3551
Epoch 18/20
14/14 [==============================] - 14s 1s/step - loss: 0.6867 - accuracy: 0.9252 - val_loss: 1.7806 - val_acc
uracy: 0.3364
Epoch 19/20
14/14 [==============================] - 14s 1s/step - loss: 0.6647 - accuracy: 0.9463 - val_loss: 1.7089 - val_acc
uracy: 0.3645
Epoch 20/20
14/14 [==============================] - 13s 959ms/step - loss: 0.6623 - accuracy: 0.9393 - val_loss: 1.6708 - val_
accuracy: 0.3551
```

```
In [59]: alexnet_model.save('alexnet.h5')
```

TRAINING AND LOSS CURVES

```
In [60]: save_train_loss_curves(history, 'alexnet_loss_acc.png')
```





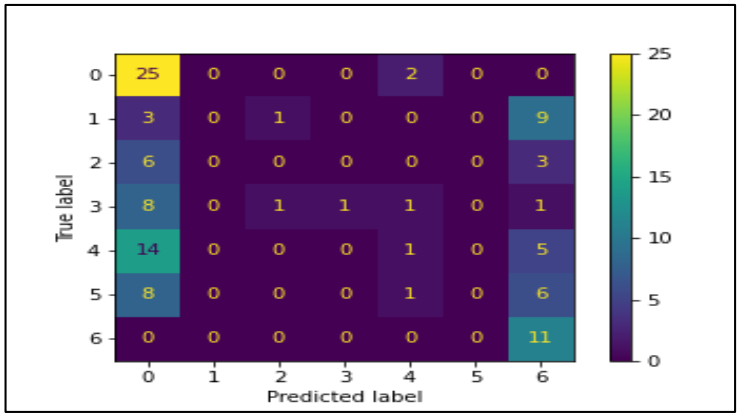
PERFORMANCE

