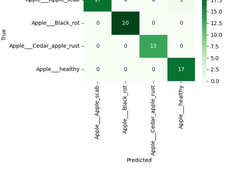
```
[1]: %%
%%capture
%%bash
pip install -q -U
pip install -q -U keras-nlp

from tensorflow.keras.models import TransformerEncoder
from keras.models import Model
from tensorflow.keras.layers import Input
from keras.layers import Dense, Flatten, Dropout
from keras.layers import LayerNormalization, MultiHeadAttention
from keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint, LearningRate
from keras.utils import plot_model
from keras.models import load_model
from keras.utils import to_categorical
from keras.optimizers.legacy import Adam, SGD, RMSprop, Adagrad, Adadelta, Adamax, Nadam, Ftrl
from keras.metrics import CategoricalAccuracy, SparseCategoricalAccuracy, TopKCategoricalAccuracy, CategoricalCrossentropy, SparseCategoricalCrossentropy, KLDivergence, CosineSimilarity, MeanAbsolutePercentageError, MeanAbsoluteError, MeanSquaredError, RootMeanSquaredError, LogCoshError, Precision, Recall, AUC
from keras.models import Sequential
from keras.layers import Embedding, GlobalAveragePooling1D, MaxPooling1D, Conv1D, LSTM, GRU, SimpleRNN, Bidirectional, TimeDistributed
from keras.preferences import keras_preferences
from keras.backend.common.keras_export import keras_export
from keras.saving.serialization_lib import SerializationLibrary
from keras.saving.object_registration import CustomObjectScope
from keras.saving.saving_lib import save_weights_only, load_weights_only
from keras.utils import get_custom_objects
from keras.saving.serialization_lib import deserialize_keras_object
from keras.src.layers.activations.softmax import Softmax
from keras.src.layers.activations.relu import ReLU
from keras.src.layers.activations.leaky_relu import LeakyReLU
from keras.src.layers.activations.prelu import PReLU
from keras.src.layers.activations.elu import ELU
from keras.src.layers.activations.thresholded_relu import ThresholdedReLU
from keras.src.layers.activations.exponential import Exponential
from keras.src.layers.activations.gelu import Gelu
from keras.src.layers.activations.hard_sigmoid import HardSigmoid
from keras.src.layers.activations.hard_silu import HardSilu
from keras.src.layers.activations.hard_tanh import HardTanh
from keras.src.layers.activations.linear import Linear
from keras.src.layers.activations.mish import Mish
from keras.src.layers.activations.log_softmax import LogSoftmax
from keras.src.layers.activations.sparsemax import Sparsemax
from keras.src.layers.activations.squeeze_excitation import SqueezeExcitation
from keras.src.layers.activations.squeeze_excitation import SqueezeExcitation1D
from keras.src.layers.activations.squeeze_excitation import SqueezeExcitation2D
from keras.src.layers.activations.squeeze_excitation import SqueezeExcitation3D
from keras.src.layers.activations.squeeze_excitation import SqueezeExcitation4D
from keras.src.layers.activations.squeeze_excitation import SqueezeExcitation5D
from keras.src.layers.activations.squeeze_excitation import SqueezeExcitation6D
from keras.src.layers.activations.squeeze_excitation import SqueezeExcitation7D
from keras.src.layers.activations.squeeze_excitation import SqueezeExcitation8D
from keras.src.layers.activations.squeeze_excitation import SqueezeExcitation9D
from keras.src.layers.activations.squeeze_excitation import SqueezeExcitation10D
from keras.src.layers.activations.squeeze_excitation import
```



In [ ]:

In [ ]:

```
## InceptionV3
```

In [52]:

```
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.applications.inception_v3 import preprocess_input as inception_preprocess
```

