

CAP 6619 - Deep Learning

Project: Genomic Benchmarking

Jordan Winemiller

[Colab Project Link](#)

[Project Github Link](#)

[Overleaf Report](#)

PIP Package Installation, Setup, and Imports

```
In [74]: %%capture_code --path "/content/code/pip_install.py"
# PIP Package Installation
!pip install -q mermaid-py
!pip install -q genomic-benchmarks
!pip install -q jupyter-capture-output
```

Output saved by overwriting previous file at /content/code/pip_install.py.

```
In [50]: %%capture_code --path "/content/code/setup_and_imports.py"
# Setup and Imports
from pathlib import Path
import random
import os
import warnings

from genomic_benchmarks.data_check import (
    info,
    is_downloaded,
    list_datasets,
)
from genomic_benchmarks.loc2seq import download_dataset
import jupyter_capture_output
import keras
from keras import backend as K
from keras.layers import (
    Activation,
    BatchNormalization,
    Conv1D,
    Dense,
    Dropout,
    Flatten,
    GlobalAveragePooling1D,
    Input,
    Lambda,
    MaxPooling1D,
    TextVectorization,
)
from keras.losses import (
    BinaryCrossentropy,
    CategoricalCrossentropy,
)
from keras.metrics import (
```

```

        BinaryAccuracy,
        CategoricalAccuracy,
    )
    from keras.models import Sequential
    import keras.ops as ops
    import matplotlib.pyplot as plt
    from mermaid import Mermaid
    import numpy as np
    import pandas as pd
    import tensorflow as tf

    SEED = 1234

    os.environ["PYTHONHASHSEED"] = str(SEED)
    random.seed(SEED)
    np.random.seed(SEED)
    keras.utils.set_random_seed(SEED)
    tf.random.set_seed(SEED)

    # Suppress Keras warnings
    warnings.filterwarnings("ignore")

```

Note: The /content/code directory was successfully created.
Output saved by creating file at /content/code/setup_and_imports.py.

Datasets

```

In [52]: %%capture_code --path "/content/code/datasets.py"
        from genomic_benchmarks.data_check import list_datasets

        list_datasets()

```

```

Out[52]: ['demo_human_or_worm',
          'drosophila_enhancers_stark',
          'dummy_mouse_enhancers_ensembl',
          'human_ensembl_regulatory',
          'human_enhancers_cohn',
          'demo_coding_vs_intergenomic_seqs',
          'human_nontata_promoters',
          'human_enhancers_ensembl',
          'human_ocr_ensembl']

```

Output saved by creating file at /content/code/datasets.py.

```

In [53]: %%capture_code --path "/content/code/download_datasets.py"
        selected_dataset_list = [
            "dummy_mouse_enhancers_ensembl",
            "drosophila_enhancers_stark",
            "human_enhancers_cohn",
            "human_nontata_promoters",
        ]

        for dataset in selected_dataset_list:
            if not is_downloaded(dataset):
                download_dataset(dataset)
            print(info(dataset, description=True))
            print("\n")

```

Output saved by creating file at /content/code/download_datasets.py.

Create Training and Test Sets

```
In [6]: batch_size = 64
        selected_dataset = selected_dataset_list[1]
        SEQ_PATH = Path.home() / ".genomic_benchmarks" / selected_dataset
        SEQ_PATH
```

```
Out[6]: PosixPath('/root/.genomic_benchmarks/drosophila_enhancers_stark')
```

```
In [7]: classes = [
        x.stem for x
        in (SEQ_PATH/"train").iterdir()
        if x.is_dir()
    ]
    num_classes = len(classes)

    train_set = tf.keras.preprocessing.text_dataset_from_directory(
        SEQ_PATH / "train",
        batch_size=batch_size,
        class_names=classes,
        shuffle=True,
        seed=SEED,
    )

    test_set = tf.keras.preprocessing.text_dataset_from_directory(
        SEQ_PATH / "test",
        batch_size=batch_size,
        class_names=classes,
    )

    if num_classes > 2:
        train_set = train_set.map(
            lambda x, y: (x, tf.one_hot(y, depth=num_classes)))

    if num_classes > 2:
        test_set = test_set.map(
            lambda x, y: (x, tf.one_hot(y, depth=num_classes)))
```

Found 5184 files belonging to 2 classes.

Found 1730 files belonging to 2 classes.

```
In [8]: print(f"Training Set Type: {type(train_set)}")
        print(f"Testing Set Type: {type(test_set)}")
```

Training Set Type: <class 'tensorflow.python.data.ops.prefetch_op._PrefetchDataset'>

Testing Set Type: <class 'tensorflow.python.data.ops.prefetch_op._PrefetchDataset'>

Vectorize the Dataset

NOTE: This code for the model and vectorize layer had to be update for use with python3.11 and Keras>=3.0

```
In [54]: %%capture_code --path "/content/code/vectorize_layer.py"
        char_split_fn = lambda x: tf.strings.unicode_split(x, input_encoding="UTF-8")
        vectorize_layer = keras.layers.TextVectorization(
            output_mode="int",
            split=char_split_fn,
        )

        vectorize_layer.adapt(train_set.map(lambda x, y: x))
```

```
vocab_size = len(vectorize_layer.get_vocabulary())
vectorize_layer.get_vocabulary()
```

Output saved by creating file at /content/code/vectorize_layer.py.

```
Out[54]: ['', '[UNK]', np.str_('t'), np.str_('a'), np.str_('c'), np.str_('g')]
```

```
In [55]: %%capture_code --path "/content/code/vectorize_text.py"
def vectorize_text(text, label):
    """Returns a vector representation of the text.

    :param text: The text to vectorize
    :param label: The label of the text
    :return: A vector representation of the text
    """
    text = tf.expand_dims(text, axis=-1)
    return vectorize_layer(text)-2, label
```

Output saved by creating file at /content/code/vectorize_text.py.

```
In [11]: train_ds = train_set.map(vectorize_text)
test_ds = test_set.map(vectorize_text)

print(f"Training Set Type: {type(train_ds)}")
print(f"Testing Set Type: {type(test_ds)}")
```

Training Set Type: <class 'tensorflow.python.data.ops.map_op._MapDataset'>

Testing Set Type: <class 'tensorflow.python.data.ops.map_op._MapDataset'>

Check Sequence Length

```
In [56]: %%capture_code --path "/content/code/check_lengths.py"
def check_seq_lengths(dataset, use_padding):
    """Returns the maximum sequence length and the length of the sequence with
    tokens.

    :param dataset: List of sequences
    :param use_padding: Padding
    :return: Maximum sequence length and length of sequence with tokens
    """
    max_seq_len = max([len(dataset[i][0]) for i in range(len(dataset))])
    print(f"Max Sequence Length: {max_seq_len}")
    same_length = [len(dataset[i][0]) == max_seq_len for i in range(len(dataset))]
    if not all(same_length):
        print("not all sequences are of the same length")

    if use_padding:
        len_with_tokens = max_seq_len + 3
    else:
        len_with_tokens = max_seq_len + 2

    return max_seq_len, len_with_tokens
```

Output saved by creating file at /content/code/check_lengths.py.

```
In [13]: data_list = list(train_ds.as_numpy_iterator())
max_seq_len, nn_input_len = check_seq_lengths(data_list, use_padding=True)
```

Max Sequence Length: 64

Update Dataset

```
In [57]: %%capture_code --path "/content/code/update_train_test_sets.py"
def update_train_test_sets(id_num, batch_size=64):
    """Updates the training and testing sets, and returns the name of the
    dataset.

    :param id_num: The ID number of the dataset
    :param batch_size: The batch size
    """

    batch_size = batch_size
    selected_dataset = selected_dataset_list[id_num]
    SEQ_PATH = Path.home() / ".genomic_benchmarks" / selected_dataset

    classes = [
        x.stem for x
        in (SEQ_PATH/"train").iterdir()
        if x.is_dir()
    ]
    num_classes = len(classes)

    train_set = tf.keras.preprocessing.text_dataset_from_directory(
        SEQ_PATH / "train",
        batch_size=batch_size,
        class_names=classes,
        shuffle=True,
        seed=SEED,
    )

    test_set = tf.keras.preprocessing.text_dataset_from_directory(
        SEQ_PATH / "test",
        batch_size=batch_size,
        class_names=classes,
    )

    if num_classes > 2:
        train_set = train_set.map(
            lambda x, y: (x, tf.one_hot(y, depth=num_classes)))

    if num_classes > 2:
        test_set = test_set.map(
            lambda x, y: (x, tf.one_hot(y, depth=num_classes)))

    vectorize_layer.adapt(train_set.map(lambda x, y: x))
    vocab_size = len(vectorize_layer.get_vocabulary())
    vectorize_layer.get_vocabulary()

    train_ds = train_set.map(vectorize_text)
    test_ds = test_set.map(vectorize_text)

    return selected_dataset
```

Output saved by creating file at /content/code/update_train_test_sets.py.