```
Confusion Matrix:
[[25  0  0  0  2  0  0]
 [ 3  0  1  0  0  0  9]
 [ 6  0  0  0  0  0  3]
 [ 8  0  1  1  1  0  1]
 [14  0  0  0  1  0  5]
 [ 8  0  0  0  1  0  6]
 [ 0  0  0  0  0  0 11]]
```

Performance Evaluation:

	precision	recall	f1-score	support
0	0.39	0.93	0.55	27
1	0.00	0.00	0.00	13
2	0.00	0.00	0.00	9
3	1.00	0.08	0.15	12
4	0.20	0.05	0.08	20
5	0.00	0.00	0.00	15
6	0.31	1.00	0.48	11
accuracy			0.36	107
macro avg	0.27	0.29	0.18	107
weighted avg	0.28	0.36	0.22	107

Accuracy Score:
0.35514018691588783



GOOGLENET

IMPORTS, DATA PREPROCESSING AND INCEPTION MODULE

GOOGLENET

In [63]:
```
from tensorflow.keras.layers import Flatten, Input, Conv2D, Dense, Dropout, MaxPooling2D, AveragePooling2D, GlobalAv
from tensorflow.keras.layers import concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.regularizers import l2
```

In [64]:
```
train_images_normalized = train_images/255.0
test_images_normalized = test_images/255.0
```

In [65]:
```
def inception(x,
              filters_1x1,
              filters_3x3_reduce,
              filters_3x3,
              filters_5x5_reduce,
              filters_5x5,
              filters_pool):
    path1 = layers.Conv2D(filters_1x1, (1, 1), padding='same',    activation='relu')(x)
    path2 = layers.Conv2D(filters_3x3_reduce, (1, 1), padding='same', activation='relu')(x)
    path2 = layers.Conv2D(filters_3x3, (1, 1), padding='same', activation='relu')(path2)
    path3 = layers.Conv2D(filters_5x5_reduce, (1, 1), padding='same', activation='relu')(x)
    path3 = layers.Conv2D(filters_5x5, (1, 1), padding='same', activation='relu')(path3)
    path4 = layers.MaxPool2D((3, 3), strides=(1, 1), padding='same')(x)
    path4 = layers.Conv2D(filters_pool, (1, 1), padding='same', activation='relu')(path4)
    return tf.concat([path1, path2, path3, path4], axis=3)
```

MODEL ARCHITECTURE

In [66]:
```
inp = layers.Input(shape=(224, 224, 3))
x = layers.Conv2D(64, 7, strides=2, padding='same', activation='relu')(inp)
x = layers.MaxPooling2D(3, strides=2)(x)
x = layers.Conv2D(64, 1, strides=1, padding='same', activation='relu')(x)
x = layers.Conv2D(192, 3, strides=1, padding='same', activation='relu')(x)
x = layers.MaxPooling2D(3, strides=2)(x)
x = inception(x, filters_1x1=64, filters_3x3_reduce=96, filters_3x3=128, filters_5x5_reduce=16, filters_5x5=32, filt
x = inception(x, filters_1x1=128, filters_3x3_reduce=128, filters_3x3=192, filters_5x5_reduce=32, filters_5x5=96, fi
x = layers.MaxPooling2D(3, strides=2)(x)
x = inception(x, filters_1x1=192, filters_3x3_reduce=96, filters_3x3=208, filters_5x5_reduce=16, filters_5x5=48, fil
aux1 = layers.AveragePooling2D((5, 5), strides=3)(x)
aux1 =layers.Conv2D(128, 1, padding='same', activation='relu')(aux1)
aux1 = layers.Flatten()(aux1)
aux1 = layers.Dense(1024, activation='relu')(aux1)
aux1 = layers.Dropout(0.7)(aux1)
aux1 = layers.Dense(10, activation='softmax')(aux1)
x = inception(x, filters_1x1=160, filters_3x3_reduce=112, filters_3x3=224, filters_5x5_reduce=24, filters_5x5=64, fi
x = inception(x, filters_1x1=128, filters_3x3_reduce=128, filters_3x3=256, filters_5x5_reduce=24, filters_5x5=64, fi
x = inception(x, filters_1x1=112, filters_3x3_reduce=144, filters_3x3=288, filters_5x5_reduce=32, filters_5x5=64, fi
aux2 = layers.AveragePooling2D((5, 5), strides=3)(x)
aux2 =layers.Conv2D(128, 1, padding='same', activation='relu')(aux2)
aux2 = layers.Flatten()(aux2)
aux2 = layers.Dense(1024, activation='relu')(aux2)
aux2 = layers.Dropout(0.7)(aux2)
aux2 = layers.Dense(10, activation='softmax')(aux2)
x = inception(x, filters_1x1=256, filters_3x3_reduce=160, filters_3x3=320, filters_5x5_reduce=32, filters_5x5=128, f
x = layers.MaxPooling2D(3, strides=2)(x)
x = inception(x, filters_1x1=256, filters_3x3_reduce=160, filters_3x3=320, filters_5x5_reduce=32, filters_5x5=128, f
x = inception(x, filters_1x1=384, filters_3x3_reduce=192, filters_3x3=384, filters_5x5_reduce=48, filters_5x5=128, f
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dropout(0.4)(x)
out = layers.Dense(10, activation='softmax')(x)
```

In [67]:
```
googlenet_model = Model(inputs = inp, outputs = [out, aux1, aux2])
```

COMPILATION

In [67]:
```
googlenet_model = Model(inputs = inp, outputs = [out, aux1, aux2])
```

In [68]:
```
googlenet_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.01, epsilon=0.05),
              loss=[keras.losses.sparse_categorical_crossentropy,
                    keras.losses.sparse_categorical_crossentropy,
                    keras.losses.sparse_categorical_crossentropy],
              loss_weights=[1, 0.3, 0.3],
              metrics=['accuracy'])
```

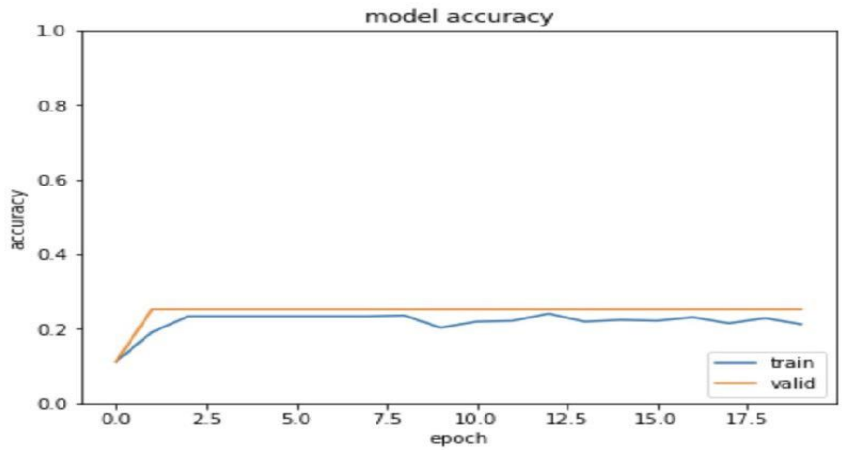
TRAINING

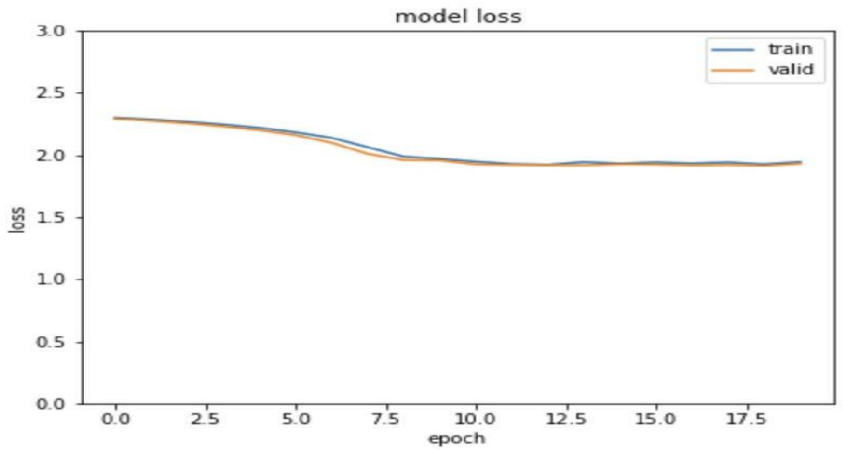
```
In [69]: history = googlenet_model.fit(train_images_normalized, [train_labels, train_labels, train_labels], \
                     validation_data=(test_images_normalized, [test_labels, test_labels, test_labels]), \
                     batch_size=64, epochs=20)
```