In [54]:

```
image_size = (224, 224)
num_classes = len(apple_classes)

# ~~~~~ Image Branch (InceptionV3) ~~~~~
image_input_i = layers.Input(shape=(image_size[0], image_size[1], 3), name="image")
x = inception_preprocess(image_input_i)

base_inception = InceptionV3(
    include_top=False,
    weights="imagenet",
    input_shape=(image_size[0], image_size[1], 3)
)

# Freeze backbone (avoid overfitting on small dataset)
base_inception.trainable = False

x = base_inception(x, training=False)
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dropout(0.3)(x)
image_features_i = layers.Dense(128, activation="relu", name="image_features_inception")(x)

# ~~~~~ Tabular Branch (MLP) ~~~~~
tab_input_i = layers.Input(shape=len(feature_cols),), name="tabular")
t = layers.Dense(64, activation="relu")(tab_input_i)
t = layers.Dropout(0.3)(t)
t = layers.Dense(32, activation="relu")(t)
tab_features_i = layers.Dense(16, activation="relu", name="tab_features_inception")(t)

# ~~~~~ Fusion Layer ~~~~~
fusion_i = layers.Concatenate(name="fusion_layer_inception")([image_features_i, tab_features_i])
h = layers.Dense(64, activation="relu")(fusion_i)
h = layers.Dropout(0.4)(h)
output_i = layers.Dense(num_classes, activation="softmax", name="output_inception")(h)

# ~~~~~ Build Model ~~~~~
inception_multimodal_model = models.Model(
    inputs=[image_input_i, tab_input_i],
    outputs=output_i,
    name="InceptionV3_MLP_Multimodal"
)

# ~~~~~ Compile ~~~~~
inception_multimodal_model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
    loss='sparse_categorical_crossentropy',
    metrics=["accuracy"]
)

inception_multimodal_model.summary()
```

Model: "InceptionV3_MLP_Multimodal"

Layer (type)	Output Shape	Param #	Connected to
image ([InputLayer])	(None, 224, 224, 3)	0	–
true_divide_1 ([TrueDivide])	(None, 224, 224, 3)	0	image[0][0]
subtract_1 ([Subtract])	(None, 224, 224, 3)	0	true_divide_1[0]…
tabular ([InputLayer])	(None, 7)	0	–
inception_v3 ([Functional])	(None, 5, 5, 2048)	21,802,784	subtract_1[0][0]
dense_6 (Dense)	(None, 64)	512	tabular[0][0]
global_average_poo… ([GlobalAveragePool…	(None, 2048)	0	inception_v3[0][….
dropout_7 (Dropout)	(None, 64)	0	dense_6[0][0]
dropout_6 (Dropout)	(None, 2048)	0	global_average_p…
dense_7 (Dense)	(None, 32)	2,080	dropout_7[0][0]
image_features_inc… ([Dense])	(None, 128)	262,272	dropout_6[0][0]
tab_features_incep… ([Dense])	(None, 16)	528	dense_7[0][0]
fusion_layer_incep… ([ConcatenateV2])	(None, 144)	0	image_features_i… tab_features_inc…
dense_8 (Dense)	(None, 64)	9,280	fusion_layer_inc…
dropout_8 (Dropout)	(None, 64)	0	dense_8[0][0]
output_inception ([Dense])	(None, 4)	260	dropout_8[0][0]

Total params: 22,077,716 (84.22 MB)
Trainable params: 274,932 (1.05 MB)
Non-trainable params: 21,802,784 (83.17 MB)