Model Creation

Metrics

```
In [58]: %%capture_code --path "/content/code/f1_score.py"
def f1_score(y_true, y_pred):
    """Returns the F1 score.

    :param y_true: The true labels
```

```

:param y_pred: The predicted labels
:return: The F1 score
"""
def precision(y_true, y_pred):
    """Returns the precision.

    :param y_true: The true labels
    :param y_pred: The predicted labels
    :return: The precision
    """
    true_positives = ops.sum(
        ops.round(ops.clip(tf.cast(y_true, tf.float32) * y_pred, 0, 1)))
    predicted_positives = ops.sum(ops.round(ops.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision

def recall(y_true, y_pred):
    """Returns the recall.

    :param y_true: The true labels
    :param y_pred: The predicted labels
    :return: The recall
    """
    true_positives = ops.sum(
        ops.round(ops.clip(tf.cast(y_true, tf.float32) * y_pred, 0, 1)))
    possible_positives = tf.cast(
        ops.sum(ops.round(ops.clip(y_true, 0, 1))), tf.float32)
    recall = (
        tf.cast(true_positives, tf.float32)
        / (possible_positives + K.epsilon())
    )
    return recall

precision = precision(y_true, y_pred)
recall = recall(y_true, y_pred)
return 2 * ((precision * recall) / (precision + recall + K.epsilon()))

```

Output saved by creating file at /content/code/f1_score.py.

Initial Model

```

In [64]: %%capture_code --path "/content/code/initial_model.py"
def create_basic_cnn_model(num_classes, vocab_size):
    """Returns a basic CNN model and model name.

    :param num_classes: The number of classes to classify
    :param vocab_size: The size of the vocabulary
    :return: Model name and CNN model
    """
    name = "basic"

    if num_classes == 2:
        last_layer = Dense(1, activation="sigmoid")
        loss = BinaryCrossentropy(from_logits=True)
        acc = BinaryAccuracy()

    else:
        last_layer = Dense(num_classes, activation="softmax")
        loss = "categorical_crossentropy"
        acc = CategoricalAccuracy()

    onehot_layer = Lambda(

```

```

        lambda x: tf.one_hot(tf.cast(x, "int64"), depth=vocab_size))

print(onehot_layer)

model = Sequential(
    [
        onehot_layer,
        Conv1D(
            filters=32,
            kernel_size=8,
            data_format="channels_last",
            activation="relu",
        ),
        BatchNormalization(),
        MaxPooling1D(),
        Conv1D(
            filters=16,
            kernel_size=8,
            data_format="channels_last",
            activation="relu",
        ),
        BatchNormalization(),
        MaxPooling1D(),
        Conv1D(
            filters=4,
            kernel_size=8,
            data_format="channels_last",
            activation="relu",
        ),
        BatchNormalization(),
        MaxPooling1D(),
        Dropout(0.3),
        GlobalAveragePooling1D(),
        last_layer,
    ]
)

model.compile(
    optimizer="adam",
    loss=loss,
    metrics=[acc, f1_score],
)

return name, model

```

Output saved by creating file at /content/code/initial_model.py.

Final Model

NOTE: This model has the updated architecture adding the Global Average Pooling Layer

```

In [62]: %%capture_code --path "/content/code/final_model.py"
def create_final_cnn_model(num_classes, vocab_size):
    """Returns a CNN model and model name.

    :param num_classes: The number of classes to classify
    :param vocab_size: The size of the vocabulary
    :return: Model name and CNN model
    """
    name = "final"

    if num_classes == 2:

```

```

        last_layer = Dense(1, activation="sigmoid")
        loss = BinaryCrossentropy(from_logits=True)
        acc = BinaryAccuracy(threshold=0.5)
    else:
        last_layer = Dense(num_classes, activation="softmax")
        loss = "categorical_crossentropy"
        acc = CategoricalAccuracy()

    onehot_layer = Lambda(
        lambda x: tf.one_hot(tf.cast(x, "int64"), depth=vocab_size))

    model = Sequential(
        [
            onehot_layer,

            Conv1D(
                filters=16,
                kernel_size=8,
                data_format="channels_last",
            ),
            BatchNormalization(),
            Activation("relu"),
            MaxPooling1D(),

            Conv1D(
                filters=8,
                kernel_size=8,
                data_format="channels_last",
                activation="relu",
            ),
            BatchNormalization(),
            MaxPooling1D(),

            Conv1D(
                filters=4,
                kernel_size=8,
                data_format="channels_last",
                activation="relu",
            ),
            BatchNormalization(),
            MaxPooling1D(),

            GlobalAveragePooling1D(),
            Flatten(),
            Dense(units=512, activation="relu"),
            last_layer,
        ]
    )

    model.compile(
        optimizer="adam",
        loss=loss,
        metrics=[acc, f1_score],
    )

    return name, model

```

Output saved by creating file at /content/code/final_model.py.

Run Models

Metrics and Plots

```
In [67]: %%capture_code --path "/content/code/display_metrics.py"
def plot_metrics(model_info_dict, set_name, horizontal=False):
    """Shows the plots for the training accuracy, f1 score, and loss.
    Then shows the scores for the model.

    :param model_info_dict: Dictionary of model information
    :param set_name: Name of dataset
    :param horizontal: Whether to show the plots horizontally
    """
    b_hist = model_info_dict["basic_history"]
    acc_1 = np.array(b_hist.history["binary_accuracy"])
    f1_1 = np.array(b_hist.history["f1_score"])
    loss_1 = np.array(b_hist.history["loss"])

    f_hist = model_info_dict["final_history"]
    acc_2 = np.array(f_hist.history["binary_accuracy"])
    f1_2 = np.array(f_hist.history["f1_score"])
    loss_2 = np.array(f_hist.history["loss"])

    # Shifted the starting index to start at 1 instead of 0
    epochs = np.arange(loss_1.shape[0]) + 1

    fig_size = (15, 5) if horizontal else (5, 15)
    plt.figure(figsize=fig_size)
    models = [model_info_dict["basic_name"], model_info_dict["final_name"]]

    rows = 1 if horizontal else 3
    cols = 3 if horizontal else 1

    plt.subplot(rows, cols, 1)
    plt.plot(epochs, acc_1, epochs, acc_2)
    plt.title("Training Accuracy")
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.legend(models, loc="lower right")

    plt.subplot(rows, cols, 2)
    plt.plot(epochs, f1_1, epochs, f1_2, linestyle="--")
    plt.title("Training F1 Score")
    plt.xlabel("Epochs")
    plt.ylabel("F1 Score")
    plt.legend(models, loc="lower right")

    plt.subplot(rows, cols, 3)
    plt.plot(epochs, loss_1, epochs, loss_2)
    plt.title("Loss")
    plt.xlabel("Epochs")
    plt.ylabel("Crossentropy Loss")
    plt.legend(models, loc="upper right")

    plt.show()

    for m in models:
        eval = model_info_dict[f"{m}_evaluation"]
        print(f"{m.title()} Model for Set: {set_name}:")
        print(f"Total Loss: {round(eval['loss'], 6)}")
        print(f"Accuracy: {round(eval['binary_accuracy'] * 100, 2)}%")
        print(f"F1 Score: {round(eval['f1_score'], 2)}")
        print()
```

Output saved by creating file at /content/code/display_metrics.py.

Train and Evaluate Models

```
In [68]: %%capture_code --path "/content/code/training_model.py"
def train_and_evaluate_models(set_name, epochs):
    """Returns the training and evaluation the metrics for the models.

    :param set_name: Name of dataset
    :param epochs: Number of epochs to train
    :return: Dictionary of model information
    """
    model_info = {
        "basic_name": None,
        "basic_history": None,
        "basic_evaluation": None,
        "final_name": None,
        "final_history": None,
        "final_evaluation": None,
    }

    print(f"Training Models for: {set_name}")
    basic_name, basic_model = create_basic_cnn_model(num_classes, vocab_size)
    model_info["basic_name"] = basic_name
    basic_history = basic_model.fit(
        train_ds,
        epochs=epochs,
        verbose=0,
    )
    model_info["basic_history"] = basic_history
    model_info["basic_evaluation"] = basic_model.evaluate(
        test_ds, verbose=0, return_dict=True)

    final_name, final_model = create_final_cnn_model(num_classes, vocab_size)
    model_info["final_name"] = final_name
    final_history = final_model.fit(
        train_ds,
        epochs=epochs,
        verbose=0,
    )
    model_info["final_history"] = final_history
    model_info["final_evaluation"] = final_model.evaluate(
        test_ds, verbose=0, return_dict=True)

    return model_info
```

Output saved by creating file at /content/code/training_model.py.

```
In [20]: epochs = 10
```

Mouse Enhancer regions vs Random background regions from GRCm38

Dataset 0: dummy_mouse_enhancers_ensembl

- The length of genomic intervals ranges from 331 to 4776, with average 2369.5769 and median 2381.0
- Totally 1210 sequences have been found, 968 for training and 242 for testing

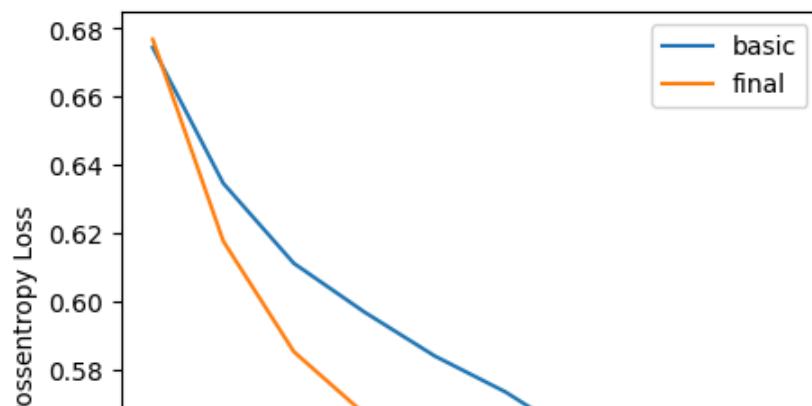
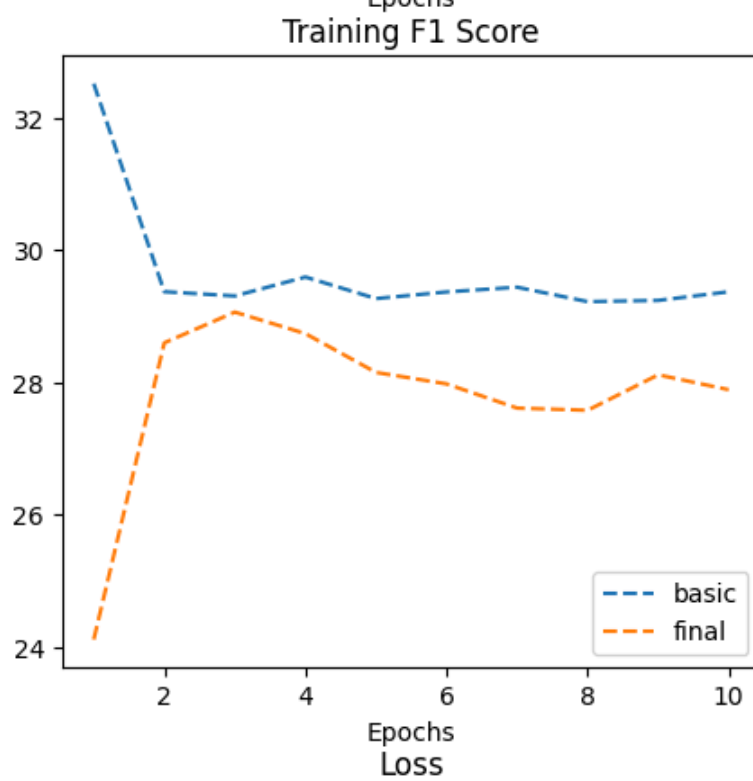
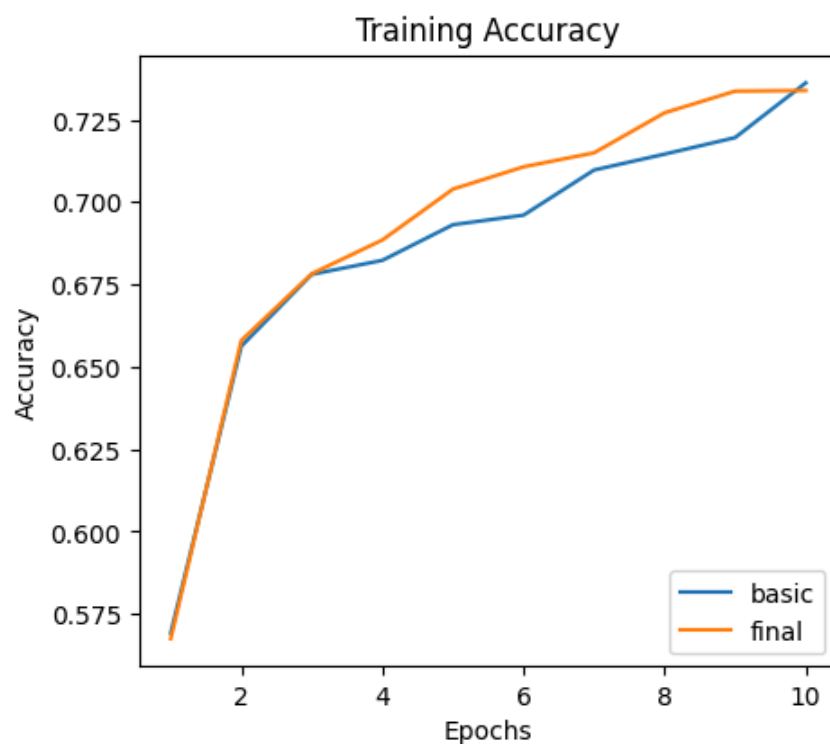
```
In [72]: dataset = update_train_test_sets(0)
model_0 = train_and_evaluate_models(dataset, epochs)
```

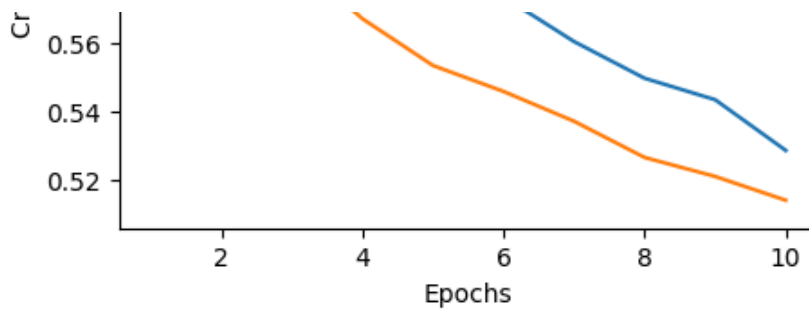
Output saved by overwring previous file at /content/code/example_model_run.py.

```
In [22]: %%capture_img --path "/content/images/model_0.png"
%%capture_text --path "/content/images/model_0.txt"
plot_metrics(model_0, dataset)
```

Note: The /content/images directory was successfully created.
Output saved by creating file at /content/images/model_0.png.
Output saved by creating file at /content/images/model_0.txt.
Basic Model for Set: dummy_mouse_enhancers_ensembl:
Total Loss: 0.579737
Accuracy: 66.82%
F1 Score: 33.13

Final Model for Set: dummy_mouse_enhancers_ensembl:
Total Loss: 0.999781
Accuracy: 50.64%
F1 Score: 0.76



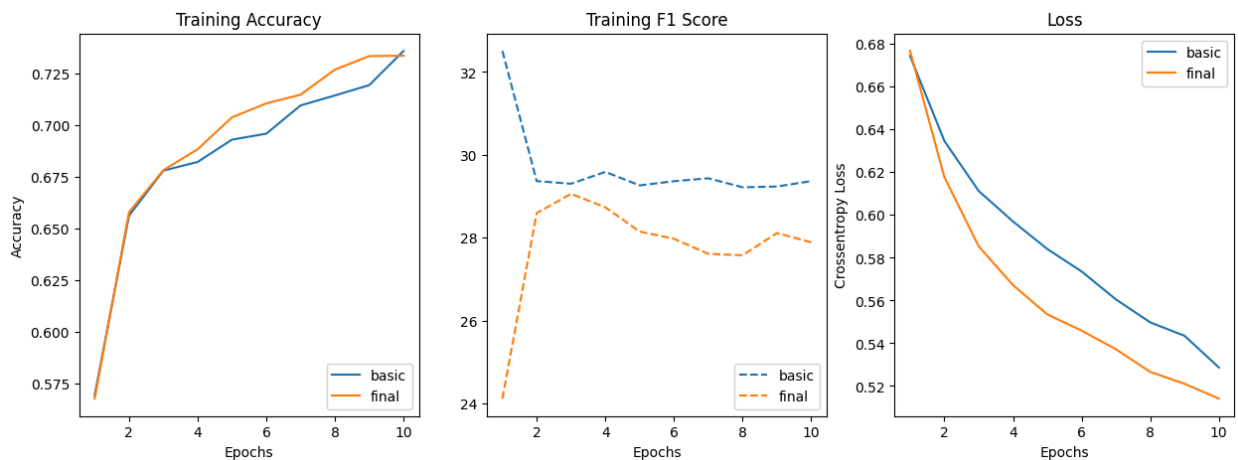


In [23]: `# For presentation`

```
%%capture_img --path "/content/images/model_0_pp.png"
plot_metrics(model_0, dataset, True)
```

Output saved by creating file at /content/images/model_0_pp.png.
 Output saved by creating file at /content/images/model_0_pp.txt.
 Basic Model for Set: dummy_mouse_enhancers_ensembl:
 Total Loss: 0.579737
 Accuracy: 66.82%
 F1 Score: 33.13

Final Model for Set: dummy_mouse_enhancers_ensembl:
 Total Loss: 0.999781
 Accuracy: 50.64%
 F1 Score: 0.76



Drosophila enhancers vs Random background regions from dm6

Dataset 1: *drosophila_enhancers_stark*

- The length of genomic intervals ranges from 236 to 3237, with average 2118.1238 and median 2142.0
- Totally 6914 sequences have been found, 5184 for training and 1730 for testing

In [24]: `dataset = update_train_test_sets(1)`
`model_1 = train_and_evaluate_models(dataset, epochs)`

Found 5184 files belonging to 2 classes.
 Found 1730 files belonging to 2 classes.
 Training Models for: *drosophila_enhancers_stark*
 <Lambda name=lambda_2, built=False>

```
In [25]: %%capture_img --path "/content/images/model_1.png"
%%capture_text --path "/content/images/model_1.txt"
plot_metrics(model_1, dataset)
```

Output saved by creating file at /content/images/model_1.png.

Output saved by creating file at /content/images/model_1.txt.

Basic Model for Set: drosophila_enhancers_stark:

Total Loss: 1.369164

Accuracy: 49.94%

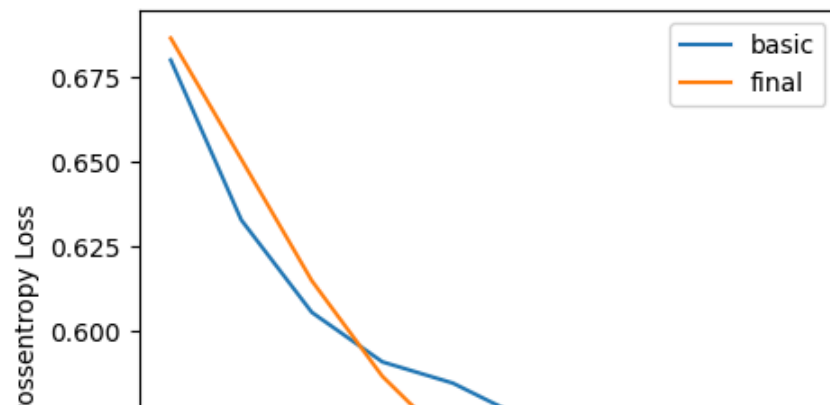
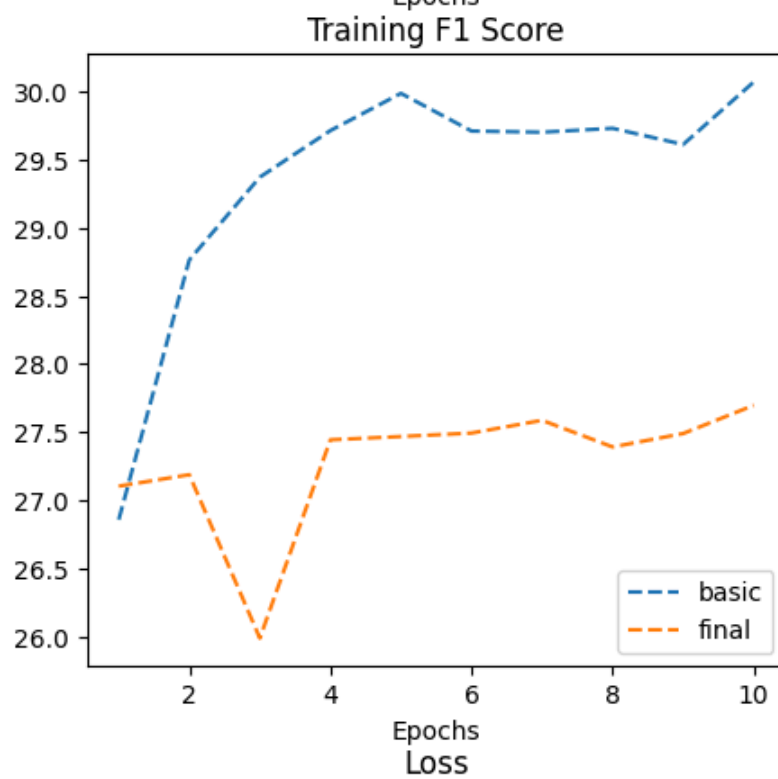
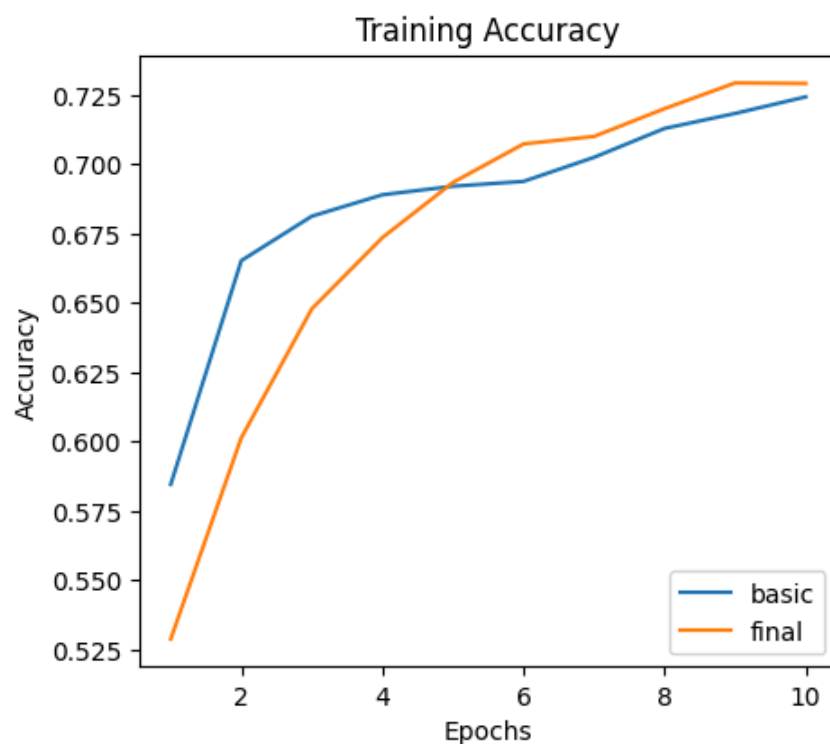
F1 Score: 40.93

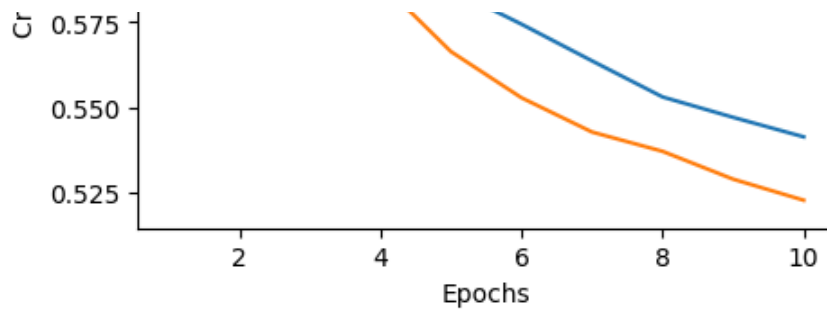
Final Model for Set: drosophila_enhancers_stark:

Total Loss: 0.892226

Accuracy: 50.46%

F1 Score: 0.82





In [26]: `# For presentation`

```
%%capture_img --path "/content/images/model_1_pp.png"
plot_metrics(model_1, dataset, True)
```

Output saved by creating file at /content/images/model_1_pp.png.

Basic Model for Set: drosophila_enhancers_stark:

Total Loss: 1.369164

Accuracy: 49.94%

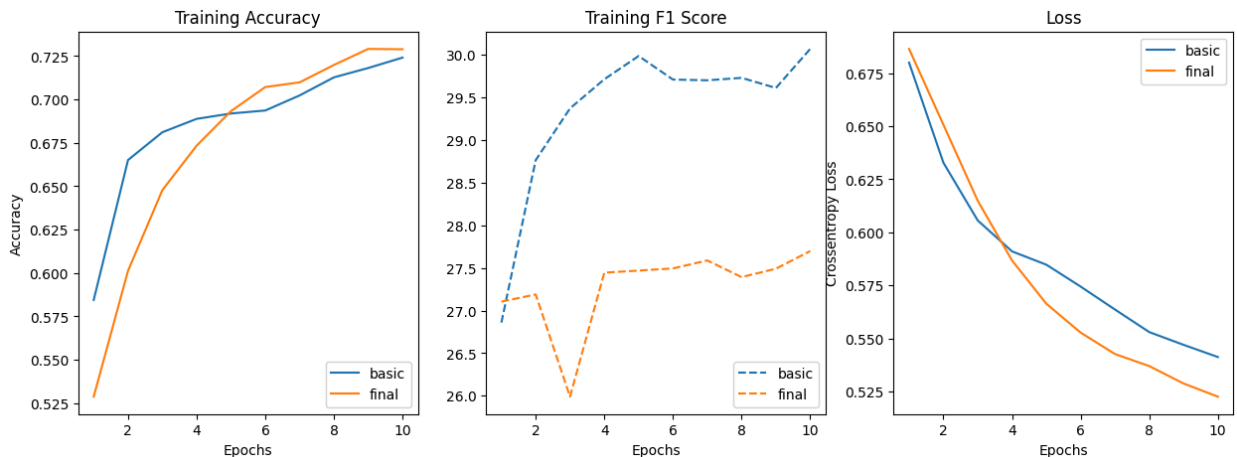
F1 Score: 40.93

Final Model for Set: drosophila_enhancers_stark:

Total Loss: 0.892226

Accuracy: 50.46%

F1 Score: 0.82



Human Enhancers

Dataset 2: human_enhancers_cohn

- All lengths of genomic intervals equals 500
- Totally 27791 sequences have been found, 20843 for training and 6948 for testing

```
In [27]: dataset = update_train_test_sets(2)
model_2 = train_and_evaluate_models(dataset, epochs)
```

Found 20843 files belonging to 2 classes.

Found 6948 files belonging to 2 classes.

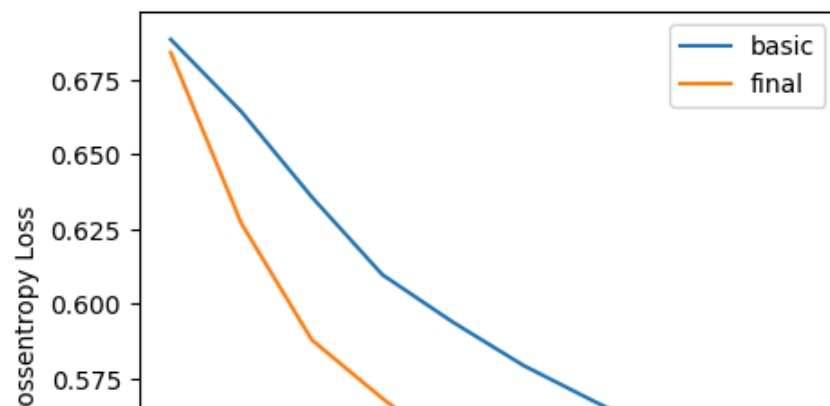
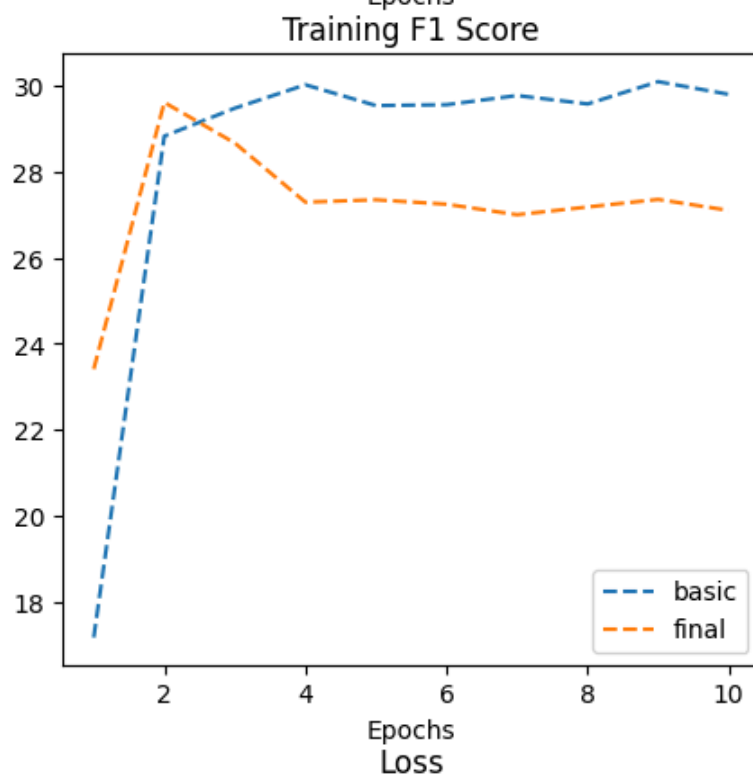
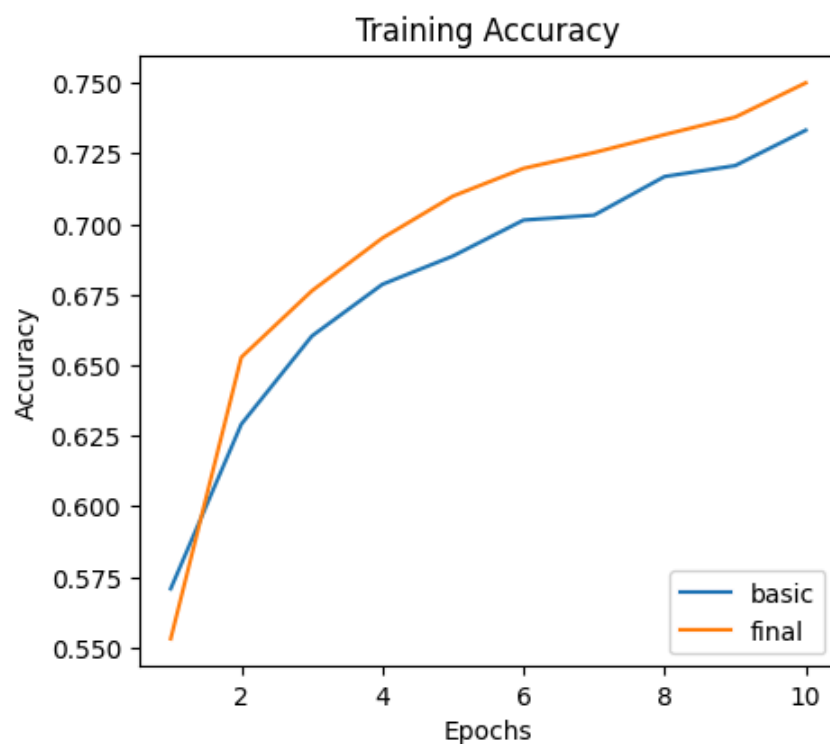
Training Models for: human_enhancers_cohn

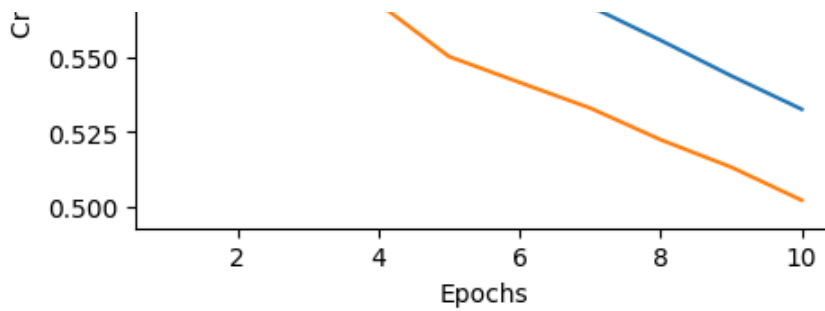
<Lambda name=lambda_4, built=False>

```
In [28]: %%capture_img --path "/content/images/model_2.png"
%%capture_text --path "/content/images/model_2.txt"
plot_metrics(model_2, dataset)
```

Output saved by creating file at /content/images/model_2.png.
Output saved by creating file at /content/images/model_2.txt.
Basic Model for Set: human_enhancers_cohn:
Total Loss: 0.792922
Accuracy: 52.6%
F1 Score: 40.45

Final Model for Set: human_enhancers_cohn:
Total Loss: 0.626275
Accuracy: 65.09%
F1 Score: 14.97





In [29]: `# For presentation`

```
%%capture_img --path "/content/images/model_2_pp.png"
plot_metrics(model_2, dataset, True)
```

Output saved by creating file at /content/images/model_2_pp.png.

Basic Model for Set: human_enhancers_cohn:

Total Loss: 0.792922

Accuracy: 52.6%

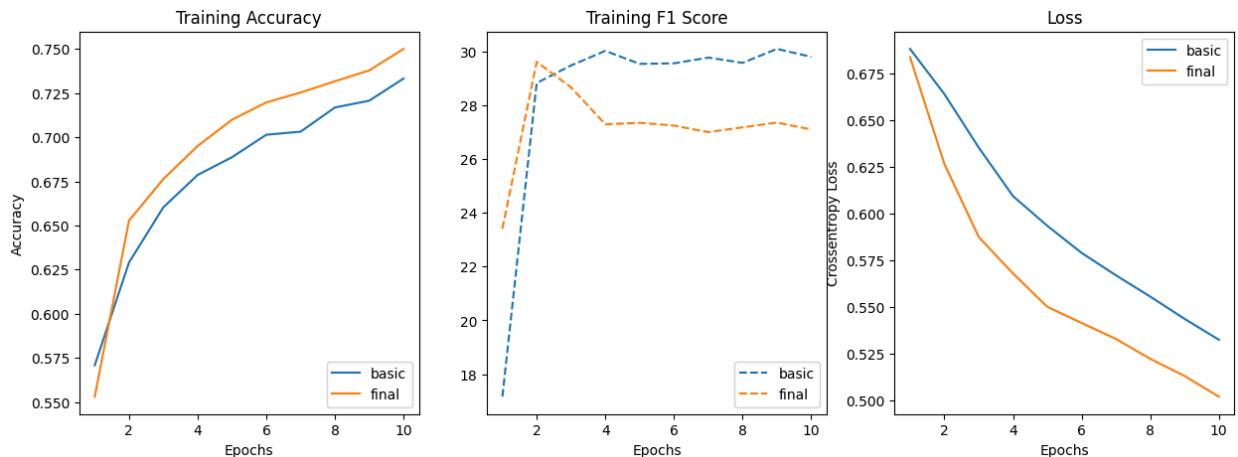
F1 Score: 40.45

Final Model for Set: human_enhancers_cohn:

Total Loss: 0.626275

Accuracy: 65.09%

F1 Score: 14.97



Human non-TATA promoters

Dataset 3: *human_nontata_promoters*

- All lengths of genomic intervals equals 251
- Totally 36131 sequences have been found, 27097 for training and 9034 for testing

In [30]: `dataset = update_train_test_sets(3)`
`model_3 = train_and_evaluate_models(dataset, epochs)`

Found 27097 files belonging to 2 classes.

Found 9034 files belonging to 2 classes.

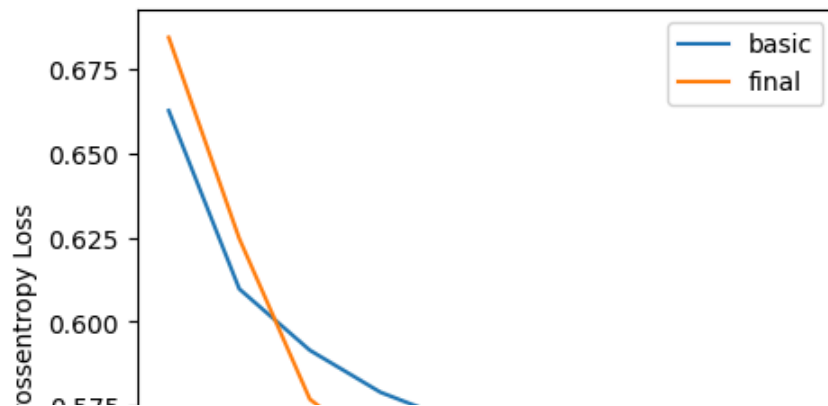
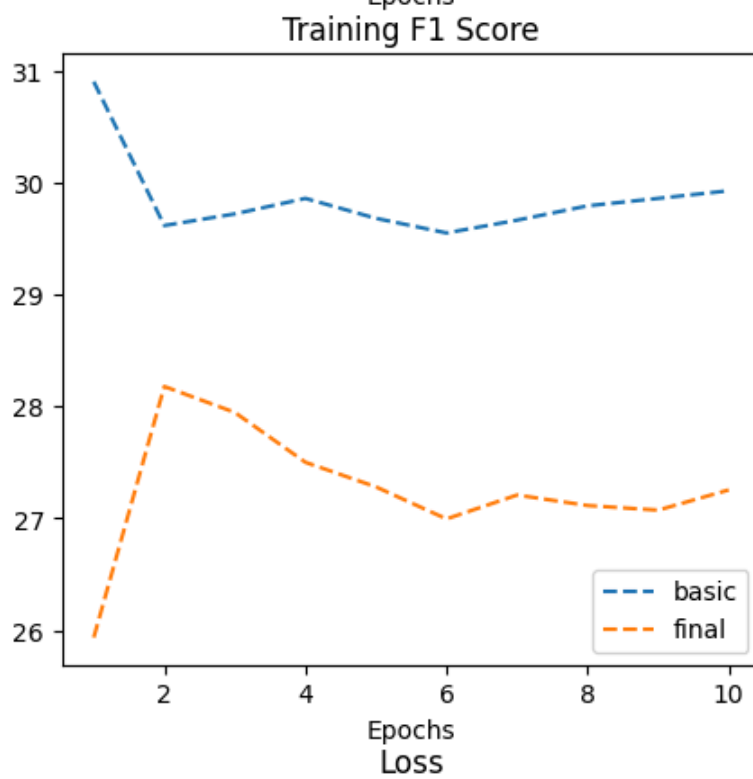
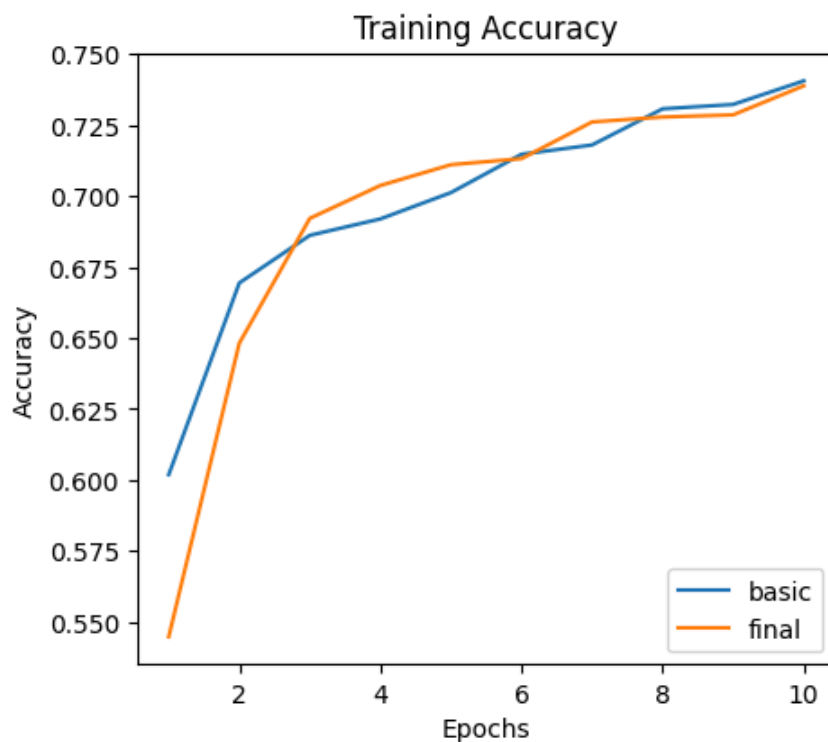
Training Models for: human_nontata_promoters

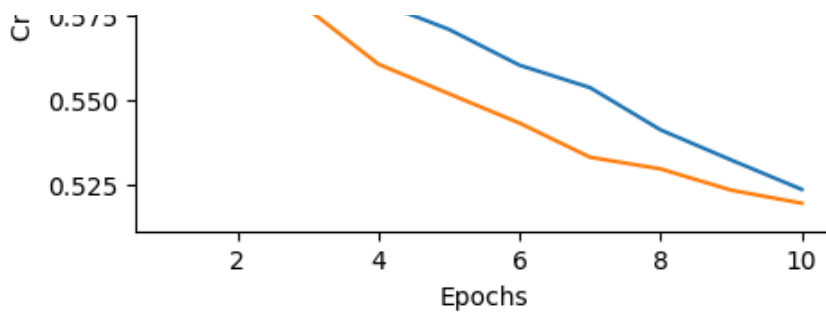
<Lambda name=lambda_6, built=False>

In [31]: `%%capture_img --path "/content/images/model_3.png"`
`%%capture_text --path "/content/images/model_3.txt"`
`plot_metrics(model_3, dataset)`

Output saved by creating file at /content/images/model_3.png.
Output saved by creating file at /content/images/model_3.txt.
Basic Model for Set: human_nontata_promoters:
Total Loss: 0.581651
Accuracy: 71.56%
F1 Score: 23.3

Final Model for Set: human_nontata_promoters:
Total Loss: 0.606759
Accuracy: 66.36%
F1 Score: 20.33





In [32]: `# For presentation`

```
%%capture_img --path "/content/images/model_3_pp.png"
plot_metrics(model_3, dataset, True)
```

Output saved by creating file at /content/images/model_3_pp.png.

Basic Model for Set: human_nontata_promoters:

Total Loss: 0.581651

Accuracy: 71.56%

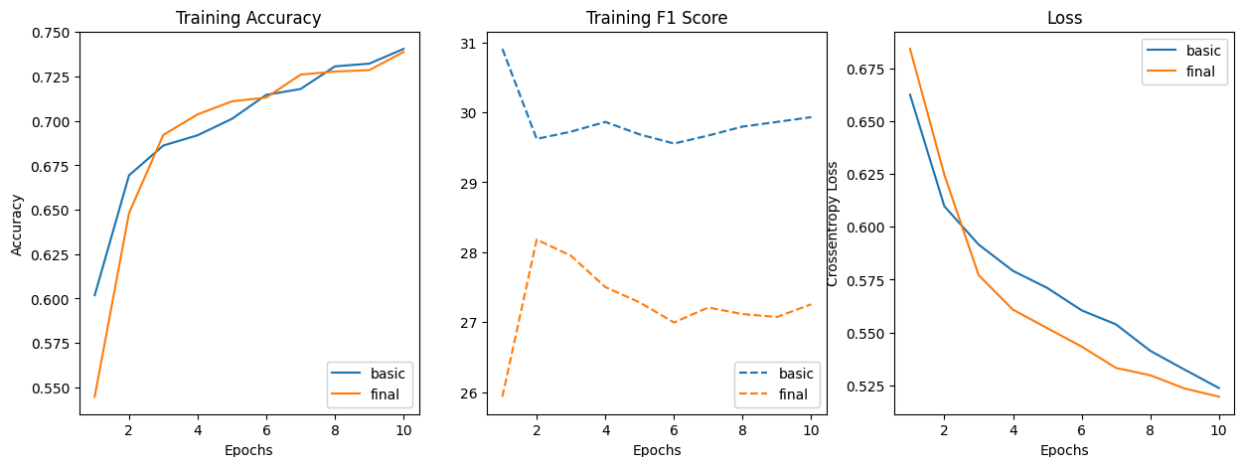
F1 Score: 23.3

Final Model for Set: human_nontata_promoters:

Total Loss: 0.606759

Accuracy: 66.36%

F1 Score: 20.33



Mitigation and Workarounds

The issues that could not be solve have been added to this text block so the commands do not execute.

Tensorflow-Addons Issue:

```
!pip install -q tensorflow-addons
```

After I was able to get the package install there was an issue with the *Keras* version and the removal of the engine object in the new version of *Keras*. Also a this would have only worked for the first tested model. So, a new F_1 Score metric was written instead.

Torchtext Issue:

There was problem with the compability of the torch version with torchtext. There also was a problem with the python and cuda versions with a older version of torch, but a reinstall was attempted.

```
import torch
print(torch.__version__)
```

Output:

2.6.0+cu124

```
!python --version
```

```
!sudo apt-get -qq install python3.11 \
    python3.11-distutils \
    python3-pip
```

```
!sudo update-alternatives --install /usr/local/bin/python3 \
    python3 \
    /usr/bin/python3.11 1
```

```
!sudo update-alternatives --config python3
```

```
!python --version
```

```
!pip uninstall torch torchtext -y
```

```
!pip cache purg
```

Example for a specific CUDA version (check PyTorch website for the exact command)

```
!pip install torch==2.2.0 torchvision==0.17.0 torchtext==0.18.0
torchaudio==2.2.0 --index-url https://download.pytorch.org/whl/cu12
```

Output:

ERROR: Could not find a version that satisfies the requirement torchtext==0.18.0 (from versions: 0.5.0, 0.6.0, 0.15.0+cpu, 0.15.1+cpu, 0.15.2+cpu, 0.16.0+cpu, 0.16.1+cpu, 0.16.2+cpu, 0.17.0+cpu).

ERROR: No matching distribution found for torchtext==0.18.0

Genomic-benchmarks Issues:

These imports were could not be used do to *tensorflow-addons* package or the version update of *keras*.

```
from genomic_benchmarks.models.tf import (
    get_basic_cnn_model_v0,
    vectorize_layer,
)
```

These imports relied on the version of *torch* and *torchtext* packages from the above issue.

```
from genomic_benchmarks.dataset_getters.pytorch_datasets import (
    get_dataset,
)
```

```
from genomic_benchmarks.dataset_getters.utils import (
    build_vocab,
    check_seq_lengths,
    check_config,
    coll_factory,
    LetterTokenizer,
    VARIABLE_LENGTH_DATASETS,
)
```

```
from genomic_benchmarks.models.torch import CNN
```

Dataset Issue:

Testing moving the datasets from a prefetchdataset to a native *numpy* arrays.

```
import tensorflow_datasets as tfds
x1 = np.asarray(list(map(lambda x: x[0], tfds.as_numpy(train_ds))))
x1
```

Output:

```
-----
ValueError Traceback (most recent call last)
/tmp/ipython-input-80-1145905047.py in ()
1 import tensorflow_datasets as tfds
----> 2 x1 = np.asarray(list(map(lambda x: x[0], tfds.as_numpy(train_ds))))
3 x1

ValueError: setting an array element with a sequence. The requested array has an inhomogeneous shape after 1
dimensions. The detected shape was (16,) + inhomogeneous part.
```

Keras Issue (Solved):

The built-in *F1Score* in the metrics package in *keras* would return this error when the model was training. Output:

```
/usr/local/lib/python3.11/dist-packages/keras/src/metrics/f_score_metrics.py in _build(self, y_true_shape,
y_pred_shape)
122 def _build(self, y_true_shape, y_pred_shape):
123 if len(y_pred_shape) != 2 or len(y_true_shape) != 2:
--> 124 raise ValueError(
125 "FBetaScore expects 2D inputs with shape "
126 "(batch_size, output_dim). Received input "
ValueError: FBetaScore expects 2D inputs with shape (batch_size, output_dim). Received input shapes:
y_pred.shape=(None, 1) and y_true.shape=(None,).
```

Fix: A function was added to gather the *F1 Score* metric.

Jupyter Capture Issue:

The package was successful installed and the cell output capture is reporting a working status, but outputs are not appearing in the directory. *FIX: The capture magic methods did work for some image and text outputs.*

Mermaid-py Issue:

The package was cutting off text in the layer diagram. *FIX: I used the online editor at [Mermaid Live Editor](#)*

Mermaid Diagrams

```
In [47]: basic_cnn_graph = """
%%{init:{'flowchart':{'nodeSpacing':1, 'rankSpacing':10}}}%

flowchart TD
    classDef withMargins fill-opacity:0.0,color:#FFFFFF,stroke-width:0px;
    i(Embedding)

    subgraph conv1[Convolution Layer 1]
        space1["<p style='width:100px;height:0px;margin:0'>Space</p>"]:::withMargins;
        c1@{ shape: st-rect, label: "Conv1D:<br>32 filters<br>kernel 8" }
        b1@{ shape: st-rect, label: "Batch Normalization" }
        p1@{ shape: st-rect, label: "Max Pooling" }
```

```

end
%% Define a class to make the padding subgraph invisible
classDef padding stroke:none,fill:none

subgraph conv2[Convolution Layer 2]
    space2["<p style='width:100px;height:0px;margin:0'>Space</p>"]::withMargins;
    c2@{ shape: st-rect, label: "Conv1D:\n16 filters – kernel 8" }
    b2@{ shape: st-rect, label: "Batch Normalization" }
    p2@{ shape: st-rect, label: "Max Pooling" }
end

subgraph conv3[Convolution Layer 3]
    space3["<p style='width:100px;height:0px;margin:0'>Space</p>"]::withMargins;
    c3@{ shape: st-rect, label: "Conv1D:\n4 filters – kernel 8" }
    b3@{ shape: st-rect, label: "Batch Normalization" }
    p3@{ shape: st-rect, label: "Max Pooling" }
end

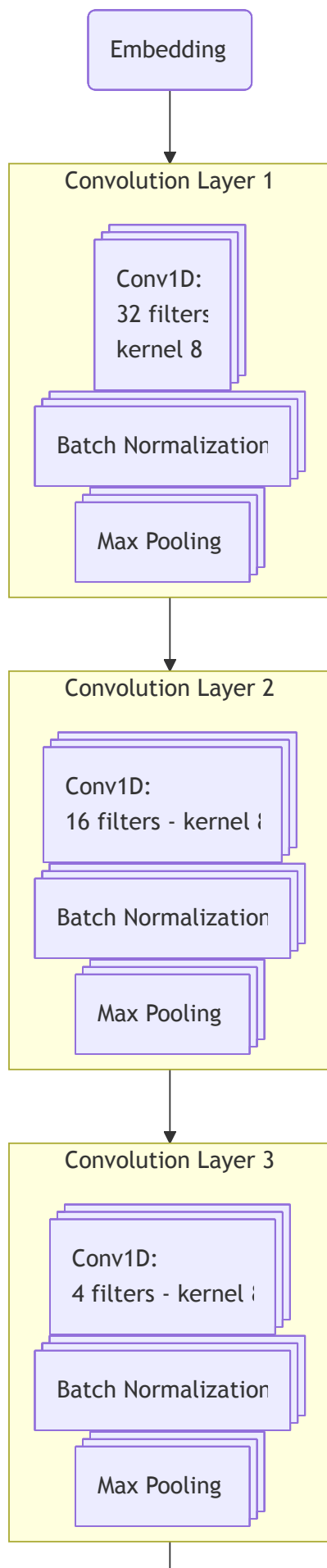
subgraph dense[Full Connected Layer]
    space4["<p style='width:100px;height:0px;margin:0'>Space</p>"]::withMargins;
    do1[Dropout]
    p4[Global Average Pooling]
    d1[Dense]
end

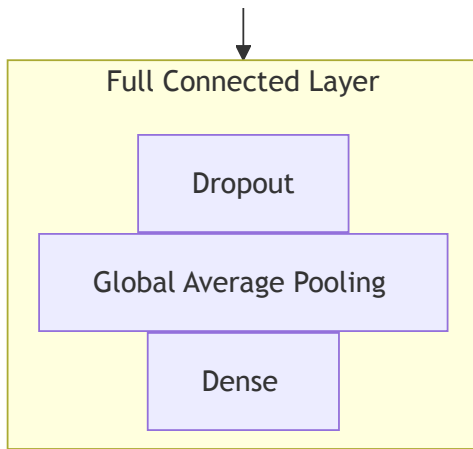
i -----> conv1
conv1 -----> conv2
conv2 -----> conv3
conv3 -----> dense

```

Mermaid(basic_cnn_graph)

Out[47]:





```
In [49]: final_cnn_graph = """
%%{init:{'flowchart':{'nodeSpacing':1, 'rankSpacing':10}}}%

flowchart TD
    classDef withMargins fill-opacity:0.0,color:#FFFFFF,stroke-width:0px;
    i(Embedding)

    subgraph conv1[Convolution Layer 1]
        space1["<p style='width:100px;height:0px;margin:0'>Space</p>"]::withMargins;
        c1@{ shape: st-rect, label: "Conv1D:\n16 filters - kernel 8" }
        b1@{ shape: st-rect, label: "Batch Normalization" }
        a1@{ shape: st-rect, label: "Activation ReLU" }
        p1@{ shape: st-rect, label: "Max Pooling" }
    end
    %% Define a class to make the padding subgraph invisible
    classDef padding stroke:none,fill:none

    subgraph conv2[Convolution Layer 2]
        space2["<p style='width:100px;height:0px;margin:0'>Space</p>"]::withMargins;
        c2@{ shape: st-rect, label: "Conv1D:\n8 filters - kernel 8" }
        b2@{ shape: st-rect, label: "Batch Normalization" }
        p2@{ shape: st-rect, label: "Max Pooling" }
    end

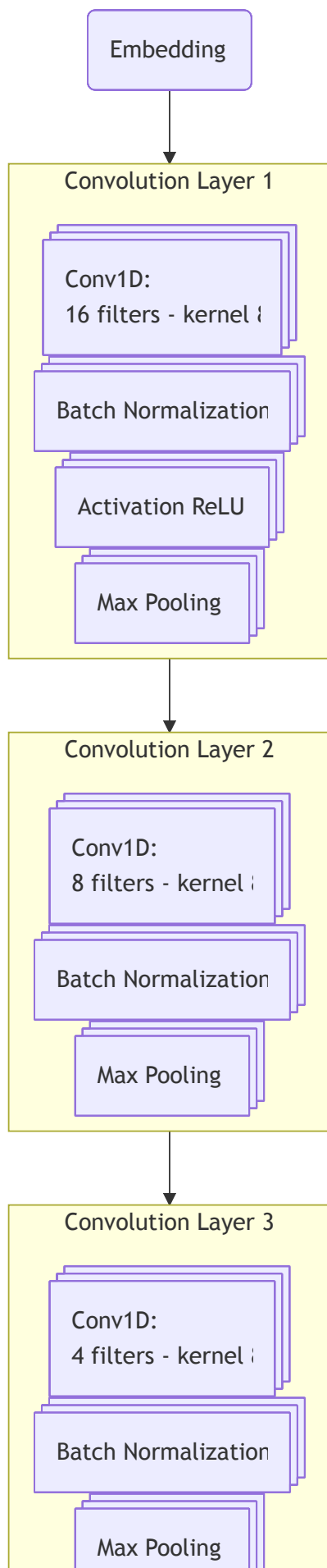
    subgraph conv3[Convolution Layer 3]
        space3["<p style='width:100px;height:0px;margin:0'>Space</p>"]::withMargins;
        c3@{ shape: st-rect, label: "Conv1D:\n4 filters - kernel 8" }
        b3@{ shape: st-rect, label: "Batch Normalization" }
        p3@{ shape: st-rect, label: "Max Pooling" }
    end

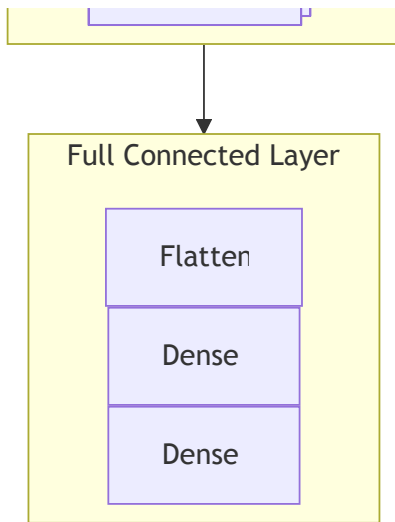
    subgraph dense[Full Connected Layer]
        space4["<p style='width:100px;height:0px;margin:0'>Space</p>"]::withMargins;
        f[Flatten]
        d1[Dense]
        d2[Dense]
    end

    i -----> conv1
    conv1 -----> conv2
    conv2 -----> conv3
    conv3 -----> dense
    """

Mermaid(final_cnn_graph)
```

Out[49]:





```

In [73]: %%capture_code --path "/content/code/mermaid_example.py"
final_cnn_graph_update = """
%%{init:{'flowchart':{'nodeSpacing':1, 'rankSpacing':10}}}%

flowchart TD
    classDef withMargins fill-opacity:0.0,color:#FFFFFF,stroke-width:0px;
    i(Embedding)

    subgraph conv1[Convolution Layer 1]
        space1["<p style='width:100px;height:0px;margin:0'>Space</p>"]::withMargins;
        c1@{ shape: st-rect, label: "Conv1D:\n16 filters – kernel 8" }
        b1@{ shape: st-rect, label: "Batch Normalization" }
        a1@{ shape: st-rect, label: "Activation ReLU" }
        p1@{ shape: st-rect, label: "Max Pooling" }
    end

    %% Define a class to make the padding subgraph invisible
    classDef padding stroke:none,fill:none

    subgraph conv2[Convolution Layer 2]
        space2["<p style='width:100px;height:0px;margin:0'>Space</p>"]::withMargins;
        c2@{ shape: st-rect, label: "Conv1D:\n8 filters – kernel 8" }
        b2@{ shape: st-rect, label: "Batch Normalization" }
        p2@{ shape: st-rect, label: "Max Pooling" }
    end

    subgraph conv3[Convolution Layer 3]
        space3["<p style='width:100px;height:0px;margin:0'>Space</p>"]::withMargins;
        c3@{ shape: st-rect, label: "Conv1D:\n4 filters – kernel 8" }
        b3@{ shape: st-rect, label: "Batch Normalization" }
        p3@{ shape: st-rect, label: "Max Pooling" }
    end

    subgraph dense[Full Connected Layer]
        space4["<p style='width:100px;height:0px;margin:0'>Space</p>"]::withMargins;
        p4[Global Average Pooling]
        f[Flatten]
        d1[Dense]
        d2[Dense]
    end

    i -----> conv1
    conv1 -----> conv2
    conv2 -----> conv3

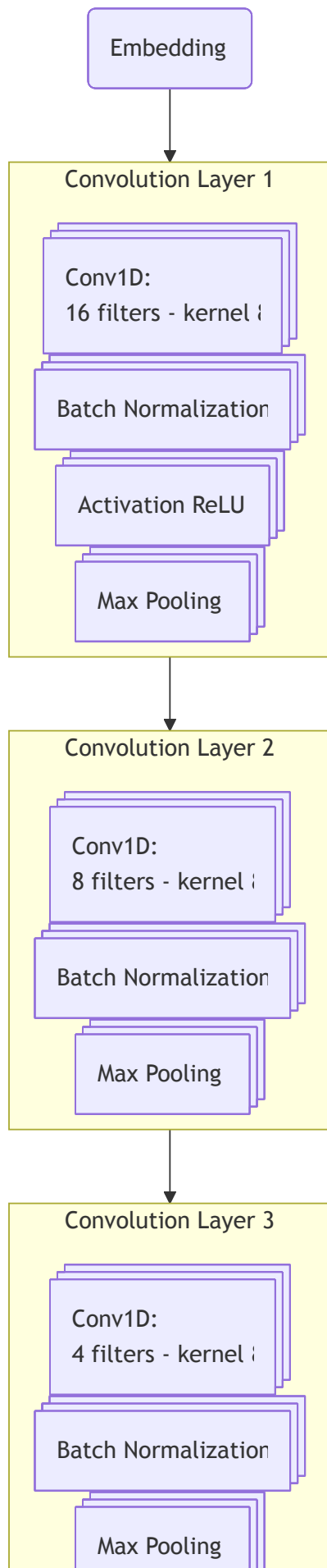
```

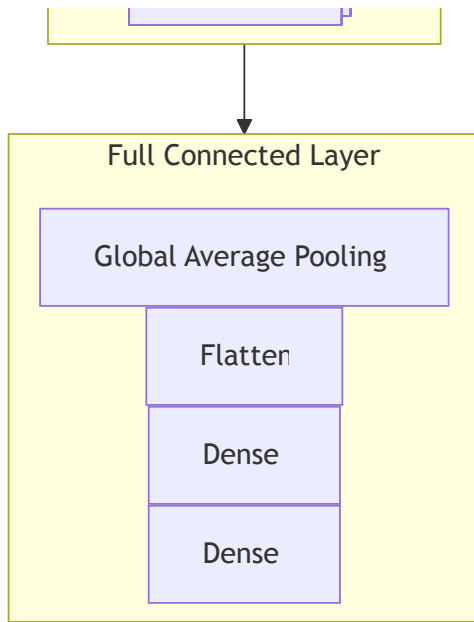
```
conv3 -----> dense
.....

Mermaid(final_cnn_graph_update)
```

Output saved by creating file at /content/code/mermaid_example.py.

Out[73]:





End of Project