```
7/7 [==============================] - 10s 1s/step - loss: 3.1050 - dense_17_loss: 1.9521 - dense_14_loss: 1.9469
dense_16_loss: 1.9627 - dense_17_accuracy: 0.2313 - dense_14_accuracy: 0.2430 - dense_16_accuracy: 0.2360 - val_lo
s: 3.0746 - val_dense_17_loss: 1.9167 - val_dense_14_loss: 1.9268 - val_dense_16_loss: 1.9331 - val_dense_17_accur
cy: 0.2523 - val_dense_14_accuracy: 0.2523 - val_dense_16_accuracy: 0.2523
Epoch 18/20
7/7 [==============================] - 10s 1s/step - loss: 3.1114 - dense_17_loss: 1.9405 - dense_14_loss: 1.9557
dense_16_loss: 1.9471 - dense_17_accuracy: 0.2150 - dense_14_accuracy: 0.2430 - dense_16_accuracy: 0.2407 - val_lo
s: 3.0775 - val_dense_17_loss: 1.9196 - val_dense_14_loss: 1.9239 - val_dense_16_loss: 1.9359 - val_dense_17_accur
cy: 0.2523 - val_dense_14_accuracy: 0.2523 - val_dense_16_accuracy: 0.2523
Epoch 19/20
7/7 [==============================] - 10s 1s/step - loss: 3.1000 - dense_17_loss: 1.9256 - dense_14_loss: 1.9555
dense_16_loss: 1.9588 - dense_17_accuracy: 0.2290 - dense_14_accuracy: 0.2103 - dense_16_accuracy: 0.2033 - val_lo
s: 3.0635 - val_dense_17_loss: 1.9141 - val_dense_14_loss: 1.9109 - val_dense_16_loss: 1.9202 - val_dense_17_accur
cy: 0.2523 - val_dense_14_accuracy: 0.2523 - val_dense_16_accuracy: 0.2523
Epoch 20/20

7/7 [==============================] - 11s 2s/step - loss: 3.1215 - dense_17_loss: 1.9437 - dense_14_loss: 1.9741
dense_16_loss: 1.9520 - dense_17_accuracy: 0.2126 - dense_14_accuracy: 0.2196 - dense_16_accuracy: 0.2103 - val_lo
s: 3.0879 - val_dense_17_loss: 1.9301 - val_dense_14_loss: 1.9298 - val_dense_16_loss: 1.9295 - val_dense_17_accur
cy: 0.2523 - val_dense_14_accuracy: 0.2523 - val_dense_16_accuracy: 0.2523
```

TRAINING AND LOSS CURVES

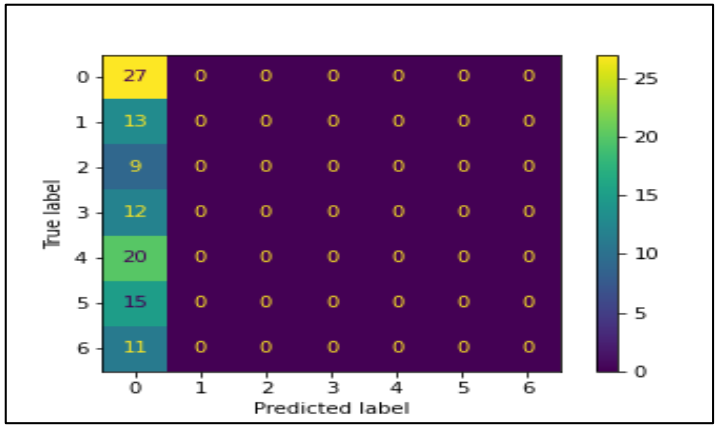
In [72]:
```
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.plot(history.history['dense_17_accuracy'])
plt.plot(history.history['val_dense_17_accuracy'])
plt.ylim(0, 1)
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='lower right')
plt.subplot(1,2,2)
plt.plot(history.history['dense_17_loss'])
plt.plot(history.history['val_dense_17_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='upper right')
plt.ylim([0,3])
plt.savefig("googlenet_loss_acc.png")
```





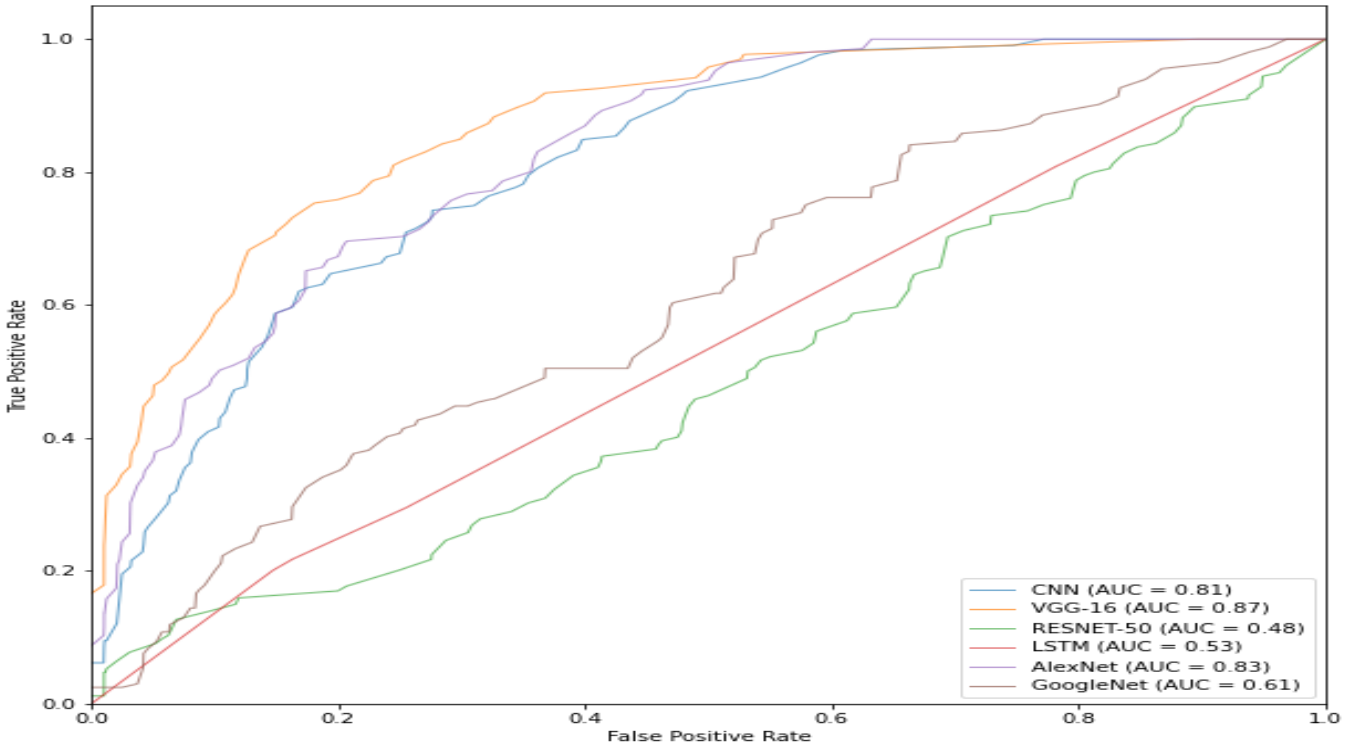
PERFORMANCE

```
Confusion Matrix:
[[27  0  0  0  0  0  0]
 [13  0  0  0  0  0  0]
 [ 9  0  0  0  0  0  0]
 [12  0  0  0  0  0  0]
 [20  0  0  0  0  0  0]
 [15  0  0  0  0  0  0]
 [11  0  0  0  0  0  0]]
--------------------------------------------------------
--------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           0       0.25      1.00      0.40        27
           1       0.00      0.00      0.00        13
           2       0.00      0.00      0.00         9
           3       0.00      0.00      0.00        12
           4       0.00      0.00      0.00        20
           5       0.00      0.00      0.00        15
           6       0.00      0.00      0.00        11

    accuracy                           0.25       107
   macro avg       0.04      0.14      0.06       107
weighted avg       0.06      0.25      0.10       107

--------------------------------------------------------
--------------------------------------------------------
Accuracy Score:
0.2523364485981308
```



COMPARISON BETWEEN CNN, VGG-16, RESNET-50, LSTM(RNN), ALEXNET AND GOOGLENET.

ROC CURVES AND AUC COMPARISON



ROC CURVES AND AUC COMPARISON - CODE (I)

In [77]:

```
def plot_macro_avg_roc_curve(classifier, n_classes, name, X_test, y_test, single_output=True):

    y_test_binarized = label_binarize(y_test, classes=list(range(0,n_classes)))
    y_score = classifier.predict(X_test)

    #for cases where the model (like googlenet) has extra (auxillary) outputs
    if not single_output:
        y_score = y_score[0]

    # Compute ROC curve and ROC area for each class
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], y_score[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # First aggregate all false positive rates
    all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))

    # Then interpolate all ROC curves at this points
    mean_tpr = np.zeros_like(all_fpr)
    for i in range(n_classes):
        mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])

    # Finally average it and compute AUC
    mean_tpr /= n_classes

    fpr["macro"] = all_fpr
    tpr["macro"] = mean_tpr
    roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

    plt.plot(
        fpr["macro"],
        tpr["macro"],
        label="{0} (AUC = {1:0.2f})".format(name, roc_auc["macro"]),
        alpha=0.7,
        linewidth=1,
    )

    return plt
```

ROC CURVES AND AUC COMPARISON - CODE (II)

In [78]:
```
cnn = load_model('cnn.h5')
vgg_based_model = load_model('vgg16.h5')
resnet_based_model = load_model('resnet50.h5')
lstm_based_model = load_model('lstm.h5')
alexnet_model = load_model('alexnet.h5')
googlenet_model = load_model('googlenet.h5')
```

In [79]:
```
plt.figure(figsize=(10, 10))
plot_macro_avg_roc_curve(cnn, num_classes, "CNN", test_images, test_labels)
plot_macro_avg_roc_curve(vgg_based_model, num_classes, "VGG-16", test_images_p, test_labels)
plot_macro_avg_roc_curve(resnet_based_model, num_classes, "RESNET-50", test_images_pr, test_labels)
plot_macro_avg_roc_curve(lstm_based_model, num_classes, "LSTM", test_images, test_labels)
plot_macro_avg_roc_curve(alexnet_model, num_classes, "AlexNet", test_images, test_labels)
plot_macro_avg_roc_curve(googlenet_model, num_classes, "GoogleNet", test_images_normalized, test_labels, single_outp

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend(loc="lower right")

plt.savefig('roc_auc.png')
```

COMPARISON OF PERFORMANCE

CLASSIFIER	Accuracy	Precision	Recall	F1-Score	AUC
CNN	0.43	0.68	0.43	0.38	0.81
VGG-16	0.49	0.57	0.49	0.49	0.87
RESNET-50	0.12	0.01	0.12	0.03	0.48
LSTM	0.25	0.06	0.25	0.10	0.53
ALEXNET	0.36	0.28	0.36	0.22	0.83
GOOGLENET	0.25	0.06	0.25	0.10	0.61

CONCLUSION:

VGG-16 performed better than the other models. AlexNet and the simple CNN did respectable jobs. However, all the models performed poorly. It appears that it is hard to generalise patterns as all humans convey emotions differently.