In [56]:

```
epochs = 20

early_stop_inception = tf.keras.callbacks.EarlyStopping(
    monitor="val_loss",
    patience=5,
    restore_best_weights=True
)

history_inception_mm = inception_multimodal_model.fit(
    ds_train,
    epochs=epochs,
    validation_data=ds_val,
    callbacks=[early_stop_inception]
)
```

```
Epoch 1/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 19s 662ms/step - accuracy: 0.3206 - loss: 1.5222 - val_accuracy: 0.3857 - val_loss: 1.2921
Epoch 2/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 9s 541ms/step - accuracy: 0.3550 - loss: 1.2759 - val_accuracy: 0.5857 - val_loss: 1.1605
Epoch 3/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 9s 538ms/step - accuracy: 0.4427 - loss: 1.1643 - val_accuracy: 0.6857 - val_loss: 1.0358
Epoch 4/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 9s 543ms/step - accuracy: 0.5191 - loss: 1.1144 - val_accuracy: 0.7286 - val_loss: 0.9443
Epoch 5/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 9s 556ms/step - accuracy: 0.6069 - loss: 0.9384 - val_accuracy: 0.8429 - val_loss: 0.8301
Epoch 6/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 9s 557ms/step - accuracy: 0.6565 - loss: 0.8824 - val_accuracy: 0.7714 - val_loss: 0.7847
Epoch 7/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 567ms/step - accuracy: 0.6878 - loss: 0.8372 - val_accuracy: 0.8571 - val_loss: 0.6718
Epoch 8/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 585ms/step - accuracy: 0.7063 - loss: 0.6964 - val_accuracy: 0.8857 - val_loss: 0.6283
Epoch 9/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 1s/step - accuracy: 0.7001 - loss: 0.6520 - val_accuracy: 0.8429 - val_loss: 0.5597
Epoch 10/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 13s 740ms/step - accuracy: 0.8244 - loss: 0.5823 - val_accuracy: 0.8286 - val_loss: 0.5158
Epoch 11/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 579ms/step - accuracy: 0.8092 - loss: 0.5708 - val_accuracy: 0.8571 - val_loss: 0.4736
Epoch 12/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 634ms/step - accuracy: 0.8626 - loss: 0.4936 - val_accuracy: 0.8571 - val_loss: 0.4323
Epoch 13/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 628ms/step - accuracy: 0.8359 - loss: 0.5163 - val_accuracy: 0.8714 - val_loss: 0.4035
Epoch 14/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 568ms/step - accuracy: 0.8473 - loss: 0.4423 - val_accuracy: 0.8714 - val_loss: 0.3693
Epoch 15/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 595ms/step - accuracy: 0.8893 - loss: 0.3827 - val_accuracy: 0.8857 - val_loss: 0.3388
Epoch 16/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 639ms/step - accuracy: 0.8702 - loss: 0.4196 - val_accuracy: 0.9143 - val_loss: 0.3015
Epoch 17/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 609ms/step - accuracy: 0.9000 - loss: 0.3337 - val_accuracy: 0.9143 - val_loss: 0.2880
Epoch 18/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 584ms/step - accuracy: 0.9000 - loss: 0.3483 - val_accuracy: 0.9429 - val_loss: 0.2608
Epoch 19/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 632ms/step - accuracy: 0.9427 - loss: 0.2571 - val_accuracy: 0.9429 - val_loss: 0.2362
Epoch 20/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 10s 598ms/step - accuracy: 0.9000 - loss: 0.2813 - val_accuracy: 0.9143 - val_loss: 0.2367
```

In [58]:

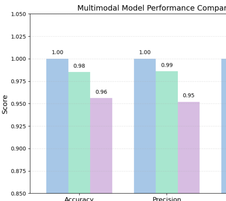
```
# ~~~~~ Predictions ~~~~~
y_prob_i = inception_multimodal_model.predict(ds_test)
y_pred_i = np.argmax(y_prob_i, axis=1)
y_true_i = test_labels

print("\nClassification Report (Multimodal InceptionV3 + MLP):\n")
print(classification_report(y_true_i, y_pred_i, target_names=apple_classes))

# ~~~~~ Confusion Matrix ~~~~~
cm_i = confusion_matrix(y_true_i, y_pred_i)
print("\nConfusion Matrix (Multimodal InceptionV3):\n", cm_i)

# ~~~~~ Metrics ~~~~~
acc_i = accuracy_score(y_true_i, y_pred_i)
prec_i = precision_score(y_true_i, y_pred_i, average="macro")
rec_i = recall_score(y_true_i, y_pred_i, average="macro")
f1_i = f1_score(y_true_i, y_pred_i, average="macro")

print("\nFinal Multimodal InceptionV3 Metrics:")
print("Accuracy:", acc_i)
print("Precision:", prec_i)
print("Recall:", rec_i)
print("F1-score:", f1_i)


# ~~~~~ Plot confusion matrix for InceptionV3 ~~~~~
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(6,5))

sns.heatmap(
    cm_i,                   # confusion matrix for Inception
    annot=True,             # show numbers
    fmt="d",                # display integers
    cmap="Purples",         # ~~ purple colormap
    xticklabels=class_names,
    yticklabels=class_names
)

plt.title("Confusion Matrix - InceptionV3 + MLP")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.tight_layout()
plt.show()
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 6s 814ms/step

Classification Report (Multimodal InceptionV3 + MLP):

                     precision    recall  f1-score   support

       Apple___Apple_scab       0.94      0.89      0.91        18
        Apple___Black_rot       1.00      1.00      1.00        20
Apple___Cedar_apple_rust       0.92      0.92      0.92        13
         Apple___healthy       0.94      1.00      0.97        17

               accuracy                           0.96        68
              macro avg       0.95      0.95      0.95        68
           weighted avg       0.96      0.96      0.96        68


Confusion Matrix (Multimodal InceptionV3):
 [[16  0  1  1]
 [ 0 20  0  0]
 [ 1  0 12  0]
 [ 0  0  0 17]]

Final Multimodal InceptionV3 Metrics:
Accuracy: 0.9558823529411765
Precision: 0.9527140555174007
Recall: 0.9529914529914529
F1-score: 0.9521978021978021
```

In [ ]:

In [61]:

```
import matplotlib.pyplot as plt
import numpy as np

# Models
models = ["MobileNetV2 + MLP", "ResNet50 + MLP", "InceptionV3 + MLP"]

# Metrics per metric
accuracy = [1, 0.985, 0.956]
precision = [1, 0.986, 0.952]
recall = [1, 0.986, 0.953]
f1 = [1, 0.9857, 0.9521]

metric_names = ["Accuracy", "Precision", "Recall", "F1-Score"]

metrics_by_model = [
    [accuracy[0], precision[0], recall[0], f1[0]],  # MobileNetV2 + MLP
    [accuracy[1], precision[1], recall[1], f1[1]],  # ResNet50 + MLP
    [accuracy[2], precision[2], recall[2], f1[2]]   # InceptionV3 + MLP
]

# Light pastel colors (as requested)
colors = [
    "#A7C7E7",  # Pastel Blue - MobileNetV2
    "#A8E6CF",  # Pastel Green - ResNet50
    "#D7BDE2"   # Pastel Purple - InceptionV3
]

x = np.arange(len(metric_names))  # [0,1,2,3]
width = 0.25

plt.figure(figsize=(12, 6))

# Plot bars per model
for i, model in enumerate(models):
    plt.bar(
        x + i*width,
        metrics_by_model[i],
        width,
        label=model,
        color=colors[i]
    )

# Add values on top of bars
for i in range(len(models)):
    for j in range(len(metric_names)):
        plt.text(
            x[j] + i*width,
            metrics_by_model[i][j] + 0.005,
            f"{metrics_by_model[i][j]:.2f}",
            ha="center",
            fontsize=10
        )

plt.xticks(x + width, metric_names, fontsize=12)
plt.ylim(0.85, 1.05)
plt.ylabel("Score", fontsize=13)
plt.title("Multimodal Model Performance Comparison (Apple Disease Detection)", fontsize=14)
plt.legend()
plt.grid(axis="y", linestyle="--", alpha=0.3)

plt.show()
```



In [ ]: